Git - Open Source VCS software

GitHub – it is the company that hosts files on the internet using Git

**Windows CMD Command :**

Change directories – cd

Make a new directory – mkdir

List files in current directory – dir

Clear screen – cls

Difference between Git and git :

* Git typically refers to entire project and git refers to some specific code
* Git is developed by Linus Torvald in 2005 in 3 days

Download VS Code :

* Code.visualstudio.com

Download Git :

* <https://git-scm.com/>
* Install git on the computer. (follow video)

Github.com = github website

FIRST WE SETUP VS CODE -> THEN GIT -> THEN GITHUB ACCOUNT -> CONFIGURE GIT

Install Github desktop

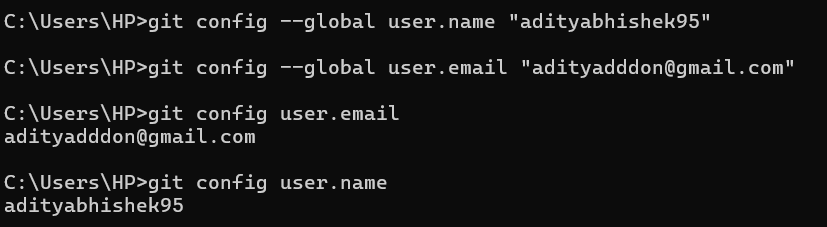
**Git Commands :**

**To check :**

* git config user.name
* git config user.email

**To config**

* git config --global user.name “user”
* git config –global user.email “email”

****

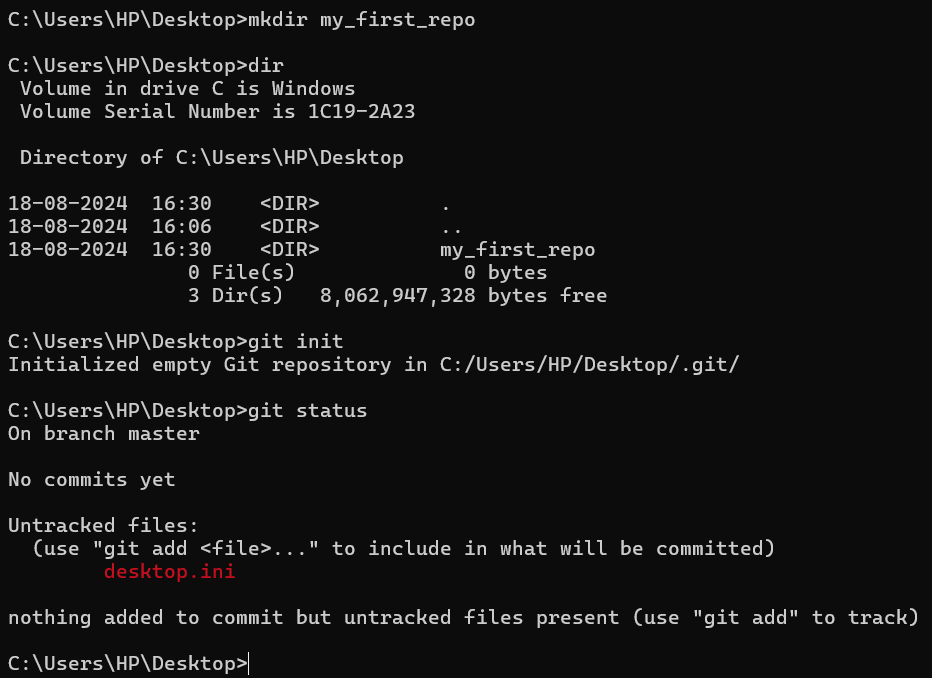
**Repository :**  The main place where we track and manage our files

**How to create a repository in Git locally:**

1. Open cmd
2. git init this will initialize a git repository in my local computer. We need to run this command once per project

