

Creating a Remote Repo on GitHub

In this lesson, we will work both on local Git repo and remote [GitHub](#) repo. We will see the interactions between them.

Let's go and sign into [GitHub](#), then create a new repository but don't forget to select initiating a **Read Me** file. This will automatically create our first commit, so later we will be able to clone it to our local repo and work on it. For this demo we will name our repo *my-demo-repo*. Click **Create repository** when you are done.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner: guile-clarusway

Repository name *: my-demo-repo ✓

Great repository names are [short](#), [descriptive](#), and [easy to find](#). Avoid [names](#) about shiny-tribble?

Description (optional)

☒ Public
Anyone can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☒ Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None | Add a license: None ⓘ

Create repository

New repo has been created. Now click on the **Clone or download** button.

Search or jump to... Pull requests Issues Marketplace Explore

guile-clarusway / my-demo-repo

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

No description, website, or topics provided. Edit

Manage topics

1 commit 1 branch 0 packages 0 releases 1 contributor

Branch: master New pull request

Create new file Upload files Find file Clone or download

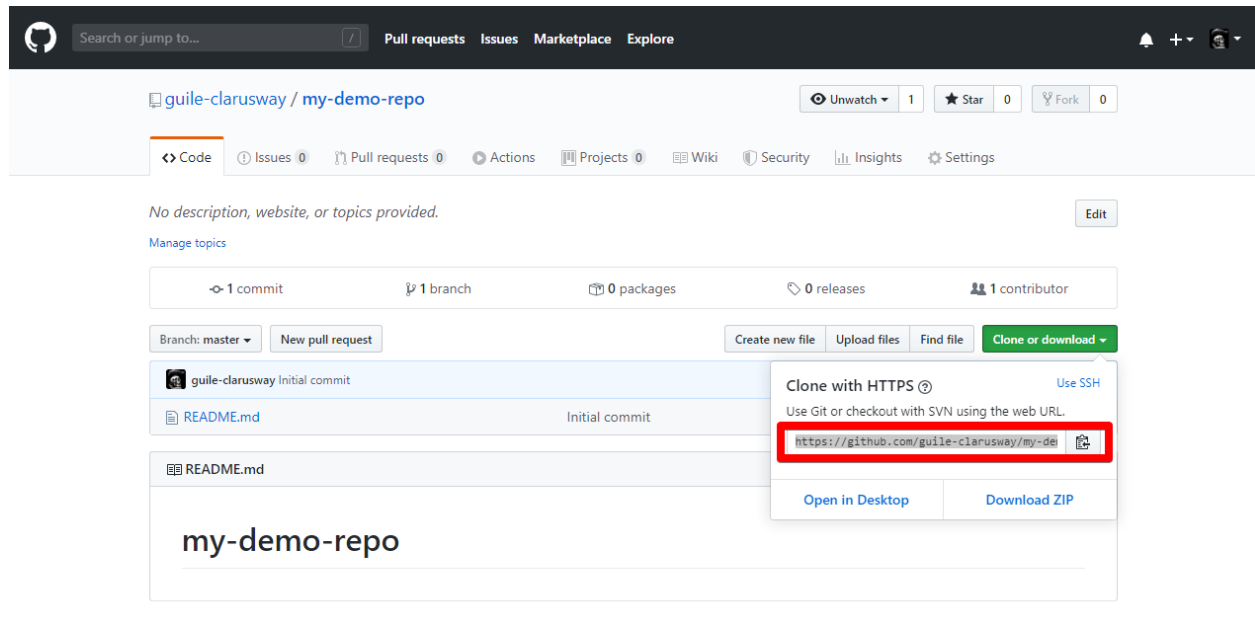
guile-clarusway Initial commit Latest commit #5f7cd3 1 minute ago

README.md Initial commit 1 minute ago

README.md

my-demo-repo

Then copy the URL of the repo. We will use this URL later on cloning.



