GIT

- Free and Open Source Version Control System.
- Programmers use it on a daily basis.
- Version control is a way for programmers to track code changes.
- We can see all our previous versions of the code and go back to a previous version too or track bugs.
- Most programs are interacted with using the command line. We will do the same with GIT instead of using programs like Github Desktop.
- Git is the tool that tracks changes to your code overtime. Github is a website to host all our repositories.
- Being online makes it easy to work with groups and other people and also acts as a portfolio to show your work to employers.

Important Terms – cd (Change Directory), Repository (Project, or the folder/place where your project is kept)

Git Commands -

Clone -> Bring a repository that is hosted somewhere like Github into a folder on your local machine

add -> Track your files and changes in Git

commit -> Save your files in Git

push -> Upload Git commits to a remote repo, like Github

pull -> Download changes from remote repo to your local machine, the opposite of push

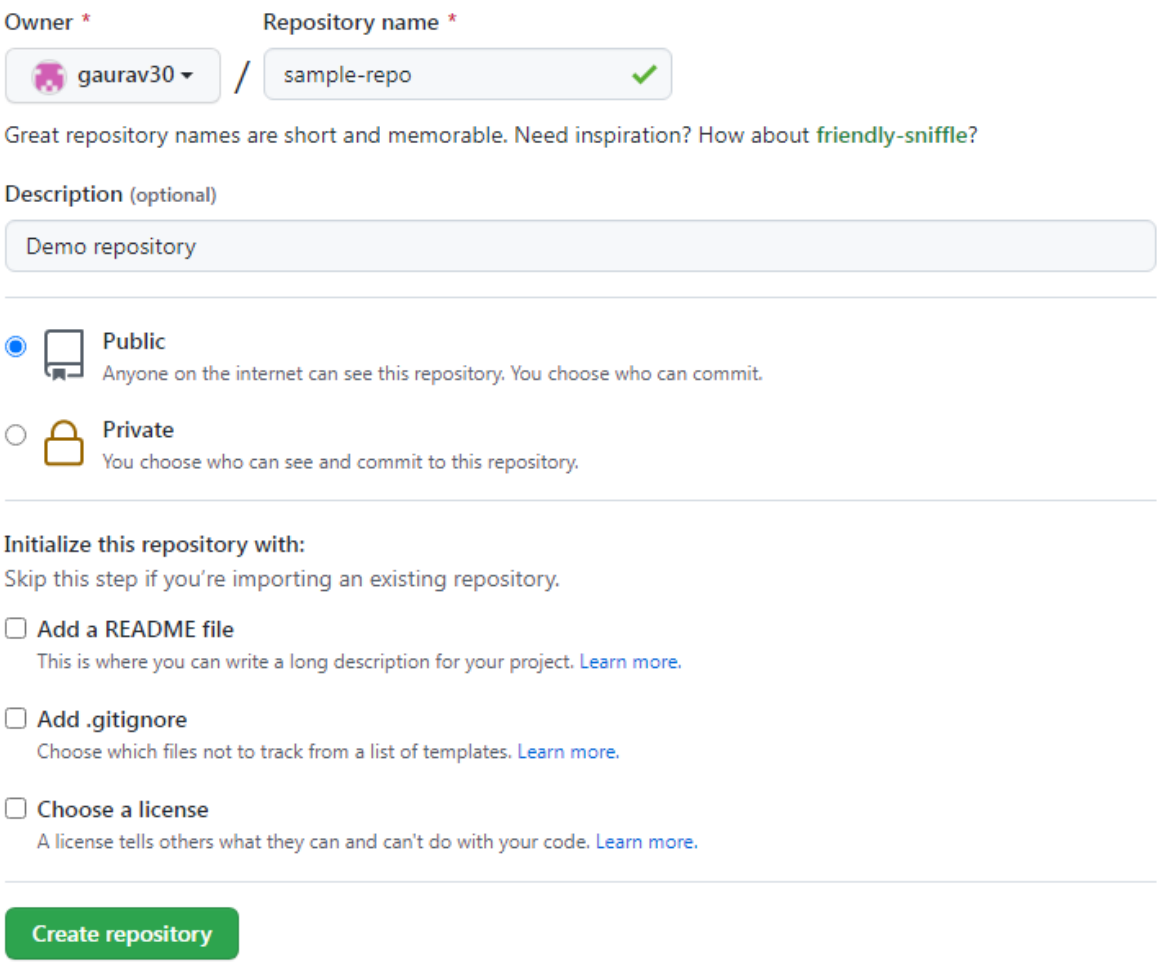
Implementation -

Sign up on Github.

After login, The profile page and repositories can be accessed from the top right dropdown.

A repository holds the project files.

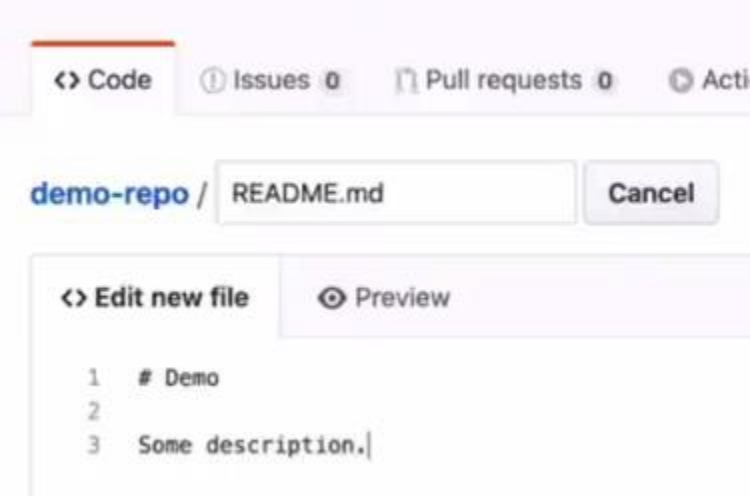
Clicking on the green New button lets us create a new repository.



