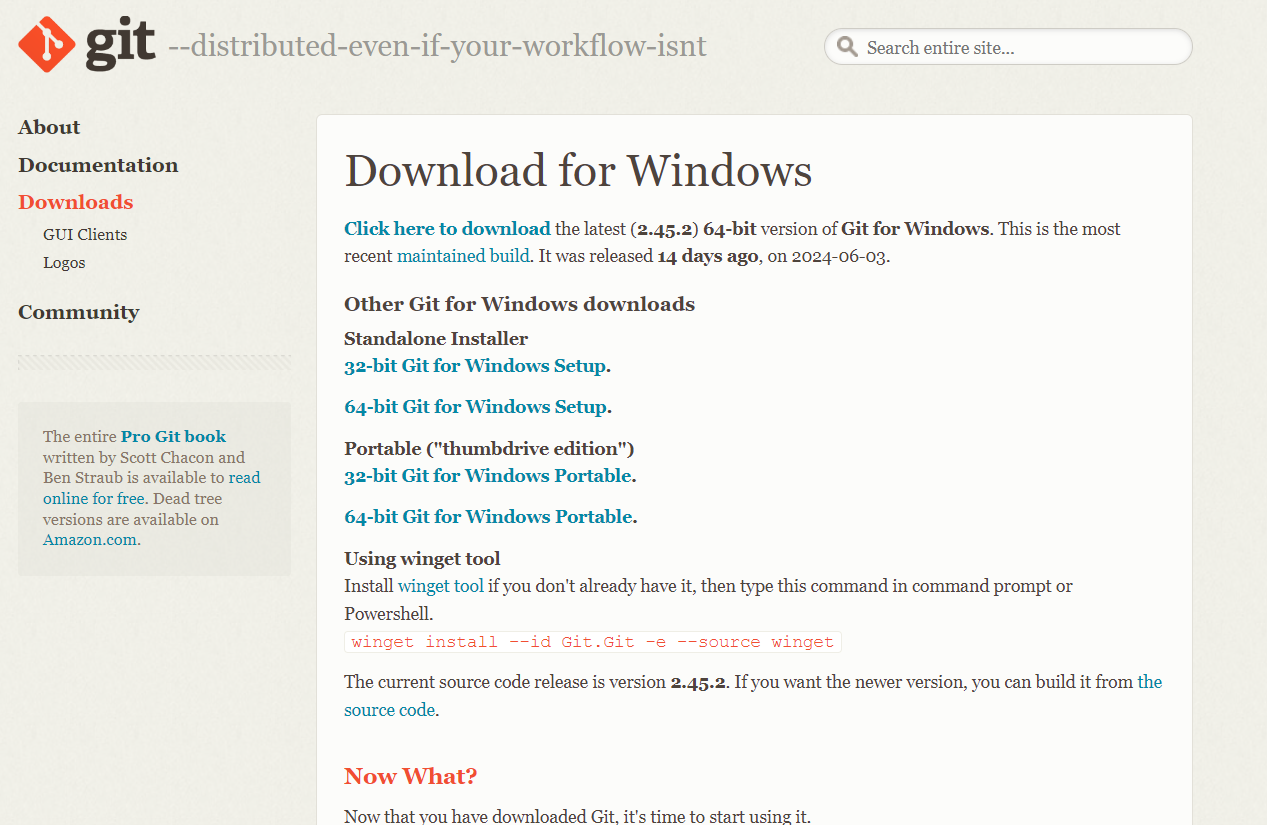
1. **Download git.**

<https://git-scm.com/download/win>



* **Choosing the git standalone installer**
* **Setup the pin and authenticator**
* **Click on .exe file and Use next-next and finish the git installation**

**Open git bash terminal from windows:**

(can be used as a general terminal as well like for run python command and all)

* Create user name and email

*git config --global user.name siddesh*

*git config --global user.email desaisiddesh223*

* check the user name and email values

*git config --global user.name*

*git config --global user.email*

**Two approaches in git first one is init and second is cloning:**

1. **Init:**

* Going inside the folder named “git\_learning” using terminal

Command: *git init*

creates the .git named folder

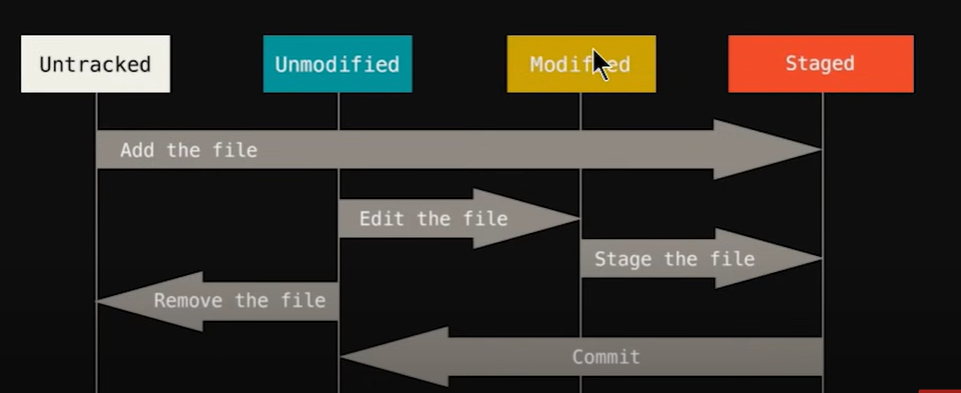
this can be seen using the command else simple *ls* won’t show it

Command: *ls -lart*

* Check the status

Command: *git status*

(provide the all file’s status like untracked, modified, staged or not commited)



Keeping the file at the staging area:

Command: *git add index.html*

*(only selected file will be added)*

committing the file

1. With vim editor

Command: *git commit*

*opens a vim editor*

*press I to type a text*

*type initial commit (it’s a message of file, description in short)*

*press esc then :wq*

1. Without vim editor:

Committing the changes with – m command as it will not require the vim editor changes

Command: git commit -m 'added the text in about.html'

*check the status again:*

Command: *git status*

*(keep on using it)*

* Case when additional new file is created

creating a file using git bash terminal

Command: *touch filename*

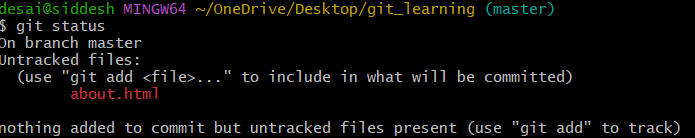
creating the about.html file

touch about.html

now check status

:

Command: *git status*

**

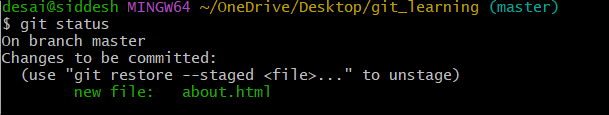
Adding multiple new additional files into staging area

Command: *git add -A*

now check status

:

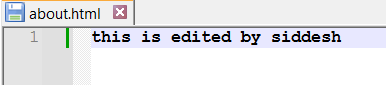
Command: *git status*

**

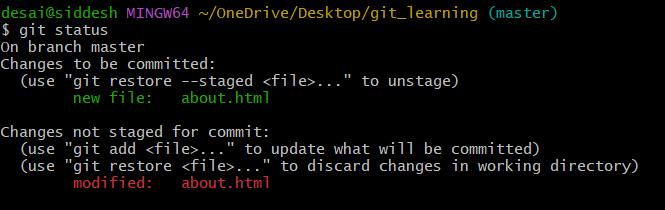
Shows the new file about.html.

* Case when file in staging and later also modified in the system

about.html modification



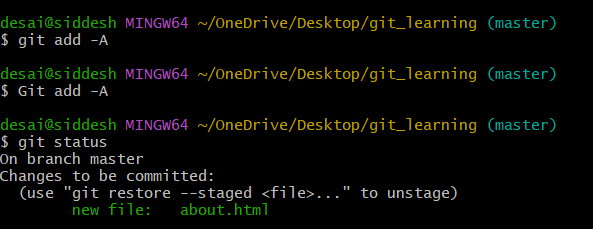
Shows the new file staged not committed and modified but not staged files



Now add it to the staging area

Command: git add -A

Check status: Git status



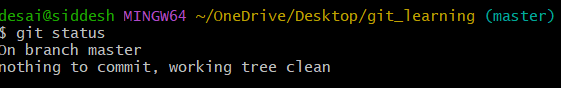
Now status of modified is not there as it is staged. Only new file status in staged is present.

