1. **Introduce Yourself to Git**

To set users:

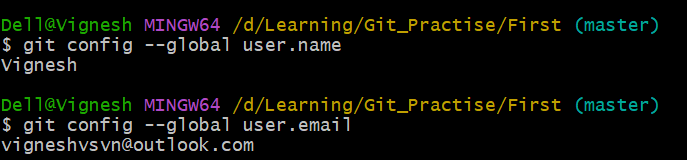
**$ git config --global user.name “Vignesh”**

**$ git config --global user.email** [**vigneshvsvn@outlook.com**](mailto:vigneshvsvn@outlook.com)

To check users.

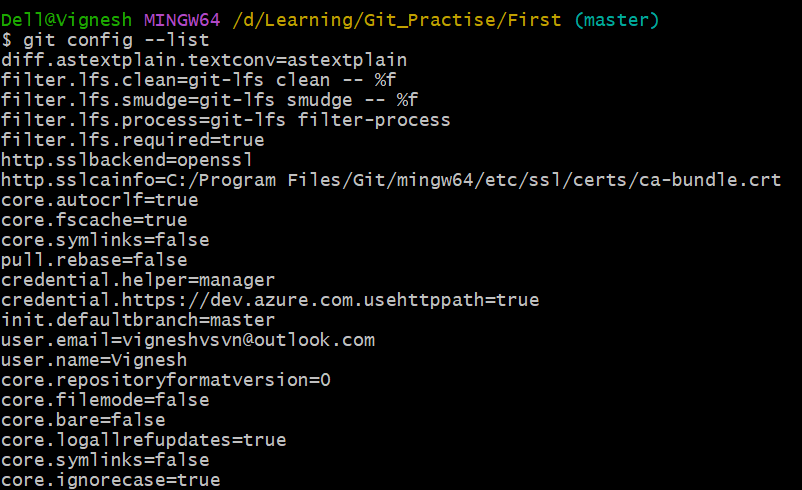
**$ git config --global user.name**

**$ git config --global user.email**

****

1. **To See All Configurations of Git**

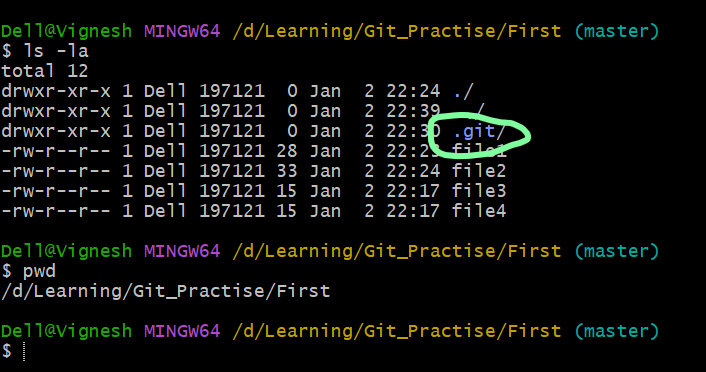
**$ git config –list**

****

1. **To Create New Empty Repository.**

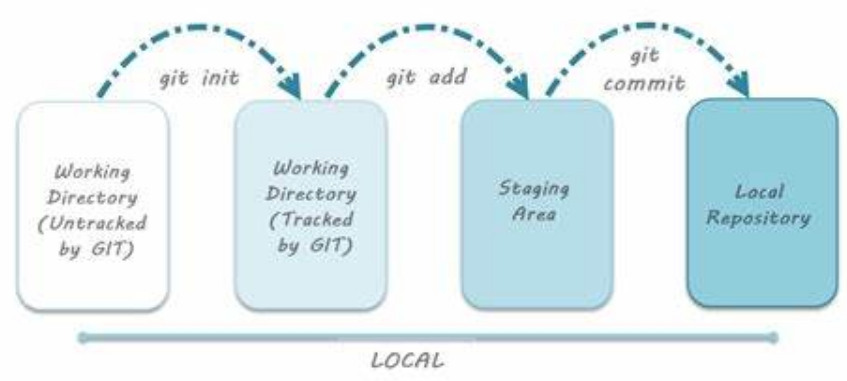
**$ git init .**

* This will create empty repository in current working directory.
* The file created withing this directory will be tracked by git.

****

* If you see .git hidden folder created then it is git local repository.

1. **Git Stages:**

****

*Working Area:*

* Folder which converted into a git repository called working directory.
* Git will track files in working area but it does not mean that it’s added to repository.

*Staging Area:*

* To keep a track of modifications or changes in the file we have to bring that changes to our staging area which we bring by using **Staging**

To Add Specific files to Staging

**$ git add file1 file2 …**

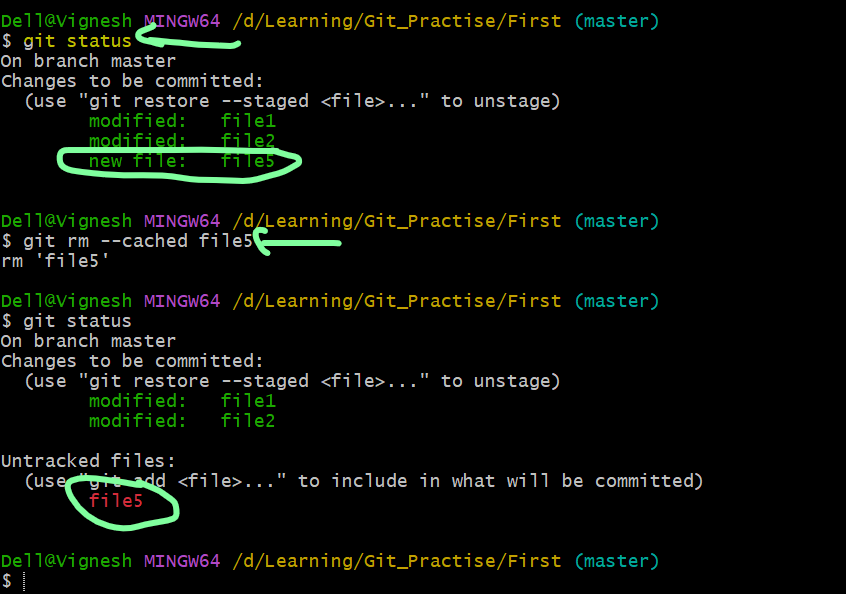
**$ git add –all**

**$ git add .**

**$ git add \***

To Unadd or remove from Staging

**$ git rm –catched file5**



Repository:

* File available in Staging only can be added or committed to Local Repository.

**$git commit “This is my first commit”**

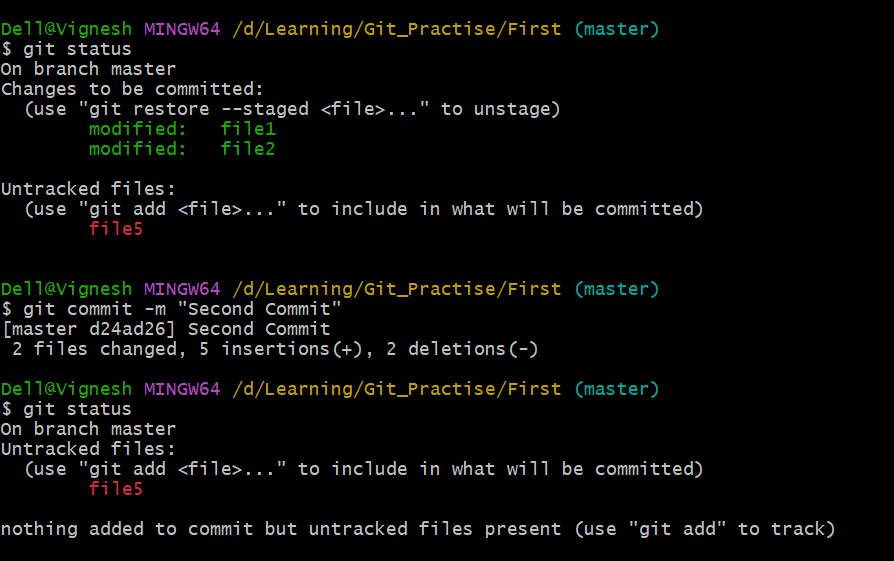
* Committing with -m will create commit id. It will help to in future to refer or revert.
* It’s recommended to use -m always

**$git commit -m “This is my first commit”**

*To Check the Status:*

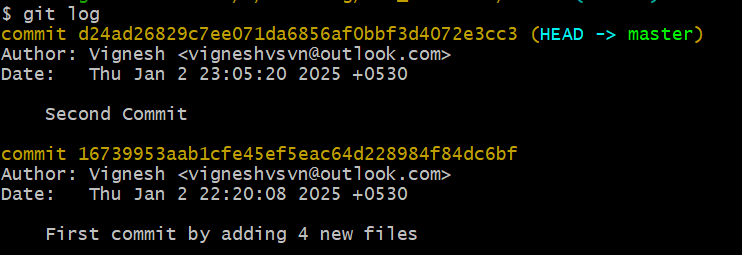
* To check the status of git branch.
* It will show the files in working area without tracking.
* Files in Staging without commit

