Published in Towards Data Science

Julia Kho Follow
Jun 2, 2021 · 12 min read ★ · Listen

GETTING STARTED

An Easy Beginner's Guide to Git Part 1

Learn the Basics of Git in Just 12 Minutes

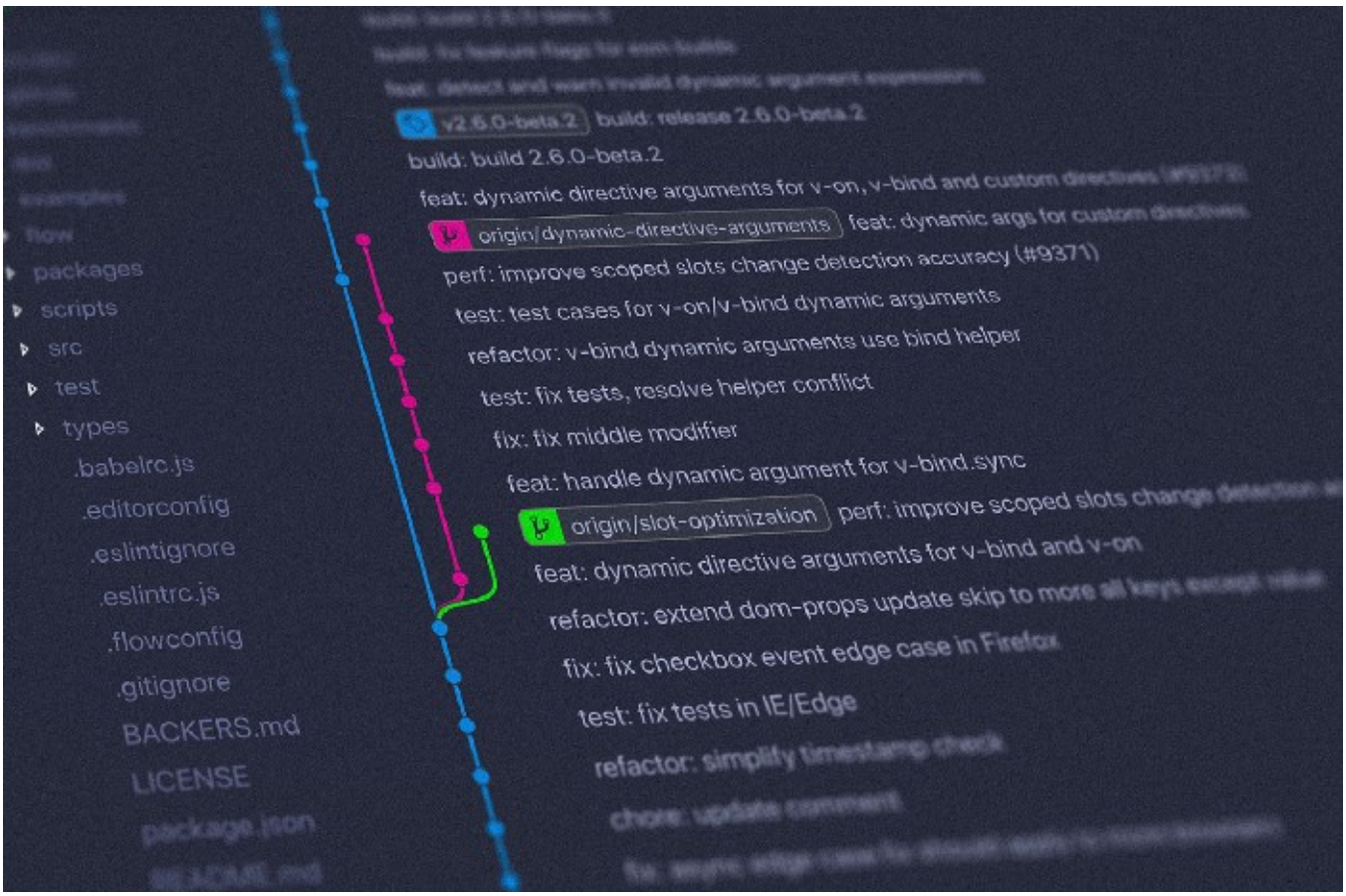


Photo by Yancy Min on Unsplash

If you're new to coding, you're definitely going to want to learn about Git!

there are no code conflicts as well as allow developers the ability to revert files or entire projects back to a previous version of their code.

Git can track what changes were made, who made the change, when they made the change, and why they made the change. How cool is that? Better yet, it's all **free**!

In this article, I will help you navigate using Git. Once you get the hang of it, it'll be easy to use. Here's an outline of the topics:

1. Installing Git
2. Creating a Local Git Repository
3. Committing Files to the Local Repository
4. Review All Commits Made
5. Ignoring Files in Commit
6. Git Branches
7. Create a Remote Repository to Share with Others
8. Moving Code Between Local and Remote Repository
9. Bonus Content! Tagging

Installing Git

If you don't have Git installed on your computer already, follow the instructions in the link below for help with installation.