Committing the changes with – m command as it will not require the vim editor changes

Command: git commit -m 'added the text in about.html'

Now check the status

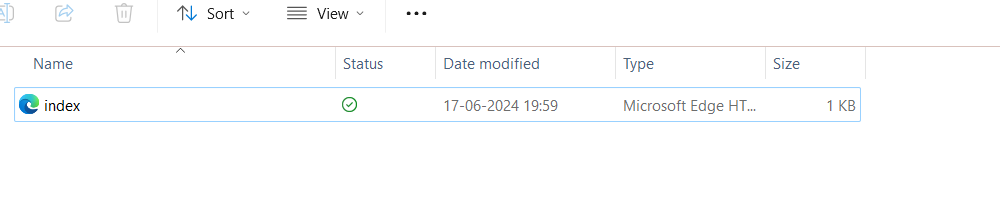
Command: git status



*File recovery* when the accidently delete the file about.html or deleted the content of it

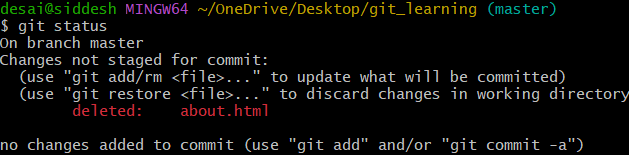
* Case Single file edited

Deleted the file about.html



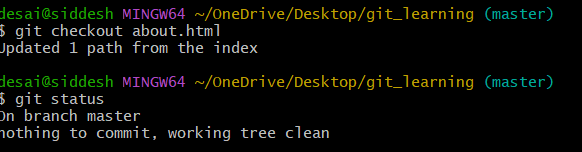
Check status:

*Command: git status*

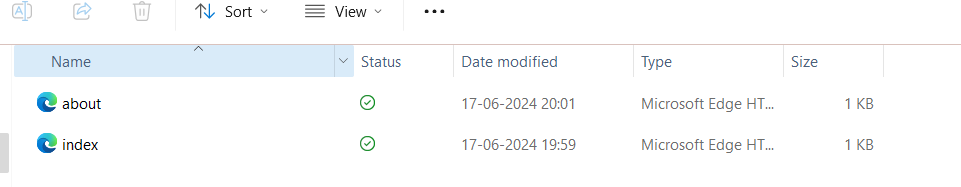


*Command: git checkout about.html*

(matches with the last commit to the given file name)



File recovered

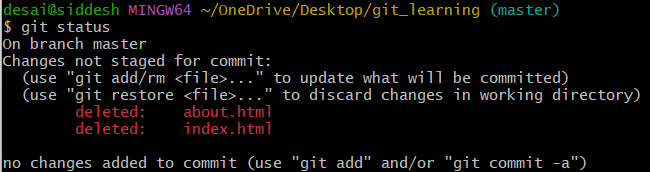


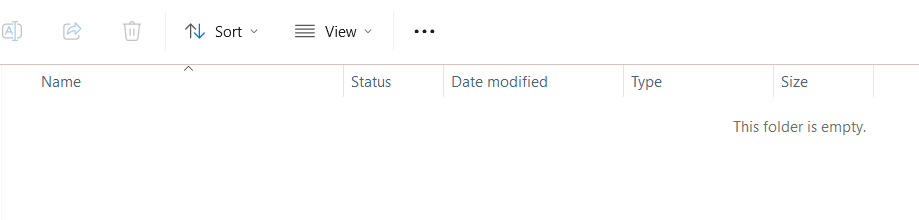
* Multiple file editing

To do this deleting all the files inside the folder

Check status

*Command: git*

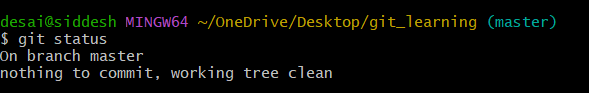




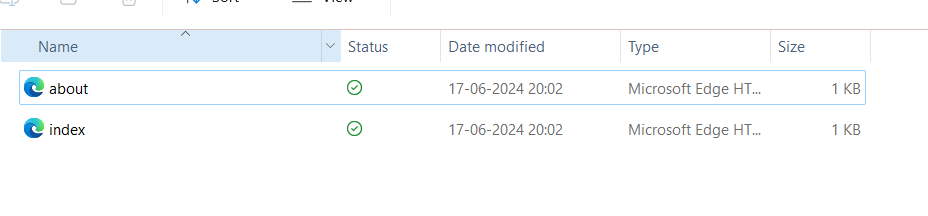
*Command: git checkout -f*

Check status

*Command: git status*

**

Both the files are recovered

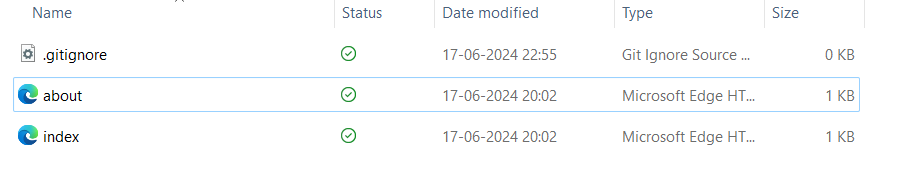


Ignore file

In windows just create the .gitignore file

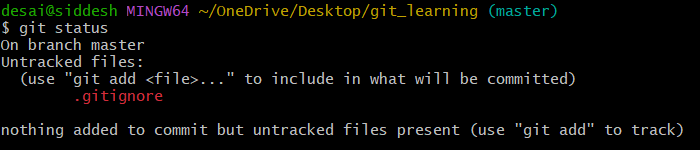
Command: touch .gitignore

Now the .gitignore file is created



.gitignore file add the file which are unwanted to track or need backup or which can be regenerated like log, images as it will sloq down the transfer of file in push pull stage for git hub.

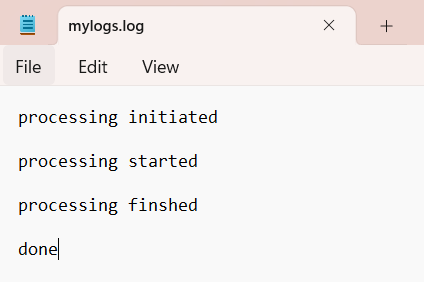
Command: git status



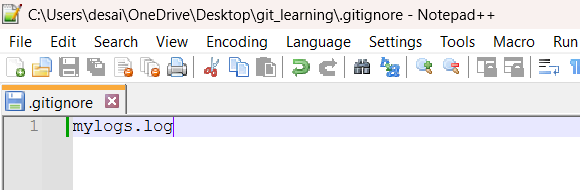
* Creating the log file for demonstrating the usage of .gitignore file

Command: touch mylogs.log

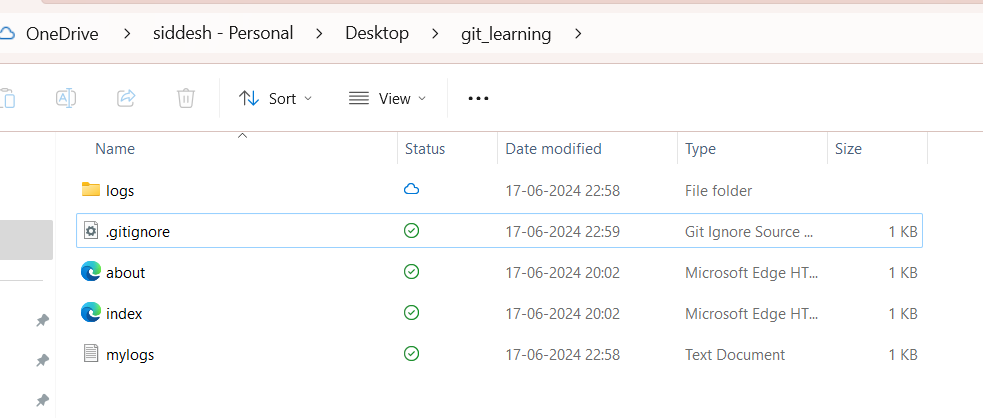
Adiding some random things in mylogs.log file

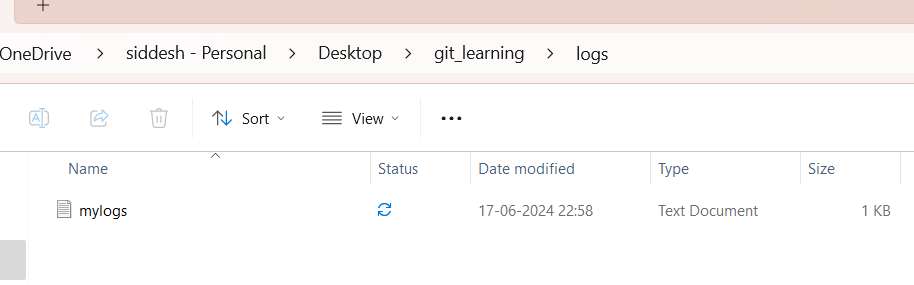


Now add this file name in .gitignore



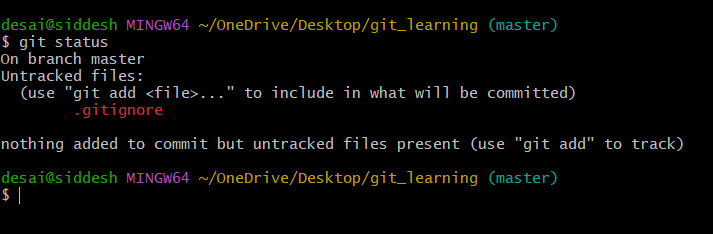
So it will ignore committing the mylogs.log file wherever it is present. Even if it is inside some folder. For example logs folder is created and mylogs.log file is also created within it.