**$git status**

**

*To Show logs of history commit:*

***$ git log***

**

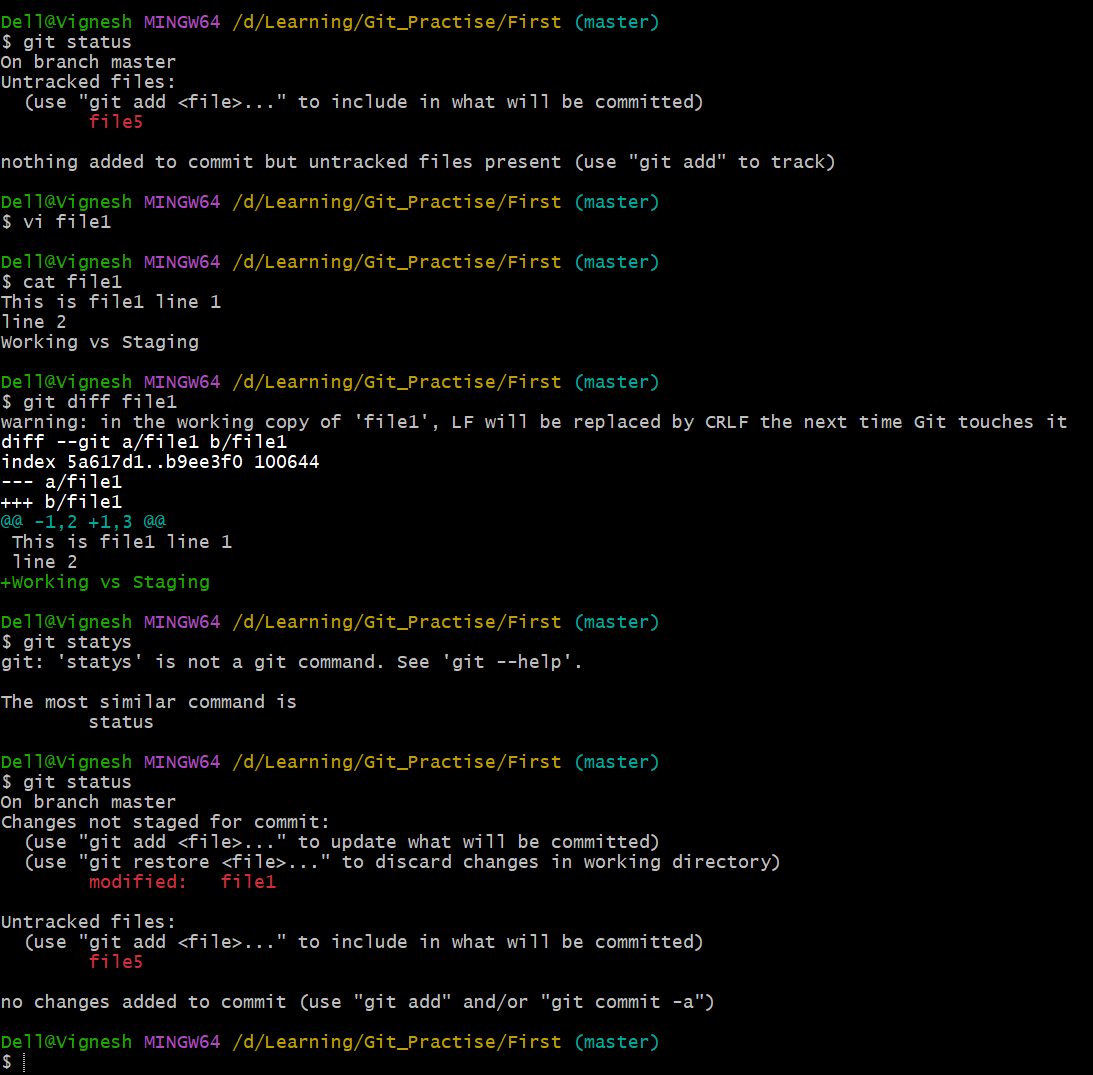
1. **Git Comparisons:**

To Compare files in ***working directory Vs file in staging.***

**$ git diff file1**

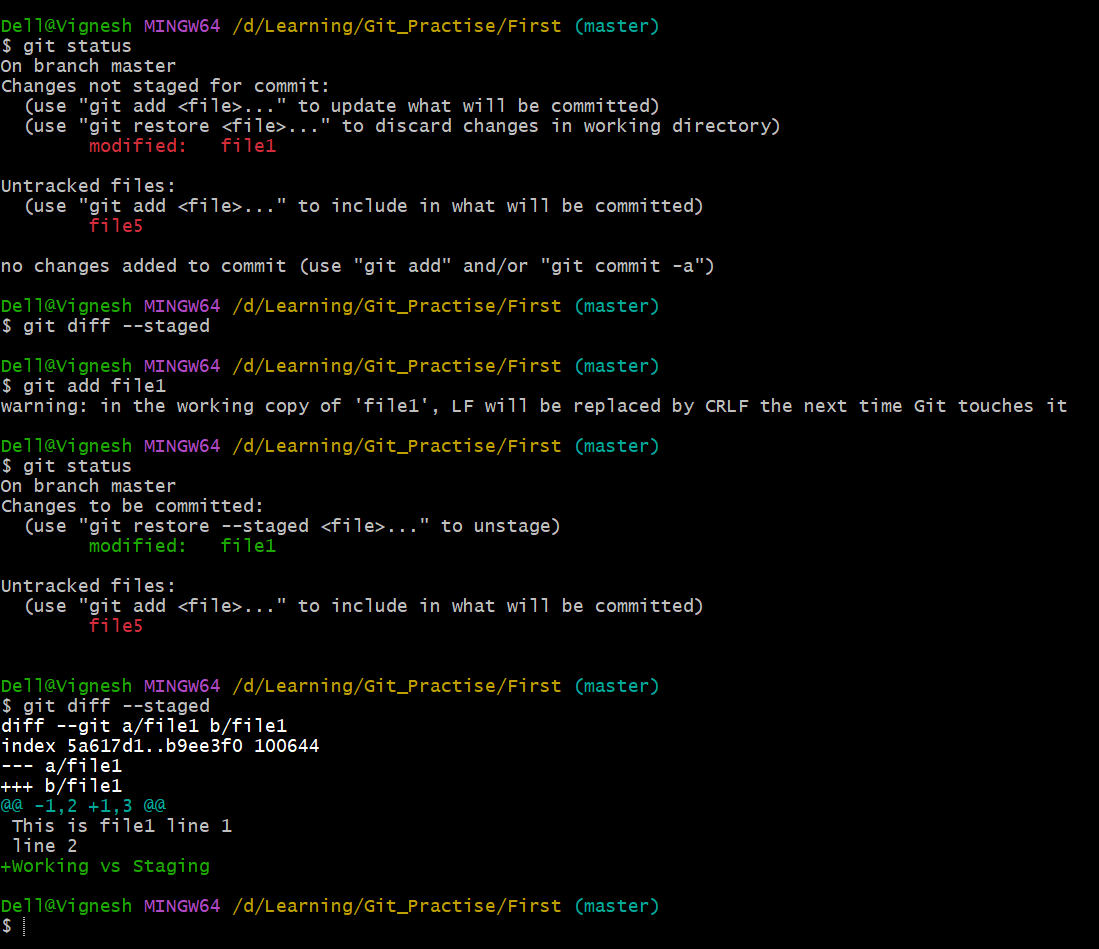
To compare all files in working vs staging

**$ git diff**

****

To Compare Files in Staging Vs Files in Local Repository

**$ git diff --staged**



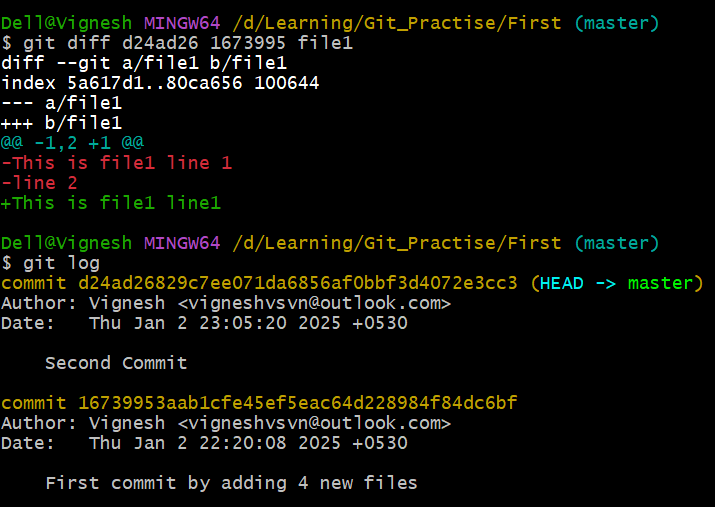
To Compare Working Area file Vs Local Repository

* HEAD means local repository which points latest Commit.

