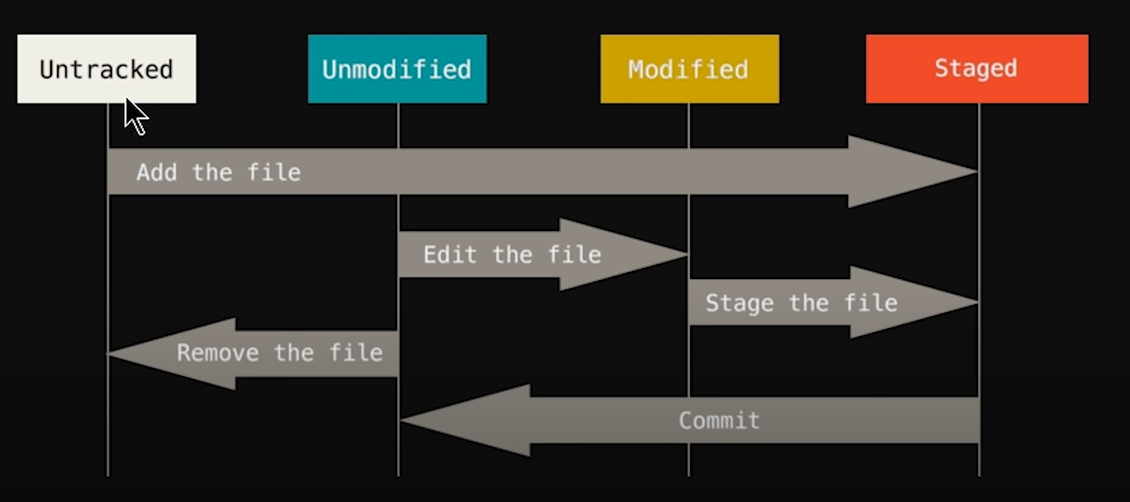
**Git and GitHub CheatSheet**



Commands:

1. ls ==> to list all my files in a specified folder... In a way shows a list of items or files or folders.

2. pwd ==> tells us the file or folder path

3. git config --global user.name [name] ==> tells our local git the name of the developer(need to be done once)

4. git config --global user.email [email id] ==> tells our local git the email id of the developer(need to be done once)

5. git config --global user.name ==> Shows us the developer name if it is prepared properly

6. git config --global user.email ==> Shows us the developer email if it is prepared properly

7. code . ==> to open the folder where git bash is run in vs code.

8. git init ==> Initializes empty Git repository in the specified folder

9. ls -lart ==> shows all the hidden folders and files.... After running the above command, a git file will also be shown in git bash.

10. git status ==> show the status of the files that is whether the files are untracked or staged or committed.

11 git add [file name] ==> takes the untracked file to the staging area... that is specified file is now added.

12. <Not preferable> git commit ==> On entering the mentioned command, a vim editor opens up. we have to click 'i' to enter commit message. After that we have to press (escape key+:+w+q) one after another to exit the vim editor.

13. touch [file name with extension mentioned] ==> Creates a blank file with the mentioned name.

14. git add -A ==> to add multiple files from untracked to staging area together.

15. git commit -m "[Commit message]" ==> shortcut of the previous git commit.... After making any change to file and adding it once again, this commit message informs any other developer what changes are performed in the already made file, for better understanding. Thus this commit message is very important.

16. git checkout [file name with extension] ==> helps in retrieving the file in our pc that accidentally got deleted or changed from the file saved in our git.

17. git checkout -f ==> If more than one files get accidentally removed or replaced, then this command retrieves all the files into our pc from our local git.

18. git log ==> tells us about what are the things that the developer has committed during the modifications, in the complete process of development.

19. clear ==> to clear the terminal.

20. git log -p -[number of commits one wants to see] ==> shows the number of commits one ones to see out of the whole list of commits. Then we have to press q to escape and return to the git bash terminal.

21. git diff ==> Compares working tree with staging area

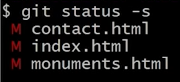
22. git diff --staged ==> Compares staging area with last commit

23. git commit -a -m "[Commit message]" ==> to directly commit my set of files in a specific folder without adding the files into staging area separately, and then committing.

24. git rm [file name with extension] ==> to remove the file mentioned both from working directory and staging area... in a way, completely deletes the files from the folder.