Now we can create the files and folders for this repository on our local machine or on the online editor of Github.

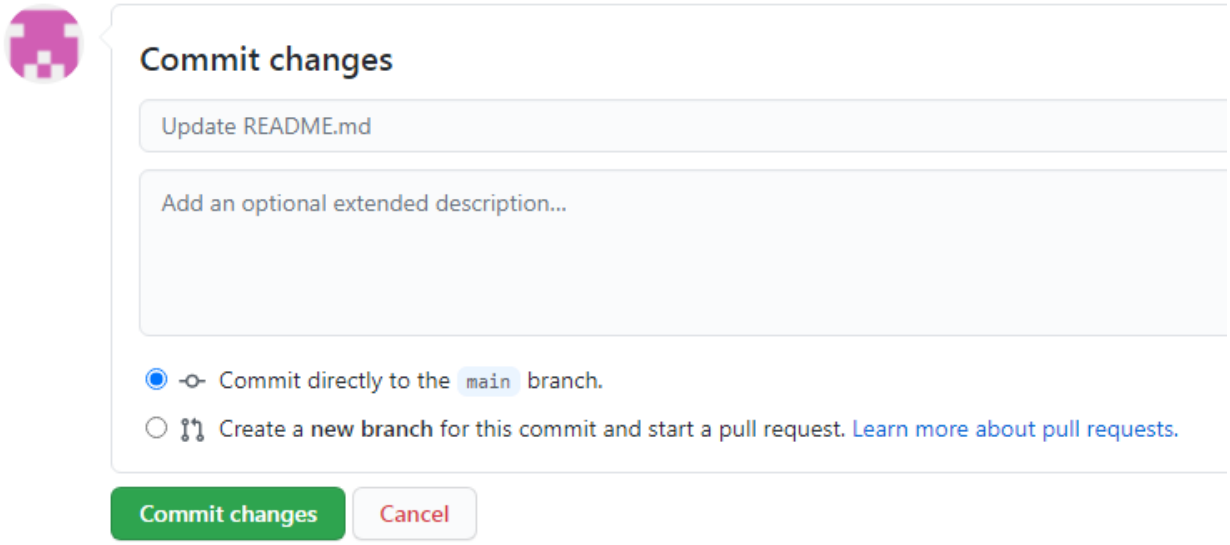
We can create a basic markdown file called Readme.md. Mostly, This is the file in every project that contains text to describe what the project is about.

Click on creating a new file.



Scroll to the bottom and click on commit new file. The default commit message is “Create README.md”. We can also provide a commit message of our own.

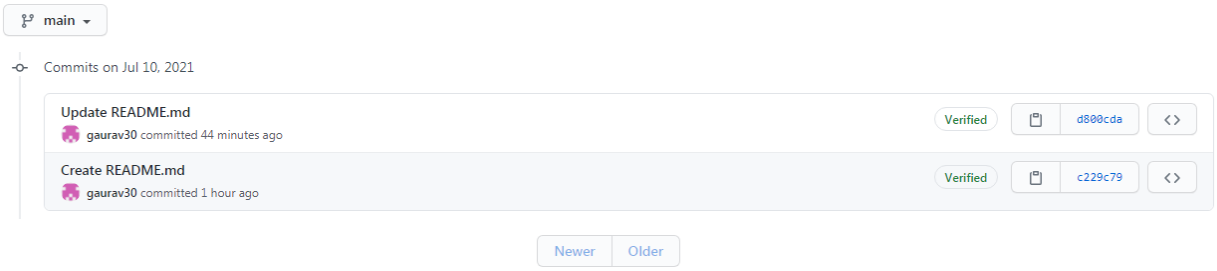
Now Click on the file name and edit it. Make some changes to the file’s contents.



Scroll to the bottom and see the default commit message is Update README.md. Commit the changes.

Click on the repo name. And we can see the latest commit message as Update README.md.

If we click on the file name and click the history button on the right, we can see all the commits to the file.

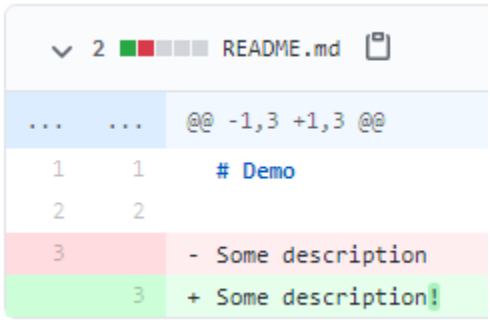


Each commit has a unique ID and we can see all the commit messages too.

If we click on the first commit, we can see a plus sign to the left of each line that indicates that these lines were added.

0 comments on commit `c229c79`

In the Update Commit, we could see the lines that changed.



Anything white means it stayed the same.

Download Git for Desktop - https://git-scm.com/downloads

On our local machine, we can see if Git is installed by typing **git –version** on cmd.

Git should be used through Git Bash.

Create a new folder and open it using VS code. Now we will pull the repo that we created on Github to our local system.

In VS code go to View -> Terminal. Type command :

git clone <your repository URL>

Now we will have a folder in VS code of the repo that we pulled down from Github. Use cd command to move inside the folder.

There is a hidden folder inside every repository folder on your system called as .git. It is only visible if the hidden files are shown on the system.

This hidden git folder actually stores all of the files that save your commits or your code changes overtime. It has all of the changes recorded in the history of this repository which includes the ones we made on github.com.

