



Github

Version Control



Contents

1. VERSION CONTROL	3
2. CENTRALIZED VS DISTRIBUTED	4
2.1. CENTRALIZED VERSION CONTROL.....	4
2.2. DISTRIBUTED VERSION CONTROL.....	4
3. GIT BASICS	6
3.1. GIT BASIC COMMANDS	6
3.2. BRANCHING & MERGING	6
3.3. GITHUB.....	7
3.3.1. Terminology.....	8
3.3.2. Common Commands.....	8
4. GIT VS GITHUB	10
4.1. SIGNUP GITHUB	12
4.2. PUSHING TO GITHUB.....	14
4.3. UPDATING REMOTE	16
4.4. RECAP.....	18
5. UPLOAD PERSONAL WEBSITE	19

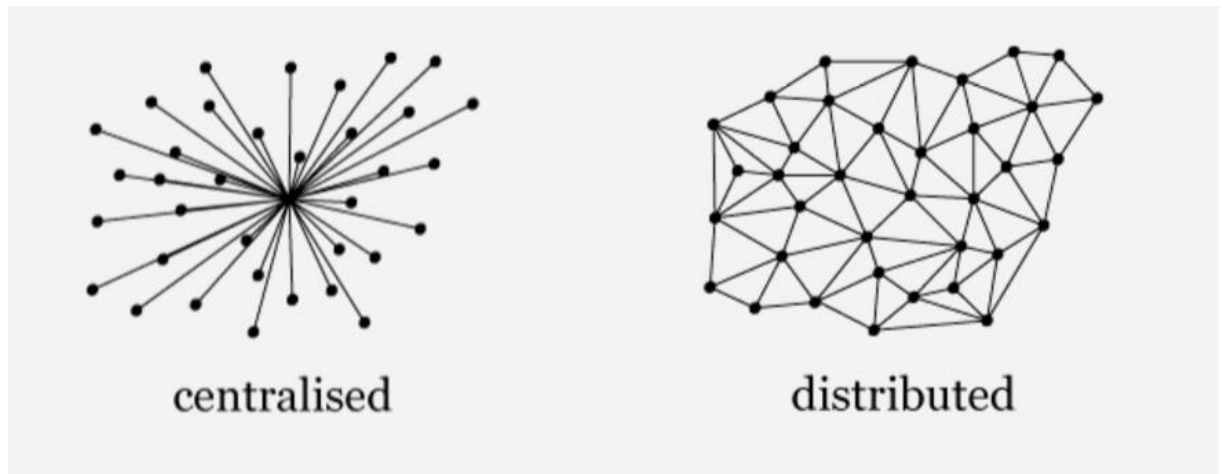
1. Version Control

Version control systems help development teams keep track of changes to their source code over time. It is also commonly referred to as SCM (Source Code Management).

Version Control Systems keep track of every modification made to the code base and allow you to go back and forward in time as it relates to changes in your source code. It also facilitates comparing code at different points in time to help fix any bugs or issues that may have been the result of a specific change.

2. Centralized vs Distributed

For this course we are going to focus on Distributed Version Control with Git but it is worth discussing the difference between Distributed and Centralized Version Control.



2.1. Centralized Version Control

Centralized Version Controls systems, like SVN (Subversion), require a single **central** copy of your project typically on a server somewhere. Developers **commit** their changes to this central copy. If this central copy is not available, the developer is not able to save changes to their work within the VCS until they have access to the server.

Typical Workflow for CVC

- Developer checks out code from version control or pulls down latest changes to code already checked out.
- Developer makes some changes and tests their work
- Developer commits changes back to central repository for others to pull into their local copies

2.2. Distributed Version Control

Distributed Version Control systems, like Git, don't rely on a central server to store all the versions of a project's files. You can however utilize a service like GitHub to simulate this functionality. Every developer will **clone** an existing repository (or, repo) from a service like GitHub, or initiate a repo locally. A copy of a repo, and the full history of the project, is stored locally with no need to connect to the centralized server to track changes.

The developer can make changes to the source code and commit those changes to the local repo on their system.

When they are ready to share changes with other developers they can **push** their changes, and all the attached history, to the remote repo. Once pushed, other developers can now update their local copies with those changes.

Typical Workflow for DVC

- Developer checks out code from version control or pulls down latest changes to code already checked out.
- Developer makes some changes and tests their work.
- Developer makes a local **commit** that represents the changes made
- Developer repeats this cycle locally until they are ready to share their changes with other developers.
- Developer commits changes back to central repo for others to **pull** into their local copies, or provide a patch file to another developer if they choose not to use a service like GitHub.

3. Git Basics

3.1. Git Basic Commands

`$ git` => This is the top level command. You will typically type git and then an action you want git to execute.

`$ git --version` => You may remember this command when we checked what version of git we had installed by running

`$ git init` => This will tell git that the directory you are in when this command is run should be tracked by git. You are creating the repository when you run this command.

`$ git add` => .. followed by a path will let git know you want to add files at that path to be tracked by version control and staged for commit.

`$ git commit` => Takes your staged files and commits them to version control and creates a point in your version control history that can be referenced later. You will almost always find the -m **COMMIT MESSAGE** flag used to provide a message related to your commit.

`$ git status` => Displays the files that have changed in your repo as it relates to the latest commit. It will show what files have been modified and which files are staged for your next commit. It also shows information about which branch you are currently working with.

`$ git diff` => Displays the line by line differences between files in your repo compared to the last commit.

3.2. Branching & Merging

Branching is the concept of going down a different path in the code base from a specific commit. Usually branching off master in its latest state to make a particular set of changes and not modify the state of the the original code you branched from. Merging is the concept of applying changes made in one branch to another.

Commands:

`$ git branch` => Shows you what branches exist in your repository and what branch you are on. Your current branch is marked with a *

git branch with a branch name provided creates the branch but does not check it out.

```
$ git branch -d BRANCH_NAME
```

 => Deletes a branch

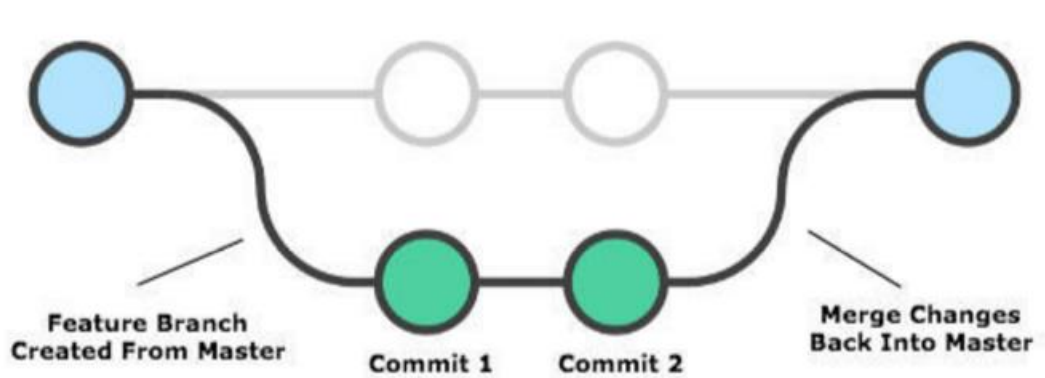
```
$ git checkout BRANCH_NAME
```

 => Moves you from current branch to specified branch. Adding -b between "checkout" and the branch name creates a branch and checks out that branch.

```
$ git merge BRANCH_NAME
```

 => Merges the named branch into the branch you are currently on. All the changes that exist in the branch you are merging from now exist in your branch.

