# 1. What is Git

* Git is a Distribution Version Control System. It’s system that is going to record changes to our files over time.
* We can recall any specific versions of those files at any time.
* Many people can easilly collaborate on a project and have their own version of project files on their computer.

### Why Use Git

* Store revision in project history in just one directory.
* Rewind to any revision in the project I wanted.
* Work on new features without messing up the main codebase.

Easilly collaborate with other programmers.

### What is GitHub

* Online service that hosts our projects.
* Share our code with other developers.
* Developers can download the projects and work them.
* They can re-upload their edits and merge them with the main codebase.

# 2. Installing Git on Your System

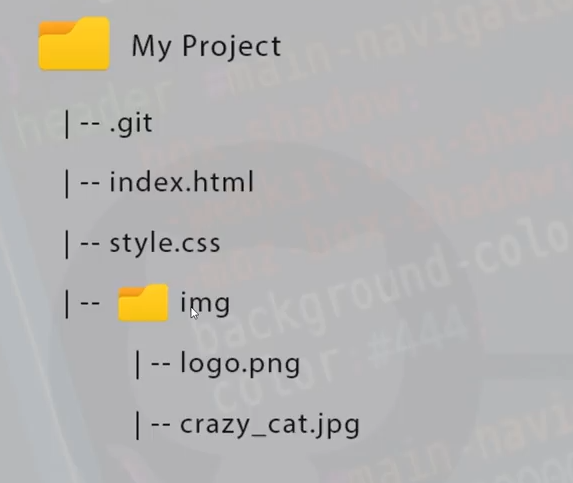
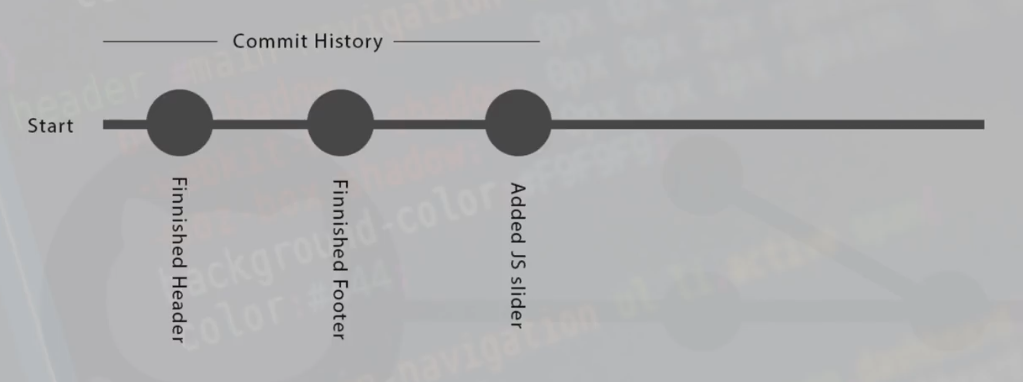
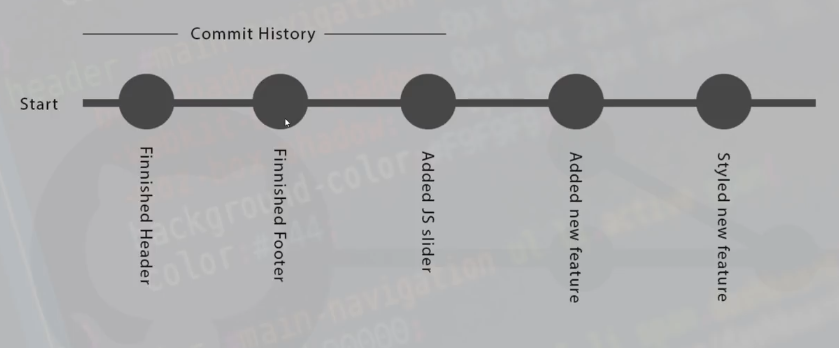
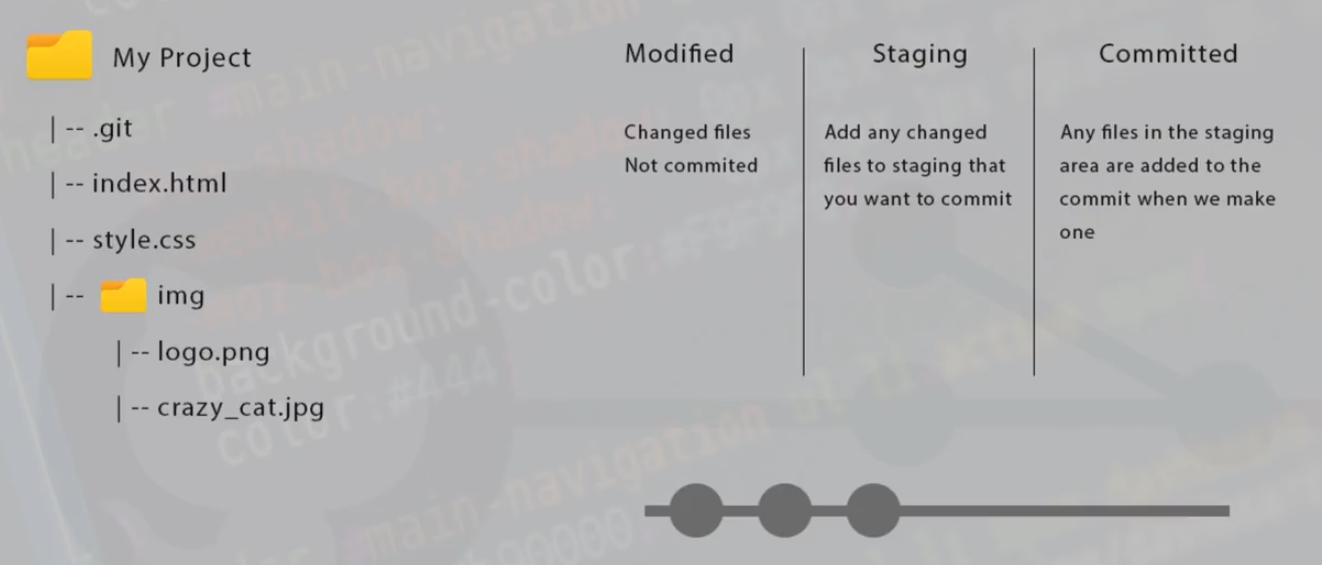
* [Link To install Git](https://git-scm.com/downloads)
* Download command line Interface if you’re using Windows.
  + [Click to download cmder](https://cmder.net/)
* To check if you have git install, just type:
* $ git --version
* Now, we need to configure Git by adding our username, email, and password if you like.
  + [Linke to configure Git](https://www.theodinproject.com/courses/web-development-101/lessons/setting-up-git)

*Important*: As developer, you need to know some base Command Line to easy browse the directory.

