→ Traversing in your commits & changing head :-

The last commit you have made on a given branch is where your head is.

If you run command git log you can see head written on side of the hash of that commit.

Now, what if you want to see the files in some of your previous commit, for that you need to run command

git checkout <hash value of that commit>

for this hash code run 'git log --one line'

After running this (ead your files will be in the state of

When you run this command it will change the position of your head to that commit state of your repository.

If you want to come back to your original state run command

```
git switch -
```

or

```
git checkout master
```

Let you are at previous commit and want to change something is your code. If you do that and commit that commit will be no where on master branch as, already there are commits there after that particular commit so, it will ruin the order.

So, to do so what you have to do is after commiting at previous commit run

```
git switch -C <new branch name>
```

it will create new branch after that commit and will store all changes done in that and you will have your master branch.

NOw, what will you do if you actully fall back to some previous commit not only to watch files but actully go back in that position so that all commit after that is deleted.

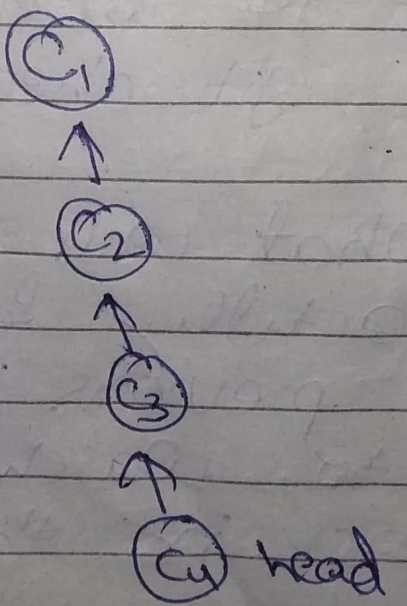
By using git reset --soft <hash of commit> you will fall back to that commit but your will not lost but will be in staged area.
earlier work

for that run command

git reset --hard <hash of that commit>

also note that as you fall back to that commit you written hash of, but also all commit after that you have made earlier will be deleted.

For example you have made four commit



Currently your head is at C_4 as it is your last commit.

If you run command

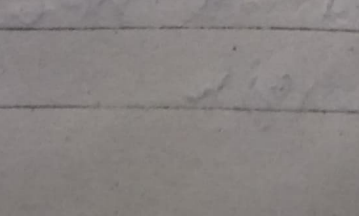
git checkout <hash of C_2>

you will see files in the state of C_2, but you can't do commit because it is not in any branch, it is just one commit.

If you run command

git reset --hard <hash of C_2>

you will fall back to C_2, all your files will be in state of C_2, also C_2 will be head and last commit, as (3) & (4) will be deleted.



→ How to remove remote repository from your git repository and adding new remote repository :-
If you are working on project with remote repository 'demo' and you want to remove that repository and add new remote repository, first you have to remove that repository by running cmd

git remote remove demo

now if you check git remote it will show nothing, to add new remote run command

git remote add origin <URL of remote repository>

It will add a new remote repository origin

• Grit show command!-

Grit show command helps us to see what changes have done is a particular commit for that you have to write

Grit show <hash of that commit>

→ Merging branches in Grit!-

Let we have two branches one master branch and another main branch, and we have to merge them, for this remember these rule-

- Firstly both branches should be clear.
- Come in that branch in which you went to add/merge other branch. In current case master branch.

