INTRODUCTION TO GIT AND GITHUB

By: Atharva Joshi

Contents

15. The End

What is GIT?

A free and open source version control system.

What is version control?

"The management of changes to documents, computer programs, large web sites and other collections of information."

It is basically a way that we as programmers track our code changes, we basically save an initial version of our code into GIT. When we update code we can save it into GIT again and again and again and throughout time as our code continues to change we can look back at all of the changes we 've made over time. This helps us to see what we did and when as well as track down bugs or go back to previous version of the code if we need to.

Terms used in this book

- Directory -> Folder.
- Terminal or command line -> Interface for Text Commands.
- CLI -> Command Line Interface.
- cd -> Change Directory.
- Code Editor -> Word Processor for Writing Code (VSC).
- Repository -> Project, or the folder/place where your project is kept.
- GitHub - > A website to host your repositories online.

GIT v/s GIT HUB

GIT is the tool that tracks the changes in your code over time whereas **GITHUB** is a website to host your repositories online. Being online it makes easy to work in groups with other people and organize your projects into a portfolio for you to show potential employers.

Git Commands

Here are some of the GIT commands we are going to cover in this section of the book that you are going to type in Terminal (MAC) or command line (WINDOWS) in your computer. They are:

- **clone** -> Bring a repository that is hosted somewhere like Github into a folder on your local machine. For example if there is a repository that is not on your local machine but it's on Github and you want to bring it down on your local machine, So you can use it locally.
- **add** -> Track your files and change in Git.
  When you have updated files or created or deleted files in folder you're gonna want to tell Git that you made changes and that you would like to get track of those changes so you use the add command.
- **commit** -> Save your files in Git
  As I said earlier that Git is there for you to save the changes of your code so you do that through a commit and we call it committing your changes.
- **push** -> Upload Git commits to a remote repository, like Github
  Once you have made changes locally on your computer and you are ready to put them in Git. You tell Git to track them through the add command you save your files to get through the commit command and then you upload your files to a place like Github or another what we call remote repository or Github alternative like Bitbucket, Gitlab and there's many of them you do this through push command.
- **pull** -> Download changes from remote repo to your local machine, the opposite of push.
  When there are changes to you code on Github and you want to bring those to your local machine then you use the pull command. You pull down the changes from the remote repository.

Follow the Steps as given below.

1. The first you will need to do is of course sign up for an account on github.

   Once you are signed-up/logged-in you will be directed to home page or profile page where you can see a green button with text "New" in it. It creates a new repository.

2. Click the new button to create new repository, give it a name and click create repository button.

   My repository name is "testproject".

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner *   Repository name *

[ ] atharva20-coder ▾   /   [   ]

Great repository names are short and memorable. Need inspiration? How about **verbose-fortnight**?

Description (optional)

[   ]

(●) 🖥 **Public**
    Anyone on the internet can see this repository. You choose who can commit.

( ) 🔒 **Private**
    You choose who can see and commit to this repository.

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

[ ] Add a README file
    This is where you can write a long description for your project. Learn more.

[ ] Add .gitignore
    Choose which files not to track from a list of templates. Learn more.

[ ] Choose a license
    A license tells others what they can and can't do with your code. Learn more.

[Create repository]

Now you can create your files and folders for this repository locally on your machine or you can create them straight in the online editor on the Github website. So I ' m gonna go ahead

3. Create a basic markdown file called a README and this is the most basic file that you will find in almost every project that contains text to describe what the project is about, what it does and any other relevant information.

Quick setup — if you've done this kind of thing before

[📲 Set up in Desktop] or [HTTPS SSH] https://github.com/atharva20-coder/testproject.git   [📋]

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

Markdown is basically an easy way to format you text in this sort of files. Now markdown has all these shortcut like one hash tag "#" for a main header.

testproject / README.md   [Cancel]

<> Edit new file   ⊙ Preview   Spaces ▾ 2 ▾ No wrap ▾

```
1   #testproject
2
3   Some Description
```

4. Scroll down to commit the file in other word save this file . I just want to point out quickly that this *"create README.md"* is a placeholder text, but if you don't write anything else here then when you commit this it will this text as the default.

5. Click Commit new file. And a README.md file can be seen on next screen.

6. If you edit REAFME.md, while committing file now placeholder shows update README.md, and a new message will be shown their which says Update README.md.
7. After committing this if we want to check history click History on top right corner.
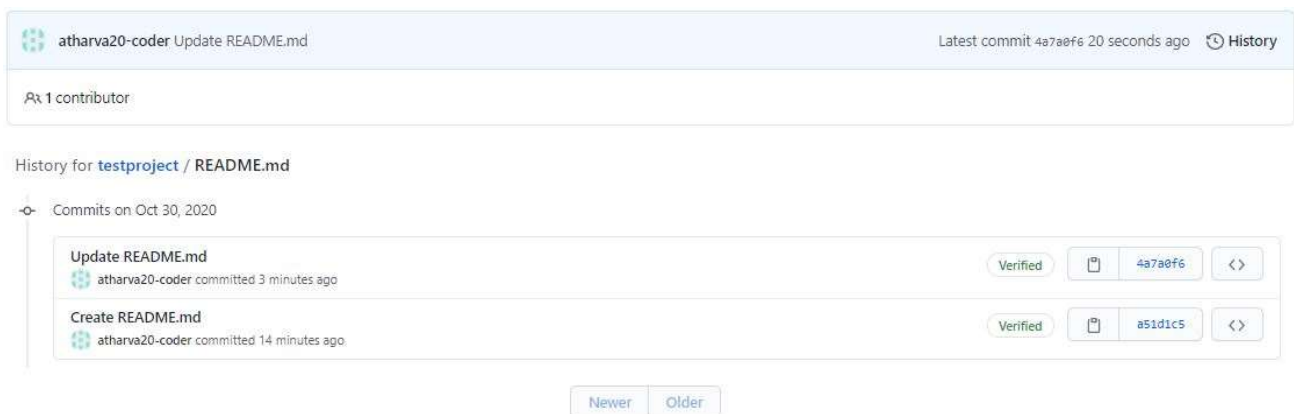
Doing same thing in local machine.

If you are using a Mac or Linux OS you should already have GIT installed you can check this by opening you terminal or command line application and typing in:

```
~ git –version
git version 2.20.1 (Apple Git-117)
~
```

If Git is not already installed it shows the following (go to last page and read how to install.)

For windows installation

When you download the Git Windows installer I recommend you to choose the Git bash option from the install menu. You can of course use command prompt and it will set Git in command prompt for you but in the past I' ve had a easier time with Git bash on Windows. For Mac it gives you few different option to install but I highly recommend to you use the homebrew package manager if you can and it takes through all of the steps here but again you probably have Git installed if you are using a Mac or Linux OS.

The last thing to do is getting a code editor but again I recommend Visual Studio Code.

8. Now open Visual code editor and create a new folder, I named this folder test git.

Pulling/Cloning the repository on local machine.

Now we are going to pull the repo (repository) that we created in the beginning to our code editor. So in visual studio code we have access to a terminal, Press " **ctrl+shift+`** " keys to open terminal.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                                        2: powershell         ∨   +  ⊡  🗑  ∧  ×

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\GITHUB\test git> []
```

It shows path of our created folder. This is because I opened the folder and it automatically uses that as the directory for my terminal, so now I want to pull down my repo to my local machine I want to pull that code down here so I can work with it locally I'm gonna use a command called **git clone.**

Go to terminal in VS code and type the command git clone SSH/HTTPS code by copying it from Github as shown in above image.