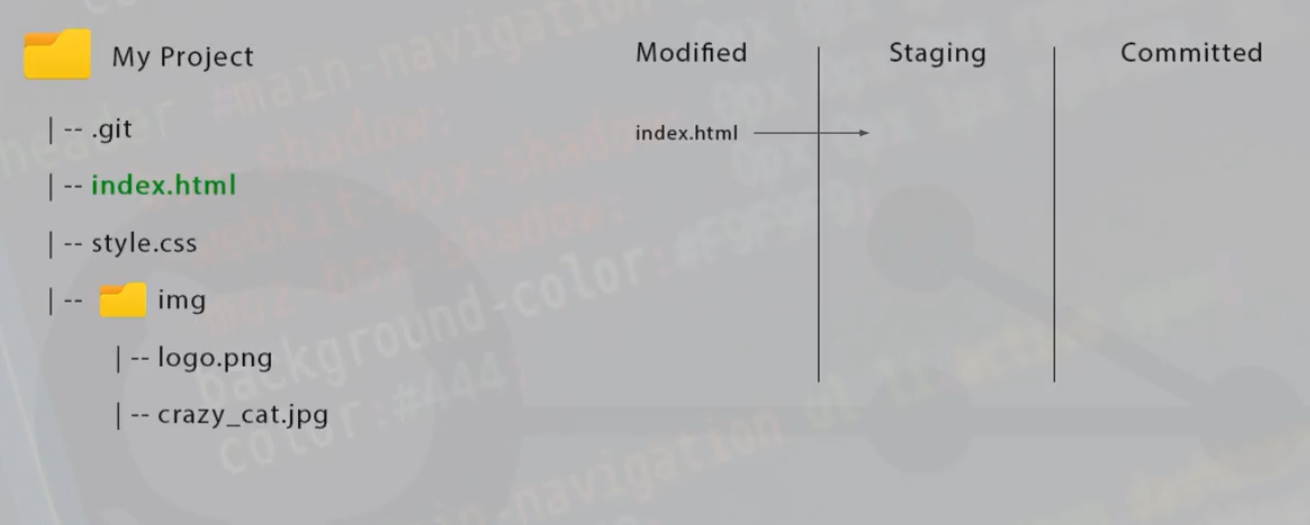
**Example:**

* *touch, mkdir, nano, rm, rmdir, rm -rf, ls, cd, cd .., cp*

# 3. How Git Works

* At the hard of git there are these called *Repositories* or *Repo’s*.
* A repo is a container for a project you want to tract with Git (Eg. A Website Project).
* Can have many different repo’s for many different projects on your computer.
* Like a project folder which Git tracks the contents of for us.
* 
* Commit *(It’s like saved point)*
* 
* 
* There are three steps we need to know while commit.
  + Working Area
  + Staging Area
  + Repo
  + 

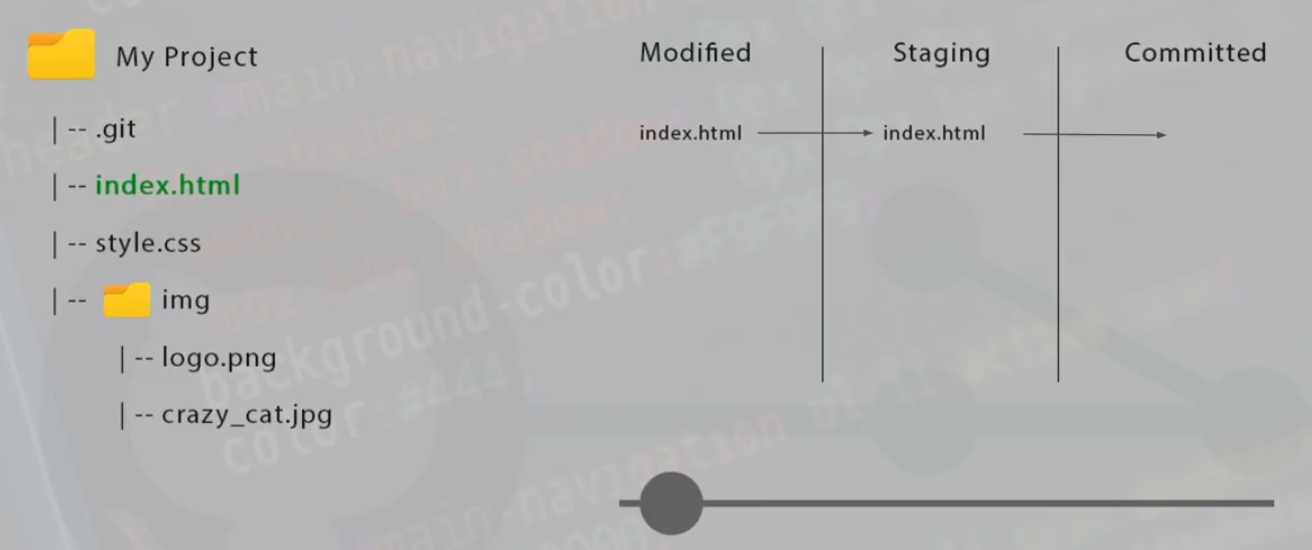
# 4. Git and GitHub

* Creating a Repository
  + We need navigate inside the project and type:
* $ git init
* After creating the repo, create an index.html file inside your project folder.
* 

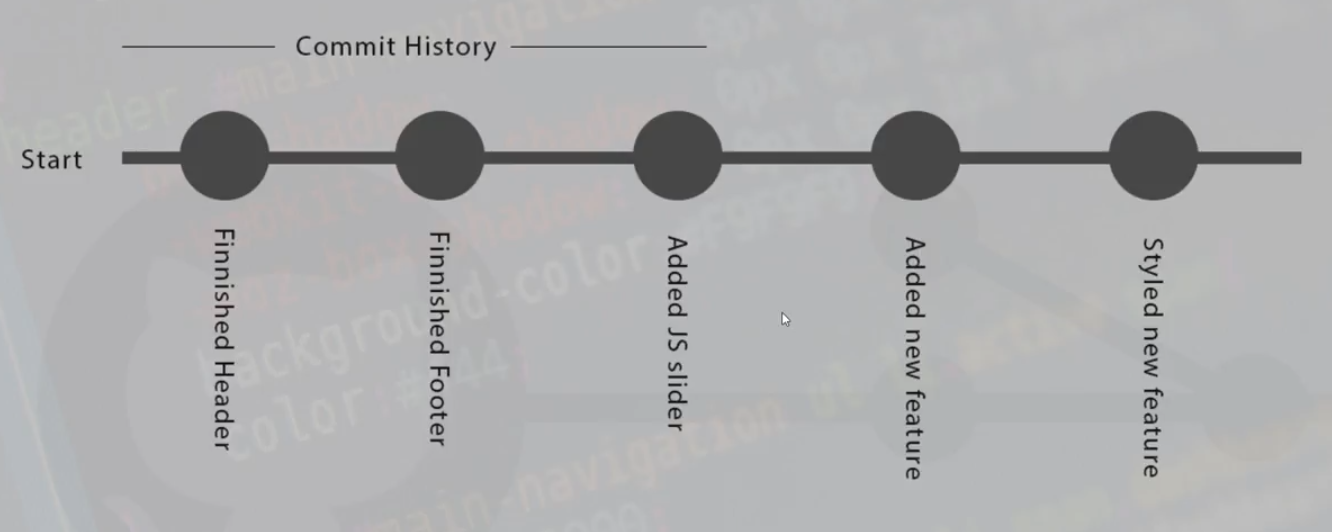
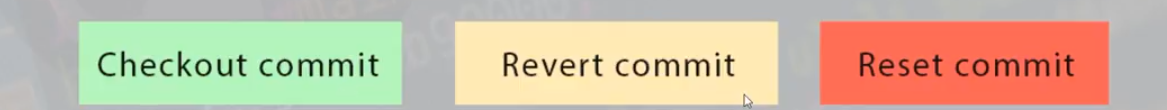
# 5. Staging Files

* Now, we modify our index.htmlfile but, it is not commit yet, to get it commit it needs to be placed in the staging area.
* See all files that are in the staging are, we use this command:
* $ git status
* If the color is red, it means that our file is untracked and not in the staging area and also is not ready to get commit.
* If the color of the file is green that means that the file is in the staging area and ready to be commit.
* To add a file or files in the staging area, we use this command:
* $ git add [the name of the file]
* $ git add index.html
* We could check the status: git status.
* If we make a mistake by add the file in the staging area, we can rollback, and put it in the working area.
* $ git rm --cache [name of the file]
* $ git rm --cache index.html
* What is we would like to add many files in the staging area?
  + We use this command: *git add .*.
* Create another file named it style.css to make the practice.
* Then, add both files index.html and style.css in the staging area.
* $ git add .
* We can check the status: git status, and you will both are in the staging area and ready to be commit.

