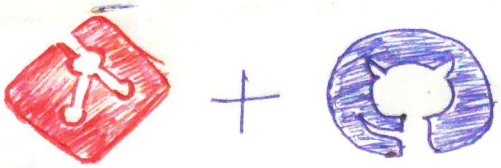


Page :
Date :

--Git & Github--

→ What is Git?

↳ Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then. It is used for:

- → Tracking code changes
- → Tracking who made changes
- → Coding collaboration

→ Allows us to maintain history of project at what particular time, which person made which change where in the project.

→ Version control system → aka source control, is the practice of tracking and managing changes to software code. Or Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

→ Why Git?

- → Over 70% of developers use Git!
- → Developers can work together from anywhere in the world.
- → Developers can see the full history of the project.
- → Developers can revert to earlier versions of a project

What is Github?

→ Github is a platform, an online website that allows us to host our Git Repositories.

→ Repository is just folder where all the changes are saved.

Page :
Date :

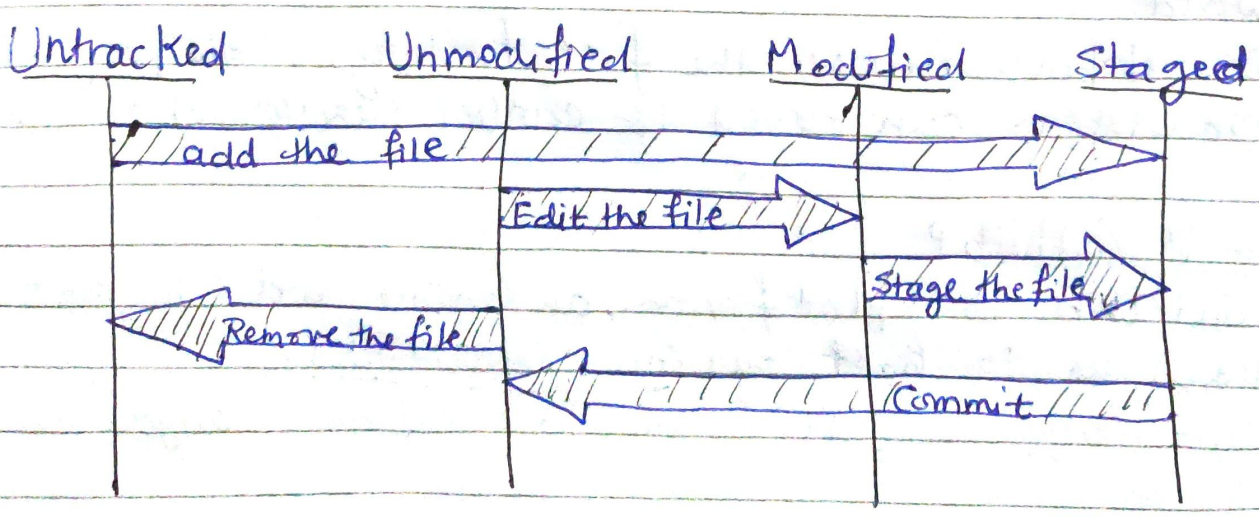
→ like Github there are platforms that allows you to host your repository :
  → Bitbucket
  → Gitlab etc.

→ Git is not the same as Github.
→ Github makes tools that use Git.
→ Github is the largest host of source code in the world, and has been owned by Microsoft since 2018.

→ Configuring git for the first time :-
$ git config --global user.name "< Enter your user name >"
$ git config --global user.email "< Enter your email >"

⇒ General Git features :-
→
Initializing Git :-
$ git init
↳ Git now knows that it should watch the folder you initiated it on. Git creates a hidden folder (.git) to keep track of changes.

The three stages of Git.



Page :
Date :

Untracked file are file which if shared on Github no one knows that the file is added there by user on that particular time.

→ Status of files
$ git status
↳ it shows ok, displays the State of the working directory and the staging area.
It let you see which changes have been staged, which haven't, and which files aren't being tracked by Git.

Staging files / Adding files to Git repo :-

↳ Staged files are files that are ready to be committed to the repository you are working on.
When you first add files to an empty repository, they are all untracked. To get track them, you need to stage them, or add them to the Staging environment. And we do that via
$ git add <filename with extension>
⇓
you can write individual file that you want to ~~com~~ add to Staging area. Or.

$ gitadd --all
or
$ git add -a
or
$ git add .
} Staging all files to Staging area.

Page :
Date :

Making a Commit :-

Adding commits keep of our progress and changes as we work. Git considers each commit change point or "Save point". It is a point in the project you can go back to if you find a bug or want to make a change.