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

If you are not sure whether you have it, you can type the following in your terminal to see if it was installed.

```
git --version
```

Note: To use Git, Mac users will be using the terminal to interact. For Windows users, it




If you're starting fresh with a brand new project, you will need to create a local Git repository. This is where your files and checkpoints of your changes will be stored to allow for versioning control.

Before we initialize our local repository, let's create a folder on your Desktop for our example. Let's name it "git-demo-example".

Now, open up your terminal (or Command Prompt). Copy the following commands below. Line 1 will navigate to the folder we just created and line 2 will create our local repository.

```
cd ~/Desktop/git-demo-example
git init
```

That's it! We have created our first local repository which resides in the git-demo-example folder.

Next, let's put some files into the folder so that we have something to work with. Create a file called fruits.py in our git-demo-example folder and copy the following piece of code into the file. *Note: If you don't have Python installed, you can create a .txt file instead of a .py file and still follow along.*

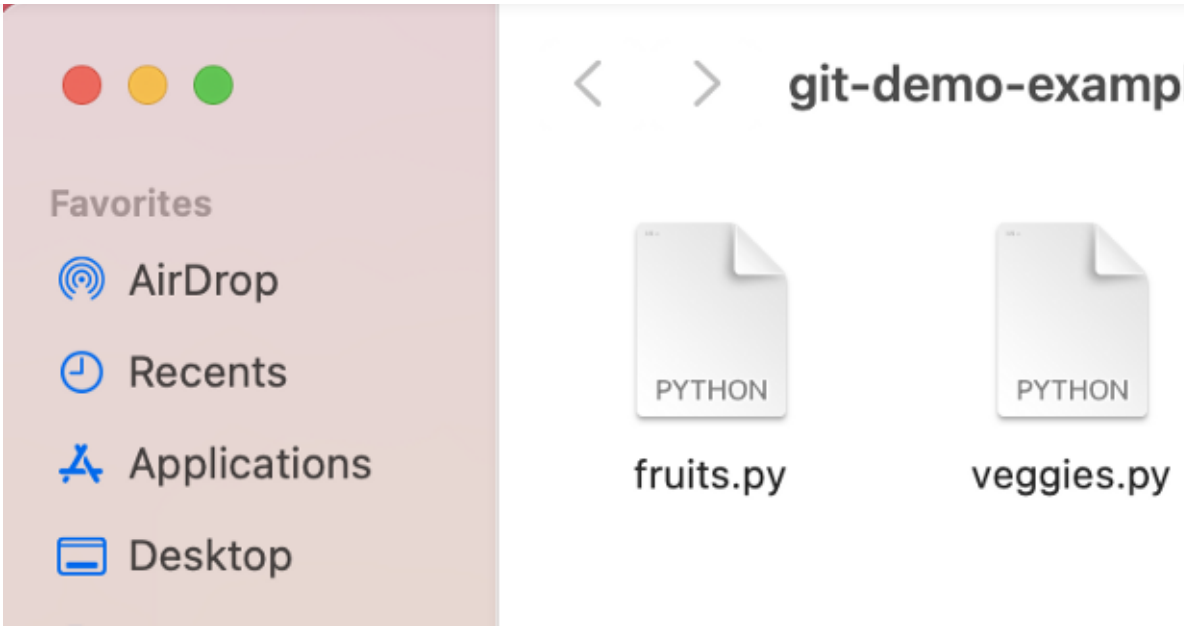
```
fruits = ['apple', 'dragonfruit', 'peach', 'banana', 'grape',
'apple', 'peach', 'watermelon', 'grape', 'grape']
```

Save the document and exit. Create one more file called veggies.py in the same folder and copy the following piece of code into the file.

```
veggies = ['cabbage', 'carrot', 'spinach', 'asparagus', 'artichoke',
'pumpkin', 'lettuce']
```

You should now have two files in the git-demo-example folder.