```
PS E:\GITHUB\test git> git clone git@github.com:atharva20-coder/testproject.git[]
```

9. Now check your folder and it would have the README.md folder that we created if the command run successfully.
10. Now I'm going to move inside this folder real quick from the command line which I can just do by typing **cd name of folder.**
11. When we go into the readme file and add a sub header and then a small text (basically we want to make changes in the file).
12. Now that I have changed this file I need to save the changes in Git. So I do that through a commit statement, so first I'm gonna use the **git status** command:

```
PS E:\GITHUB\test git\testproject> git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

It says your branch is up to date with 'origin/main'. Nothing to commit, working tree clean. Git status command shows me all of the files that were created updated or deleted but haven't been saved in a commit yet so for example.

13. If I create a new file let's say index.html in our folder and I am going to put a quick div inside our index.html file with nothing else and now let us rerun our command.

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

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

It shows untracked file (index.html) meaning that git doesn't know about this file yet, So you have to tell git to track the file before you can save it to git, to do that you need to use the git add command and then tell it which files to track.

Now most of the time lot of the time you see people use a period which means you are telling git to track all of the files that are listed, in the both untracked and the modified section, so both the changes we made to README.md and the new file index.html would be staged with git if we did get add period because it includes all of the files. You could also optionally tell it just the names of each individual file or folder that you want to get track in this case we are just going to use " **.** " as follows:

```
PS E:\GITHUB\test git\testproject> git add .
```

Adding all files

Or

```
PS E:\GITHUB\test git\testproject> git add index.html
```

Adding individual file.

Now, check the status again and it would show as follows:

```
PS E:\GITHUB\test git\testproject> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   index.html
```

Now they are ready to get committed.

14. So I 'm gonna go to terminal and type *git commit –m "message" –m "Some description"*.

[Note if it shows fatal error message then try run this commands:]

```
$ git config --local user.name "your-name"
$ git config --local user.email "your-email"
```

```
PS E:\GITHUB\test git\testproject> git commit -m "added index.html" -m "Some description"
[main 3b44c63] added index.html
 2 files changed, 7 insertions(+)
 create mode 100644 index.html
```

15. We still saved our code locally the commit isn't live on github yet.
16. We make it live by using another git command called git push click enter and authorize by signing into your github account through new window that pop up.

[Note if you get an error you might not have added a SSH key to github to add SSH key follow the steps]

SSH KEYs (Secure Shell)

An SSH key is an access credential for the SSH (secure shell) network protocol. This authenticated and encrypted secure network protocol is used for remote communication between machines on an unsecured open network. SSH is used for remote file transfer, network management, and remote operating system access.

In order to push them to github you have to prove them that you are the owner of the account, so you have to connect your local machine to your github account somehow. The way this is done is by using SSH keys. You need to start generating key locally using the SSH keygen command as follows:

Windows environments do not have a standard default unix shell. External shell programs will need to be installed for to have a complete keygen experience. The most straight forward option is to utilize Git Bash. Once Git Bash is installed the same steps for Linux and Mac can be followed within the Git Bash shell.

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

> Enter a file in which to save the key (/Users/you/.ssh/id_rsa):
[Press enter/keyname(optional)]

> Enter passphrase (empty for no passphrase): [Type a passphrase]
> Enter same passphrase again: [Type passphrase again]
```

Make sure that the email address you entered here is the same one you use to log in to your github account.

Now the default file for the SSH key is inside is inside the given directory `/Users/you/.ssh/id_rsa` .

You can search for the key that you generated by the commands

```
Ls | grep test key
```

Now the key with .pub is the key that you are going to upload to your github interface PUB stand for Public it's called your public key which means that it is okay for other people to see this key the that was generated without the .pub extension is called you private key and is the one that you have to keep secure on your local machine. How it works is that the public key you put on github and then every time you want to connect to github or push your code on github or use you account via your local machine you use your private key to show github that you are the one that generated this public key it's the mathematical proof that only the private key could have generated Public key.

By writing the command below copy the whole key and paste it in your github profile.

```
$ clip < ~/.ssh/id_rsa.pub
# Copies the contents of the id_rsa.pub file to your clipboard
```

Tip: If `clip` isn't working, you can locate the hidden `.ssh` folder, open the file in your favorite text editor, and copy it to your clipboard.

Follow the step for adding key to your github profile.
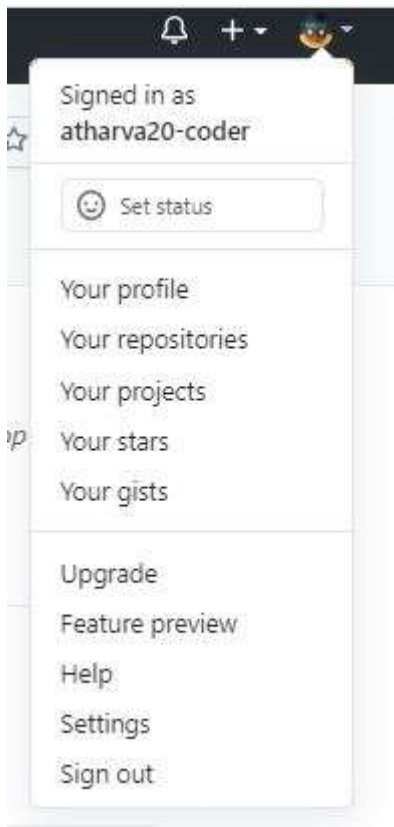
click on Settings

click SSH and GPG key

click New SSH key button. And paste the key that you copied in the key box ad give it a suitable title that you want like "my first SSH key". Click Add key and you are done.

SSH keys / Add new

Title

[ ]

Key

Begins with 'ssh-rsa', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

Add SSH key

The only thing left to do is to make sure that your local git command line interface knows about the key we just generated you can also read this from clicking the link :------

1. You need to start SSH agent, then if you are using version 10.12.0.2 or later you will need to modify this file "~/.ssh/config".

CLINK THE LINK TO READ THE FULL SETUP.

GIT PUSH

Now there are two more arguments we have to put in the terminal, the first one is origin and yhe second one is master.

Origin is an option set for us here and is basically a word that stands for the location of our git repository.

Master is the branch that we want to push to

```
git push origin master
```

And now our code changes are all live on github.

README.md

#testproject

header

Some Description!

subheader

Write BOOK properly
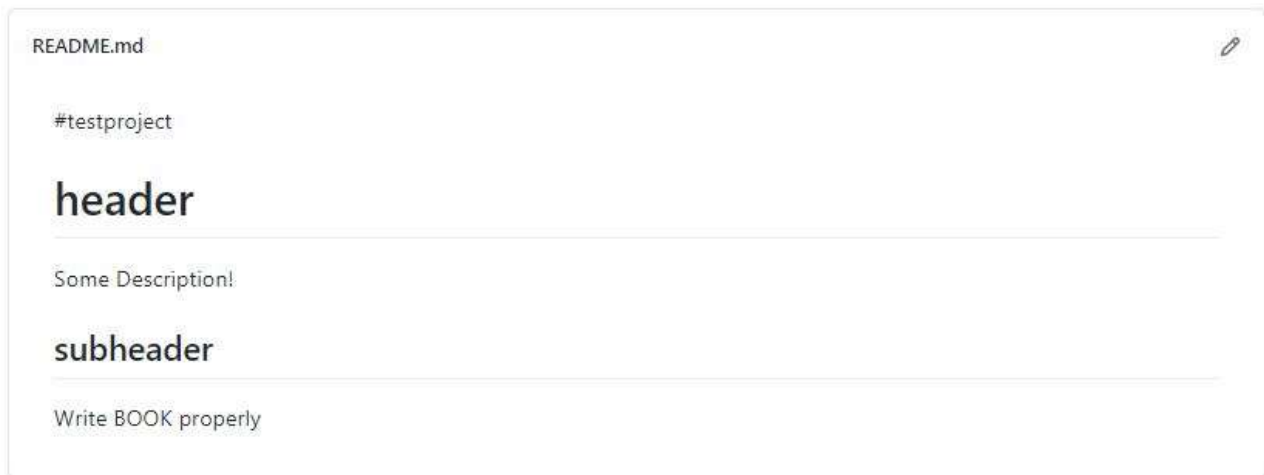
So this is to demonstrate that how to create a new git repository github but what about if we start a repo locally.

Creating repository on local machine.

Create a new folder and name it (I named it as demo-repo).

1. Nothing in there in this folder, so I'm going to move into that folder in the terminal by this command:

```
cd ../demo-repo
```

2. Now let us add a README.md file in demo-repo folder and add header and some text. Now if I want to turn this into a git repository I use the command line in VS code terminal.