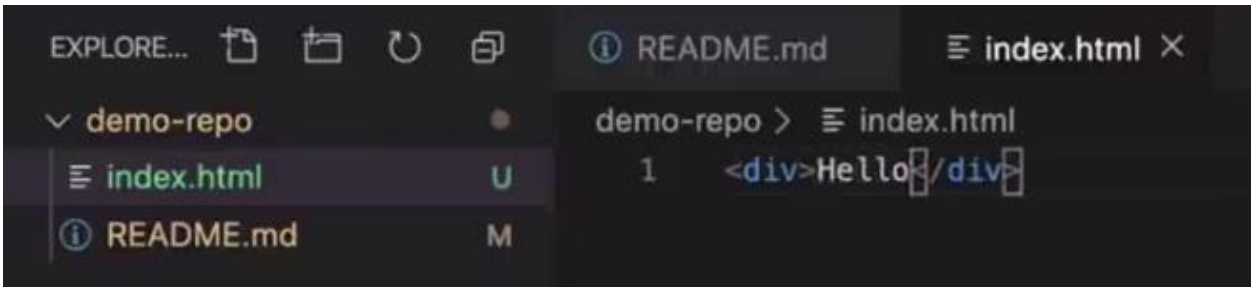
Now edit the README.md file contents on VS code .

Now we need to save the changes in git.

Type command : **git status**

This command shows us all of the files that were updated, created or deleted but have not been saved in a commit yet.

Suppose we create a new file index.html ,



Now run the **git status** command again. We can see that one file is modified and one file is untracked that means git does not know about the file yet.

So we need to tell Git to track the file before we can save it to Git.

To do that we use command : **git add <file name>** or **git add .**

The . tells Git to track all of the files that are listed in Git status. This includes changes to the README file and the new index.html file. We can also write the name of a particular file like **git add index.html**

Now write **git status** again and we see that all the changes have been tracked.

Now we need to commit the changes,

```
→  demo-repo git:(master) ✗ git commit -m "Added index.html" -m "some des
cription"
[master 3a1d48e] Added index.html
 2 files changed, 5 insertions(+)
 create mode 100644 index.html
```

-m is for the commit message title and the second –m is for the commit message description. The title is to be given but description is optional. The message should ideally state what and why is being committed.

But we have still only saved our code locally. It is not yet live on Github.

We need to use the push command to update on Github. In order to push them to Github under your account, we have to prove to Github that we are the owner of that account. We need to connect our Local Machine to our Github account.

This is done by generating a SSH key locally using the below command.

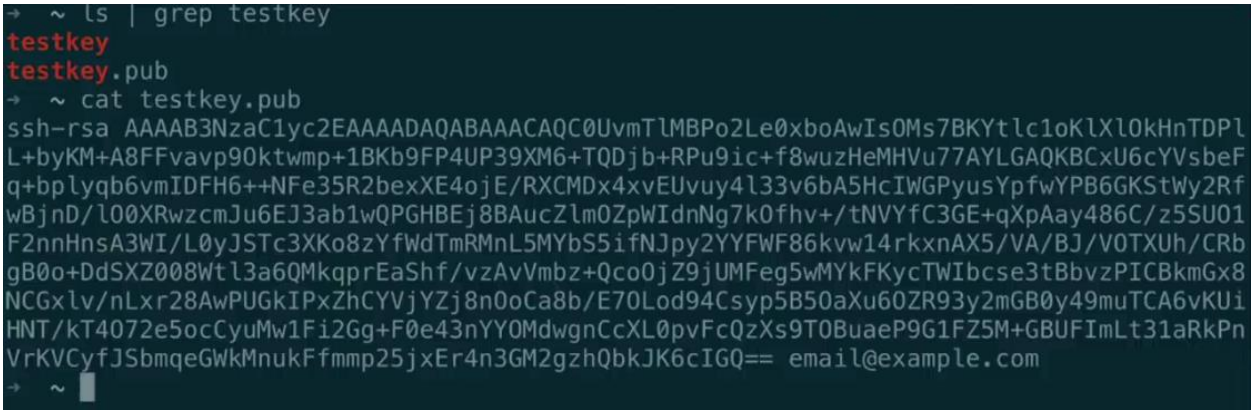
```
→  ~ ssh-keygen -t rsa -b 4096 -C "email@example.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/gwen/.ssh/id_rsa):
```

-t is for the type of encryption. –b is for the strength of encryption. And also the Github email address.

The file name will be id_rsa by default. We give the name testkey to it.

```
→  ~ ssh-keygen -t rsa -b 4096 -C "email@example.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/gwen/.ssh/id_rsa): testkey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in testkey.
Your public key has been saved in testkey.pub.
The key fingerprint is:
SHA256:gHmbQn8S/hAhl2GMFzAap1moz2HYzD0wf6LkisCW9J0 email@example.com
The key's randomart image is:
+---[RSA 4096]----+
|   ..*+*+        |
|  +B.B+.         |
| *+=+.=          |
|o.B.=+.*         |
|o*ooo+O S        |
|.+=. E =         |
|+.       .       |
|o                |
|                 |
+----[SHA256]-----+
→  ~
```

Now we search for the testkey,



The public key is uploaded to Github and we have to keep the private key secure on our system and whenever we need to access our account we use the private key.

Copy the key.

On Github for the top right dropdown, go to Settings -> SSH and GPG keys -> New SSH Key. Enter the title and the key.

Now we need to make sure our local GIT command line know about the key we just generated.

https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

https://docs.github.com/en/get-started/getting-started-with-git/managing-remote-repositories#switching-remote-urls-from-https-to-ssh

Type command

```
→  demo-repo git:(master) git push origin master
```

Origin refers to our repository. Master is the branch. Now if we refresh our page on Github we can see that all the changes are live.

There are 3 commits now. Two made on Github and one made locally and pushed to Github.

