

GITHUB CLI COMMANDS

- Simple commands for creating/modifying any file/files and push that to remote repository.

^ **git init**

- ⌘ When you create a folder in your local machine, it is not a *local git repository* by default.
- ⌘ Use this command to make that directory a local git repository.
- ⌘ You must see a folder **.git**.
- ⌘ This folder keep the track of all things that is happening or was happened inside your repository.
- ⌘ Executing the **git init** command does nothing but create a **.git** folder in your directory.
- ⌘ Without that **.git** folder, you can't execute any git command like **git add** or **git commit** or anything.

```
alokr@Alok MINGW64 /c/my files/git_pract
$ ls -a
./ ../

alokr@Alok MINGW64 /c/my files/git_pract
$ git status
fatal: not a git repository (or any of the parent directories): .git

alokr@Alok MINGW64 /c/my files/git_pract
$ git init
Initialized empty Git repository in C:/my files/git_pract/.git/

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ ls -a
./ ../ .git/

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

- * You can see here, before **git init**, there was no **.git** folder, that's why I was unable to execute any git command.
- * After executing **git init**, one folder **.git** got created and then git got to know that this folder is a git repository.

• **git add**

- This command make the untracked changes (like adding any new file, or modifying any file, or deleting any file inside the working directory) as tracked.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ touch test1.txt
```

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test1.txt
```

- nothing added to commit but untracked files present (use "git add" to track)
- You can see, I created one file, so now it's showing as an **untracked** file as git doesn't know this file.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git add test1.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   test1.txt
```

- Now, I added this file using **git add** command, and now git was able to track this file. So, its telling to commit this file so that it'll be updated in the local repo.

➤ **git commit**

- ✎ This command will update the **tracked changes** (changes that was added using **git add** command) to your local repository (means inside **.git** folder) against a comment that you give while executing **git commit** command.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git commit -m "created a file test1.txt"
[main (root-commit) d1e1893] created a file test1.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test1.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git log --oneline
d1e1893 (HEAD -> main) created a file test1.txt
```

- ✎ I committed that change (means creation of file test1.txt) and when I executed the command **git log**, it displayed my commit and what was done in that commit.
 - * That's why giving a proper message while executing **git commit** is a good practice to know why that commit was made for.

➤ I created one git repository in the github (remotely) and I want to connect my this local repo to that remote repo. **(it has to be done once only)**

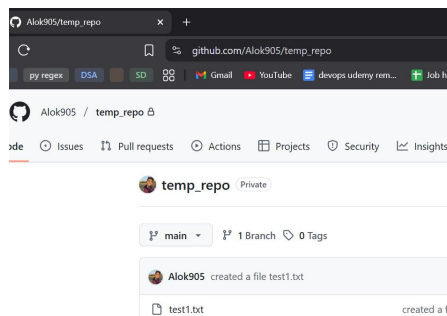
- ✎ **git remote add origin <url of the remote repo>**

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git remote add origin https://github.com/Alok905/temp_repo.git
```

➤ After connecting with the remote repo, you can push the changes into the remote repository.

- ✎ Till **git commit**, all the changes were happening locally.
- ✎ When you execute **git push**, it'll update the remote repository.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 225 bytes | 225.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Alok905/temp_repo.git
 * [new branch]      main -> main
```



✎

➤ Stages:

♣ Working directory:

- ✎ It is your local project where you add/edit/delete files.
- ✎ Changes that are not tracked after the previous commit.
- ✎ Mostly you'll see **modified** and **untracked** files.
- ✎ **git status**
 - ⌘ This command can be used to see the status of your current working directory.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ echo "hello" > test1.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ touch test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test2.txt

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

- ✎ I added some texts inside the *test1.txt* (which is already tracked by git) and created a new file *test2.txt*.
- ✎ As git knows *test1.txt* file, but it sees that something has been changed inside that file, so it marks it as **modified**.
- ✎ Where as, git doesn't know about the file *test2.txt* as it is created now only, so it marks it as **untracked**.

♣ Staging Area:

- ✎ After executing **git add** command, the untracked/modified files go to staging area.
- ✎ Its kind of a bucket where all the modifications are collected which will be submitted in the next commit.

- Suppose you added/modified something in your working directory and added those to staging area. And again added/modified something in the working directory but didn't stage those.
- * Now if you commit, then only the modifications those were staged will be committed. Not those which were modified but didn't stage.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ echo "hello" > test1.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ touch test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   test1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test2.txt

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

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git add .
warning: in the working copy of 'test1.txt', LF will be replaced by CRLF

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   test1.txt
    new file:   test2.txt
```