```
git init
```

```
PS E:\GITHUB\demo-repo> git init
Initialized empty Git repository in E:/GITHUB/demo-repo/.git/
PS E:\GITHUB\demo-repo>
```

3. Now if we check status "git status"

```
PS E:\GITHUB\demo-repo> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

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

We can see that untracked README.md file.

4. Now we will go ahead and add that file either with the period or the name of the file.

```
git add README.md
```

Now again checking the status.

```
PS E:\GITHUB\demo-repo> git status
On branch master

No commits yet

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

5. Now the file is ready to get committed, now I can commit.

```
git commit –m "Created README file" –m "Some description"
```

```
PS E:\GITHUB\demo-repo> git commit -m "crated README file" -m "some description"
[master (root-commit) f543055] crated README file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
```

Now what if I want to push this live, let us try:

```
git push origin master
```

```
PS E:\GITHUB\demo-repo> git push origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```
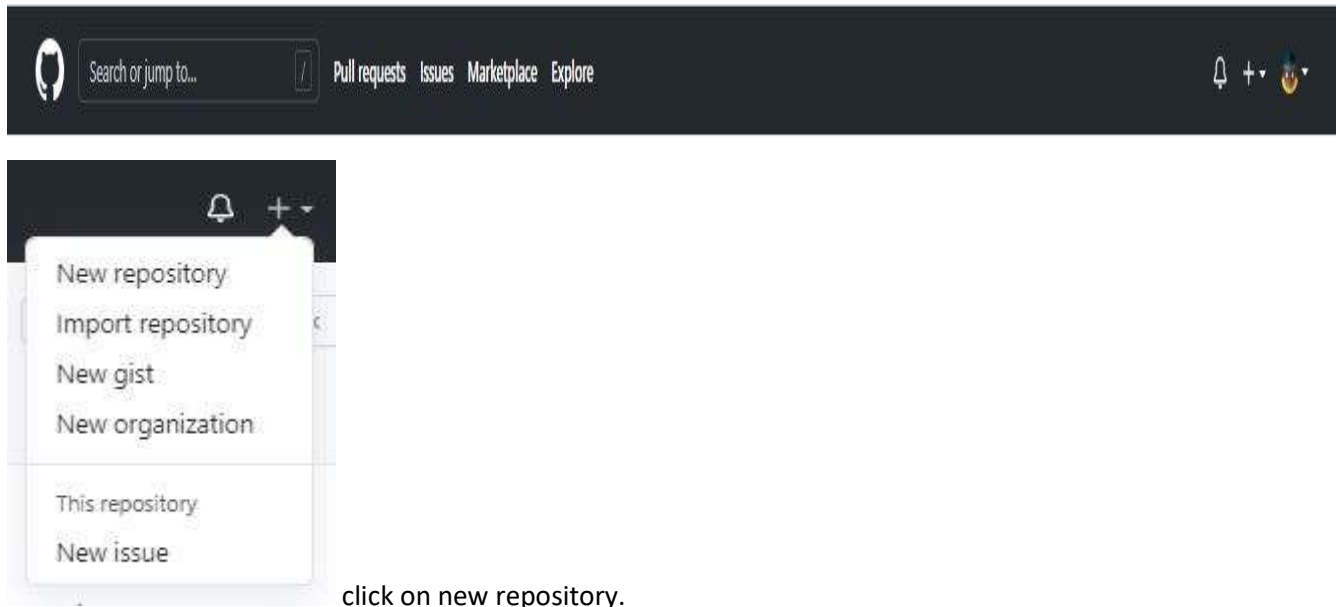
It shows fatal error origin does not appear to be a git repository because we didn't clone this down from a git repository, we didn't already created this repo on github we created it locally, so git is saying that I have no idea where to push this because this is not connected to anything.

So we have to create that connection.

And first the easiest way to do this by an empty git repository up on github.

Steps:-

1. Go to top of your navbar and on the right corner there will be a **"+"** sign click on it.

 click on new repository.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner *

Repository name *

atharva20-coder ▾ / [   ]

Great repository names are short and memorable. Need inspiration? How about **verbose-fortnight**?

Description (optional)

[   ]

⦿ 🖥 **Public**
   Anyone on the internet can see this repository. You choose who can commit.

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

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

☐ **Add a README file**
   This is where you can write a long description for your project. Learn more.

☐ **Add .gitignore**
   Choose which files not to track from a list of templates. Learn more.

☐ **Choose a license**
   A license tells others what they can and can't do with your code. Learn more.

Create repository

Now follow the steps as done previously.

2. Now we have another empty repository so to start pushing we are going to copy the SSH key.
3. Now, in VSC terminal we pass our command

```
git remote add origin git@github.com:atharva20-coder/demo-repo2.git
```

Remote means somewhere else but not on this computer we're gonna use this to add a reference to the remote repository on github and then I'm gonna say **add origin** and paste the copied key.

4. And we can check that by passing the command

```
git remote –v
```

```
PS E:\GITHUB\demo-repo> git remote -v
origin  git@github.com:atharva20-coder/demo-repo2.git (fetch)
origin  git@github.com:atharva20-coder/demo-repo2.git (push)
```

It shows any remote repository that I have connected to this repo, now that these are setup.

5. We can now pass git push command

Theirs is a shortcut here because I don't want to type out this whole thing every time in the future I can just use git push but I have to set something called an upstream meaning this is where I want to push it to by default so we can apply this thing as follows:

```
git push –u origin master
```