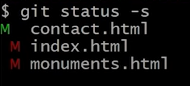
25. git rm --cached [file name with extension] ==> to remove the file mentioned from the staging area but not from the working directory.. in a way the file remains in the folder.

26. git status -s ==> to see the file modification in a shorter and more presentable way.



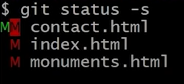
Here red M’s denote that the working directory is modified

After we add the first file in our staging area, and run {git status –s}, we find



It means that the first file is updated and added in staging area but not yet committed but the other files are modified in working directory, but neither added in staging area nor committed.

After modifying the above file once again but not adding it in staging area, we find



It means that the 1st file is modified both in working directory staging area but not yet committed.

27. touch .gitignore ==> forms a file with extension gitignore in vs code to form a collection of the files that needs to be ignored.

[If we mention a certain file name in .gitignore file with vs code, we will ignore the data of that file to be present in our git]



Saving the above text in .gitignore file, it will only ignore the mylogs.log file that is present in the main directory without considering such files in any sub directories.



On saving the above text in .gitignore file, it will ignore all log files within that particular folder.



In order to ignore a sub folder named ignore, we need to save the above text in .gitignore file, so that git ignores the mentioned folder; thus any changes made in the above folder will not affect git in any way.

28. git add feature1 ==> While development, if we want to make changes to any specific part of a file or folder, without affecting the main folder which might be a website running, then we run the mentioned command , to add a ‘feature1’ named branch along with the main running ‘master’ branch.

29. git branch ==> shows how many branches are present in our git tree, and in which branch we are currently present.

30. git checkout feature1 ==> to switch from master branch to another branch named feature1.

Now on running git status command, we will get information about the present condition of feature1 branch.

31. git checkout master ==> If we want to switch to master from feature1 even after committing faeture1, we run the mentioned command to retrace back to our master branch.

32. git checkout faeture1 ==> to re-enter feature1 branch from master branch.

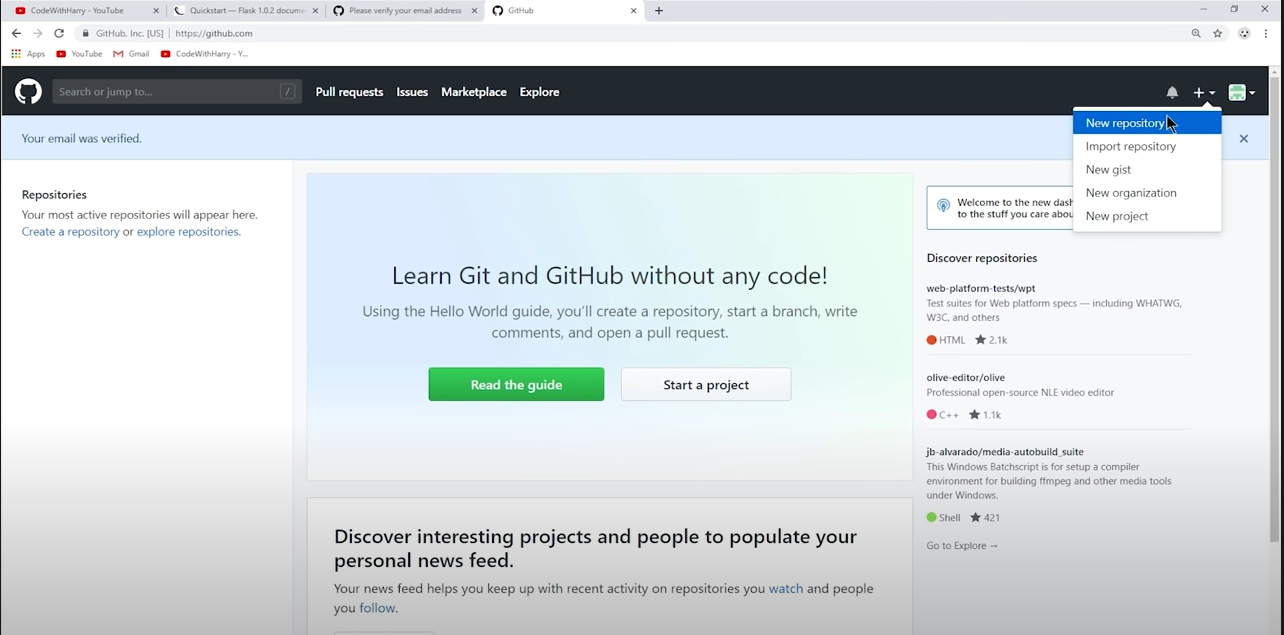
33. git merge faeture1 ==> to merge faeture1 branch to master branch, after making all required modifications in feature1 branch.