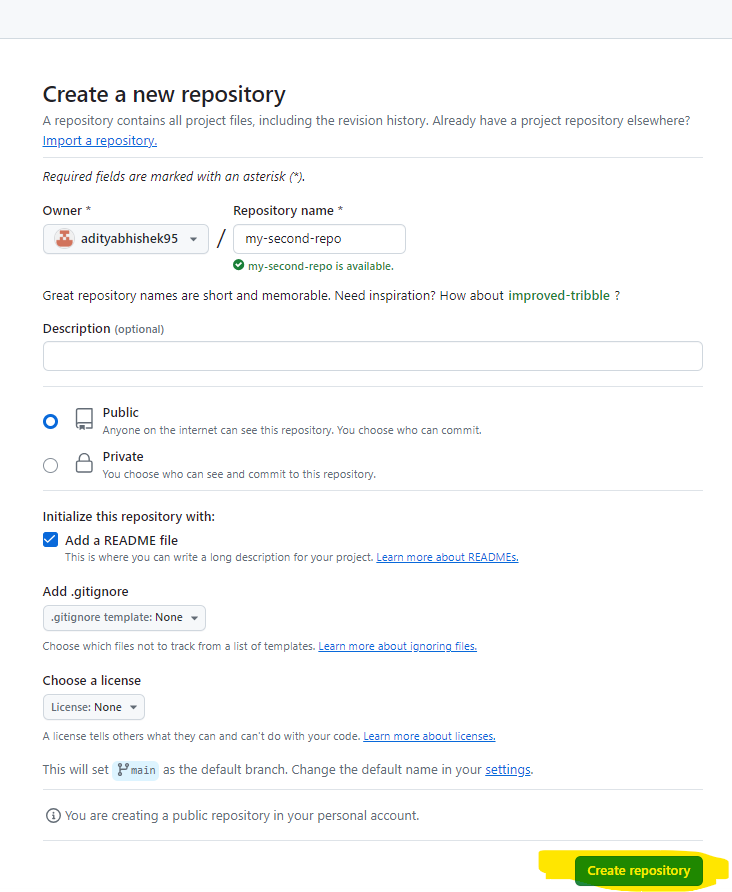
git status status of my repo

1. .git file will get created which will manage the version of the files inside
2. We should not run git init inside already created another repo inside. It will create a mess.
3. git clone to clone our local repo



**How to create repository in GitHub?**

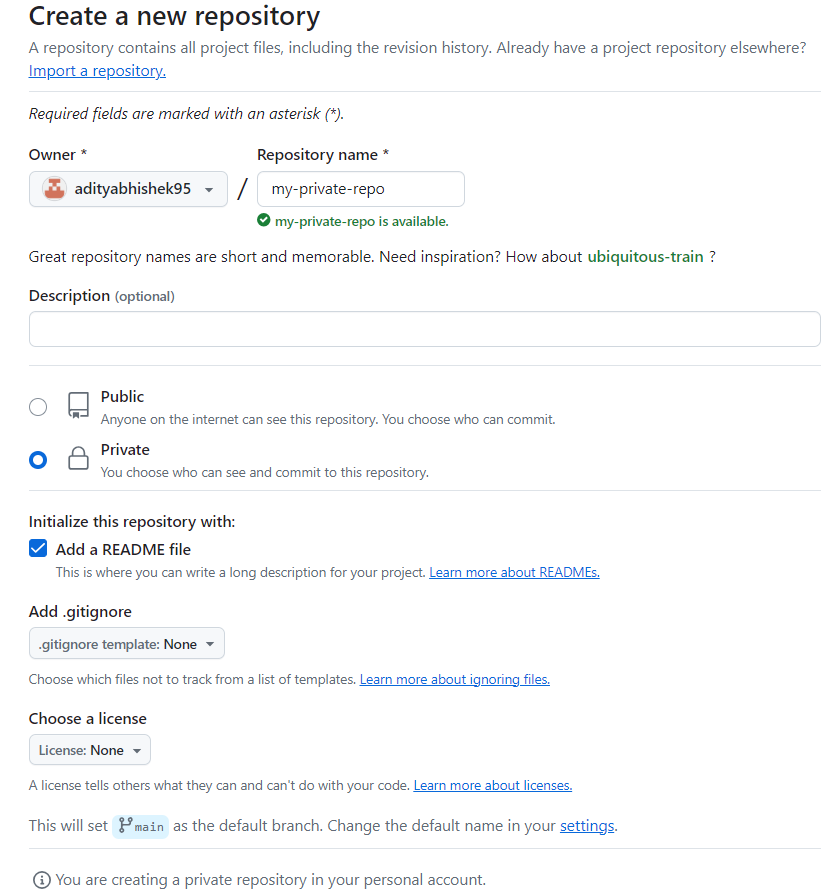
Go to github.com and then Create repository



How to clone this repo in my local ?

1. Go to code -> copy url
2. In cmd, come to Desktop since we do not want to create repo inside another repo
3. Open cmd -> git clone <https://github.com/adityabhishek95/my-second-repo.git>
4. 

How to create a private repo and pull it in desktop ?



How to create clone of the private repo?

* One way – we can create a PAT ( Personal Access Token ) on the Github, so that when we run our git clone command we can refer PAT.
* Second way – We can use GitHub desktop application.

How do we generate PAT in GitHub ?

1. Go to github.com -> settings -> developer settings -> generate PAT -> tick mark : control all private repo -> copy PAT

PAT : ghp\_KDUAostWb2rXmCC9GlfCBRBBI5ZO122htGOU

Now run this command in cmd : **git clone** [**http://token@url**](http://token@url)

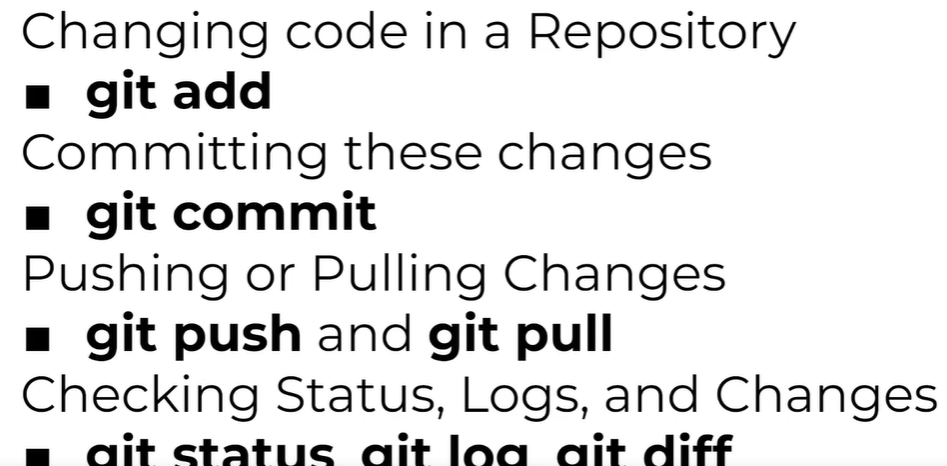
git clone https://ghp\_j4t4OQj0Q6lNnDdrSmGKQHmFJFGOjo4Vp1eX@github.com/adityabhishek95/my-private-rep

o.git

**How to clone via GitHub desktop?**

Sign in to github desktop -> click on clone repository option -> select repository and location -> hit enter

***DAY 2***



Concept :

Working Directory - - - > **[git add program.py]-** - -> Staging Area - - -> **[ git commit ] -** - -> Repository - - **>[git push]-** - -> GitHub website

We may have many files in our working directory. If we want to push them to github, we have to follow above process.

1. Create a folder under C:\Users\HP\OneDrive\Documents called LEARN\_GIT. This is not yet repository.
2. Open VS Code and open folder LEARN\_GIT from option
3. Open Terminal -> Command Prompt
4. Go to this folder and run git status (it should show fatal as this is not yet repository)
5. Make this a repo -> git init
6. Create a file inside this folder LEARN\_GIT -> first-gile.txt
7. If we do git status, we will see one file untracked
8. Git add first-file.txt, this will pass the new file to staging area
9. git commit -m "i just added first-file.txt from staging area to repository"

this will add first-file.txt to repo

SO WHAT WE DID IS, WE CAN HAVE ANY FILE INSIDE A REPO. TO ADD THAT FILE TO THE REPO, WE HAVE TO FIRST PASS THE FILE TO STAGING AREA AND THEN COMMIT TO THE REPO ( SO THAT IN FUTURE WE CAN HAVE THAT IN GITHUB)

How do we git add multiple files ? git add . file1 file2 path/to/file/location

If you accidentally created any repo, we can simply delete.git folder and the folder will no longer be a repo!!

How to push local code to remote repo like GitHub?

1 git remote -v to view the url of any cloned repo we are in

2 git remote rename <old> <new>

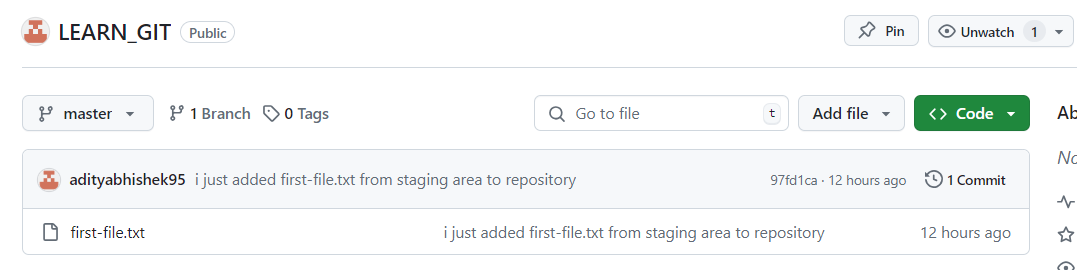
3 git remote remove <name> this will separate the connection from the local repo

4 git push -u origin main/master

5 git branch -M main this will change the branch name to main

How to connect a local repo to the GitHub repo ?

1. create a repo in GitHub
2. open in CMD local where repo is present
3. git status
4. git remote -v this will also show nothing since no repo has been connected
5. git remote add origin <https://github.com/adityabhishek95/LEARN_GIT.git> ---> but we cant push anything like git push -u origin master
6. so we will do 🡪 git remote add origin <https://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@github.com/adityabhishek95/LEARN_GIT.git>
7. Now we will do : git push -u origin master
8. Now we can see in GitHub, all the files that were under local LEARN\_GIT :



IN GIT/GITHUB’s language we call local repo as master (newly aka main) and GitHub repo as origin

How to pull from a remote repository to local repo?

1 Create a file in remote repo by clicking “create file”

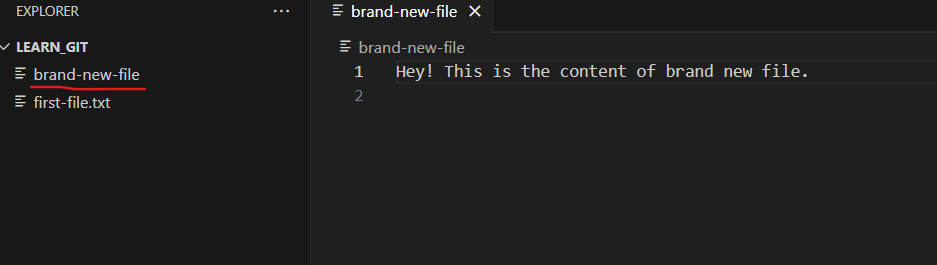
2 Come to CMD and go to the linked repo 🡪 remote repo folder and write : git fetch

This will only show the file in remote origin repo but actually the file has not been written to your local directory.

3 git checkout origin/master – You can checkout the files in the origin remote repo

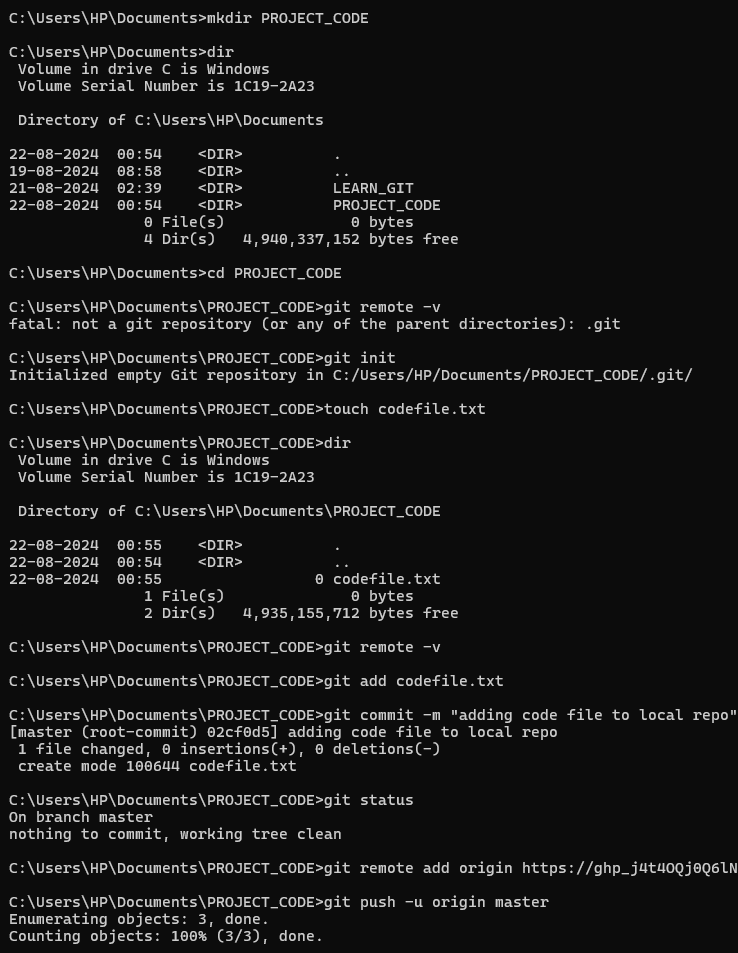
3 git switch master – This is switch to local repo master branch

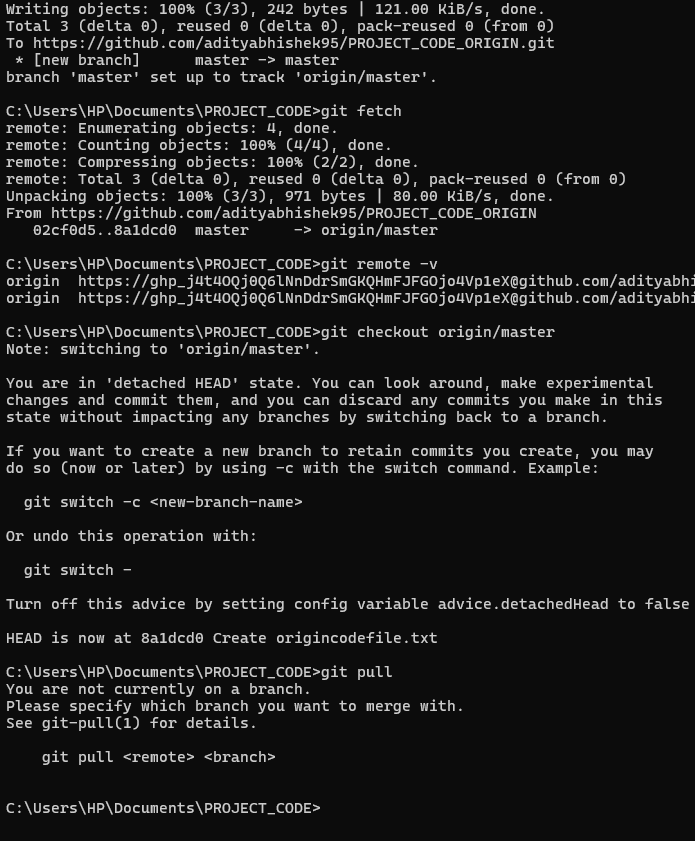
4 git pull – it will write the remote file to your local directory.



Assignment –

Create a local repo -> add a file -> add that file to local repo -> create a github repo -> connect local repo to origin repo -> push file -> create a file in origin repo -> pull that file to local repo





*DAY 3*

Branches can branch out from the main development workflow and later merge into one :

Box 2 will have Box 1 commit address because we hit commit on box1 only.

Branches and HEAD

HEAD just refers to the current working branch and branch itself will contain pointer to the latest commit.

How to create a repo and a new branch and play with commits :

1 create a dir and later repo

* mkdir DAYTHREE and create a file myfile.txt
* write some line : LINE ONE, LINE TWO, LINE THREE

2 create another branch

* git branch this will show which branch youre in and what are all the branches available
* git branch new\_branch this will create a new branch

3 add the file to staging area and then commit

* git add myfile.txt and git commit -m “my first commit”

4 the HEAD is still pointing to master branch which means this commit has been done under master branch only. When we will switch to new\_branch, these commits ( or content ) of the file we wont be able to see

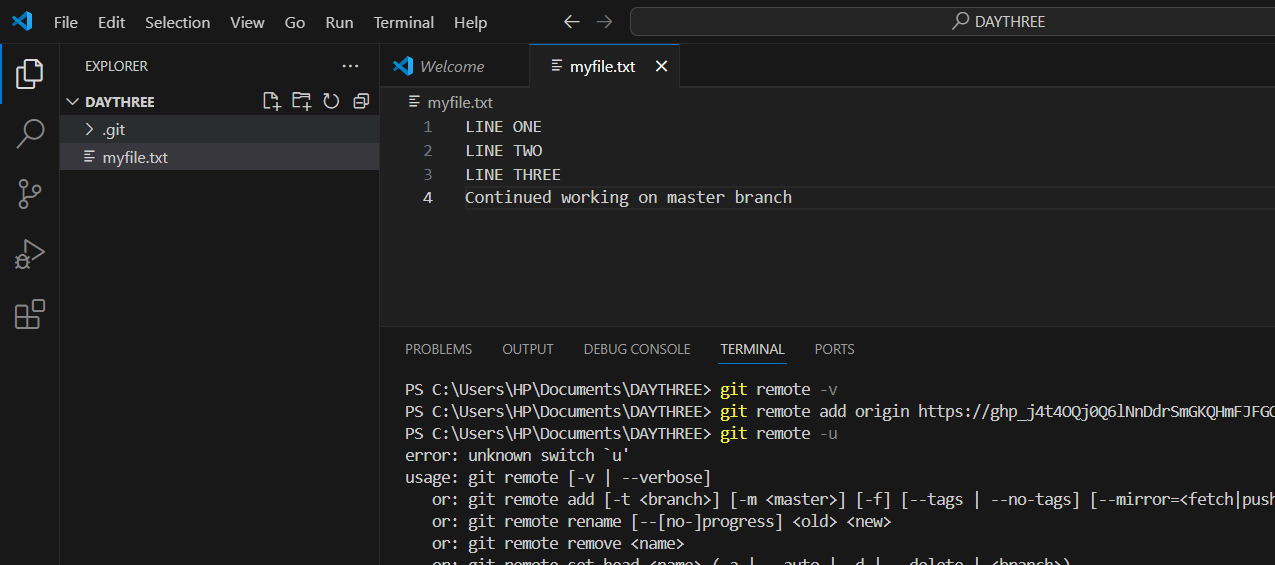
5 switch to new\_branch and add few more lines

* git switch new\_branch
* add LINE FOUR
* git add myfile.txt and git commit -m “my new\_branch commit”

6 Now when we come to new\_branch we will be able to see only “LINE FOUR” text as this commit has been done after switching to new\_branch

7 to view git logs, status

* git log, git status

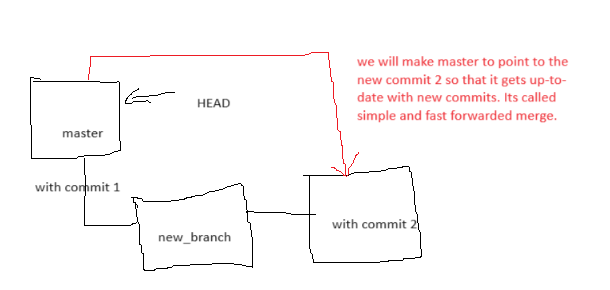


How to rename and delete branches?

* To rename a branch, we have to be on that branch!
* git switch experimental -> git branch -m need\_to\_delete
* To delete a branch, we MUST NOT be on that branch
* git switch master -> git branch -D need\_to\_delete

MERGE :

1 FAST-FORWARD MERGE – It means we have made some commit on the new\_branch and we are essentially fast forwarding the master branch to reach to new\_branch updated commits.



1. PS C:\Users\HP\Documents\DAYFOUR> git status

On branch master

2 make myfile.txt and add some text

git add myfile.txt

git commit -m "first commit from master"

git branch new\_branch

git branch =

\* master

new\_branch

add some more text in myfile.txt

git add myfile.txt

git commit -m "first commit from new\_branch"

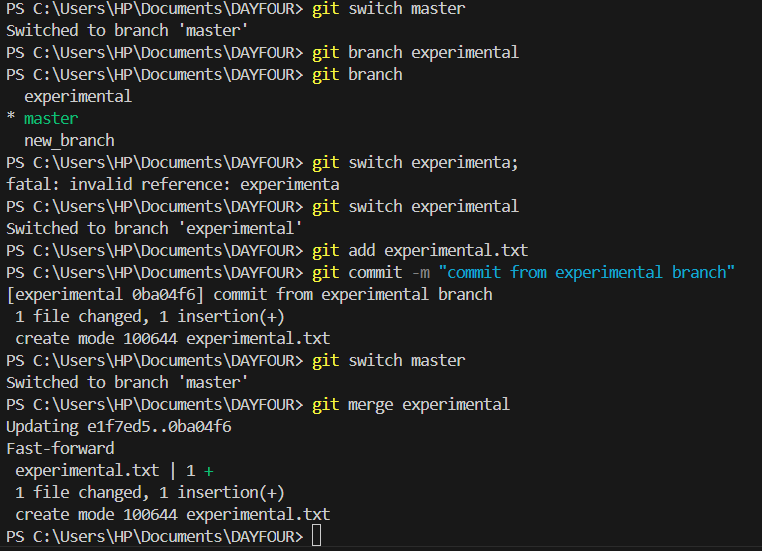
NOW TO MERGE NEW\_BRANCH WITH MASTER, WE HAVE TO SWITCH TO MASTER :

git switch master

git merge new\_branch

NOW MASTER AND NEW\_BRANCH HAVE THE SAME LEVEL OF CHANGES IN THEM.

2 MERGE WITH NO CONFLICTS



So here after switching to master, we create a branch experimental and switch to it. Create a file called experiemental.txt and commit it. We switch back to master and try to merge experimental. Finally in the master branch we will be able to see both the files. ( one with master already + another newly created in experimental branch).

3 MERGE WITH CONFLICTS : CONFLICTS BECAUSE BOTH FILES SAME LINE NUMBER CONTAINS DIFFERENT DATA. NOW WHICH ONE TO SAVE?

1 we switch to master and update file A.txt with 1st line “HELLO WORLD!”

2 add it to staging area and commit it

3 create a branch new\_branch\_2 and switch to new\_branch\_2 and update A.txt with 1st line “HELLO AGAIN!”

4 add it to staging area and commit it

5 come back to master branch and try to merge with new\_branch\_2

6 Now there will be conflicts because we have different texts is 1st line

7 We can choose to accept current changes / accept incoming changes in the VS Code itself

8 After choosing one option, we have to add and commit the file to repo

PS C:\Users\HP\Documents\DAYFOUR> git status

On branch master

nothing to commit, working tree clean

PS C:\Users\HP\Documents\DAYFOUR> git branch new\_branch\_2

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

PS C:\Users\HP\Documents\DAYFOUR> git commit -m "commit from master"

[master 30bf54b] commit from master

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_W

fatal: invalid reference: new\_branch\_W

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

PS C:\Users\HP\Documents\DAYFOUR> git status

On branch master

nothing to commit, working tree clean

PS C:\Users\HP\Documents\DAYFOUR> git branch new\_branch\_2

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

PS C:\Users\HP\Documents\DAYFOUR> git commit -m "commit from master"

[master 30bf54b] commit from master

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_W

fatal: invalid reference: new\_branch\_W

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

nothing to commit, working tree clean

PS C:\Users\HP\Documents\DAYFOUR> git branch new\_branch\_2

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

PS C:\Users\HP\Documents\DAYFOUR> git commit -m "commit from master"

[master 30bf54b] commit from master

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_W

fatal: invalid reference: new\_branch\_W

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

PS C:\Users\HP\Documents\DAYFOUR> git commit -m "commit from master"

[master 30bf54b] commit from master

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_W

fatal: invalid reference: new\_branch\_W

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

PS C:\Users\HP\Documents\DAYFOUR> git commit -m "commit from master"

[master 30bf54b] commit from master

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_W

fatal: invalid reference: new\_branch\_W

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_W

fatal: invalid reference: new\_branch\_W

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_W

fatal: invalid reference: new\_branch\_W

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

Switched to branch 'new\_branch\_2'

fatal: invalid reference: new\_branch\_W

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

Switched to branch 'new\_branch\_2'

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

PS C:\Users\HP\Documents\DAYFOUR> git switch new\_branch\_2

Switched to branch 'new\_branch\_2'

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

Switched to branch 'new\_branch\_2'

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

PS C:\Users\HP\Documents\DAYFOUR> git commit -m "commit from new\_branch\_2"

[new\_branch\_2 2839e8a] commit from new\_branch\_2

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch master

[new\_branch\_2 2839e8a] commit from new\_branch\_2

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch master

1 file changed, 2 insertions(+), 1 deletion(-)

PS C:\Users\HP\Documents\DAYFOUR> git switch master

PS C:\Users\HP\Documents\DAYFOUR> git switch master

Switched to branch 'master'

PS C:\Users\HP\Documents\DAYFOUR> git branch

experimental

\* master

new\_branch

new\_branch\_2

PS C:\Users\HP\Documents\DAYFOUR> git merge new\_branch\_2

Auto-merging experimental.txt

CONFLICT (content): Merge conflict in experimental.txt

Automatic merge failed; fix conflicts and then commit the result.

PS C:\Users\HP\Documents\DAYFOUR> git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: experimental.txt

no changes added to commit (use "git add" and/or "git commit -a")

PS C:\Users\HP\Documents\DAYFOUR> git add experimental.txt

PS C:\Users\HP\Documents\DAYFOUR> git commit -m "resolved conflicts"

[master 6004504] resolved conflicts

PS C:\Users\HP\Documents\DAYFOUR>

How to use git diif : ( you can see the differences until you saved but not “committed” the changes in 2nd branch )

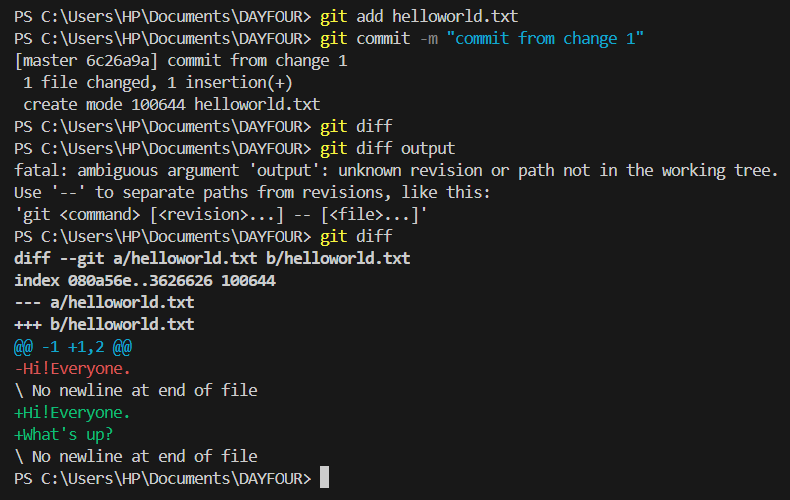
1 create a file helloworld.txt and add text “Hello! Everyone.”

2 stage and commit it

3 add some more text “What’s up?”

4 now type git diff

5 Now it will show :



To know the difference between a single file but two different branches

git diff branch\_name\_1 branch\_name\_2

***Day 4***

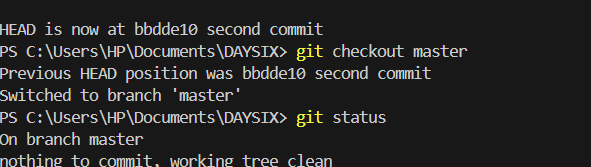
Git Checkout :

* Act of switching between different versions of the entities
* Can be used on files, commits and branches
* We can use git checkout branch\_name to switch to that branch. (switch was introduced later)
* We can use git checkout “7 digits of the commit hash” of the commit from git log - -oneline command

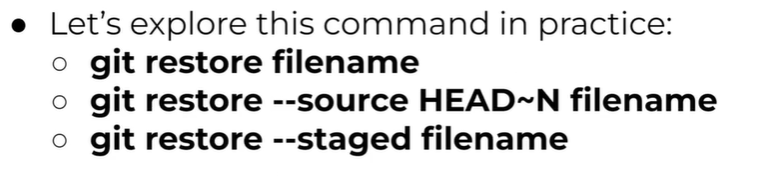
Practice :

* Create a repo and a file
* Add comment and do commit
* Add another comment n do commit
* Add another comment n do commit
* Now we have 3 commits
* Now do git log –oneline to see first 7 digits of the commits so far
* git checkout “7 digits of the commit hash” and the HEAD will point to that git commit
* ***git checkout hashcode* will make you see how it looked like when you did the commit hashcode**
* **Also, HEAD will be in de-attached mode i.e it is pointing to the commit, not any branch!**
* To join HEAD back to master, type git checkout master





Git Restore : restore is more with the file rather than commits. It means it will restore the file status back how it was



1st command : git restore filename : it will take the file back to as it was in the lastest commit.

2nd command : git restore –source HEAD~2 filename. It will take back the file to 2 commits from the HEAD as it was.

3rd command : git restore –stage filename it will unstage any filename mentioned

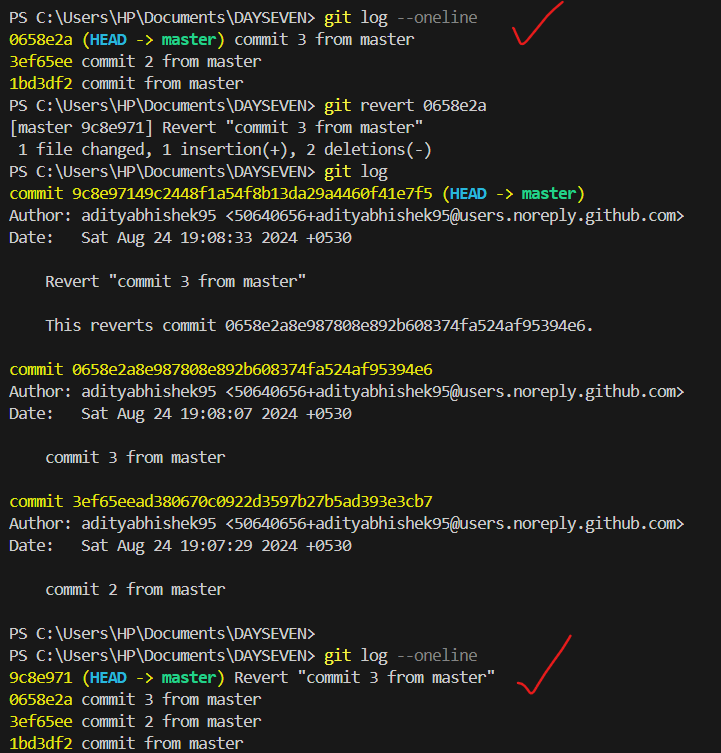
Git Reset :

* Git reset hashcode : it will take back to this “hashcode” and make all other files back to the working directory as it was.
* Git reset hashcode –hard : it will take back to this “hashcode” and make all other files changes DELETED!

Git Revert :

git revert hashcode

Git revert will revert the mentioned hashcode of the commit but it will keep a copy of the revert in the log



Here initially we had 3 commits in the log. Now in the log we have 4 commits ( 1 from reverting a commit ).

Exercise :

s

GET THE URL TO CLONE THE REPO FROM GITHUB!

It's a very simple repository, so you'll notice it's essentially just one file, my file, txt.

And if you open that, it's just the first line in the first commit, second line in the second, commit

a third line in the third commit and then last line and the most recent commit.

And if you take a look at the history, there's only first, second, third and fourth commit.

So that way you can very easily identify which commit you are just by reading the file.

So go ahead and clone that and then you'll be ready for the next set of tasks.

So we want you to do is without permanently changing the content of the file at any commit view.

What the file looked like after the first commit inside your text editor.

This should give you a detached head at the first commit.

Once you're done viewing the file and what it looked like at that first commit, move the head back

to the current main branch, which is essentially the same thing as moving it back to the most recent

commit on the main branch.

And when you clone this, there should only be one branch.

Then go back to the commit that has the message.

Second commit.

So you'll need to understand how to look up that hash and then create a new branch at that point called

new branch.

Keep in mind we do not want you to delete any commits that came after this one.

We just want you to create that new branch.

Now, in order to test our skills, we're going to have you stay on the main branch and pretend like

you forgot to switch to your new branch.

So while you're on the main branch, we're first going to test your abilities to undo changes.

So while you're on the main branch, viewing the second commit, go ahead and delete everything in the

file using the text editor and your delete key.

So let's pretend that you accidentally highlighted and delete everything.

Now using git commands only, not just control z, restore the file so it matches what it looked like

at the second commit right before you deleted everything.

So once you're able to restore that text that you deleted just using a git command while you're still

on the main branch at the second commit, add a new line that says new branch text and then add and

commit this change to the file.

You're going to realize that you actually meant to do this on your new branch, so you're going to have

to figure out how to undo that.

Commit on the main branch without losing your work.

Then switch branches and do the add and commit procedure that has that new line of new branch text.

GIT STASH :

git stash means if we are working on a file and suppose we did not add it to stage / commit it. Git stash will save the changes (and it will go away from the screen also ) and can we later recovered by git stash pop and git stash pop apply.

git stash = to save the file invisibly in a box

git stash pop = to unbox that save item

git stash apply = to unbox that change but still keep it in box for future use.