-u is for upstream. Now in the future I can just use **"get push"** without typing out **"origin master"** .

```
PS E:\GITHUB\demo-repo> git push -u origin master
Enter passphrase for key '/c/Users/HP/.ssh/id_rsa':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 238 bytes | 18.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:atharva20-coder/demo-repo2.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

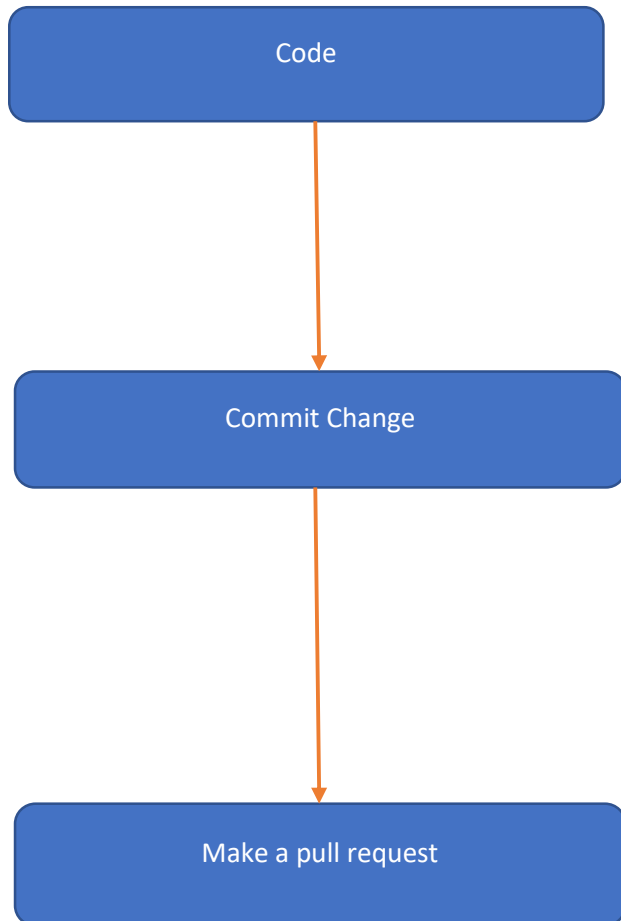**Now refresh your github page,** and you can see you README file their.
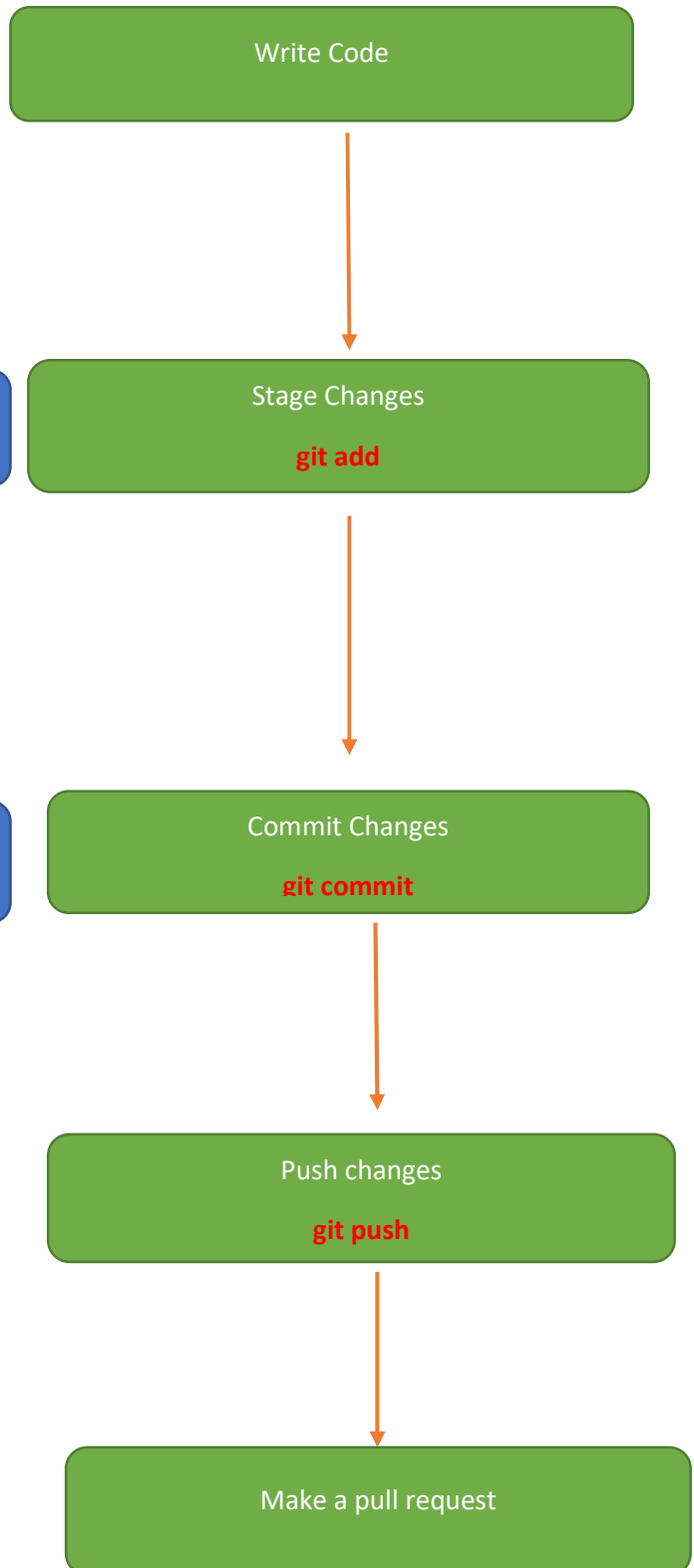
testproject / README.md   Cancel

<> Edit new file   ⊙ Preview   Spaces ⬍ 2 ⬍ No wrap ⬍

1   #testproject
2
3   Some Description

So that's how we initialize a repository locally and push it to a remote location like github.

Github Workflow

Local Git Workflow

```
Code
```

```
Write Code
```

```
Commit Change
```

```
Stage Changes

git add
```

```
Make a pull request
```

```
Commit Changes

git commit
```

```
Push changes

git push
```

```
Make a pull request
```

Git Branching

🟥 **Master Branch**

🟩 **Feature Branch**

🟨 **Hot Fix Branch**

Commit #1

Commit #1	Commit # 2	Commit #3	commit #4	Merge	Merge
🟥	🟥	🟥	🟥	🟥	🟥

Commit #1   Commit #2
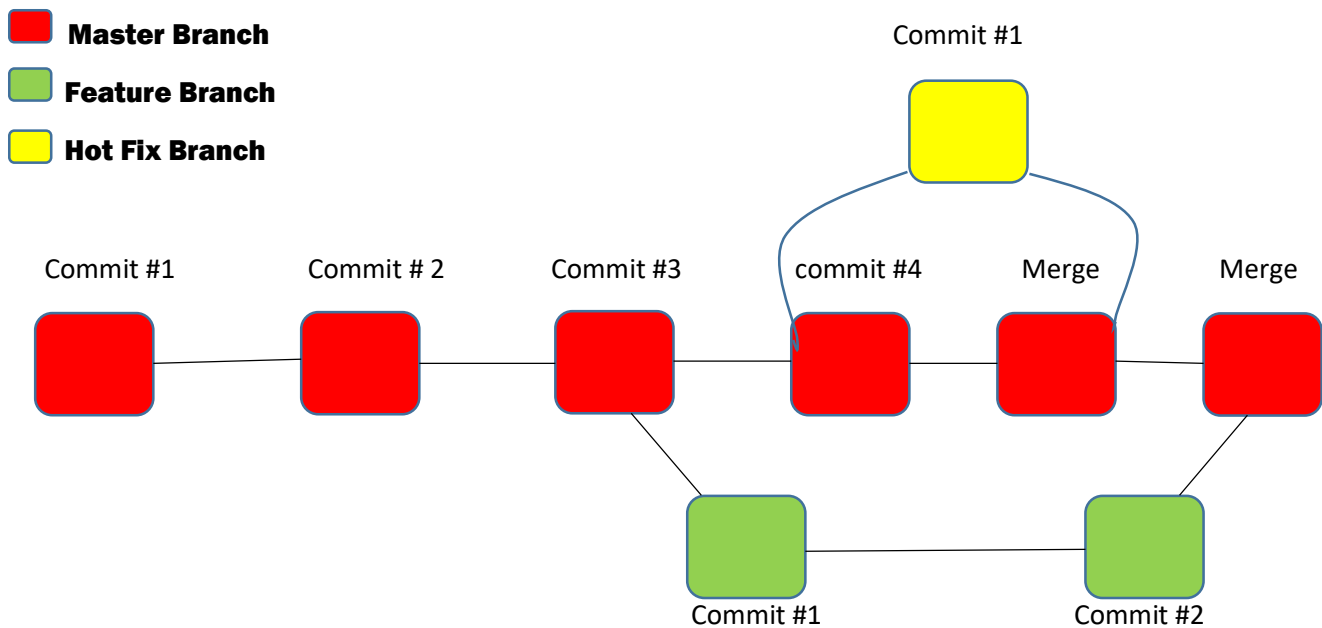
Let's first go over some of the concepts of branching then we'll dive into the practical example.

What is Git branching?

You have seen in our example that were on something called the master branch, now master is the naming convention for the main or the default branch in a repository so if you are working off of one branch that's where your code will live and all your code and all your commits everything will be in that one branch now this is called branching because it starts to look more like a tree when you have multiple branches.

So we saw earlier that we had a master branch we were committing to that master branch but we could also make another branch we'll call it feature branch. So let's talk about what happens when we create a new branch. At first the code on master and this new feature branch will be exactly the same but as you make updates the feature branch those changes are only seen in the feature branch so if you make changes on the feature branch you make a commit to save those changes onto that branch and then you switch back to the master branch, you won't be able to see any of the changes that you made on the feature branch each individual branch has no way of knowing what commits and what changes have been made to any other branch each is only keeping track of changes are made on its own branch now you can continue to change the code on master branch and save it through a commit to the master branch and then if you switch back to feature branch likewise you will not be able to see the changes that you just saved to master.

Why this is useful it's extremely useful? , because you will be building out the new features to applications that may break your code or they are not finished yet and you don't want to save them to the main master branch you want to work on them in kind of a sandbox area so you can write all the code you need and get it correct and in the state that you want before you merge it back into master branch of the code base and this is really helpful when have many different people working in the same repository or a lot of different branches going on at once.

One very common thing that you see in development is that you'll be working on let's say this feature branch for week or more and you have all these changes made and then you find out there's this major bug that you have to fix real quick so for that this other branch called it a hotfix branch here represented by yellow and you can see that I made the change in one commit to fix whatever bug was wrong with the code on its own branch I made sure it was working correctly and then I merged it back in to master branch.

Now that we've gone over the concepts of branching let's dive into the command line try it out.

1. Let us first check the branch in our project.