Typical Git Workflow:



⇒ It is best practices to create a branch for each local grouping of changes you are making to your project.

3.3. GitHub



- ⇒ GitHub is a service that allows you to make your Git repositories available to other developers, and the world, through a Web-based graphical interface. It has options on how to grant access to your code, as well as how people can contribute to your code. It also provides several collaboration features, such as a wikis and basic task management tools, for every project.
- ⇒ Github is web-based, remote repository. It incorporates all of the functionality of git, as well as some enhanced features. Github also has a desktop version of its software so those that don't have command line skills can use github as well.
- ⇒ Additionally, github makes git a social atmosphere. It allows for code collaboration, commenting, thorough documentation, wikis, bug tracking and more!

3.3.1.Terminology

Repository => A repo(sitory) is a location where all the files for a particular project are stored. Each project will have its own repo, and can be accessed by a unique URL.

Forking => Creating a new project based off of a project that already exists. If you find a project on GitHub that you'd like to contribute to, you can fork the repo, make the changes you'd like, and release the revised project as a new repo. If the original repo that you forked to create your new project gets updated, you can easily add those updates to your current fork. You can also request that changes you have made can be merged into the original project. This is facilitated through Pull Requests.

Pull Requests => A pull request is a way of facilitating a merge through the GitHub interface. If you or someone on your development team has made some changes in a branch or a forked repository, you can create a pull request asking for those changes to be merged into the master branch or any other branch that exists in the repo. The owner of the project can then review your changes and merge those changes if all is good. Pull requests can be created from forked repos as well.

3.3.2.Common Commands

`$ git remote add` => To communicate with the outside world, git uses what are called remotes. These are repos other than the one on your local disk which you can push your changes into (so that other people can see them) or pull from (so that you can get others changes).

`$ git remote add NAME_OF_REMOTE URL_OF_REMOTE` => The command creates a new remote called whatever you put in place of NAME_OF_REMOTE located at URL_OF_REMOTE. NAME_OF_REMOTE is usually origin and URL_OF_REMOTE is the link to your GitHub repo. This lets git know that you are connected to a remote repo.

`$ git push` => This is command says, "Push the commits in the local branch to the remote branch." Once this is executed, your commits since your last push will be available on GitHub.

`$ git push -u origin master` => Our initial push may look something like which says, **Push my code to my remote named origin into the master branch.** Once this has been done the first time you can just use git push from that branch and don't need the rest of the command.


`$ git pull` => This is command says, **Pull the commits in the remote branch to the local branch.** Once this is executed, any commits made to the remote branch since your last pull will be available in your local copy of the repo.

`$ git clone` => This command is usually followed by the URL of the remote repo on GitHub you would like to **clone**, or make a local copy of, on your computer. On any repo you will see a **Clone or download** button. Clicking that button will provide you with the url required to clone that repo.

4. Git vs Github

- ⇒ Git is the most highly used, and now pretty much standard, version control system. It is primarily used in software development, but... you could technically use it as part of anything like writing an essay.
- ⇒ Version control is a way of saving different **versions** or **snapshots** of your code. We can save step 1, step 2, step 3 and so on. This is so if we make a major mistake in step 4, we can easily revert back to a previous working version of our code. These versions are saved in a local (on your computer) repository.

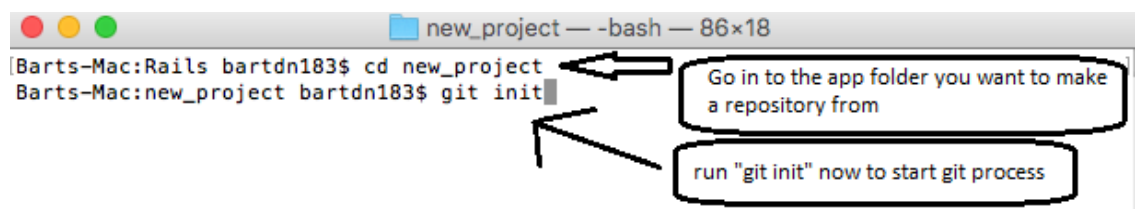
Check if you have git installed!



```
Barts-Mac:Rails bartdn183$ git --version
git version 2.8.2
Barts-Mac:Rails bartdn183$
```

How to use git to keep track of your progress?

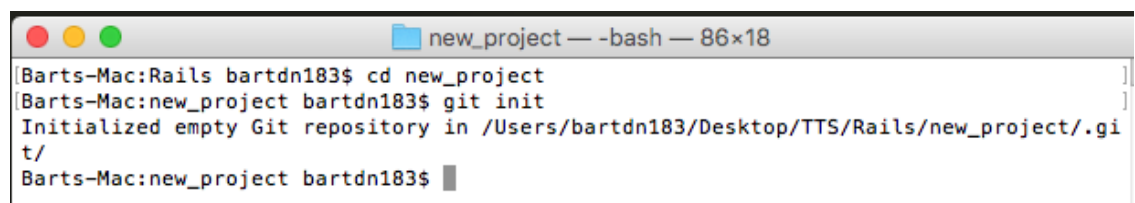
- 1) First initialize your repository. This will start the process of git tracking what is in your current directory.



```
Barts-Mac:Rails bartdn183$ cd new_project
Barts-Mac:new_project bartdn183$ git init
```

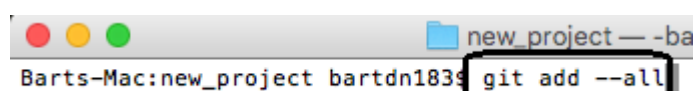
Go in to the app folder you want to make a repository from

run "git init" now to start git process

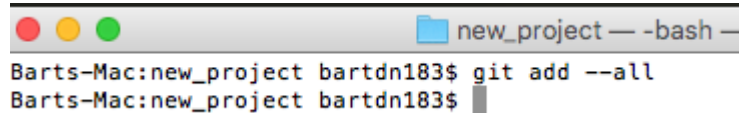


```
Barts-Mac:Rails bartdn183$ cd new_project
Barts-Mac:new_project bartdn183$ git init
Initialized empty Git repository in /Users/bartdn183/Desktop/TTS/Rails/new_project/.git/
Barts-Mac:new_project bartdn183$
```

- 2) If you have already some files in your repository you will want to add those files to a staging area. We can only commit files have been added to this staging area.