**$ git diff HEAD**

To Compare different commits

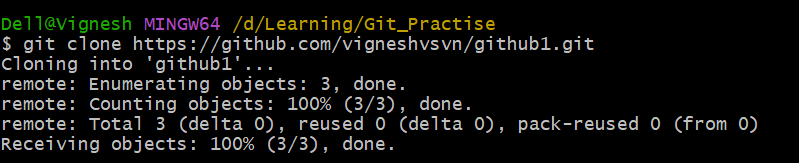
**$ git diff <commitid1> <commitid2>**

****

1. **To Clone Repository from Remote repository GitHub (Distributed VCS)**

* To Copy github repository to local repository

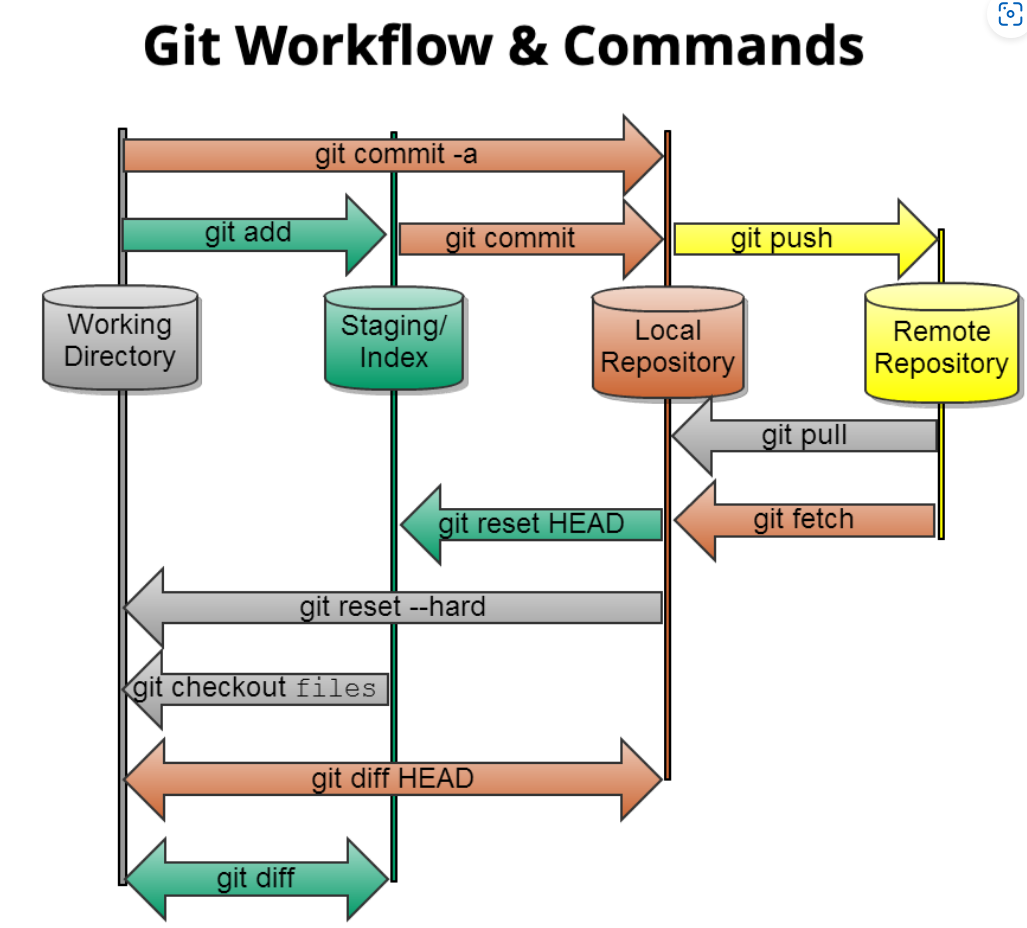
**$git clone <url>**

****

1. **To push code from Local repository to remote repository (Git Hub).**

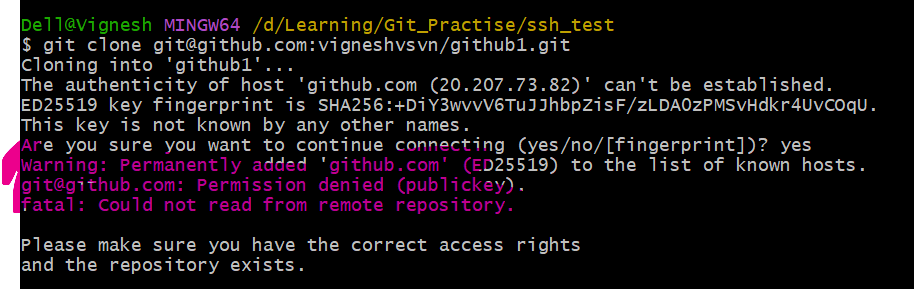
**$ git push origin <branch name>**

**$ git push origin master**

****

1. **SSH Step:**

Before ssh Key setup: Unable to clone using ssh.

****

Step 1: To Create New Key:

**$ ssh-keygen -t ed25519 -C "vigneshvsvn@outlook.com"**

**or**

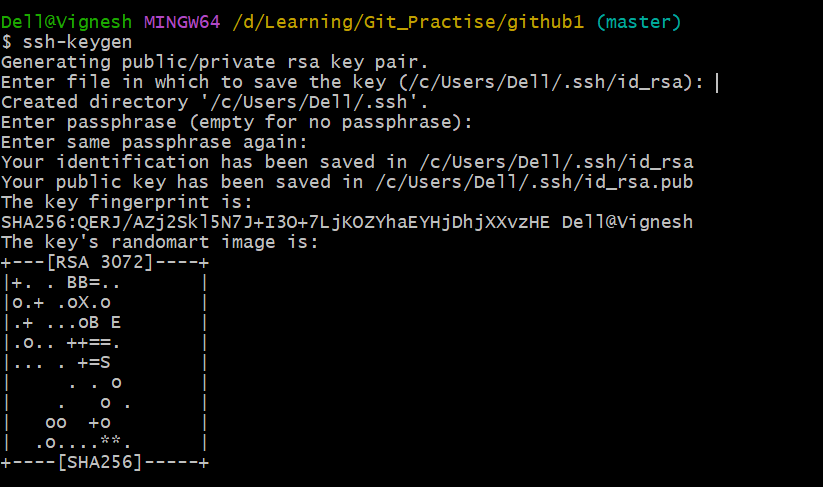
**For Legacy System**

**$ ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"**

**or**

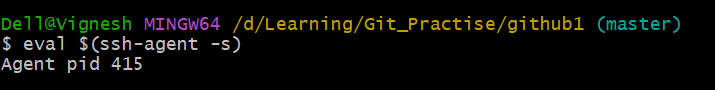
**In simple:**

**$ ssh-keygen**

****

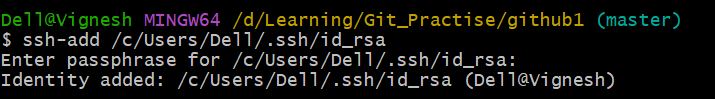
Step 2: To start Agent

**$ eval $(ssh-agent -s)**

****

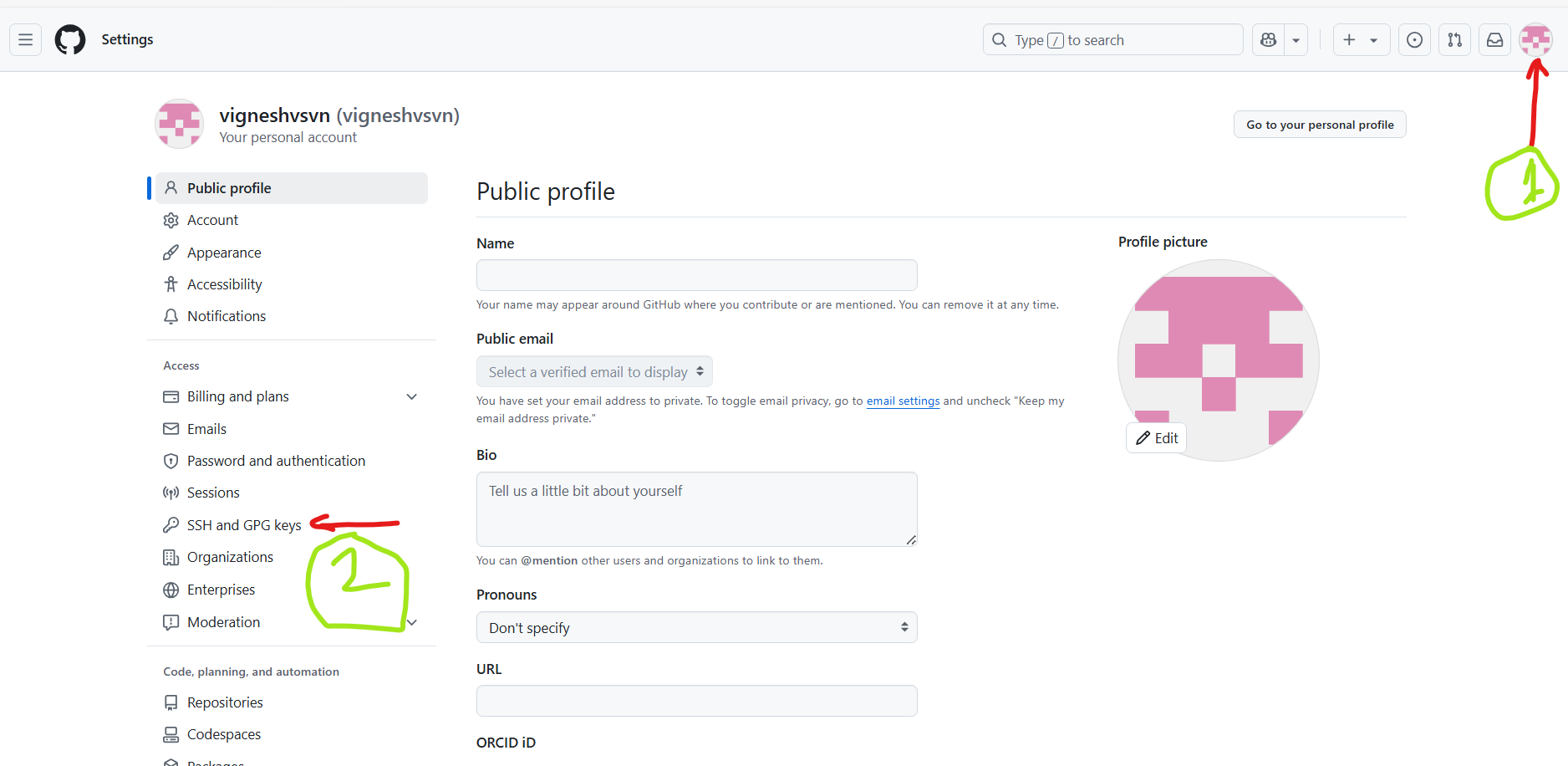
Step 3: To Add SSH Private Key to ssh-agent