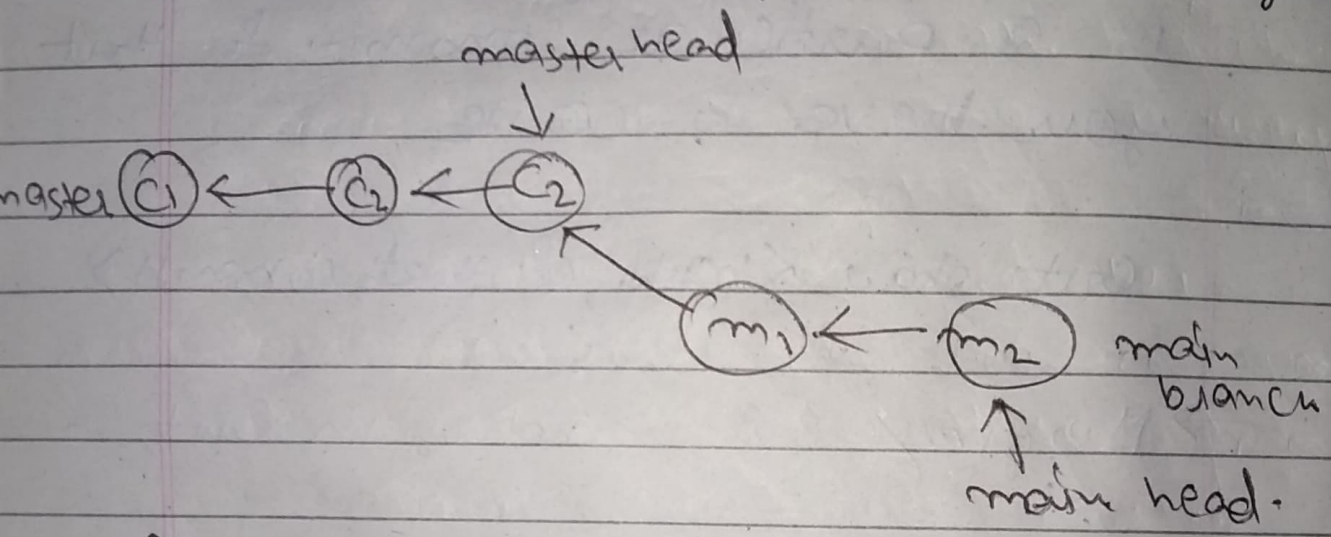
then run command

Grit merge <other branch name>

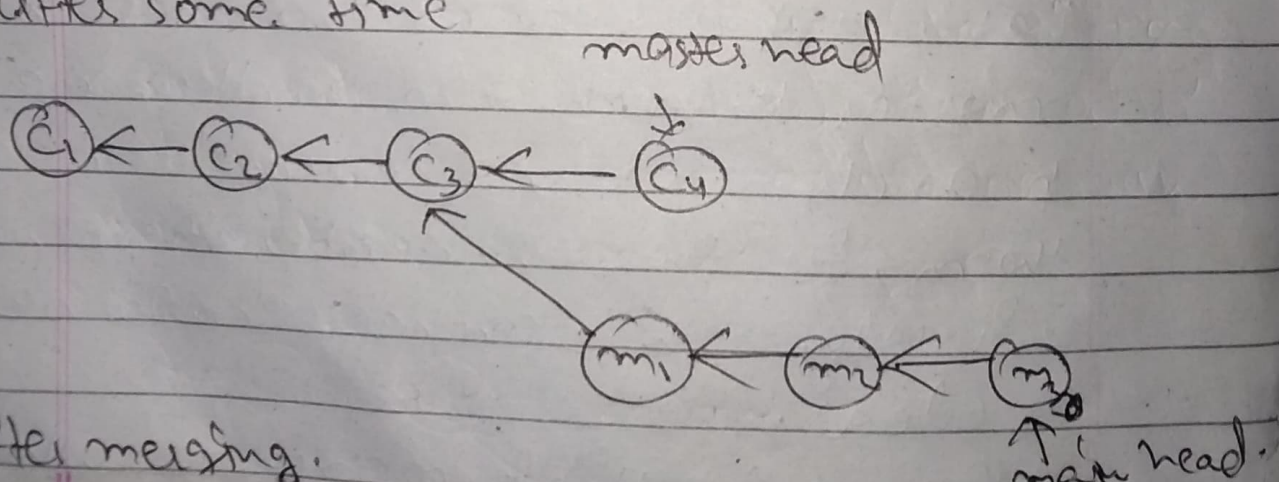
after merging you have to commit.

So in this case
Crit merge main

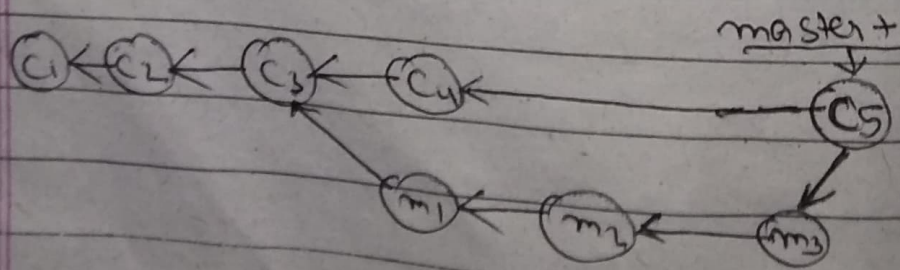
Let's see what happens with diagram



after some time



after merging.