# 6. Making Commits

* Remember that commit is saving our project in a particular point.
* 
* How do we make commit? to commit we use this:
* $ git commit -m "Added index and style files"
  + The (-m) message should be logical, clear and descriptive.
* If we want to see the commit history, we use:
* $ git log
* If we want to see them condense, or simplify we could use:
* $ git log --oneline

# 7. Undoing Things

* If everything goes wrong, what to do…
* 
* To undo (Rewinding) thing, there are three methods that we can use.
* 
  + Green -> Safe
  + Yellow -> Less safe
  + Red -> Not safe
* Let’s add two more commits to pratice.
* To see information about a commit, we need to git log --oneline and then type `git checkout [id of the commit]

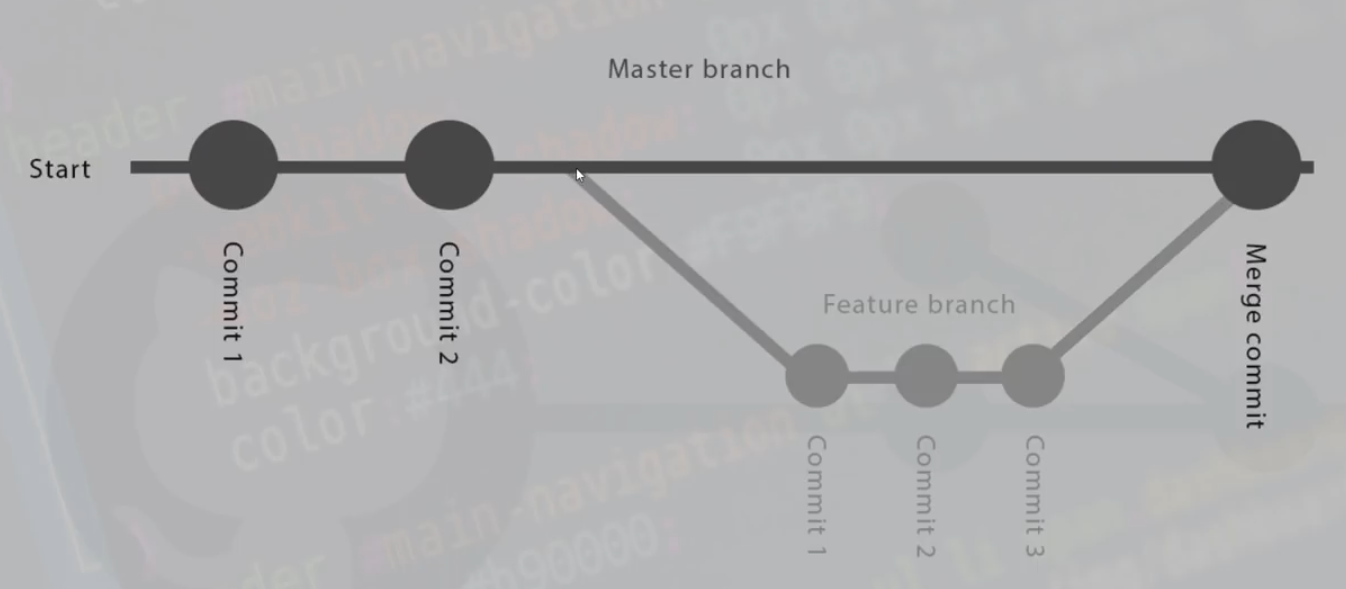
1. First

* $ git log --oneline

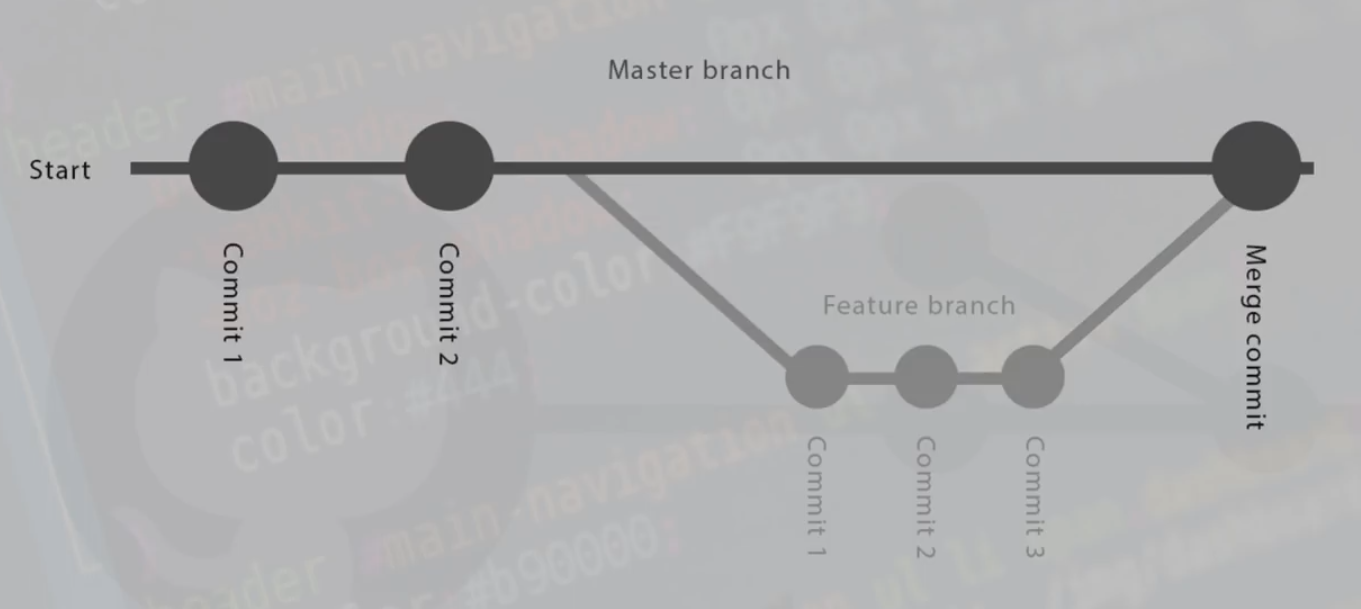
1. Second shell $ git checkout [id]
2. What we just did is detached from the master branch. The way to get back is:

* $ git checkout master
* To revert, we do:
* $ git revert [id of the commit]
* To reset, we do:
* $ git reset [id of the commit]
  + It reset the commit to but the files still stay the same.
* To reset and remove the file:
* $ git reset --hard [id of the commit]
  + It removes everyting thing we did in the files.

# 8. Branches

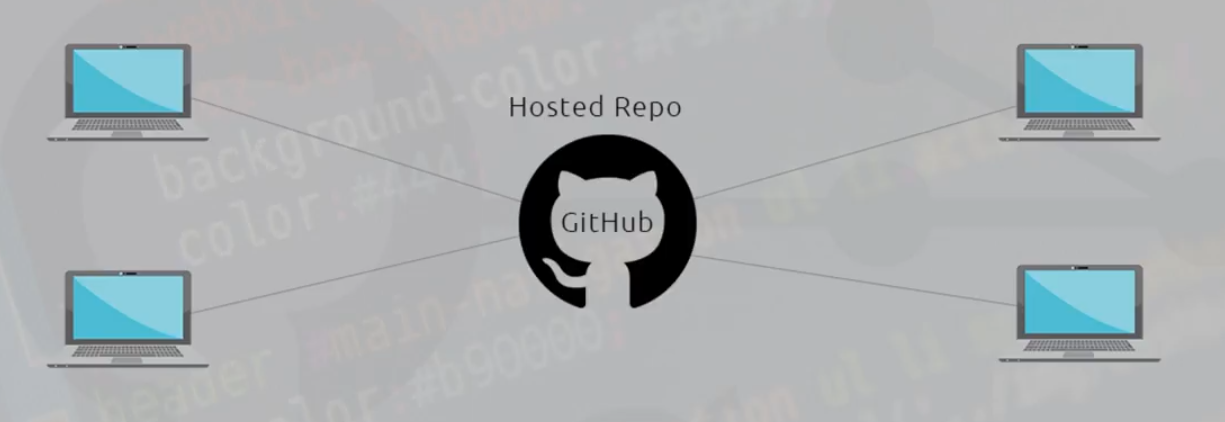
* Our First commit was to master branch.
* 
* It is the that we are goint to publish, so we not really want to add new features on that branch. So, we need other branches.
* 
* How to create a new branch? We use: git branch [name of the branch].
* $ git branch feature-1
* If we want to see all the branched, we can do:
* $ git branch -a
  + This symbol ’\*’ indicates that we are on the branch that follows it.
* We created the branch, but how to use it? we do this: git checkout [name of the branch]
* $ git checkout feature-1
  + We can check now, and you will see the change.
  + Now, we can start working on that new feature.
* If we would like to check the master branch, we could do:
* $ git checkout master
  + You will notice that the modification we did in the feature-1 branch is not part of the master branch.
* If we want to delete a branch that we created, we need to do: git branch -d.
* $ git branch -d feature-1
* An error will appear since we have not merge the branch yet.
* We can force to delete the branch by using a capital (-D) instead a small (-d).
* $ git branch -d feature-1
  + Now, the branch is deleted.
  + Check git branch -a.
* We can create a branch and switch to it at the same time. It is quicker. we do: git checkout -b [name of the branch].
* $ git checkout -b feature-1

# 9. Merging Branches and Conflics



* To merge, we have to be in the branch we want to merge into, we need to be in master branch. So, we do this: git merge [name of the branch to merge].
* $ git merge feature-1
* The branches are merged now.
* Sometimes, we might come across what we call as conflicts.
* Conflict happens when we use different branches and then edit a group or a line of code in the same project.
* So, how to fix that?
  + We need to talk to the guy who does this to now editing the master branch.
  + We could also do:
    - git add .
    - git commit (Without the message)
    - If a scary screen appears, just shift in colon ":" and type "wq" and type Enter to exit the screen.
    - So, our merge is fixed and done.
* If we type git log --oneline, we will see the merge appears.