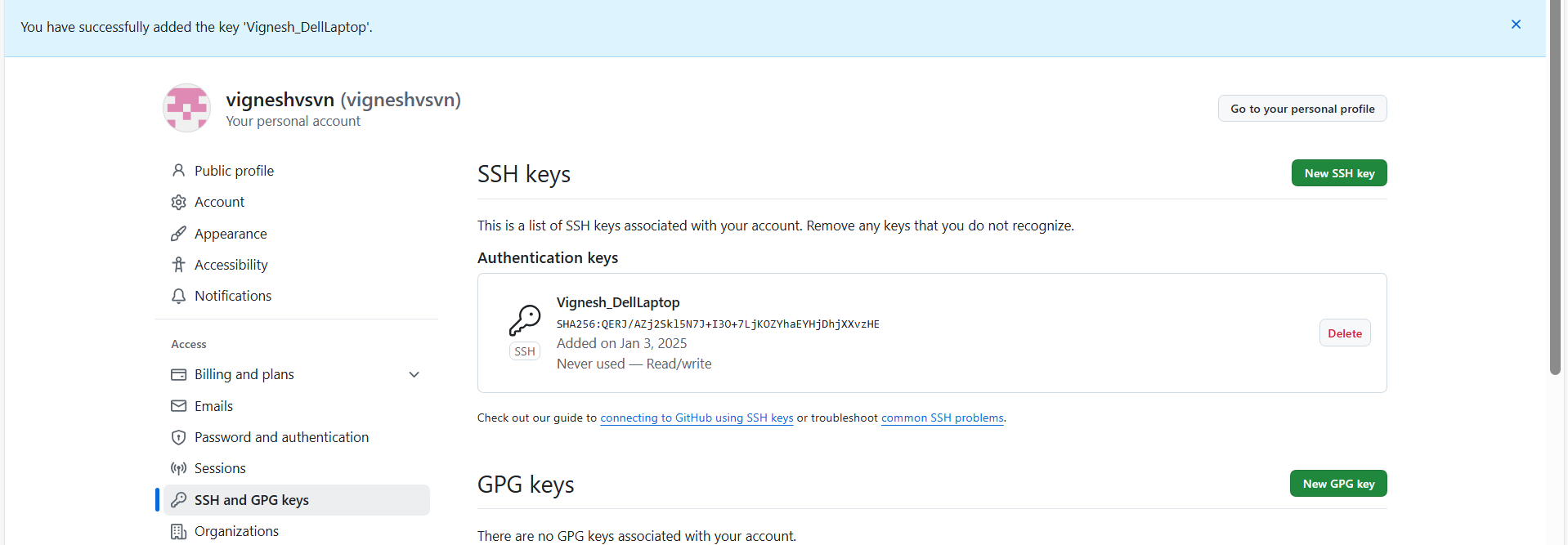
**$ssh-add /c/Users/Dell/.ssh/id\_rsa**

****

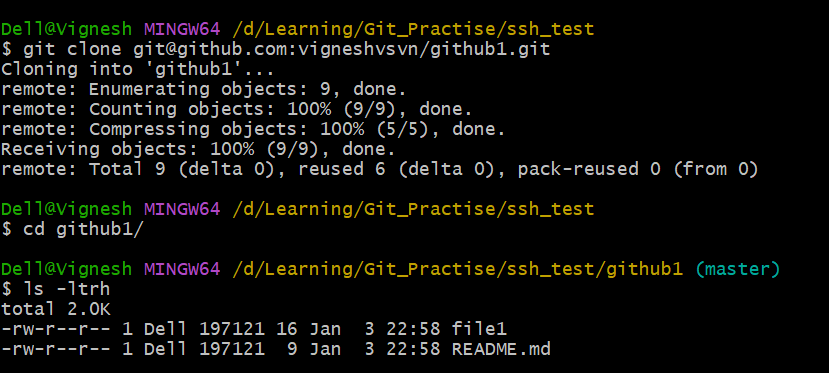
Step 4: Add Public SSH Key to your github account.

Go to Account setting on Github.





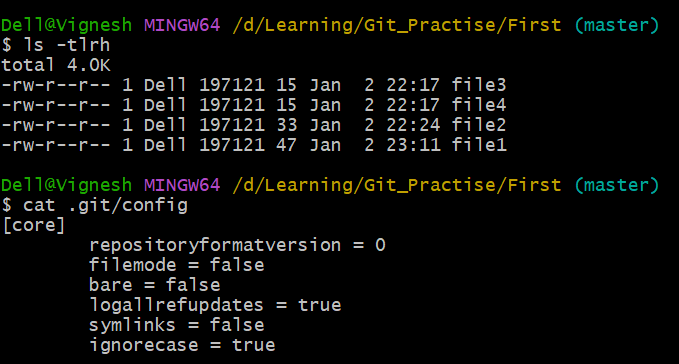
Step 5: Clone the repository using ssh.



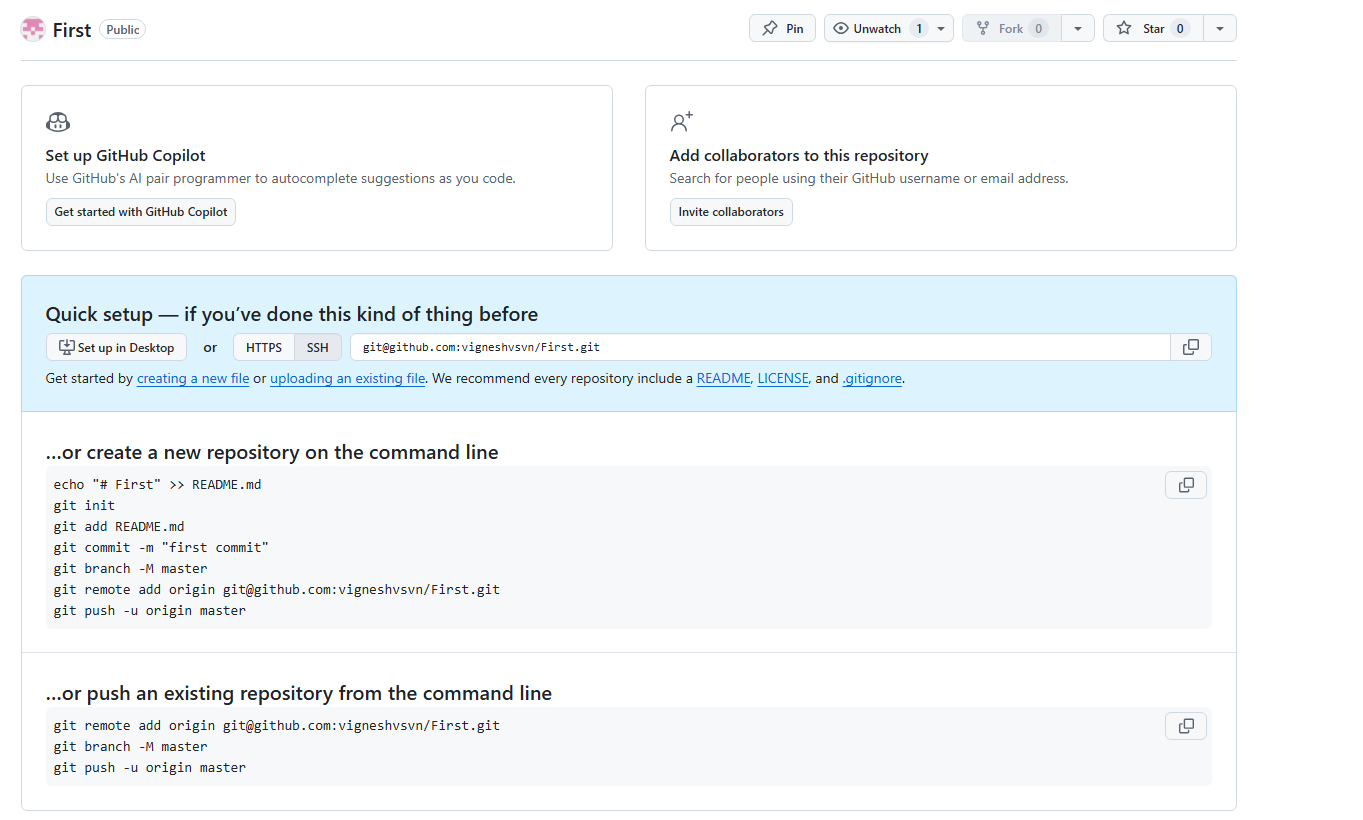
1. **To Map local repository with remote repository.**

* Assume We have local repository created using git init instead of cloning from remote.
* How to update local new repo to remote repo.

Local Repo: Not linked to remote repo.

