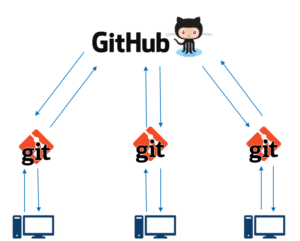
**GITHUB**

GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code. To understand exactly what GitHub is, you need to know two connected principles:

* Version control
* Git



## Step 1: Create a local git repository

When creating a new project on your local machine using git, you'll first create a new [**repository**](https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository)(or often, '**repo**', for short).

Getting a Git Repository

You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
2. You can **clone** an existing Git repository from elsewhere.

In either case, you end up with a Git repository on your local machine, ready for work.

### Initializing a Repository in an Existing Directory

for Windows:

$ cd C:/Users/user/my\_project

and type:

$ git init

### Cloning an Existing Repository

If you want to get a copy of an existing Git repository — for example, a project you’d like to contribute to — the command you need is git clone.

You clone a repository with git clone <url>. For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

$ git clone https://github.com/libgit2/libgit2

To use git we'll be using the terminal. If you don't have much experience with the terminal and basic commands, check out [this tutorial](http://mac.appstorm.net/how-to/utilities-how-to/how-to-use-terminal-the-basics/) (especially the 'Navigating the Filesystem' and 'Moving Around' sections).

To begin, open up a terminal and move to where you want to place the project on your local machine using the cd (change directory) command. For example, if you have a 'projects' folder on your desktop, you'd do something like:

mnelson:Desktop mnelson$ cd ~/Desktop

mnelson:Desktop mnelson$ mkdir myproject

mnelson:Desktop mnelson$ cd myproject/

INIT Command

mnelson:myproject mnelson$ git init

Initialized empty Git repository in /Users/mnelson/Desktop/myproject/.git/

git-init - Create an empty Git repository or reinitialize an existing one

SYNOPSIS

**git init** [-q | --quiet] [--bare] [--template=<template\_directory>]

[--separate-git-dir <git dir>]

[--shared[=<permissions>]] [directory]

**-q**

**--quiet**

Only print error and warning messages; all other output will be suppressed.

**--bare**

Create a bare repository. If GIT\_DIR environment is not set, it is set to the current working directory.

**--template=<template\_directory>**

Specify the directory from which templates will be used. (See the "TEMPLATE DIRECTORY" section below.)

**--separate-git-dir=<git dir>**

Instead of initializing the repository as a directory to either $GIT\_DIR or ./.git/, create a text file there containing the path to the actual repository. This file acts as filesystem-agnostic Git symbolic link to the repository.

If this is reinitialization, the repository will be moved to the specified path.

**--shared[=(false|true|umask|group|all|world|everybody|0xxx)]**

## Step 2:Add a new file to the repo

using any text editor you like or running a [touch](http://linux.die.net/man/1/touch) command.

mnelson:myproject mnelson$ touch mnelson.txt

mnelson:myproject mnelson$ ls

mnelson.txt

Touch Command

touch - change file timestamps

SYNOPSIS1

**touch** [*OPTION*]... *FILE*...

After creating the new file, you can use the [**git status**](http://git-scm.com/docs/git-status) command to see which files git knows exist.

mnelson:myproject mnelson$ git status

On branch master

Initial commit

Untracked files:

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

mnelson.txt

nothing added to commit but untracked files present (use "git add" to track)

Git status Command

git-status - Show the working tree status

SYNOPSIS2

**git status** [<options>…​] [--] [<pathspec>…​]

### An interlude: The staging environment, the commit, and you

A [**commit**](http://git-scm.com/docs/git-commit)is a record of what files you have changed since the last time you made a commit. Essentially, you make changes to your repo (for example, adding a file or modifying one) and then tell git to put those files into a commit.

Commits make up the essence of your project and allow you to go back to the state of a project at any point.

So, how do you tell git which files to put into a commit? This is where the [**staging environment**or **index**](https://git-scm.com/book/en/v2/Getting-Started-Git-Basics) come in. As seen in Step 2, when you make changes to your repo, git notices that a file has changed but won't do anything with it (like adding it in a commit).