```
git branch
# it shows
*master
```

The output of the command reflects **"*master".** It means we currently have only one branch and it's master and you can also notice that there is a star beside it and that means that I am currently on that branch.

2. Now, let's create new branch and call/name it **"feature-readme-instruction",** so we do that by passing suit of commands.

```
git checkout –b feature-readme-instructions
switched to new branch 'feature-readme-instructions'
```

Git checkout is what you will use to switch between branches but to tell it to create a new branch use **"-b"** and then you give it the name of your branch, now if you are working in a real application with people you want to make this name as descriptive as possible some people like to use **"-", "/"** you can make this longer if you want**.** I named this branch **"feature-readme-instructions"** because I am going to be updating a section of the **"readme".**

You can see it says **"switched to new branch 'feature-readme-instructions'"** it means now what else we do it's going to save in this current **'feature-readme-instructions'** branch.

3. Now, if we again check branch

```
git branch
# it shows
*feature-readme-instructions
master
```

It gives star mark beside **'feature-readme-instructions'**.

4. Now, if we want to switch our branch we can pass a command like

```
git checkout master
# it shows
Switched to branch 'master'
```

You do not have to do this as this is just to show you how to interchange between branches.

[**Note:** further work will be done in **'feature-readme-instructions'** branch.]

[**Tip:** click the tab button and it auto filled the rest of the branch so I didn't have to type out the whole long name you can also hit tab even if you don't start at the beginning we could actually start typing instructions

and hit tab and it still auto completes, now this is very useful because eventually when you work in a repo you are gonna start a lot of stuff with feature or with bug or hotfix. **]**

5. Now we are going to make some changes to this branch.
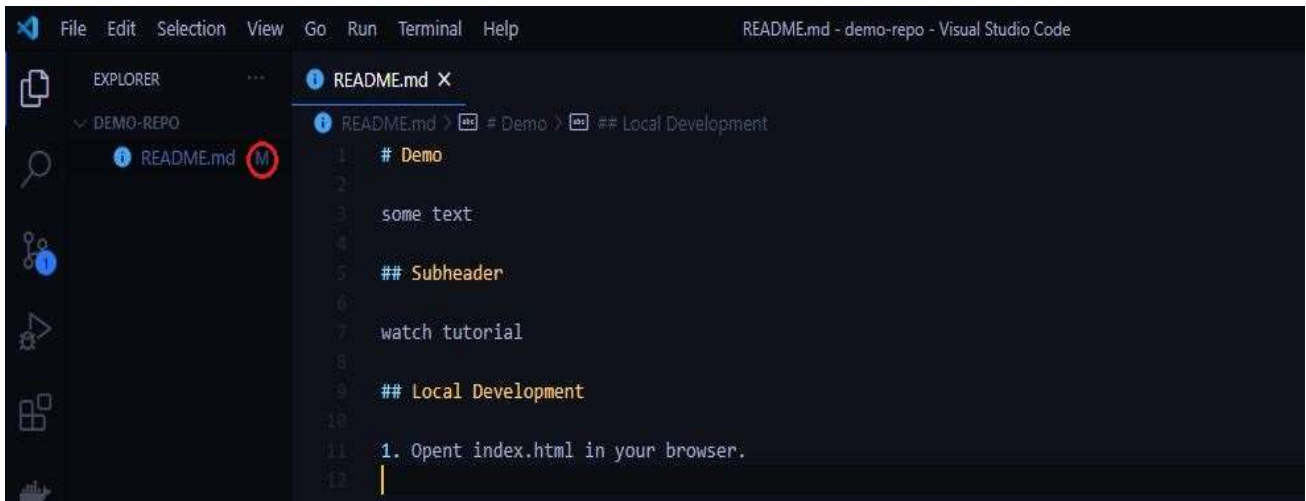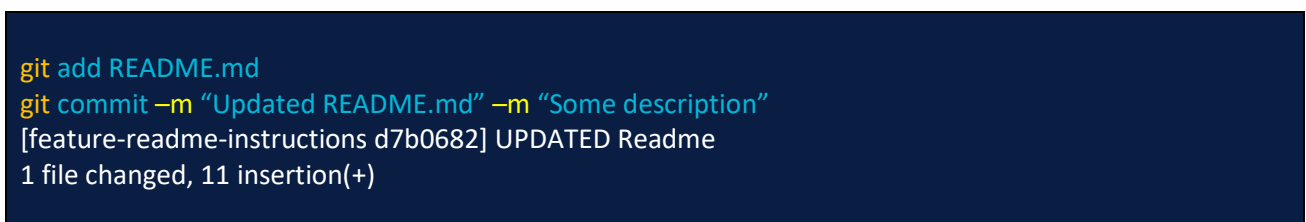

When you save your modification you can see letter 'M' beside your file this signifies that this file has been modified.

6. Now if again check the status.
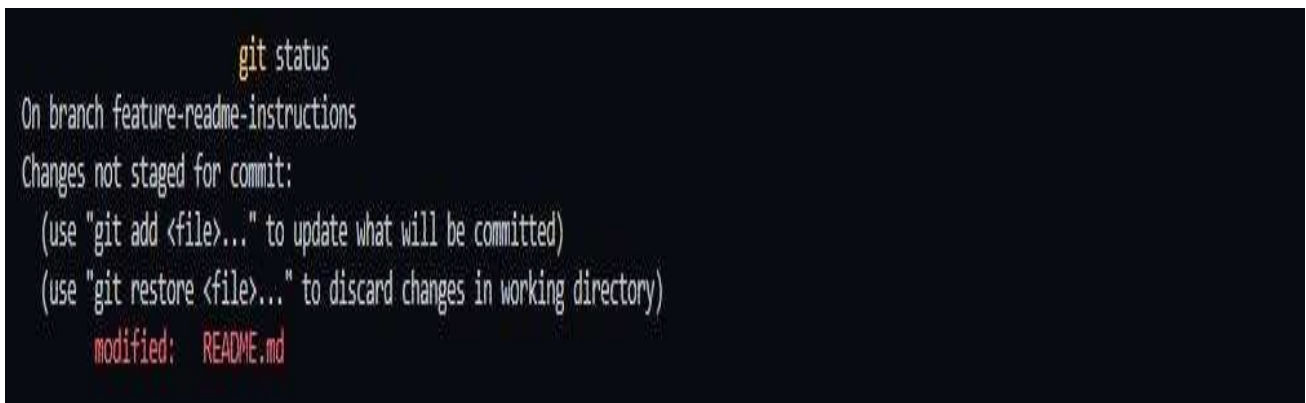
It says modified README.md, So first I am going to save things that I've changed on this branch.
7. For saving pass the below command

git add README.md
git commit –m "Updated README.md" –m "Some description"
[feature-readme-instructions d7b0682] UPDATED Readme
1 file changed, 11 insertion(+)

It's only saved on this **'feature-readme-instructions'**.

8. If I do git checkout to master and see the editor.

git checkout master

We can see that there is no changes in the editor.**[Now all the work will be done in the master branch]**

9. Now, if I want to here I can go ahead and merge these two branches which is locally using the git merge command. But before doing that I want to check and see the code that emerging in which I can do through the git diff command.
Now diff or diffing shows me what changes have been made it compares two versions of the code and shows us all of the lines that have been changed