When we commit, we should always include a message.

$ git commit -m "<Enter you message here>"

⇒ Git Commit without Stage :-

Sometimes, when you make small changes, using the staging area/environment is like a waste of time. It is possible to commit changes directly, skipping the staging environment.

$ git commit -a -m "<Enter your message here>"

→ Log of a file :-

↳ Log is used to view the history of commits for a repository.

$ git log

$ git log --oneline

→ to get out of it press q/Q on keyboard.

⇒ Adding Something to your file :-

$ vi "<file name>

and to go outside of it after making changes press in windows esc and the shift + ; and then x. And go outside of it.

→ Check what is available in a file :-

$ cat "<file name>"

Page :
Date :

⇒ If you want to remove any file without commiting it we can do that via:
$ git restore --staged "<file name>"

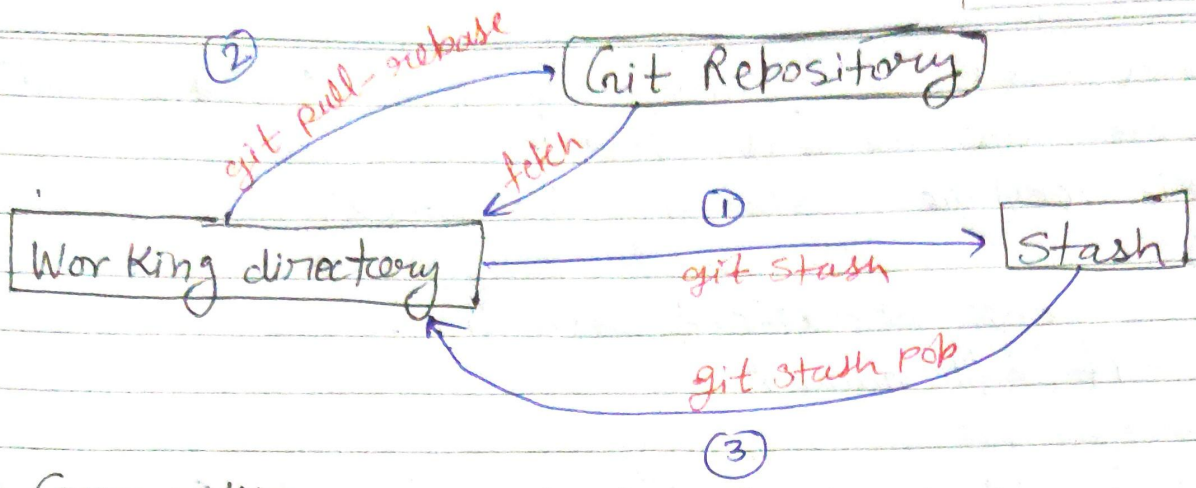
⇒ If we want to "delete any file".
$ rm -rf "<file name>"

⇒ Now If we want "delete any Commit" from your history of project.
→ each commit has hash id / commit id which is of 40 alphanumetic
→ we can't delete any commit from middle because each commit built on top of each other.
Now we want to delete any commit just copy the hash id / commit id of the bellow commit of the commit and than
⇒ $ git reset "<past the hash id / commit id here>"

⇒ After deleting the commits, you wondering that what happened to all the files that were modified or changed or whatever in the previous commits they are now in unstaged area.

$ git status

⇒ Git Stash
↳ git stash temporarily shelves (or stashes) changes you've made to your working copy so you can work on something else, and then a user can retrieve all files put in the stash with git stash pop and .git stash apply commands.



⇒ Commands:-
$ git stash
and to retrieve
$ git stash pop

⇒ and for delete your files which are stored in Stash area, the command is:
$ git stash clear
↳ And now those are gone we can't back them.

Working with Github :-

↳ Create a github account to create your remote repositories. Now, create a new repo where we will be uploading

Notes:- Local repository (repo) mean the repo. which is on our system whereas, remote repo. means the one which is on other remote system/server, for eg. - Github, Gitlab, Bitbucket, etc

Now after creating a new repository.

Page :
Date :

Push local repo to Github:-

Now copy the url or the link of the repo that we just created. As an example
[https://github.com/Suraj-Kumar/Learning-Git-Github]
and we want this URL to be attached to ~~one~~ local repository.

⇒ $ git remote add origin <paste copied URL here>
→ remote just mean you are working with URL
→ add basically means you are adding new URL
→ Origin means what is the name of the URL going to be that you are going to add.

⇒ $ git remote -v
↳ means it show all the urls attached ~~with~~ to the folder that you made.