Start a repo locally – Create a new folder in VS Code. This is not a git repo yet. Move to this folder in the terminal.

Create a README.md file inside this folder.

If we want to turn this folder into a git repository, we can write the command **git init**

Type **git status** to see the untracked README file.

Type **git add README.md**. Type **git status**.

```
→  demo-repo2 git:(master) ✗ git commit -m "Created README" -m "descripti
on"
```

To push this live we type **git push origin master**. We get a error because git does not know where to push it to because it is not connected to anything.

The easiest way is to create an empty Git repository on Github.

Name the repository demo-repo2 and run command :

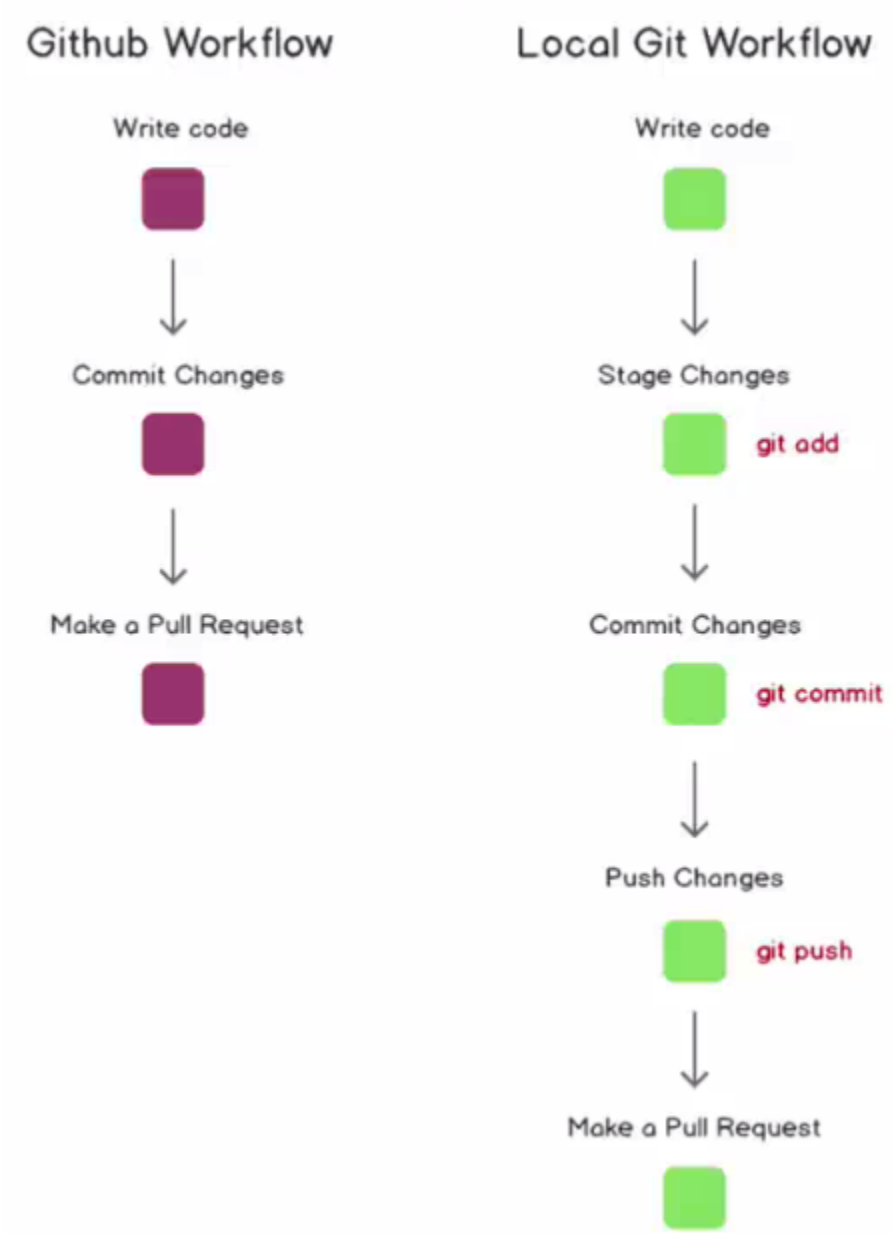
```
→  demo-repo2 git:(master) git remote add origin git@github.com:gwenf/dem
o-repo2.git
```

Type **git remote –v** to verify.

Now we type **git push origin master** to push to the repo. For future we don't want to specify the repo everytime so we can just use the command **git push** but we need to set an upstream to it.