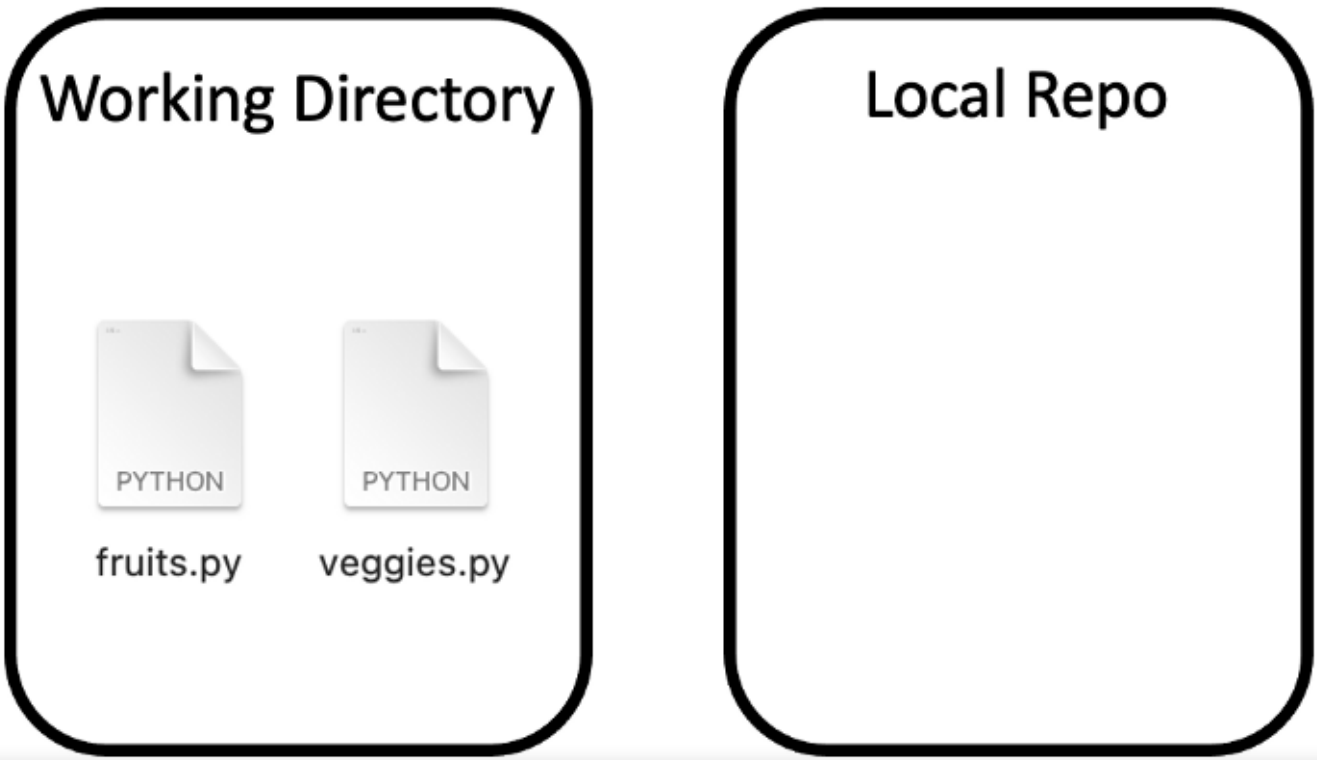


Source: Author

Committing Files to the Local Repository

So far, what we've done is created a local repository and added two files into our working directory (the git-demo-example folder). **We will have to add the files into the local repository in order to be able to start tracking the files and changes we make.** This process of adding a file to a local repository is called *committing*.

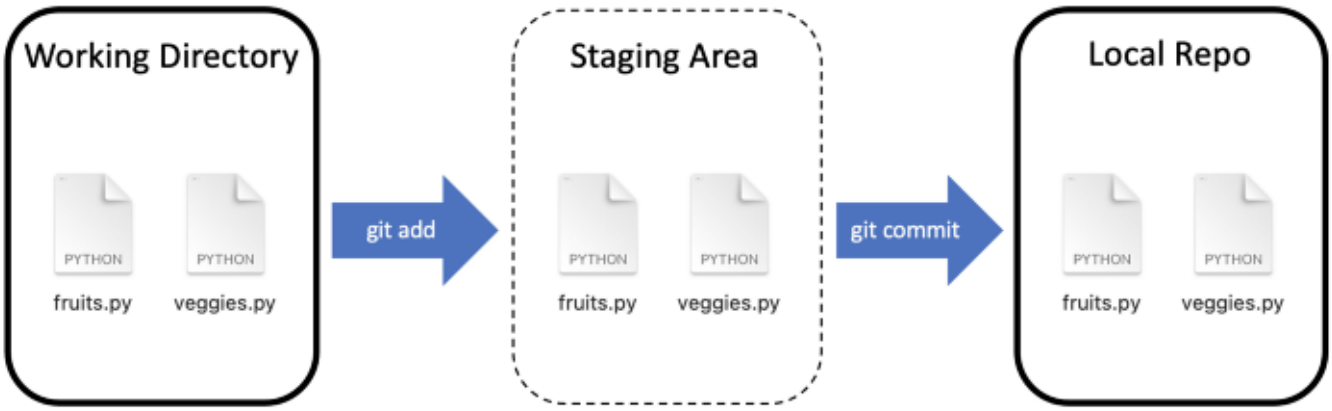


process gives you control over what files you actually want to commit to the local repository as well as allow you to change your mind if you accidentally choose the wrong file initially.