Cloning Remote Repo to Local

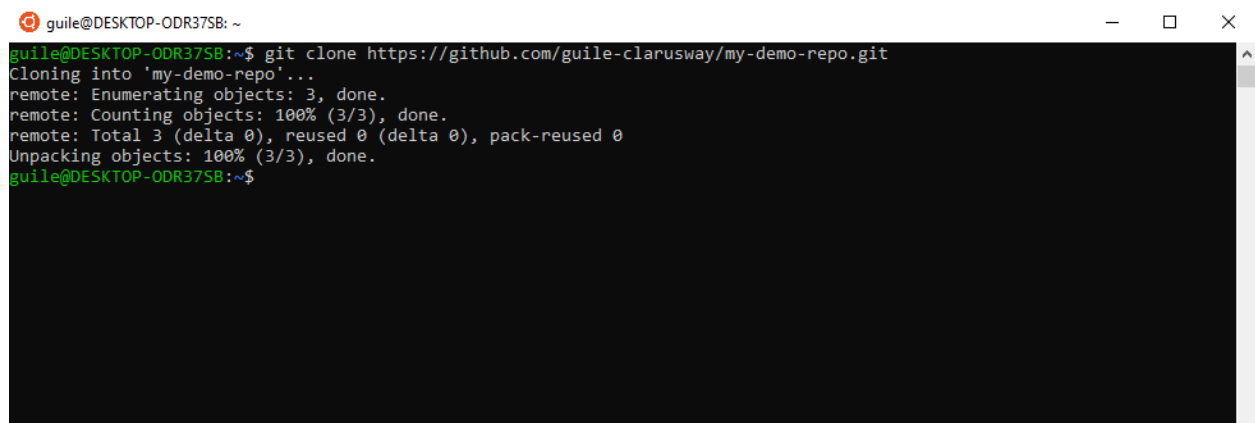
Now that, our repo is created and ready, we can start working on it. But remember, Git is a Distributed VCS and some other colleagues in the same project may need to work with the same repo. Thus, we will clone it to our local repo and work on it locally.

Open GitBash/Terminal and type the command below. Use your own repo URL which you copied after creating it.

```
$ git clone YOUR_REPO_URL
```

That is the example for this case. You will use **your own repo URL**.

```
$ git clone https://github.com/guile-clarusway/my-demo-repo.git
```



If you see something like that you have successfully cloned it to your local repo. Let's check,

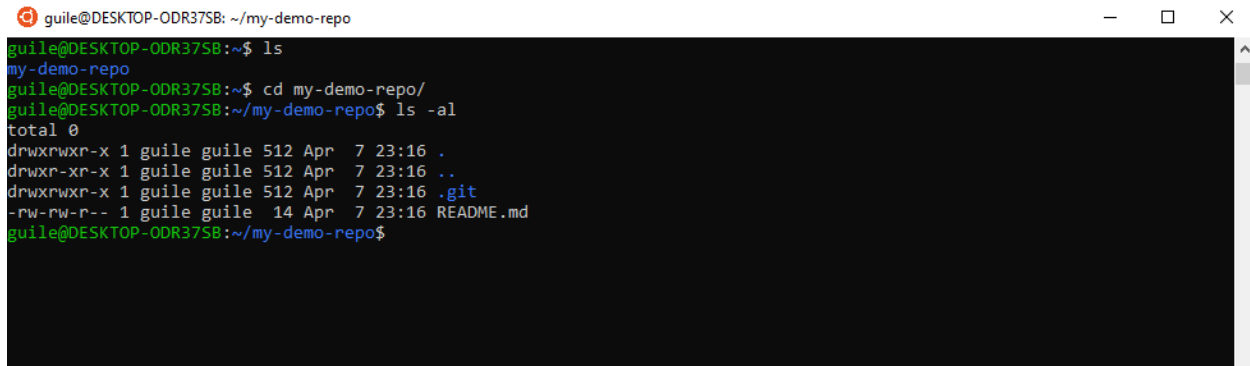
```
$ ls
```

and open the folder.

```
$ cd my-demo-repo
```

Now let's see what is in the repo.

```
$ ls -al
```

A terminal window titled 'guile@DESKTOP-ODR37SB: ~/my-demo-repo' with standard window controls. The terminal shows the following commands and output:

```
guile@DESKTOP-ODR37SB:~$ ls
my-demo-repo
guile@DESKTOP-ODR37SB:~$ cd my-demo-repo/
guile@DESKTOP-ODR37SB:~/my-demo-repo$ ls -al
total 0
drwxrwxr-x 1 guile guile 512 Apr  7 23:16 .
drwxr-xr-x 1 guile guile 512 Apr  7 23:16 ..
drwxrwxr-x 1 guile guile 512 Apr  7 23:16 .git
-rw-rw-r-- 1 guile guile 14 Apr  7 23:16 README.md
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

You should notice that **.git** already exists in addition to the **README.md** which is the only file in the repo. You don't need to use **git init** to initiate a repo when you clone it from remote.

Now we can make changes on our repo locally.

Creating a New Branch

Let's assume we have a new idea but we are not exactly sure about it. We want to have an experiment. Branches are the keys to this kind of situation. You can create a new branch, make some progress on it and you can go back to the master branch if you are not satisfied with the new branch. You can think branches as a reference or bookmark to a specific commit. For this demo, we will create a new branch and work on it.

First, check your current branch

```
$ git branch
```

Then create a new branch. We will use **demobranch** for this lesson.

```
$ git branch demobranch
```

Check again and as you see the new branch is created but we are still on the master branch since it's green colored. Now switch to the demobranch

```
$ git checkout demobranch
```

```
$ git branch
```

We are now on the new branch and we can start working.