# 10. Introduction to GitHub

* GitHub is a service that lets us set up hosted repositories.
* Central online repository which multiple team-members could access.
* 
* Now, we assuming everyone has a GitHub account. Let’s creat a new remote repo to push our local repo. Let’s go to our GitHub account.
* We can create repo two ways:
  + Locally as we just did.
  + Remotely

1. If we use SSH:

* echo "# test" >> README.md  
  git init  
  git add README.md  
  git commit -m "first commit"  
  git remote add origin git@github.com:djken/test.git  
  git push -u origin master

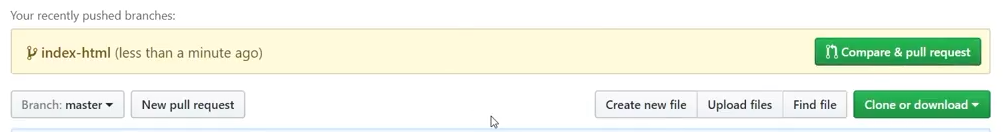
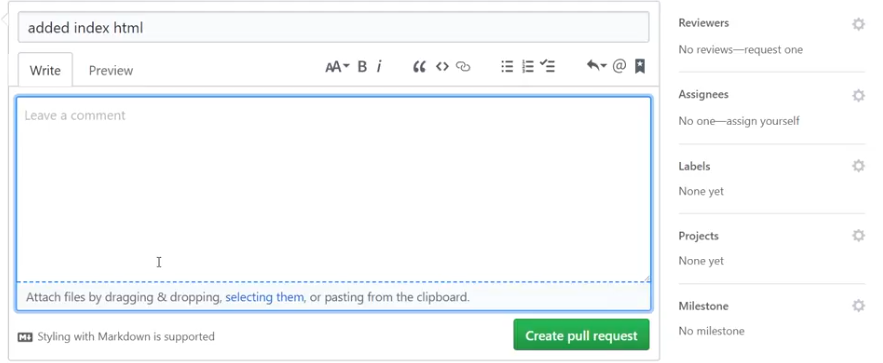
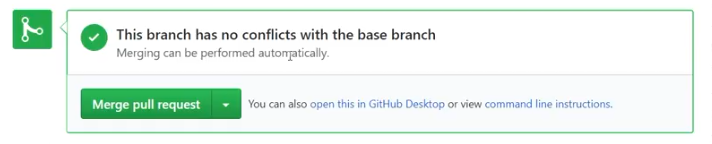
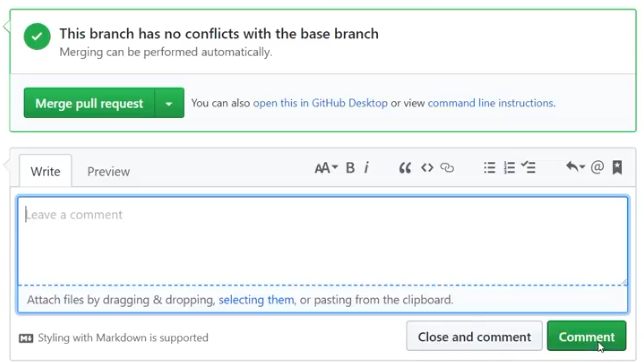
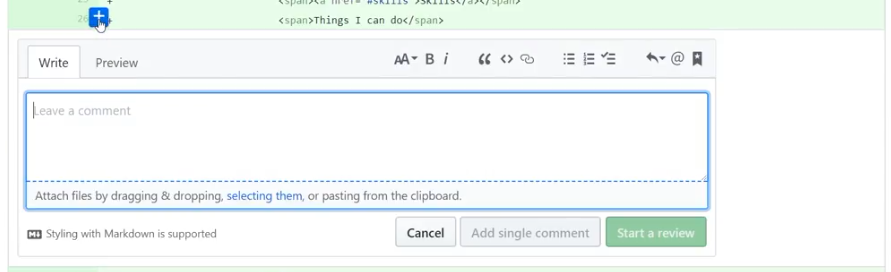
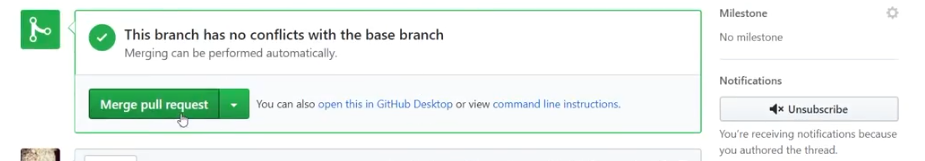
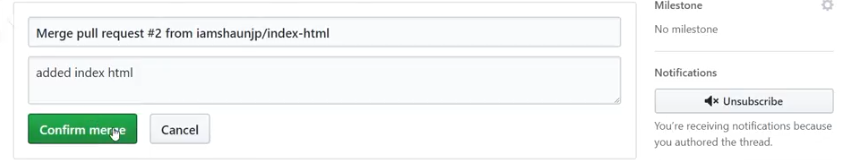
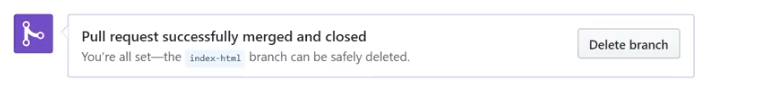
1. If we use HTTPS:

* echo "# test" >> README.md  
  git init  
  git add README.md  
  git commit -m "first commit"  
  git remote add origin https://github.com/djken/test.git  
  git push -u origin master
* How to clone a remote repo? we do: git clone [link of the repo].
* $ git clone git@github.com:djken/test.git
* To check if the origin is setup, we do:
* $ git remote -v
* We should see somethung like:
* origin git@github.com:djken/laravel-notes.git (fetch)  
  origin git@github.com:djken/laravel-notes.git (push)
  + push to send the code to the online repo.
  + fetch to bring down the code the local repo sometimes.

# 11. Collaborating on GitHub

* Make a pull to keep our file up-to-date is any developers have made some changes.
* Make sure that we are in master branch before making the pull. We do:
* $ git pull origin master
* Git pull fetches all the code in the master from the remote and then merges them into our master branch on our local computer.
  + When editing, always make sure that we are not on the master branch. Why………? Answer from you…..
* Steps when pushing to a different branch (not the master).
  1. Let’s create a new branch so we can demonstrate how to collaborate no GitHub.
  2. after, write our code. Add, commit.
  3. Then, we are going to push to code our remote repository, but we are going to push it to the master branch. If we do, other developers and your boss can get mad and fire you. Your colleagues or your manager need to review that code and decide if they should merge it.