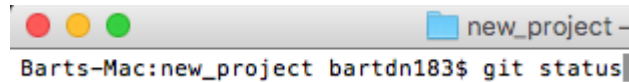


```
Barts-Mac:new_project bartdn183$ git add --all
```

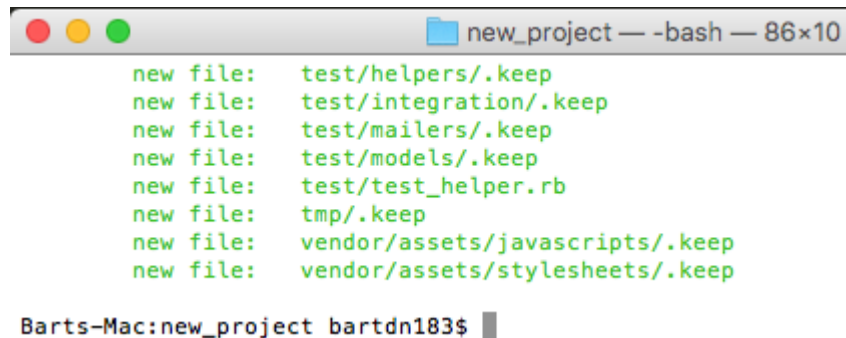


```
new_project — -bash —  
Barts-Mac:new_project bartdn183$ git add --all  
Barts-Mac:new_project bartdn183$
```

Check status of you repository!



```
new_project —  
Barts-Mac:new_project bartdn183$ git status
```



```
new_project — -bash — 86x10  
  
new file:   test/helpers/.keep  
new file:   test/integration/.keep  
new file:   test/mailers/.keep  
new file:   test/models/.keep  
new file:   test/test_helper.rb  
new file:   tmp/.keep  
new file:   vendor/assets/javascripts/.keep  
new file:   vendor/assets/stylesheets/.keep  
  
Barts-Mac:new_project bartdn183$
```

Yay! Our first commit!

- ⇒ Now we have a saved version of our application! No matter where are in the timeline of our application, we can always come back to this spot.

This is useful for a variety of reasons:

- 1) Revert back to this when you make a killer mistake.
- 2) Revert back to this if you want to pivot your application and to a spot where you can pick up.
- 3) Loop back at your code to see what you were thinking at that point and time.

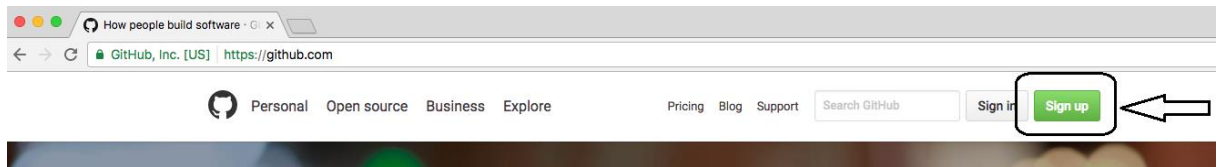
Local Vs Remote?

- ⇒ Right now your commit is just living on your local repository (or locally on your computer). Often times we want our code to live in a remote repository (somewhere else, usually the cloud) so that we (or someone else) can access it from somewhere other than our own computer.
- ⇒ The most popular remote repository system is Github, the developer social network. We literally create a copy of our repository on Github, and now anyone can access our code! This is huge for open source, innovation, learning and collaboration.

4.1. Signup Github

1) If you do not have a GitHub account, let's create one now at <https://github.com> . You will have to verify your email address to complete the sign-up process.

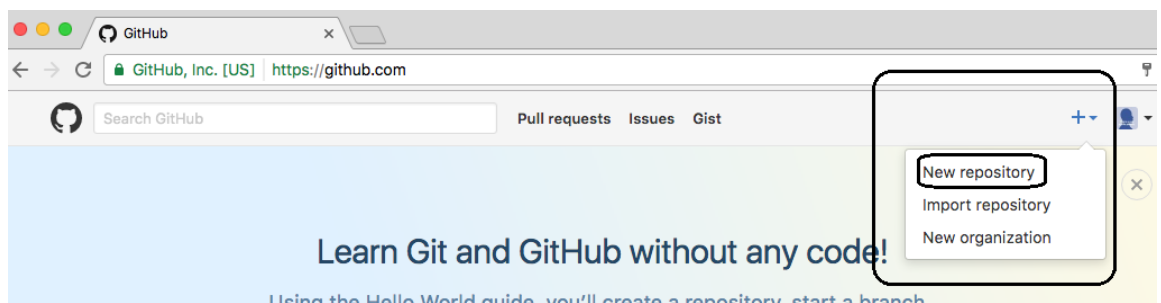
2) Click on **Sign Up**:



3) Create your Personal Account.

4) Sign In to your Github Account.

5) Go to the + in the upper right hand corner and hit create new repository.



6) Give your Repository a name and you can give it a brief description.

Create a New Repository

GitHub, Inc. [US] <https://github.com/new>

Search GitHub Pull requests Issues Gist

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: bartdn183

Repository name: new_project ✓

Great repository names are short and memorable. Need inspiration? How about **super-duper-sniffle**.

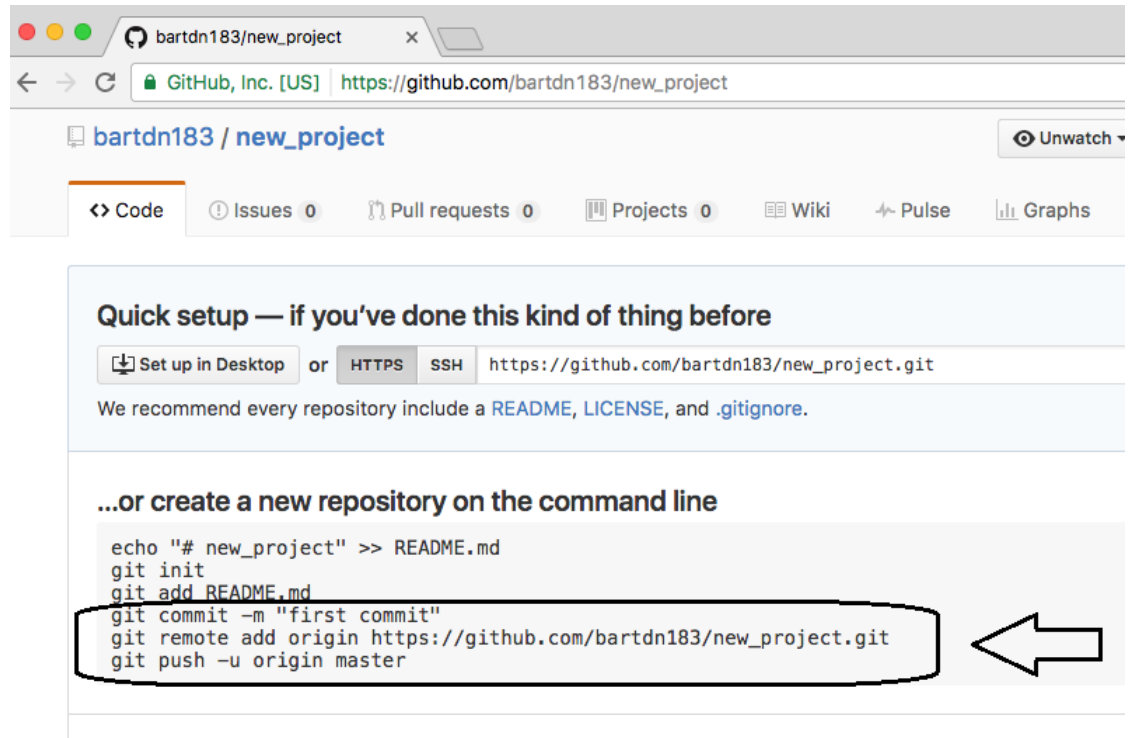
Description (optional)