```
guile@DESKTOP-ODR37SB: ~/my-demo-repo
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git branch
* master
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git branch demobranh
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git branch
 demobranh
* master
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git checkout demobranh
Switched to branch 'demobranh'
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git branch
* demobranh
  master
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

Let's add a simple Python file to our project. Use your favorite editor, add `print("Hello World")` to the file.

```
$ vim hello.py
```

You can check it if you want.

```
$ cat hello.py
```

```
guile@DESKTOP-ODR37SB: ~/my-demo-repo
guile@DESKTOP-ODR37SB:~/my-demo-repo$ vim hello.py
guile@DESKTOP-ODR37SB:~/my-demo-repo$ cat hello.py
print("Hello World")
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

We can stage,

```
$ git add .
```

And commit it now.

```
$ git commit -m "New python file added"
```

Let's check our log

```
$ git log
```

```
guile@DESKTOP-ODR37SB: ~/my-demo-repo
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git add .
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git commit -m "New python file added"
[demobranh aff9ce6] New python file added
1 file changed, 1 insertion(+)
create mode 100644 hello.py
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git log
commit aff9ce61534ccdb1196de4d3c3dd9f997eee0f47 (HEAD -> demobranh)
Author: guile <guile@clarusway.com>
Date: Tue Apr 7 23:54:50 2020 +0300

    New python file added

commit f5f7cd3ef4ee19268b67a0a71730d0006fcfaa17 (origin/master, origin/HEAD, master)
Author: guile-clarusway <59561051+guile-clarusway@users.noreply.github.com>
Date: Tue Apr 7 22:36:37 2020 +0300

    Initial commit
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

Pushing Branch to Remote

Now that we are done with the local repo, let's push it to remote. Command below will upload our local repo content to the remote repo.

```
$ git push origin demobbranch
```

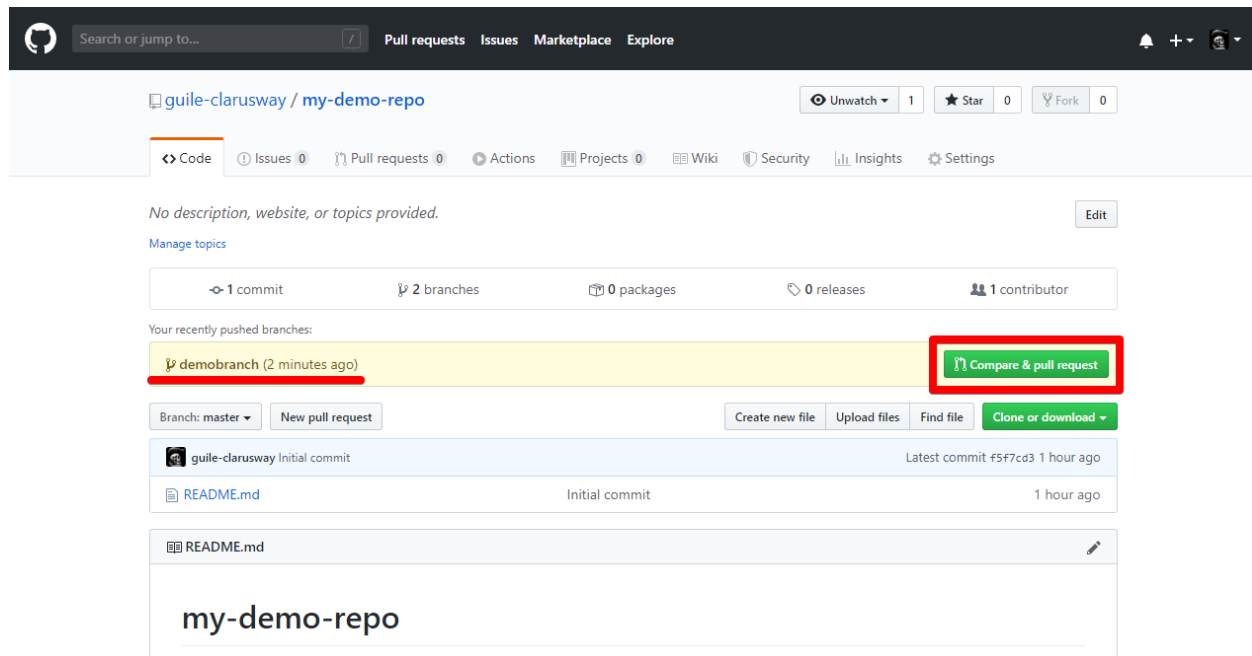
Enter your [Github](#) username and password if asked.

If you remember the first [Github](#) Lesson, we used `git remote add origin` command previous to `git push` command. Here we don't need this command because we cloned our repo from remote and both the remote and the local repo are already associated with each other.