To add a file to a commit, you first need to add it to the staging environment. To do this, you can use the [**git add**](http://git-scm.com/docs/git-add)**<filename>**command

mnelson:myproject mnelson$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: mnelson.txt

Git-add Command

git-add - Add file contents to the index

SYNOPSIS3

**git add** [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]

[--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]]

[--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing] [--renormalize]

[--chmod=(+|-)x] [--pathspec-from-file=<file> [--pathspec-file-nul]]

[--] [<pathspec>…​]

Please see [git-commit[1]](https://git-scm.com/docs/git-commit) for alternative ways to add content to a commit.

## Step 3:Add a file to the staging environment

Add a file to the staging environment using the **git add** command.

If you rerun the git status command, you'll see that git has added the file to the staging environment (notice the "Changes to be committed" line)

mnelson:myproject mnelson$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: mnelson.txt

To reiterate, the file has **not**yet been added to a commit, but it's about to be

## Step 4:Create a commit

It's time to create your first commit!

Run the command git commit -m "Your message about the commit"

mnelson:myproject mnelson$ git commit -m "This is my first commit!"

[master (root-commit) b345d9a] This is my first commit!

1 file changed, 1 insertion(+)

create mode 100644 mnelson.txt

The message at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo. Don't put a message like "asdfadsf" or "foobar".

## Step 5:Create a new branch

To make a new feature but are worried about making changes to the main project while developing the feature. This is where [**git branches**](https://git-scm.com/book/en/v1/Git-Branching-What-a-Branch-Is)come in.

mnelson:myproject mnelson$ git branch

master

\* my-new-branch

The branch name with the asterisk next to it indicates which branch you're pointed to at that given time.

Now, if you switch back to the master branch and make some more commits, your new branch won't see any of those changes until you [**merge**](http://git-scm.com/docs/git-merge) those changes onto your new branch.

Git-merge Command

git-merge - Join two or more development histories together

SYNOPSIS1

**git merge** [-n] [--stat] [--no-commit] [--squash] [--[no-]edit]

[--no-verify] [-s <strategy>] [-X <strategy-option>] [-S[<keyid>]]

[--[no-]allow-unrelated-histories]

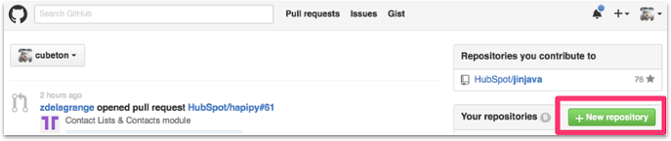
[--[no-]rerere-autoupdate] [-m <msg>] [-F <file>] [<commit>…​]

**git merge** (--continue | --abort | --quit)

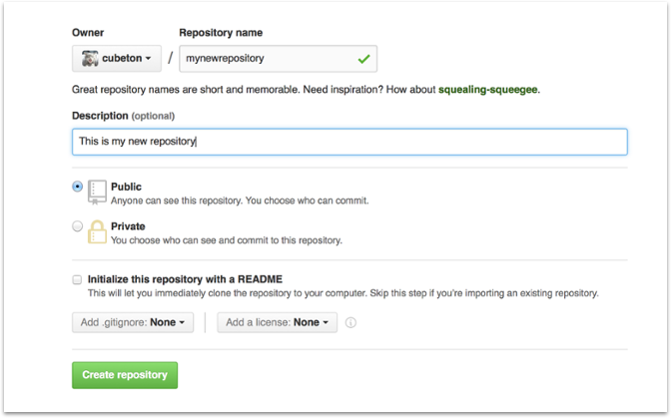
## Step 6:Create a new repository on GitHub

If you only want to keep track of your code locally, you don't need to use GitHub. But if you want to work with a team, you can use GitHub to collaboratively modify the project's code.

To create a new repo on GitHub, log in and go to the GitHub home page. You should see a green '+ New repository' button:



After clicking the button, GitHub will ask you to name your repo and provide a brief description:



When you're done filling out the information, press the 'Create repository' button to make your new repo.

GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the **'....or push an existing repository from the command line'** section:

mnelson:myproject mnelson$ git remote add origin https://github.com/cubeton/mynewrepository.git

mnelson:myproject mnelson$ git push -u origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 263 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/cubeton/mynewrepository.git

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

(You'll want to change the URL in the first command line to what GitHub lists in this section since your GitHub username and repo name are different.)

## Step 7:Push a branch to GitHub

Now we'll **push** the commit in your branch to your new GitHub repo. This allows other people to see the changes you've made. If they're approved by the repository's owner, the changes can then be merged into the master branch.