34. git checkout –b [new branch name] ==> makes a new branch with the specified branch name and simultaneously switch into that branch for working and modification.

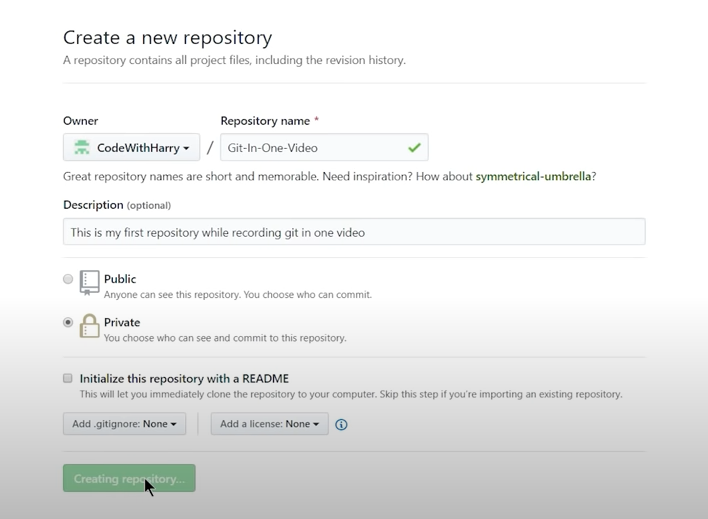
GitHub ==> Used to host the local git made into systems into an open source platform.

**Making a new GitHub Repository procedure:**

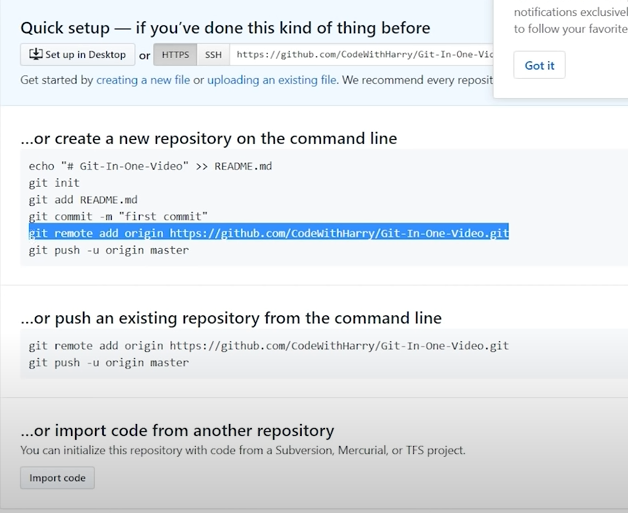
1. Go to ‘+’ and press New Repository option.



1. Fill all necessary information as shown to create a new repository. Here it is shown to make a private repository that only developer can view, to make it public, just click public option.



1. In the next tab, copy the highlighted portion as it appears for our GitHub accounts, to link our local git to global. Thus the short name of the below mentioned url will turn out to be origin.



1. Then paste the above highlighted portion in your git terminal and press Enter; like this :



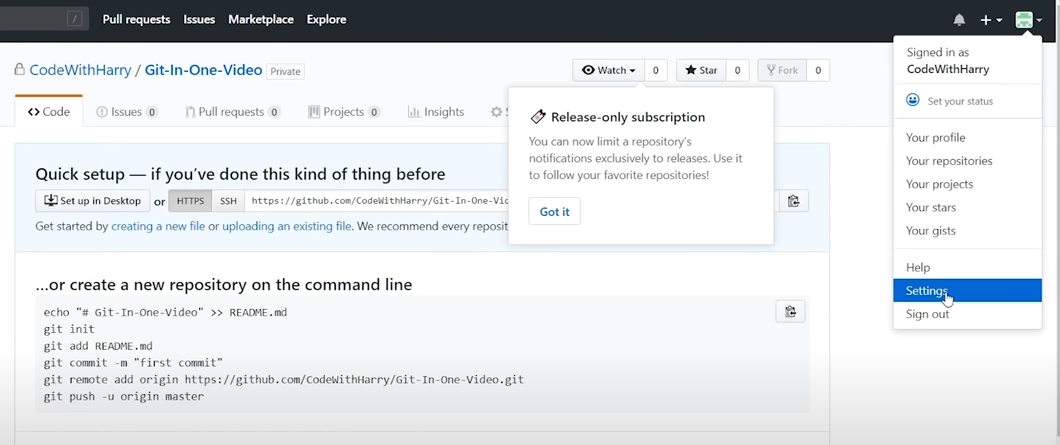
35. git remote ==> to show name of the remote present; that in our case will be origin[short name of the copied url]

36. git remote –v ==> shows the url from where we will fetch and push any global git or our local git.