That is what you are supposed to see after pushing.

```
guile@DESKTOP-ODR37SB: ~/my-demo-repo
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git push origin demobbranch
Username for 'https://github.com': guile-clarusway
Password for 'https://guile-clarusway@github.com':
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes | 60.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'demobbranch' on GitHub by visiting:
remote:   https://github.com/guile-clarusway/my-demo-repo/pull/new/demobbranch
remote:
To https://github.com/guile-clarusway/my-demo-repo.git
 * [new branch]   demobbranch -> demobbranch
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

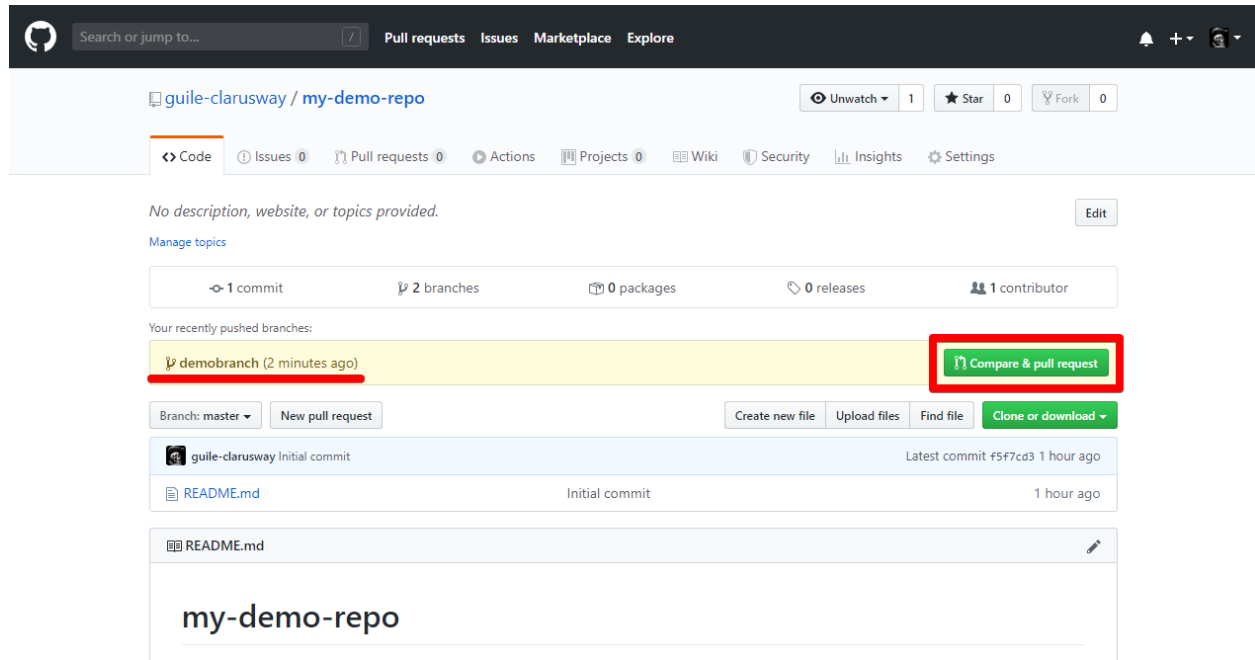
Now let's go and check our remote repo. If you have successfully pushed your branch, you should see the **demobbranch**. Remember we had only one branch when we first created our remote repo. Now we have two.



Creating Pull Request (PR) on GitHub

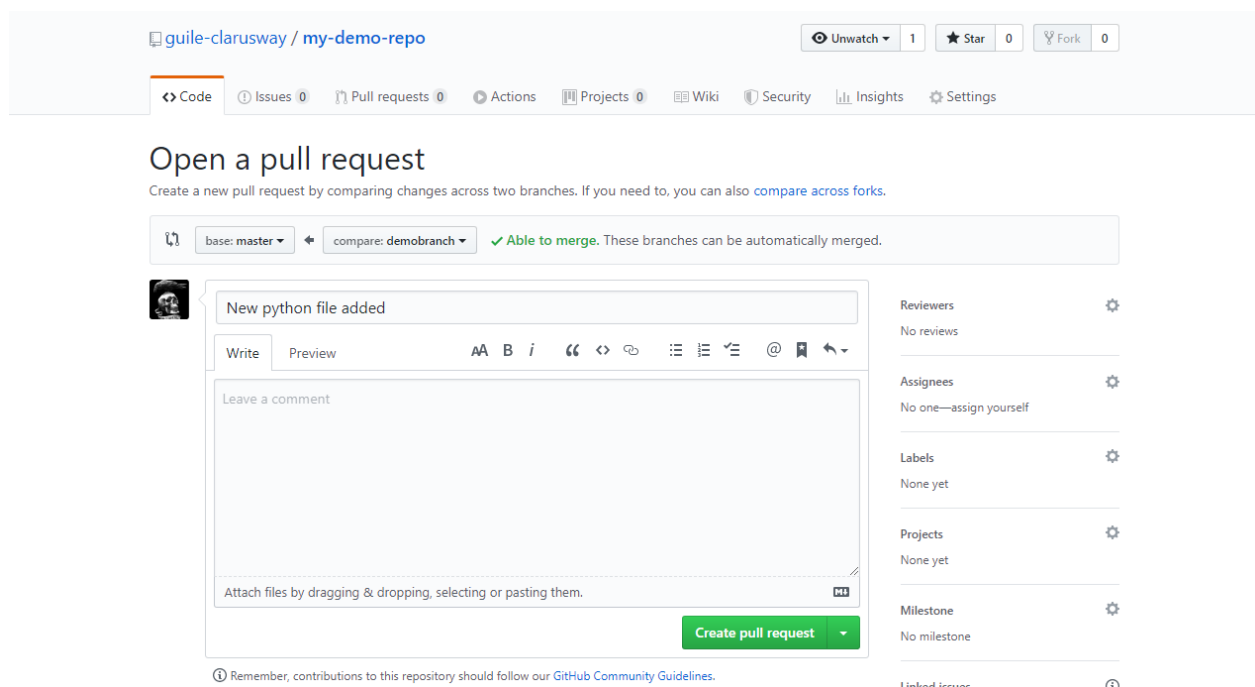
Pull Request (PR) is a method that you tell other contributors or team members that you had some changes (fixing bugs, adding new features, etc.) on the repo and you want to contribute these to the repo. After those changes have been checked and confirmed by the authorized people it is merged to the repo.

Here we will create a pull request for our own repo. Click **Compare & pull request** to open a pull request.



Notice the **Able to merge** remark. Our branch can be merged. You can write a description if you want, your last commit message will be inserted automatically ("New python file added").