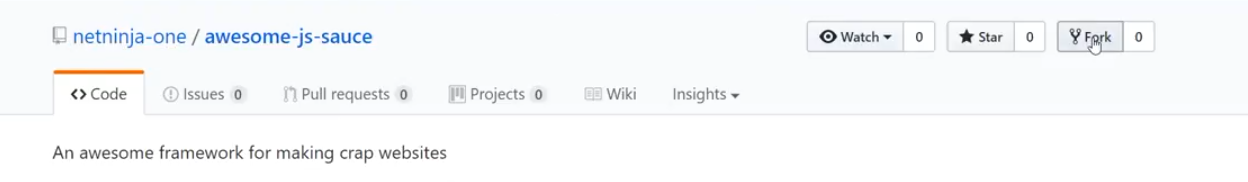
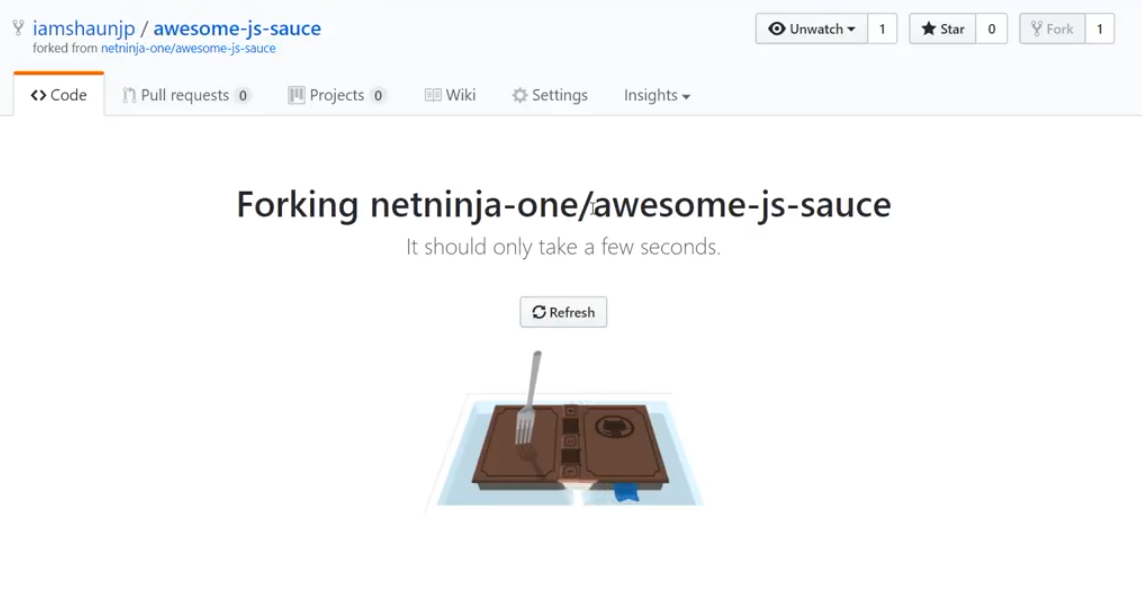
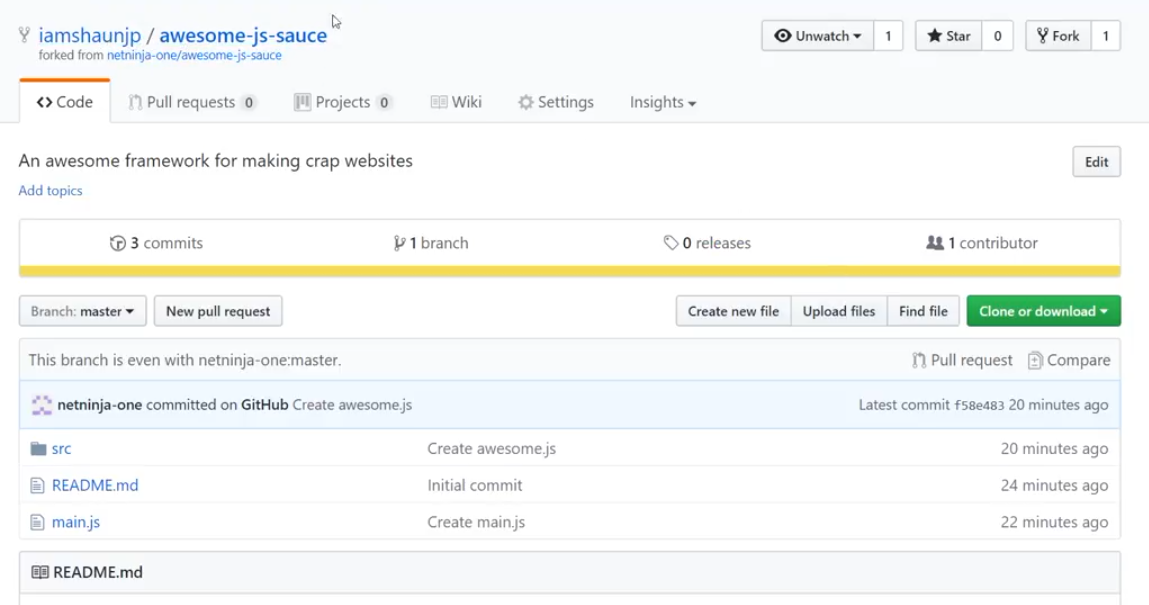
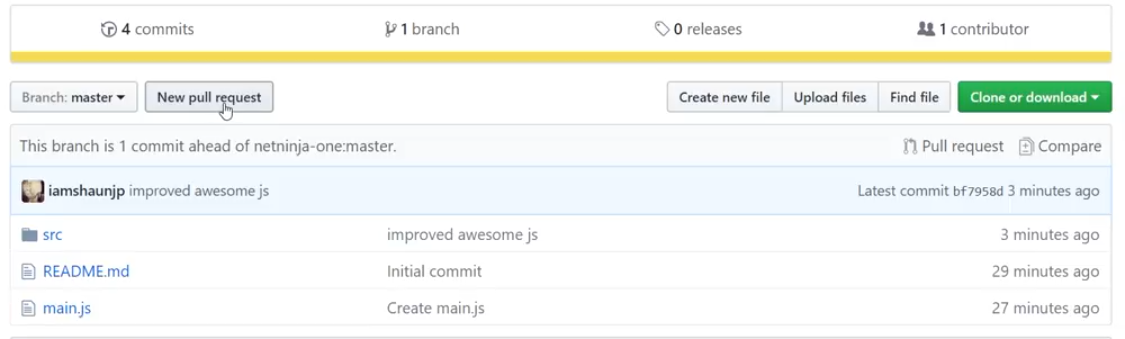
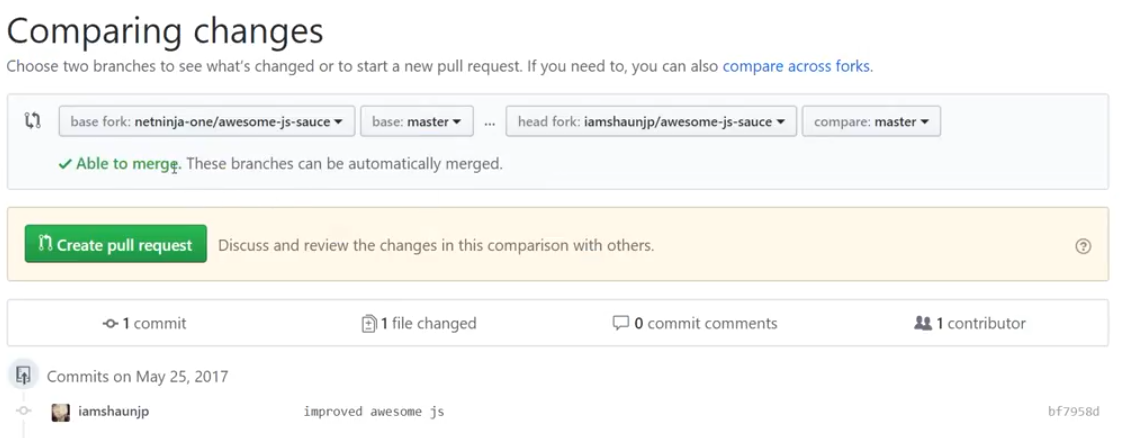
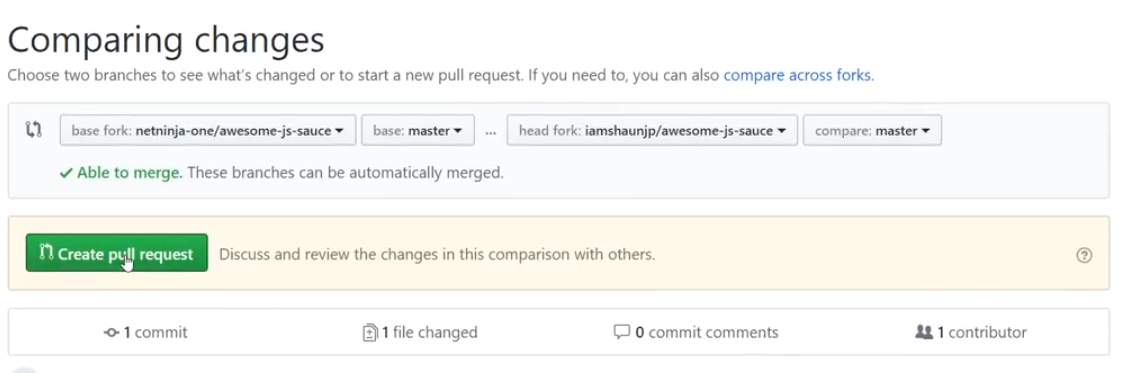
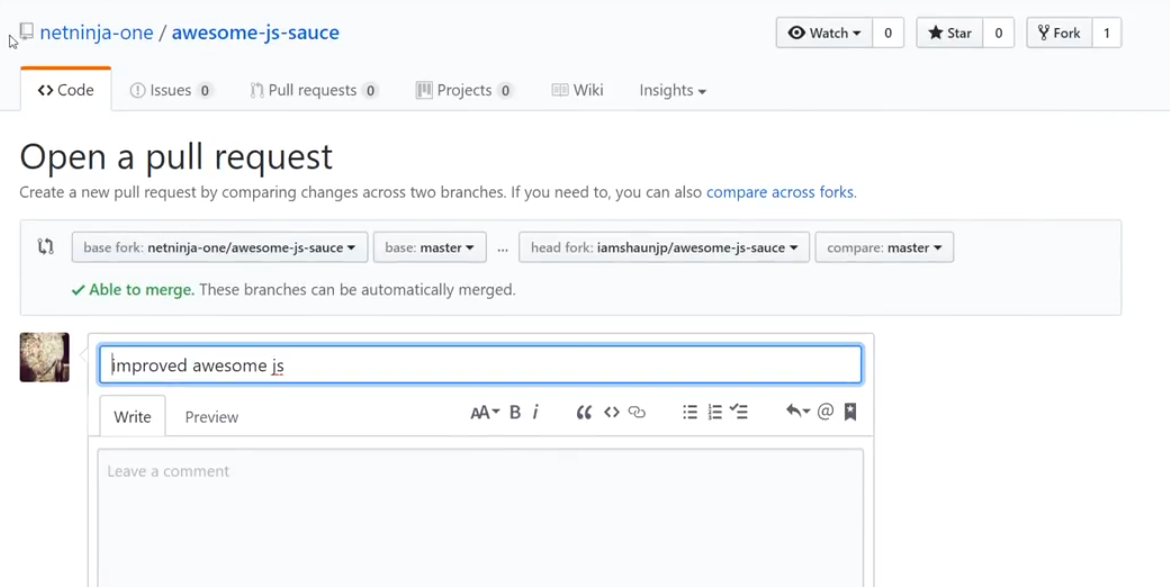