To add only a single file to the staging area, you can use `git add fruit.py`

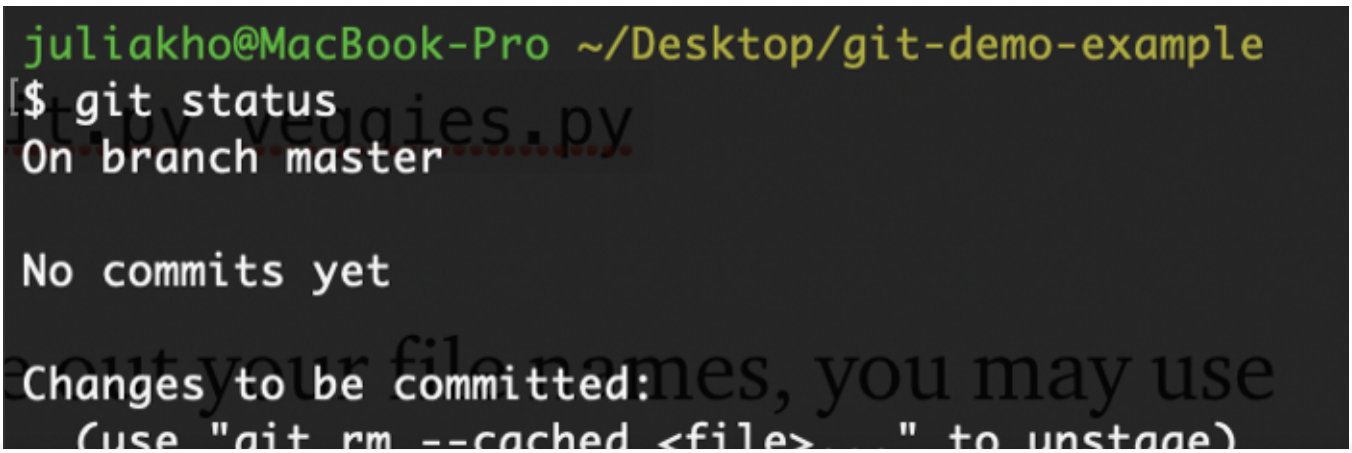
To add multiple files, use this: `git add fruit.py veggies.py`

If you have many files or are too lazy to type out your file names, you may use `git add .` to add all files that are in the folder.



Source: Author

Go ahead and add both of the files the the staging area using either of the two options mentioned above. If you want to double check what is in the staging area, you can use `git status` to check the staging area. You can see our two new files below and that changes have not yet been committed.



Source: Author

To commit the file, we will use `git commit –m "My First Commit" .` "My First Commit" can be any sort of comment you want to make about the code or about what changes you made. It's definitely a good idea to type something relevant so that future you can remember what was done.

We've just made our very first commit! Great job!




Silly me. In our list of veggies, we listed pumpkins as a vegetable, but actually it's a fruit! Did you know that? Let's fix that mistake. Open up the two files we created and remove pumpkin from veggies and add it to the fruits. After you make the changes to those document, go ahead and practice doing another commit.

```
git add .
git commit -m "Changed pumpkin from veggie to fruit"
```

> I used to be addicted to pumpkin, but then I went on the patch.

Photo by Maddy Baker on Unsplash

Review All Commits Made

If you want to see a list of all the commits we've made thus far, you can use `git log`. The log is where you will find the author of the commit, the date it was committed, and the commit message for every commit that was made.

```
$ git log
commit d11208d88b62127a3da505934f8c5294ac111a10 (HEAD -> test, master)
Author: Julia Kho <juliakho@MacBook-Pro.local>
Date:   Fri May 14 17:14:50 2021 -0700

    Changed pumpkin from veggie to fruit

commit 479d92cb80e32b575558862587ab4f4e98db4961
Author: Julia Kho <juliakho@MacBook-Pro.local>
Date:   Thu May 13 11:01:21 2021 -0700

    My First Commit
```

If you've made many commits, you could be scrolling through this log for a while. It might be worthwhile to use `git log --pretty=oneline` which will condense your commit messages to just one line.

```
$ git log --pretty=oneline
d11208d88b62127a3da505934f8c5294ac111a10 (HEAD -> test, master) Changed pumpkin from veggie to fruit
479d92cb80e32b575558862587ab4f4e98db4961 My First Commit
```

Source: Author

There are other cool parameters you can use like `git log – author=julia` if you want to see all the commits made by Julia.

If you are curious, this article here shows you 10 different ways you can use the log tool.

Ignoring Files in Commits

If there are files in which you would never want committed, you can create a .gitignore file to assist with that problem. This .gitignore file will tell git which files or patterns should be ignored so that they don't accidentally get staged and committed. It's the smart thing to do!