- I added the files *test1.txt* and *test2.txt* into staging area using command **git add .**

- **.** means you are staging all the changes that you made your working directory.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ echo "hello from test2.txt" > test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   test1.txt
    new file:   test2.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   test2.txt
```


- Now I added some text inside *test2.txt* and not staging it.
- Green* color texts shows what are inside the staging area, and *red* color shows the changes which are not staged yet.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git commit -m "created one file test2.txt; added some text in test1.txt"
[main 0177d5b] created one file test2.txt; added some text in test1.txt
2 files changed, 1 insertion(+)
create mode 100644 test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test2.txt

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

- Now you can see, only the staged modification got committed, the changed which were not staged are still there.

Local Repository:

- Whatever you commit, it stays in your local repository (you can think it is stored inside the *.git* folder)
- But the remote repository is still not aware about this.

Remote Repository:

- After the commit, when you push the changes, it goes to the remote repository.
- After this, your local repository and remote repository will be in sync.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 308 bytes | 308.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Alok905/temp_repo.git
d1e1893..0177d5b  main -> main
```

- Now it is updated in the remote repository.

- If you have made some changes in a file, and want to revert it back as it is there in the last commit then use **git restore** command.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ echo "Hellooooooooo" > test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ cat test2.txt
Hellooooooooo

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test2.txt

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

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git restore test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ cat test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
nothing to commit, working tree clean
```

- If you have staged some modification and want to undo those; means you want to transfer the changes from staging area to the working directory then you can use the command **git restore --staged <filename>**

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git restore --staged test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test2.txt

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

- ✶ Here, the changes in the file was staged. Then I executed **git restore --staged test2.txt** command and now the staged changes became unstaged.

- If you have made a commit but the code but now want to revert back to the previous commit.

- ♣ Let the commits are C1 -> C2 -> C3 -> C4 -> C5

- ♣ Latest commit is C5

- ♣ You want to go back to C3 (let)

- ♣ The command for doing this is: **git reset --hard <commit id>**

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git log --oneline
69abb8a (HEAD -> main) added one more text in test2.txt
a08975a added one text to test2.txt
05fd2a4 (origin/main) added some text in test2.txt
0177d5b created one file test2.txt; added some text in test1.txt
d1e1893 created a file test1.txt
```

- ♣ These are my current commits. I want to go back to *05fd2a4*.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git reset --hard 05fd2a4
HEAD is now at 05fd2a4 added some text in test2.txt