Now check the status:

Command : git status

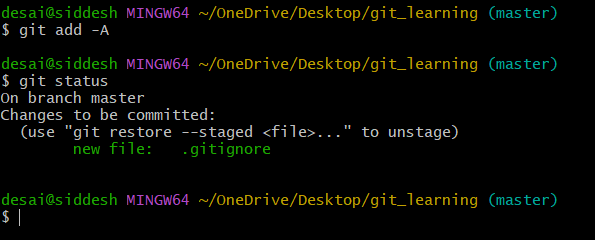


It is showing that all the three new added contents are not tracked

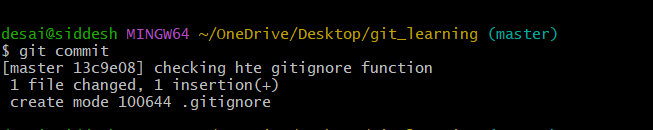
Now adding the files to staging and the nchecking the status

Command : git add -A

Command : git status

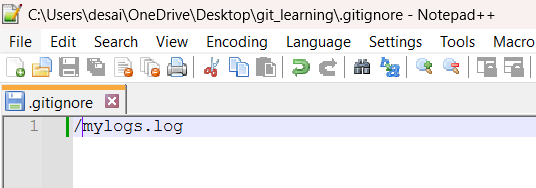


Check the files mylogs.log and log folders are ignored to commit and stage



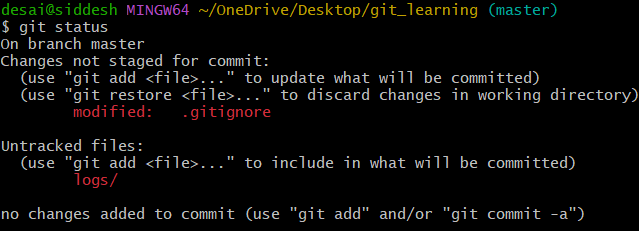
Now if we want to gitignore the mylogs file inside the root directory only then use

/mylogs.log



Now check git status:

Command: git status

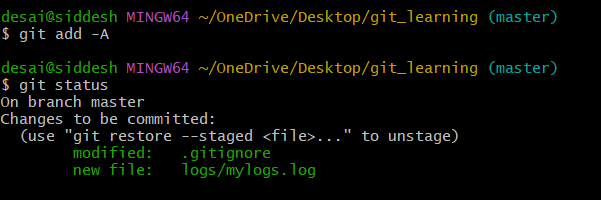


Now you can see logs folder is getting noticed.

Adding to the stages and checking the status

Command: git add -A

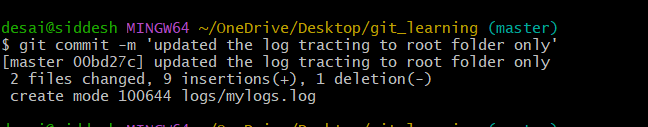
Command: git status

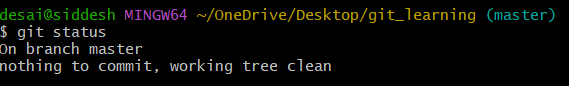


You can see now the new log folder is added and modified .gitignore file is also added.

Command:

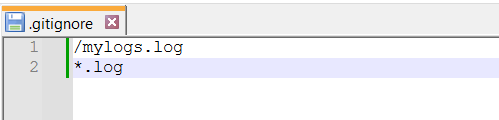
git commit -m 'updated the log tracting to root folder only'





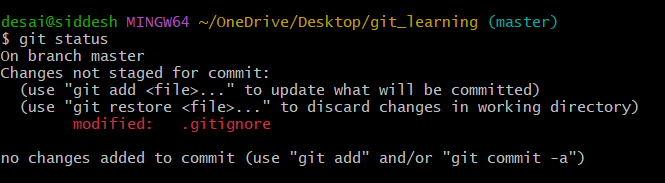
Command: git status to check the staus.

All file having .log extention name to be ignore.



Check git status

Command: git status

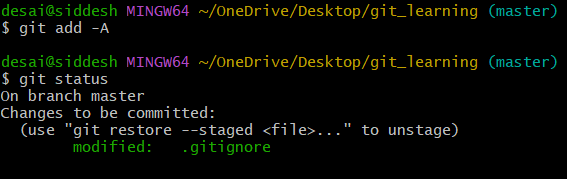


Adding to stages

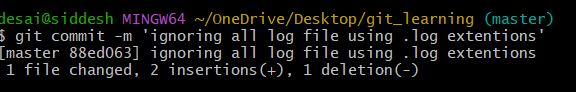
Command: git add -A

Check status

Command: git status



git commit -m 'ignoring all log file using .log extentions'



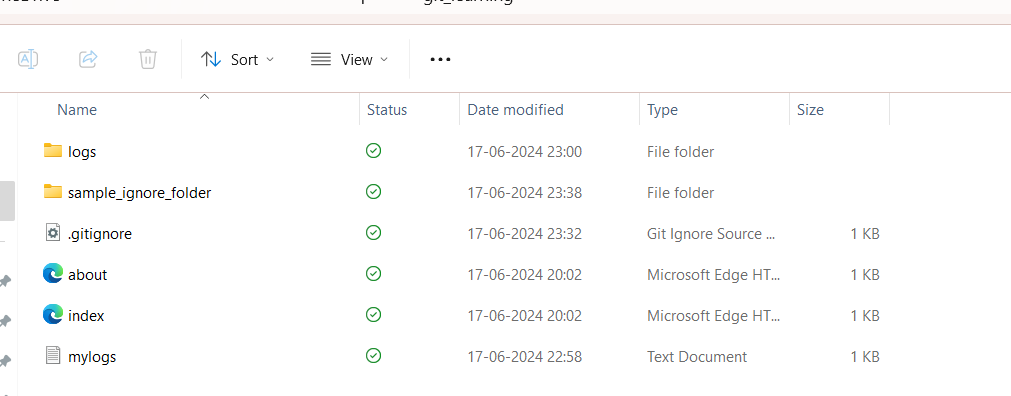
Similarly one can try for \*.py, \*.cpp and so on….

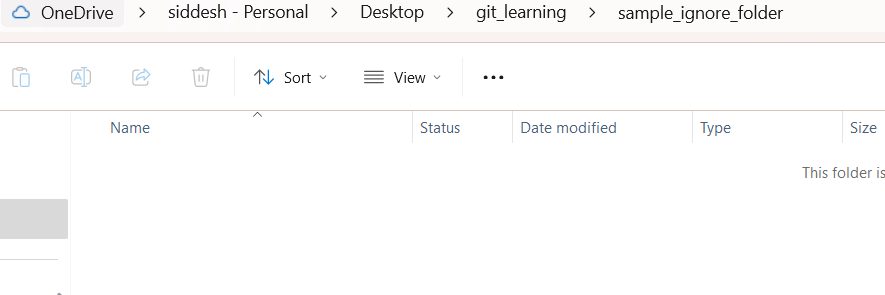
Now ignoring a folder

Then use folder\_name/ in the .gitignore

Example:

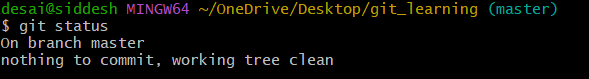
To demonstrated it an empty sample\_ignore\_folder is created





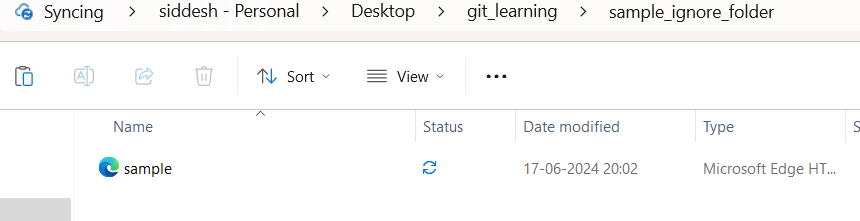
Now check status:

Command: git status



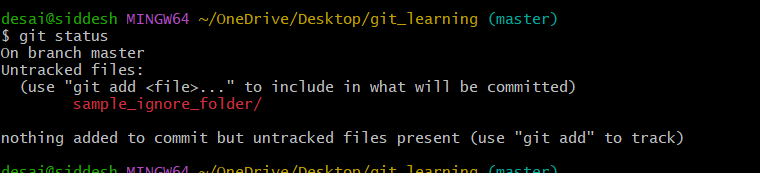
It is not showing anything since the sample\_ignore\_folder is empty

Now add a sample.html file



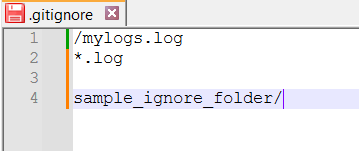
Now checking the status.