Click **Create pull request**.



Merging Pull Request

Now it's time to merge the pull request. Since we are the only owner and the contributor of this repo, we will merge our own pull request.

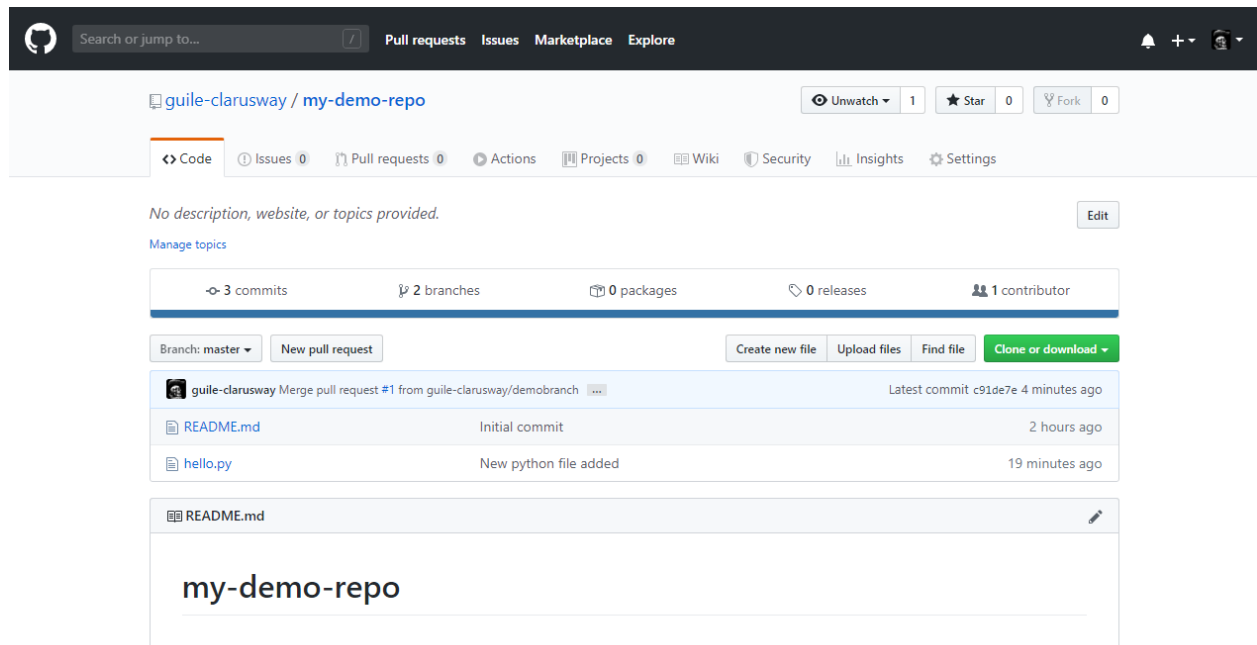
As you see there is no conflict to merge. Click **Merge pull request** and confirm it.

The screenshot shows a GitHub pull request interface. At the top, the title is "New python file added #1". Below the title, it says "guile-clarusway wants to merge 1 commit into master from demobranh". The pull request details show a commit "New python file added" by guile-clarusway. A green box highlights the status: "Continuous integration has not been set up" and "This branch has no conflicts with the base branch". The "Merge pull request" button is visible.

Pull request successfully merged and closed.

The screenshot shows the same GitHub pull request interface after it has been merged. The title is still "New python file added #1", but the status is now "Merged". The "Merge pull request" button is replaced by a "Revert" button. A message states "Pull request successfully merged and closed". The "demobranh" branch is now merged into the "master" branch.

Click **Code** tab to see how the repo looks like after merge. As you see **demobranh** is merged to the **master** branch.



Updating Local Repo

As you know we have created a new branch on our local repo and pushed it to remote repo. Now, our remote repo ([GitHub](#) repo) is on master branch and up to date, but local repo is not. So, let's go back to GitBash/Terminal and update our local master branch.

First, switch to master branch.