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

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

☐ Initialize this repository with a README

- 7) Leave it **public** (this means anyone can see it), but it's free.
- 8) You do NOT need to initialize a readme.
- 9) Click on **Create repository**.
- 10) The next page will give you all the instructions on how to push files to your new remote repo. We have already done already a few steps but not the last 3 steps.



4.2. Pushing To Github

⇒ We will create first a commit to github to get all the files ready.

```
Barts-Mac:new_project bartdn183$ git commit -m "First Commit"
```

```
create mode 100644 test/fixtures/files/.keep
create mode 100644 test/helpers/.keep
create mode 100644 test/integration/.keep
create mode 100644 test/mailers/.keep
create mode 100644 test/models/.keep
create mode 100644 test/test_helper.rb
create mode 100644 tmp/.keep
create mode 100644 vendor/assets/javascripts/.keep
create mode 100644 vendor/assets/stylesheets/.keep
Barts-Mac:new_project bartdn183$
```

⇒ Now we will link our local repository to our remote repository so that they can pass files between each other:

```
Barts-Mac:new_project bartdn183$ git remote add origin https://github.com/bartdn183/new_project.git
```

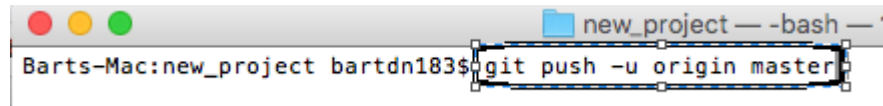


```

Barts-Mac:new_project bartdn183$ git remote add origin https://github.com/bartdn183/new_project.git
Barts-Mac:new_project bartdn183$ git remote -v
origin https://github.com/bartdn183/new_project.git (fetch)
origin https://github.com/bartdn183/new_project.git (push)
Barts-Mac:new_project bartdn183$

```

⇒ Now that they are linked we will push our local files to the remote:

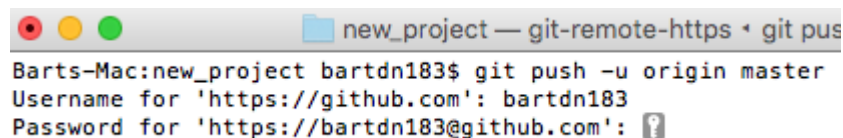


```

Barts-Mac:new_project bartdn183$ git push -u origin master

```

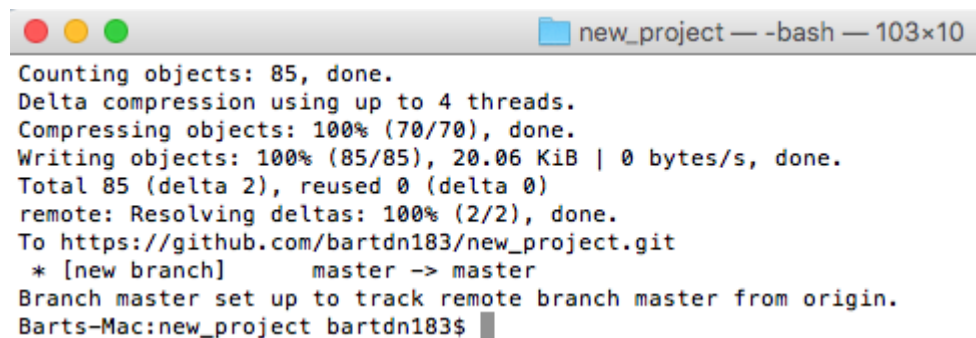
⇒ You will probably need to sign in with username and password:



```

Barts-Mac:new_project bartdn183$ git push -u origin master
Username for 'https://github.com': bartdn183
Password for 'https://bartdn183@github.com':