```
git diff feature-readme-instructions
```

```
PS E:\GITHUB\demo-repo> git diff feature-readme-instructions
diff --git a/README.md b/README.md
index aa84265..e69de29 100644
--- a/README.md
+++ b/README.md
@@ -1,11 +0,0 @@
-# Demo
-
-some text
-
-## Subheader
-
-watch tutorial
-
-## Local Development
-
-1. Opent index.html in your browser.
```

10. Now I could go ahead and like I said use the git merge command pass in the branch and it will merge in those changes but more common pattern that you are going to see is pushing this changes on that branch up to github and then making a PR so I'm going to change back to feature branch and now I'll do git status.

```
PS E:\GITHUB\demo-repo> git status
On branch feature-readme-instructions
nothing to commit, working tree clean
```

It says I have no changes to commit because they've already been committed but I do need to push them up to github so let me run git push.

```
PS E:\GITHUB\demo-repo> 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
```

You can see since I'm on new branch it says that I need to tell on github I want to push to, now the answer is almost always that the branch will be named the same thing on github and your local machine so could copy the last line of the above error that says **"git push –set-upstream origin feature-readme-instructions".**

```
git push –u  origin feature-readme-instructions
```

--set-upstream flag here is the same thing that we were doing before with the "-u" it's just a shorthand for set-upstream. [Press enter].

```
PS E:\GITHUB\demo-repo> git push --set-upstream origin feature-readme-instructions
Enter passphrase for key '/c/Users/HP/.ssh/id_rsa':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 358 bytes | 19.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature-readme-instructions' on GitHub by visiting:
remote:      https://github.com/atharva20-coder/demo-repo2/pull/new/feature-readme-instructions
remote:
To github.com:atharva20-coder/demo-repo2.git
 * [new branch]     feature-readme-instructions -> feature-readme-instructions
Branch 'feature-readme-instructions' set up to track remote branch 'feature-readme-instructions' from 'origin'.
```

Now it's pushed up to github and it's (message in above image) even giving me a hint for how to create a pull request on github by giving above instructions here.

Branch Merging.

So what is a pull request or a PR, it's basically a request to have your code pulled into another branch for example we're using a feature branch and we want have our code pulled into the master branch so we make a PR from the feature branch to the master branch now once we have made a PR anyone can review our code comment on it and ask us to make changes or updates.

Now after you make a PR you can update the code just by making additional commits and pushing them up to github as long as it's on the same branch that you're making a PR with. Once the PR is merged you'll generally delete you feature or source branch and to switch back to the master branch then when you want to make additional coding changes you will create another new branch and start the process over make your commits, make a PR and then merge again and I could go too straight to this URL but I'm going to show you how to do it manually on github first .
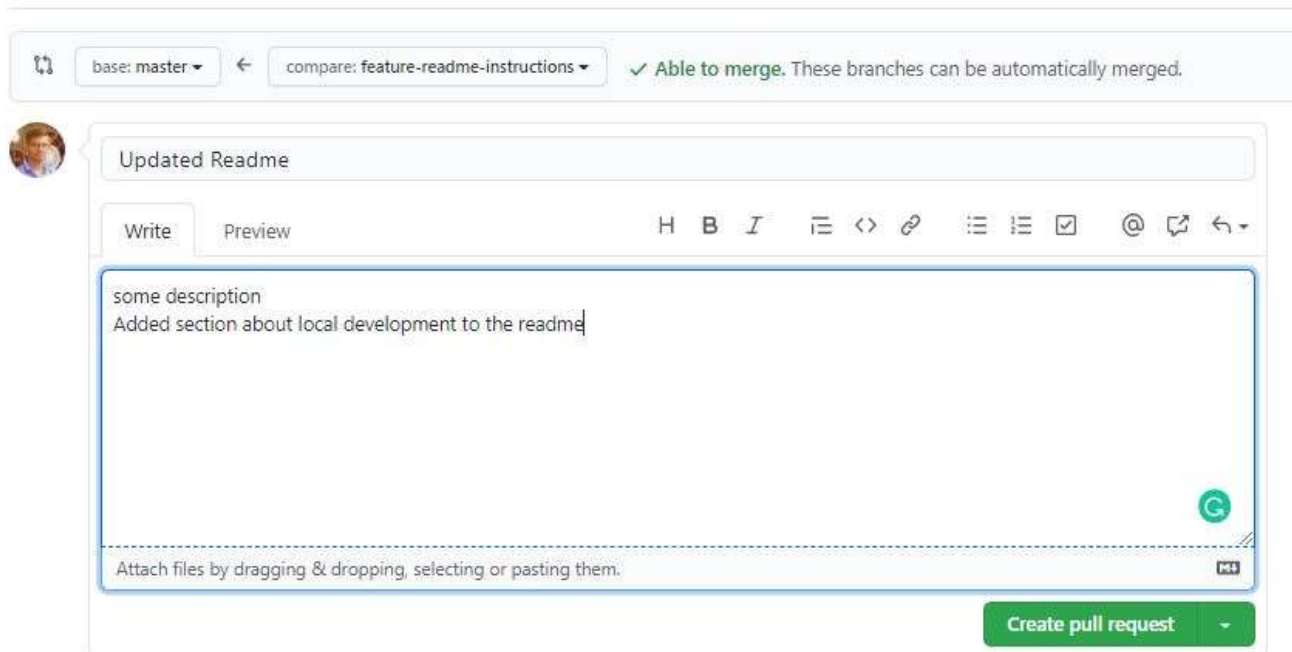
So go to your github interface

⎇ feature-readme-instructions had recent pushes 28 minutes ago   Compare & pull request

You can see already github picked up on the fact that I just pushed a new branch. So now its asking me right here if I want to compare this new branch to another branch and create a pull request and it takes me to this screen:-

Now sometimes this is confusing to people which branch is being merged into which branch this github interface does a pretty good job because of the arrow

 so you can see that the branch you're comparing this feature-readme-instructions will be merged into the base there's always a base branch where all the code is going to get merged into and then there is a branch you are comparing.

So now I have this message I've a title that says Updated Readme and I can put some more description so usually I'm gonna put a list of the changes that I've made or may be some description as to why I made certain decisions so here I'm just gonna say **"Added section about local development to the readme",** now I'll create a pull request.

11. Now, that I've created a pull request I actually come here on interface I can see any comments that people have made here on the pull request or I can write comments of my own look through all of commits I've made that I'm trying to merge into master right now and I can also look at all the files that have been changed and see the diff or what's been added removed or updated.

Updated Readme #1

Open · atharva20-coder wants to merge 1 commit into master from feature-readme-instructions

Conversation 0 · Commits 1 · Checks 0 · Files changed 1

atharva20-coder commented 1 minute ago · Owner

some description
Added section about local development to the readme.

Another thing that I can do here is if I want to make a comment about a particular line of code that I've written.

I can press the plus button on the blue button which appears when you hover over any line, and now I can make a comment there I'm just gonna say "added new sub header and add it as a single comment"

And click on "Add as single comment". And you can see that the comment appears at the line of code where you hit the plus button.

12. So I'm going to go ahead and merge this PR and you'll notice that I added the single comment it says resolve, now because I'm the owner here I have full permissions I don't have to resolve I can still merge the PR if have permissions or your repo set up in a certain way then you might have to resolve comments before you can merge.

-o- Updated Readme ···   d7b0682

atharva20-coder reviewed 6 minutes ago   View changes

README.md