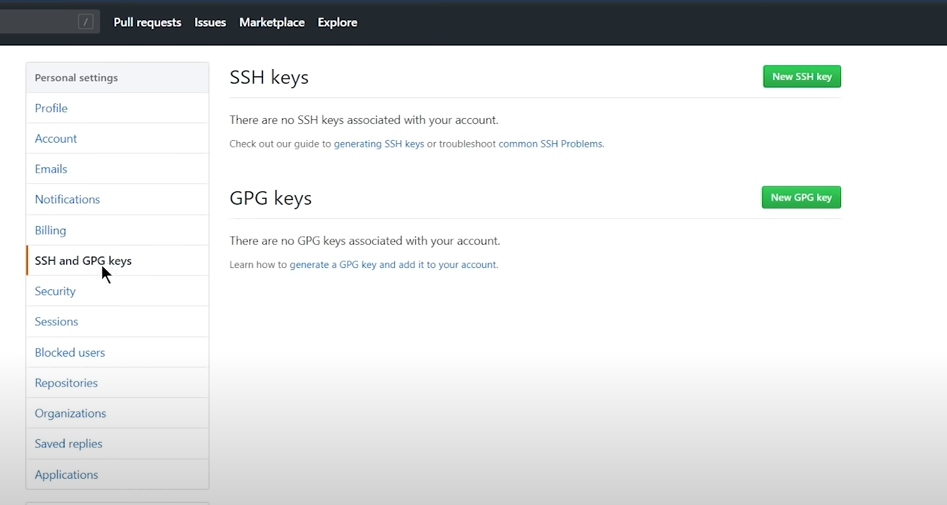
37. git push origin master ==> to push our master branch to origin, i.e. into our GitHub repository.

**To access read and write access from repository**

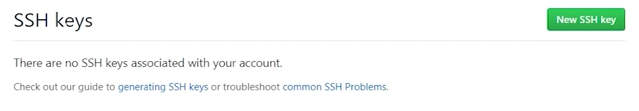
1. Go to settings.



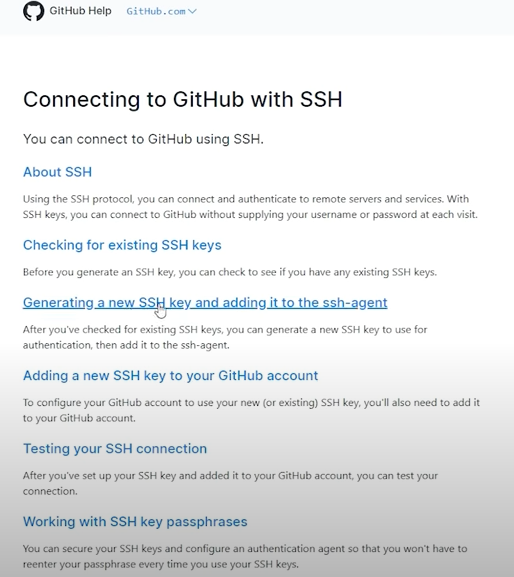
1. Go to SSH and GPG keys.



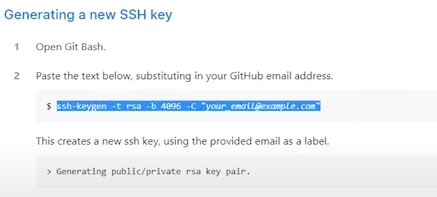
1. Click on generating SSH key, so that all work done on our git bash of pc, get updated into our GitHub account.