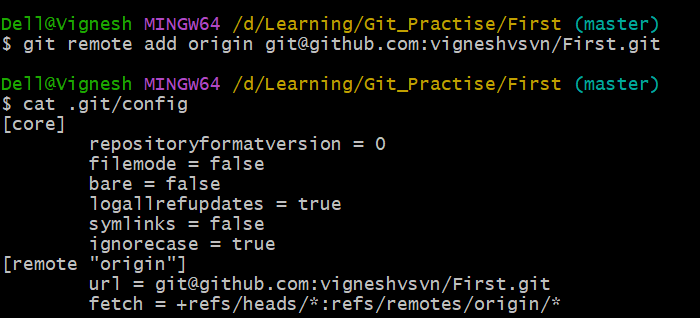


Step 1: Create remote repo in git hub



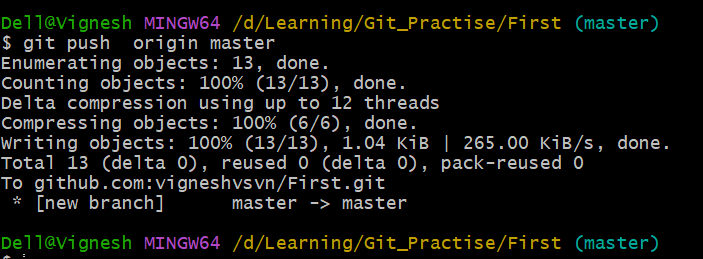
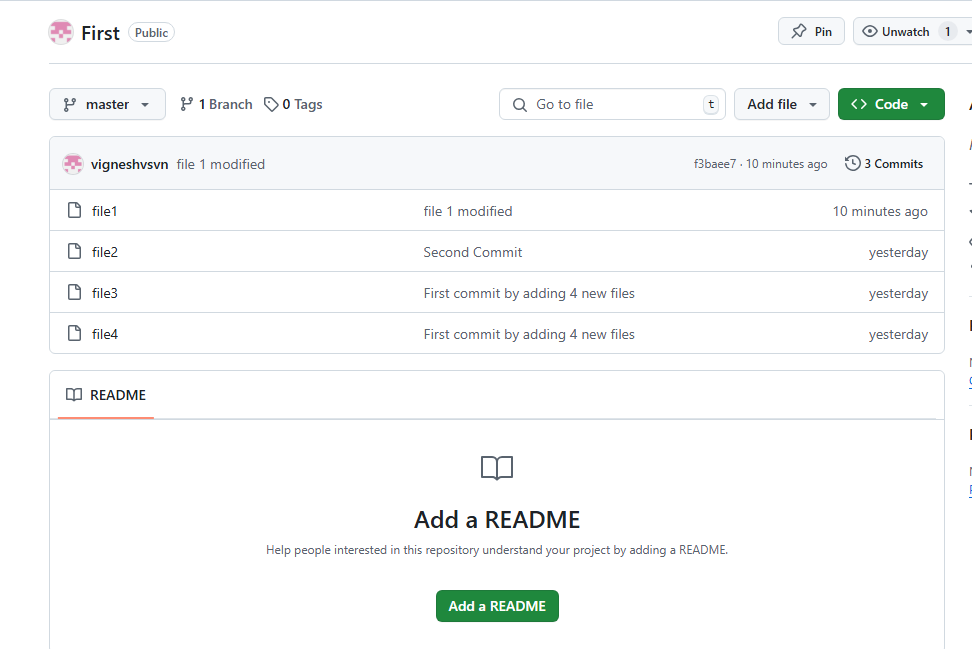
Step 2: To add Local Repository with the Remote repository.

**$ git remote add origin** [**git@github.com:vigneshvsvn/First.git**](mailto:git@github.com:vigneshvsvn/First.git)



Step 3: Now push remote using

**$ git push origin master**

1. How to check what is changed in each commit.

**$ git show <sha code or commitid>**

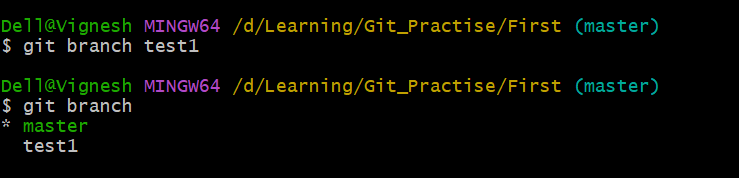
1. How to check exactly what commit changes done specific file.

**$git annotate <file name>**

1. **Branches:**

**To Create New Branch:**

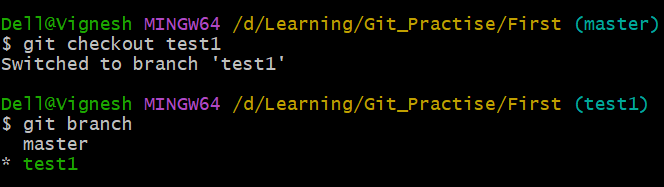
**$ git branch <New Branch name>**

****

**To Check out different branch:**

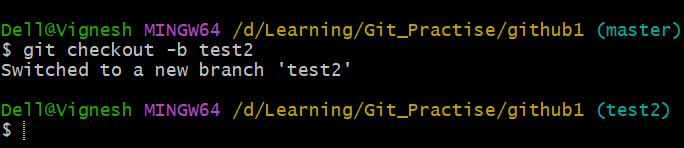
**$ git checkout test1**

* Switched to test1 branch

****

**To Create new branch and switch to new branch in single command.**

**$ git checkout -b test2**

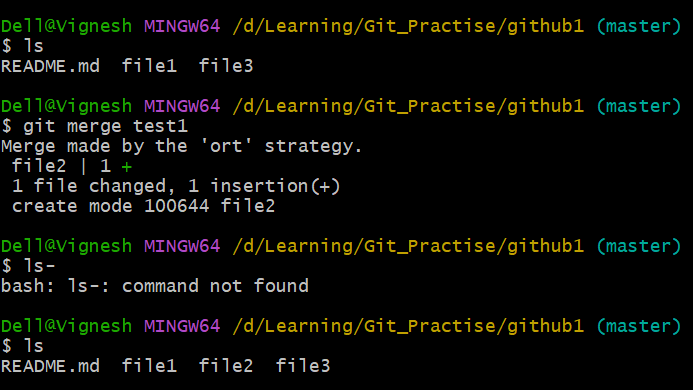
****

**To Merge Branches:**

* Need to be on Destination branch when we are executing the git merge command.
* Eg: If we want to merge the test1 branch to master branch, then we need we need to in master branch first

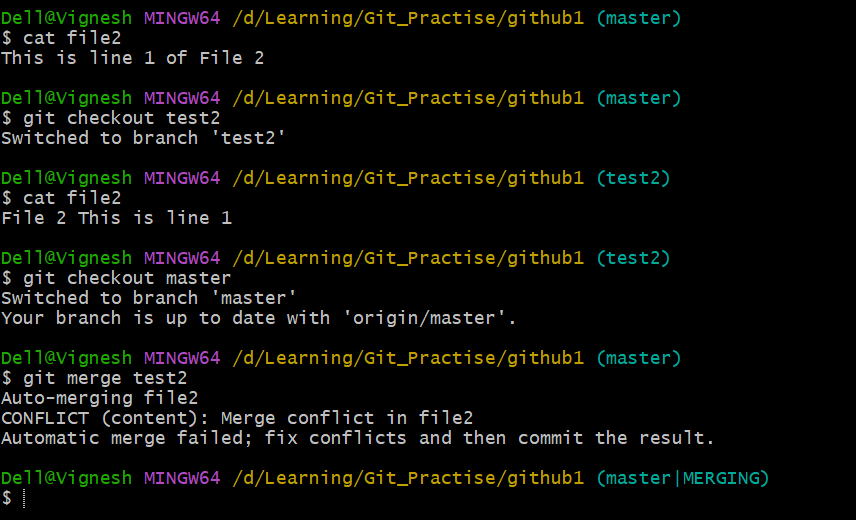
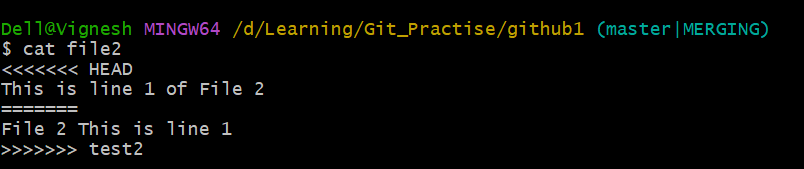
**$ git merge <source branch>**

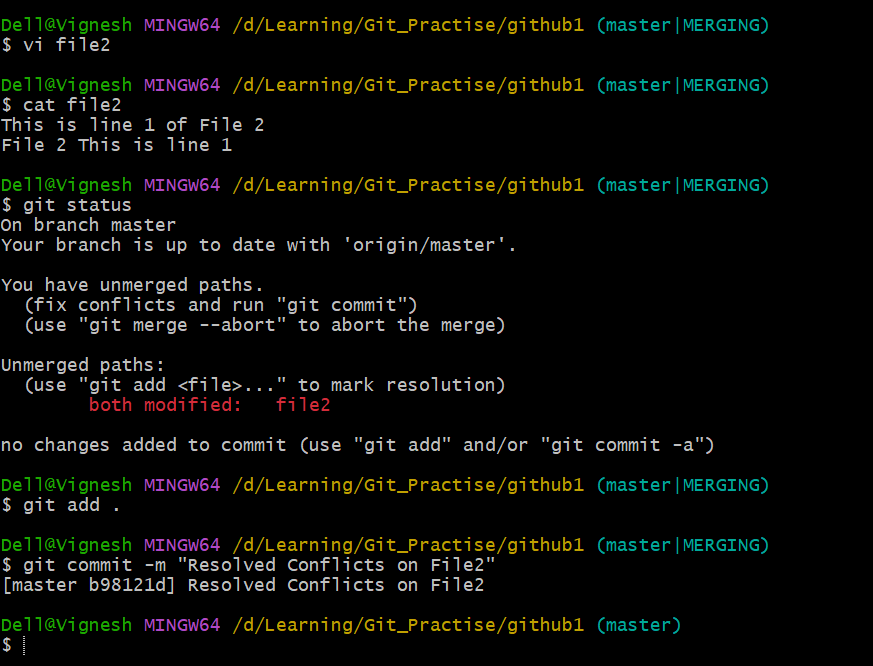
In our below case source branch is test1 and destination is master branch.