```
$ git checkout master
```

Then type

```
$ git pull origin master
```

to update our local repo.

```
guile@DESKTOP-ODR37SB: ~/my-demo-repo
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git pull origin master
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From https://github.com/guile-clarusway/my-demo-repo
* branch      master      -> FETCH_HEAD
   f5f7cd3..c91de7e master  -> origin/master
Updating f5f7cd3..c91de7e
Fast-forward
 hello.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```


Forking Projects

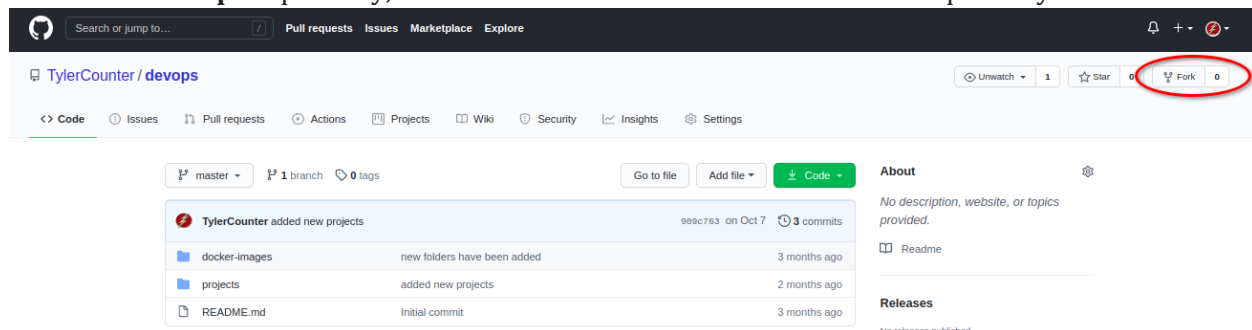
After using [GitHub](#) by yourself for a while, you may find yourself wanting to contribute to someone else's project. Or maybe you'd like to use someone's project as the starting point for your own. This process is known as **forking**.

Creating a **"fork"** is producing a personal copy of someone else's project. Forks act as a sort of bridge between the original repository and your personal copy. You can submit Pull Requests to help make other people's projects better by offering your changes up to the original project. Forking is at the core of social coding at [GitHub](#).

Fork the repository

First, go to <https://github.com/TylerCounter/devops>

To fork the **devops** repository, click the **Fork** button in the header of the repository.



When it's finished, you'll be taken to your copy of the **devops** repository.

Clone your fork

You've successfully forked the **devops** repository, but so far, it only exists on [GitHub](#). To be able to work on the project, you will need to clone it to your computer.

How you clone is up to you.

Some options are cloning with the command line, or by using [GitHub](#) Desktop.

Making and pushing changes

Go ahead and make a few changes to the project using your favorite text editor. You could, for example, change the text in README.MD to add your [GitHub](#) username.

When you're ready to submit your changes, stage, and commit your changes.

Right now, you've essentially told Git, "Okay, I've taken a snapshot of my changes!" You can continue to make more changes, and take more commit snapshots. When you're ready to push your changes up to [GitHub](#).com, push your changes to the remote.