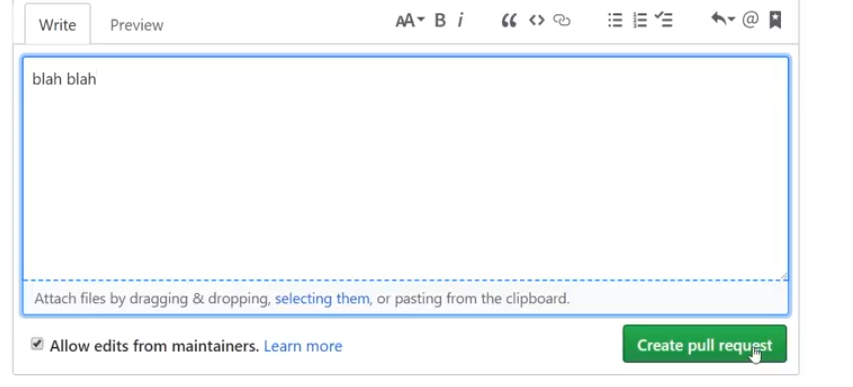
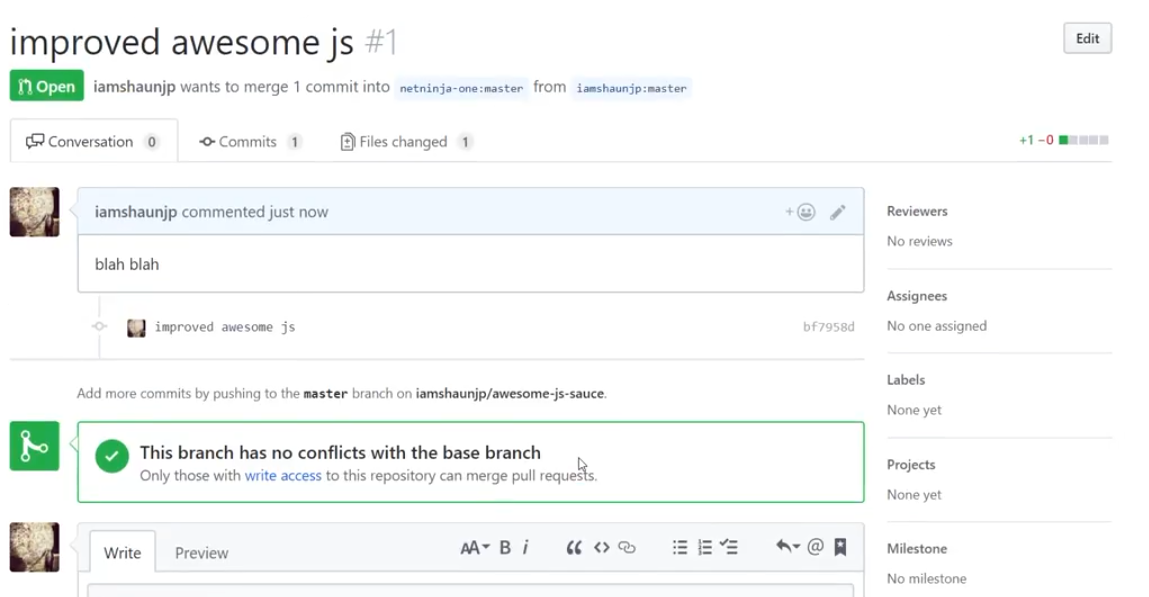
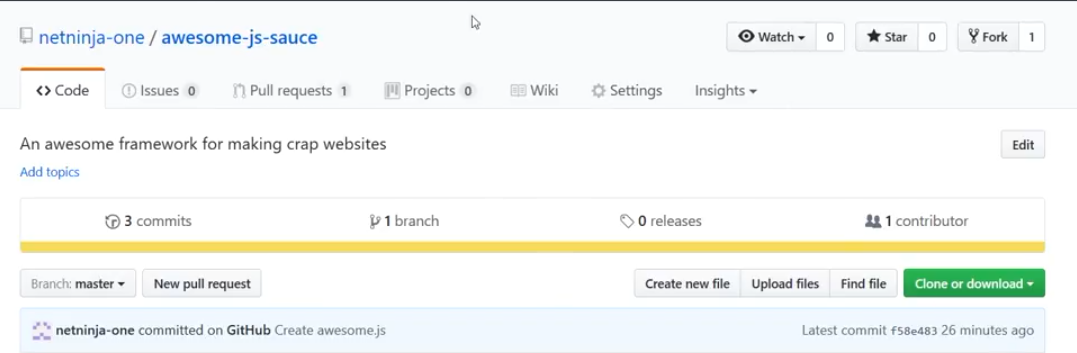
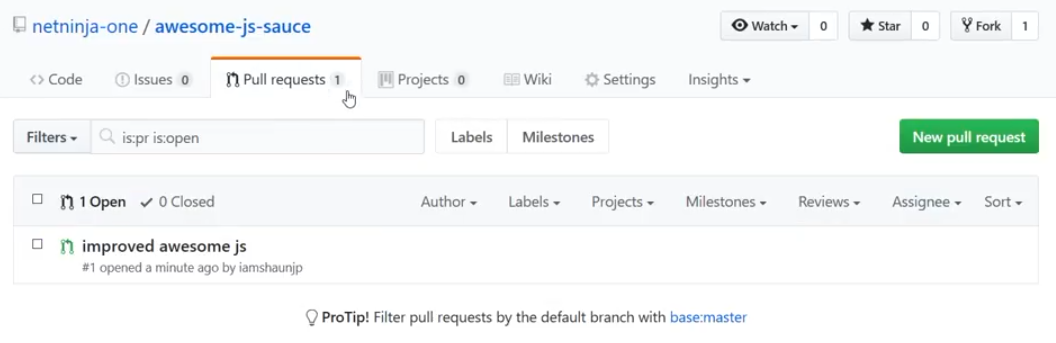
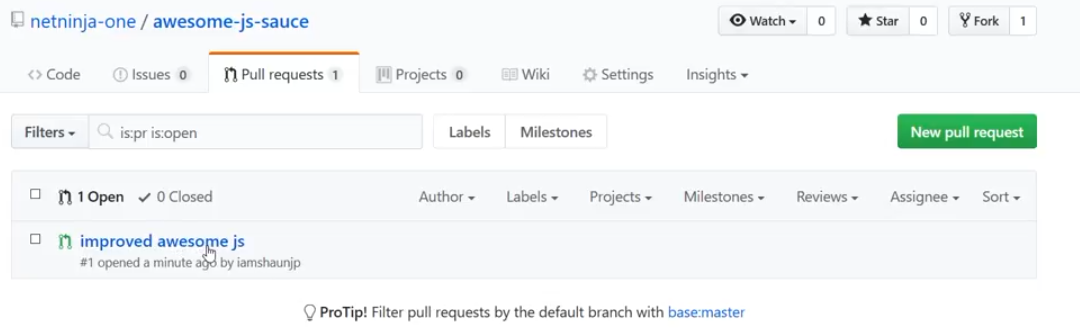
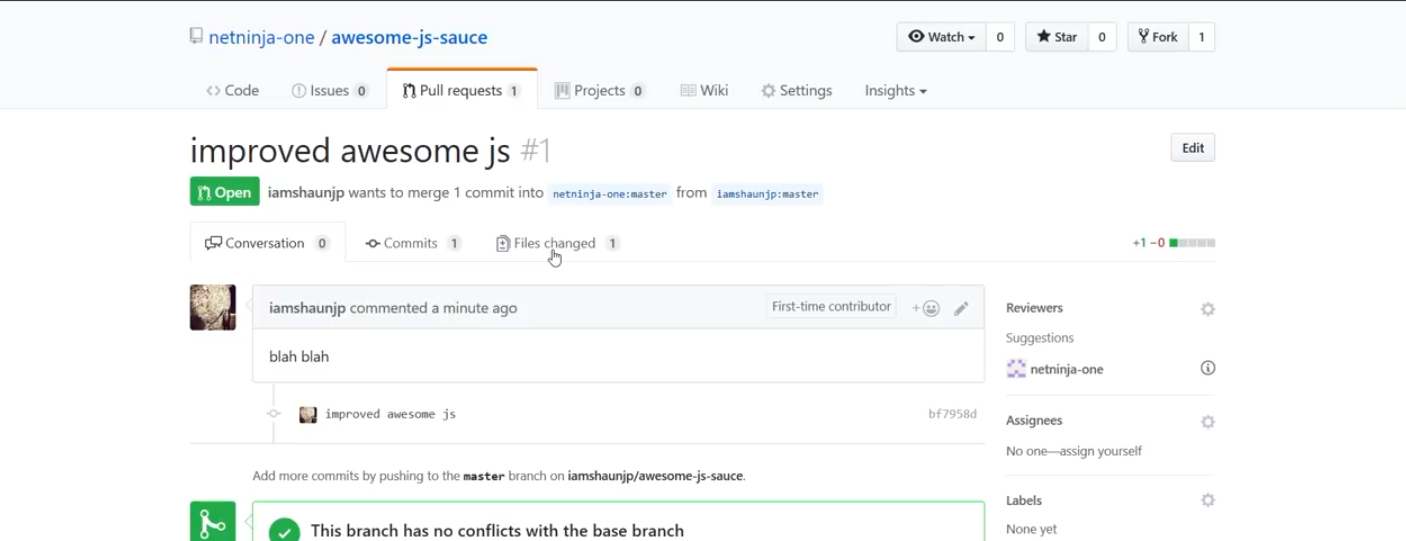