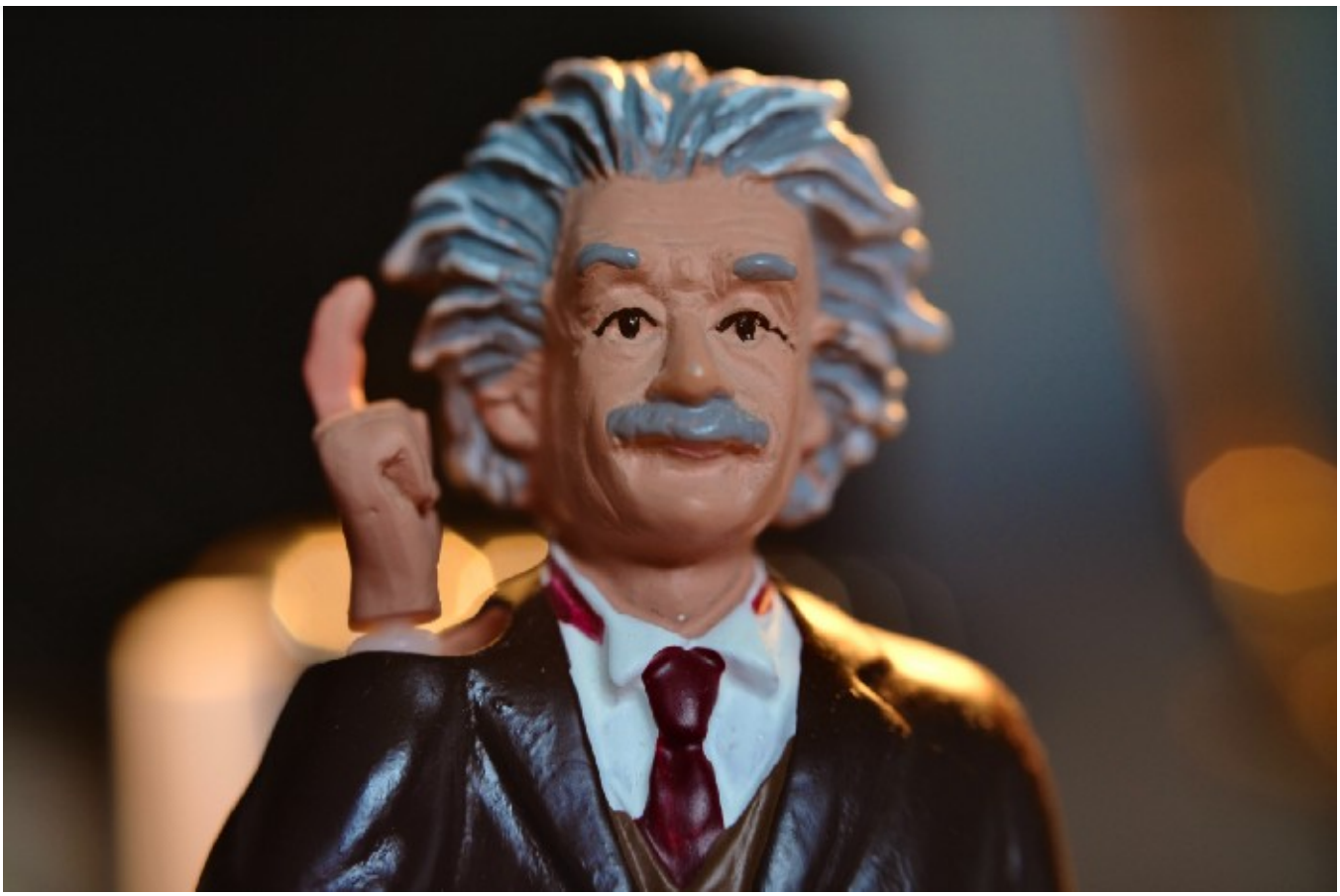



Photo by Andrew George on Unsplash

Create .gitignore file with this command in the terminal: `touch .gitignore`

Open up .gitignore and add the names of the files that you don't want to be tracked. Note: if you don't see the .gitignore, it's probably hidden, so make sure you are viewing hidden files.

The snippet below is an example of what a .gitignore file can look like. The first and second lines are names of files that I may have that I don't want to include. You can also specify patterns to be matched against file names. For example picture* will look for all files that start with the word picture. The asterisk here is essentially a wildcard that matches zero or more characters.

Source: Author

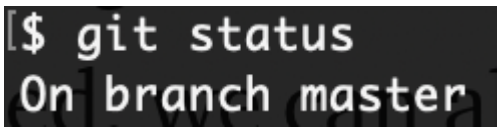
For more information on pattern matching, check out this link here.

Go ahead and practice a third commit by adding the .gitignore file to your local repository.

