GIT AND GITHUB

Git is a distributed version control system (DVCS) that helps developers track changes in their code and collaborate with others. It was created by Linus Torvalds in 2005 and has since become the standard version control system for software development. Git allows multiple developers to work on a project simultaneously, without interfering with each other's work. It also provides features such as branching, merging, and history tracking.

GitHub, on the other hand, is a web-based platform that provides hosting for Git repositories. It adds a layer of collaboration and social networking features on top of Git. GitHub allows developers to host their Git repositories online, making it easier for teams to collaborate and contribute to open-source projects. It provides features like pull requests, issue tracking, and wikis, which enhance the collaborative aspects of software development.

Some of the commands

1. ls -> it tells the lists of folders/files that we have.
2. mkdir-> creates a new folder. ex mkdir Project , now this will create a new folder named Project.
3. cd -> when I want to go inside of some folder then I use cd.

Now we have to store the history(make/delete some file or changes some code) of some folder/Project, now all of this history will be stored in another folder that is provided by git and termed as git repository, and the name of the folder is .git.

INITIALIZING A GIT REPOSITORY

```
→  project git init
hint: Using 'master' as the name for the initial branc
h. This default branch name
hint: is subject to change. To configure the initial b
ranch name to use in all
hint: of your new repositories, which will suppress th
is warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'm
ain', 'trunk' and
hint: 'development'. The just-created branch can be re
named via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/kunalkushwa
ha/Desktop/Desktop/project/.git/
→  project git:(master)
```

```
→  project git:(master) ls
→  project git:(master) ls -a
.    ..    .git
→  project git:(master) ls .git
HEAD        description info        refs
config      hooks       objects
→  project git:(master)
```

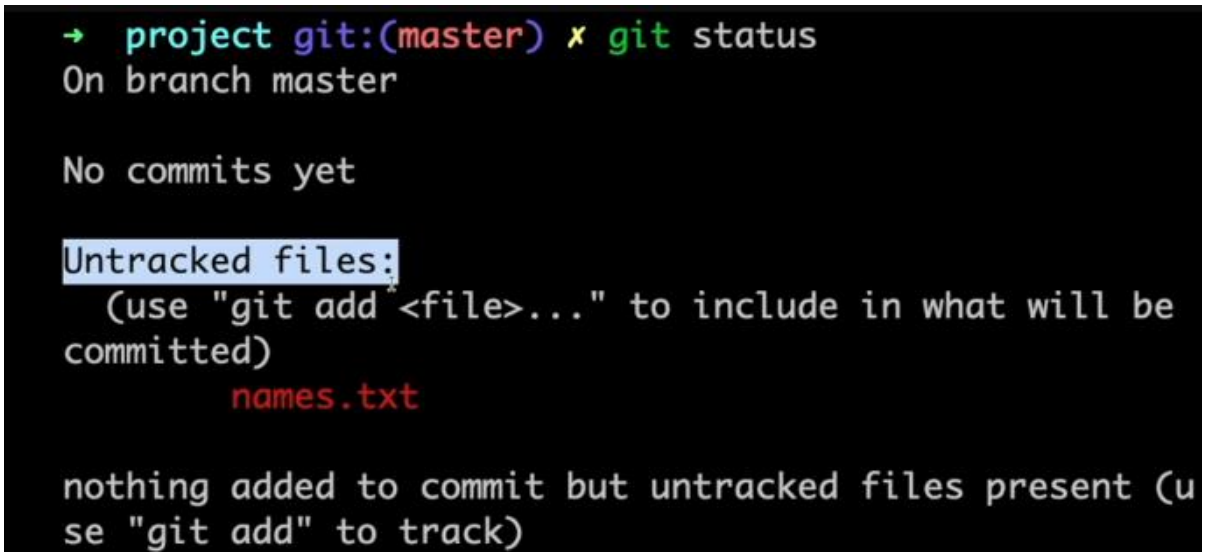
ls -a it will tell the hidden files stored in a folder.

ls .git will show the contents of .git folder.

Now any change made in this folder/Project will be detected and stored in the .git folder.