****

**Merge Conflicts:**

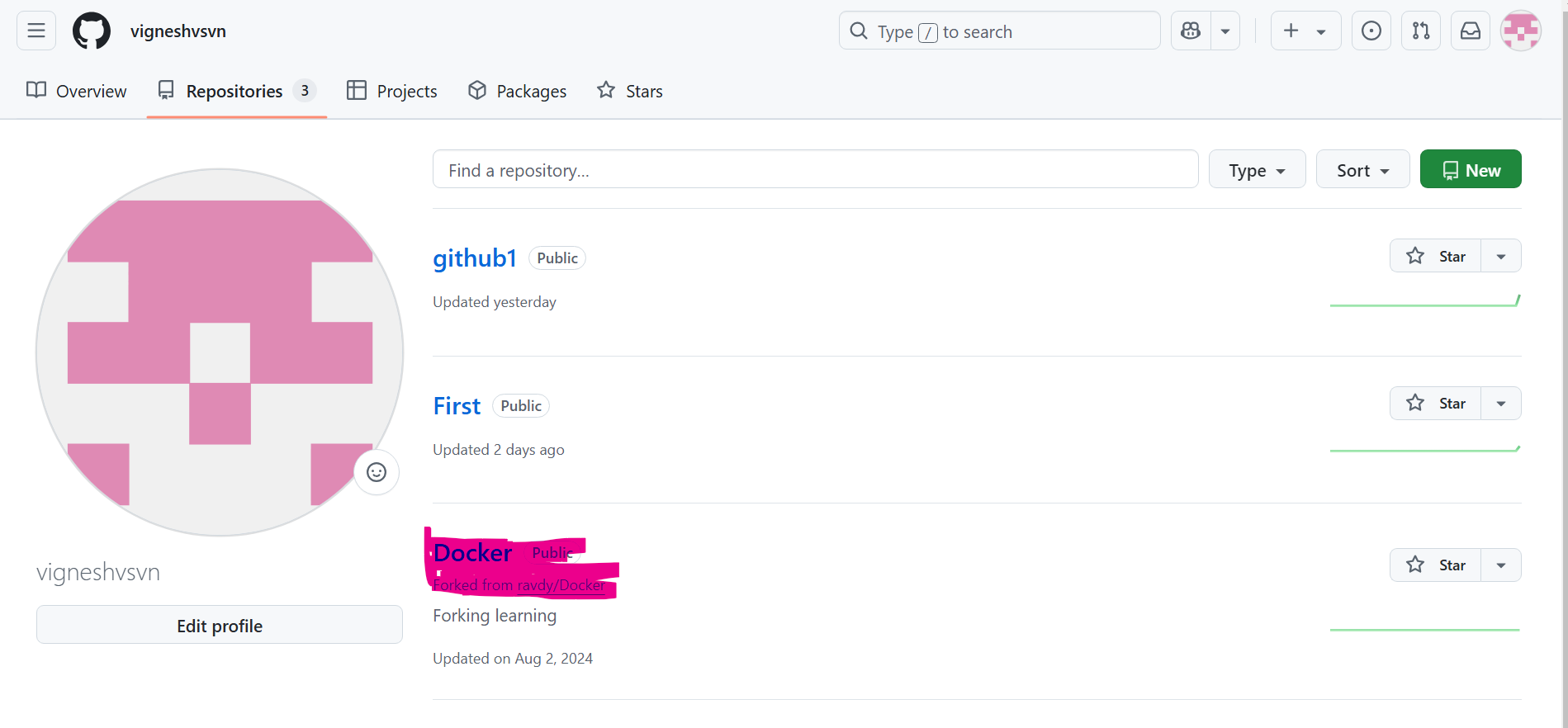
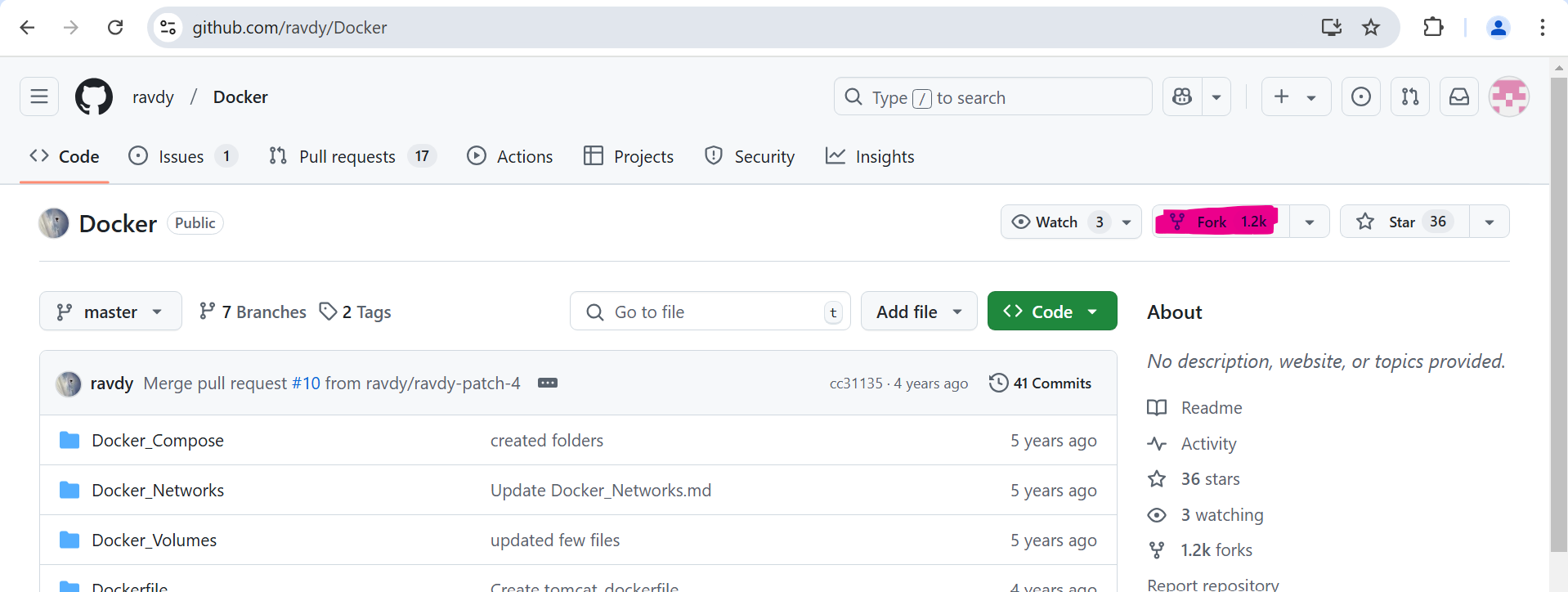
* If same file and same line updated by two different developer, then during the merge conflict will occur.

* Both Developer needs to check together on file2 and decide the final change.
* 

1. Forking

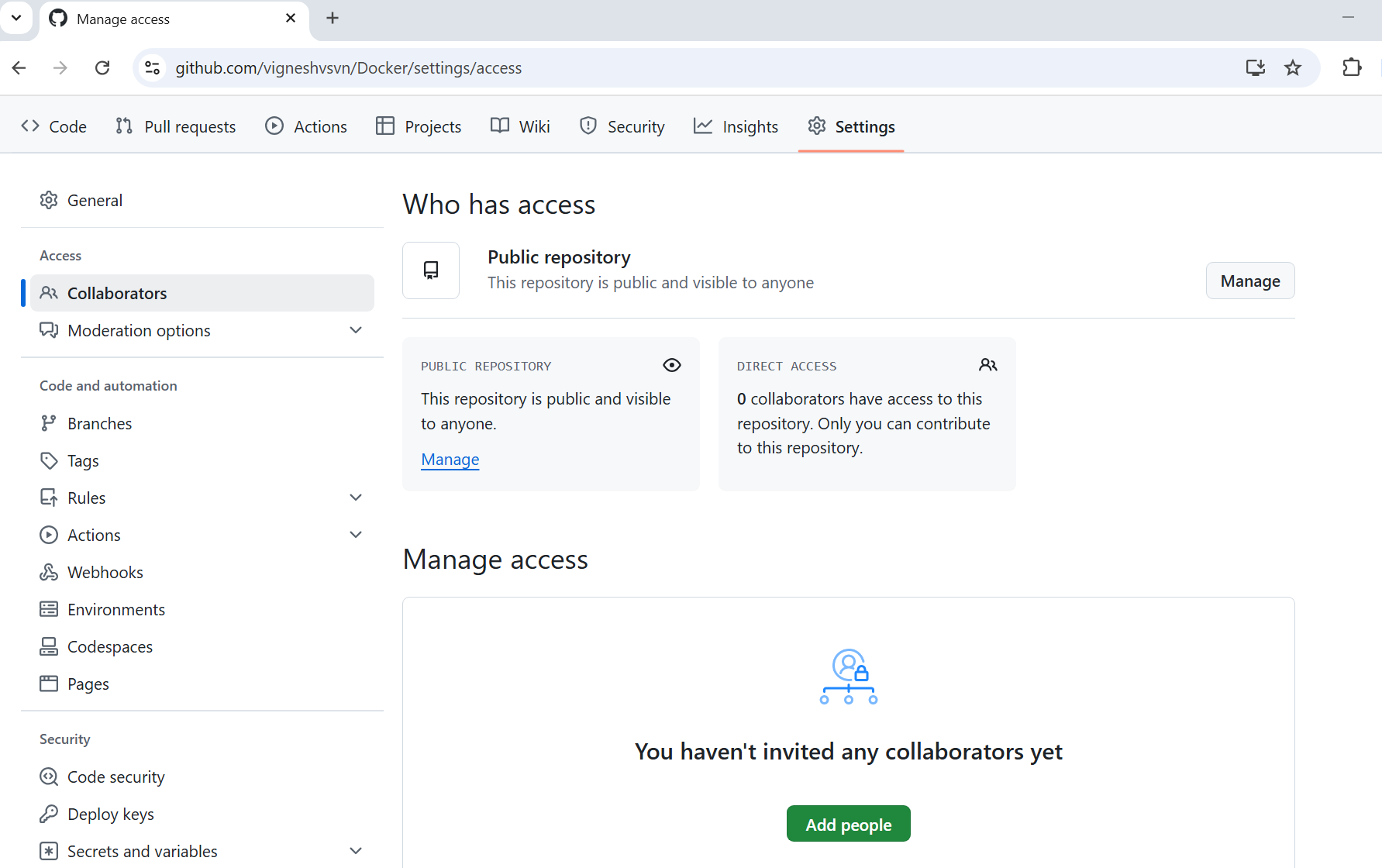
* To Other’s remote public repository to our own remote repository to work.
* Go to other’s remote repository and click on Fork. It will take a copy to our remote repository.