⇒ $ git push origin master
{this is the name of URL}
↳ branch name

↳ Here we are pushing local repo to github

⇒ Git Branching:-

In Git, a branch is a new/separate version of the main repository. Branches allow you to work on different parts of a project without impacting the main branch. When the work is completed, a branch can be merged with the main project.

→ Making a new Git Branch-
$ git branch "<name of branch>"

Page :
Date :

Checking all available Branches:-
$ git branch

Switching to other Branches:-
$ git checkout <branch name>

- Making a new branch and directly switching to it:-
$ git checkout -b <branch name>

→ Deleting a Branch
$ git branch -d <branch name>

→ Merging two Branches :-
→ It's preferred to change / switch to master branch before any branch needs to be merged with it.
⇒ $ git merge <branch name>

This will merge the specified branch with our master branch.

⇒ Git Clone from Github:-
→ We can clone a forked repo from Github on our local repo. A clone is a full copy of a repository, including all logging and versions of files. Move back to the original repository, and click the green "Code" button to get the URL to clone. Copy the URL.

→ $ git clone <copied URL>

Page :
Date :

→ To specify a specific folder to clone to, add the name of the folder after the repo URL,
→ $ git clone <copied URL> <folder name>

⇒ Why we have to fork the repo, because we can't directly make changes to anyones projects.

⇒ upstream URL → from where you have forked this project means URL, known as upstream URL by convention.

What is pull Request (PR) ?
→ When you create your own copy of others project or repository and you create any change in your copy then how do you make sure whatever create in your copy is visible in the main branch/project so you request via a "pull Request". Then people review your code, suggest some changes you will make those changes, when this is merged your code sample changes that you made in your own account's fork in any branch that will be visible in the main project, main branch.

⇒ * one pull request means one branch, one branch can only open one pull request

⇒ To delete a repository we have to force push that
→ $ git push origin :<branch name> -f

* ↳ this right i mistackaly cross it ☹.

Page :
Date :

⇒ To fetch all changes or commits →

$ git fetch --all --prune

↓

Prune means that the ones are deleted will also be fetched.

→ To merge all commit in one single commit.
we use "rebase" for that

$ git rebase -i <copy of commit id / hash id>

after that you will get commands :-
there are lots of commands but i'm using here few usable :-

↳ p, pick <commit> = use commit

↳ r, reword <commit> = use commit, but edit the commit message.

↳ e, edit <commit> = use commit, stop for amending

↳ s, squash <commit> = use commit, but meld into previous commit.

↓

means → merge

⇒ Merge Conflicts

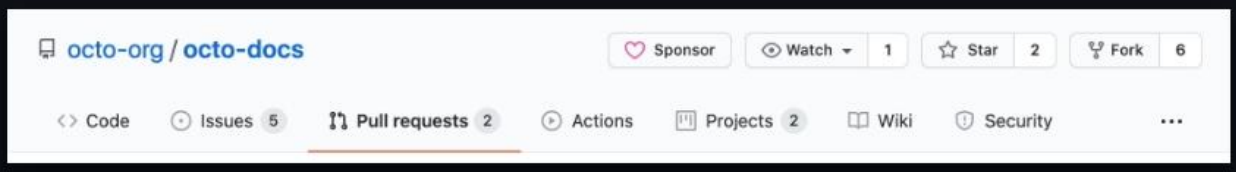
↳ In a file lets say you mad a change in the number 3 and someone else made change in the number 3, now git will get confused that should i take your or someone else changes so git will ask you to help that you want to take this chage or that change or

→ It occurs when also, when one person edits a file and another person delete the same file.

→ lets see how we resolve / fix this :-

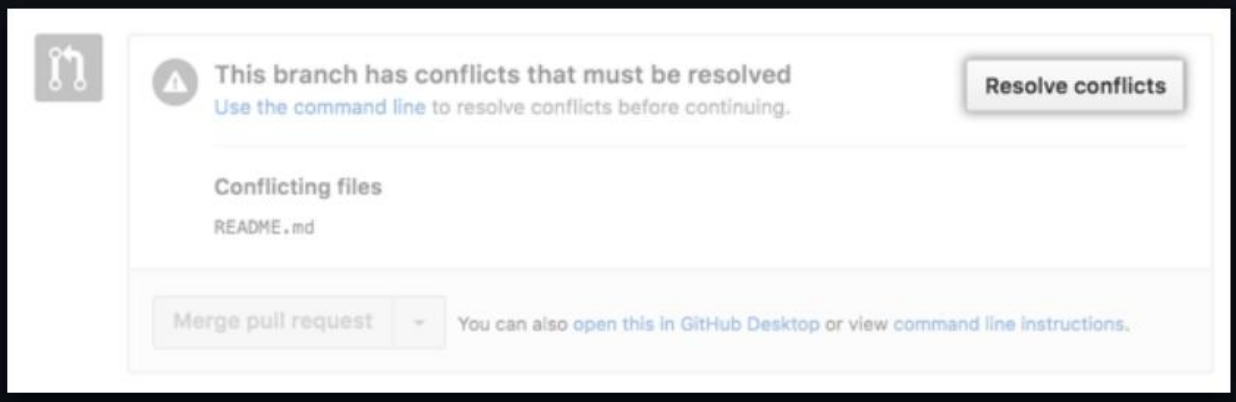
→ We have to resolve it manually.