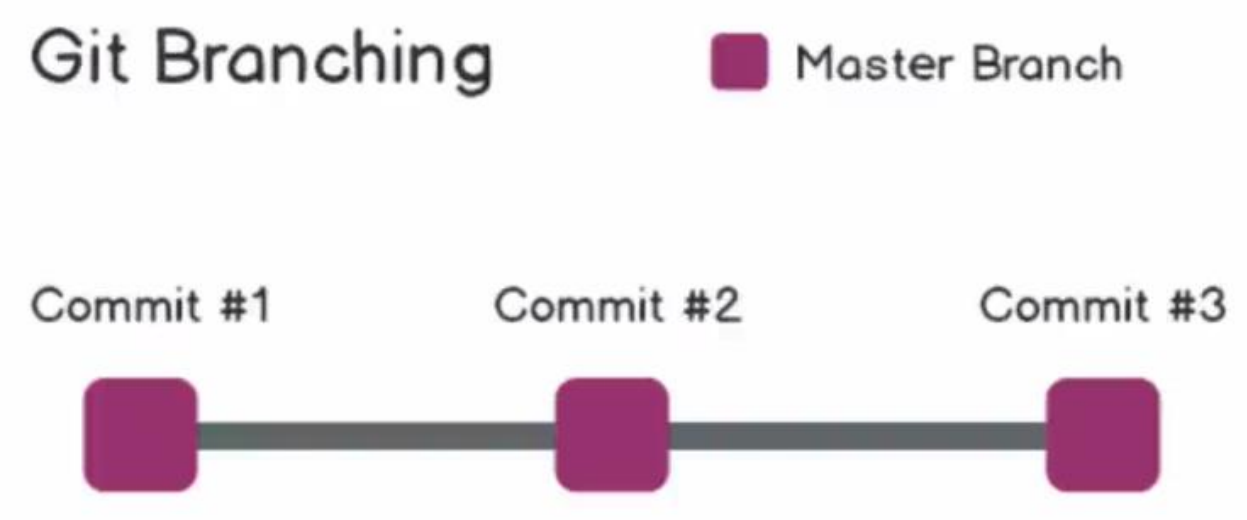
```
→  demo-repo2 git:(master) git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 243 bytes | 243.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:gwenf/demo-repo2.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Refresh the page on Github and we can see the file that we just added.

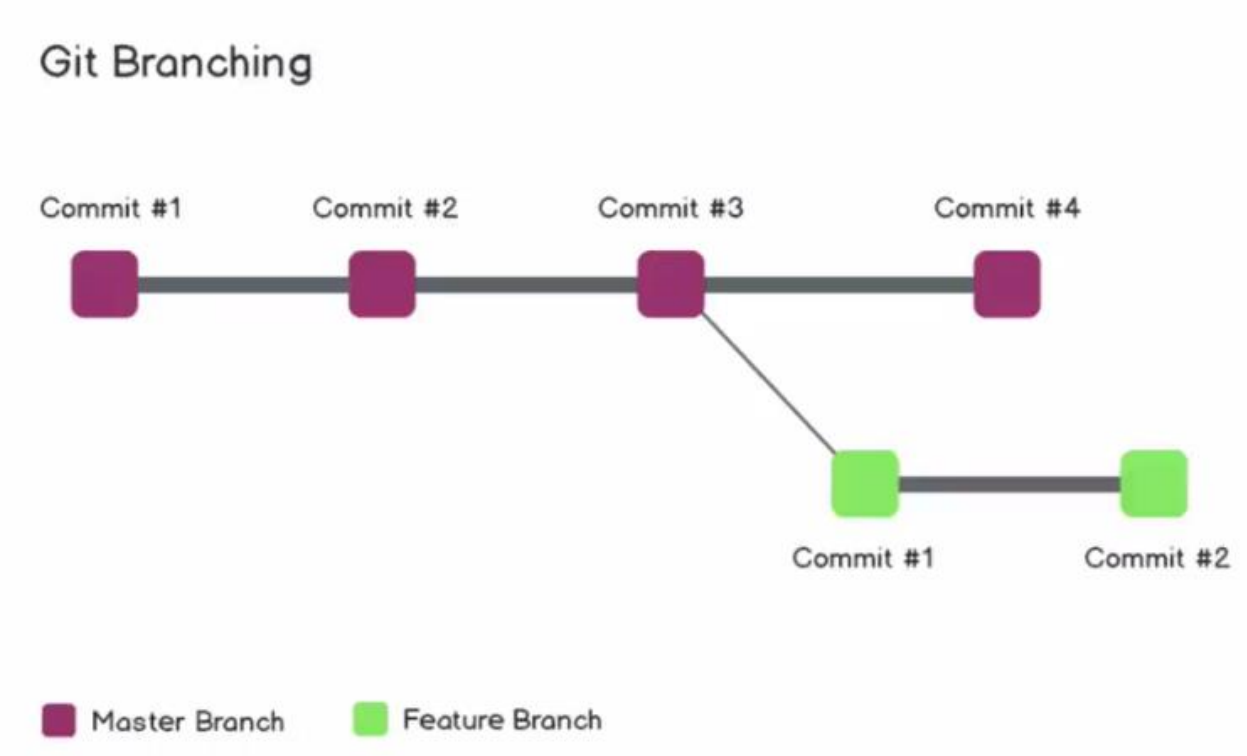


A pull request is made to edit someone's code or get access rights to a repo.

Git Branching – If we are working in a branch, all our code and all our commits will be in that one branch. Master is the main branch.

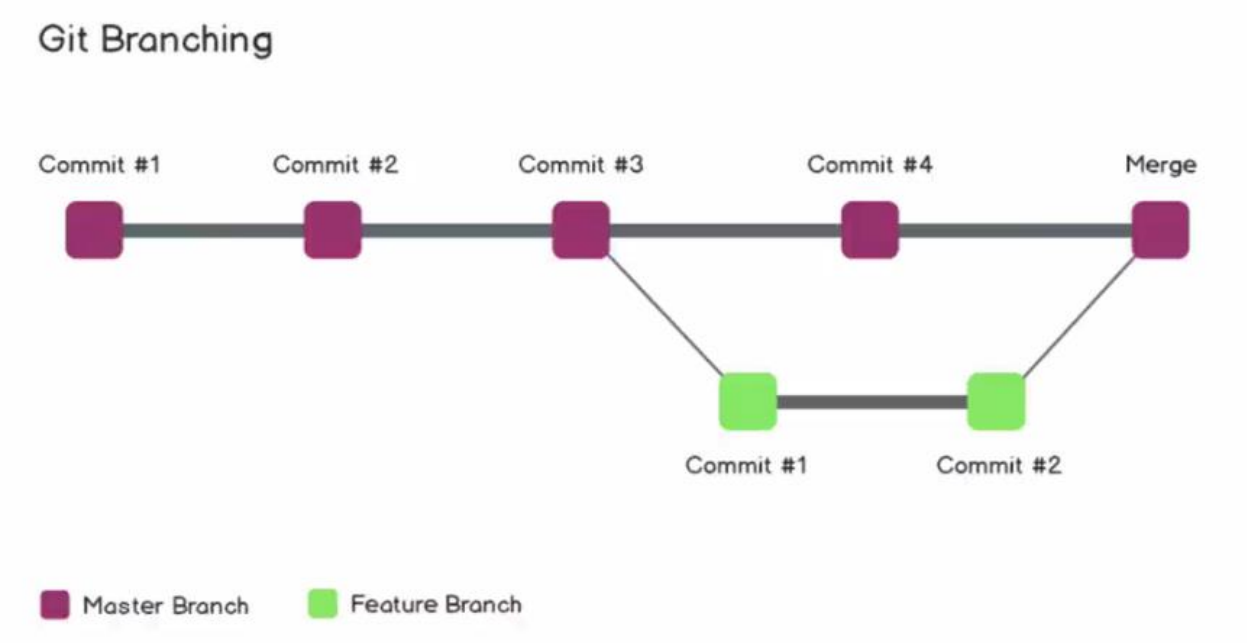


We could also have another branch. We will call it the Feature Branch.



At first the code on Master and Feature branch will be exactly the same. But as we make updates to the Feature branch, those changes will be only seen in the Feature branch. When we switch to the Master branch, we would not be able to see the commits in the Feature branch. A branch cannot know the changes and commits in other branches.

This is useful because if we are adding features to our app that may break our code and not finished yet and we do not want to work in the main branch until they are fully ready before merging in the main branch.



This is also helpful when many people work in the same repository.

Navigate to demo-repo folder in VS Code.