* Then we can clone from out remote repository to our local repository using git clone.

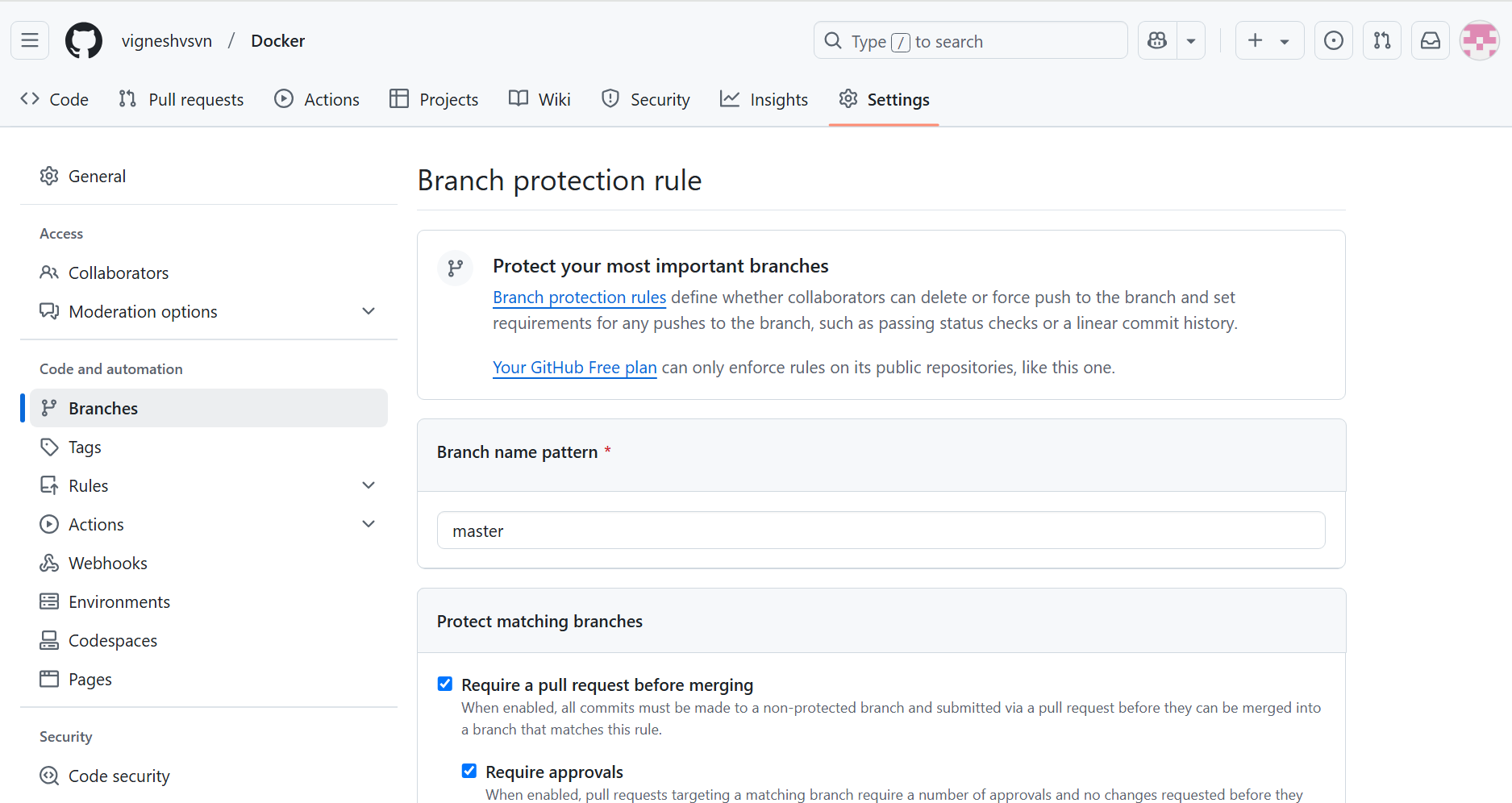
1. **Git Collaborator:**

* Collaborator the one who can work with us. Once we provide collaborator access to the repository then he/she can able to commit our repo without pull request.
* Eg: Collaborator can commit to

****

1. **Protected Branch:**

* To make our Branch secure. We can make it as Protected Branch. So even owner can’t merge directly.
* To secure working branch by avoid commit Eg: master branch
* It should flow through Review process always Eg: Pull request approval from other resource.



1. **Git Tag**

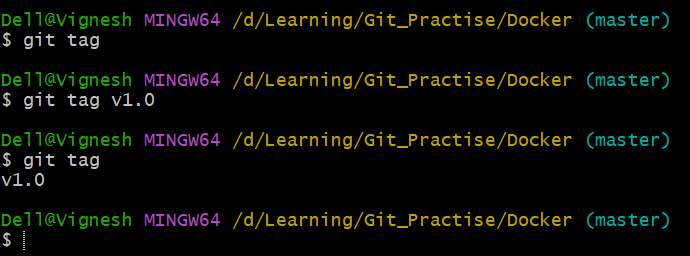
* Tags are the reference point to specific commit in Git History.
* To capture a point in history that is used for a marked version of release.
* Like a Branch that does not change.
* Helps to remember out working codes.

To list all Tags:

$ git tag

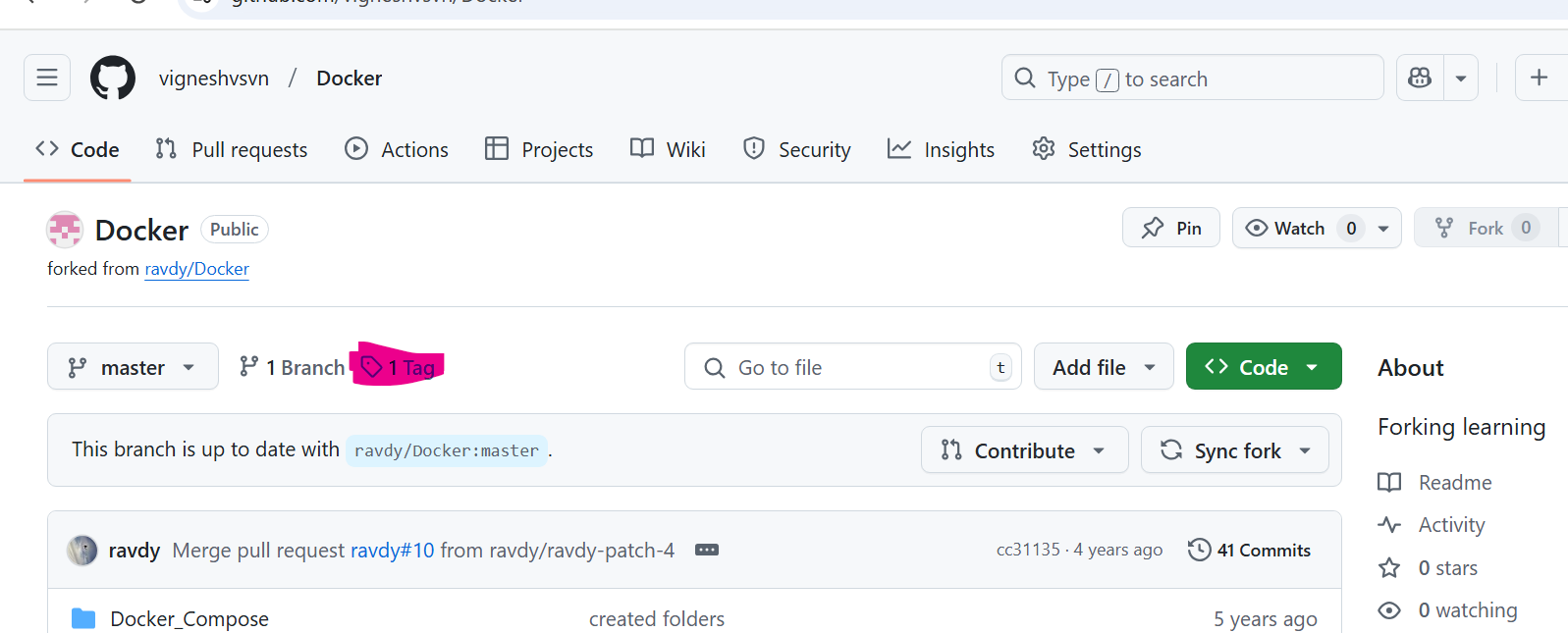
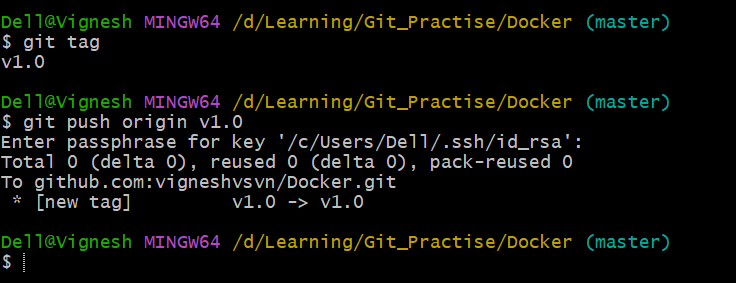
To create tag:

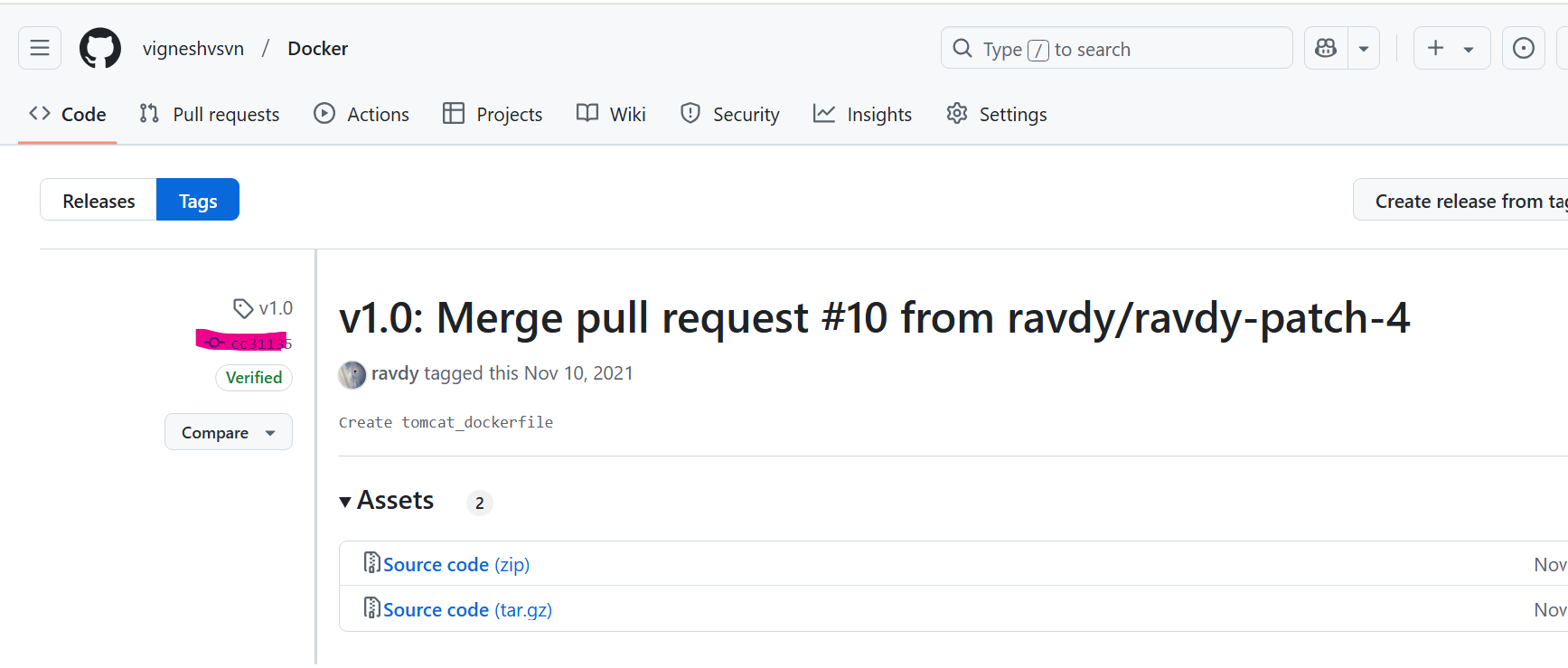
**$git tag <tag nme>**



To Push the tag to Remote repo:

**$git push origin <tag name>**





**To add another tag using specific commit id instead default of Head**

**$ git tag -a <tag name v.1.1> <commit id> -m “New working version”**

1. **Revert Changes:**

* **To Revert Working directory :**

$ git restore <file names …>

Or

$ git checkout -- <file name>

* **To revert Staging Area:**

Step1: revert changes from Staging to working area

$ git restore --staged <file name>

Step2: revert changes from working area

$ git restore <file name>

* **To revert from Local repository**

Step1: revert changes from Local to working area

$ git reset HEAD~

$ git reset HEAD~2 🡪 two commits backward from current commit

$ git reset HEAD~2 🡪 three commits backward from current commit

Step2: revert changes from working area

$ git restore <file name>

1. **Git Ignore File:**

* Help to avoid pushing unnecessary files.
* Create .gitignore add list of file need to ignore and commit .gitignore file
* Git Ignore those lists and will not track it.

1. **Git Rebase:**

* Squeeze into single commit to avoid huge history of commits.

$ git rebase HEAD~4

Last 4 commits to single commit.

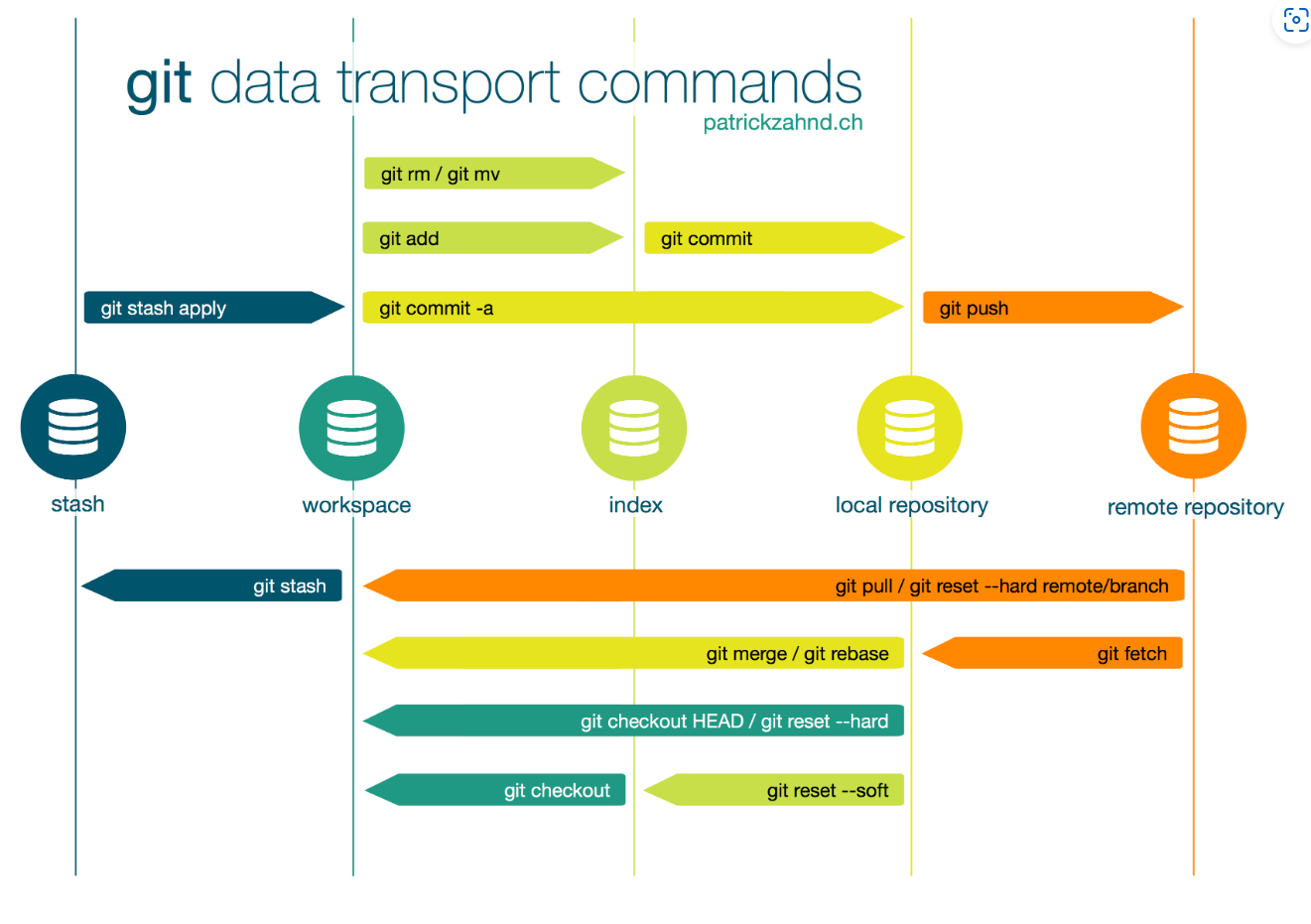
1. **Git pull Vs Git Fetch:**

Git pull:

* synch remote repository with local repository
* Pull all changes to working directory

Git Fetch:

* Only synch remote repository

****