To push changes onto a new branch on GitHub, you'll want to run [**git push**](http://git-scm.com/docs/git-push) **origin yourbranchname**. GitHub will automatically create the branch for you on the remote repository:

mnelson:myproject mnelson$ git push origin my-new-branch

Counting objects: 3, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 313 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/cubeton/mynewrepository.git

\* [new branch] my-new-branch -> my-new-branch

Git-push Command

git-push - Update remote refs along with associated objects

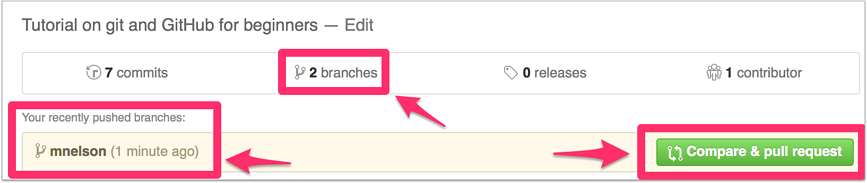
SYNOPSIS1

**git push** [--all | --mirror | --tags] [--follow-tags] [--atomic] [-n | --dry-run] [--receive-pack=<git-receive-pack>] [--repo=<repository>] [-f | --force] [-d | --delete] [--prune] [-v | --verbose] [-u | --set-upstream] [-o <string> | --push-option=<string>] [--[no-]signed|--signed=(true|false|if-asked)] [--force-with-lease[=<refname>[:<expect>]]] [--no-verify] [<repository> [<refspec>…​]]

You might be wondering what that "origin" word means in the command above. What happens is that when you clone a remote repository to your local machine, git creates an **alias** for you. In nearly all cases this alias is called "[**origin**](https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes)." It's essentially shorthand for the remote repository's URL. So, to push your changes to the remote repository, you could've used either the command:**git push git@github.com:git/git.git yourbranchname** or **git push origin yourbranchname**

(If this is your first time using GitHub locally, it might prompt you to log in with your GitHub username and password.)

If you refresh the GitHub page, you'll see note saying a branch with your name has just been pushed into the repository. You can also click the 'branches' link to see your branch listed there.

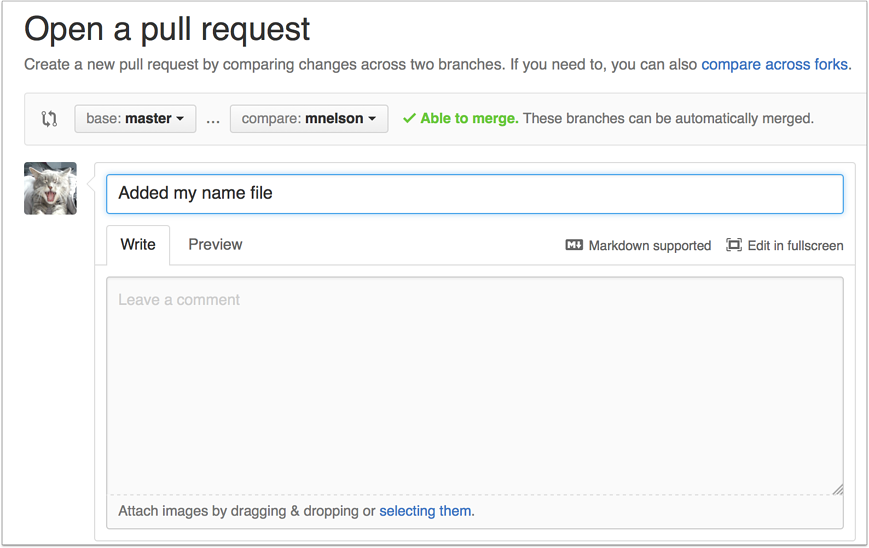
[](https://cloud.githubusercontent.com/assets/5241432/9189475/da30eb86-3fb6-11e5-934f-ca596a2cac69.png)

Now click the green button in the screenshot above. We're going to make a **pull request**!