Command: git status.



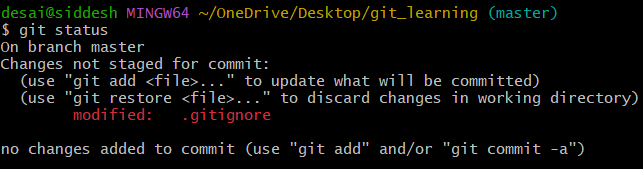
Now it is showing the untracted folder.

To ignore the sample\_ignore\_folder then add it .gitignore file.



Now check status:

Command : git status



Now tracting of sample\_ignore\_folder is vanished and modified .gitignore is shoqwn

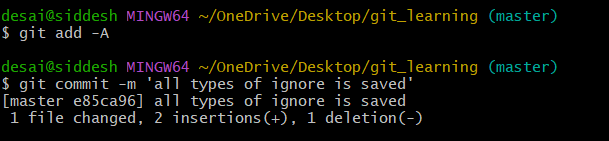
. Now commit it to save the changes.

Add staging

Command: git add -A

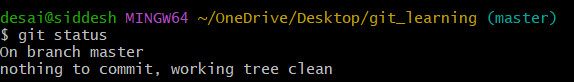
Commit Command:

git commit -m 'all types of ignore is saved'



Branches in git:

Master branch is by default branch. It is the main branch where all the work is done.



Creating a branch.

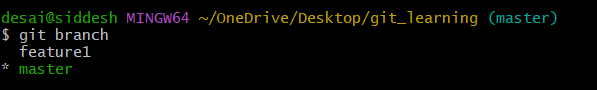
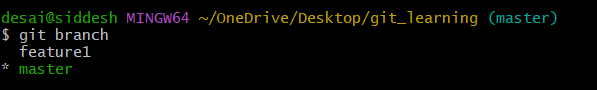
Command: git branch <branch\_name>

Command: git branch feature1

It creates a new branch where additional work can be done and then can be merged with the master branch as per the need.

Now check all the branches.

Command: git branch



(master) is showing current active branch.

feature1 and master are the git branches.

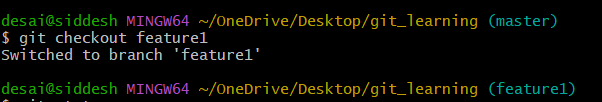
green colour in \*master indicates that it’s a current active branch in the terminal.

* switching the branches in the terminal

Command syntax: git checkout <branch\_name>

Switching to feature1

Command: git checkout feature1

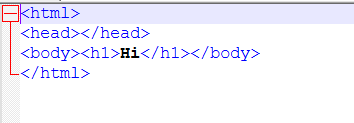


Now it is switched to feature1 one branch.

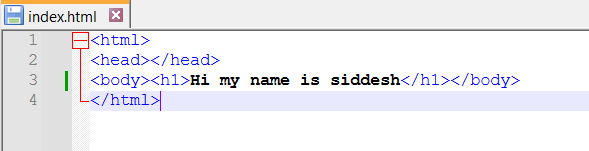
Merging feature 1 to master branch.

Now modifying the file named index.html

Old file:



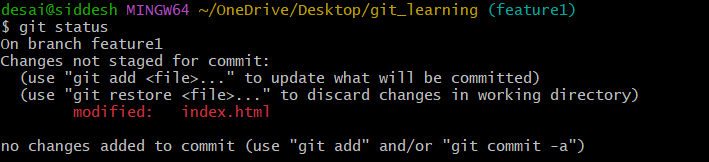
Modified file:



This modification is done in the branch named feature1.

Now check the status:

Command: git status



You can see the modification is seen in the feature1 on for the index.html file

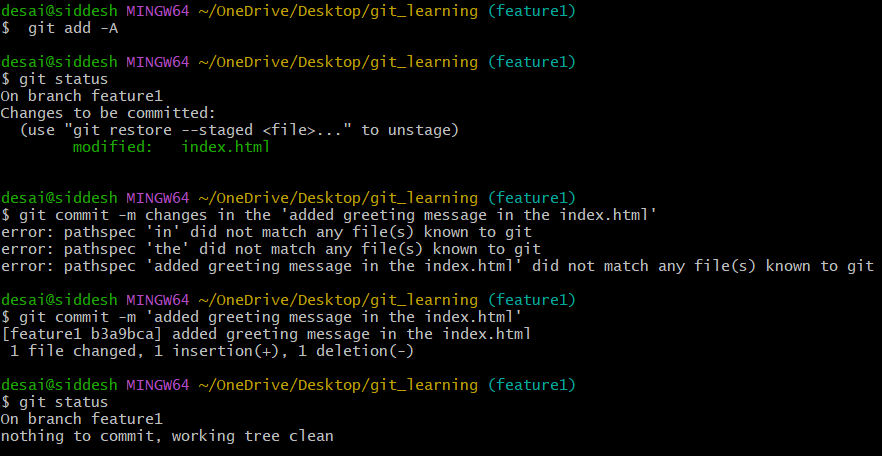
Now do the stagging and commit the changes.

Command: git add -A

Command: git status

Command: git commit -m 'added greeting message in the index.html'

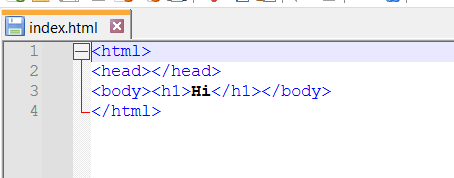
Command git status



Now if you want to rollback the previous version where the greeting messaage was only ‘hi’ in the index.html file

Command: git checkout master

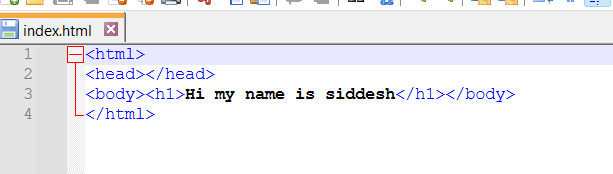
Now open the index.html file



You can see the old file which was committed to master branch is present now.

Now switch to feature1 branch again

Command: git checkout feature1



See that feature1 branch has full details of greeting. In short each branch has its own version of files.

Which can be switched at any time.

Now try to merge the feature1 files to master branch.

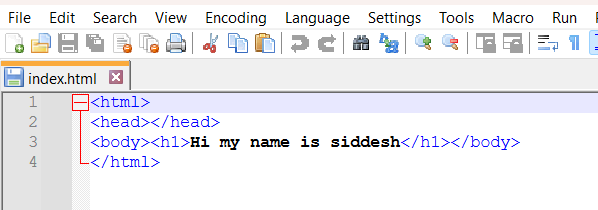
Open master branch

Command: git checkout master

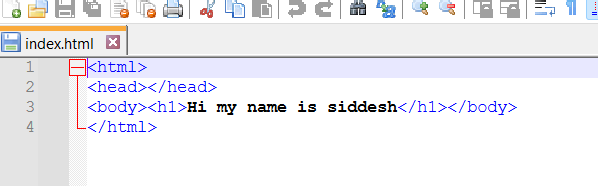
Now Merge feature 1 with master

Command: git merge feature1

Now check both the branches file



1. feature 1 branch.



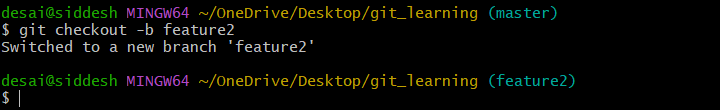
So after merging feature 1 to master branch both the files are same.

Note You can create as many branches as possible.