```

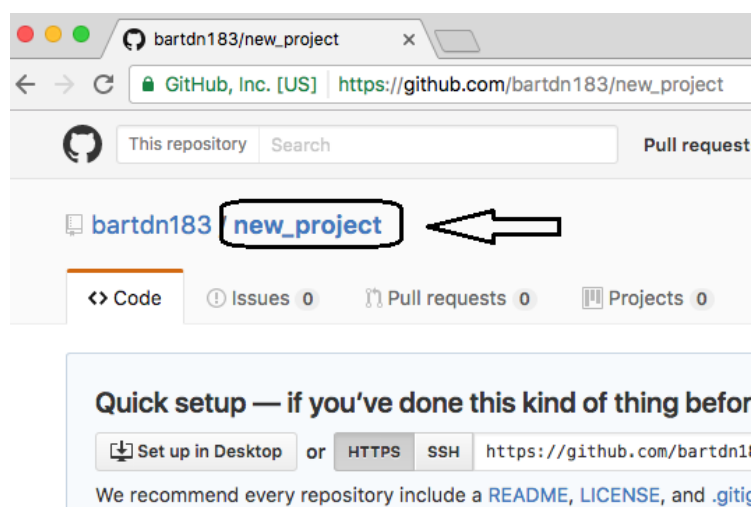


```

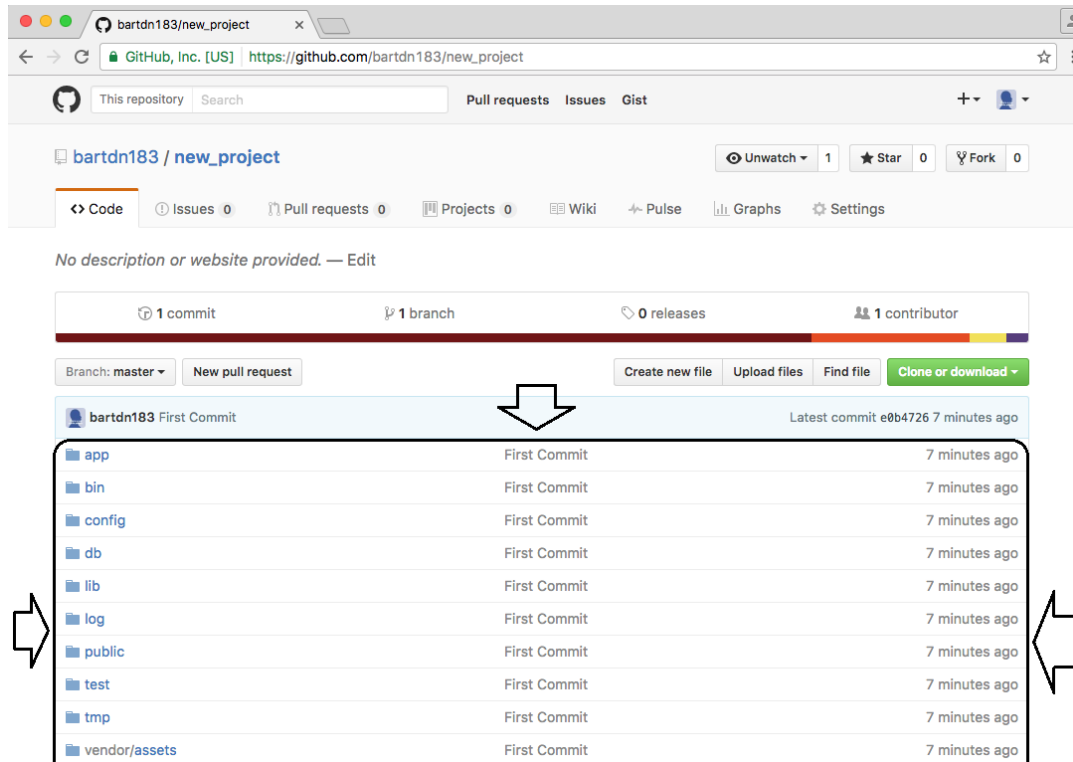
Barts-Mac:new_project bartdn183$ git push -u origin master
Counting objects: 85, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (70/70), done.
Writing objects: 100% (85/85), 20.06 KiB | 0 bytes/s, done.
Total 85 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/bartdn183/new_project.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
Barts-Mac:new_project bartdn183$

```

⇒ Click now on your project name:



⇒ All the files of our project are on the remote!



4.3. Updating Remote

⇒ Now I'll show you how you can update your remote if you did some changes in your code and you to have it in your repository. I will just drop **bread.jpg** file to the **app/assets/images** folder. Now I will start to update my repository.

1) Add everything to your local repository:

```
Barts-Mac:new_project bartdn183$ git add --all
```

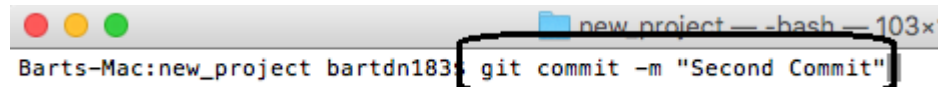
2) Check status:

```
Barts-Mac:new_project bartdn183$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

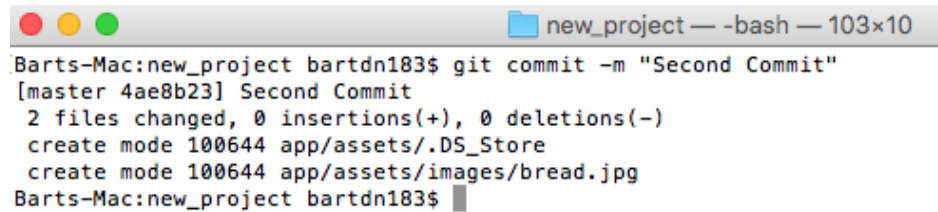
        new file:   app/assets/.DS_Store
        new file:   app/assets/images/bread.jpg

Barts-Mac:new_project bartdn183$
```


3) Commit your files:

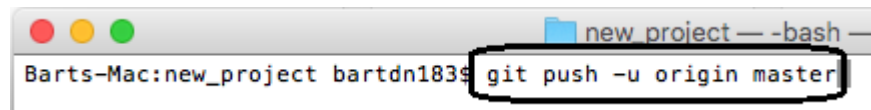


```
Barts-Mac:new_project bartdn183$ git commit -m "Second Commit"
```

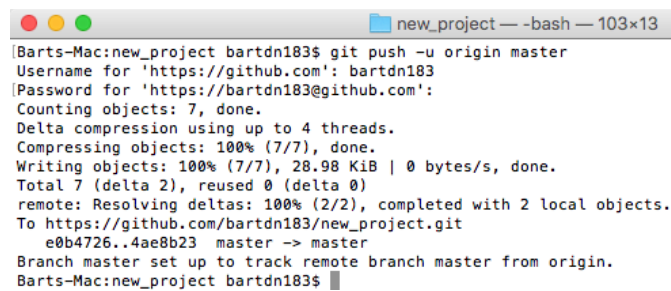


```
Barts-Mac:new_project bartdn183$ git commit -m "Second Commit"
[master 4ae8b23] Second Commit
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 app/assets/.DS_Store
create mode 100644 app/assets/images/bread.jpg
Barts-Mac:new_project bartdn183$
```