```
→  demo-repo git:(master) git branch
```

The * indicates that we are currently on that branch. Type Q to get back.

Git checkout is used to switch between branches. The below command will create a new branch.

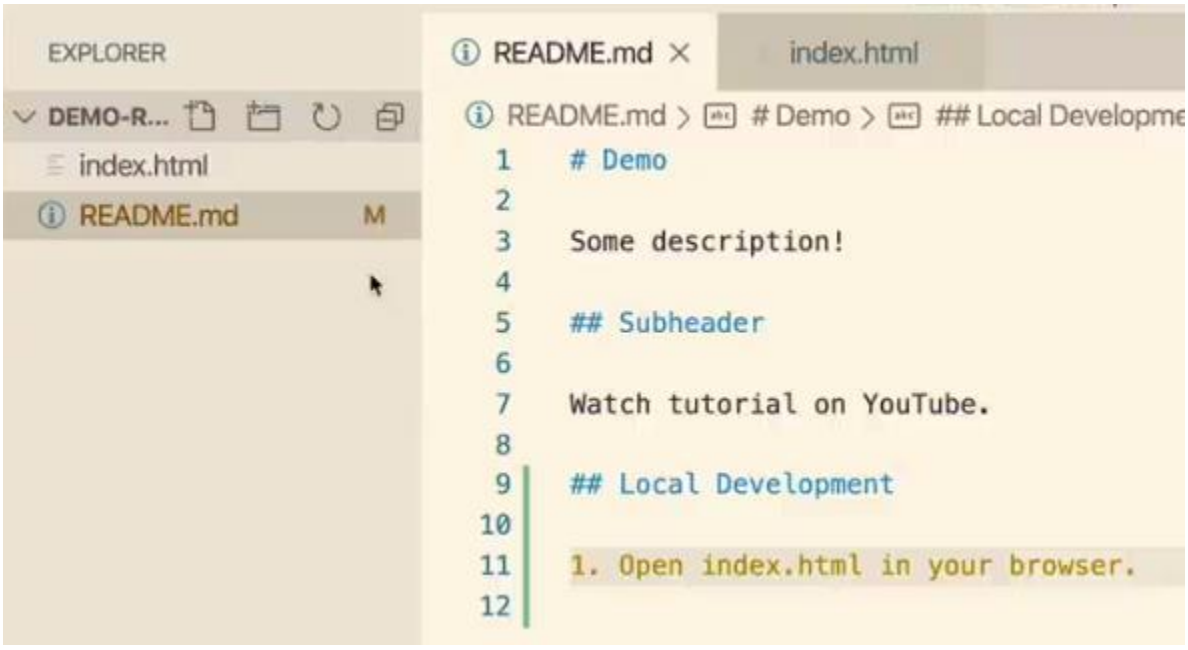
```
→  demo-repo git:(master) git checkout -b feature-readme-instructions
Switched to a new branch 'feature-readme-instructions'
```

Type **git branch** to verify that the branch changed. Type **git checkout master** to get back to master branch.

```
→  demo-repo git:(feature-readme-instructions) git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

```
→  demo-repo git:(master) git checkout feature-readme-instructions
Switched to branch 'feature-readme-instructions'
```

Now edit the files in this repo.



Type **git status**

```
→  demo-repo git:(feature-readme-instructions) ✗ git add README.md
→  demo-repo git:(feature-readme-instructions) ✗ git commit -m "updated readme"
[feature-readme-instructions cec9aeb] updated readme
 1 file changed, 4 insertions(+)
→  demo-repo git:(feature-readme-instructions) git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
→  demo-repo git:(master)
```

Now if we see the code it will be the same because the changes are on the feature branch.

