**GIT AND GIT HUB**

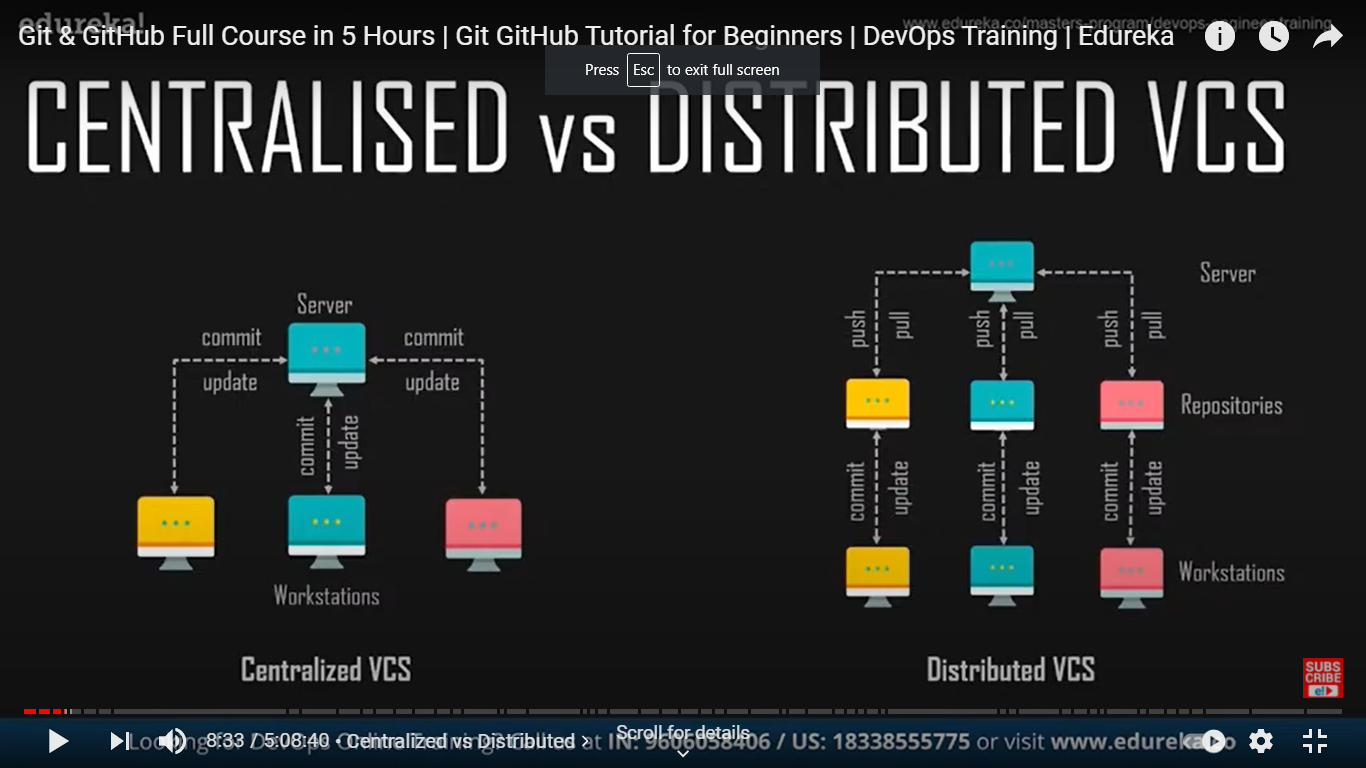
git🡪 version ctrl sys or source code ctrl system

version ctrl sys 🡪 three type local, centralized and distributed

local🡪 stores info about the repositories locally on a machine

centralized🡪 stores all info in a centralized platform

repositores 🡪 we can create repository for a folder or director then store everything in the rep, the files can be anything like image audio video files

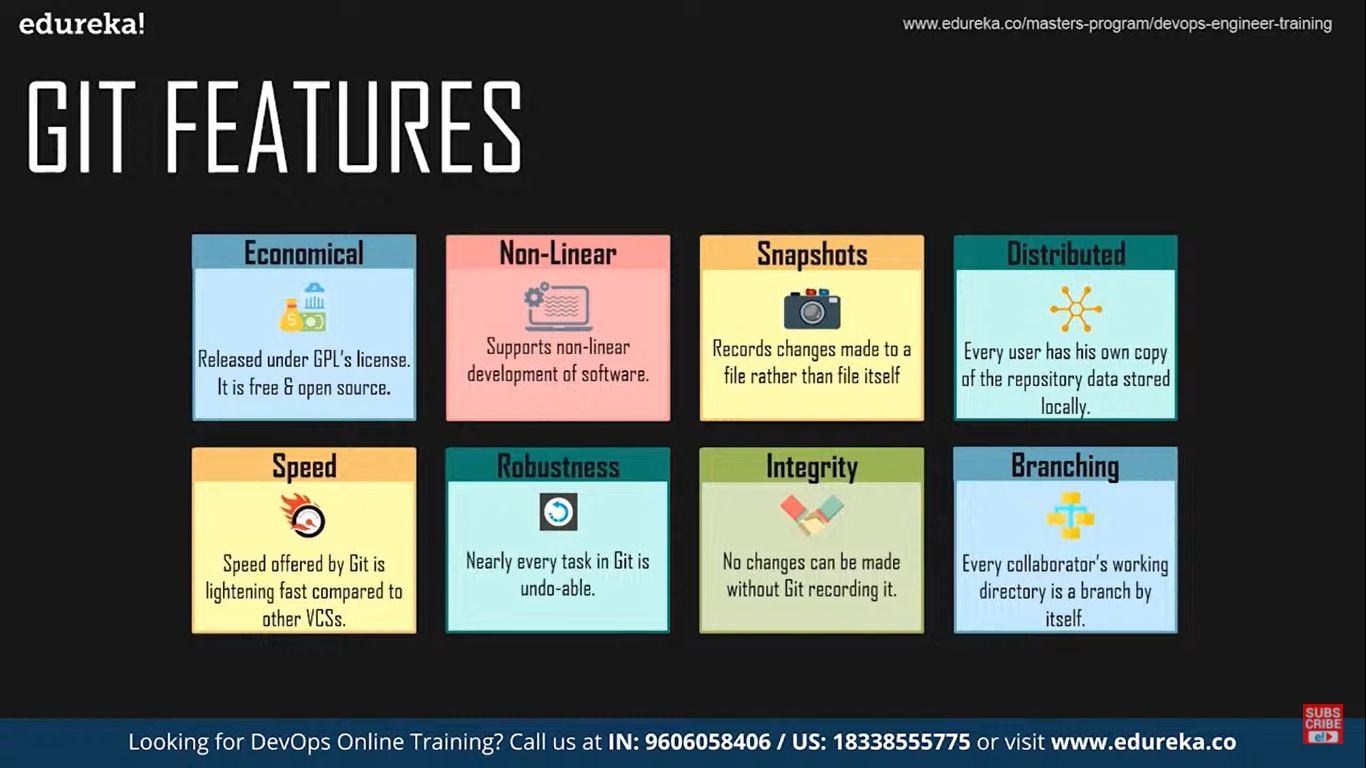


central server 🡪 a src code is managed there, then diff computers remotely connect to the repositors

a user should connect to the server update the code and commint the changes

distributed🡪 each user has a local repo once doing all updation the file can be sent to the central server

GIT🡪 distributed version ctrl sys, u take a repo work on it then share it to get, where others can use your code or file



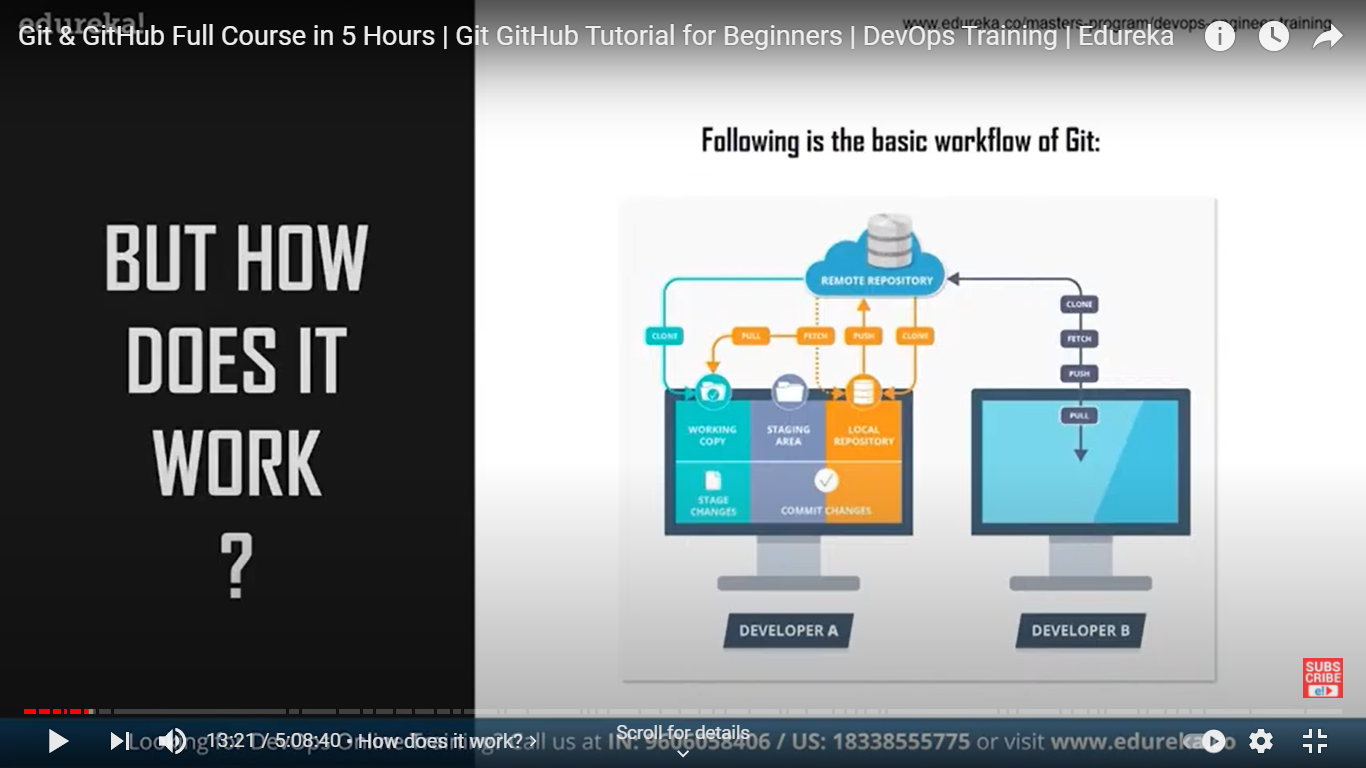
local repo🡪 in you computer

remote repo🡪 available for every one

first🡪 work in a file

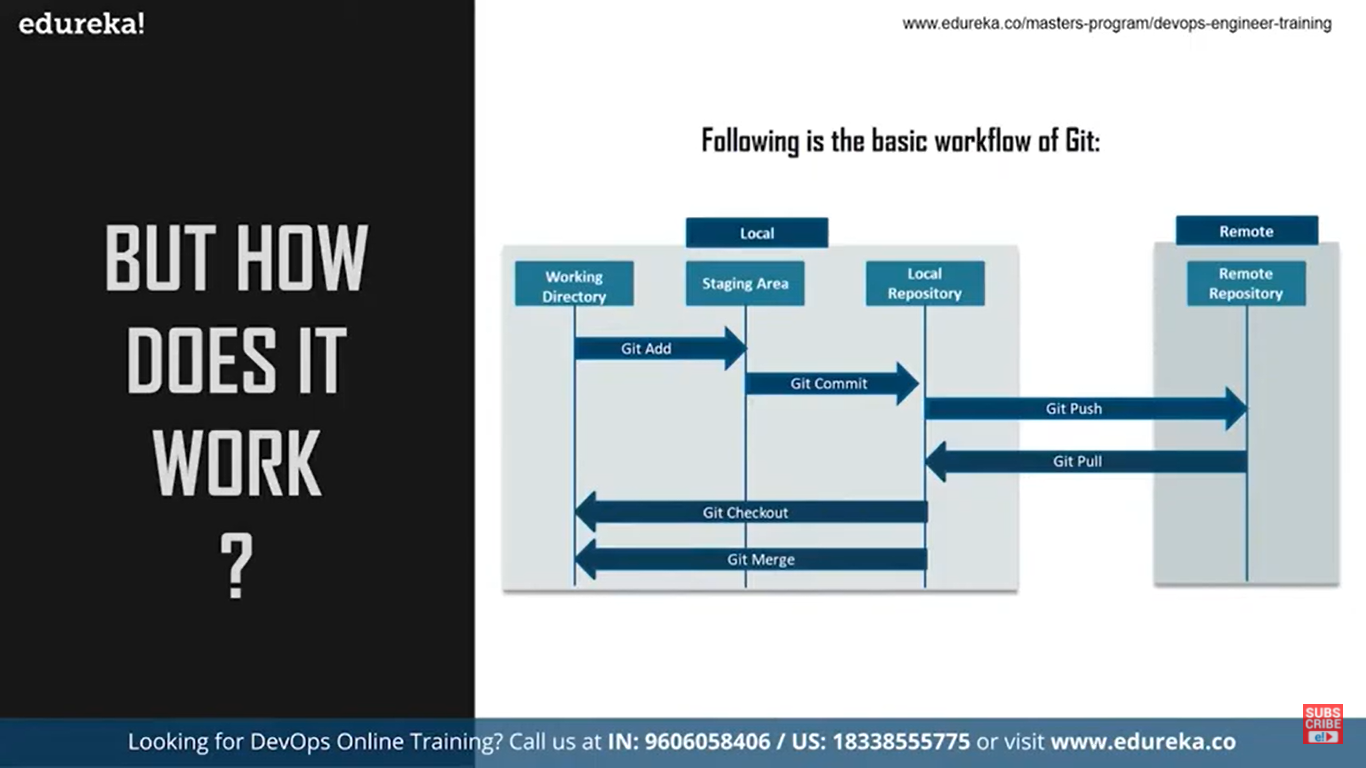
save the changes then commit the changes so that all the changes will be stored in local repository

then clone or push the local repository to the remote repository, we can also clone or pull data from the remote repo to local repo

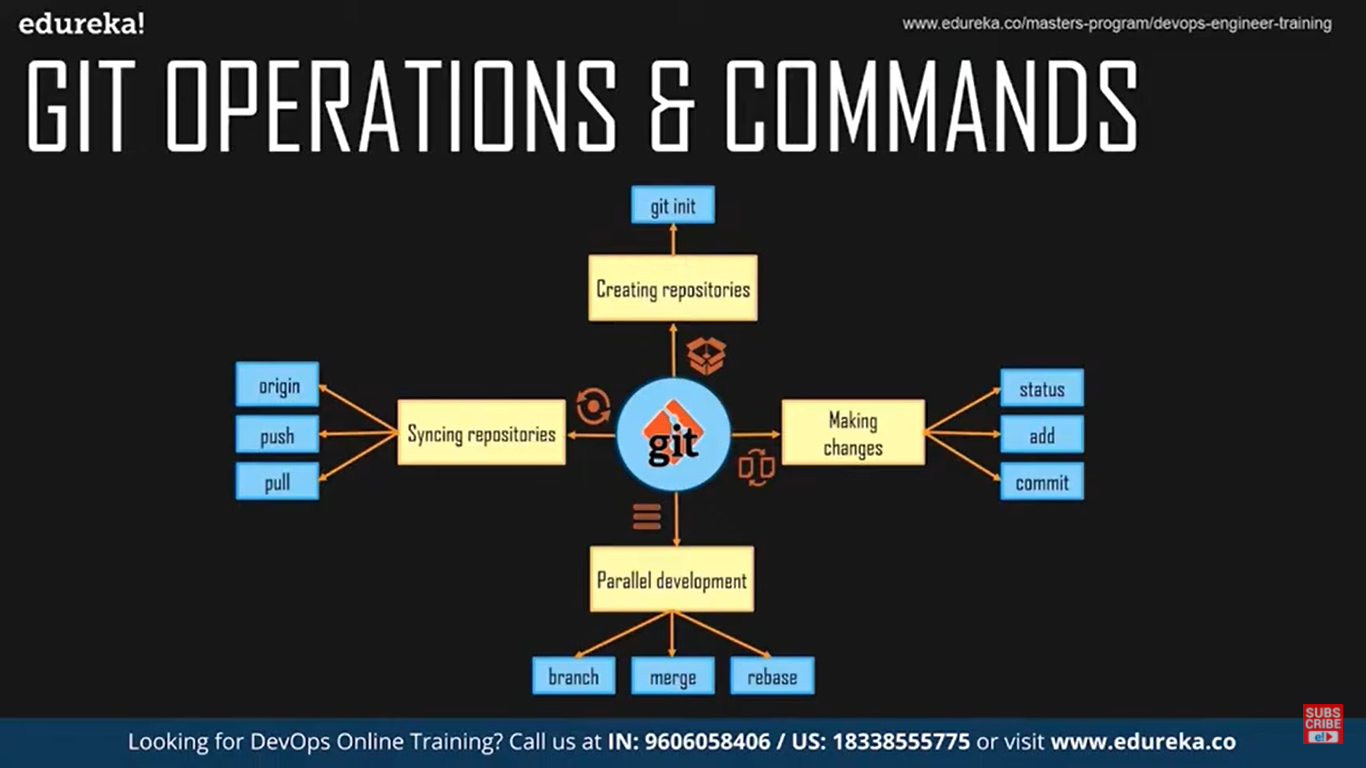


basic work flow

only when we commit the changes are pushed to local repo, if not t is present in a temporary area (memory) called staging area, after pulling any file from remo repo we can merge them with what files we have

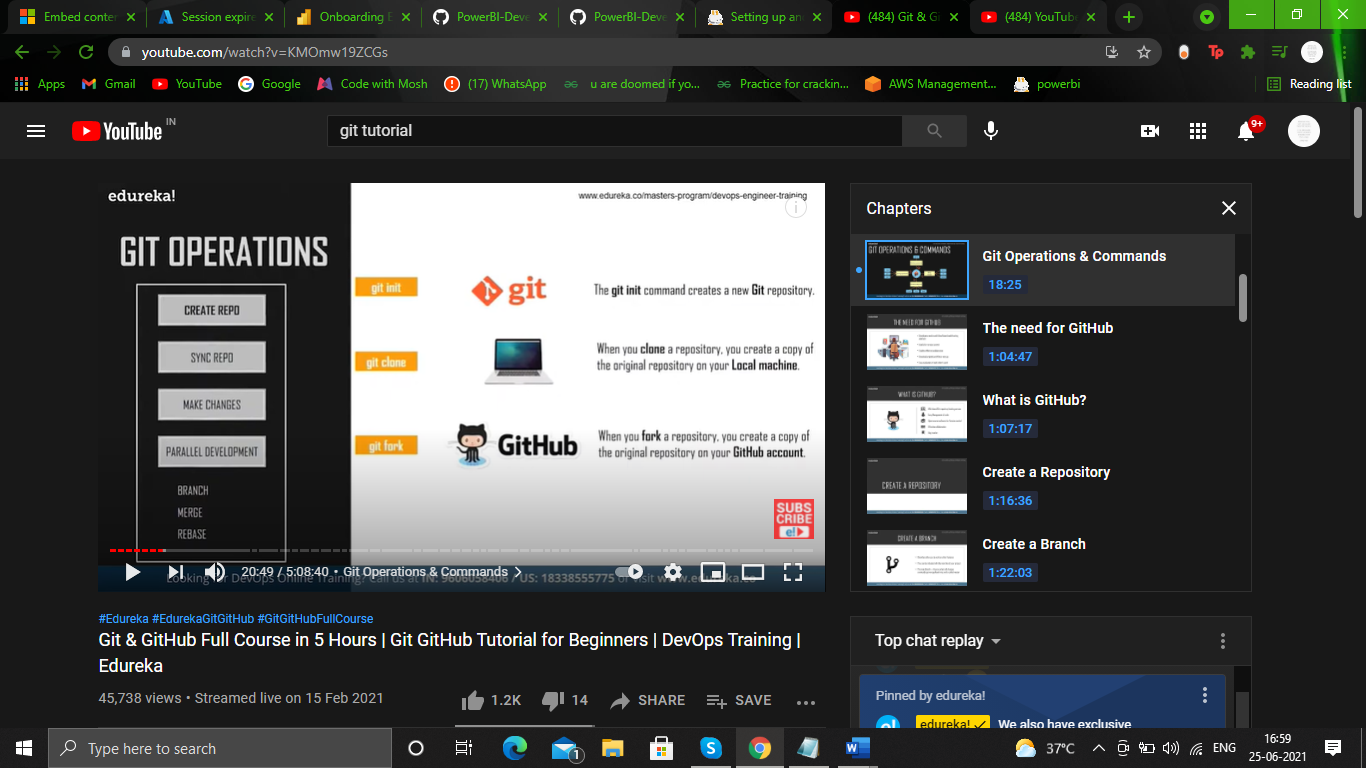
\

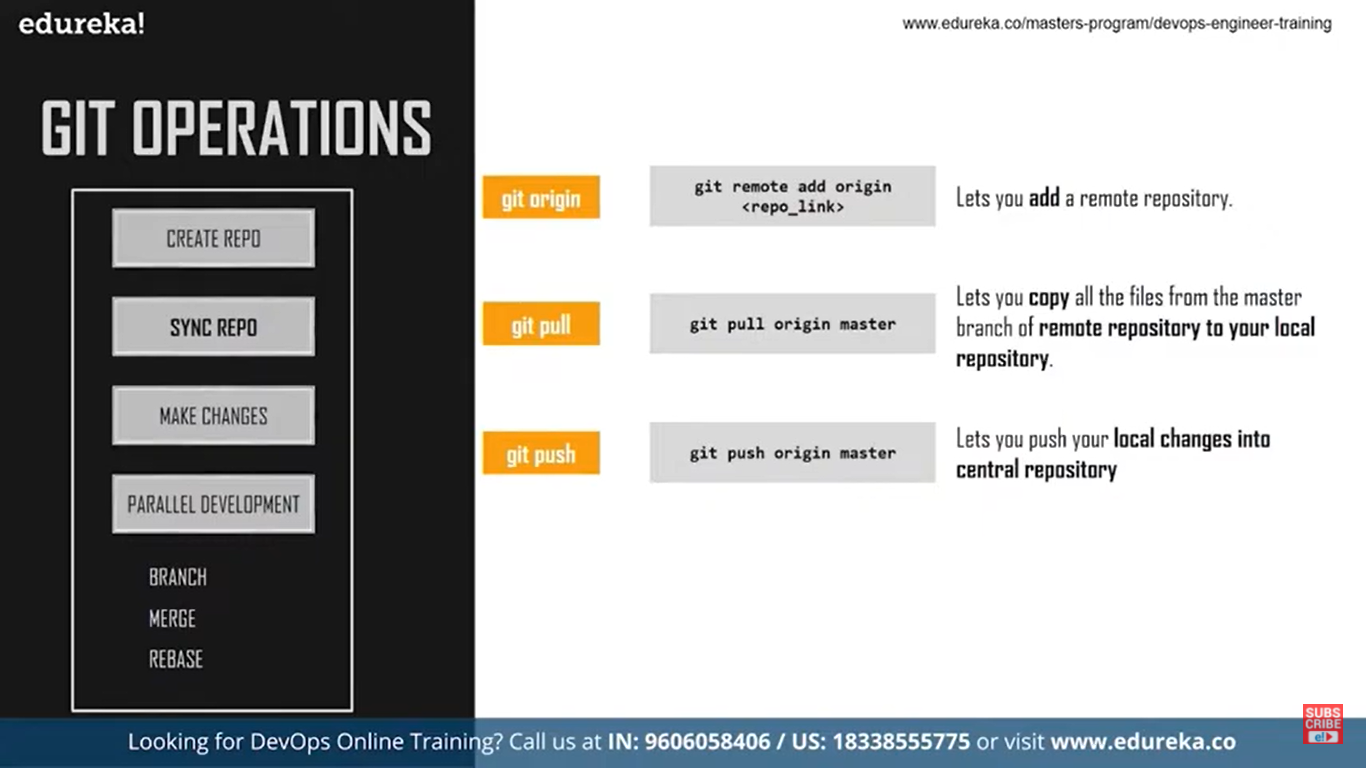
COMMANDS



**init🡪 to create repository in your pc (local)**

**branch🡪 to work parallely with multiple workers**

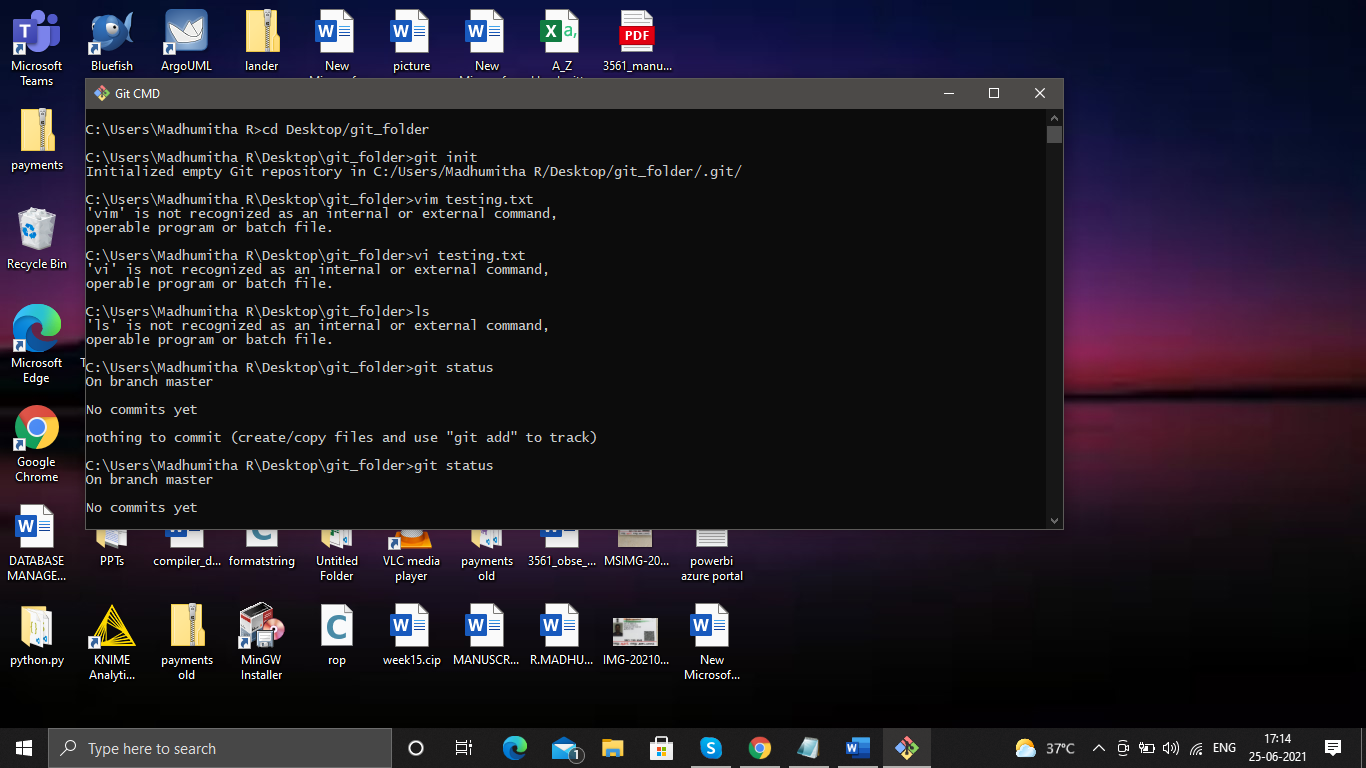




origin🡪 to add a remote repo to our local repo(name of the remote repository)

to configure a git repo, so that any changes made will be reflected we can configure the global variables like name and email

* git init
* git config –global user.name “somename”
* git config –global user.email “email address”

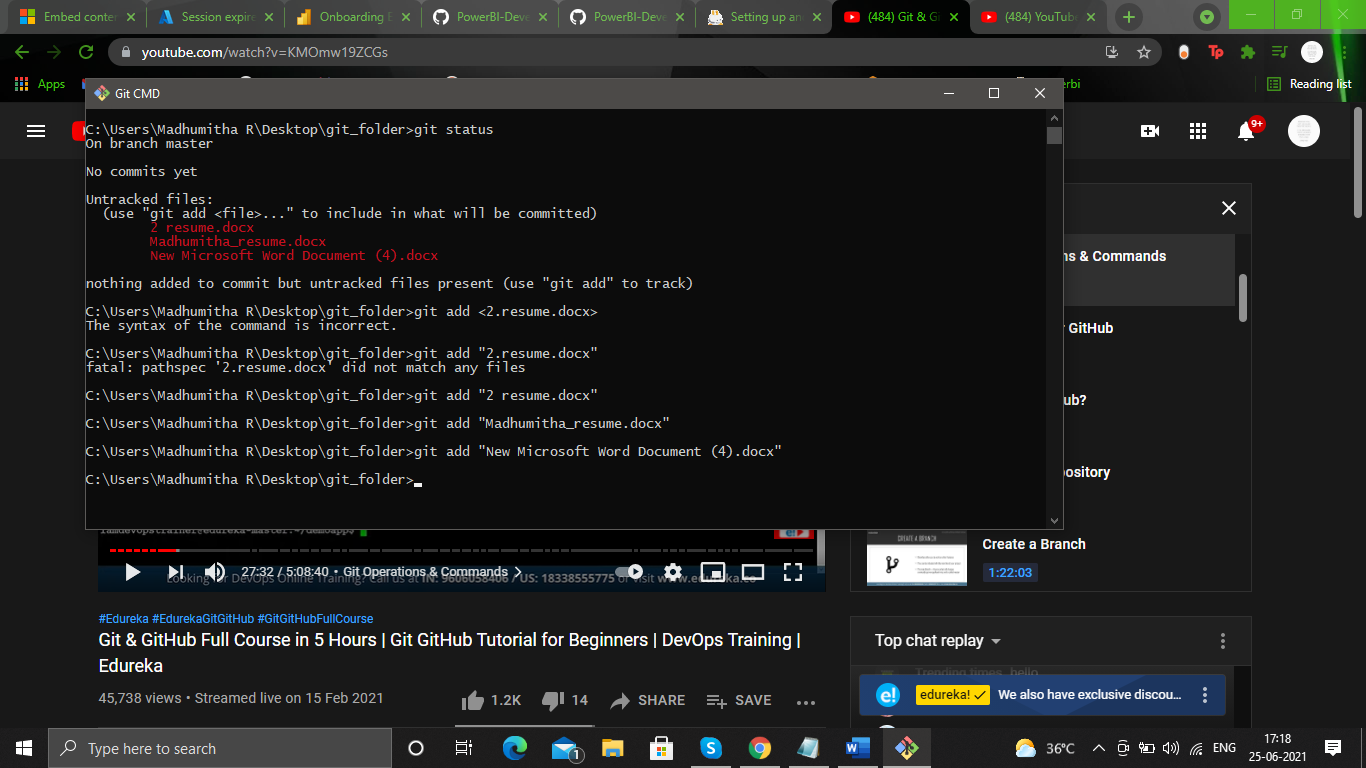


open git cmd 🡪 then go to the directory you want to create a repo

1. **give the git init comand**
2. a git folder will be created in the folded
3. then add files or anything in the folder
4. the git will keep track of it, but first we need to make sure that all the files we added newly must be moved to stage area to do so give
5. **git add “filename with extension”**
6. **git add file1.txt file2.txt fil3.py** 🡪 multiple files separated by a space
7. **git add .** 🡪 to track all the files
8. **git add ‘\*.txt’** 🡪 all files with extension txt

every time we add a new file, the git doesn’t track it so first put the file in the staging area to be tracked by git

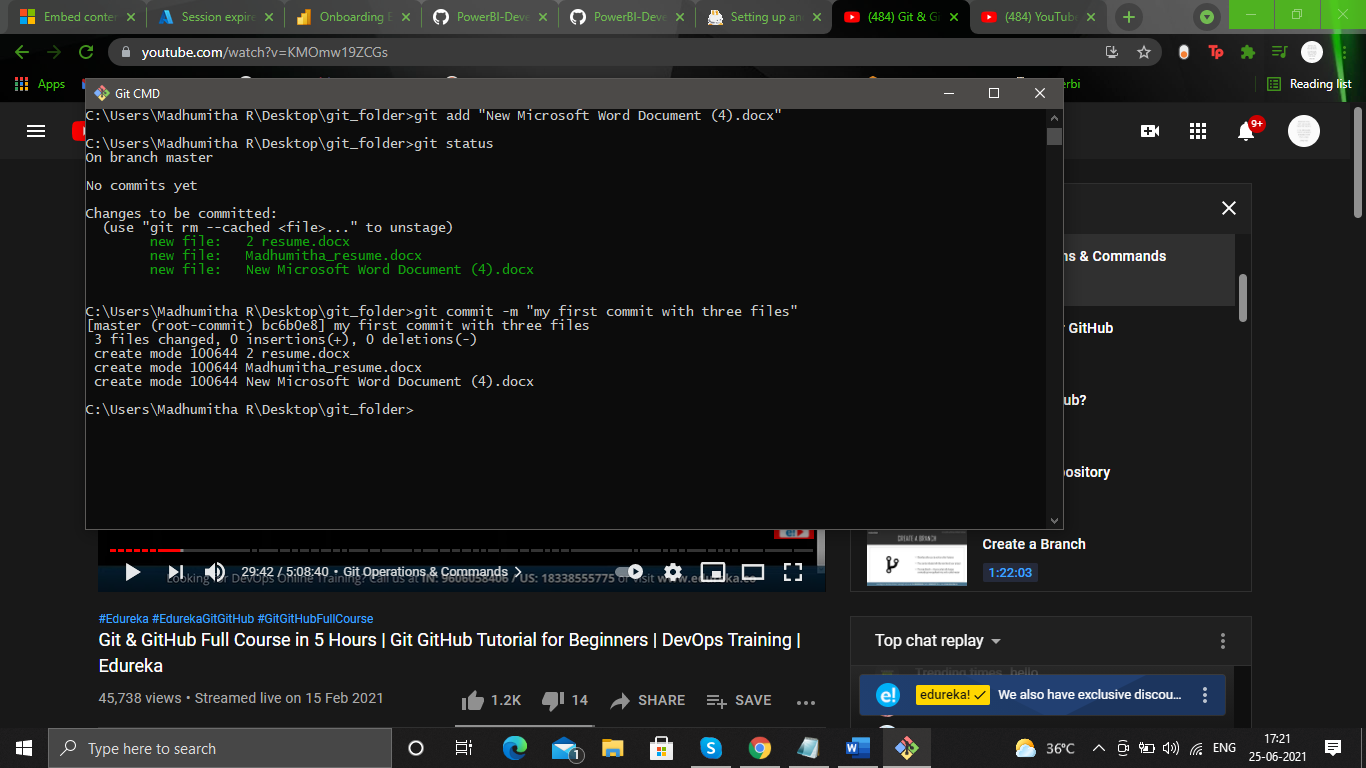
**now the files will be tracked git status will give the list of the files that are untracked**



**now all files will be tracked**

as soon as we commit the file, in the stage area the file doesn’t vanish, it stays there until the file is changed or deleted, the atage area is a temporary storage area so we save the files in that area before committing so that we can review once before we commit,while working if we find that we need to bring back what we did by mistake we can take it from stage area,

**COMMIT**



**git commit -m ”some name for the commit”**

**-**m 🡪 message

now all the files will be committed to local repository, a commit id will be generated keep a note on that so that we can reverse our changes easily

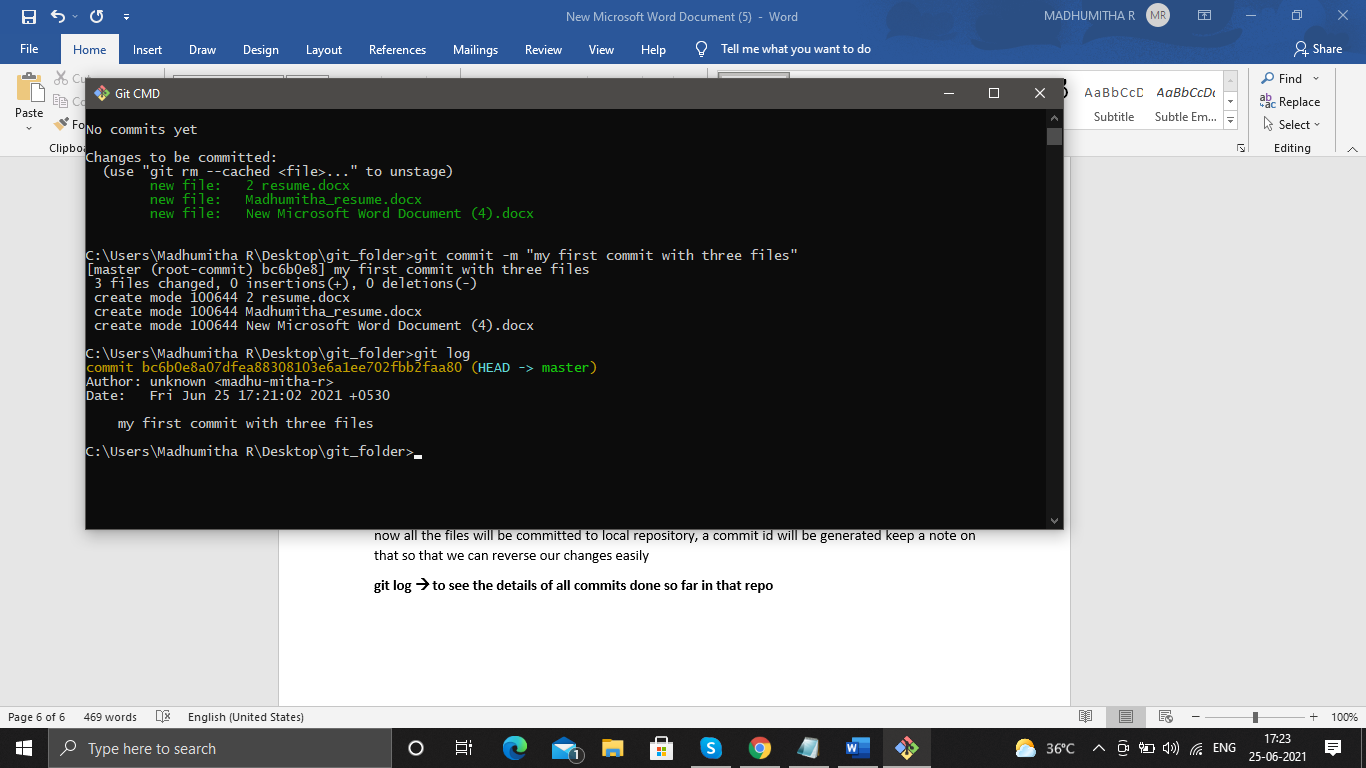
**git commit -am ”some name for the commit” 🡪 to commit all modified files**

**git commit🡪 will open a text editor we can give a description about the commit there and save the changes too**

**git log 🡪 to see the details of all commits done so far in that repo**

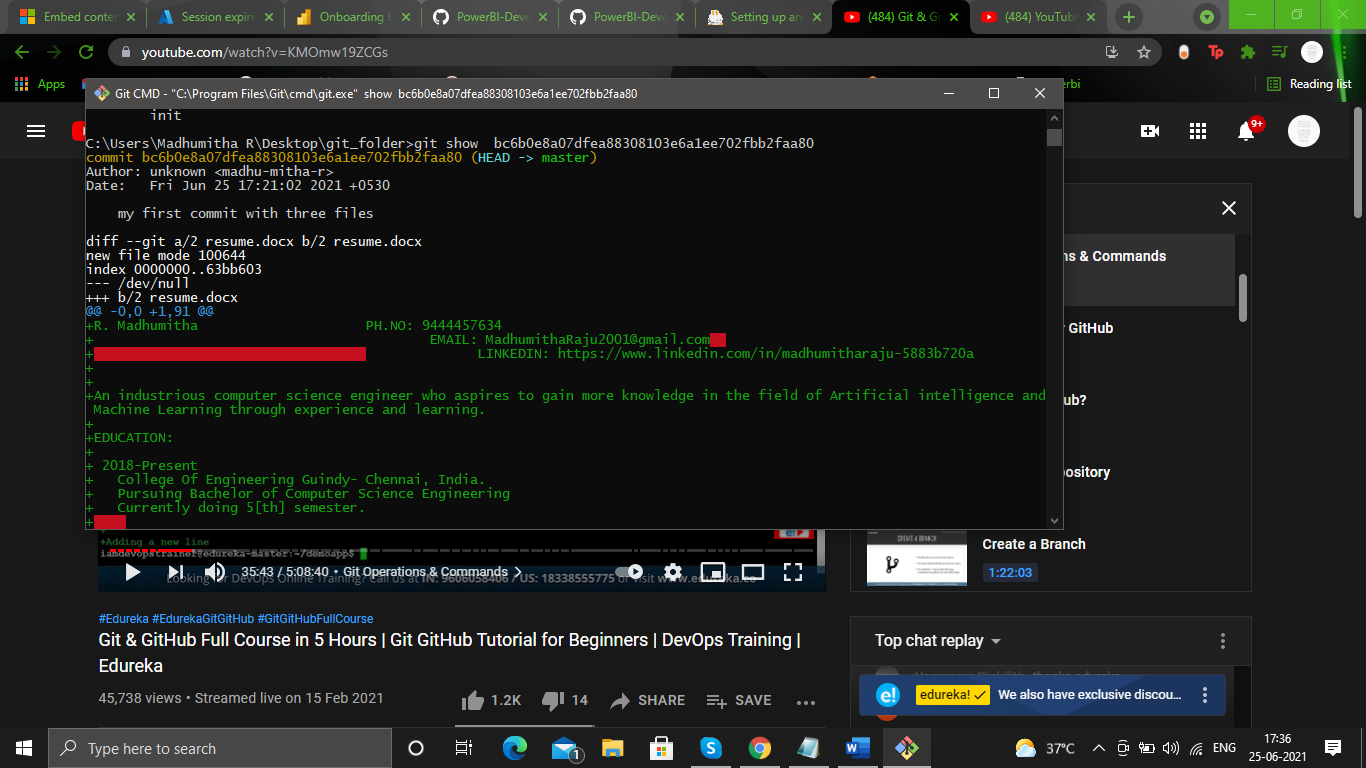
**git log –oneline🡪 to get the description alone for all the commits**

**git log --oneline --reverse 🡪 to reverse the order of commits from oldest to latest**



commit id the author and the name of the commit will be displayed

git show commit\_id 🡪 will give further info about the commit like all the places where the code or the content has been changed , but if a new doc is include then the entire content of the doc is displayed



if we give enter enter enter….. then all the content will be displayed

* **git ls-files🡪 list of all files that are being tracked (or the files in staging area)**



remember we can also commit a file before staging it but its not a good practice, we must always need a staging area

* **git remote add origin link of the remote repo**

will add the remote repo to the local repo that we are currently working on

for the remote repo create a repo in github and copy the link here in the command

**origin**🡪 alias name of the repo in the link,

so we add the repo in the link and the repo we are currently working at, using git

* **git remote -v**

to check if the repos are linked, we use this cmd

* **git push origin master**

so we are pushing the repo we are working to the remote repo using the aias name origin for the remote repo , we are actually pushing the master branch, to the repo, a master is a branch that created by default while we create the local repo

**REMOVING FILES**

delete it manually, so that the deleted file is in the staging area and not in the workstation or working directory so give

**git add deleted file name**

this will remove the file from the staging area and the changes is not committed in the local repository

to commit the changes to loc repo

**git commit -m ”deleting unused files”**

**another method:**

**git rm file1.txt file2.txt**🡪 removes the files from the **staging area** as well as from the **working directory**, then we need to commit the changes.

**TO RENAME OT MOVE FILES**

**mv file.txt main.txt 🡪 normal linux command**

file.txt is deleted and main.txt is added both changes should be committed and staged so

to reflect changes in stage area

**git add file.txt**

**git add main.txt**

to commit

**git commit…**

OR method two USE

**git mv file.txt main.txt**

**git commit…**

**IGNORING FILES:**

files that we don’t wanna share, or sync with others, these files aren’t tracked by staging area

* create a file called .gitignore this file has no name just an extension.
* add the files or folder that we don’t wanna be tracked to this file(.gitignore)
* so that the gitignore file will only be tracked not the files instead.
* move to staging are then commit the changes

**but if we track a file move it to staging area then put the file in gitgnore the git won’t ignore the files, instead it will track the files**

**type NUL > .gitignore** --> to create new file in cmd prompt of windows, use this cmd

**type NUL > filename.txt**

**echo hello world > filename.txt**

to put content in the filename remember the content that already existed will be overwritten, its better to open the file then add contents to it **or use >> to append the data to file**

**to un-track a file that is being tracked or to remove a file from staging area so that we can add it to gitignore we can use**

**git rm –cached -r folder\_name**

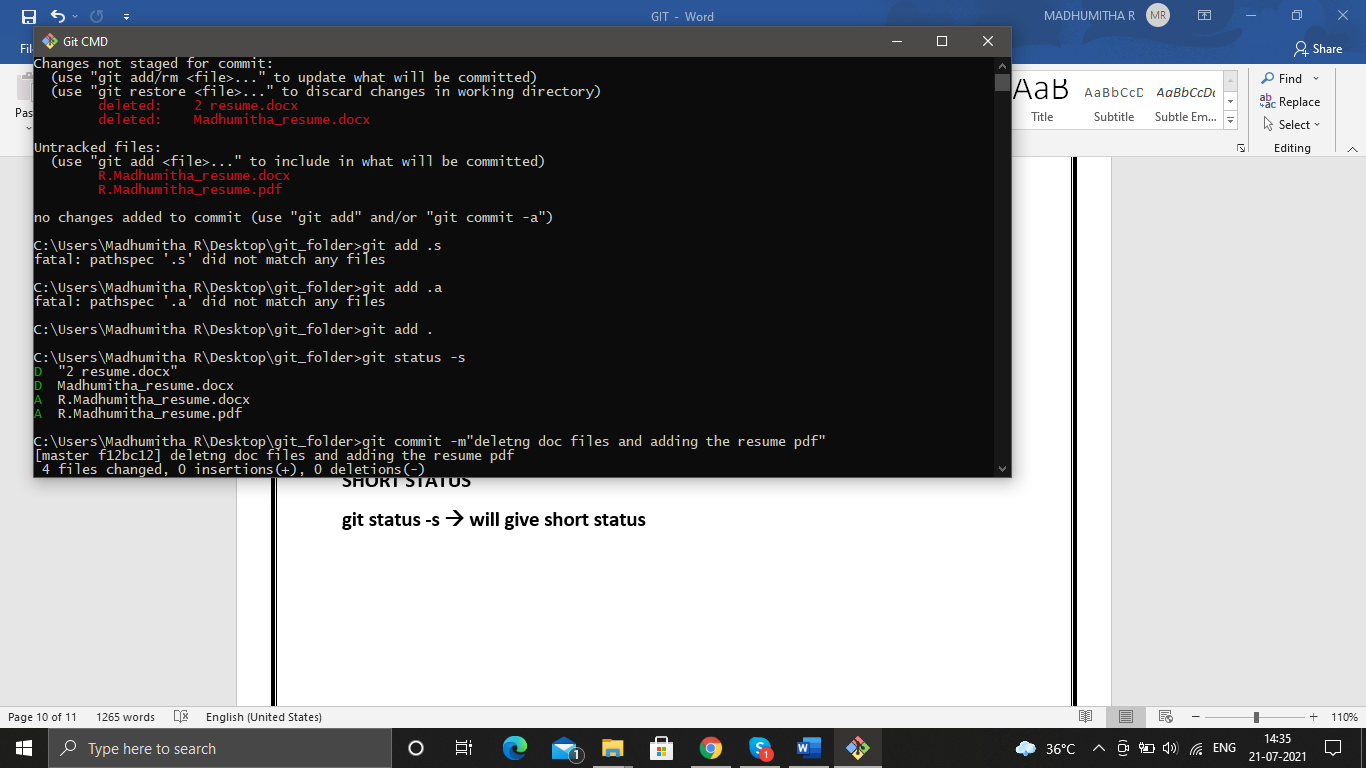
this will remove the folder from the staging area or index (index is the old name to staging area)

**-r** 🡪 means to recursively remove the files (because it’s a folder)

**git rm –cached file\_name 🡪 to remove a file from being tracked.**

**SHORT STATUS**

**git status -s 🡪 will give short status**



**MM somefile name**

**?? somefile name**

**the output has two columns**

**MM**

**??**

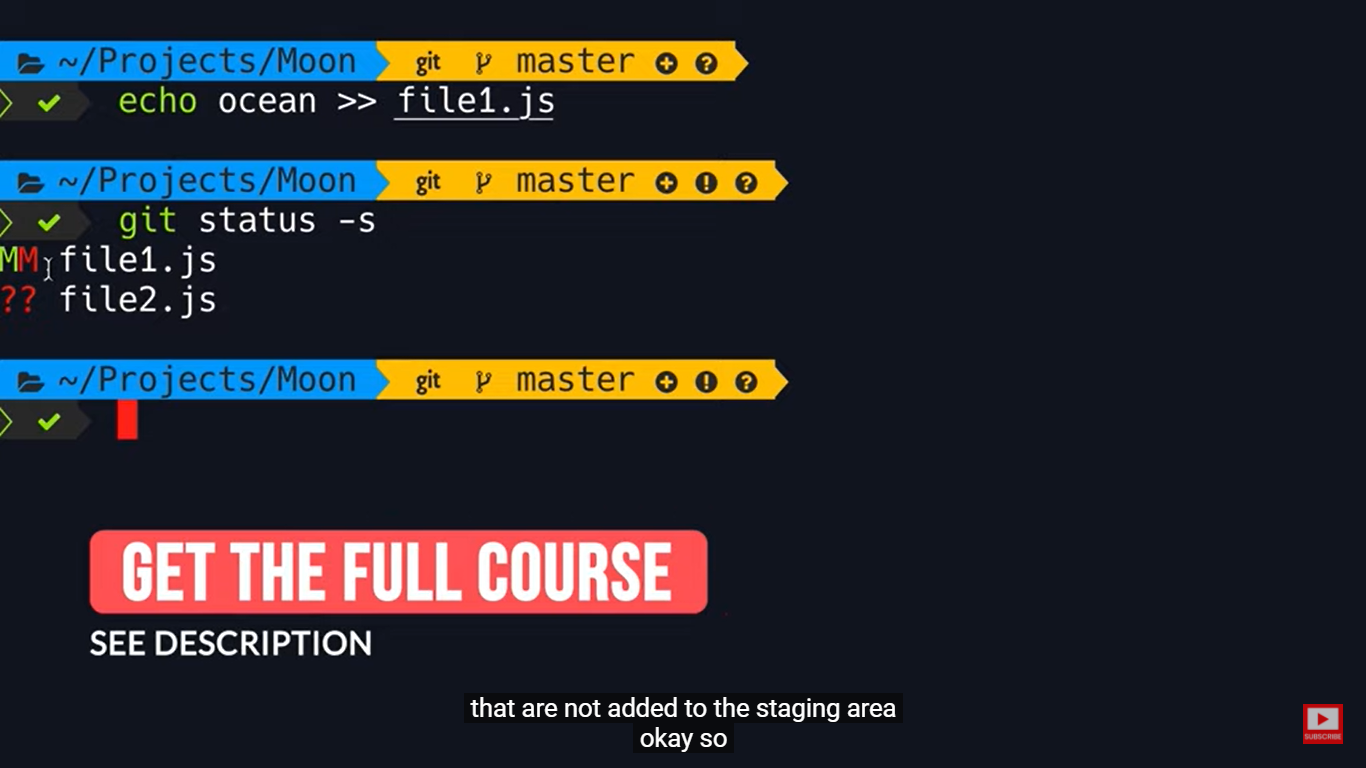
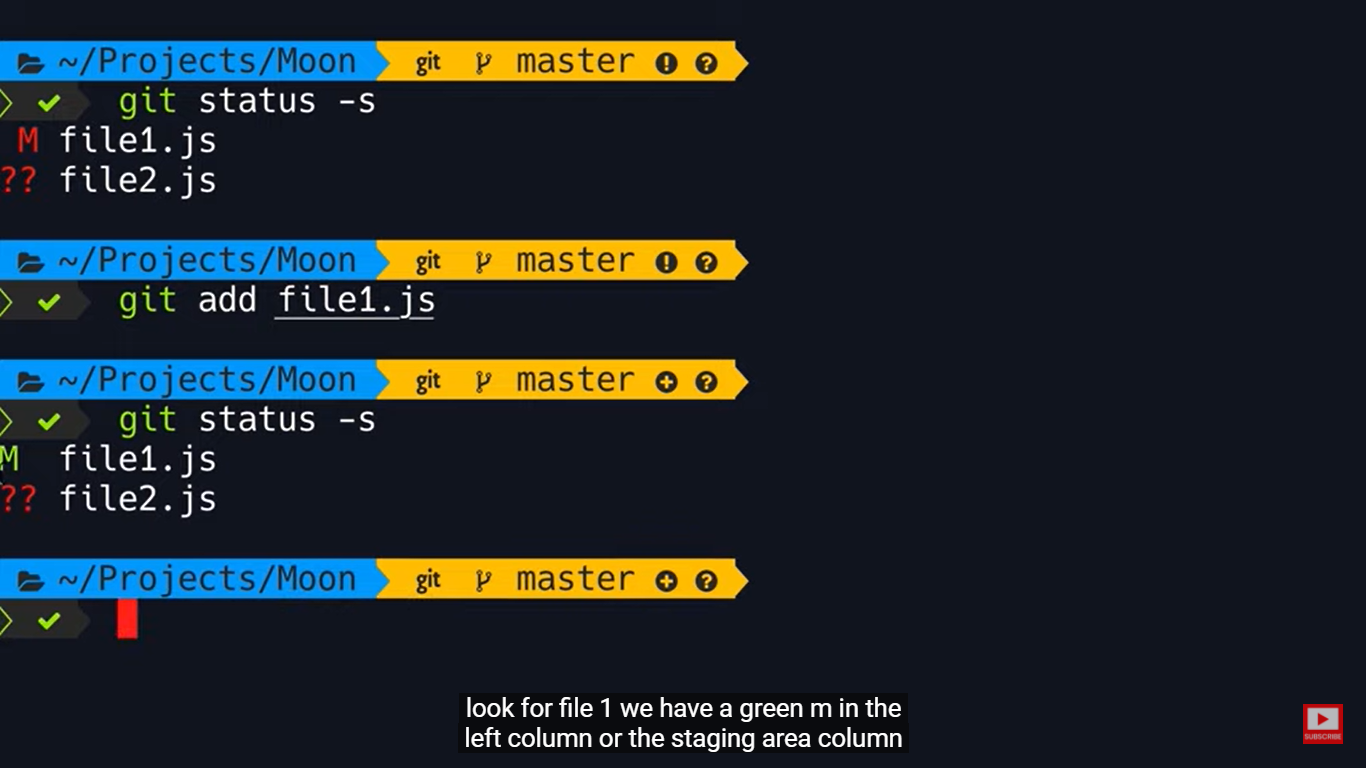
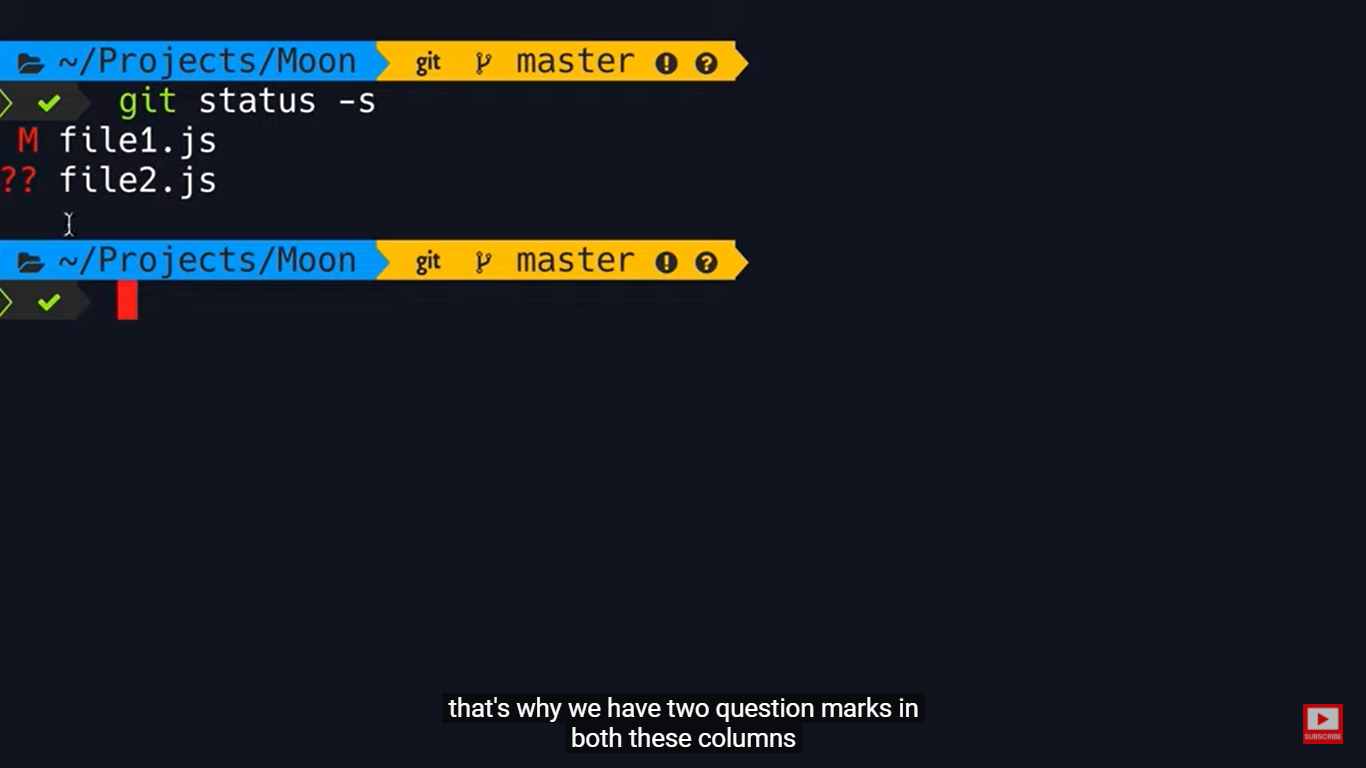
**first col for staging area and second for working repo**

**if M in second col – modified some file in the working directory**

**if M in first col – some modified file in working directory is tracked, but changes not yet committed.**

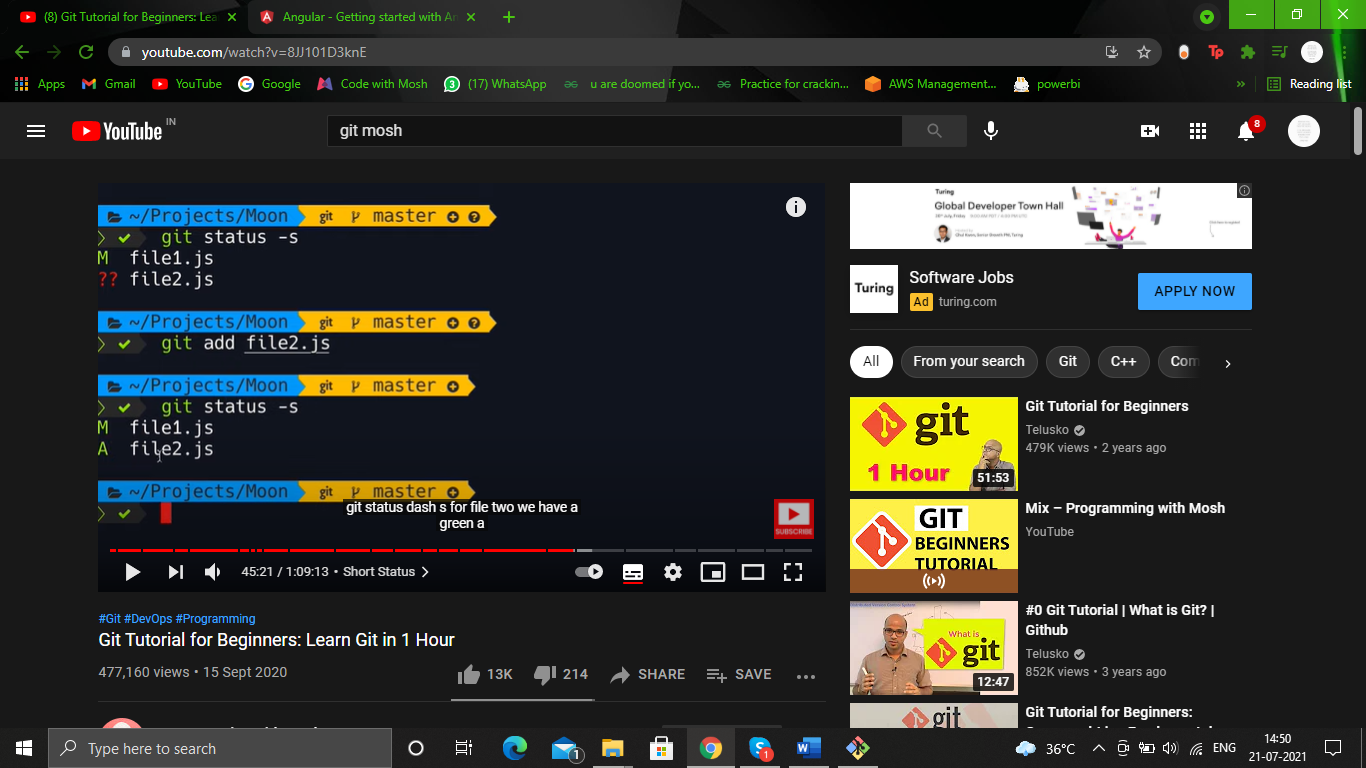
**?? some new file created in working directory and it is not yet staged too**

**so in both col we have a question mark**

****

**content added in file1.js, so modified in working directory**

**the same Modified file to be committed so M in staging area**

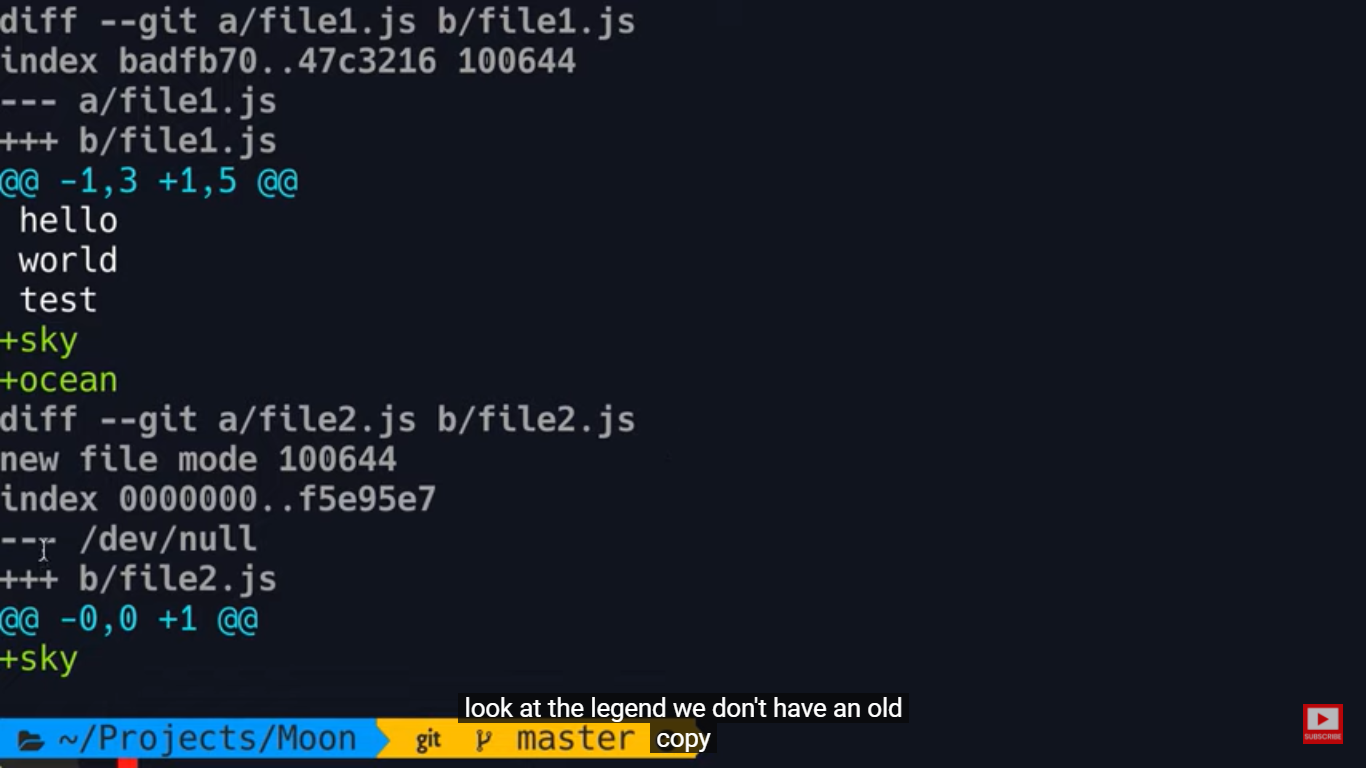


**A 🡪 added in staging area yet to be committed**

**TO VIEW STAGED AND UNSTAGED FILES**

**we should check the data or file before the file is committed.so use**

**git diff –staged 🡪** shows the difference between the files in the staged area and the local repo(staged files)



**output**

file1.txt🡪two different files of file1 are compared

a/file1 is in local repo b/file1 is the new modified file1 that is being tracked but the changes in the file1 is not committed.

then we have some metadata🡪 index…..

--- 🡪 indicates nothing has been added to the file1

+++ 🡪 something has been added to the file1

-1 3 🡪what we have in old copy,first three lines from the old copy(copy in local repo) is displayed

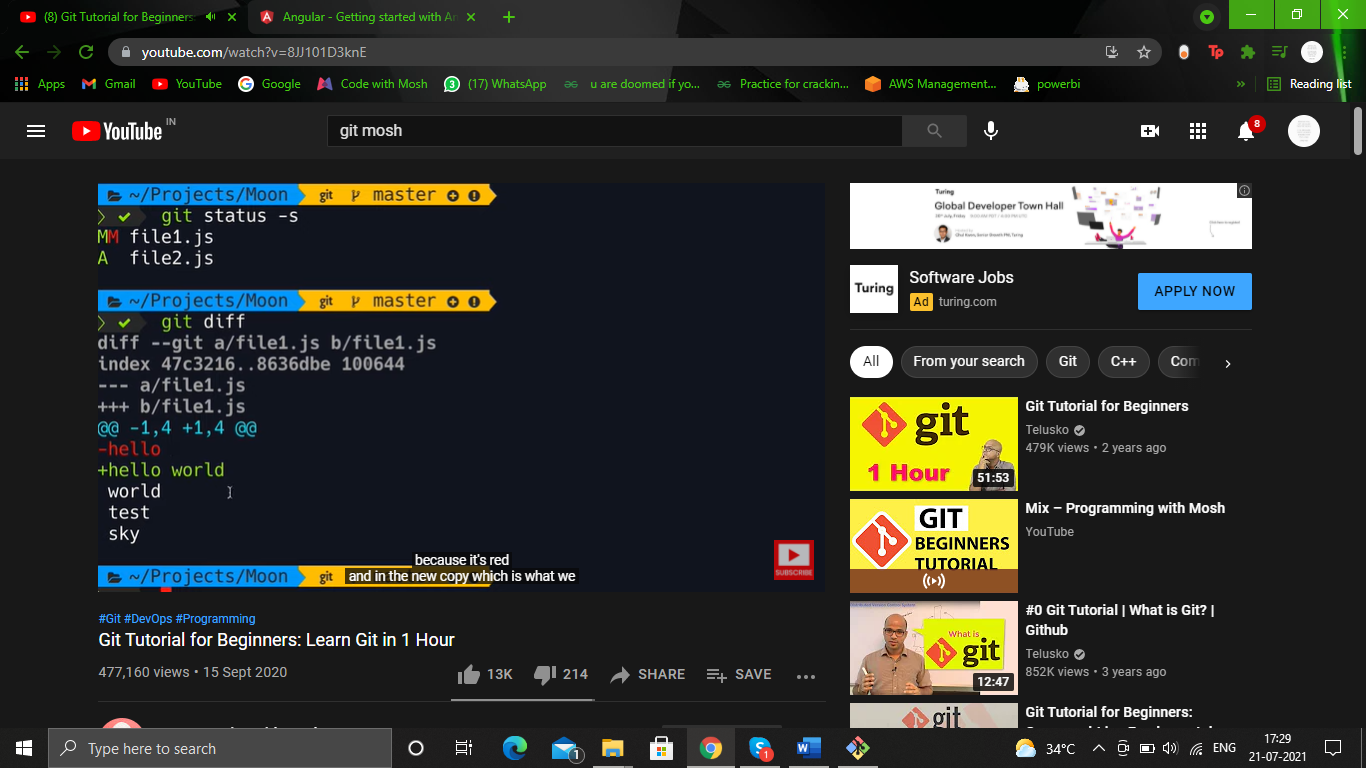
+1 5 what we have in new copy(copy in staging area) that is to be committed, first 5 lines from the new copy is displayed including sky and ocean as they area the new data added

+sky and ocean are the new data added

**in the example file2.txt it is the new file that is added, hence dev/null is there and also 0,0 lines are displayed from the old copy**

**To view the diff between unstaged and staged files, that is the files in the working directory and the files in the staged area use**

**git diff**



the two files that are compared here are the, files in working directory(unstaged files) and the file in the staged are(staged file)

a/file1🡪 staged area

b/ 🡪 working dire

first four lines in both the files are displayed

hello has been deleted and new word hello word has been added.

**Branching**

to keep old code safe, and write new code so as to enforce new changes in the code we do branching. basically dividing the work into branches. then working parallely

**git branch**

will display all the branches in that repository.

the branch in green, is the branch we are currently working at.

head branch🡪 means the branch we are currently working or active

master is the default branch that is created when we initialize a repository.

**git branch branch\_name 🡪** will create a new branch in that repository.

there are local and remote branches but we often work with local branches.

remote branches are in github or gitlab … or any central repository