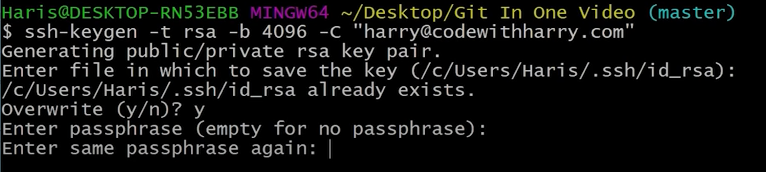
1. Click on Generating a new SSH key and adding it to the SSH agent.



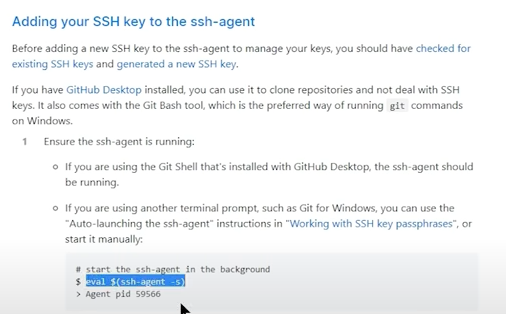
1. Then we have to paste the text in our git bash terminal (shown below) mentioned to generate SSH key with by substituting it with our GitHub email address.



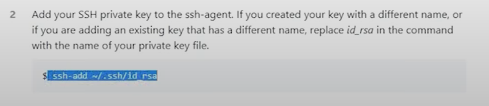
1. After pasting, we have to press enter and if you are overwriting a generated key, then type ‘y’ to overwrite as shown below. Leave the enter paraphrase tab empty, by pressing enter.



1. Paste the eval key command in your git bash and run it, to deploy your key.

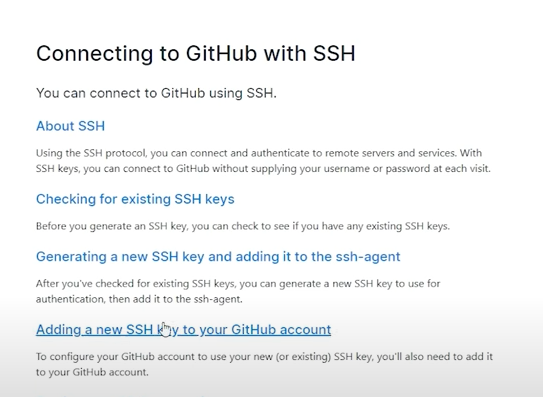


1. Finally copy paste and run the generating SSH key command in your git bash to complete the process of generating SSH key to deploy code to GitHub.



Thus procedure of adding SSH private key to SSH agent is hence completed.

1. Then go to Connecting to GitHub with SSH page and press the Adding New SSH key to GitHub account tab.



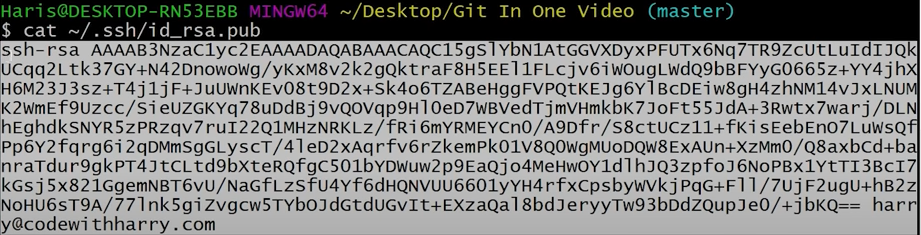
1. Then we have to just copy the content of your public key, as shown below.



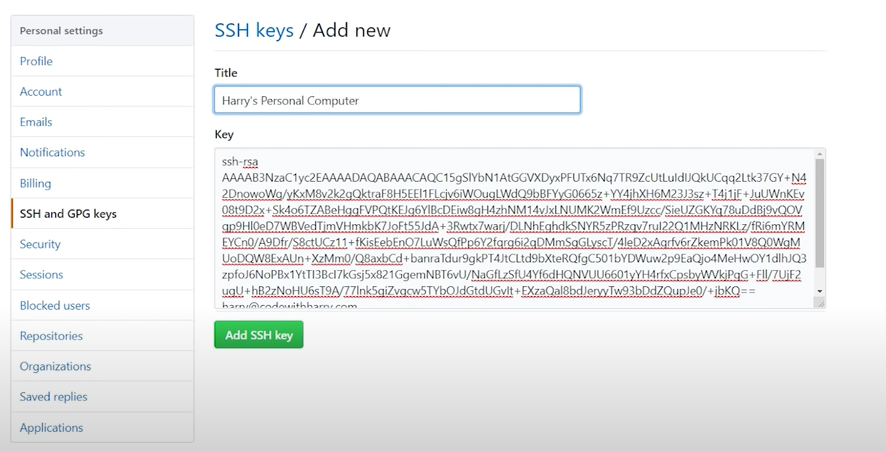
1. Write cat followed by the public key copied. The cat command basically shows the content of the SSH key.