1. Under your repository name, click **Pull requests**.

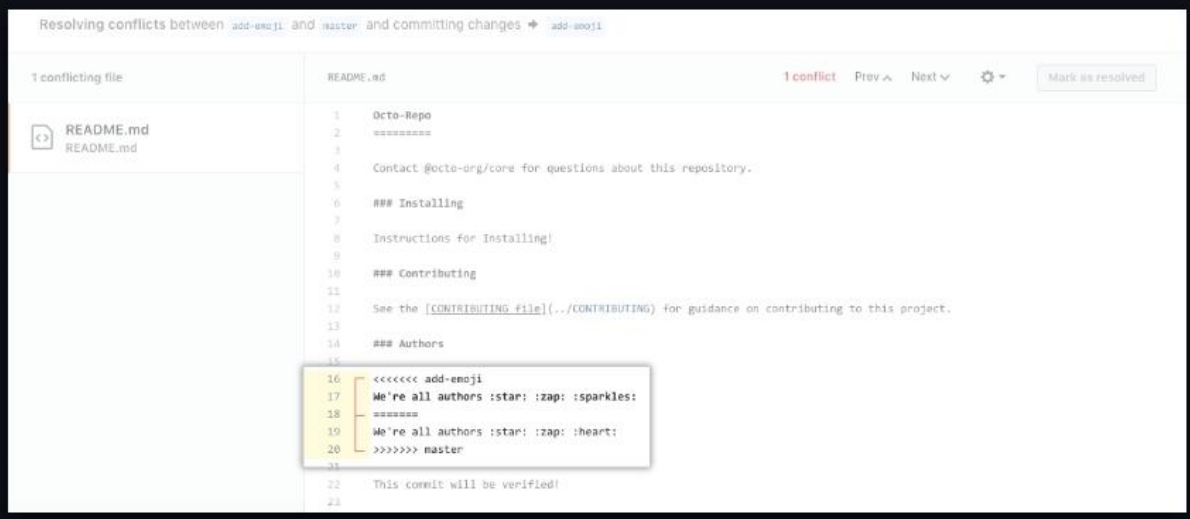


2. In the "Pull Requests" list, click the pull request with a merge conflict that you'd like to resolve.

3. Near the bottom of your pull request, click **Resolve conflicts**.



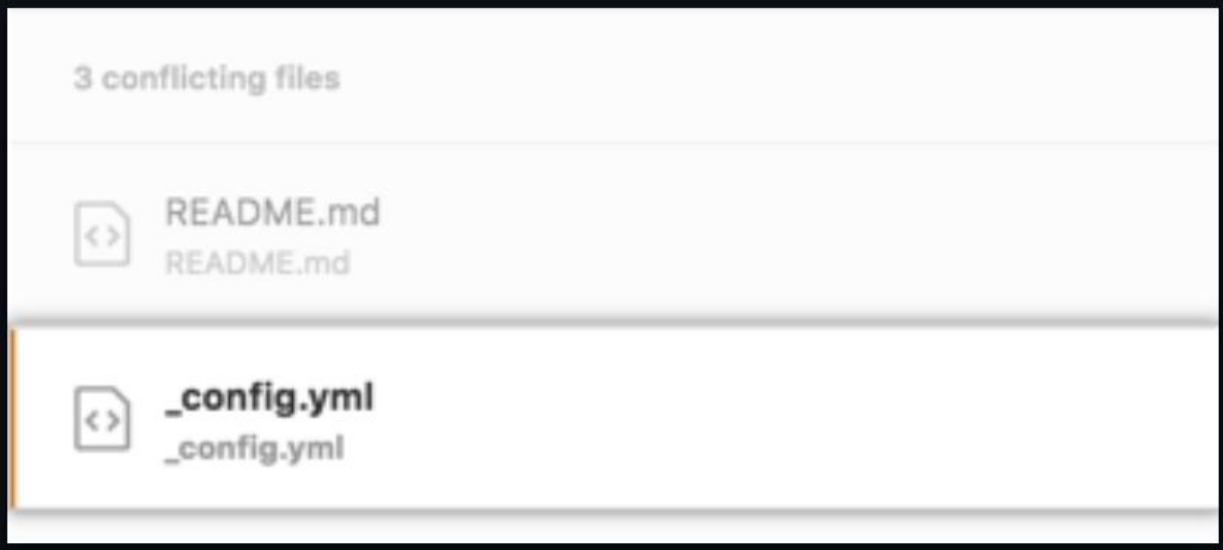
4. Decide if you want to keep only your branch's changes, keep only the other branch's changes, or make a brand new change, which may incorporate changes from both branches. Delete the conflict markers `<<<<<<<`, `=======`, `>>>>>>>` and make the changes you want in the final merge.