```
    6  +
    7  + watch tutorial
    8  +
    9  + ## Local Development
```

atharva20-coder 6 minutes ago   Author   Owner   ☺   ···

Added new subheader

Reply...

Resolve conversation

[this section is visible on conversation section]

Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

✓ This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request ▾   You can also open this in GitHub Desktop or view command line instructions.

Pull request successfully merged and closed   Delete branch
You're all set—the feature-readme-instr... branch can be safely deleted.

Now that the PR is merged I'm going to go over to the code and if I scroll down you can see that I am on the master branch and my changes are also here. So the branch was successfully merged.
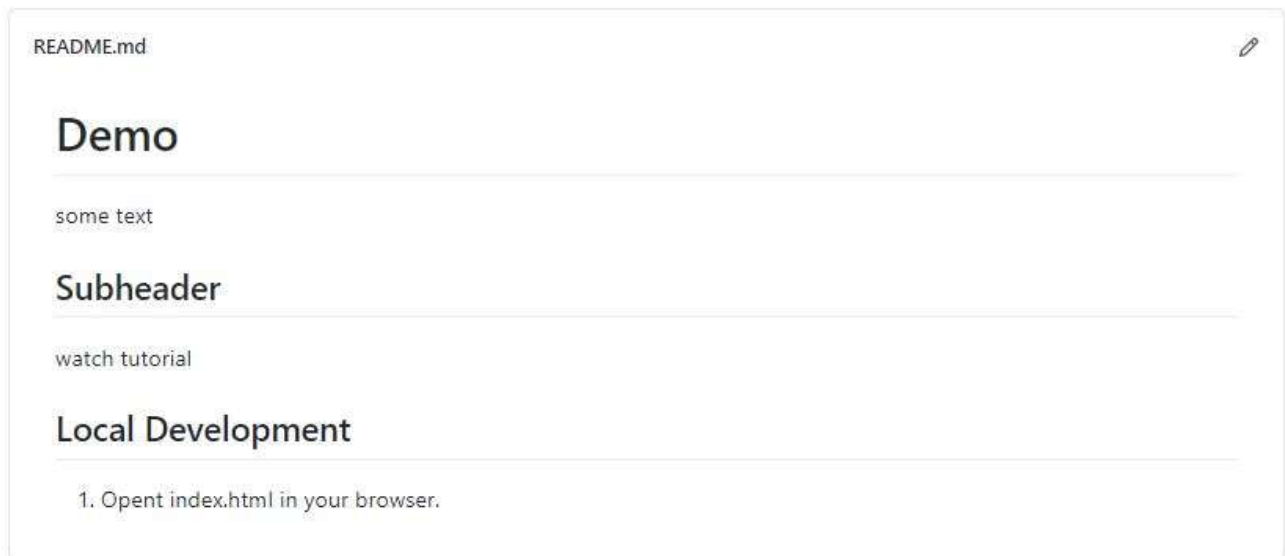
So locally now I'm going to switch to the master branch, and if you look at the code the changes aren't there yet and that's because they are only on the github and I need to pull them down to my local environment. So to get the changes on my local master branch from origin I'm going to do git pull:

```
git pull
```

If I didn't already set the upstream I would have to do "git pull origin master" but I already have an upstream set here so I'm just do git pull then press enter.

```
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), 647 bytes | 9.00 KiB/s, done.
From github.com:atharva20-coder/demo-repo2
 * branch            master     -> FETCH_HEAD
   f543055..ad04d83  master     -> origin/master
Updating f543055..ad04d83
Fast-forward
 README.md | 11 +++++++++++
 1 file changed, 11 insertions(+)
```

You can see 1 file changed and 11 insertions. And now if I look at my code I can see my local master has also been updated with the changes from branch.

And now that I'm done with that feature branch I'm going to pass a command

```
git branch
# it shows
*master
```

And It's still there even though I'm not going to be using it again because you don't really reuse branches that already merged into your master branch. So I'm gonna go ahead and delete this branch so I'm gonna pass command:

```
git branch –d feature-readme-instructions
Deleted branch feature-readme-instructions (was d7b0682).
```

[Press enter]

And now it says I deleted that branch so now if I do git branch I only see master branch again.

Git Conflicts.

In real life you are not always going to have such an easy time merging there is something called merge conflicts in git where basically you're building your own code writing a bunch of code on your own branch maybe other people are writing code on their branches and master is getting updated from multiple different places so it's possible for multiple people to change the same files and so sometimes git doesn't know which code you want to keep or which code is redundant or which code do you want to get rid of so you'll we have to manually do that, let's try that out.

Let me create another branch as created above and name it as **"quick-test".** Now come on the quick test branch now I'm gonna create a index.html file and add it using terminal. Now commit your work, also create index.html file in master branch.

Master branch	Quick-test branch
```index.html > p 1 hello 2 there ```	```index.html > p 1 Hello 2 world ```

You can see that in master branch second line in paragraph tag contains "there" and in quick-test branch it contains "world" in it, now the master has the line 2 and the new branch I created also has line 2, now everything is fine now because both branches are separate but what if I try to merge these two together whose line 2 is going to be put there, So let's try go to my quick-test branch.
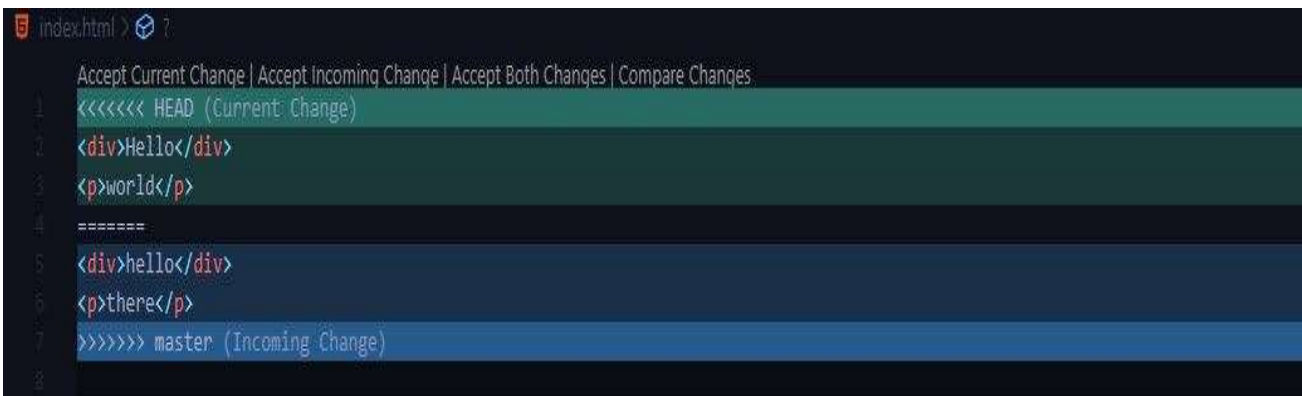
```
git checkout quick-test
```