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git log --oneline
05fd2a4 (HEAD -> main) added some text in test2.txt
0177d5b created one file test2.txt; added some text in test1.txt
d1e1893 created a file test1.txt
```

- ♣ All the commits made after *05fd2a4* got deleted.

- ♣ You can see one text “HEAD is now at 05fd2a4 ...”

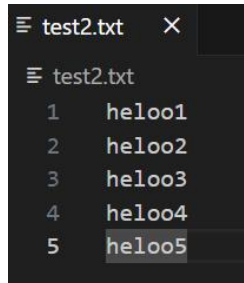
- ♣ **HEAD** is nothing but just one pointer that points to the latest commit.

- One more scenario; let you have some commits C1 -> C2 -> C3 -> C4 -> C5; you want to undo the changes made in the commit C5 but don't want to delete C5.

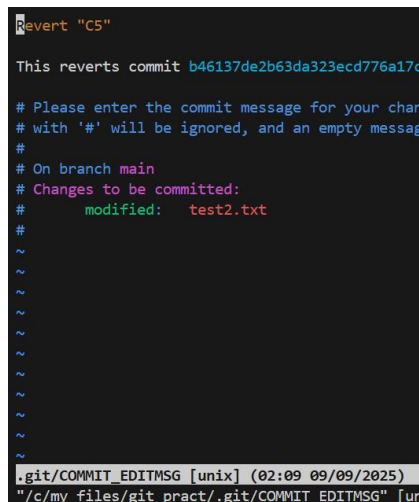
- ⌘ I just want to add one more commit which will contain the things undoing the changes done in C5.
- ⌘ Here, you can use **git revert** command.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git log --oneline
b46137d (HEAD -> main) C5
8d523cd C4
1da83f4 C3
1554579 C2
3943d52 C1
05fd2a4 added some text in test2.txt
0177d5b created one file test2.txt; added some text in test1.txt
d1e1893 created a file test1.txt
```

- ⌘ I made some changes in the file.



- * This is my file; Every line is a different commit. Means
 - ⌘ added *hello1* and committed. (C1)
 - ⌘ Added *hello2* and committed. (C2) ... and like this till C5
- ⌘ Now I want to undo the commit C5 (means I want to remove *hello5* from the file)
 - * The command will be: **git revert HEAD** (OR) **git revert b46137d**
 - ⌘ As HEAD is pointing to b46137d commit only (latest commit).
 - * After executing this command, one editor will be opened.



You can write the commit message inside this and save it.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git revert HEAD
[main 33b403c] Revert "C5"
1 file changed, 1 insertion(+), 2 deletions(-)

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git log --oneline
33b403c (HEAD -> main) Revert "C5"
b46137d (origin/main, origin/HEAD) C5
8d523cd C4
1da83f4 C3
1554579 C2
3943d52 C1
05fd2a4 added some text in test2.txt
0177d5b created one file test2.txt; added some t
d1e1893 created a file test1.txt
```

Now, one more commit got added undoing the changes of commit C5.

```
test2.txt
1 heloo1
2 heloo2
3 heloo3
4 heloo4
```

(hello5 was added in C5, which is not there now)

It is helpful if you don't want to delete the commit. Means later someone can see what had been done in that commit.

If you want to revert C3 here, it will give conflict; as after C3, you are changing the same file in commits C4, C5;

In simple term, C4 and C5 are built on top of C3 in this case.

So, it'll give conflict.

```
$ git revert 1da83f4
Auto-merging test2.txt
CONFLICT (content): Merge conflict in test2.txt
error: could not revert 1da83f4... C3
hint: After resolving the conflicts, mark them with
hint: "git add/rm <paths>", then run
hint: "git revert --continue".
hint: You can instead skip this commit with "git revert --skip".
hint: To abort and get back to the state before "git revert",
hint: run "git revert --abort".
hint: Disable this message with "git config set advice.mergeConflict false"
```

- If the later commits (C4, C5 in this case) would not be changing the same file that C3 was doing, then reverting would be successful.
- * All the changes in the commits C1 to C5 were made in *test2.txt*.
- * Now, I am adding one commit in *test1.txt*

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git log --oneline
e347927 (HEAD -> main, origin/main, origin/HEAD) C6: test1 commit
b46137d C5
8d523cd C4
1da83f4 C3
1554579 C2
3943d52 C1
05fd2a4 added some text in test2.txt
0177d5b created one file test2.txt; added some text in test1.txt
d1e1893 created a file test1.txt
```