Git Branches

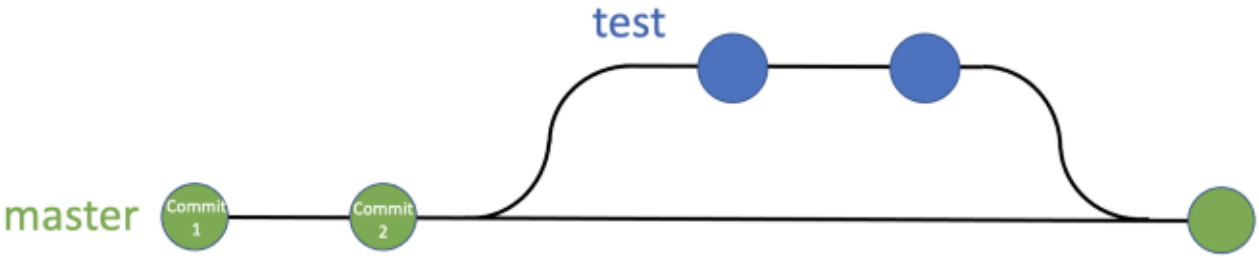
One really cool thing you can do with Git is that you can create different branches so that you can keep your work separate from the master branch. This is useful if you want to test code out.

In your terminal, if you type `git status`, you can see that you're on the master branch.



Source: Author

By default, you will always be working off the master branch in Git. The two commits we made earlier were on the master branch. We have the ability to create a new branch (for example test) where we can test new code out without messing with the code in the master branch.



Source: Author

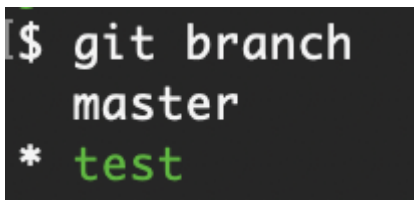
In the example above, each circle represents a commit. When we do commits in the test branch, those checkpoints will be kept separately from the master branch. If we like the code in the test branch, we can merge with the master branch. If the ideas implemented in the test branch failed, we can abandon them. When working with multiple collaborators, you can each create your own branch to work from as well.

Let's create a new branch called test.

```
git branch test
```

To see what branch you are on, you can type `git branch` and the branch you are on will be highlighted in green. We are currently on the master branch.

We will need to move from the master branch to the test branch if we want to commit code to the test branch. To do so, use the following command: `git checkout test`



Source: Author

While we're here in the test branch, let's modify our fruits.py file and commit the changes to this test branch. You may have noticed that we have duplicate fruits in our list. Add in the following code the fruits.py file to get a unique set.

```
fruits = set(fruits)
```

Save and let's commit the new change.

```
git add fruits.py
git commit –m "Removed duplicate fruits"
```

This test branch contains the first two commits we made earlier along with our latest code commit while the master branch contains only the first two commits from earlier. However, we can merge the test branch to the master branch so that the master branch has the latest code.

Before merging the two branches, you can take a look at what has changed between the branches by using `git diff test master` . Test is your source branch and master is your target branch.

To perform the merge, you will have to move back to the master branch and then use the merge function like this:

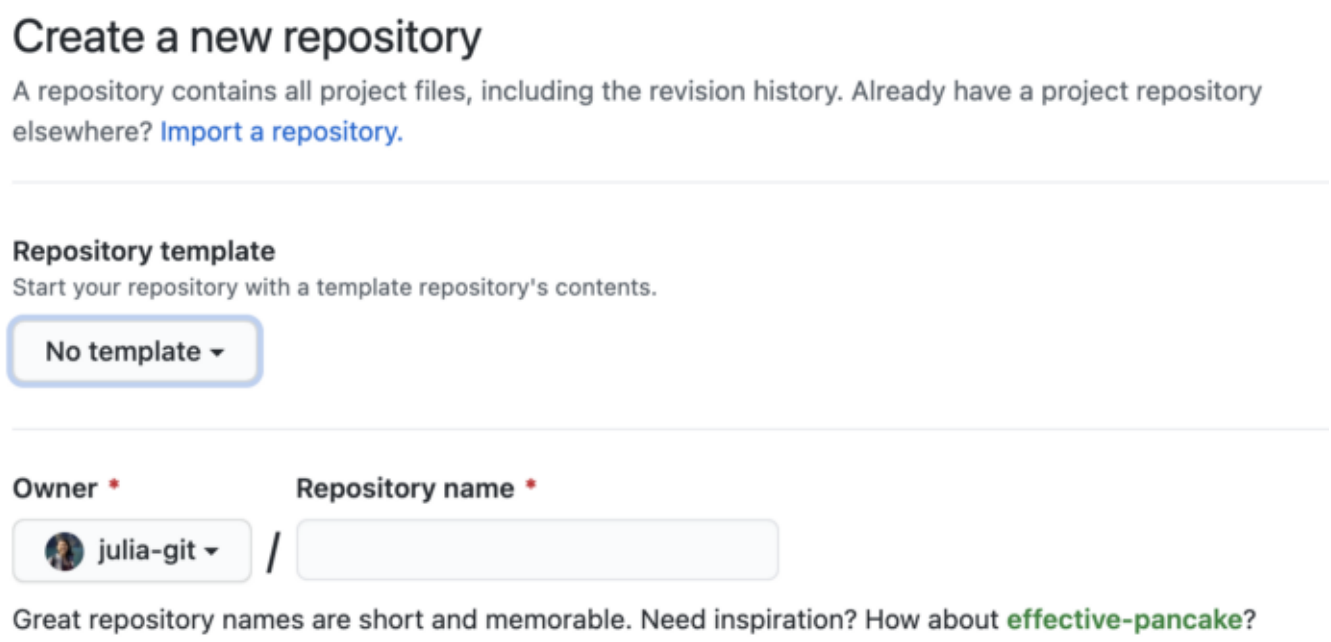
```
git checkout master
git merge test
```