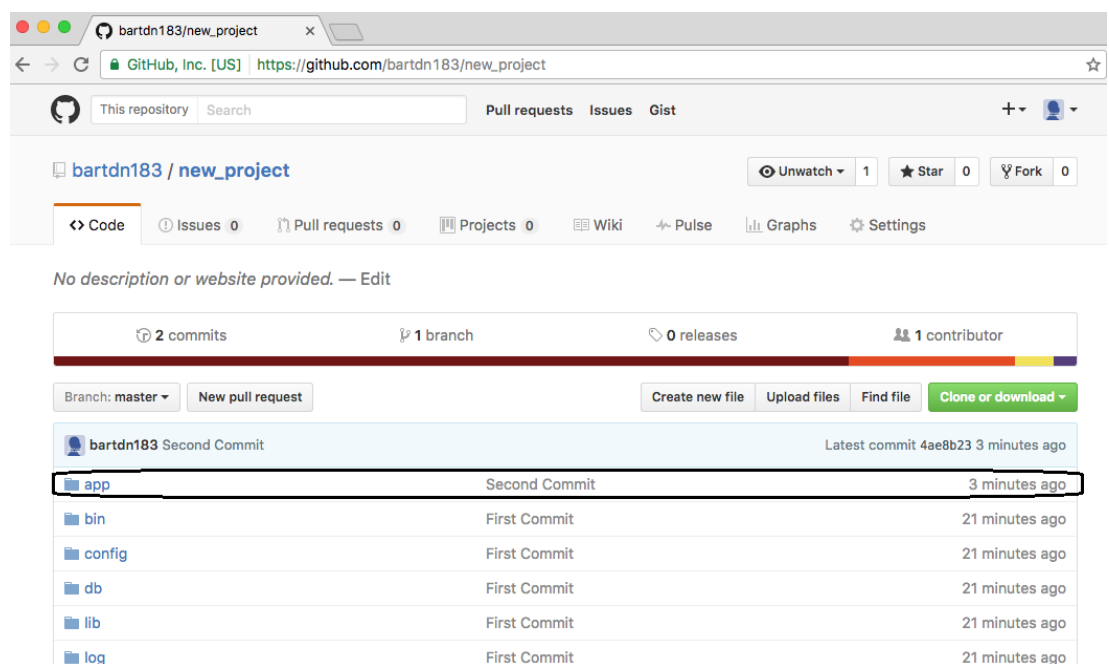
4) Push your updates to the remote:



```
Barts-Mac:new_project bartdn183$ git push -u origin master
```



```
Barts-Mac:new_project bartdn183$ git push -u origin master
Username for 'https://github.com': bartdn183
Password for 'https://bartdn183@github.com':
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 28.98 KiB | 0 bytes/s, done.
Total 7 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/bartdn183/new_project.git
 e0b4726..4ae8b23 master -> master
Branch master set up to track remote branch master from origin.
Barts-Mac:new_project bartdn183$
```

5) I added a picture in the **app/assets/images** folder. As you can see this is updated!


bartdn183/new_project

GitHub, Inc. [US] | https://github.com/bartdn183/new_project

This repository Search Pull requests Issues Gist

bartdn183 / new_project Unwatch 1 Star 0 Fork 0

<> Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

No description or website provided. — Edit

2 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

Commit	Message	Time
bartdn183 Second Commit	Second Commit	3 minutes ago
bin	First Commit	21 minutes ago
config	First Commit	21 minutes ago
db	First Commit	21 minutes ago
lib	First Commit	21 minutes ago
log	First Commit	21 minutes ago

4.4. Recap

1) New repository:

- `git init`
- `git add --all`
- `git status`

Create repository on Github website.

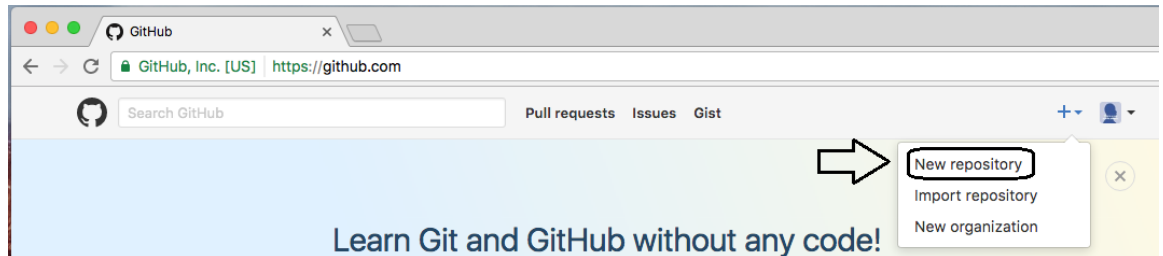
- `git commit -m "first commit"`
- `git remote add origin`
`https://github.com/*username*/*project_name*.git`
- `git remote -v`
- `git push -u origin master`

2) Update existing repository:

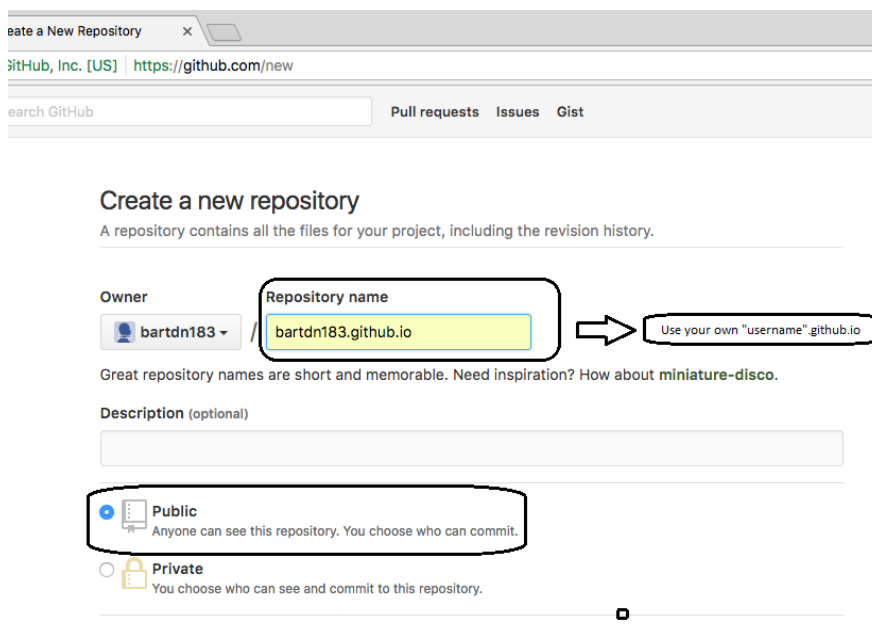
- `git add --all`
- `git status`
- `git commit -m "second commit"`
- `git push -u origin master` (or `git push`)

5. Upload Personal Website

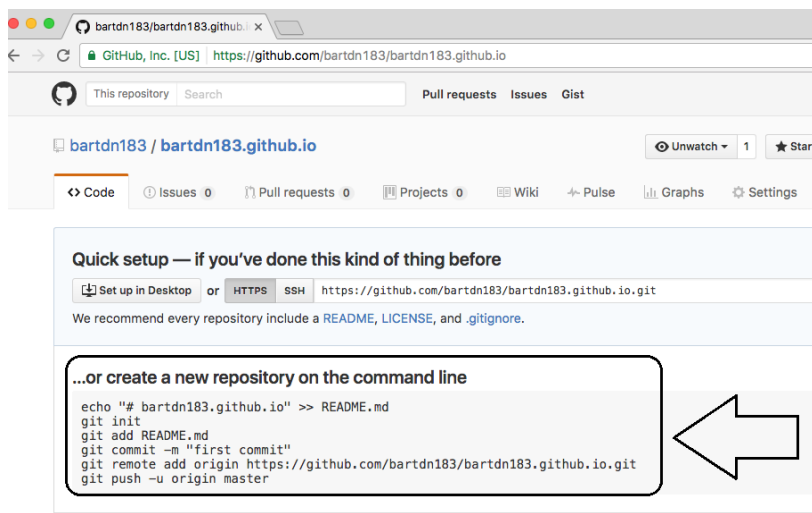
1) Log in to your Github Account and create a new repository!