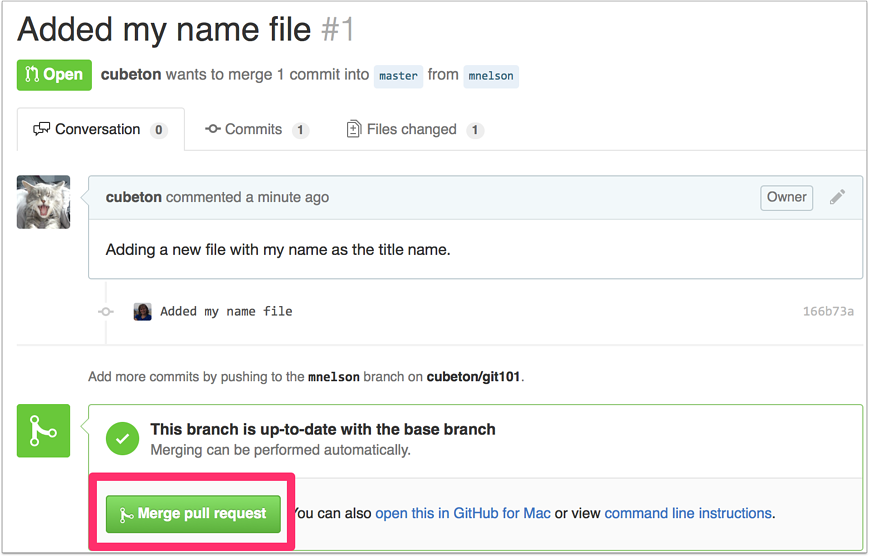
## Step 8:Create a Pull Request (PR)

A pull request (or PR) is a way to alert a repo's owners that you want to make some changes to their code. It allows them to review the code and make sure it looks good before putting your changes on the master branch.

This is what the PR page looks like before you've submitted it:

[](https://cloud.githubusercontent.com/assets/5241432/9189500/4688c07e-3fb7-11e5-99ed-d75b50ed9e48.png)

And this is what it looks like once you've submitted the PR request:

[](https://cloud.githubusercontent.com/assets/5241432/9189528/b39a7176-3fb7-11e5-87b1-7fed3e63b6bb.png)

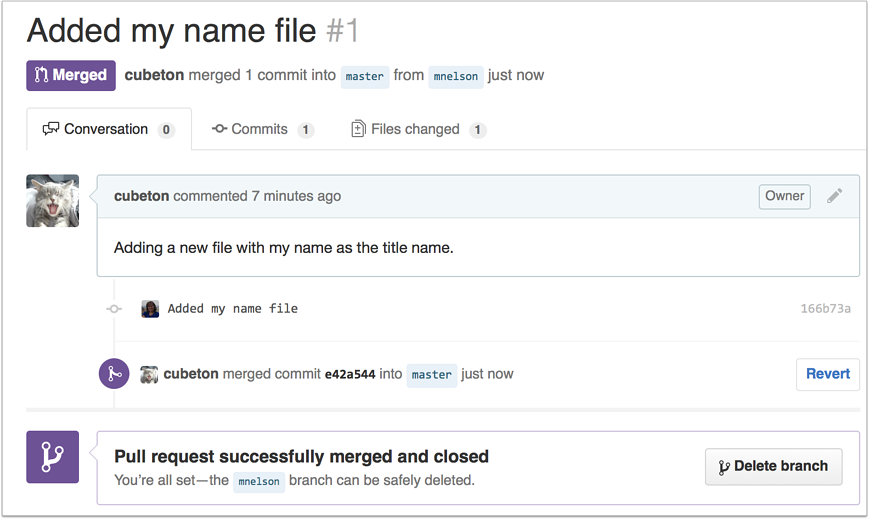
You might see a big green button at the bottom that says 'Merge pull request'. Clicking this means you'll merge your changes into the master branch.

Note that this button won't always be green. In some cases it'll be grey, which means you're faced with a **merge conflict**. This is when there is a change in one file that conflicts with a change in another file and git can't figure out which version to use. You'll have to manually go in and tell git which version to use.

Sometimes you'll be a co-owner or the sole owner of a repo, in which case you may not need to create a PR to merge your changes. However, it's still a good idea to make one so you can keep a more complete history of your updates and to make sure you always create a new branch when making changes.

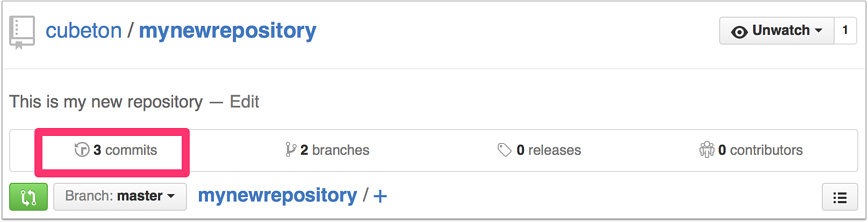
## Step 9:Merge a PR

Go ahead and click the green 'Merge pull request' button. This will merge your changes into the master branch.