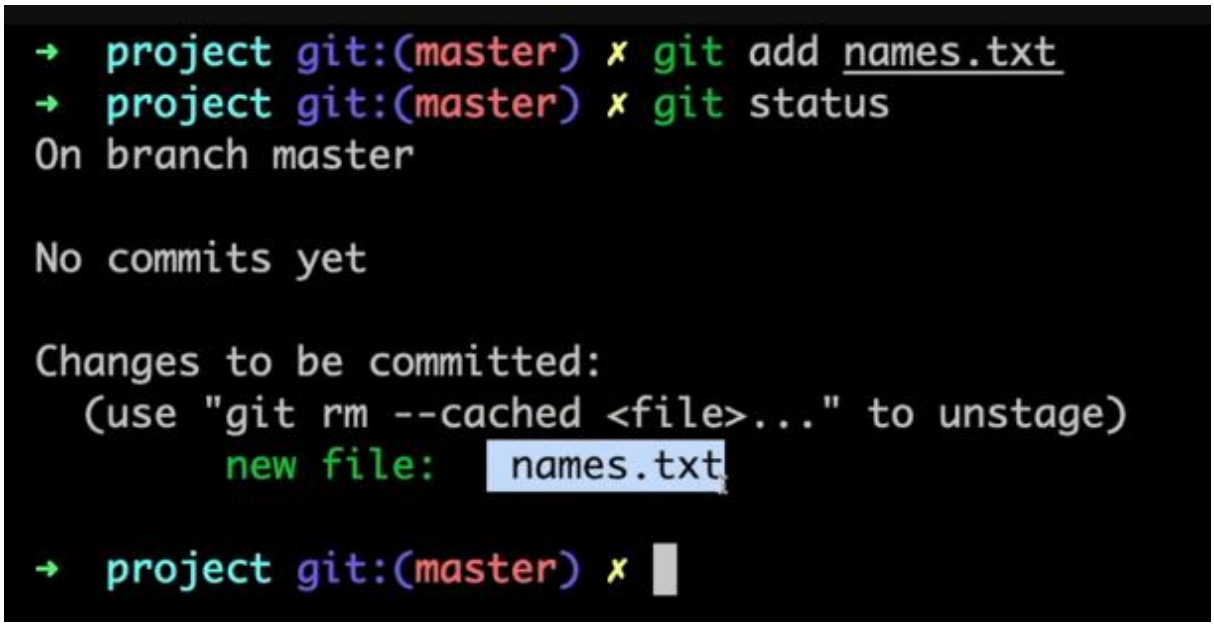
```
→  project git:(master) touch names.txt
```

echo. > example.txt (windows) and touch in Linux is used to create a new file.



git status command is used to see the history.

But the current file Project is untracked, we have to store its history too, so we will do this.



Now we are able to see the history of the file name(depicted by kunal).

We can provide a message like this:-

```
→  project git:(master) ✗ git commit -m "names.txt fil
e added"
[master (root-commit) f4954ce] names.txt file added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 names.txt
→  project git:(master)
```

we can add certain things to our file , like this :-

```
→  project git:(master) vi names.txt
→  project git:(master) ✗ cat names.txt
Kunal Kushwaha
Rahul Rana
Community Classroom
```

vi command provides one platform where we can write something that be added to our file.

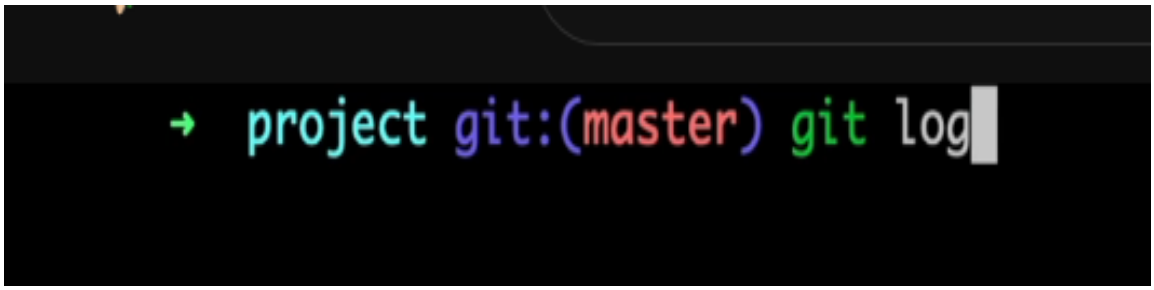
cat command provides the info about whatever we have added;

whatever changes we have done, we have to do git add so that it has the status of tracked rather than untracked.

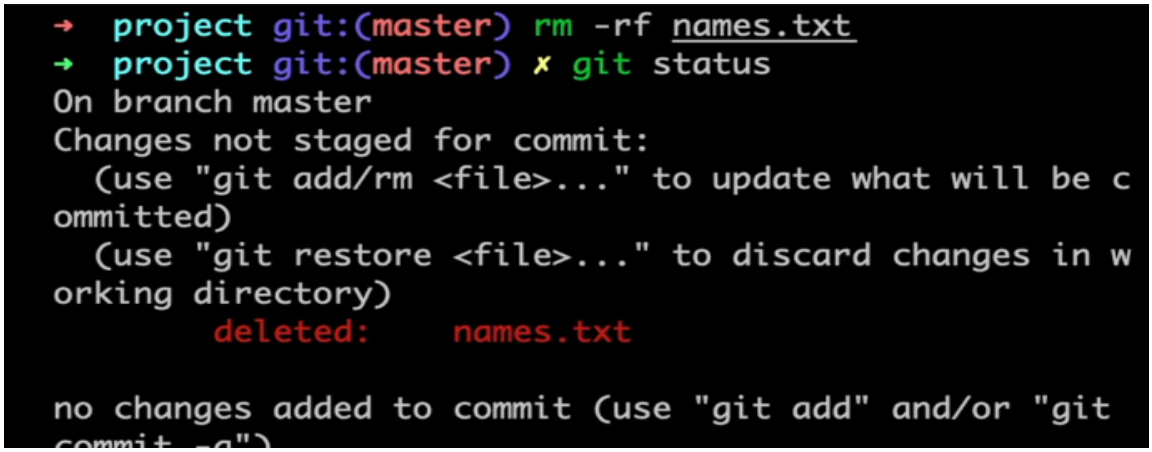
If we have to remove changes from stage, then we do this:-

```
→  project git:(master) ✗ git restore --staged names.t
xt
→  project git:(master) ✗ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be comm
itted)
  (use "git restore <file>..." to discard changes in w
orking directory)
        modified:   names.txt

no changes added to commit (use "git add" and/or "git
commit -a")
→  project git:(master) ✗
```

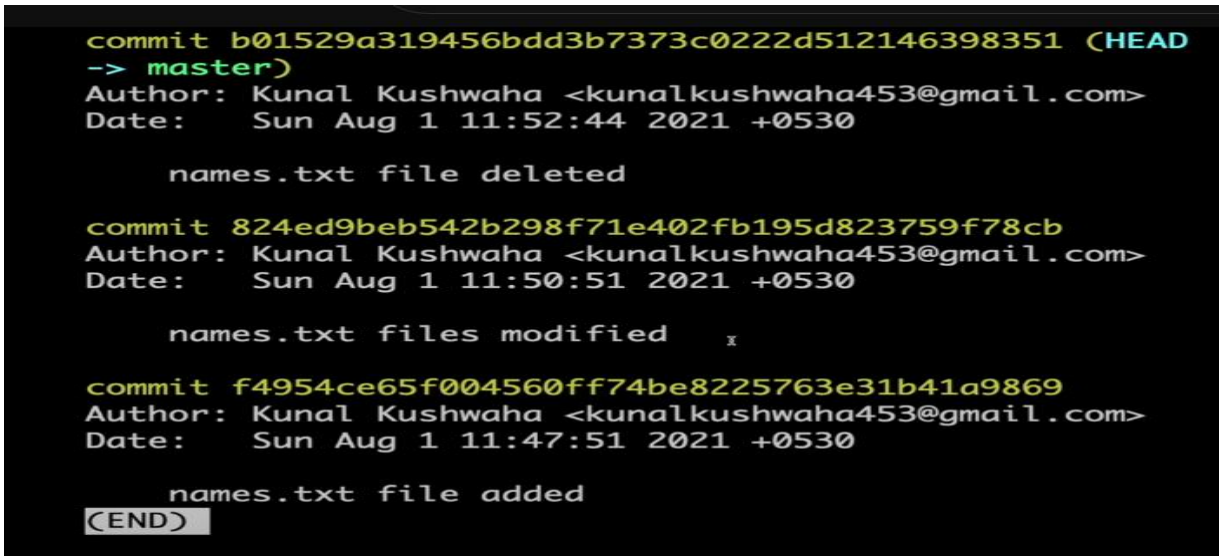
If we have to see the history of whenever we have done any changes :-



If we want to delete the file:-



If we have to restore file that has been deleted , then first we have to go in the particular status of a commit by using its hash code:-



The above view will get us by typing git log.

Now do this:-

```
→  project git:(master) git log
→  project git:(master) git reset f4954ce65f004560ff74
be8225763e31b41a9869
Unstaged changes after reset:
D       names.txt
→  project git:(master) ✗
```

Now if we do git log we will only able to see one commit, those commits before that will be deleted because are now in the situation where that particular commit and it will look like this:-

```
commit f4954ce65f004560ff74be8225763e31b41a9869 (HEAD
-> master)
Author: Kunal Kushwaha <kunalkushwaha453@gmail.com>
Date:   Sun Aug 1 11:47:51 2021 +0530

    names.txt file added
(END)
```

If we do make changes it should not be listed under the history be will store these changes under stash and use them whenever we want, we have created another folder and done some changes and stored it into the stash:-

```
→  project git:(master) ✗ vi surnames.txt
→  project git:(master) ✗ touch houses.txt
→  project git:(master) ✗ git add .
→  project git:(master) ✗ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    names.txt -> houses.txt
        new file:   surnames.txt

→  project git:(master) ✗ git stash
Saved working directory and index state WIP on master:
 f4954ce names.txt file added
→  project git:(master) git status
On branch master
nothing to commit, working tree clean
→  project git:(master) git status
```

Whatever new things you do just make it under the track by doing git add.

After doing git stash we type git status nothing comes that is the speciality of git stash.

Now folder has only a name.txt file is just like a file that is present inside the folder name project.

Now if we want to call out the changes in the stash to the staging then we do this:-

```
→  project git:(master) git stash pop
Removing names.txt
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   houses.txt
        new file:   surnames.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be c
ommitted)
  (use "git restore <file>..." to discard changes in w
orking directory)
        deleted:    names.txt

Dropped refs/stash@{0} (cdfe20121962a86efa35d57e6a5a35
4615e49371)
→  project git:(master) ✗
```

git stash pop is used to call the changes stored in stash.

now our folder has two files {sirname, house}.

The name of the {name} file has changed to the {sirname}.

Again we do git add. And git stash now every this is stored in stash , now we will clear stash and after that nothing will be stored in the stash :-

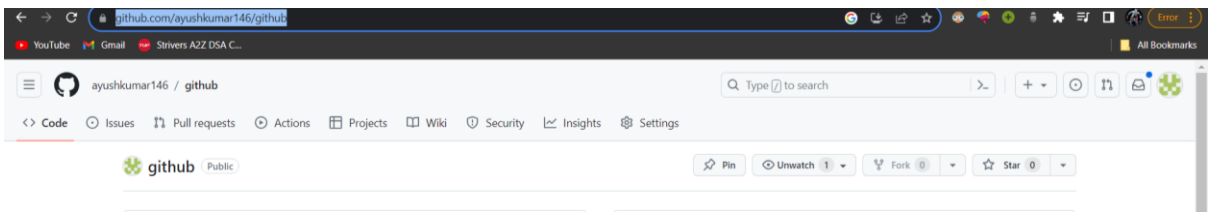
```
→  project git:(master) git stash clear
→  project git:(master)
```

GITHUB

we will add our repo to our folder:-

```
→  project git:(master) git remote add origin https://
github.com/kunal-kushwaha/CommunityClassroom-Git.git
→  project git:(master)
```

The URL after the origin is the repo link:-



```
➜  project git:(master) git remote add origin https://
github.com/kunal-kushwaha/CommunityClassroom-Git.git
➜  project git:(master) git remote -v
origin  https://github.com/kunal-kushwaha/CommunityCla
ssroom-Git.git (fetch)
origin  https://github.com/kunal-kushwaha/CommunityCla
ssroom-Git.git (push)
➜  project git:(master)
```

git remote -v will tell what links are attached to the folder.

Now whatever changes we have made in our folder, we want to send those changes in repository too so we will do this:-

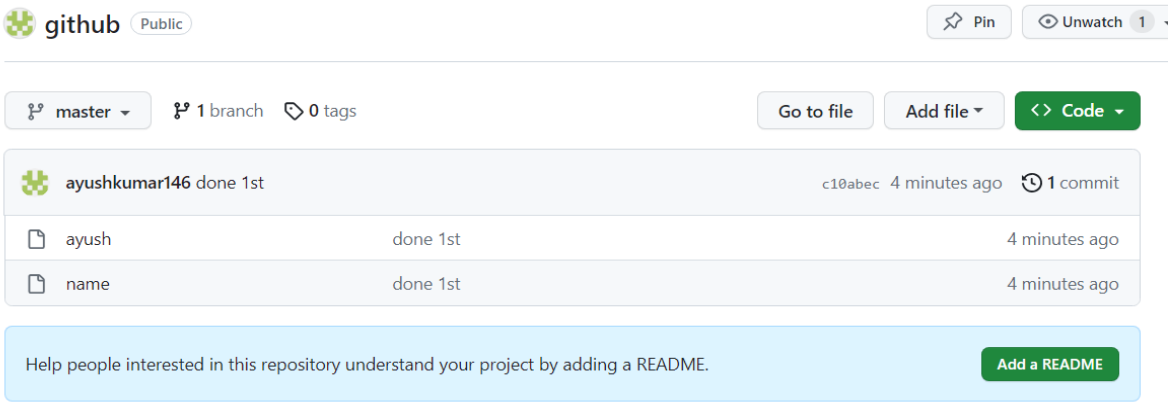
```
➜  project git:(master) git remote add origin https://
github.com/kunal-kushwaha/CommunityClassroom-Git.git
➜  project git:(master) git remote -v
origin  https://github.com/kunal-kushwaha/CommunityCla
ssroom-Git.git (fetch)
origin  https://github.com/kunal-kushwaha/CommunityCla
ssroom-Git.git (push)
➜  project git:(master) git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 224 bytes | 224.00 KiB/s,
 done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

git push origin master is used to make the same changes in the repo too.

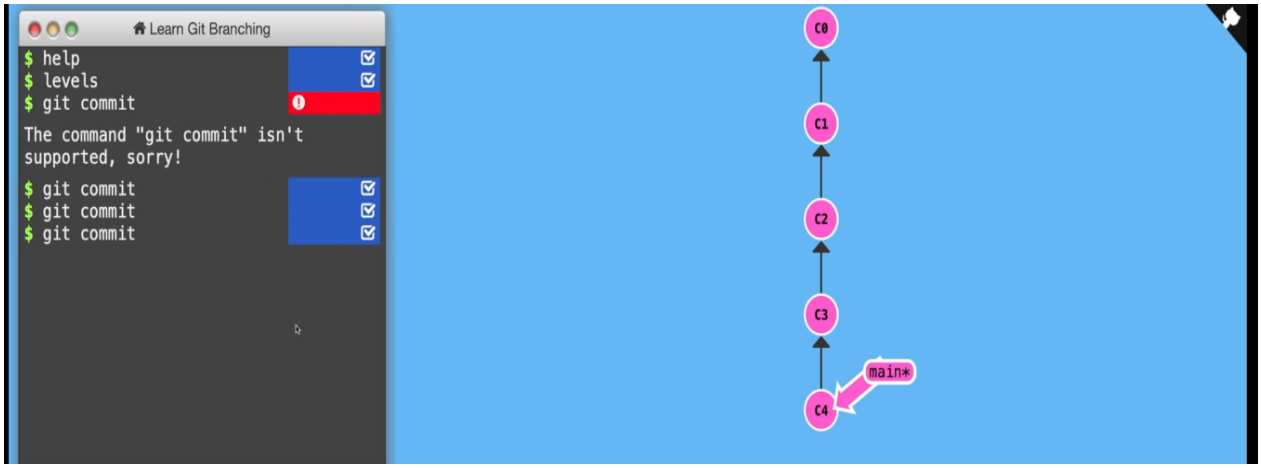
NOTE:-it is very important to commit your changes before pushing to git hub.

```
C:\Users\AYUSH KUMAR DAS\Documents\gitt>git commit -m "done 1st"
[master (root-commit) c10abec] done 1st
 2 files changed, 2 insertions(+)
 create mode 100644 ayush
 create mode 100644 name

C:\Users\AYUSH KUMAR DAS\Documents\gitt>git branch
* master

C:\Users\AYUSH KUMAR DAS\Documents\gitt>git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 291 bytes | 145.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ayushkumar146/github
 * [new branch]      master -> master
```

Now it will look like this :-



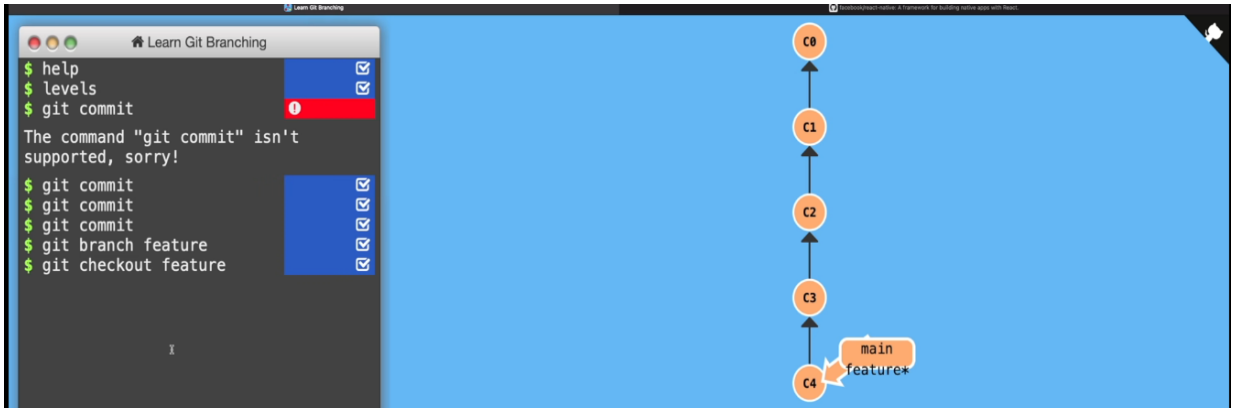
BRANCHES



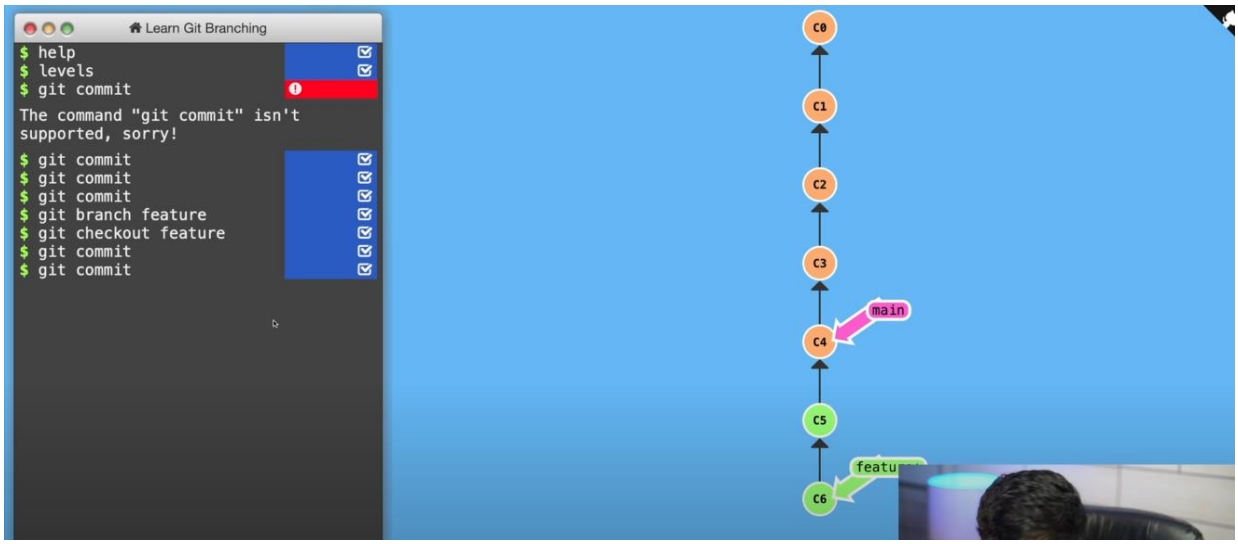
Whenever create the file and do a git commit some extra branches are created as shown in the figure.

Normally we never commit on the main branch that why we usually make another branch and commit on that so our main branch will not be affected.

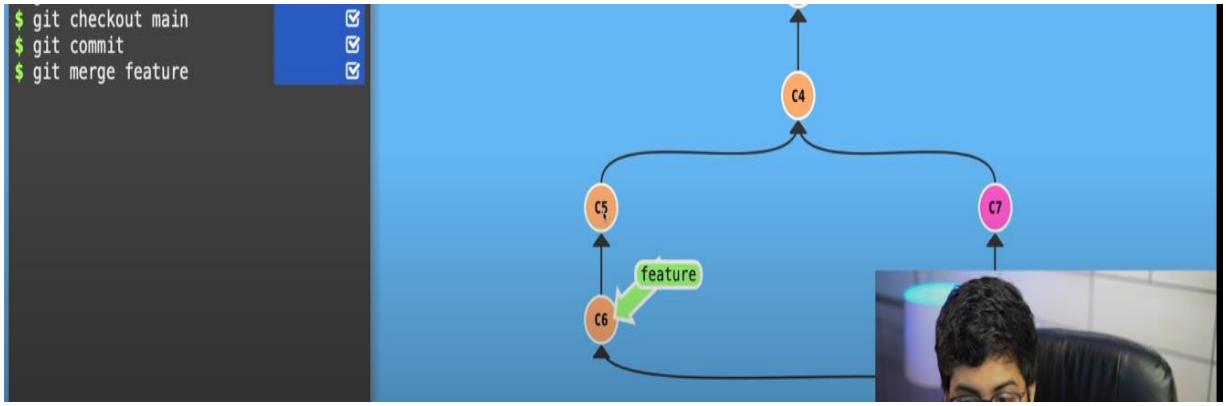
Initially, the star which is a head pointer points towards the main, we create another branch named feature and it also points towards the main now we make a head pointer to point towards the feature branch not the main branch.



Now whatever commits we make it added to the feature branch.



We can make feature branch to the part of branch by merging:-



For merging simply do this:-

```
C:\Users\AYUSH KUMAR DAS\Documents\gitt>git merge ww
Merge made by the 'ort' strategy.
 lala | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 lala

C:\Users\AYUSH KUMAR DAS\Documents\gitt>git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 267 bytes | 267.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ayushkumar146/github
   2cf1f54..4ae3564  master -> master

C:\Users\AYUSH KUMAR DAS\Documents\gitt>
```

Make sure u check out to the main/master branch

First, merge it the push it to the origin master.

For industrial use follow this video from 37 min.

https://www.youtube.com/watch?v=apGV9Kg7ics;

CLONE

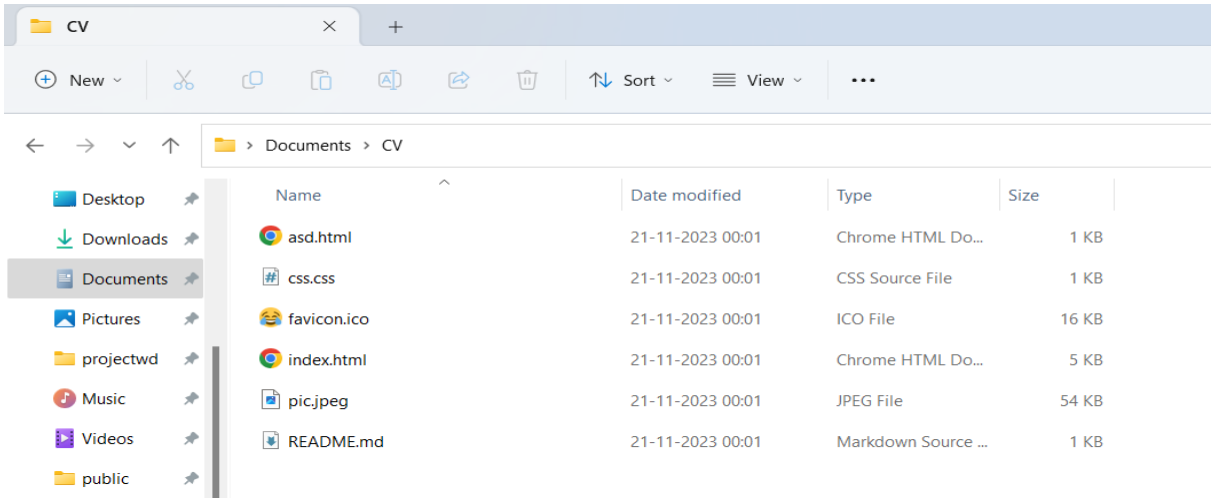
we can make clone of any repository like this:-

```
➜  Desktop git clone https://github.com/kunal-kushwaha
/commclassroomOP.git
Cloning into 'commclassroomOP'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-re
used 0
Receiving objects: 100% (3/3), done.
➜  Desktop cd commclassroomOP
➜  commclassroomOP git:(main)
```

We use git clone {url} to make same repo of other in our github account and contents of that repo will save in our desired folder.

```
C:\Users\AYUSH KUMAR DAS\Documents>git clone https://github.com/ayushkumar146/CV.git
Cloning into 'CV'...
remote: Enumerating objects: 40, done.
remote: Counting objects: 100% (40/40), done.
remote: Compressing objects: 100% (36/36), done.
Receiving objects: 100% (40/40), 68.37 KiB | 843.00 KiB/s, done.
Resolving deltas: 100% (16/16), done.


C:\Users\AYUSH KUMAR DAS\Documents>
```

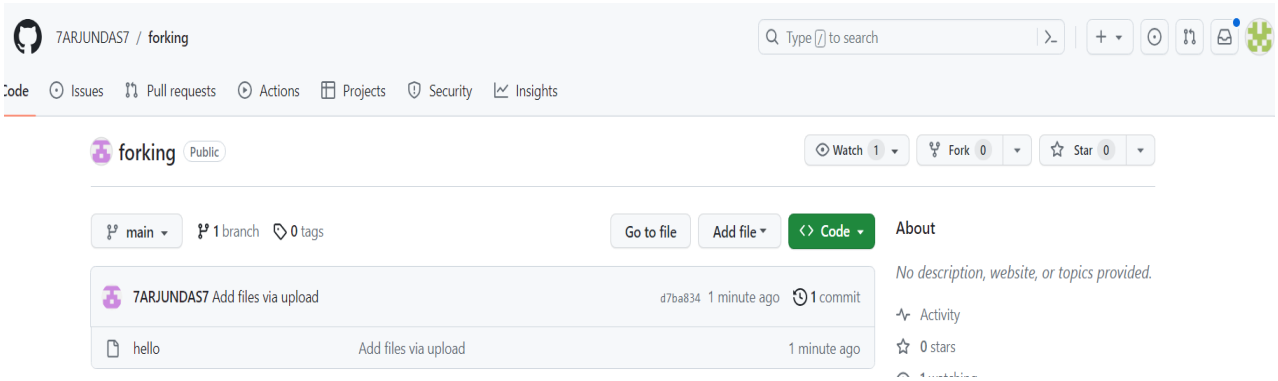
after doing this it will look like this:-



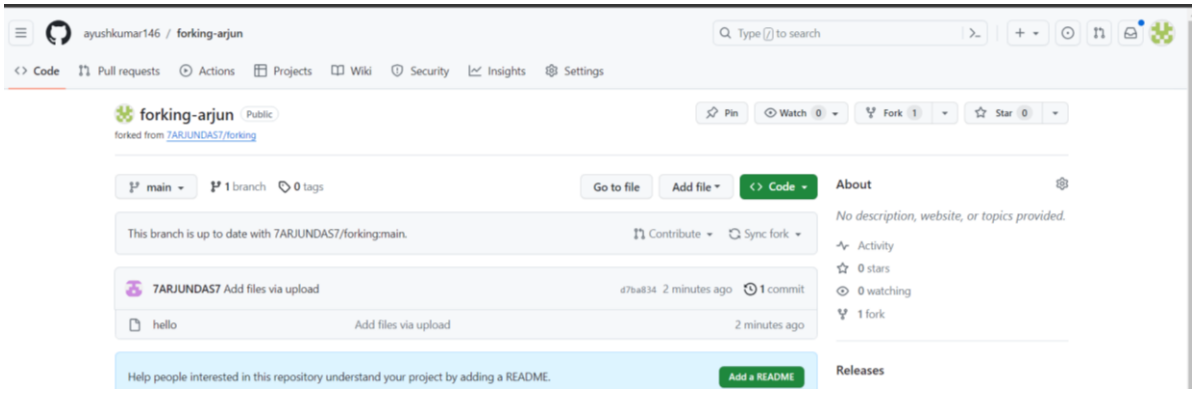
FORK

The url from which we forked a project is termed as upstream url by convention.

We do fork by clicking in fork on someone other repo:-



Then in same repo is formed in our account also like this:-



PULL REQUEST

When I make another branch other than the main branch on a forked repo create certain files and make some changes, then I want my branch to merge with the main/master branch then I do pull requests and whether It will be the part of main branch or not it depend upon the owner of the repo.

NOTE:-NEVER MAKE CHANGES IN MAIN BRANCH OF FORKED PROJECT