Making a Pull Request

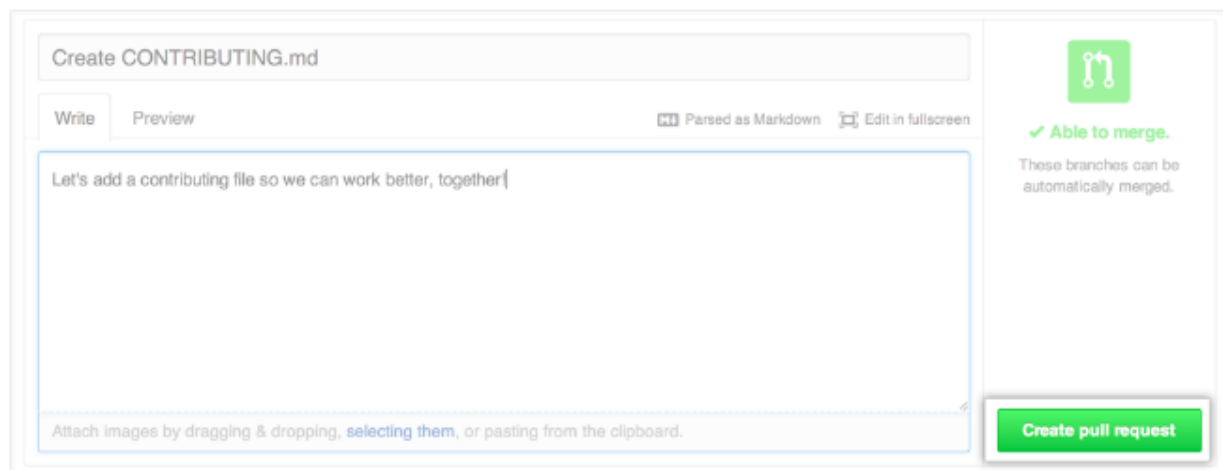
At last, you're ready to propose changes into the main project! This is the final step in producing a fork of someone else's project, and arguably the most important. If you've made a change that you feel would benefit the community as a whole, you should definitely consider contributing back.

To do so, head on over to the repository on [GitHub.com](https://github.com) where your project lives. For this example, it would be at <https://www.github.com/devops>. You'll see a banner indicating that you've recently pushed a new branch and that you can submit this branch "upstream," to the original repository:



Clicking on Compare and Pull Request sends you to a discussion page, where you can enter a title and optional description. It's important to provide as much useful information and a rationale for why you're making this Pull Request in the first place. The project owner needs to be able to determine whether your change is as useful to everyone as you think it is.

When you're ready to type out your heartfelt argument, click on Send a pull request.

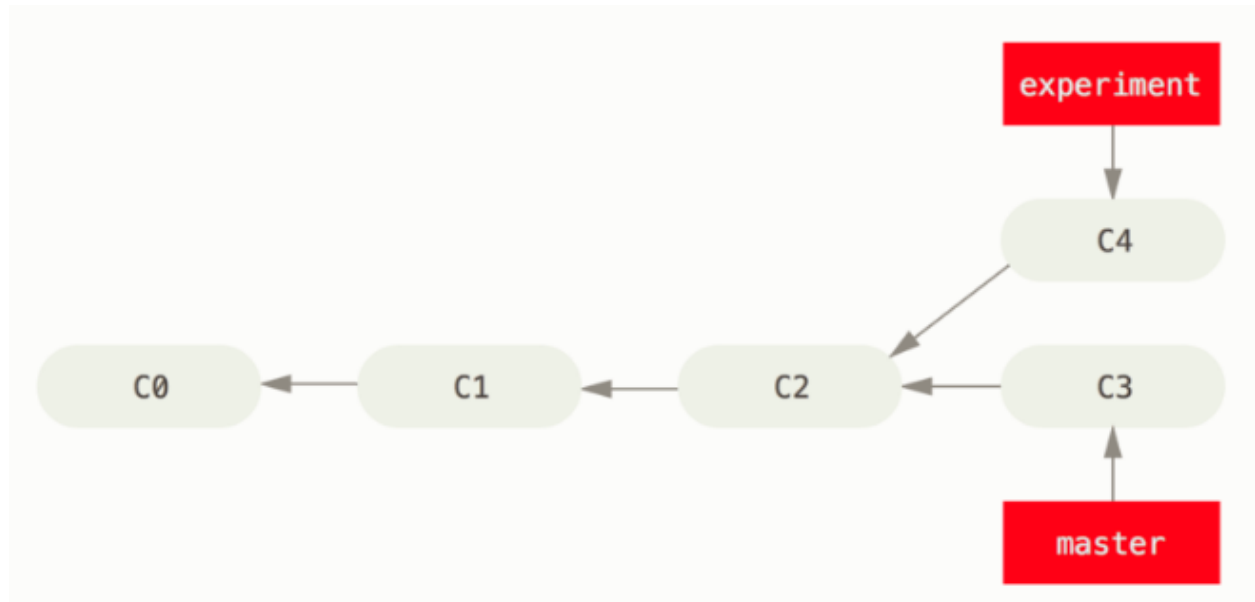


Pull Requests are an area for discussion. In this case, Tyler is very busy and probably won't merge your changes. For other projects, don't be offended if the project owner rejects your Pull Request, or asks for more information on why it's been made. It may even be that the project owner chooses not to merge your pull request, and that's totally okay. Your copy will exist in infamy on the Internet. And who knows—maybe someone you've never met will find your changes much more valuable than the original project. Share and share alike!