Some times merge conflicts also arises during merging, which we have to resolve.

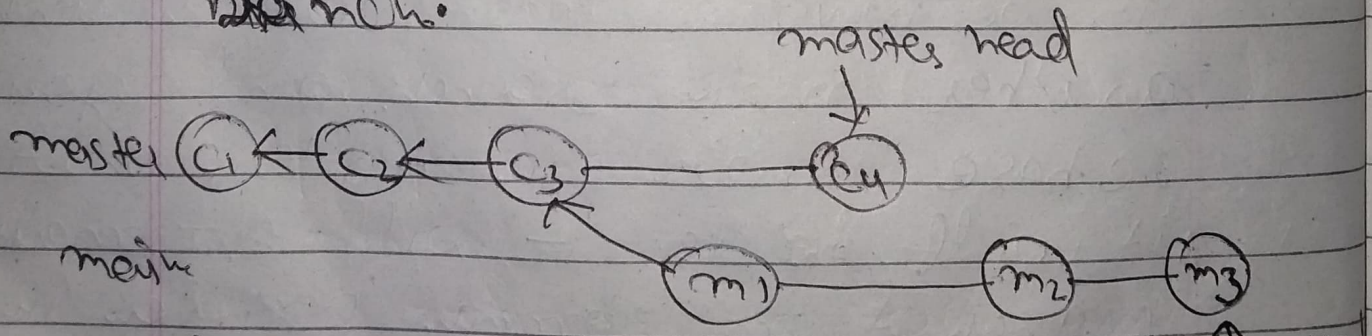
We can choose between changes or keep them depending upon us.

→ Rebasing in git :-

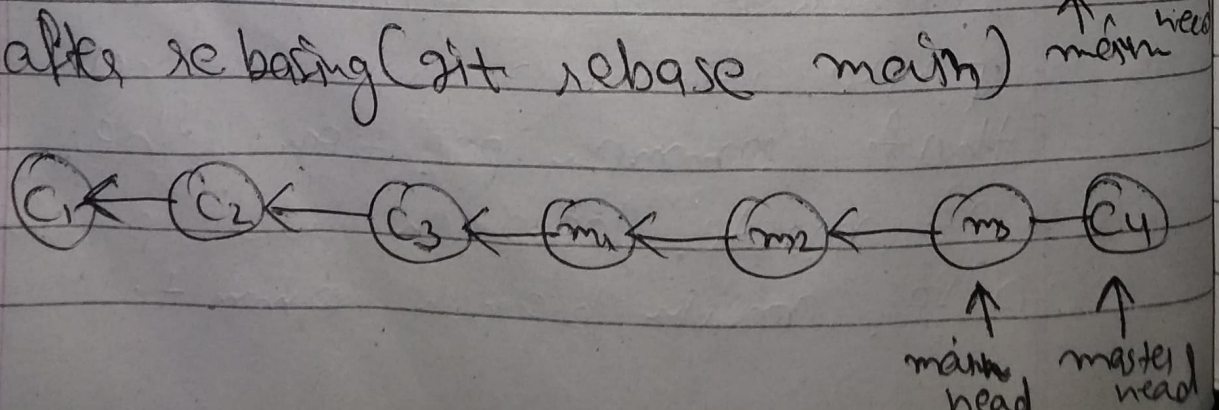
Rebasing gives similar types of result as merging but internal working is different.

If we have a master branch and another main branch and you rebase main in master, what it will do is it will cut main branch from comman position in paste it in master branch after that comman position. Other commits of master branch will placed after main branch commit.

here confusing thing is that it changes the history or order of the commits happen. Other commits of master branch which are placed after main branch commits are not same as they are before. They have same data and changes but their hash value changes. Some time it is confusing as these commits are made before main branch commit (they store time and date of creation) then also they are placed after main branch.



after rebasing (git rebase main)



advantages:-

It makes log/history of commit clearner, helps us to keep track of it easily.

disadvantages:-

It changes/rewrites the history of commit or order of commit, it might cause confusion among developer working on same projects.

→ reflog:-

git reflog is another command of git which keeps history what happens in the given repository till now according to how head moves.

It is different from git log.