Create a Remote Repository to Share with Others

It's great that you have saved all the changes in your local repository, but maybe you want to share it with your colleagues or with the world. We can create a remote repository and sync it with your local repository. That way, you can push changes to a central location where other people can access your code and they can help contribute as well.

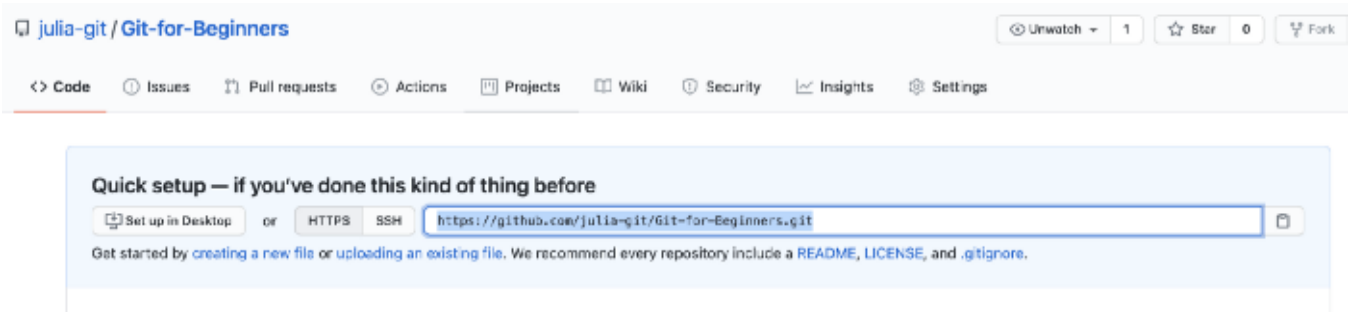



On the homepage, click Start a Project and create a repository. I'm going to name it "Git-for-Beginners".



Source: Author

After you've setup the repo, copy the url for your repository. As shown below my repository url is https://github.com/julia-git/Git-for-Beginners.git



Source: Author

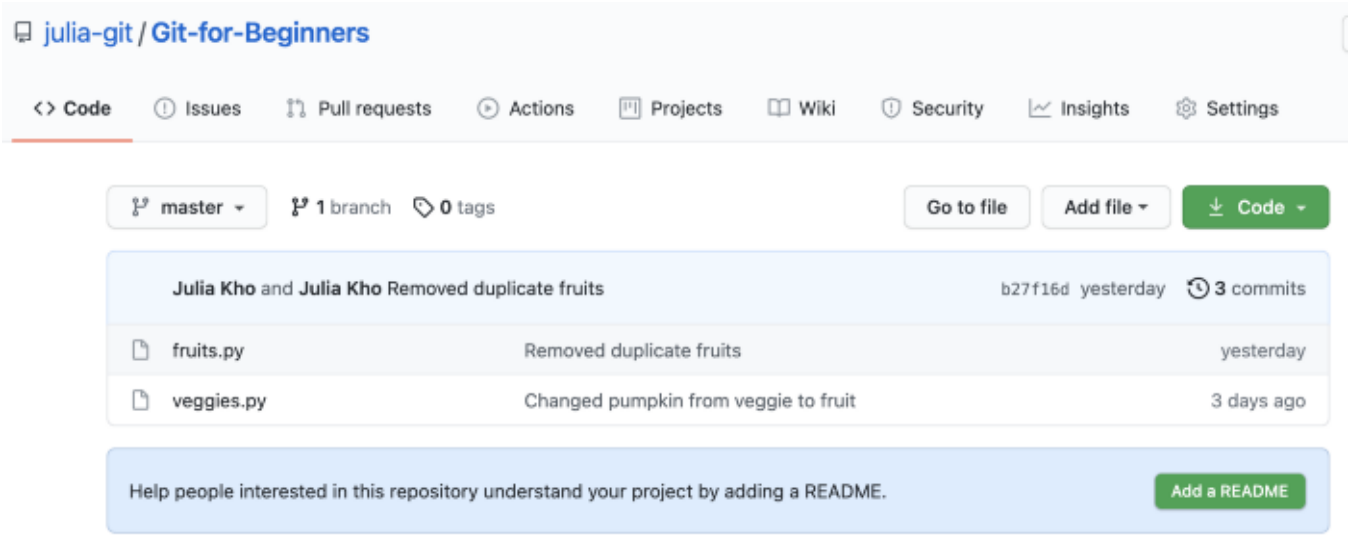
Now, we need to tell our local repository where our remote repository is sitting in order to establish a link between the two. Replace the line of code below with your own repository url.

```
git remote add origin [repository url]
```