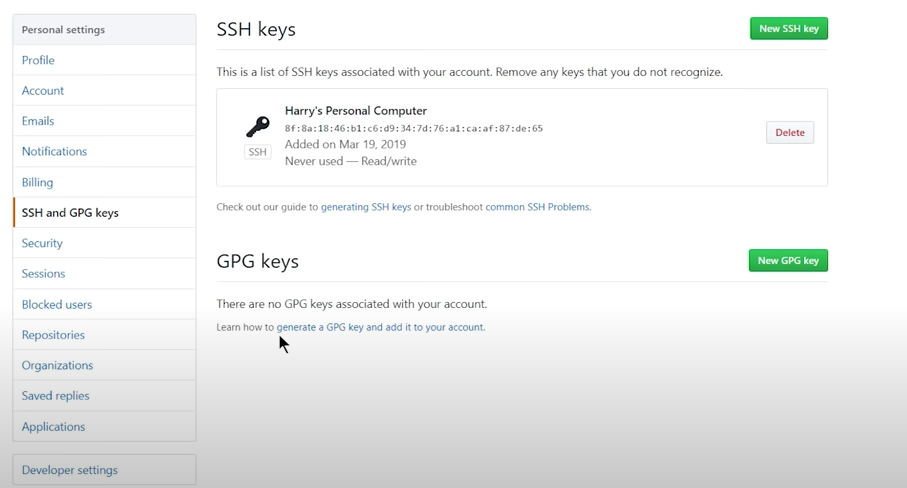
1. Then we have to just copy the content of the key as shown.



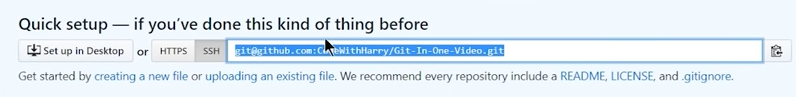
1. Then go to SSH key adding page and paste the content of the key and give a title. The title should be specific as it tells which pc has been used by developer to deploy his code to GitHub.