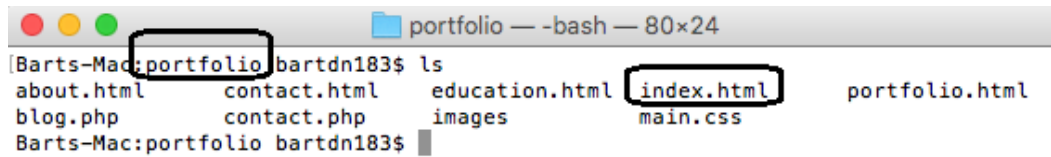
2) Create the new repository => ***yourusername.github.io***



⇒ Click the green **Create repository** button.




- 3) Go in Terminal to your website root folder. **index.html** needs to be your primary website file!



```
portfolio — -bash — 80x24
Barts-Mac:portfolio bartdn183$ ls
about.html  contact.html  education.html  index.html  portfolio.html
blog.php    contact.php   images         main.css
Barts-Mac:portfolio bartdn183$
```

- 4) Now run following git commands:

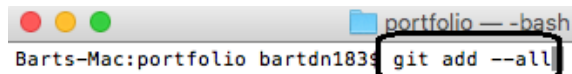
1. git init



```
portfolio —
Barts-Mac:portfolio bartdn183$ git init

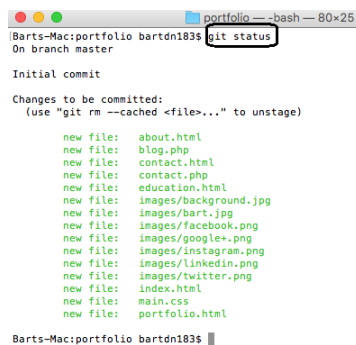
portfolio — -bash — 80x24
Barts-Mac:portfolio bartdn183$ git init
Initialized empty Git repository in /Users/bartdn183/Desktop/TTS/front_end/portfolio/.git/
```

2. git add --all (or git add .)



```
portfolio — -bash
Barts-Mac:portfolio bartdn183$ git add --all
```

3. git status



```
portfolio — -bash — 80x25
Barts-Mac:portfolio bartdn183$ git status
On branch master

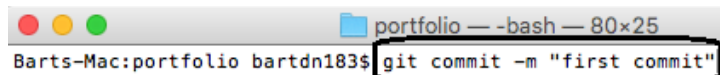
Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

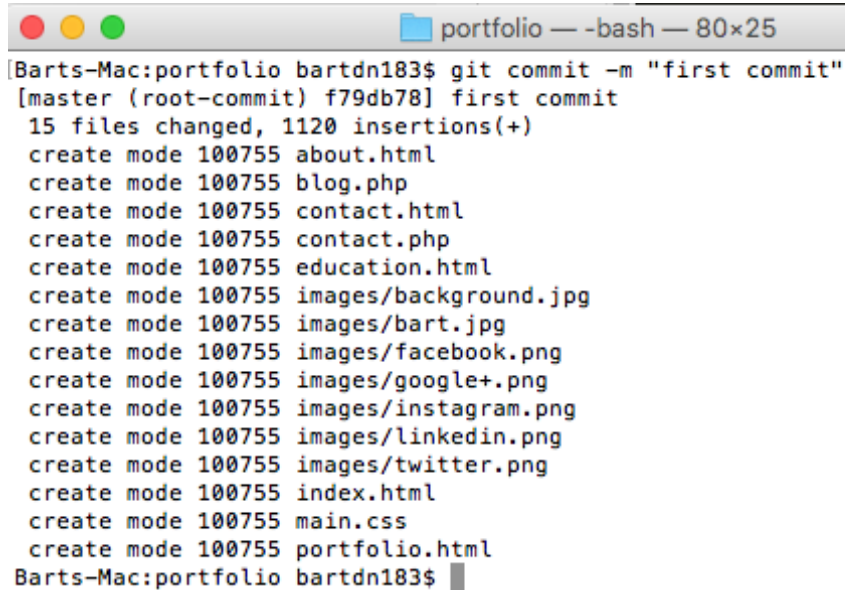
    new file:   about.html
    new file:   blog.php
    new file:   contact.html
    new file:   contact.php
    new file:   education.html
    new file:   images/background.jpg
    new file:   images/bart.jpg
    new file:   images/facebook.png
    new file:   images/google+.png
    new file:   images/instagram.png
    new file:   images/linkedin.png
    new file:   images/twitter.png
    new file:   index.html
    new file:   main.css
    new file:   portfolio.html

Barts-Mac:portfolio bartdn183$
```

4. git commit -m "first commit"



```
portfolio — -bash — 80x25
Barts-Mac:portfolio bartdn183$ git commit -m "first commit"
```

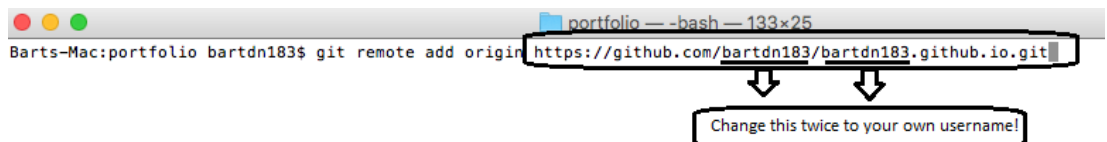


```

Barts-Mac:portfolio bartdn183$ git commit -m "first commit"
[master (root-commit) f79db78] first commit
15 files changed, 1120 insertions(+)
create mode 100755 about.html
create mode 100755 blog.php
create mode 100755 contact.html
create mode 100755 contact.php
create mode 100755 education.html
create mode 100755 images/background.jpg
create mode 100755 images/bart.jpg
create mode 100755 images/facebook.png
create mode 100755 images/google+.png
create mode 100755 images/instagram.png
create mode 100755 images/linkedin.png
create mode 100755 images/twitter.png
create mode 100755 index.html
create mode 100755 main.css
create mode 100755 portfolio.html
Barts-Mac:portfolio bartdn183$

```

5. git remote add origin
<https://github.com/bartdn183/bartdn183.github.io.git>



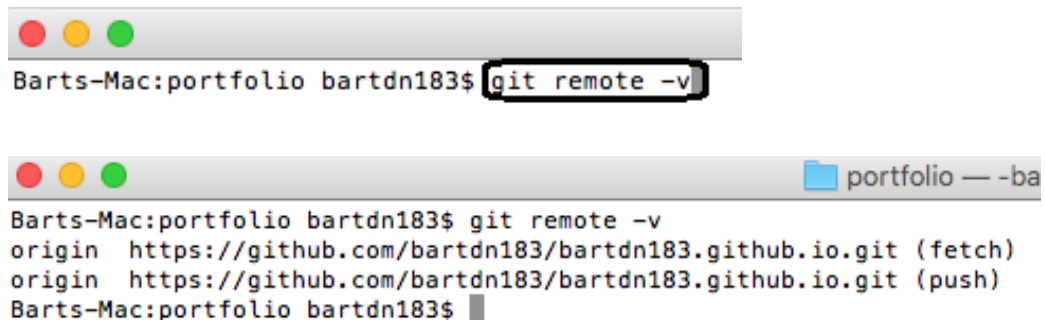
```

Barts-Mac:portfolio bartdn183$ git remote add origin https://github.com/bartdn183/bartdn183.github.io.git

```

Change this twice to your own username!