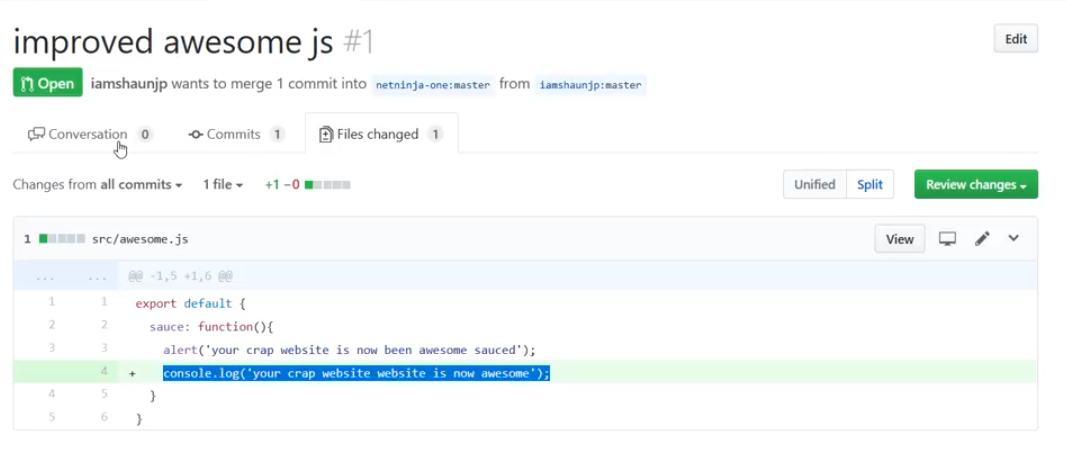
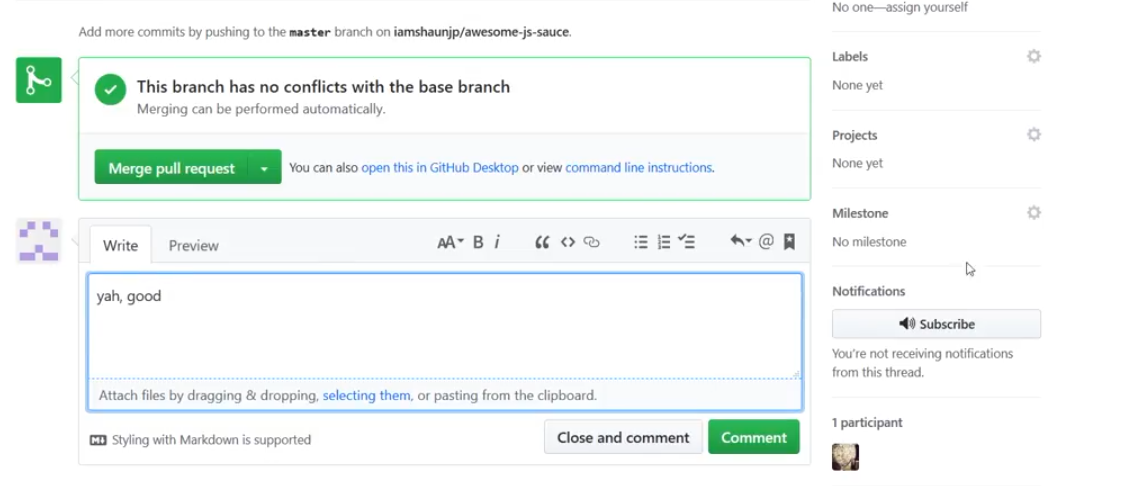
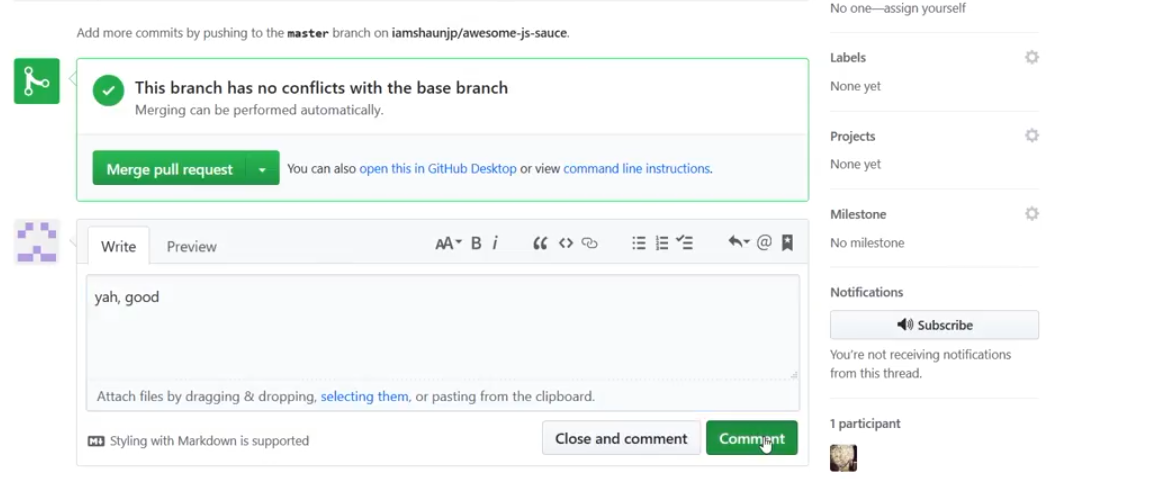
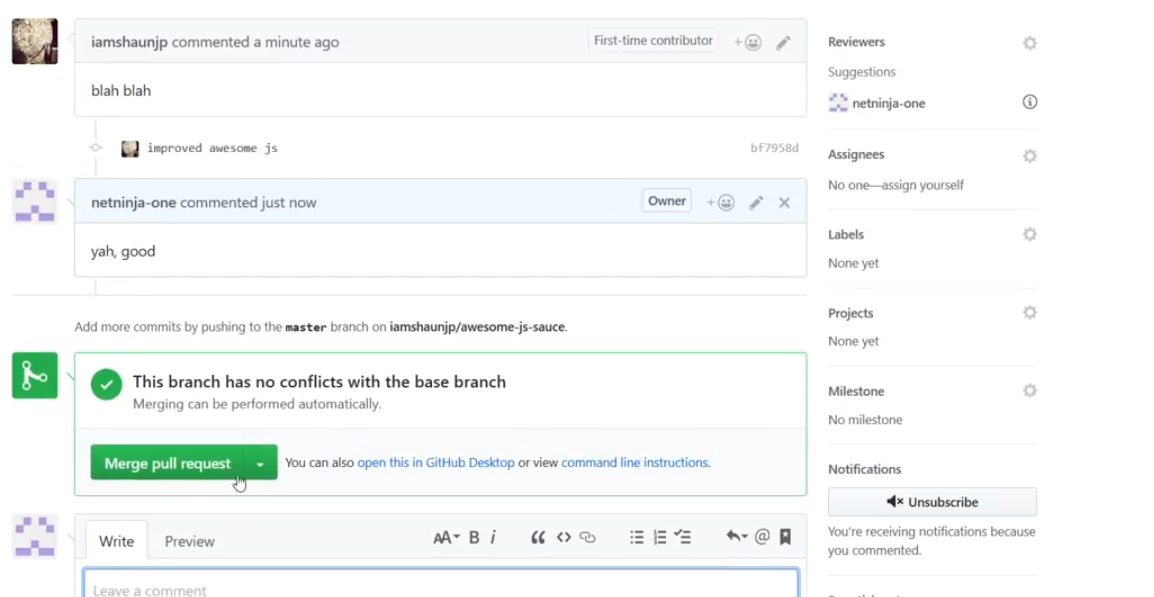
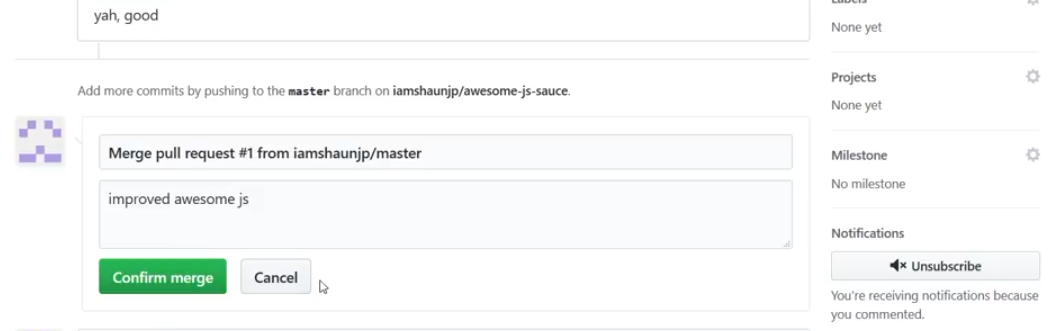
1. To push to the branch we are working, we do: git push origin [name of the branch]