↳ C6: did some changes in test2.txt file

- Now, if I try to revert the commit C5, then it'll work fine.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git revert b46137d
[main f5aba14] Revert "C5"
 1 file changed, 1 insertion(+), 2 deletions(-)

alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git log --oneline
f5aba14 (HEAD -> main) Revert "C5"
e347927 (origin/main, origin/HEAD) C6: test1 commit
b46137d C5
8d523cd C4
1da83f4 C3
1554579 C2
3943d52 C1
05fd2a4 added some text in test2.txt
0177d5b created one file test2.txt; added some text
d1e1893 created a file test1.txt
```

- ↳ It worked because the commit C5 was not hindering the commit C6; because both were made in different files.

➤ **git pull**

- ⌘ If the remote repository has been updated with some extra commits by other collaborators i.e.
 - ⌘ Local repo : C1 -> C2 -> C3 -> C4 -> C5
 - ⌘ Remote repo : C1 -> C2 -> C3 -> C4 -> C5 -> C6 -> C7
 - ⌘ In this case, you need to update your local repo with the remote repo and then do your required changes. Otherwise it'll give conflicts while pushing to the remote repo.

- ⌘ Command: **git pull origin <branch name>**

- * If I want to update my **main** branch then: **git pull origin main**

- ⌘ Conflict scenario (assume every changes are being done in a single file):

- ⌘ Initial condition:

- * Local repo : C1 -> C2 -> C3 -> C4 -> C5

- * Remote repo : C1 -> C2 -> C3 -> C4 -> C5

- ⌘ I deleted a line in remote repo and committed; But in local repo added some extra texts in that same line and committed;

- * So now, both will conflict; as the same line has been changed in both local and remote repo.

- * For example:

- heloo1

- heloo2

- heloo4

- ⌘ heloo5 (remote repo; deleted the line "hello3")

- heloo1

- heloo2

- hello3_extra

- heloo4

- ⌘ heloo5 (added some extra text on the same line "hello3_extra")

- * Local repo : C1 -> C2 -> C3 -> C4 -> C5 -> Cx

- ⌘

```
a1okr@Alok MINGW64 /c/my files/git_pract (main)
$ git log --oneline
a047cb5 (HEAD -> main) added something in line3
e347927 C6: test1 commit
b46137d C5
8d523cd C4
1da83f4 C3
1554579 C2
3943d52 C1
05fd2a4 added some text in test2.txt
0177d5b created one file test2.txt; added some t
d1e1893 created a file test1.txt
```


commits: e347927 -> **a047eb5**

Remote repo : C1 -> C2 -> C3 -> C4 -> C5 -> Cy

removed line-3 (hello3)



Alok905 pushed 1 commit to **main** • e347927...6c75a2f • 2 minutes ago

commits: e347927 -> **6c75a2f**

Now, in local and remote repo, latest commits are different.

```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git pull origin main
From https://github.com/Alok905/temp_repo
* branch          main          -> FETCH_HEAD
Auto-merging test2.txt
CONFLICT (content): Merge conflict in test2.txt
Automatic merge failed; fix conflicts and then commit the result.
```