1. Now I'm going to do merge master so remember how I said before that merging locally wasn't the regular pattern well it isn't for merging branches into aster but master gets updated as you go along as you're working on your project because maybe other people are merging into master and you don't have those changes in your branch but you don't want to get too far behind master as you're working because then it's going to be really difficult to merge later so changes are made to master in the github repo you're going to want to pull those down to your local master branch and then whatever feature branch you're working off of like quick test here you'll want to use git merge master to keep your branch up to date with what's going on in master. So that's what we are doing here:
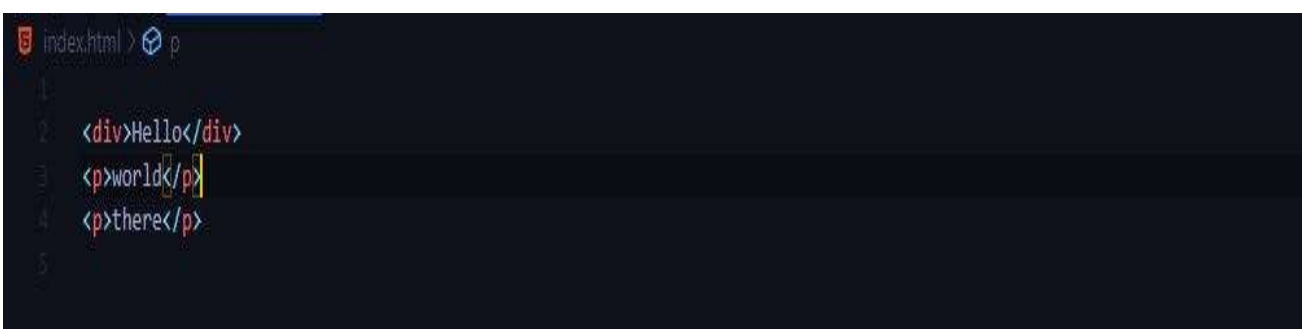
```
git merge master
```

It says we have a merge conflict in index.html

Now there are couple of ways to fix merge conflicts interfaces like github gives you an interface to fix them you can also fix it in terminal but the easiest way to fix the merge conflicts is directly in your code a nice code editor like VS code will let you click buttons like **"accept the current changes" or "accept incoming change" or "accept both changes":**

But you can also come in code editor make the changes that you want and delete the lines that git added this let's get know that there's a conflict and first between this head and these equal signs it shows the code that have on the current branch that we are on and then it shows the code coming from the other branch that we're trying to merge in. So in this case I am just going to delete the lines that git added and save both of the changes.

This just letting it know what we want to do because it didn't know how to handle the conflict. Now we have handled it and deleted the conflict markers. We now check the status again

```
PS E:\GITHUB\demo-repo> git status
On branch quick-test
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both added:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
PS E:\GITHUB\demo-repo> git checkout quik-test
error: pathspec 'quik-test' did not match any file(s) known to git
```

Now we need to make another commit because we fixed the merge conflicts.

So I'm gonna pass a command in terminal.

```
git commit –am "updated with master"
```

It shows **"updated with master"**

Now I've officially updated my branch with the changes that have been happening on master branch and I can continue coding whatever feature bug-fix or whatever I'm doing on this quick test branch.

Undoing in Git

So what if we make a mistake, what if we accidently add something or commit something to git and we didn't mean to do it we can actually undo our stages or our commits.

So let us come to README and another line there that says **"Have fun".**

And check the status again.

So now let's say I want to add README.

```
PS E:\GITHUB\demo-repo> git status
On branch quick-test
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
PS E:\GITHUB\demo-repo> git add README.md
PS E:\GITHUB\demo-repo> git status
On branch quick-test
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
```

On first check it said "Changes not staged for commit", but on second check it said "Changes to be committed", meaning they have been staged already but what if I want to undo staging what if I didn't mean to staged this file. I can actually easily undo it by git reset command I would pass git reset command with no argument or I can say git reset README.md that I want to unstage:

```
git reset
```

```
PS E:\GITHUB\demo-repo> git reset
Unstaged changes after reset:
M       README.md
PS E:\GITHUB\demo-repo>
```

Now check status (yourself). It would give the message that appears on first commit in above status check image.

So what if I want to undo a commit so let me go ahead and add the readme file and then I'm gonna commit readme. Well if I made a mistake on that commit I can use that same git reset command and this time I'm going to say git reset HEAD [all capital letters] and what head means to git is the pointer to the last commit so I'm telling you to do something with the last commit.

And then I'm gonna say till the one which is telling it instead of pointing to this last commit that I made, I want to go back one further and head will now point back one commit further meaning it will completely undo the commit that I just made.

```
git reset HEAD~1
```

It says Unstaged changes after reset: README.md.

Now if you have many different commits you might want to go back to a specific commit or undo multiple commits and for that there's no quick pointer in git like the upper case head pointer where you can just look at the last commit or use that to undo the last commit.

```
git log
```

But you can see log of all your commits and you'll see that these are arranged in reverse chronological order

You can see the latest commit that we have made and you can see with the commit message also why it's important to write a good commit message because it's easier for humans to scroll through and see which commit is which to get these commits are differentiated with unique hash **"commit 2335d08fac70168d399123fcea4f89e1904972b6"** so if you want to go back to a certain commit you can copy one of this hashes.

Forking

One thing I quickly want to mention is forking you may have seen this already in the github interface you can hit fork and then it's going to make a complete copy of the repository now why would want to fork?

Well probably don't want to fork your own repo because you already have full access to all of the code here but you don't for other people's repos so find just a random repo on github and open it. I actually don't have access to change anything in this repo so I want to make a PR against this repo and request for my changes to be added or if I just to be able to branch off of the code in this repo and do whatever I want with it then I want to fork it so I'm going to hit the fork button here and you can see that the repo is no longer under the developedbyed's organisation, now it's under my personal github.
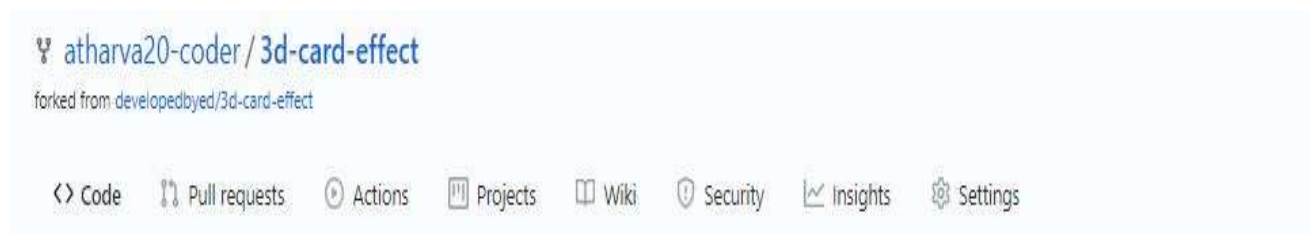
Now that I have forked this I have complete control to make any updates or changes that I want to this code.

THE END