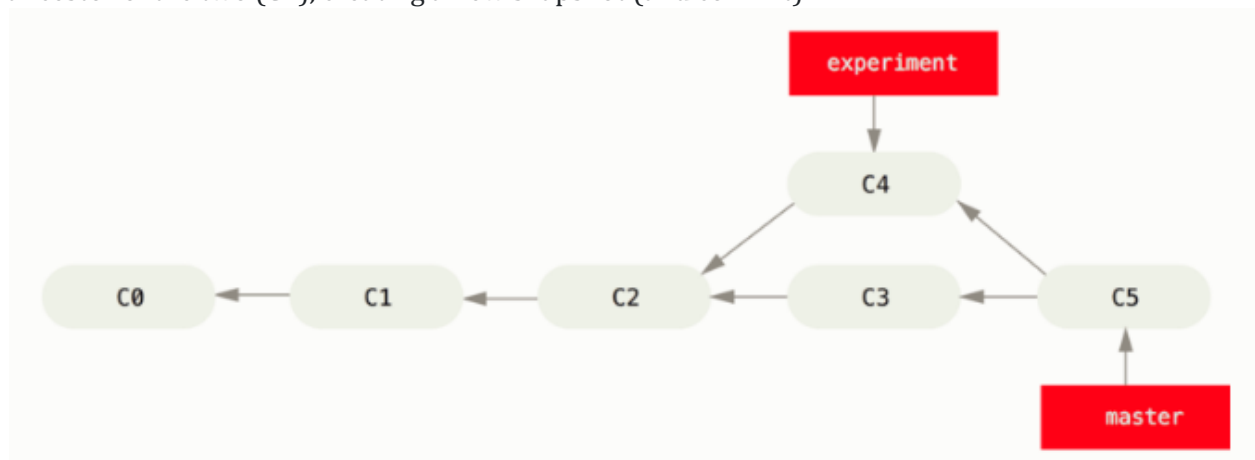
Rebasing

In Git, there are two main ways to integrate changes from one branch into another: the **merge** and the **rebase**.

If you go back to an earlier example from Basic Merging, you can see that you diverged your work and made commits on two different branches.



The easiest way to integrate the branches, as we've already covered, is the merge command. It performs a three-way merge between the two latest branch snapshots (C3 and C4) and the most recent common ancestor of the two (C2), creating a new snapshot (and commit).

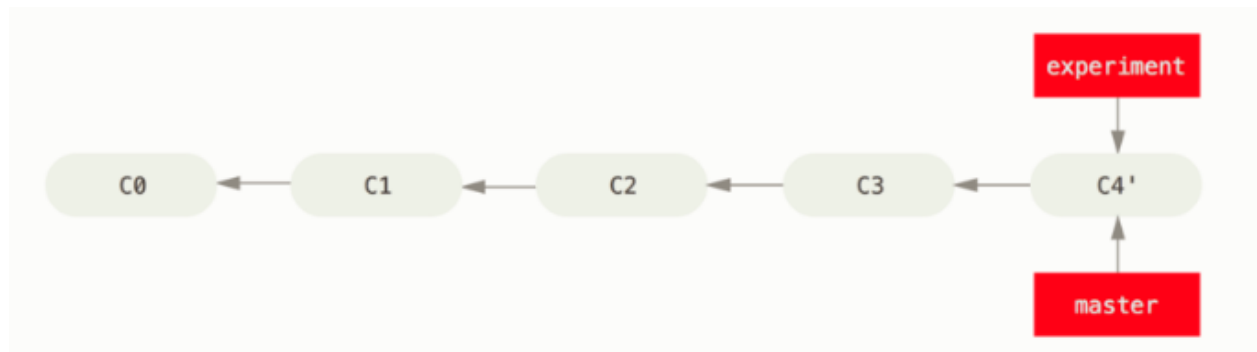


However, there is another way: you can take the patch of the change that was introduced in C4 and reapply it on top of C3. In Git, this is called **rebasing**. With the rebase command, you can take all the changes that were committed on one branch and replay them on a different branch.

For this example, you would check out the experiment branch, and then rebase it onto the master branch as follows:

```
$ git checkout experiment
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: added staged command
```

This operation works by going to the common ancestor of the two branches (the one you're on and the one you're rebasing onto), getting the diff introduced by each commit of the branch you're on, saving those diffs to temporary files, resetting the current branch to the same commit as the branch you are rebasing onto, and finally applying each change in turn.



Now, the snapshot pointed to by C4' is exactly the same as the one that was pointed to by C5 in the merge example. There is no difference in the end product of the integration, but rebasing makes for a cleaner history. If you examine the log of a rebased branch, it looks like a linear history: it appears that all the work happened in series, even when it originally happened in parallel.

Often, you'll do this to make sure your commits apply cleanly on a remote branch — perhaps in a project to which you're trying to contribute but that you don't maintain. In this case, you'd do your work in a branch and then rebase your work onto origin/master when you were ready to submit your patches to the main project. That way, the maintainer doesn't have to do any integration work — just a fast-forward or a clean apply.

Congratulations - end of lesson reached

Well done!