Commnad to create the new branch and open the new branch

Command syntax: git checkout -b <new\_branch\_name>

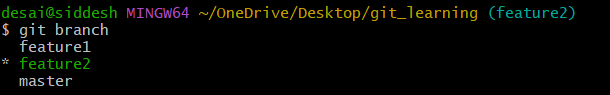
Command: git checkout -b feature2



You can see now the current working branch is now the feature2

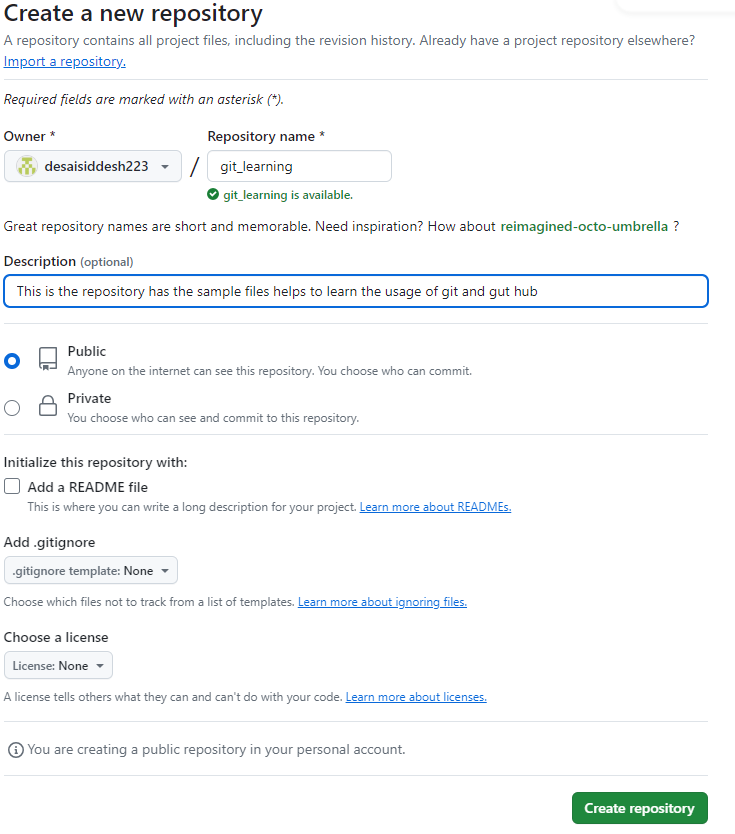
Check branches now.