Now that a connection has been establish, to move all the code in your local repository to the remote repository, use `git push –u origin master .`

With this line of code, everything from the master branch on our local repository is going to now also be in the remote repository. If you wish to push a different branch, just replace "master" with the name of your branch that you want to push.

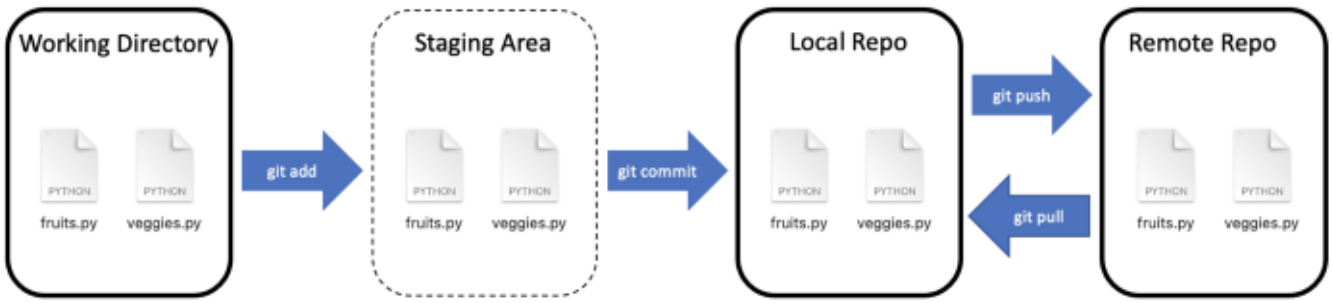
If you go back to your browser tab, you should see that your items have successfully been pushed the remote repository.



Source: Author

If someone else was working on this project and pushed changes to the remote repository, you can pull them down to your local repository on your computer. You will use the following to pull from the master branch: `git pull origin master .`

In summary, the commands below are some of what we've learned thus far.



Now let's try something different. Let's say that you're not starting fresh on a project, but you're helping out your teammate on their project. We will have to go to their remote repository and clone it to our computer to create a local repository. Feel free to go ahead and clone the one I created for this article using the following repository url: https://github.com/julia-git/Git-for-Beginners

```
git clone https://github.com/julia-git/Git-for-Beginners
```

This command essentially downloads the repository to your local computer and preserves the connection between the two repositories.

You've now learned all the basic commands to navigate with Git. If you get stuck and need help figuring out what command to use, you can always use `git --help`. Also, here's a handy cheat sheet in case you forget any of the Git commands.

If you'd like to continue to learn more, check out Part 2 of An Easy Beginner's Guide to Git.

BONUS CONTENT! (for those feeling extra adventurous)

Tags

You can tag specific points in your commit history as being important, for example, software release version 1.0, 2.0, etc.

Lightweight versus Annotated Tags

There are two types of tags you can create: lightweight and annotated.

The lightweight tag is meant for more temporary purposes as it just points to a specific commit. Annotated tags are full on objects that contains the tagger's name, email, and date, a message, and verification of identity.

To create a lightweight tag for version 1.4:




To create an annotated tag for version 1.6:

```
git tag -a v1.6 -m "my version 1.6"
```

You can replace the message in the quotes above with your own message.

If you want to see what tags you currently have, you can use `git tag .`

Tagging at a Later Point in Time

If the current commit is not what you want to tag, we can look through the commit history to find the one we want.

```
git log --pretty=oneline
```

```
$ git log --pretty=oneline
b27f16dcec17b0ee3f8c4774cc42195fed267385 (HEAD -> master, origin/master, test) Removed duplicate fruits
d11208d88b62127a3da505934f8c5294ac111a10 Changed pumpkin from veggie to fruit
479d92cb80e32b575558862587ab4f4e98db4961 My First Commit
```

Let's pretend that the second commit is the one we want to tag. Copy the entire id or part of it to your tag statement. As shown in the snippet above, the id we will use is `479d92 .`

```
git tag -a v1.6 479d92 -m "new version 1.6"
```

Pushing Tags to Remote Repository

Tags will not be pushed to the remote repo unless you explicitly do it. Replace the name of the tag below with the one you want to push to the server.

```
git push origin <tagname>
```

Deleting Tags

If you want to delete a tag in the remote repository, you will use:

```
git push origin --delete <tagname>
```