1. Finally after pressing “Add SSH key “button, we will get something like this.

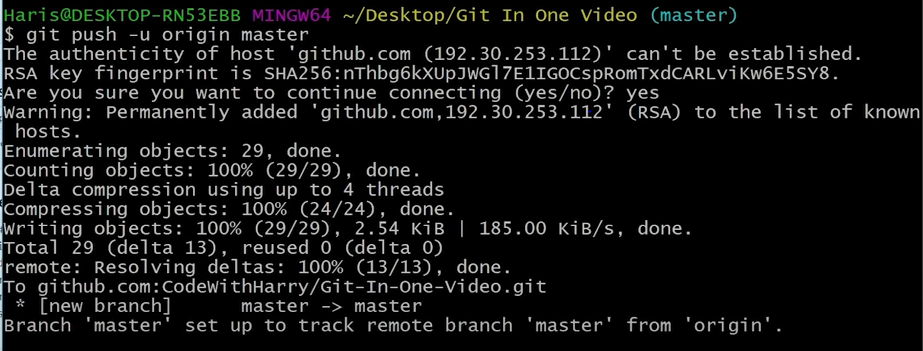


1. Then copy the SSH url and run the set url command as shown below to change the url to origin.

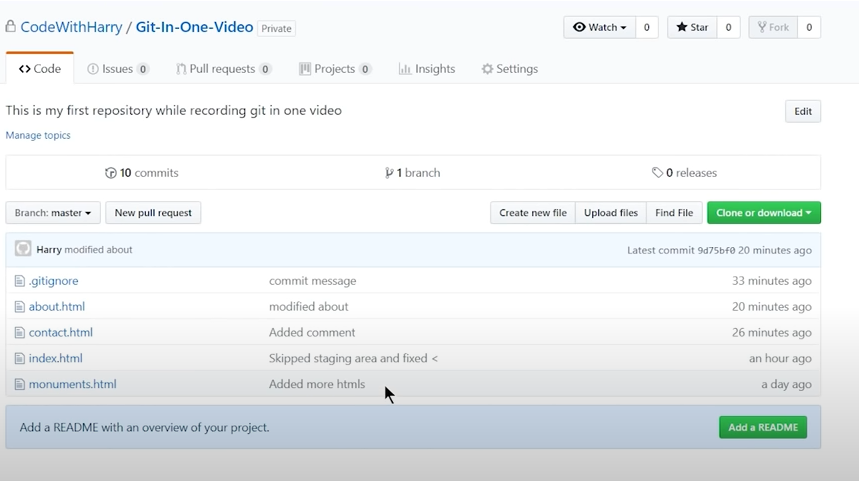




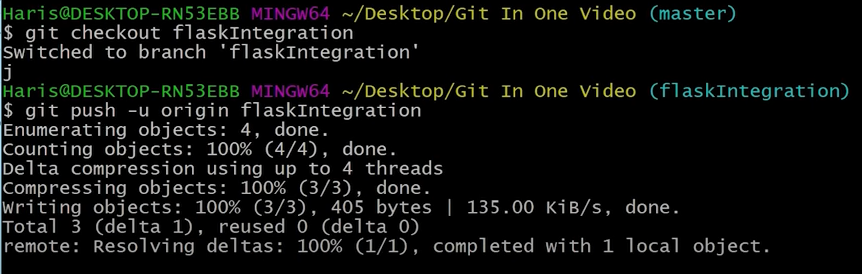
38. git push -u origin master ==> to push the master branch to GitHub after setting the SSH key.



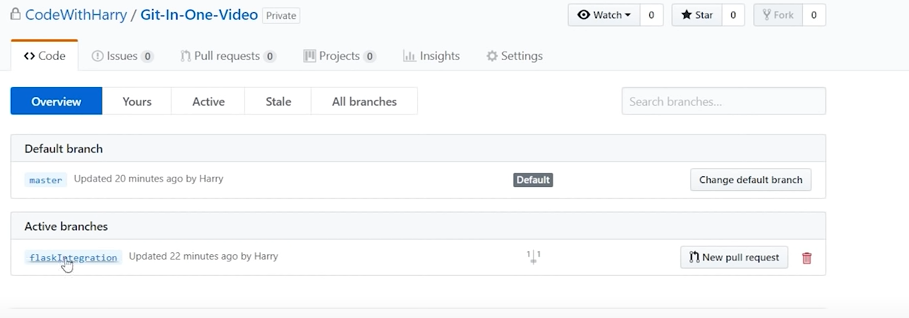
After reloading our GitHub page, we find that our code has been deployed like shown.



In order to push another branch, we have to checkout that branch and push it.

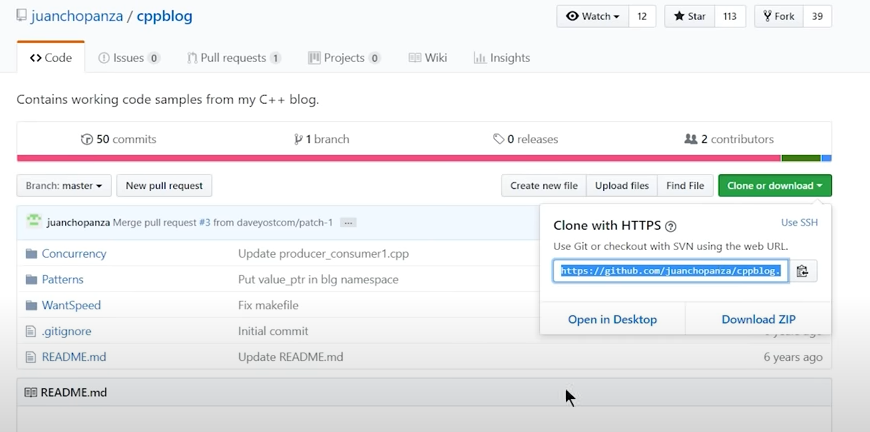


Thus two branches have appeared.



39. git push ==> after running the above command, it will push to the same branch as we last pushed with -u command.

To clone any GitHub repository into our pc, we need to open that GitHub repository and copy the url as shown below.



40. git clone [url] [folder name to clone] ==> This will clone the specified repository into the specified folder. If no folder name is mentioned, it will create a folder with the name of the repository and clone the repository in it.