5. If you have more than one merge conflict in your file, scroll down to the next set of conflict markers and repeat steps four and five to resolve your merge conflict.

6. Once you've resolved all the conflicts in the file, click **Mark as resolved**.

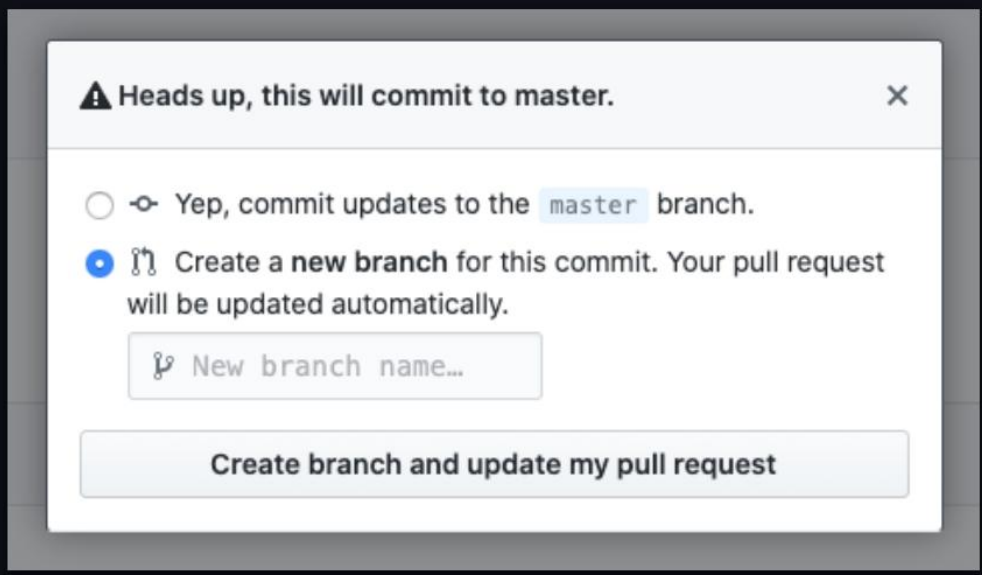
7. If you have more than one file with a conflict, select the next file you want to edit on the left side of the page under "conflicting files" and repeat steps four through seven until you've resolved all of your pull request's merge conflicts.



8. Once you've resolved all your merge conflicts, click **Commit merge**. This merges the entire base branch into your head branch.

9. If prompted, review the branch that you are committing to.

   If the head branch is the default branch of the repository, you can choose either to update this branch with the changes you made to resolve the conflict, or to create a new branch and use this as the head branch of the pull request.



   If you choose to create a new branch, enter a name for the branch.

   If the head branch of your pull request is protected you must create a new branch. You won't get the option to update the protected branch.

   Click **Create branch and update my pull request** or **I understand, continue updating *BRANCH***. The button text corresponds to the action you are performing.

10. To merge your pull request, click **Merge pull request**. For more information about other pull request merge options, see "Merging a pull request."

Page :
Date :

Git Reset :-

'reset' is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.

→ $ git reset <commithash>

⇒ " git reset --hard origin/upstream/main/master
↳ this can be utilized to stage and unstaged changes. It delets all the changes made on the current local branch, making it the same as the origin/master, and reset the HEAD pointer.

→ $ git reset --hard upstream/main

→ Git Amend :-

Commit--Amend is used to modify the most recent commit. It combines change in the staging area with the latest commit, and creates a new commit. This new commit replaces the latest commit entirely.

→ $ git commit --amend -m "Commit message"

→ Github → Suraj-Kumar00
→ Linkedin → Surajkumar00