```
heloo1
heloo2
Accept Current Change | Accept Incoming Change | Acc
<<<<<<< HEAD (Current Change)
heloo3_extra
=====
>>>>>> origin/main (Incoming Change)
heloo4
heloo5
```

You'll get something like this. Need to fix the merge conflict and then merge.

So, its better to pull the changes from the remote repo before working to avoid the merge conflicts.

Branches

➤ Some basic commands:

^ **git branch <new branch name>**

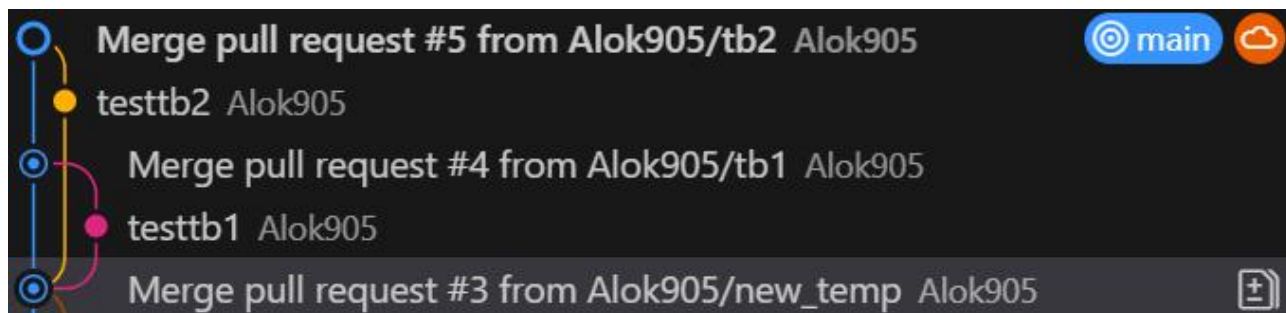
ε It'll just create a branch but doesn't switch to that new branch.

^ **git switch <branch name>**

ε It'll switch to a pre-existing branch.

^ **git checkout -b <new branch name>**

ε It'll create a branch and switch to that branch.



(Lets analyse these things ...)

^ **main** branch has commits: C1 -> C2 -> C3 -> C4 -> C5

^ I created 2 branches at this point **tb1**, **tb2** (you can see in the image)

ε In both the branches created one file i.e. in **tb1** one file and in **tb2** another file.

ε Staged those changes, committed and pushed.

ε Created 2 merge requests (one per each branch)

^ So now:

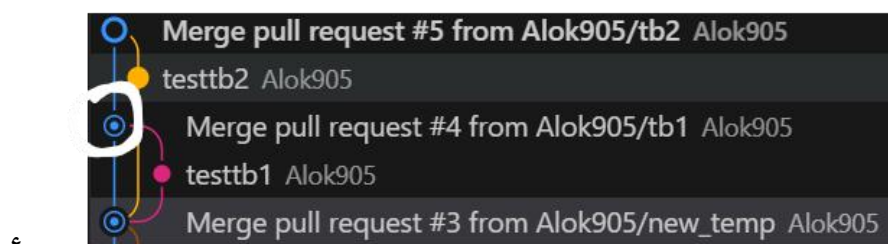
ε **main** : C1 -> C2 -> C3 -> C4 -> C5

ε **tb2** : C1 -> C2 -> C3 -> C4 -> C5 -> C_x

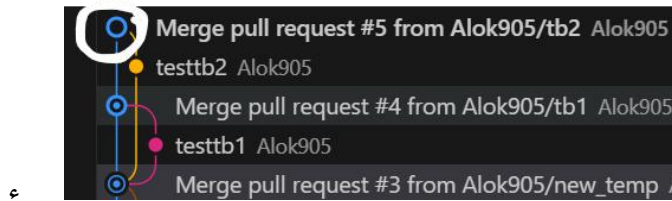
ε **tb3** : C1 -> C2 -> C3 -> C4 -> C5 -> C_y

^ I merged **tb1** with the main branch (accepted the merge request).

ε You can see **tb1** was merged to the main branch, so that one commit got created in main branch.



- After that I merged **tb2** to the main branch. So one more commit got created in the main branch.



- As these 2 branches (tb1 and tb2) were not interfering with each other or main branch's changes; so they got merged easily without any conflict.

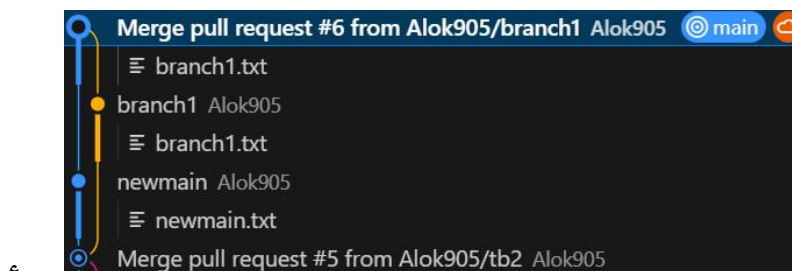
➤ One more case:

- Initially:

- main : C1 -> C2 -> C3 -> C4 -> C5
- branch1 : C1 -> C2 -> C3 -> C4 -> C5

- After this:

- main : C1 -> C2 -> C3 -> C4 -> C5 -> Cx
- branch1 : C1 -> C2 -> C3 -> C4 -> C5 -> Cy
- Here, main branch created one file (let) and branch1 created one file (means no conflict)



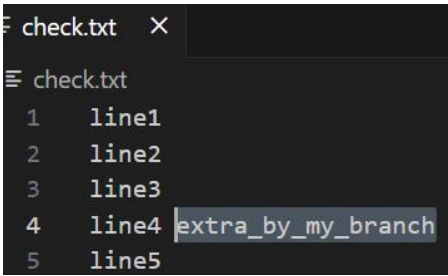
- Between creating of branch **branch1** and merging it to **main** branch, there is one commit present in **main** branch. But **branch1** was not updated with that commit but still it got merged to **main** branch without any conflict.
 - Because, **only the changes mentioned in the merge request** that was created from the branch **branch1** will be updated to the main branch; **not** all the files present in **branch1** will be updated in **main** branch.
- Let there is a file **check.txt** inside the main branch having some contents inside it.