6. Check remote status => git remote -v



```

Barts-Mac:portfolio bartdn183$ git remote -v

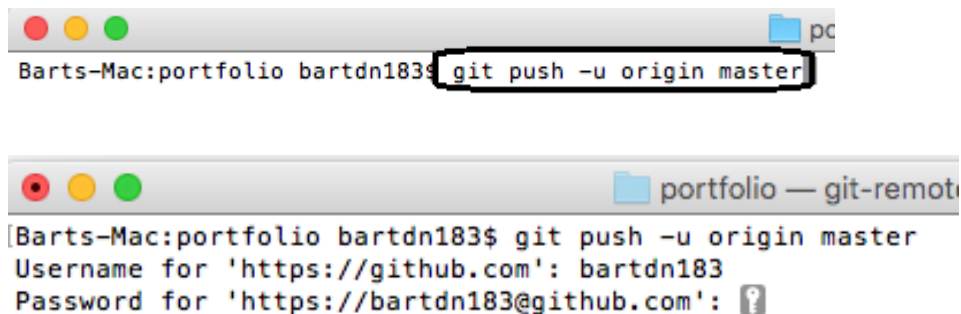
```

```

Barts-Mac:portfolio bartdn183$ git remote -v
origin https://github.com/bartdn183/bartdn183.github.io.git (fetch)
origin https://github.com/bartdn183/bartdn183.github.io.git (push)
Barts-Mac:portfolio bartdn183$

```

7. Now we will push our changes to github to the master branch.



```

Barts-Mac:portfolio bartdn183$ git push -u origin master

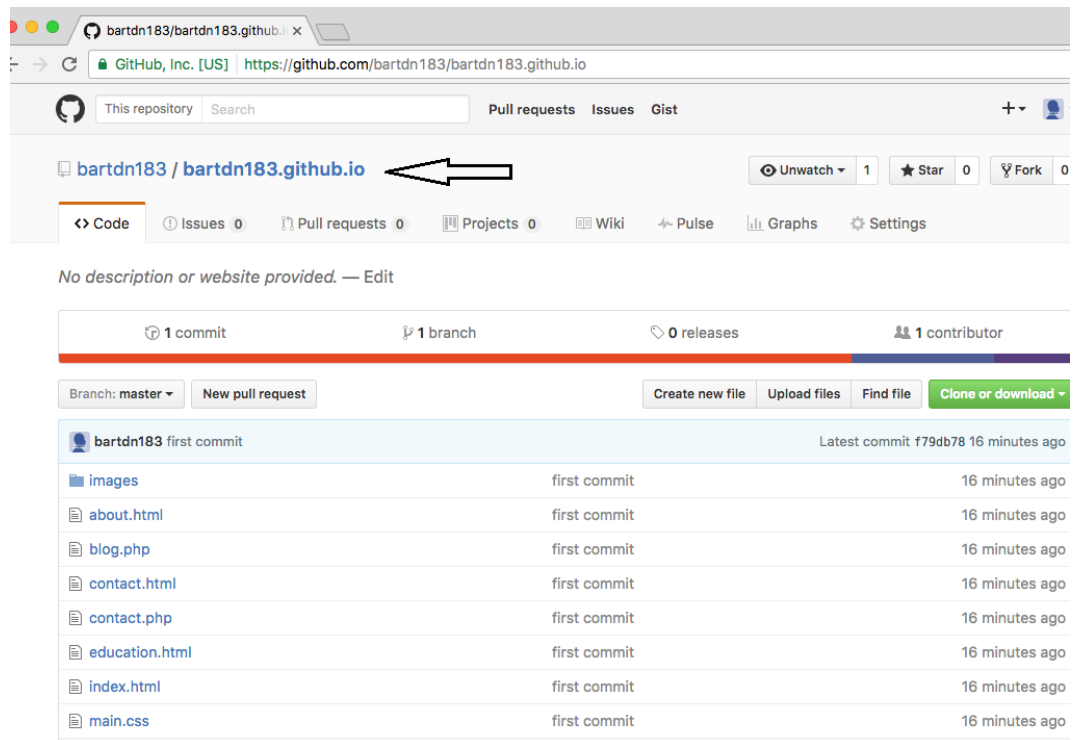
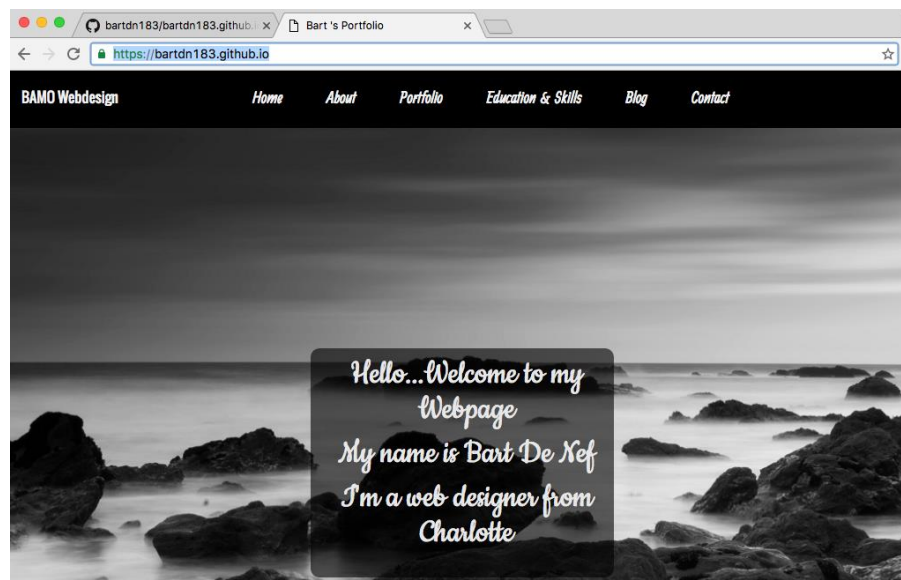
```

```

Barts-Mac:portfolio bartdn183$ git push -u origin master
Username for 'https://github.com': bartdn183
Password for 'https://bartdn183@github.com': 

```

8. Click on your repository name.

9. Check your website! => <http://bartdn183.github.io>⇒ <http://yourusername.github.io>

⇒ This works only for static webpages!

⇒ 1 free personal website per Github account!