Command: git branch

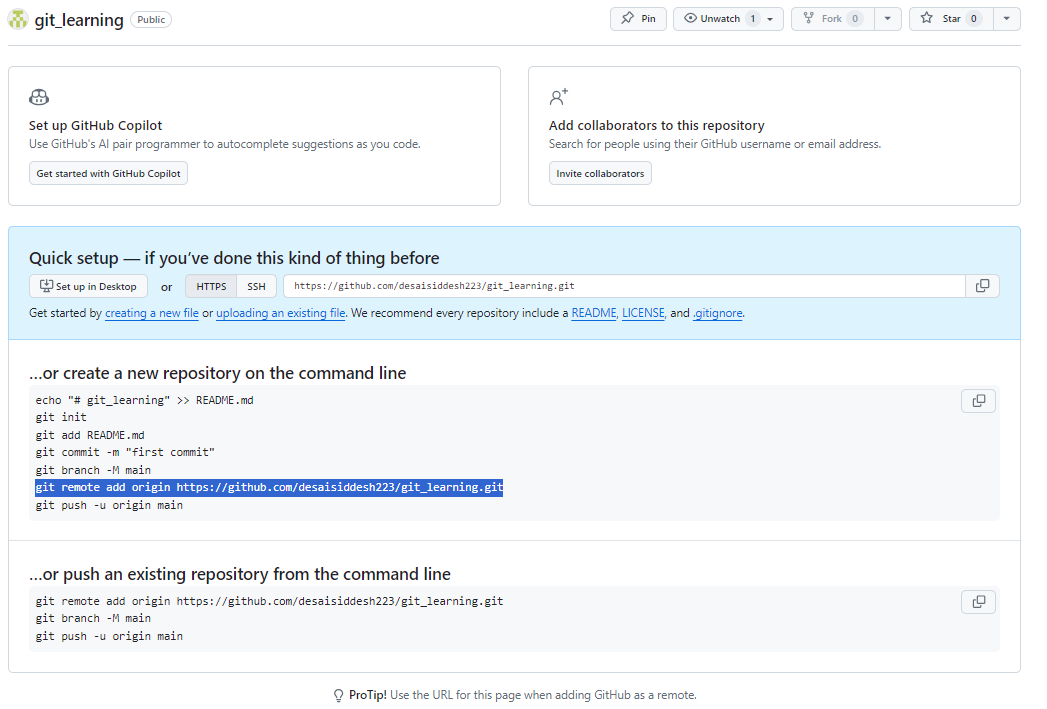


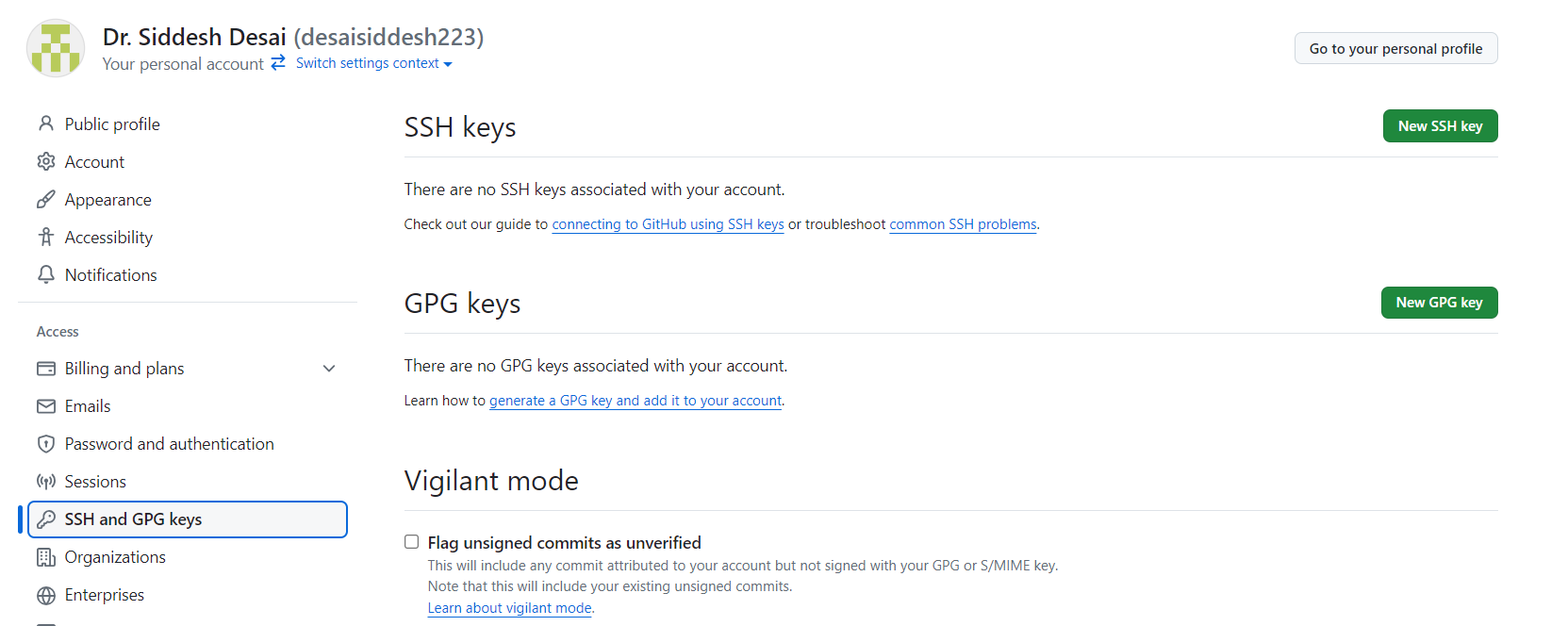
Git hub:-

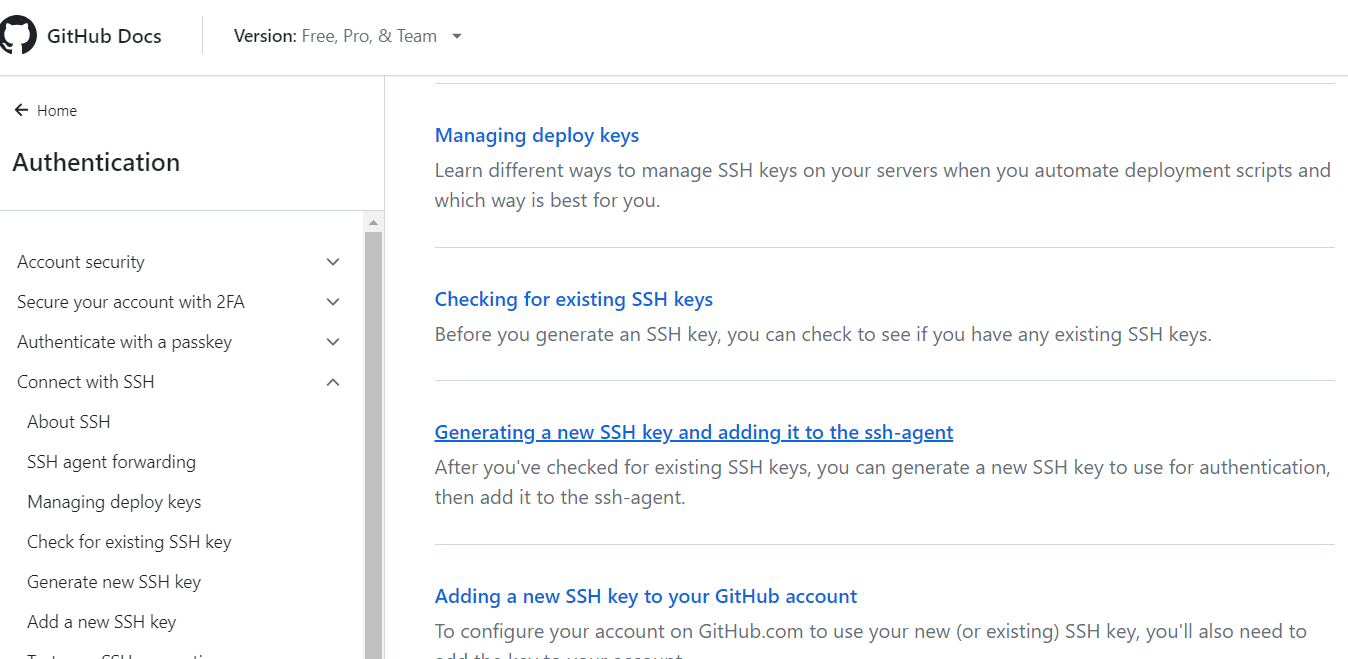
Create your account at <https://github.com>.



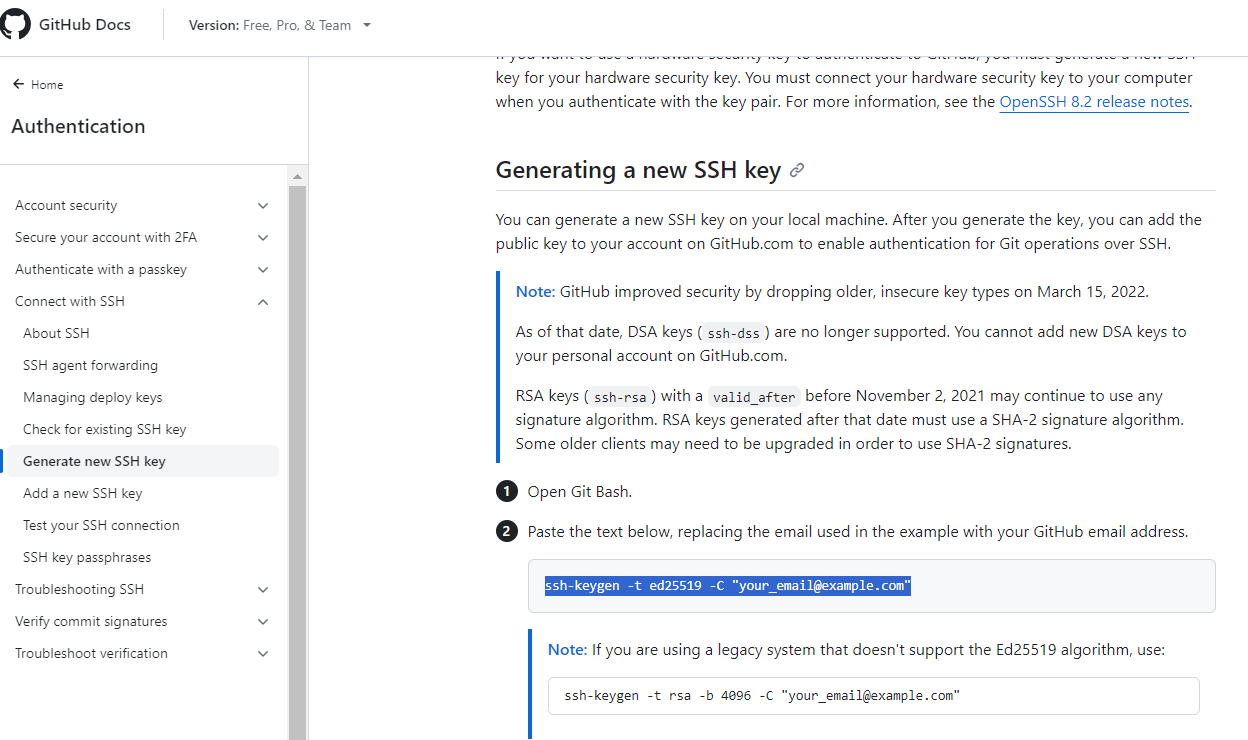
We created a remote repository and our local repository which is created in our local system need to be pushed or need to be copied in the gut hub.







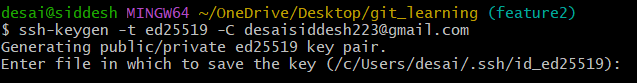
## Generating a new SSH key



Command Syntax: ssh-keygen -t ed25519 -C [your\_email@example.com](mailto:your_email@example.com)

Command: ssh-keygen -t ed25519 -C [desaisiddesh223@gmail.com](mailto:desaisiddesh223@gmail.com)

Use above command in the git bash terminal.



Need to provide the github account to local system. For that wee need SSH keys.

The key's randomart image is:

+--[ED25519 256]--+

| |

| |

| . |

| . o . .|

| \* . S o .=|

| = = . + = .E+|

| . B o o + o.+\*+|

| + \* o . + o+\*=|

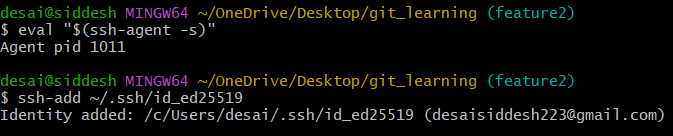
| .= + .\*ooo+.=|

+----[SHA256]-----+

## Adding your SSH key to the ssh-agent

## Command: eval "$(ssh-agent -s)"

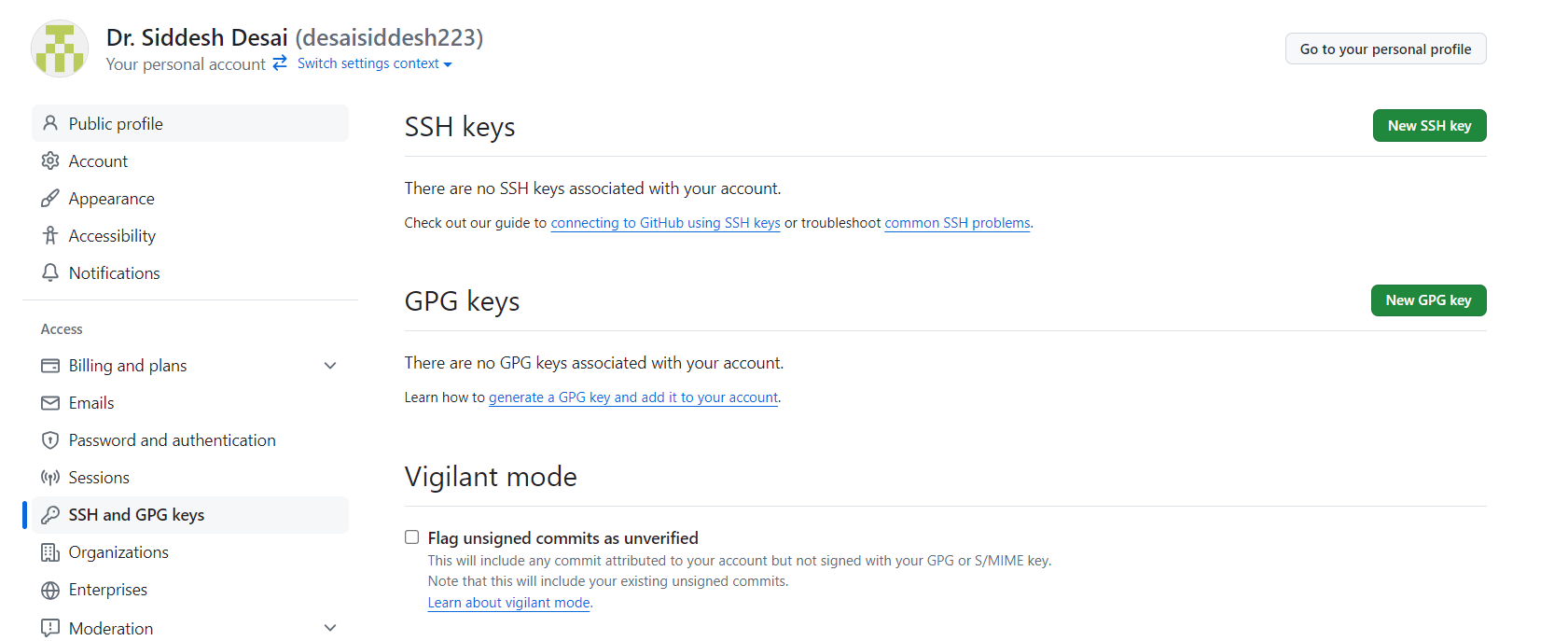
## Command: ssh-add ~/.ssh/id\_ed25519

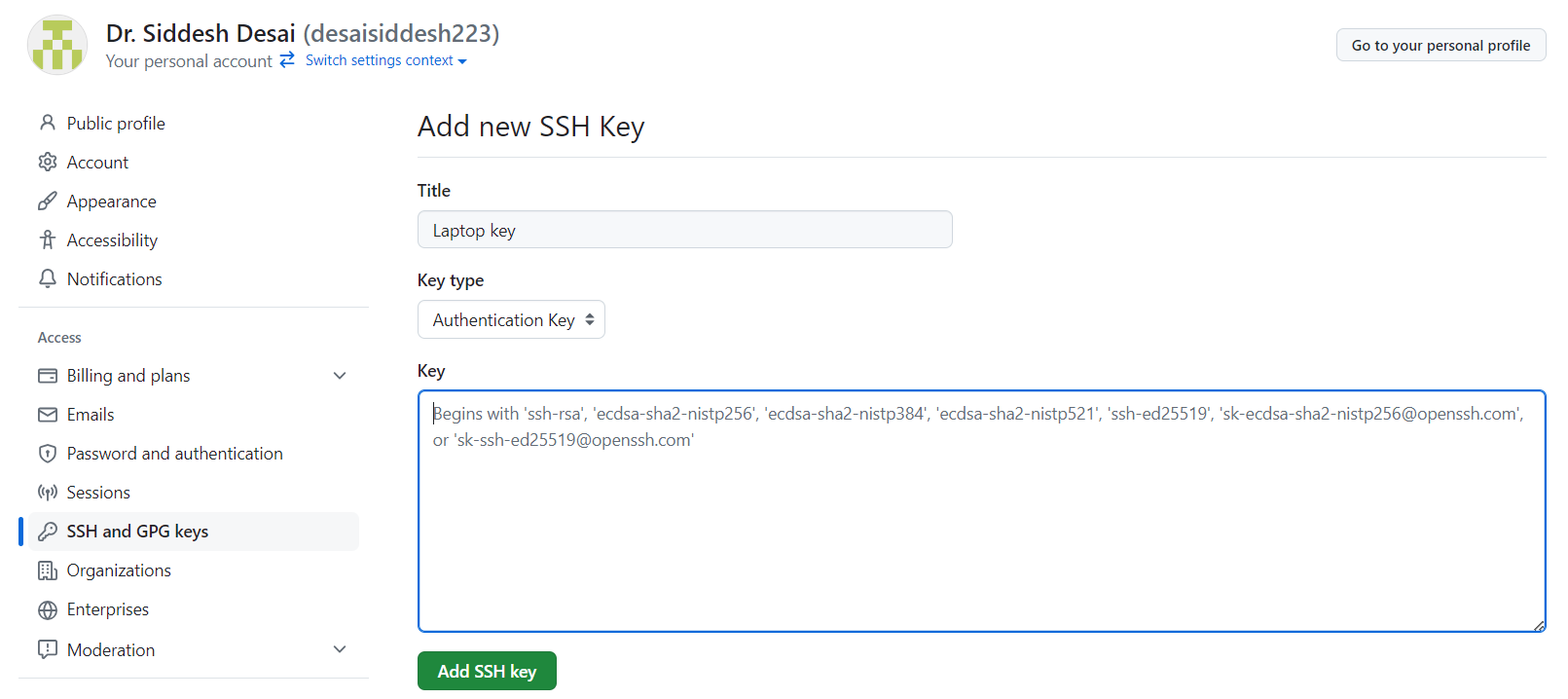


Add the SSH Key to Your GitHub Account

Command: cat ~/.ssh/id\_ed25519.pub

ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBi4lp0T31rWZn1JDfshBuUUJeV7gSpCNlWFlyh9wIa0 desaisiddesh223@gmail.com

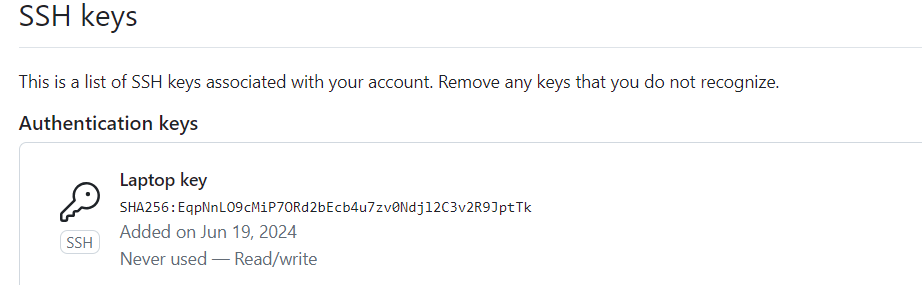




Paste your keys here and then click

add ssh key

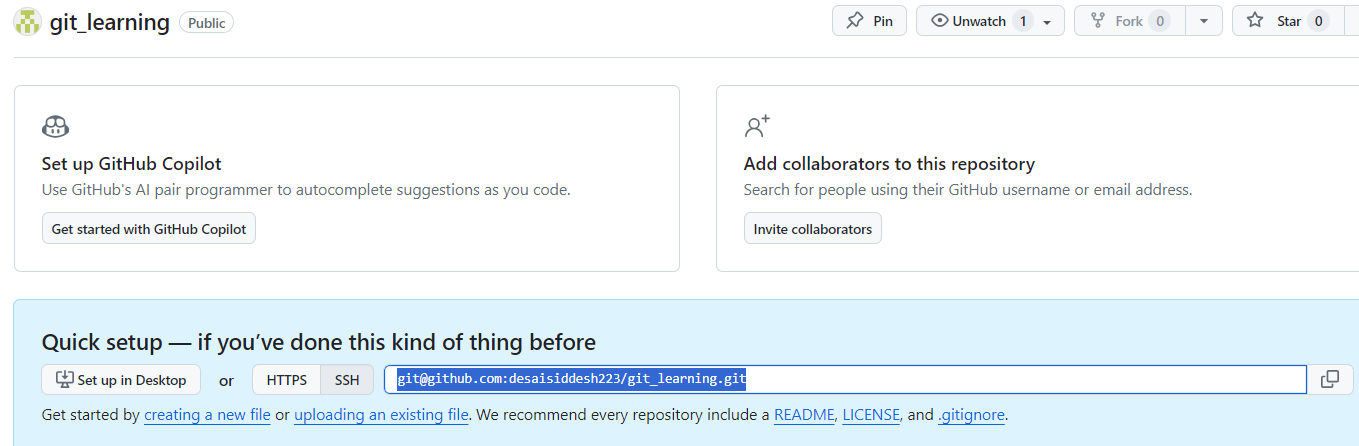
After adding required authentication details. This page confirm key addition.



Now got to git bash terminal. Use Command:

Command: git remote add origin [git@github.com:<username>/<repo](mailto:git@github.com:%3cusername%3e/%3crepo) name>.git

Command: git remote set-url origin <ssh url for repository>



So the final commands are

Command: git remote add origin git@github.com:desaisiddesh223/git\_learning.git

Command: git remote set-url origin [git@github.com:desaisiddesh223/git\_learning.git](mailto:git@github.com:desaisiddesh223/git_learning.git)

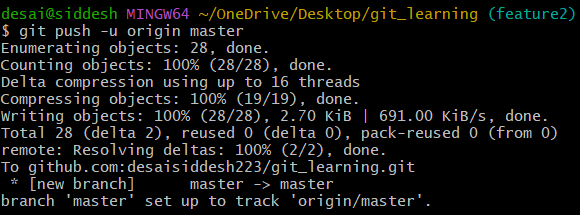
Now check remote:

Command: git remote

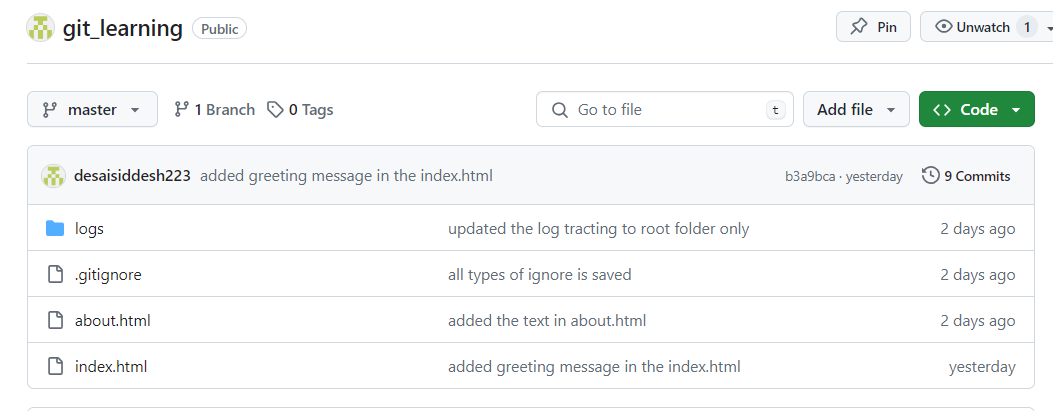


Push the changes in master branch to the remote repository:

Command: git push -u origin master

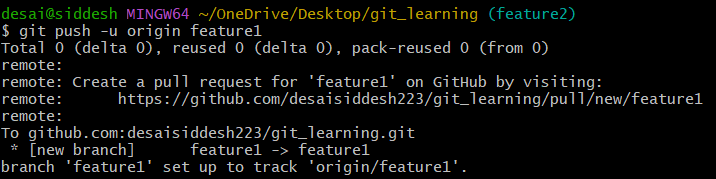


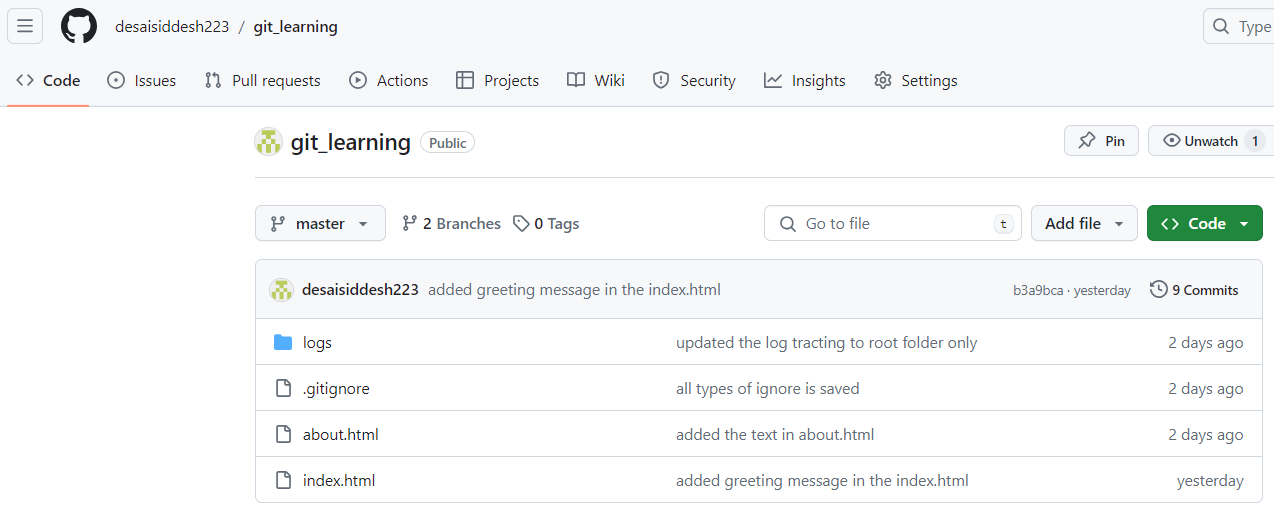
Now check the git hub.

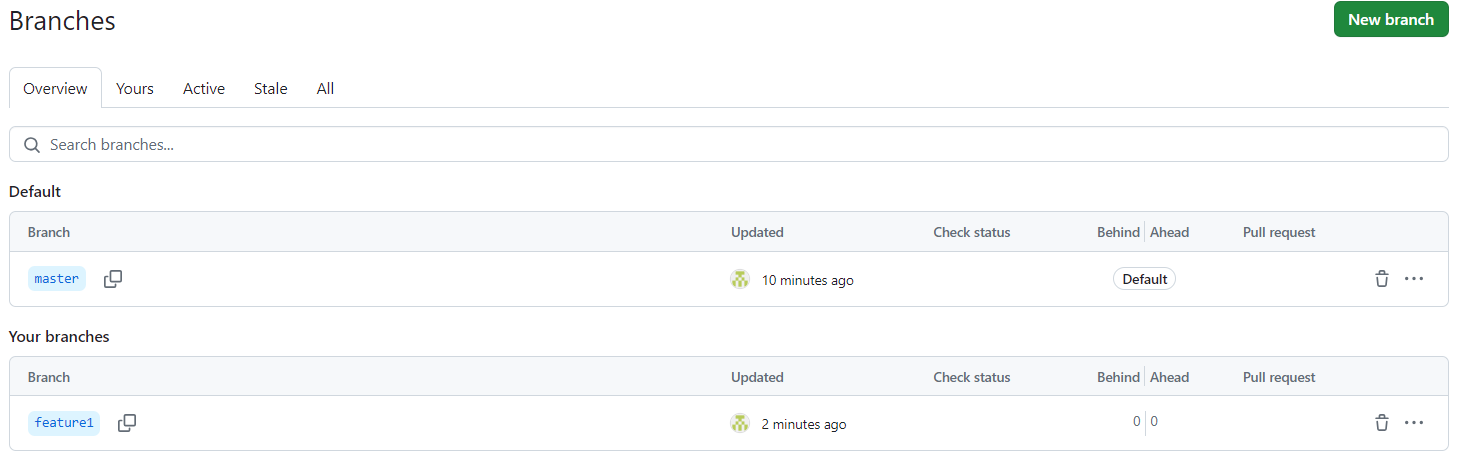


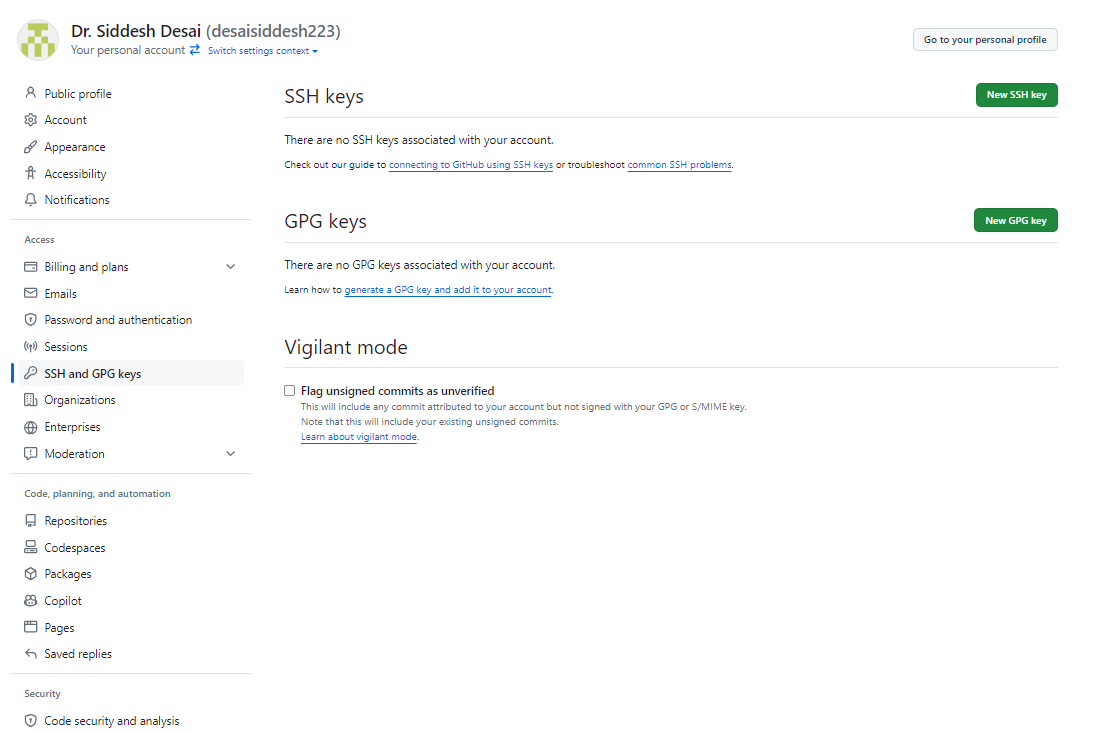
Now push the feature 1 branch:

Command: git push -u origin feature1





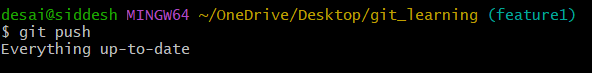




Need to use the selected command which is generated after successful repository creation in the git hub.

If we do only git push. Then it will push the active branch only to the git hub.

Command: git push



Summary commands: