**Git – Understanding**

Git operations & Commands:

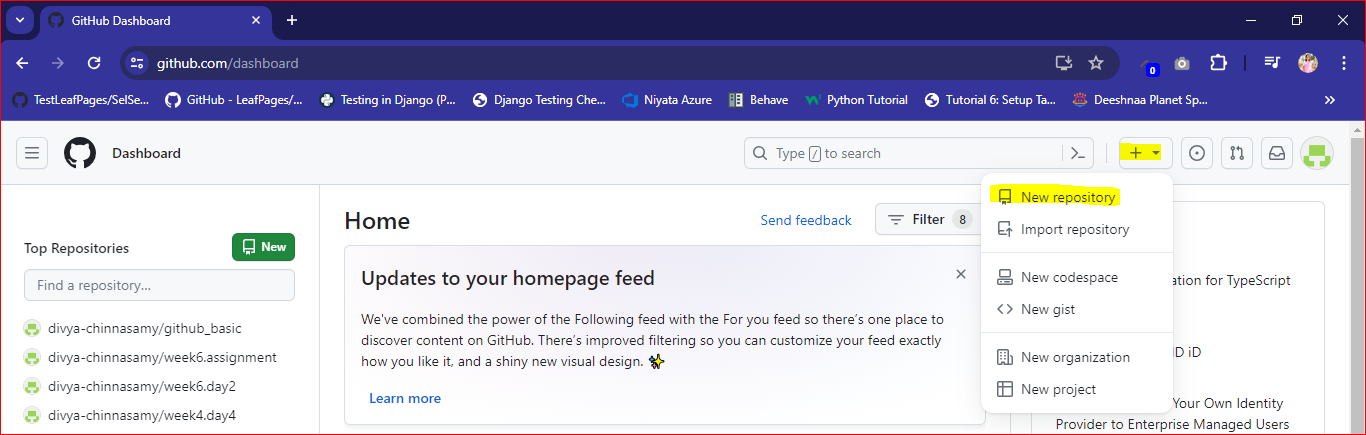
1. Add
2. Commit
3. Pull
4. Push

Some advanced Git operations are:

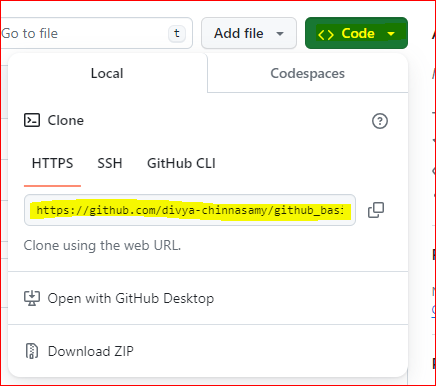
1. Branching
2. Merging
3. Rebasing

The Architecture of Git below:

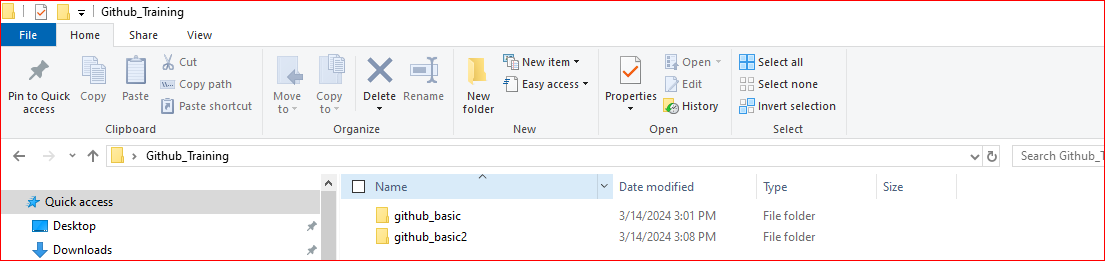




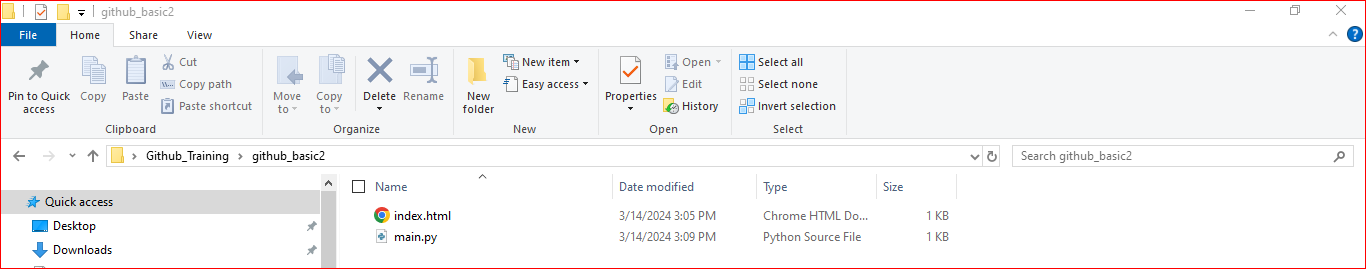
To clone, need to input the link:



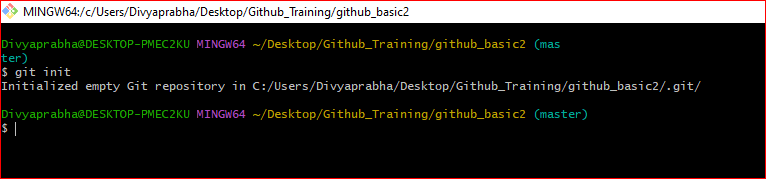
Created 2nd folder in the local and trying to move to Remote:

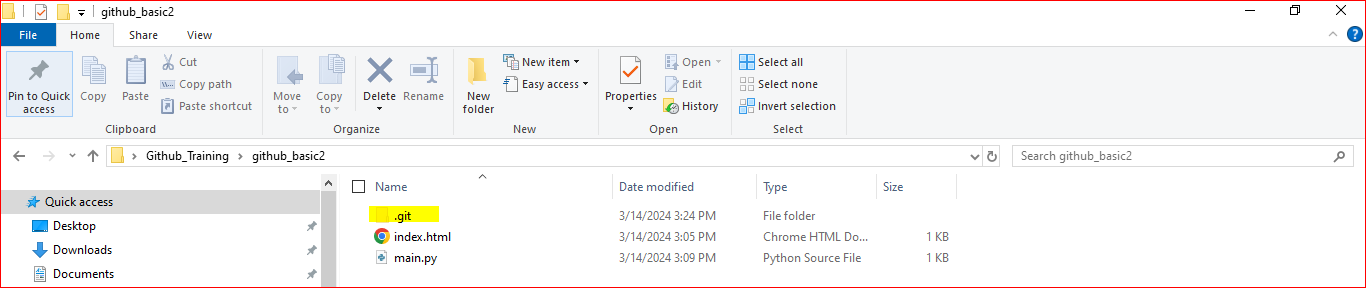


The files in the local repository github\_basic2 is shown below:

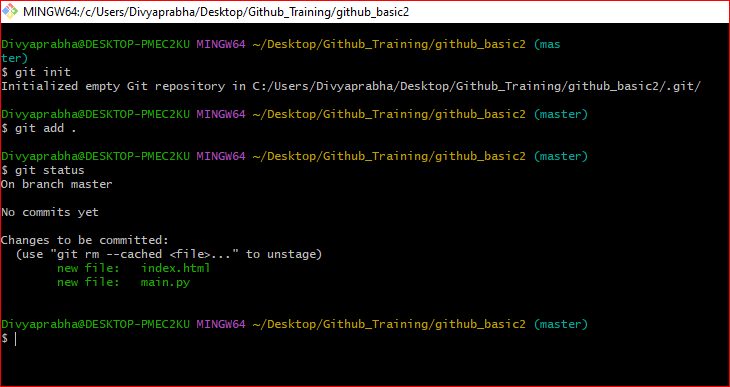


To make it as git folder, give the below command:

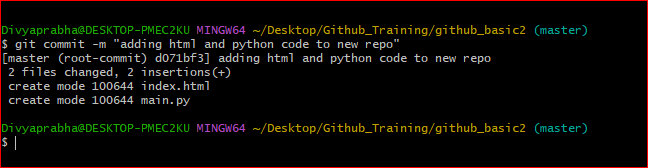




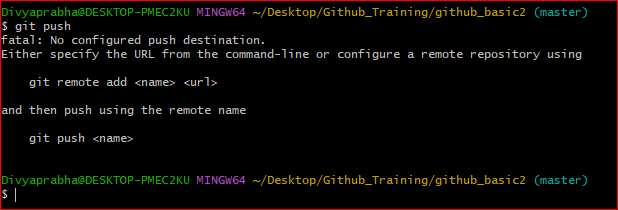
Now adding the files to staging by using add . command and check the status:



Committing the changes using git commit commands.

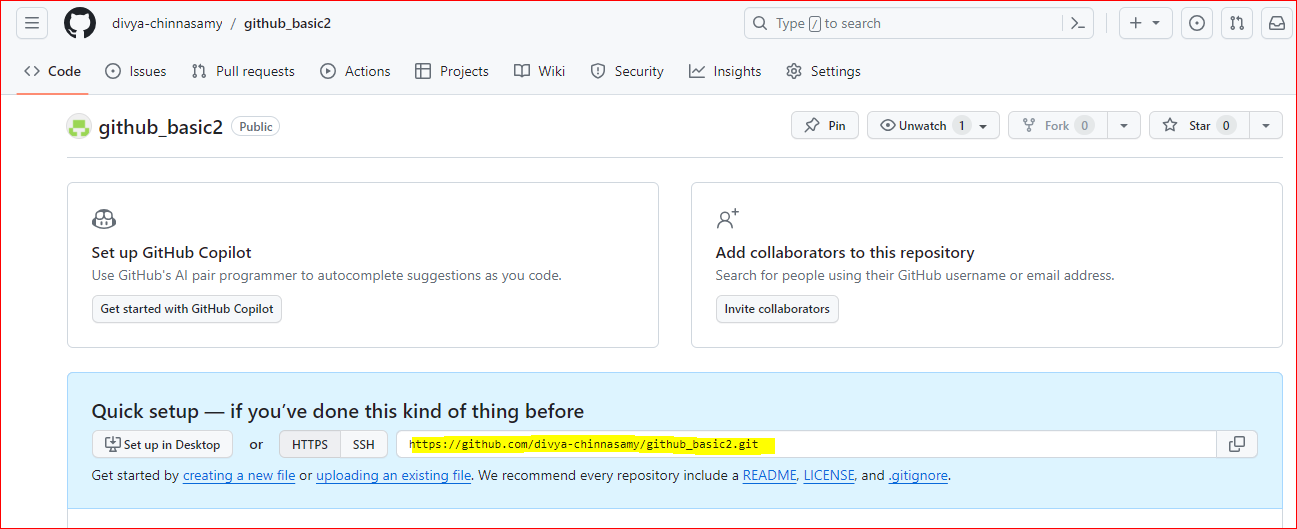


Push the code to remote using git push:

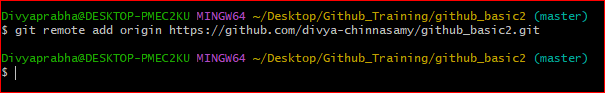


The above error is due to git couldn’t find the origin github\_basic2.

Created new repo in remote.

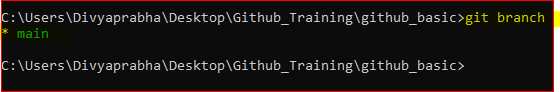


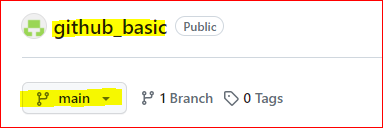
Adding the new origin using git remote add origin “link”



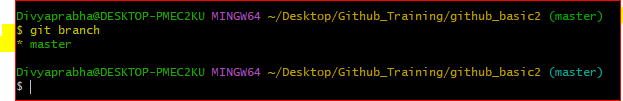
Now the local is connected to the remote origin as expected.

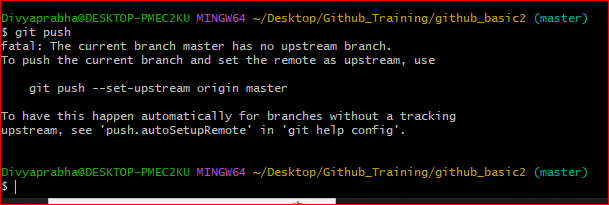
Github\_basic is in main branch. Can be checked by using git branch command.



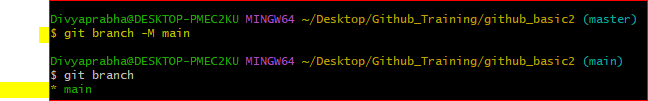


Github\_basic2 is in master branch, can be checked using the git branch command.

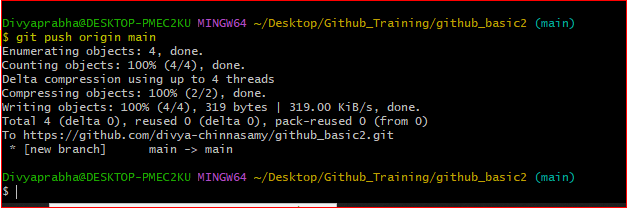




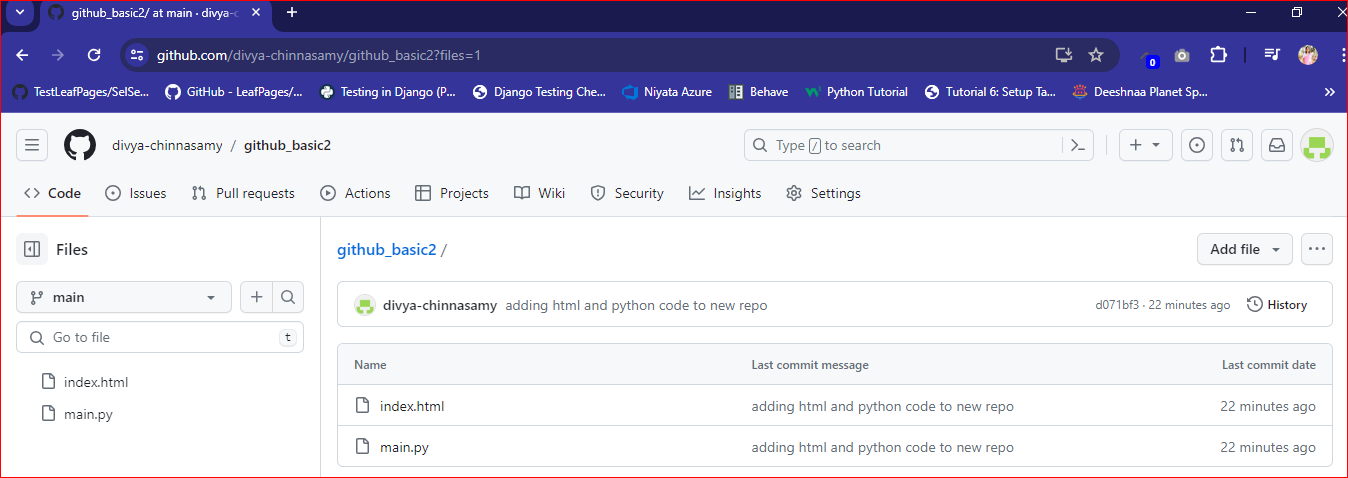
**Changing the branch name from master to main using the below command.**



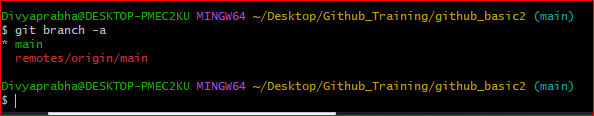
Now, Push the committed files to the remote.



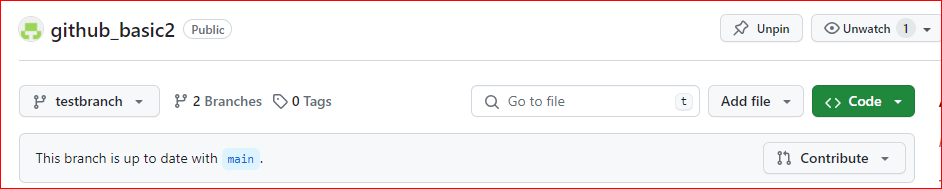
Code pushed to the remote repo:



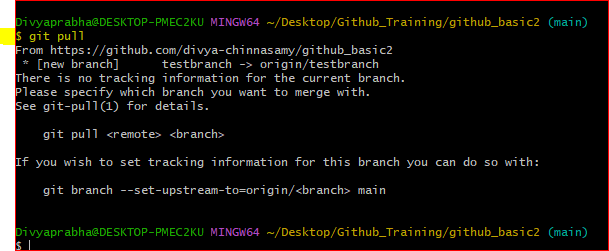
To know the branches in repo use the command.

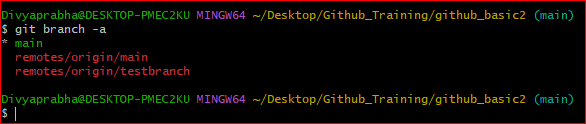


**Branching:** Created new branch in remote.

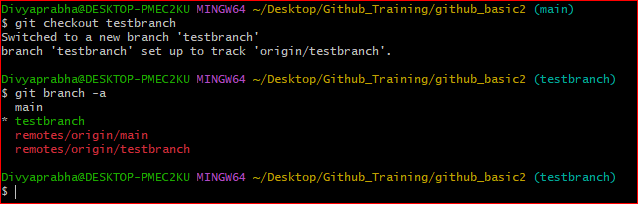


Using git pull, bringing the changes to local

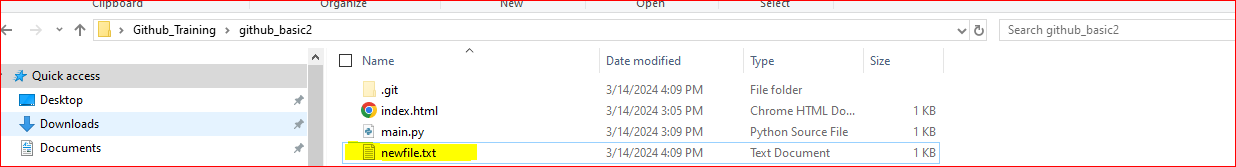


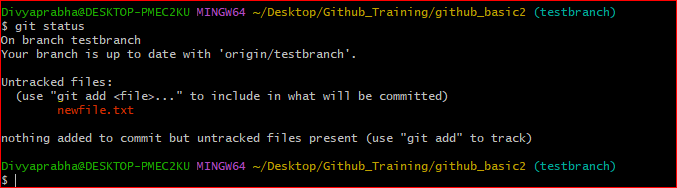


To get into testbranch, use the below command.

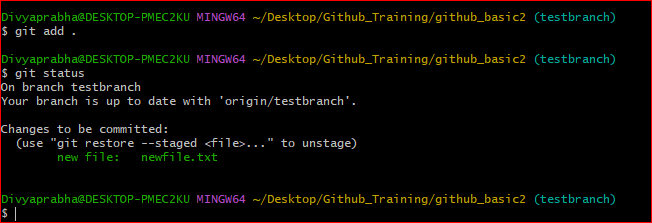


Creating a newfile in the testbranch and moving to remote.

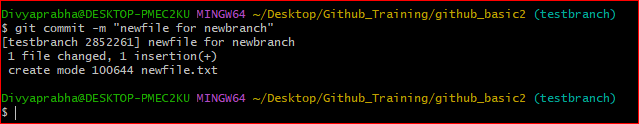




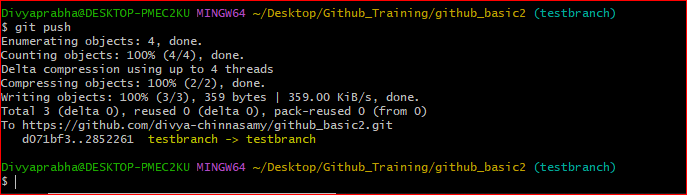
Moved to staged.

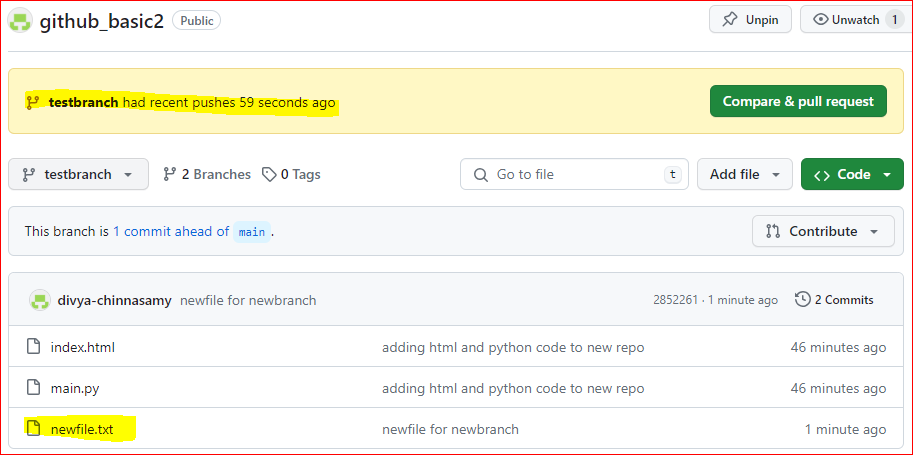


Commit with comments:

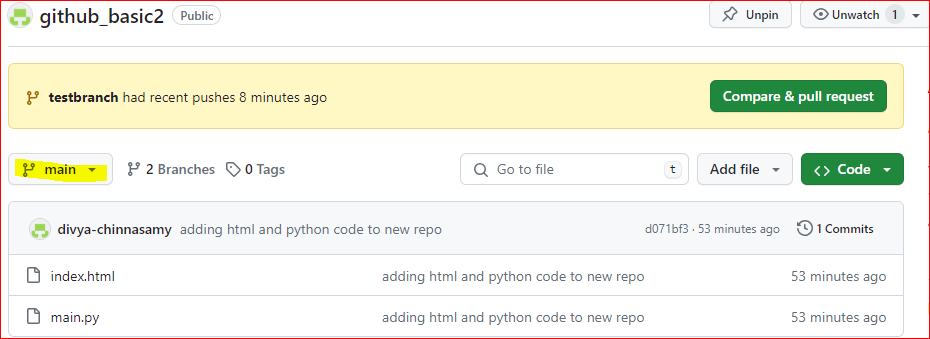


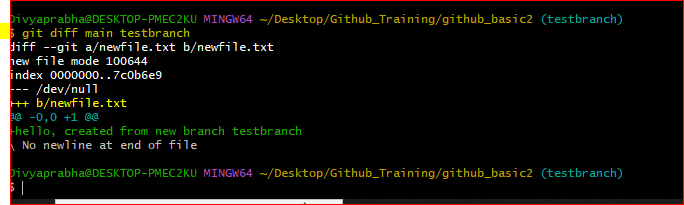
Code pushed to remote testbranch.



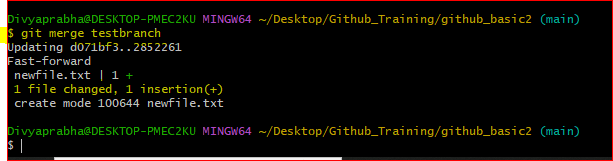


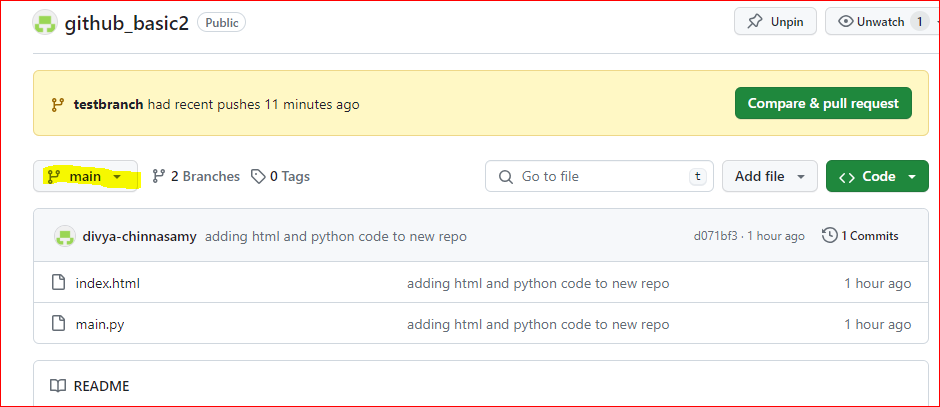
New file is not visible in main branch.



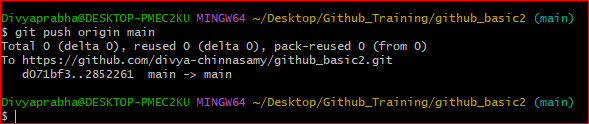
To know the difference in main and testbranch, use the git diff command.

To know which branch we are currently in:

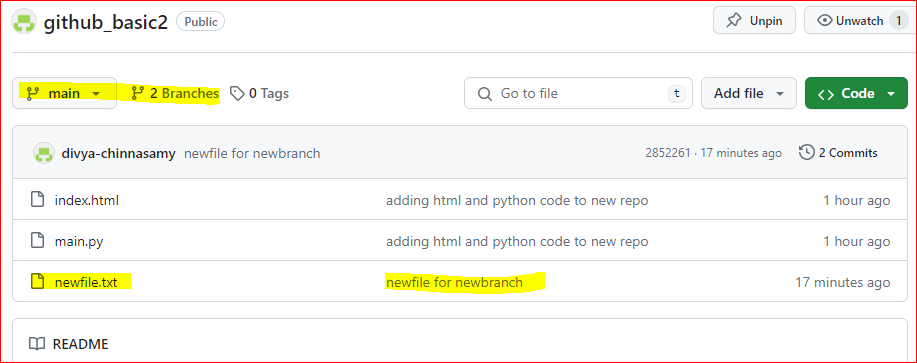
**Merging:** To Merge both branch:

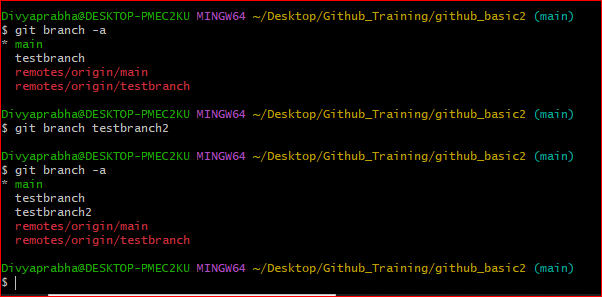
Main branch is still not showing the merged file.

To see the merged code in the main branch, use git push.

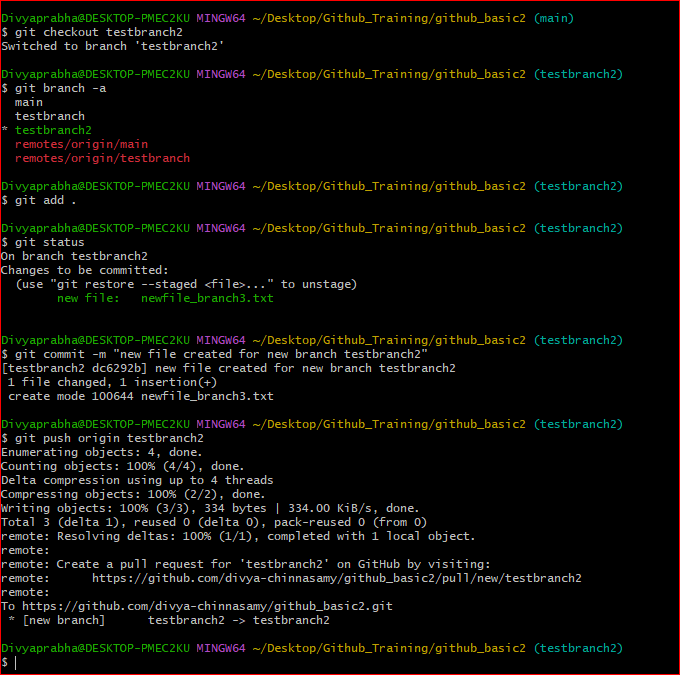


Now the changes are visible in main branch:

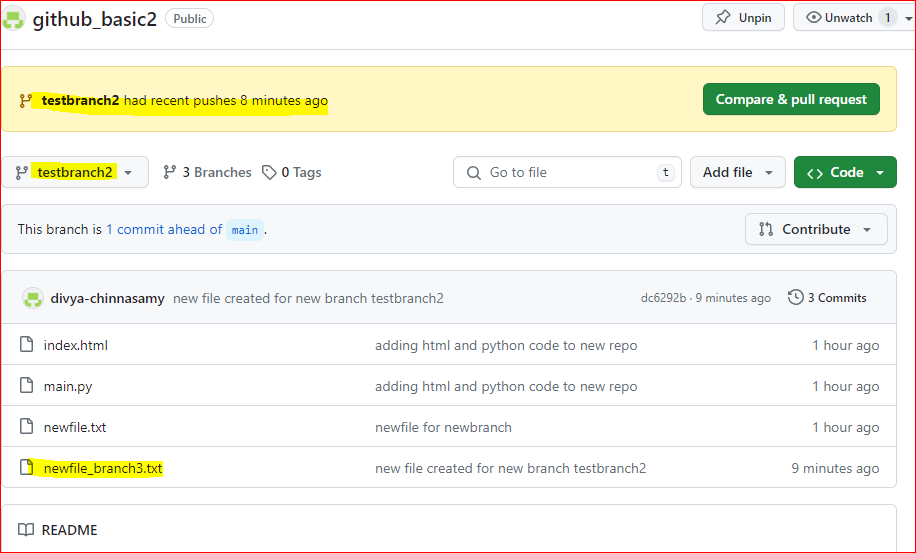




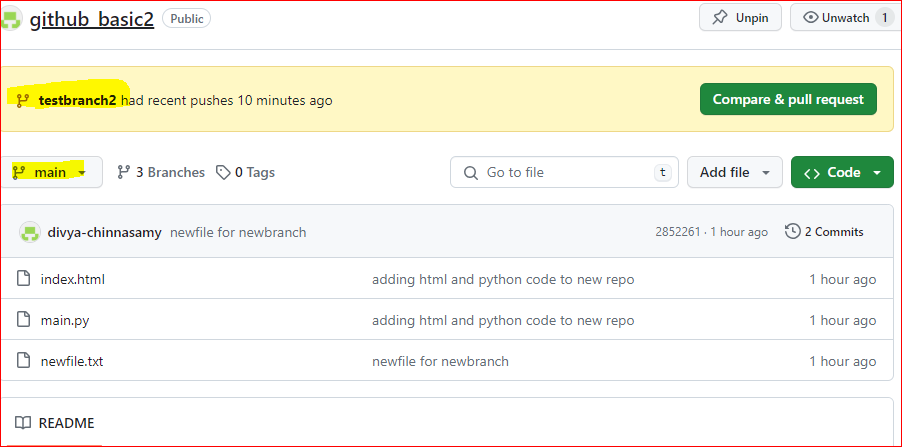
From main branch, move to new testbranch2 and create a file and push to remote.



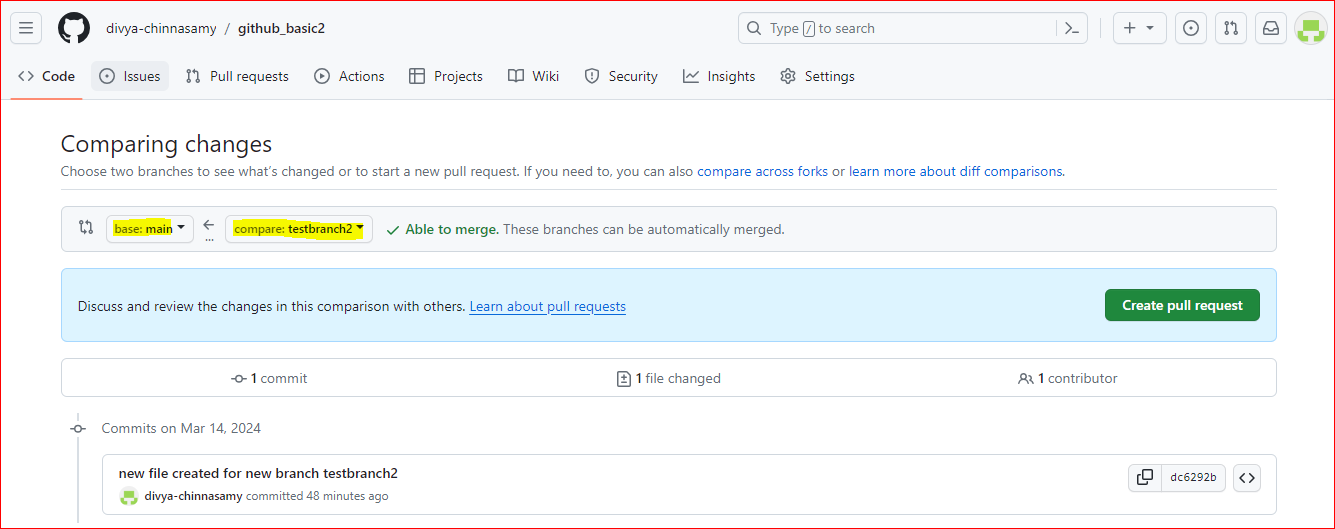
In the above process, the new branch gets created in remote automatically and the file is visible in the remote

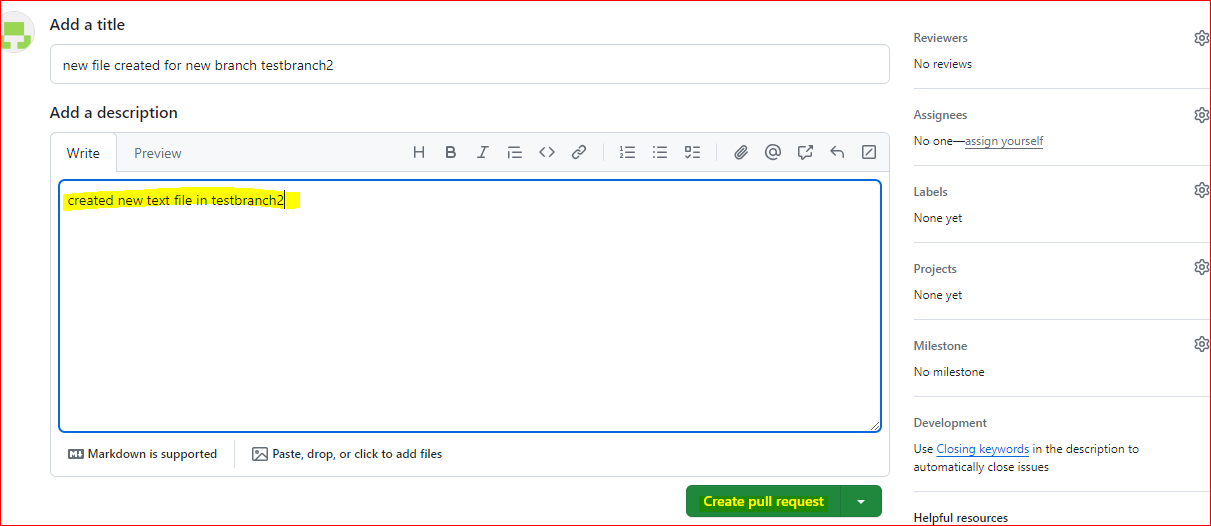


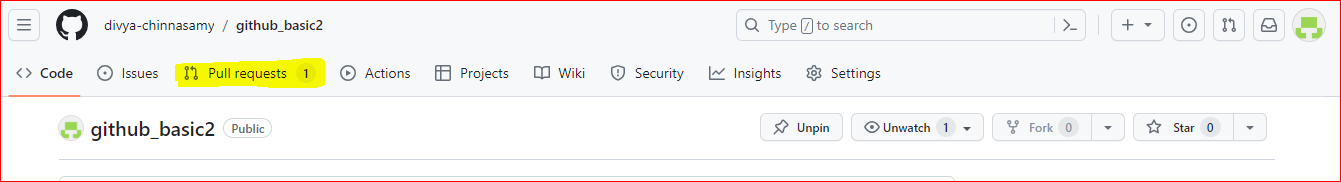
The newfile\_branch3 is not visible in main branch.



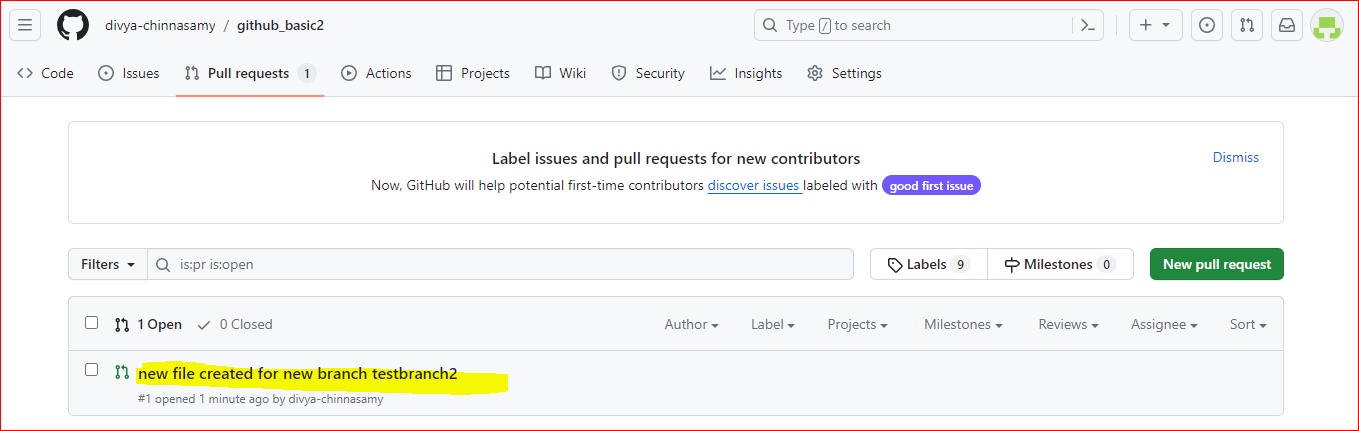
Now to merge testbranch2 with main 🡺 we cannot have direct access to merge with main branch, creating an approval request to merge with main branch. This can be done using pull method.

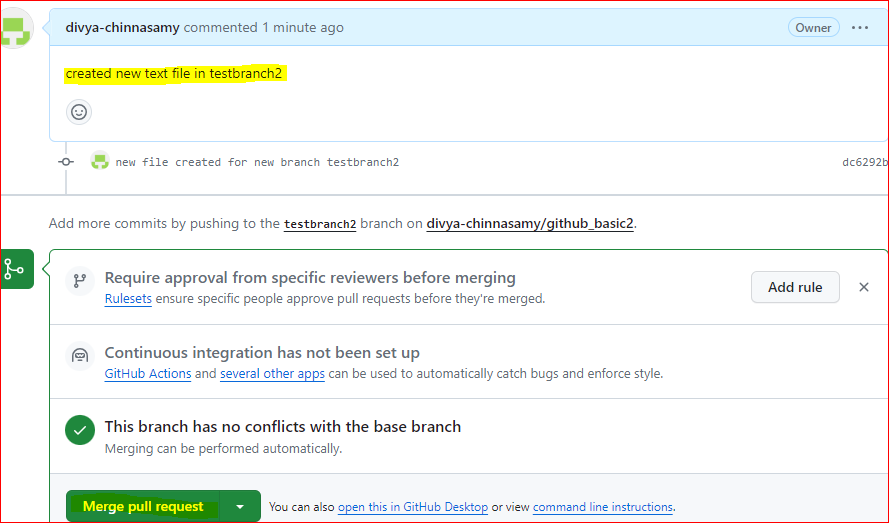


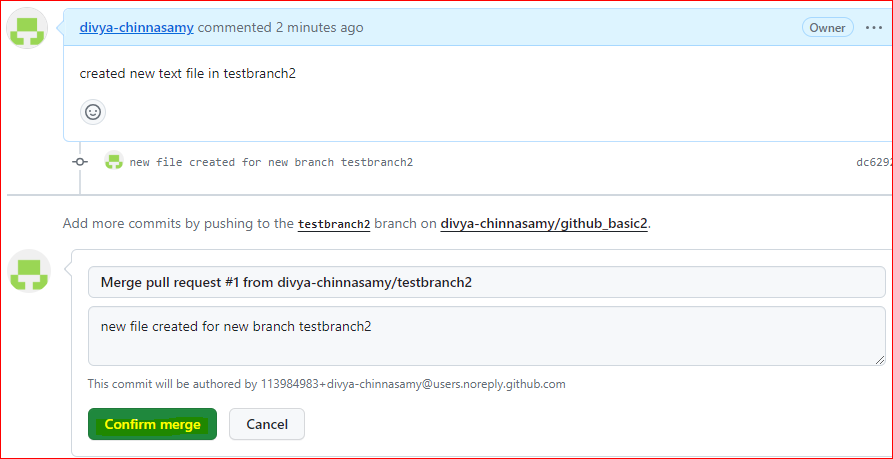
Creating pull request.



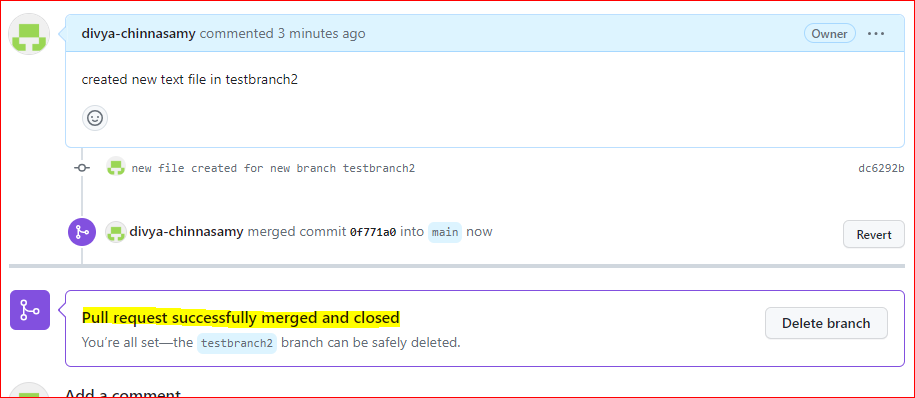
Request waiting in queue for approval:



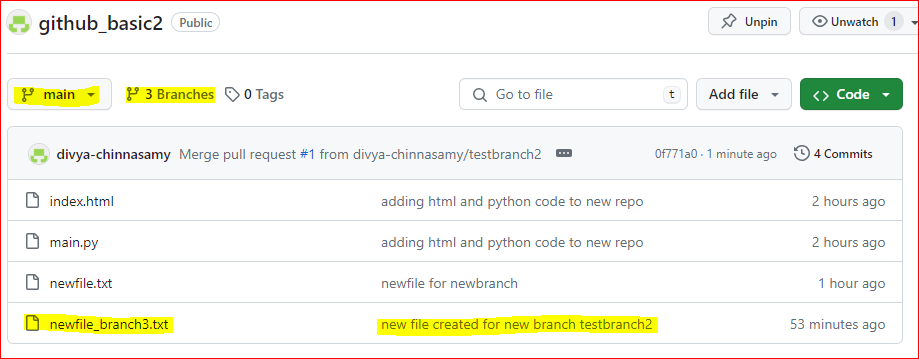
Click Merge Pull request: 



Once confirm merge is clicked, pull request success,



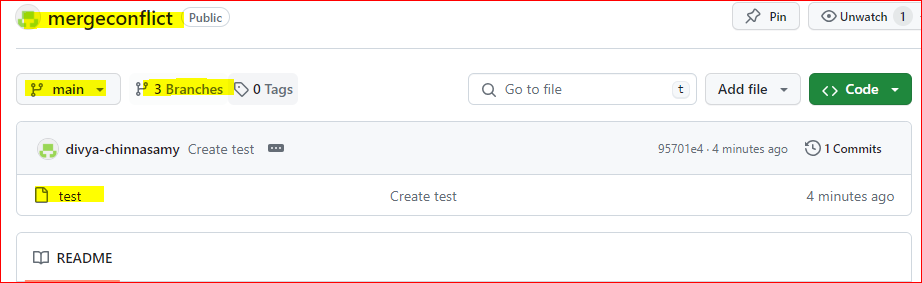
File merged with main branch:



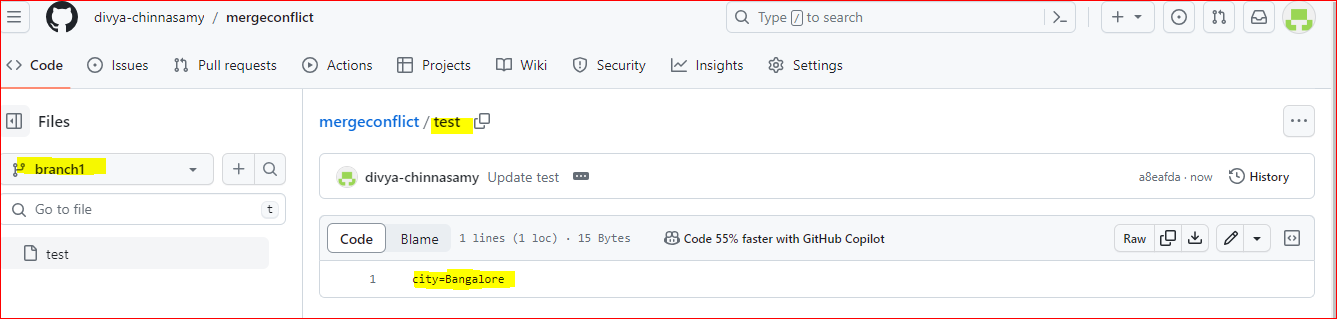
**Merge Conflicts:** When 2 person tries to change the same file, git confused.

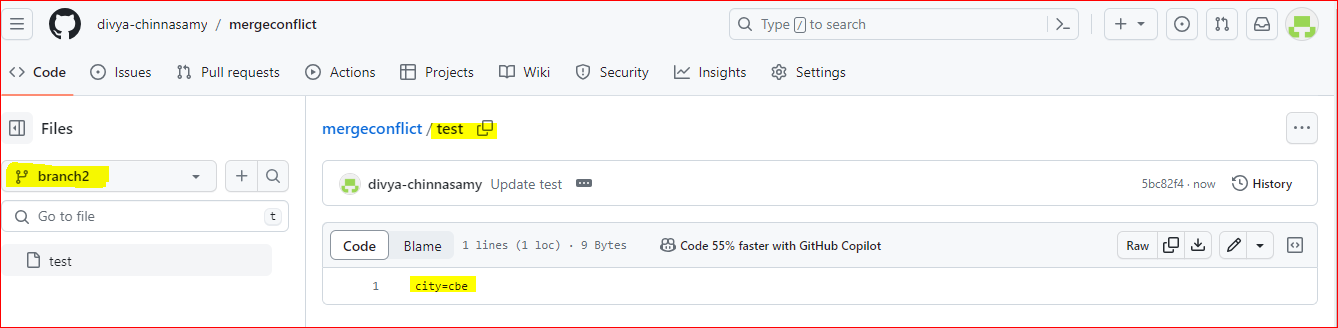
Created new repository: mergeconlfict 🡺 has file 🡪 test.txt

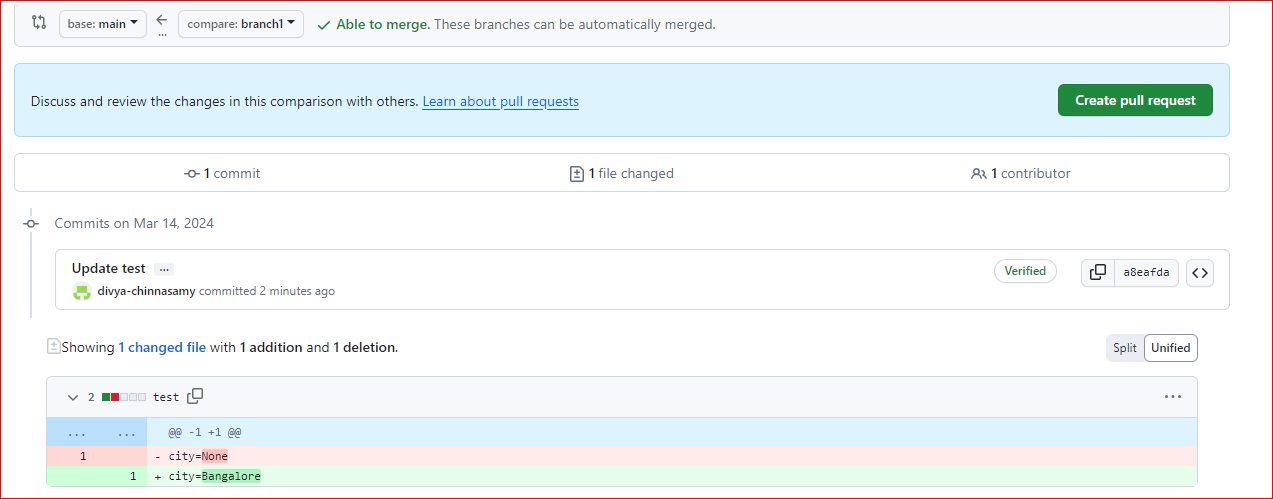
Created 2 branch for this repo 🡺 branch1, branch2.

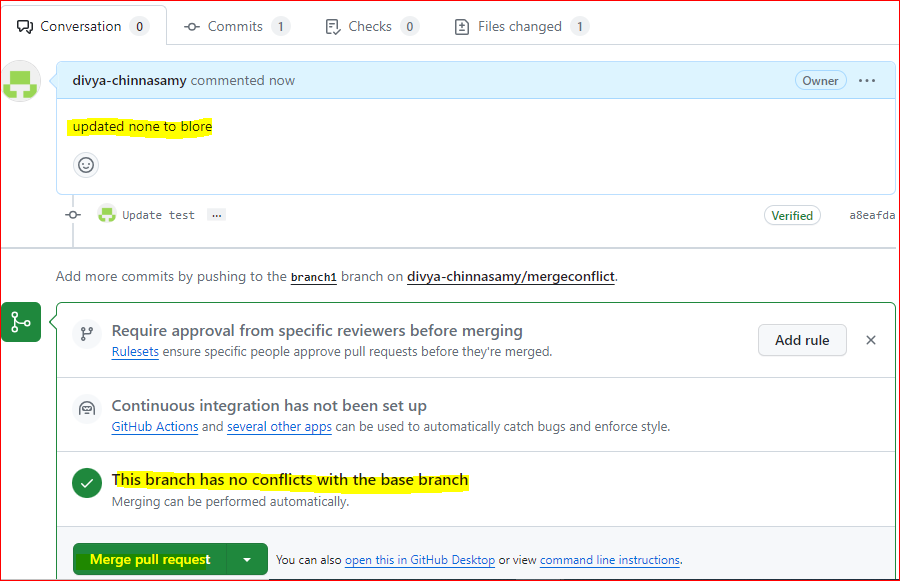


Updated test file in branch1 as below:

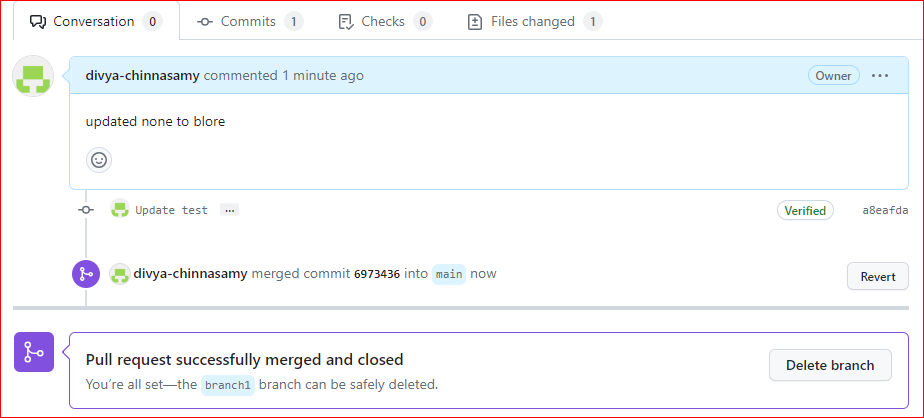


Updated test file in branch2 as below: 

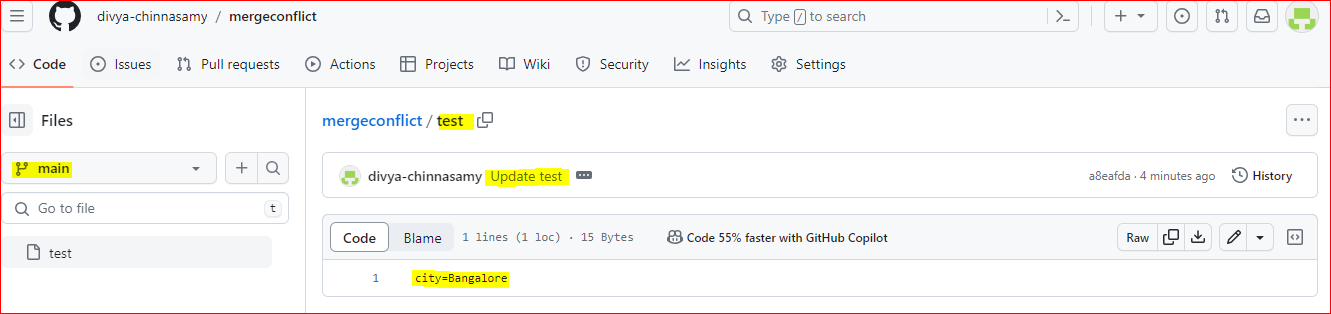
Now creating a pull request to merge with main:

No conflicts when tried to merge main with branch1:

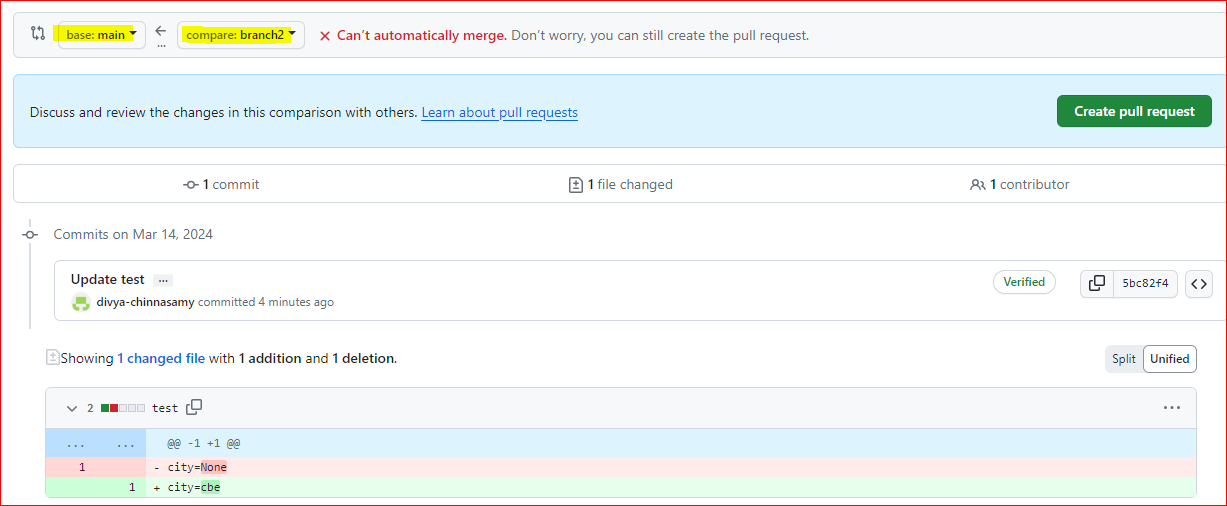
Merge completed:



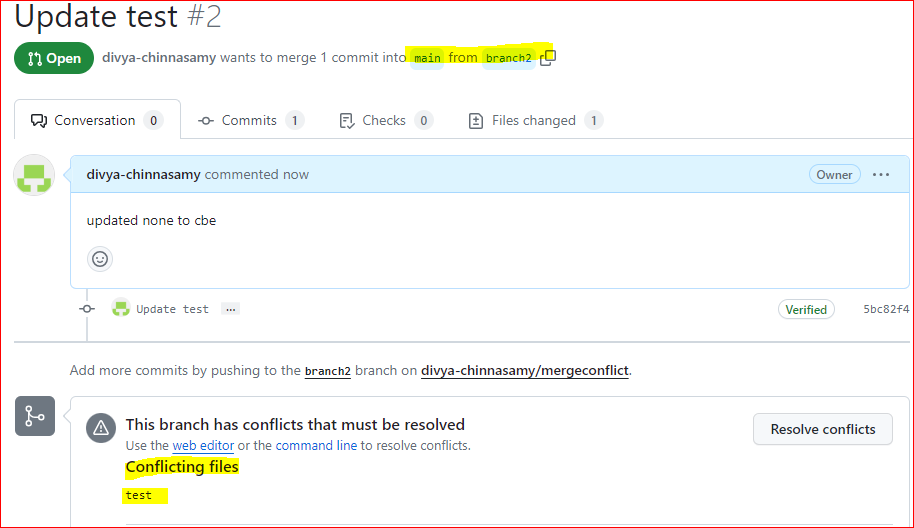
Main branch merged:



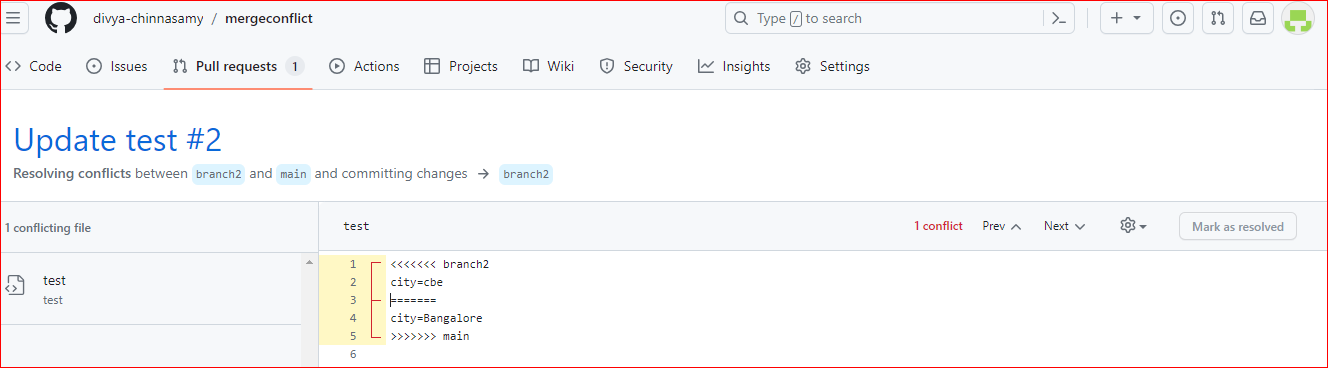
Now trying to merge main with branch2 using pull request:



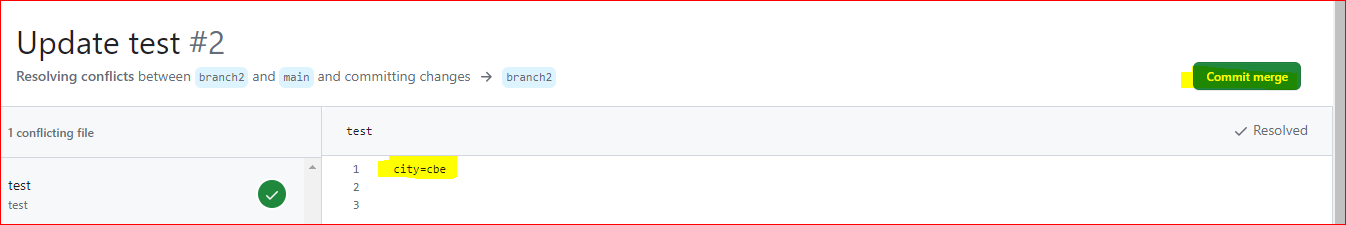
Now conflicts arises:

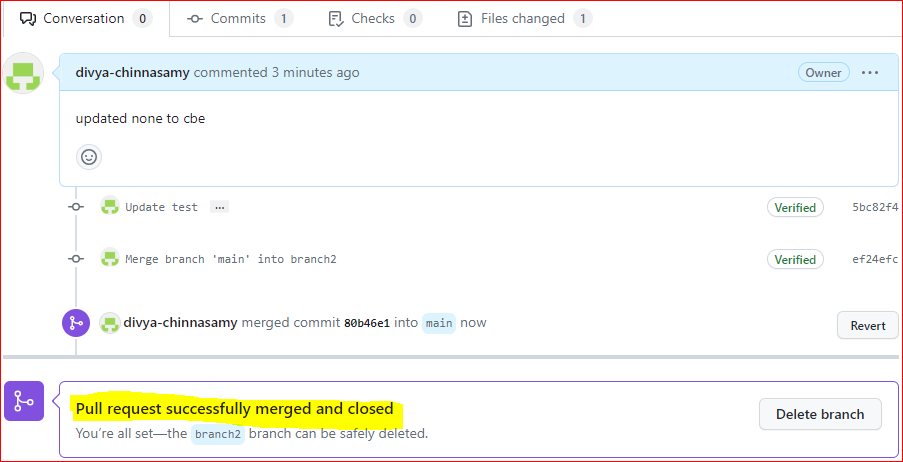


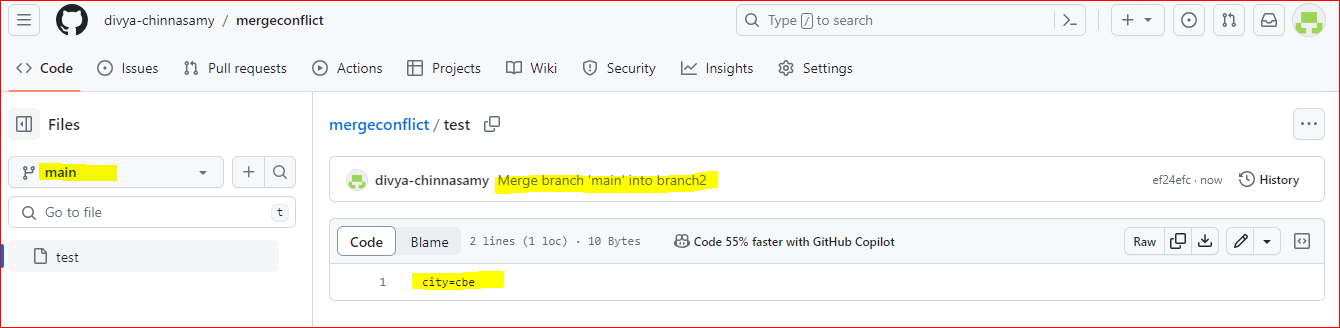
Manually resolve the conflicts by clicking resolve conflicts.



Once updated with changes, click commit merge



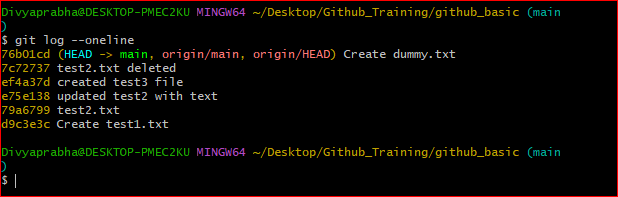


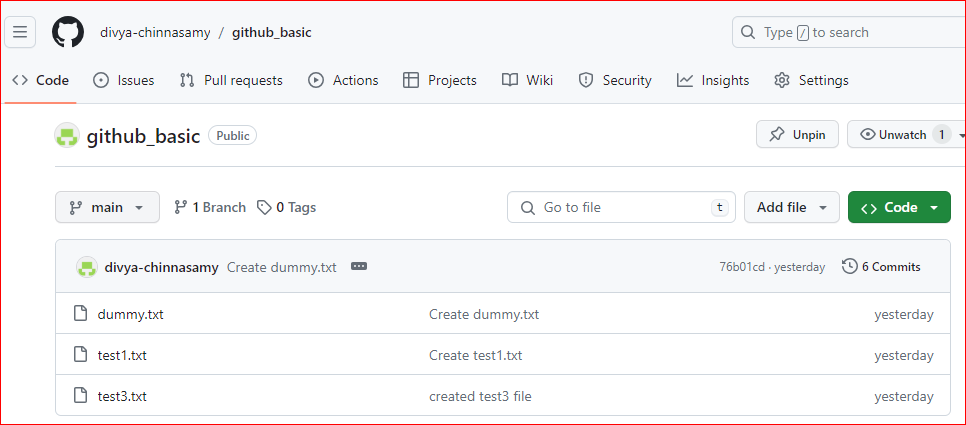


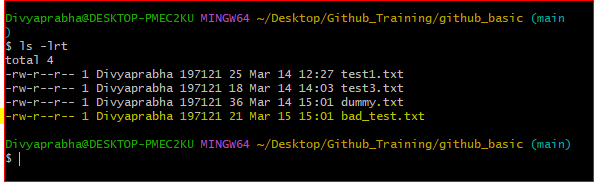
Git Advanced Operations:

1. how to revert the commit
2. how to get back to staging to wrkspace and
   1. query: local to stage, also local repo to workspace?
3. how to setup security things to setup pull request?
4. github admin operations
5. git advanced commands
6. git merge requests (merge code)
7. how to create parent MR & Child MR (Master &slave MR)
8. how to write Git manifest file?

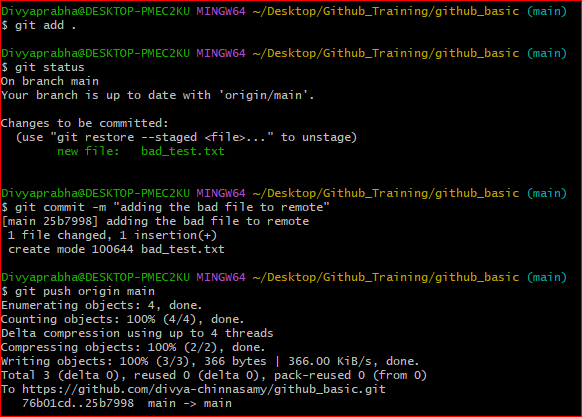
**How to revert the commit?**

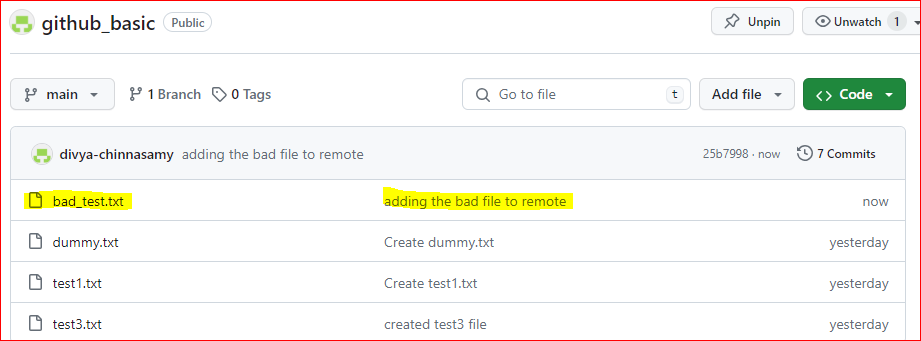


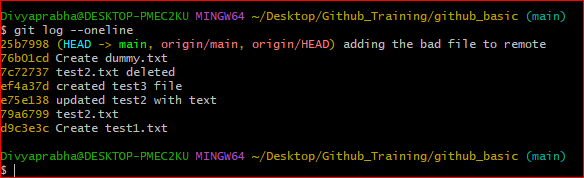




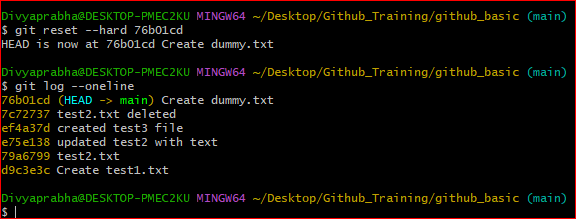
Pushing this file to remote



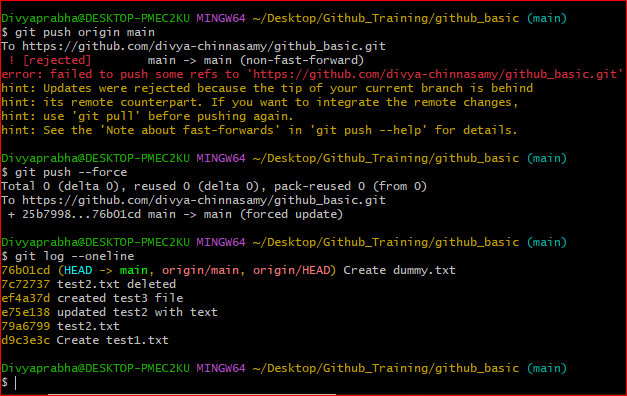




Resetting back to dummy.txt.

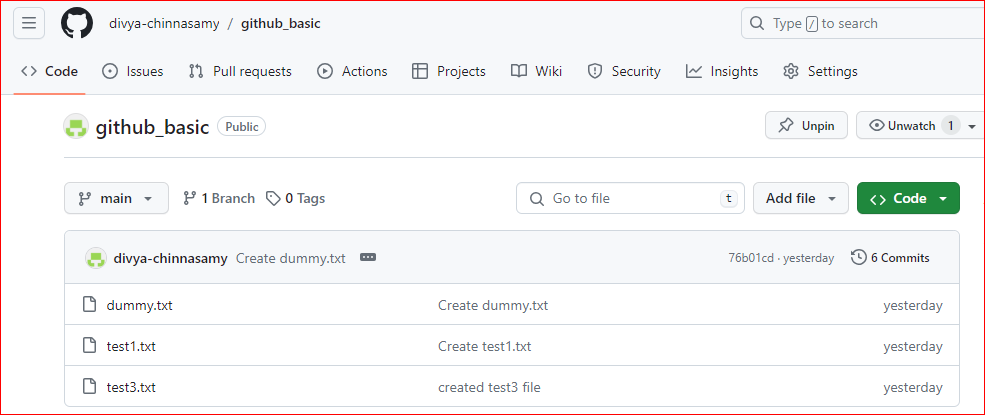


Pushing to main using git push force to reset the bad commit.

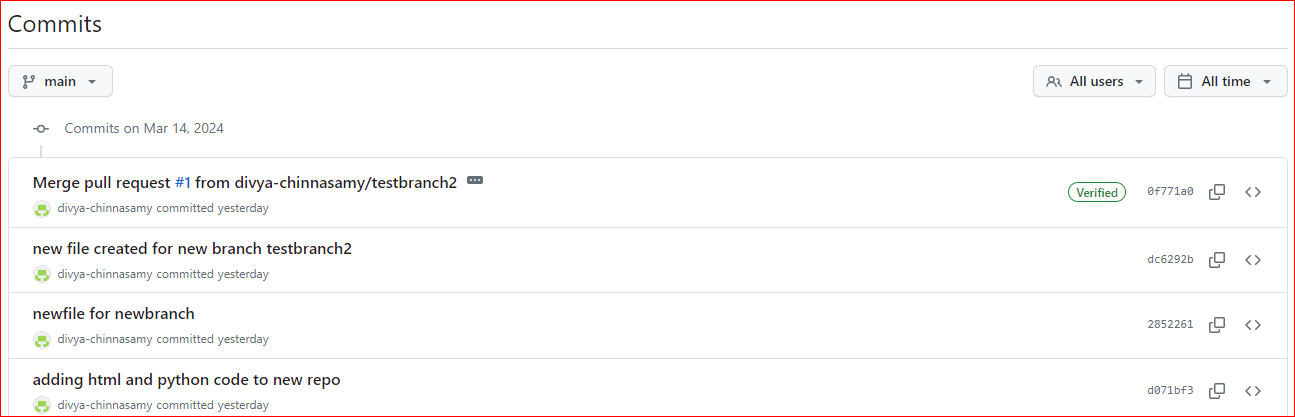
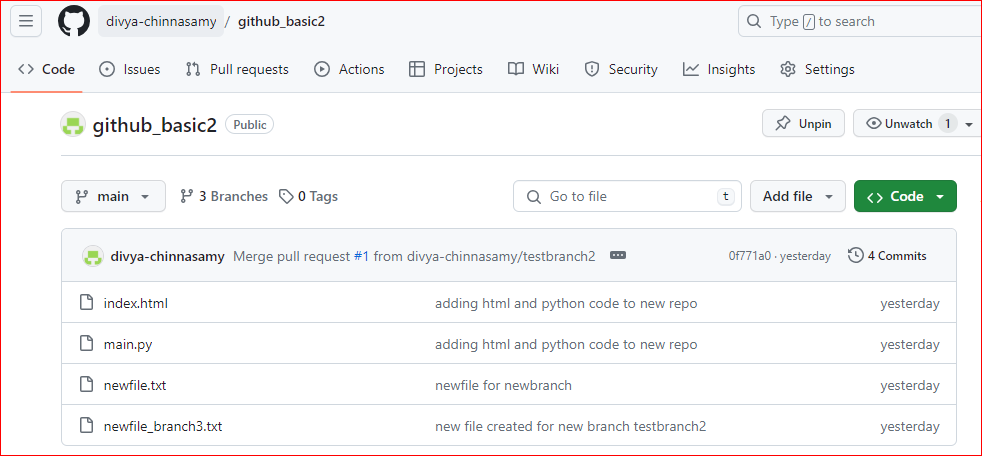


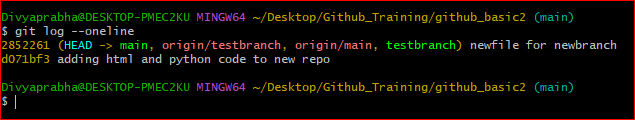
The history has been altered and this is bit problem when working with group of team.

Back to 3 files after reset:

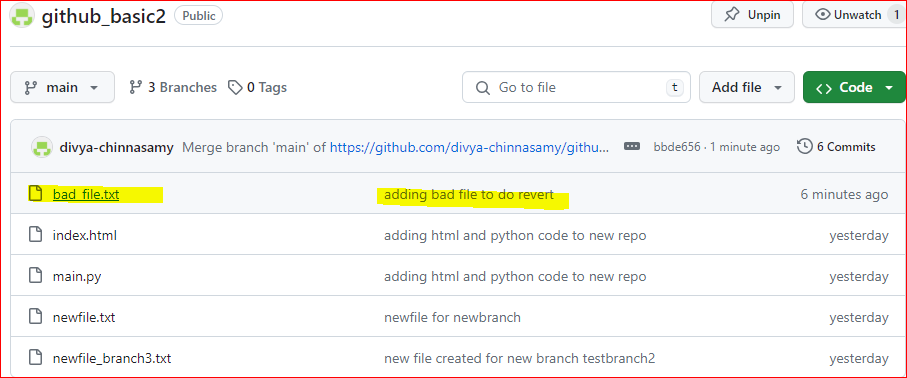


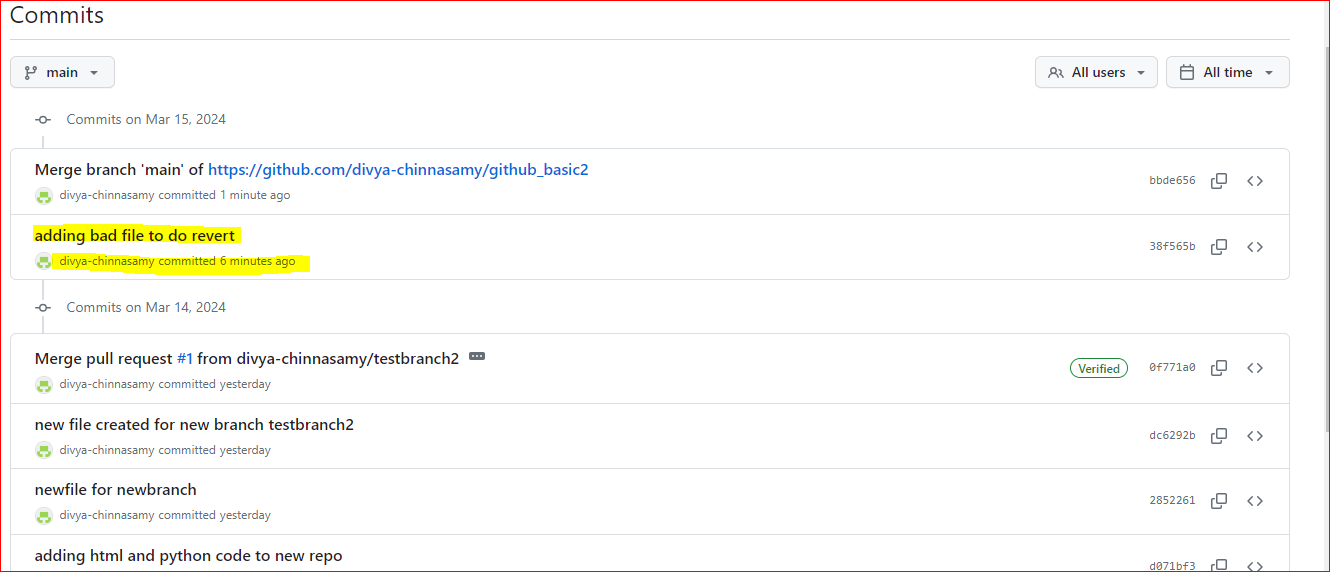
Revert is more better way compared to Reset. This would not alter the history instead it makes the history as such by removing the bad commit, pls find below the example of Revert.

Before bad commit:

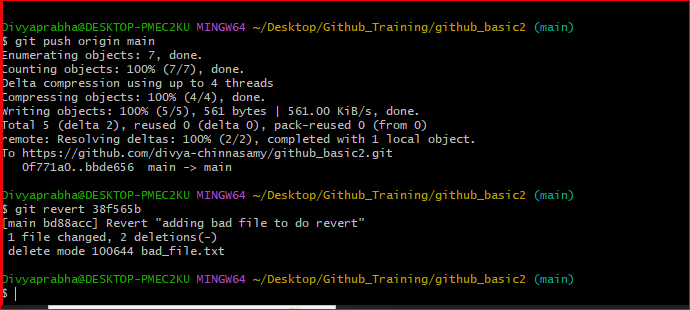


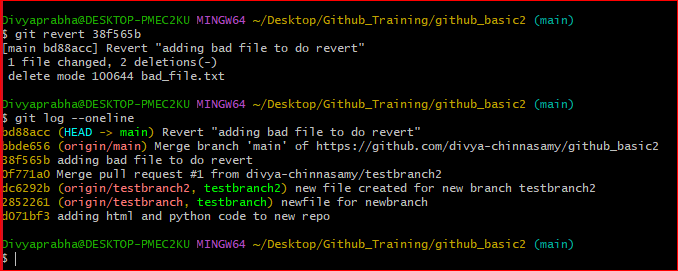
Adding bad file to test revert:

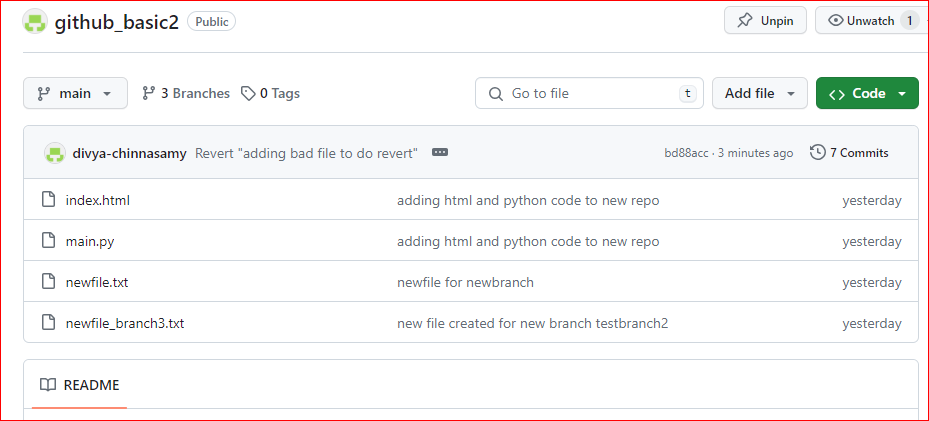




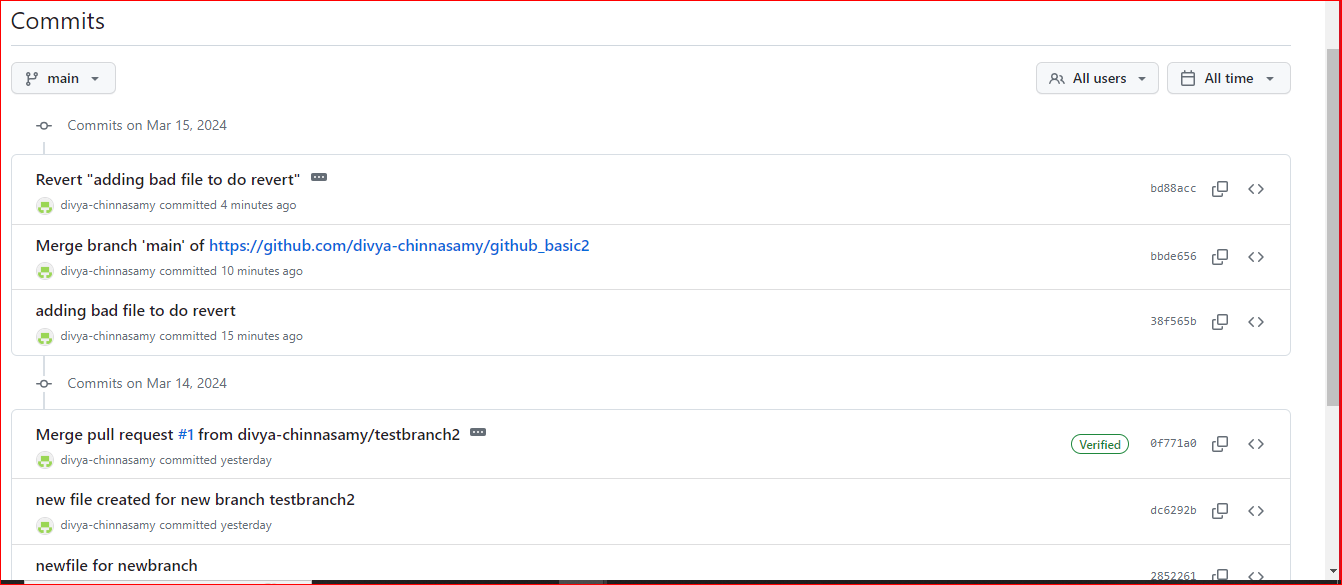
Reverting the bad commit:





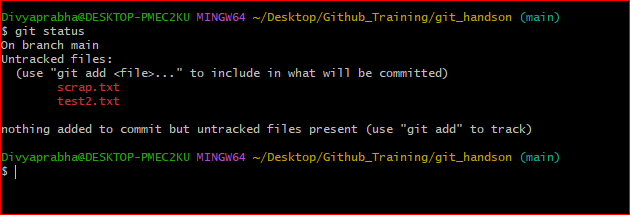
Bad file has been removed using revert command.

History not altered: can see the bad file commit and revert commit also.

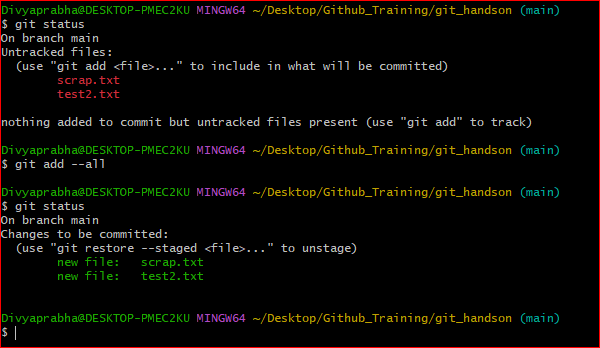


**How to get back to staging to workspace and local to stage, also local repo to workspace?**

Currently there are 2 files in workspace area



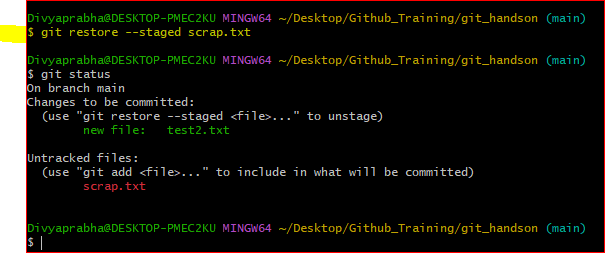
Both files are moved to Staging



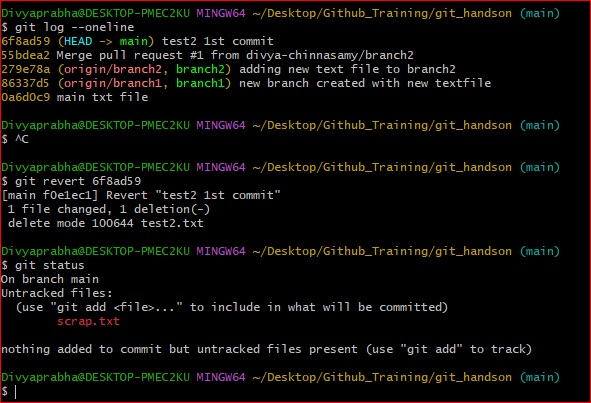
Now, Removing scrap from staging.

**3 methods:**

1. Restore method 🡺 git restore --staged <filename>
2. Reset method 🡺 git reset <filename>
3. Remove method 🡺 git rm –cached <filename>

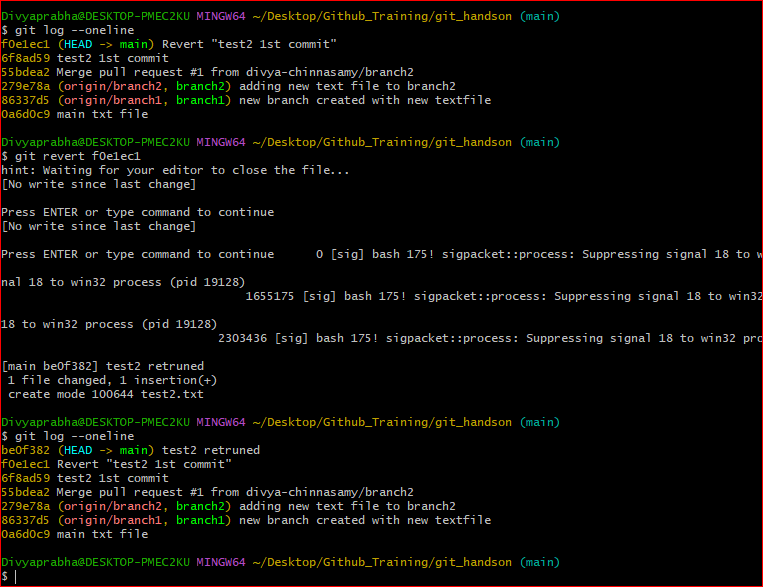


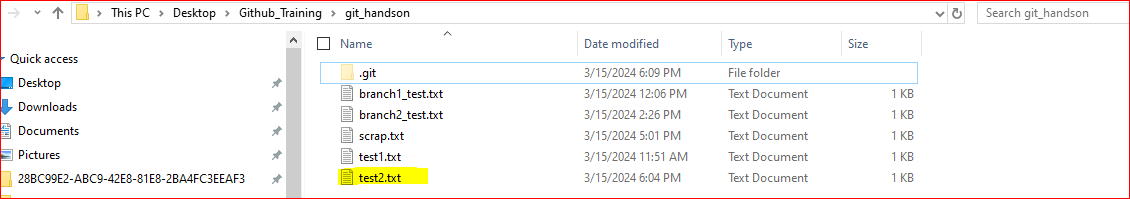
To revert from the commit, use revert command as before.



The test2 file has been removed from commit.

Now again using revert command to get the test2 file.

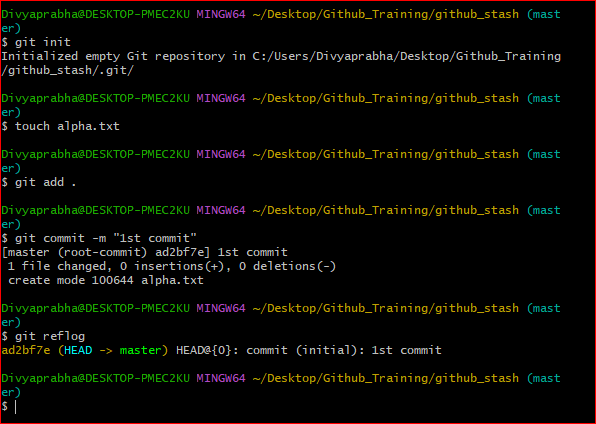


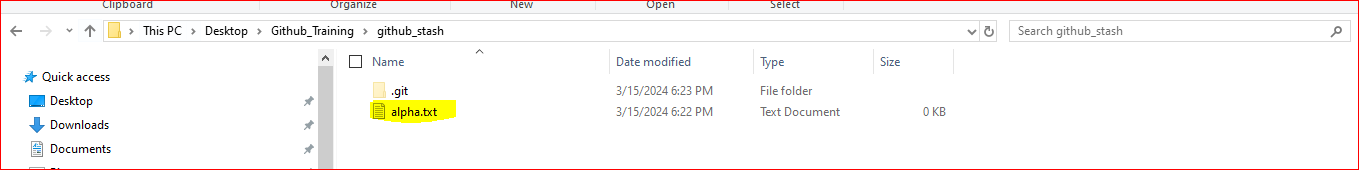


For multiple reverts, use the same command with commit ids in space.

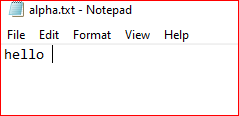
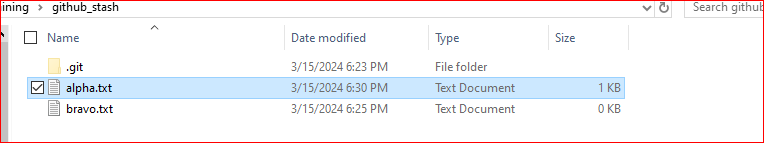
**Git revert 6f8ad59 0a6d0c9**

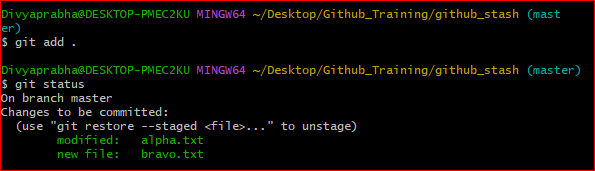
**Git Stash:**

Created new repo and added a text file in it.



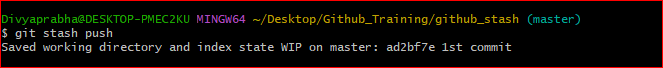
Now added new file bravo and edited alpha file.

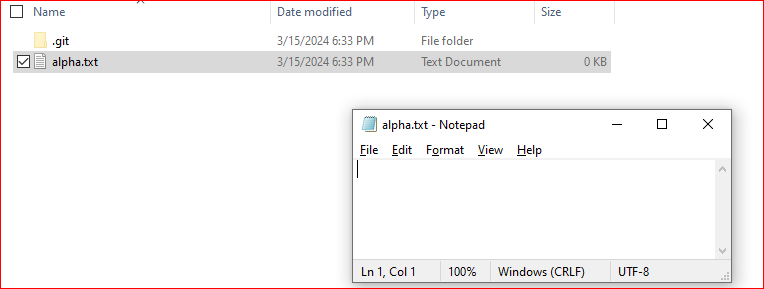


2 files has been added to staged.

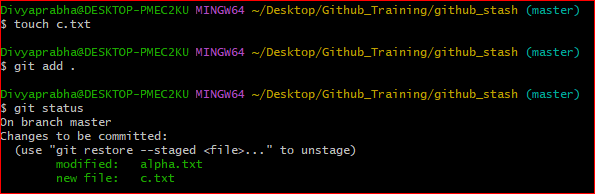
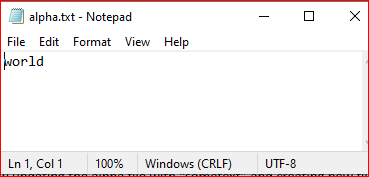
Now want to do some modification in alpha and at the sametime don’t want to delete bravo, so moving in a separate area to use it later. To achieve this, stash is used.

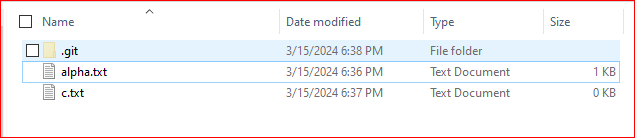


After stash command, the file is back to first commit state.

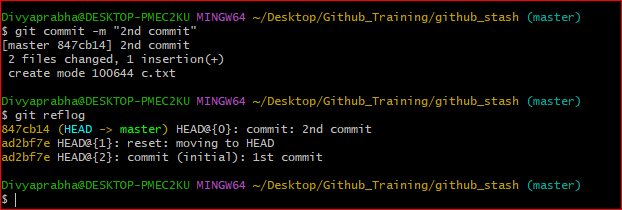


Now updating the alpha file with “sometext” and creating new file.

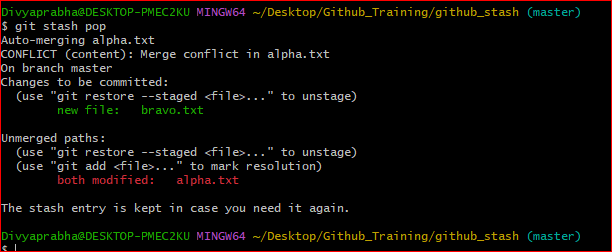


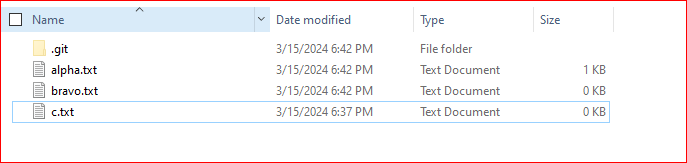


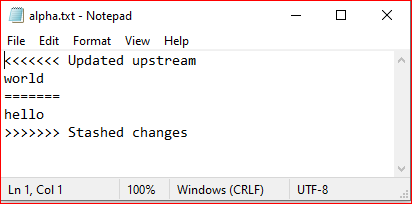
After 2nd commit:

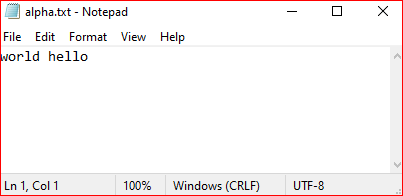


Now, getting back the records from stash:

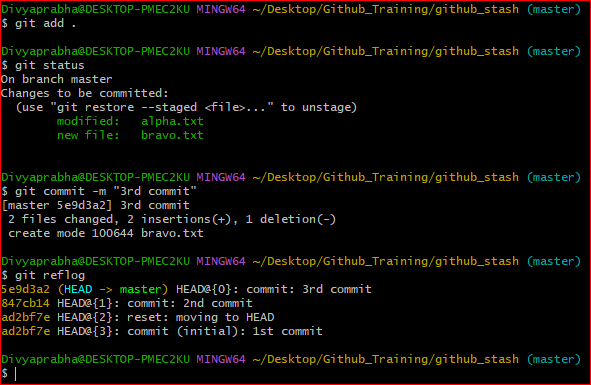


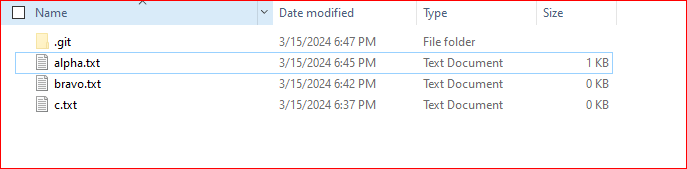


Alpha file has conflicts: 

Updated the text to below:

After updating the alpha, add, and commit the changes:





Git stash list 🡺 command to get the number of files in the stash area

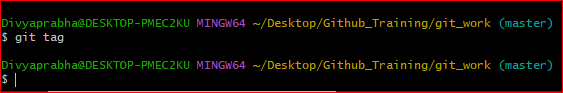
Git stash clear 🡺 command to clear the files in the stash area

Git stash pop 🡺 command to take the files from the stash to workspace area. This clears the entries in stash.

Git stash apply 🡺 command to take the files from the stash to workspace area, also keep the entries in stash area.

Git stash -u 🡺 command to move the untracked files to stash area

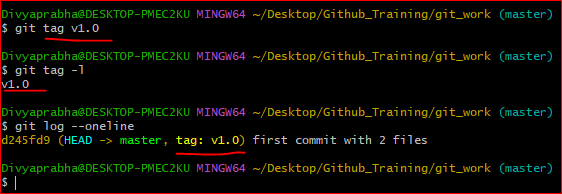
**Git Tags:**

Git has the ability to tag specific points in a repository’s history as being important. Typically, people use this functionality to mark release points (v1.0, v2.0 and so on). 

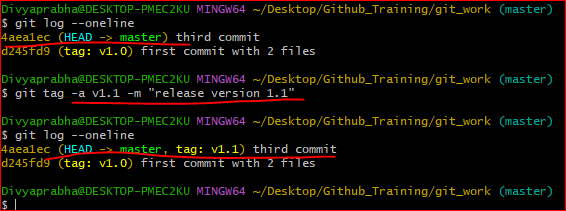
3 types of tags:

1. lightweight:
   * git tag v1.1
2. annotated:
   * git tag v1.1 -m "release version 1.1"
3. tagged later:
   * git tag v1.0 -m "release version 1.0" 6f8ad59

Lightweight tag:

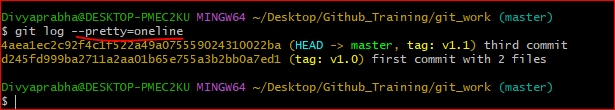
Lightweight tags are essentially pointers to specific commits.

Annotated Tag:

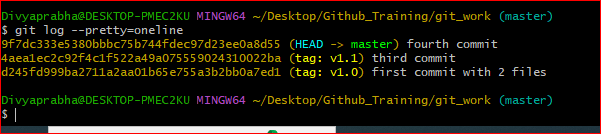
Annotated tags are similar to lightweight tags but contain additional metadata such as tagger name, email, date, and a message.

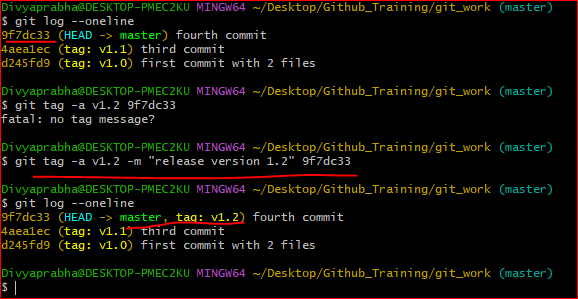
Git log gives the history of commit history.

--pretty=oneline🡺 specifies the format in which the commit history should be displayed. “oneline” format condenses each commit's information into a single line, showing only the commit hash and the commit message.

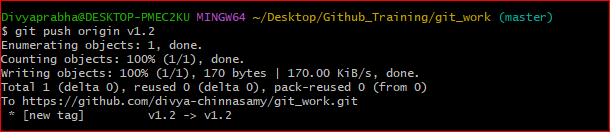


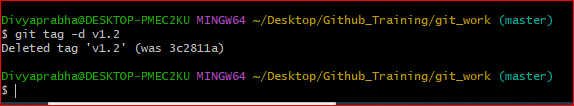
Tagging later:

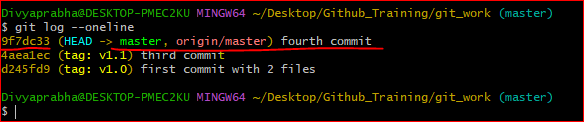
Fourth commit has not been tagged.

Tagging after commit:

Git push with tag name: Git push origin <tag\_name>

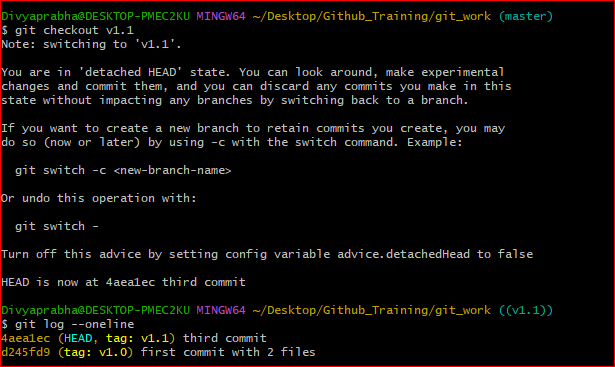


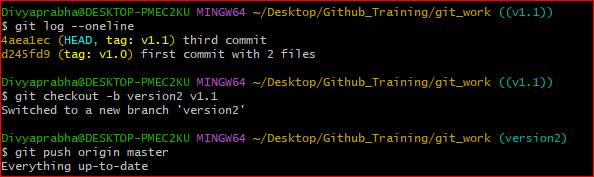
Delete tags:



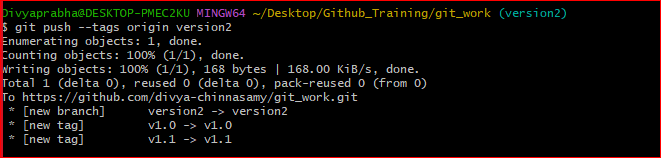
Checking out Tag:

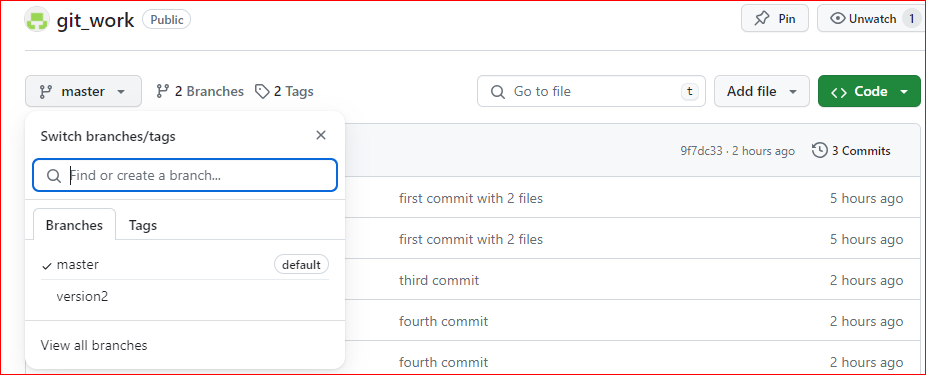
This command will switch your repository to the state it was in at the time of the tagged commit. It will put your repository in a "detached HEAD" state, meaning any changes you make won't be part of any branch. If you want to make changes based on this tag, you might want to create a new branch from the tag





Pushing to Remote repo:

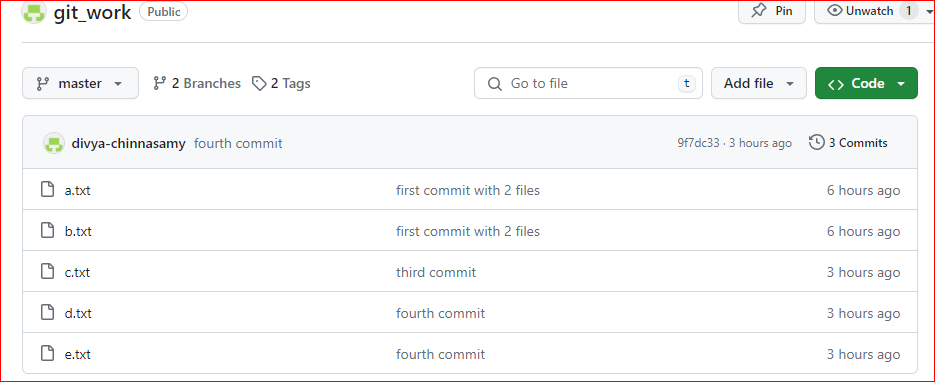
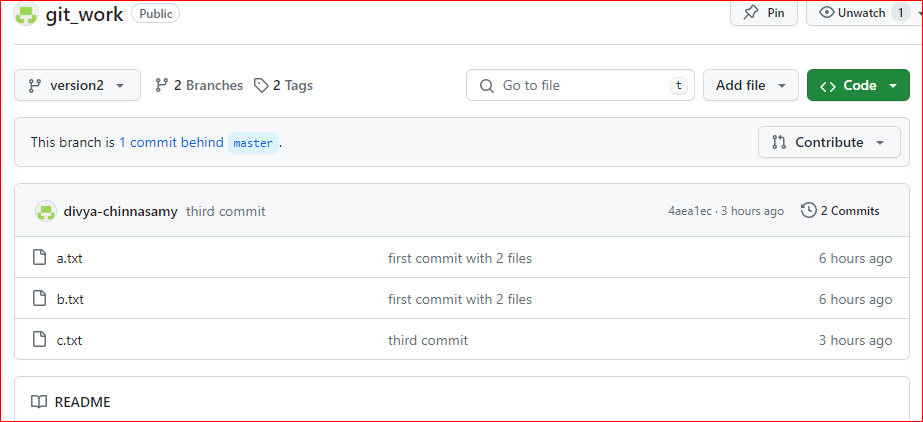


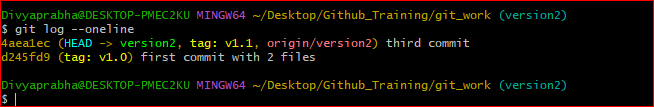
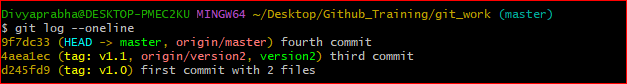
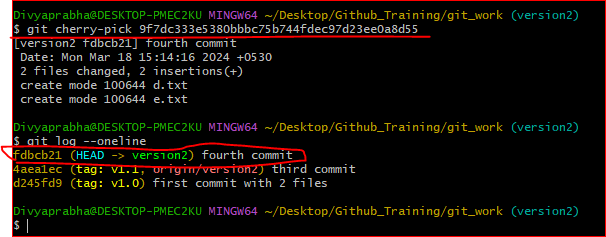


**Cherry pick:**

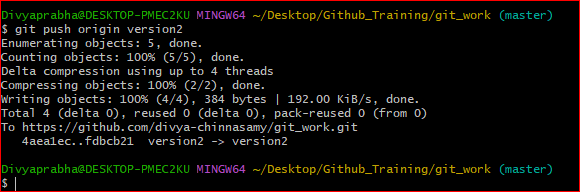
git cherry-pick is a powerful command that enables arbitrary Git commits to be picked by reference and appended to the current working HEAD. Cherry picking is the act of picking a commit from a branch and applying it to another. git cherry-pick can be useful for undoing changes.

Say we have 2 branch as given below:

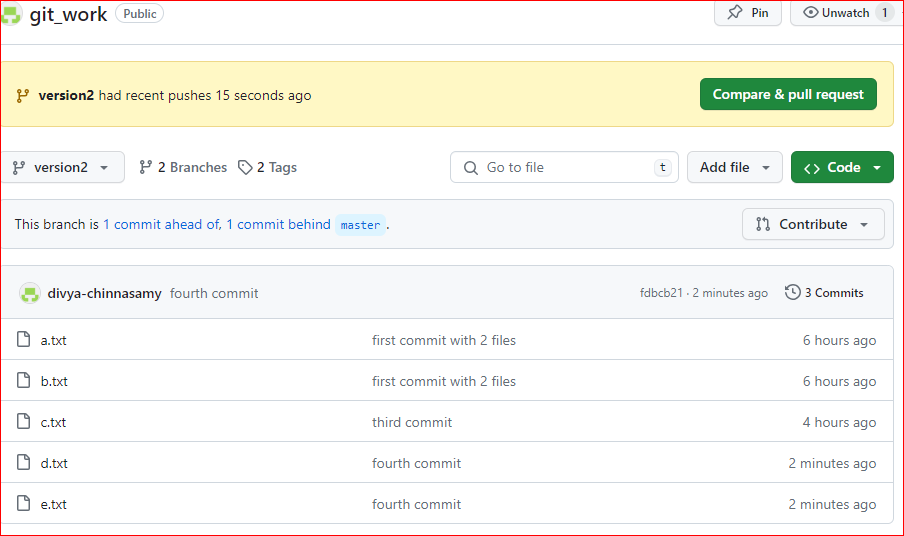


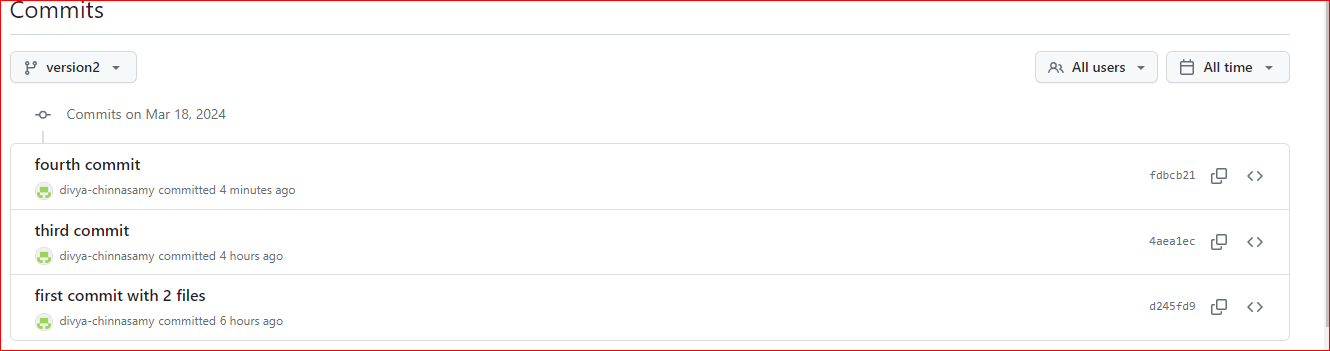
Making the fourth commit in version2 branch. So, checkout to version2 and cherry pick fourth commit

After git Pushing:



Now the fourth commit is available in version2 branch as well.





**Git Manifest File:**

In a manifest repository, there are one or more XML files, which define the relationship of all repositories of the project. The default manifest file in the manifest project is default.xml. See the following content of a manifest file as an example:

<?xml version="1.0" encoding="UTF-8"?>

<manifest>

<remote name="origin"

fetch=".."

revision="master"

review="https://codeup.aliyun.com" />

<remote name="github"

fetch="https://github.com"

revision="master"

review="" />

<default remote="origin"

revision="master"

sync-j="4" />

<project name="alibaba/git-repo-go" path="git-repo" groups="app">

<linkfile src="README.md" dest="README.md"></linkfile>

</project>

<project name="alibaba/git-repo-go-doc" path="website" groups="app"/>

<project name="jiangxin/goconfig" path="lib/goconfig" groups="lib" remote="github" />

<project name="jiangxin/multi-log" path="lib/multi-log" groups="lib" remote="github" />

</manifest>

The root element of the manifest XML file is manifest and has many other elements.