* $ git push origin feature-a
* Now we can check our GitHub repository to the change and the branch pushed.
  1. Click on the green buttom Compare & pull request. that means we are going to compare and see the change made.
  + 
  1. If we agree with the code, we will make a create request and merge it to the master branch.
     + We will see the message we added to the commit.
     + We can leave a commit for the other developers or the manager.
  2. Create a pull request.
  + 
  1. When we do this, all the developers will recieve ping or a notification so they can review your code before merging them.
  2. Then, come down and read the message at the top of the green button to see if there are any conflicts.
  + 
  1. If you want to write a comment, you can do that below the green button.
  + 
  1. We could up to see the commits made and the file change. see the picture below:
  + 
    - Green lines codes that we added.
    - Red lines are code that we deleted.
  1. WE can also write a comment line by line by clicking on small cross on the left.
  + 
  1. If we agree with all the code and commits, we can go back and merge the branch.
  + Merge:
  + 
  + Confirm:
  + 
  1. Mergig take this branch and merge it to the master branch.
* Now, if we want we can delete the branch as well.
* 

Impartant

* Before we edit our code, we need to make sure:
  + The code in the master is not modify.
  + git checkout master
  + Make a pull request on yout local directory
    - git pull origin master
  + Always create a new branch to write new code, don’t edit the master branch code directly.

# 12. Forking & Contributing

* To contribute to an open source projects.
* How does it work?
  + We need to fork the repository we need to contribute. When we fork the repo, it is going to make a copy from the forked account to your own account.
  + Process 1
  + 
  + Process 2
  + 
  + Result (Copy made to my account)
  + 
* Now, we can clone it into your local dictionary.
* $ git clone [link of the clone repo]
* Navigato into the folder and open it into a text editor.
* Now, we can:
  + Add
  + Commit
  + Push
* All what we do is in my account. Now, we need to send our contribution to the initial repository.
* We are going see this button, called New pull request.
* 
* The result will be:
* 
* Now, we need to click on the Create pull request.
* 
* It’s going to pull it into the master branch of the initial project.
* 
* now, we can comment and go down to create the pull request.
* 
* Once I create this pull request, I cannot merge it because I won’t have access to do it.
* 
* The originial creator will have access to review the changes and then merge the code we write if it is ok.
* If the original create goes to his or her account, he/she will see the number of Pull request.
* Process 1:
* 
* Process 2:
* 
* If the original creator clicks on the request:
* 
* He will see that someone want to contribute to the project.
* We can click on Files changed to see code the contributor has written.
* 
* Result of the file changed.
* 
* If the creator is happy with the code, we can click on Conversation to go back and merge the code.
* 
* We can write comment:
* 
* 
* Now, the original create can merge the contributor’s code into the main branch.
* Merge:
* 
* Confirm merge:
* 
* Now, if the creator goes to the original repo, he will be able to see the code that is just update by the contributors.