```

≡ check.txt X
≡ check.txt
1 line1
2 line2
3 line3
4 line4
5 line5

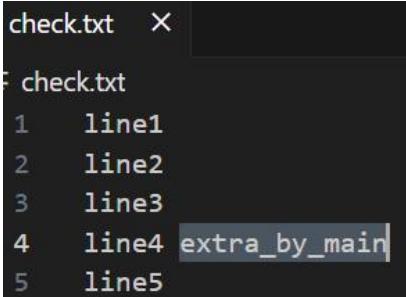
```

- Created one branch **my_branch** and made the following changes:



```
check.txt X
check.txt
1 line1
2 line2
3 line3
4 line4 extra_by_my_branch
5 line5
```

- Switched to **main** branch and made the following changes:



```
check.txt X
check.txt
1 line1
2 line2
3 line3
4 line4 extra_by_main
5 line5
```

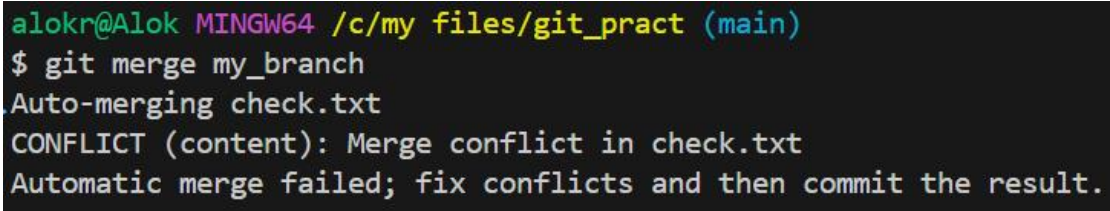
- Now we can see, both the changes are conflicting each other.

- Main : C1 -> C2 -> C3 -> C4 -> C5 -> **Cx**

- my_branch : C1 -> C2 -> C3 -> C4 -> C5 -> **Cy**

- But in this case, the commits **Cx** and **Cy** are conflicting each other.

- So, If I try to merge **my_branch** to **main** branch, then it'll give conflicts:



```
alokr@Alok MINGW64 /c/my files/git_pract (main)
$ git merge my_branch
Auto-merging check.txt
CONFLICT (content): Merge conflict in check.txt
Automatic merge failed; fix conflicts and then commit the result.
```