Now we can merge the two branches locally. But before that we want to see the difference between the two codes.

```
→  demo-repo git:(master) git diff feature-readme-instructions
```

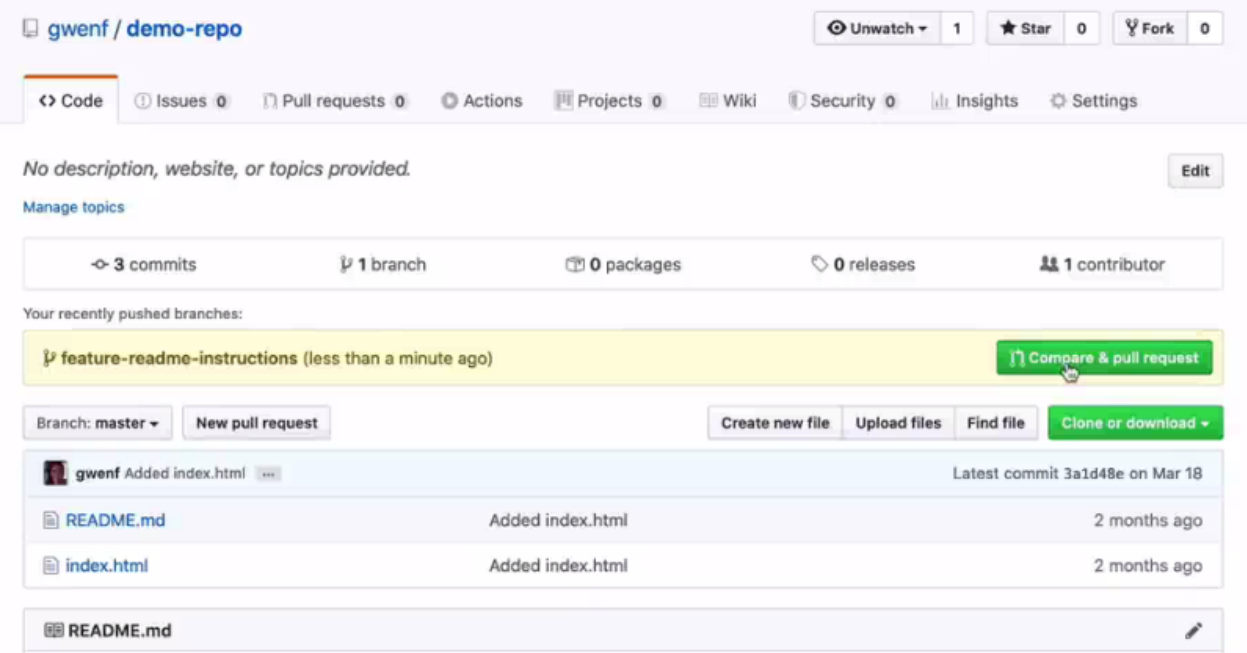
We can merge the changes to master using **git merge feature-readme-instructions** but more commonly the changes are first pushed to that branch on Github and then a Pull Request is made.

```
→  demo-repo git:(master) git checkout feature-readme-instructions
Switched to branch 'feature-readme-instructions'
→  demo-repo git:(feature-readme-instructions) git status
On branch feature-readme-instructions
nothing to commit, working tree clean
→  demo-repo git:(feature-readme-instructions) git push
fatal: The current branch feature-readme-instructions has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin feature-readme-instructions

→  demo-repo git:(feature-readme-instructions) git push -u origin feature-readme-ins
tructions
```

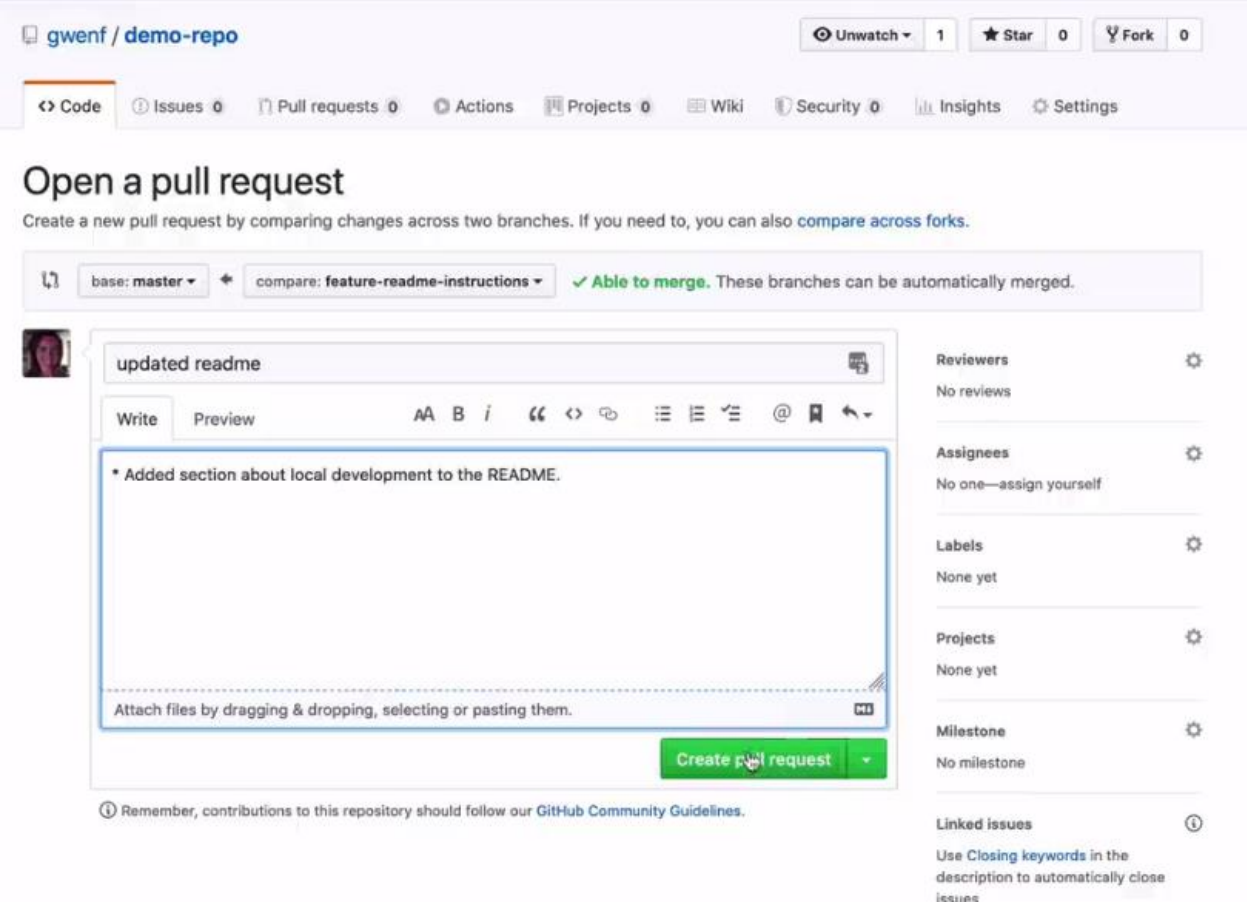
-u is same as –set-upstream.