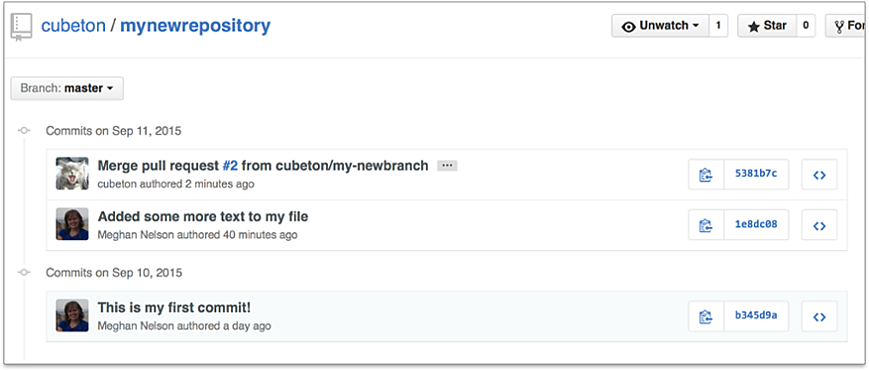
[](https://cloud.githubusercontent.com/assets/5241432/9189587/76631d98-3fb8-11e5-9fdb-17e7dec1c2a4.png)

When you're done, I recommend deleting your branch (too many branches can become messy), so hit that grey 'Delete branch' button as well.

You can double check that your commits were merged by clicking on the 'Commits' link on the first page of your new repo.



This will show you a list of all the commits in that branch. You can see the one I just merged right up top (Merge pull request #2).



You can also see the [**hash code**](https://git-scm.com/docs/git-hash-object) of the commit on the right hand side. A hash code is a unique identifier for that specific commit. It's useful for referring to specific commits and when undoing changes (use the [**git revert**](http://git-scm.com/docs/git-revert) <hash code number> command to backtrack).

## Step 10:Get changes on GitHub back to your computer

Right now, the repo on GitHub looks a little different than what you have on your local machine. For example, the commit you made in your branch and merged into the master branch doesn't exist in the master branch on your local machine.

In order to get the most recent changes that you or others have merged on GitHub, use the **git pull origin master**command (when working on the master branch).

mnelson:myproject mnelson$ git pull origin master

remote: Counting objects: 1, done.

remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (1/1), done.

From https://github.com/cubeton/mynewrepository

\* branch master -> FETCH\_HEAD

b345d9a..5381b7c master -> origin/master

Merge made by the 'recursive' strategy.

mnelson.txt | 1 +

1 file changed, 1 insertion(+)

This shows you all the files that have changed and how they've changed.

Now we can use the [**git log**](http://git-scm.com/docs/git-log) command again to see all new commits.

(You may need to switch branches back to the master branch. You can do that using the **git checkout master** command.)

mnelson:myproject mnelson$ git log

commit 3e270876db0e5ffd3e9bfc5edede89b64b83812c

Merge: 4f1cb17 5381b7c

Author: Meghan Nelson <mnelson@hubspot.com>

Date: Fri Sep 11 17:48:11 2015 -0400

Merge branch 'master' of https://github.com/cubeton/mynewrepository

commit 4f1cb1798b6e6890da797f98383e6337df577c2a

Author: Meghan Nelson <mnelson@hubspot.com>

Date: Fri Sep 11 17:48:00 2015 -0400

added a new file

commit 5381b7c53212ca92151c743b4ed7dde07d9be3ce

Merge: b345d9a 1e8dc08

Author: Meghan Nelson <meghan@meghan.net>

Date: Fri Sep 11 17:43:22 2015 -0400

Added some more text to my file

commit 1e8dc0830b4db8c93efd80479ea886264768520c

Author: Meghan Nelson <mnelson@hubspot.com>

Date: Fri Sep 11 17:06:05 2015 -0400

Added some more text to my file

commit b345d9a25353037afdeaa9fcaf9f330effd157f1

Author: Meghan Nelson <mnelson@hubspot.com>

Date: Thu Sep 10 17:42:15 2015 -0400

This is my first commit!