Pull request – It is basically a request to have your code pulled into another branch. We need to pull our code from the feature branch to the master branch so we make a PR for that. Once we have made a PR, anyone can review, comment on our code, ask to make changes or updates. After we have made the PR, we can still make the commits to the code in the same branch. Once the PR is merged, we can delete the feature branch and switch back to the master branch. When we want to make changes again, we can create the branch again, make changes and make a PR again.

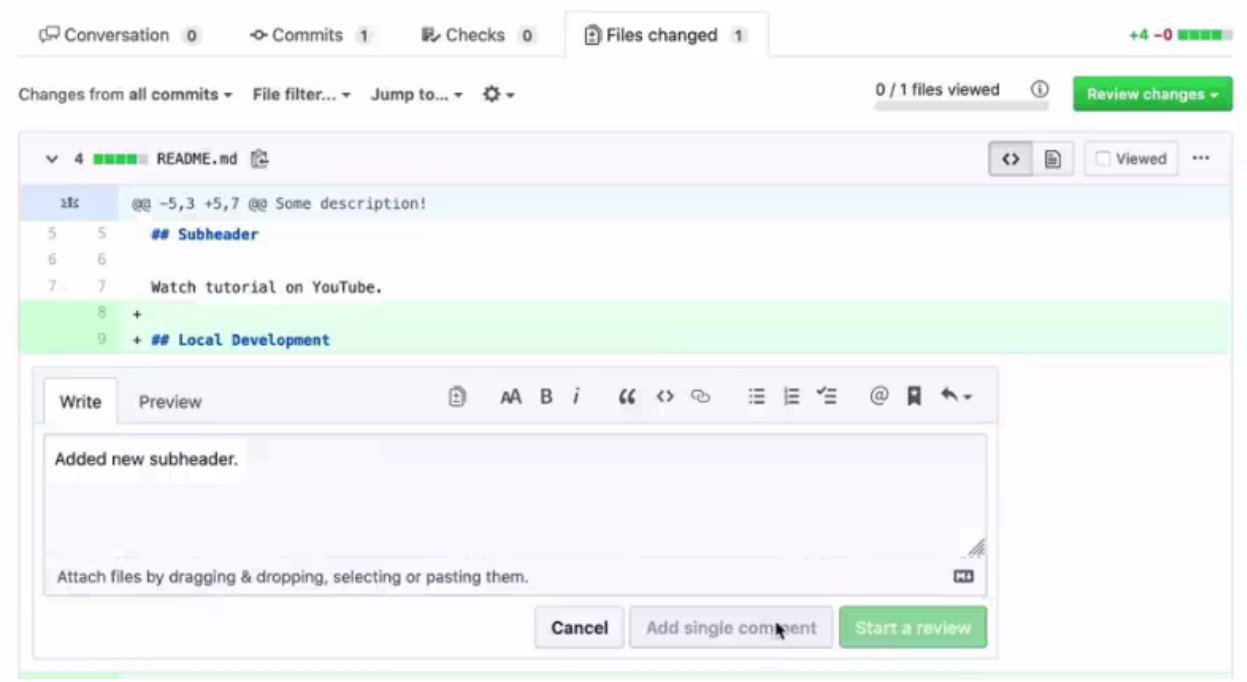
On github we can see that github picked up the new branch.



We can enter the changes made in the description.



After the PR is made, we can see any comments, commits, files changed, etc.



We can add comments to particular lines of code too.

Merge the request. Now if we go to the repo, we can see our changes and verify that the branch was successfully merged.

Now locally switch to the master branch by typing **git checkout master.**

Now if we see the code, the changes are not there. That is because they are only on github and we need to pull them down to our local environment.

```
→  demo-repo git:(master) git pull
```

Now if we see the code, we can see the changes.

If we type **git branch**, we can still see the feature branch there.

So we can delete it now.

```
→  demo-repo git:(master) git branch -d feature-readme-instructions
Deleted branch feature-readme-instructions (was cec9aeb).
```

Now **git branch** will only show master branch.

Merge Conflicts – We are building our own code on our own branch. Other people are writing code on their branches. And master is getting updated from multiple places. So it is possible for multiple people to change the same files. So sometimes Git does not know which code we want to keep and which code we want to get rid of. So we have to do that manually.

Fork – Makes a complete copy of the repository. We can make fork of other people's repositories to gain access. So we can make a PR of the repo and request our code to be added. Or we just want to do something with the code. Click the fork button on any random repo.

Now we will have the repo and we can make any changes to the code. Edit a file and commit the changes on github. The branches are copied too.

Now if we want our changes to be made a part of the original repo, then we create a new Pull Request.

We can see the base repo and the branch. We can click on Create Pull request. We can also merge into any other branch.