"ॐ"

# [GIT & GITHUB]

→ Initalizing a Git Repository
How do we get this folder (.git) which is hidden & in which all the history is saved.
→ Intialize empty git repository using: git init
→ How do we see what all files have been changed or modified or added or deleted i.e; what all changes have been made in the project that are not currently saved in the history of that project.
⇒ git status .
→ To save all the changes in the history we use:
⇒ git add . : Put all the untracked files in the staging area
⇒ git commit -m "names.txt file added"
⇒ vi names.txt : To open file in vi or vim editor
⇒ cat names.txt : Displays all the content inside the file.
→ If we want to unstage some files without or before commiting them:
git restore --staged names.txt
→ Viewing the overall history of the project:
git log
→ Remove a file : rm -rf names.txt
→ Removing a commit from the history of a project
we can't remove any commit from middle :: each commit has a hash ID & each commit is build on top of each other. If you want to go to a particular commit & remove all the commits made after it then copy its hash ID so that all the commits made after that particular commit got unstaged

Using command :↓

git reset f4954ce65f004560ff74be82257 63e31b41a9869

[Hash ID]

All the commits made after this will be removed from log history.

Q → what happened to all the files that were modified or changed : they are now in the unstaged area.

→ Stashing Changes ↓

Let's say you're working on a project on a particular feature & you want to try out something new with a clean code base, All the changes you've made you want to remove those but you also don't want to save your progress as a separate commit.

Eg → git add .
git status
touch names.txt
vi names.txt → (add some data)
touch houses.txt
git add .

→ A bunch of things have been done above, now if I don't want to commit it also don't want to loose these changes, I just want to keep these changes backstage & call whenever I need them back we can do it : git stash

git status
git log

→ Popping Stash ↓

To get back all the changes we kept backstage or unstagged area : git stash pop

→ To completly remove the changes that were kept in unstagged area ↓

git add .
git stash
git stash clear : Changes that were made that are not commited & are in unstagged area, are now gone, & we can't get those back.

→ Connecting Remote Repository to Local Repository ↓

⇒ git remote add origin http://github.com/priyankasolanki18/cs.git

⇒ git remote -v : shows all the URL's that are attached to the folder

⇒ git push origin master :

NOTE: I was not able to push files in remote repository because of the privacy settings of it so I have to remove all the commits in which PAT had used

→ Reset your repo. to fresh state, removing all previous commits that included token, using hard reset & a force push :-

STEP 1: Create a fresh commit

```
# create a temporary branch to work safely
  git checkout --orphan new_branch
# add all your files
  git add .
# commit them fresh (no token)
  git commit -m "Fresh start"
```

STEP 2: Delete old history

```
# Delete the old branch
  git branch -D master
# Rename the new branch to master
  git branch -m master
```

STEP 3: Force-Push the clean branch

```
git push -f origin master
```

NOTE: Common Git Situation: your remote (origin) already has some commits that your local (master) branch doesn't have, then Git will protect you from overwriting them.

Q. What's happening ?

- Your remote master branch has commits (maybe a README.md or .gitignore) from when repo was created on GitHub.
- Your local master branch doesn't have those commits

- Git refuses to push because it would overwrite the remote history.

Soln: [OPTION-1]

```
# Merge Remote into local
  git pull origin master --allow-unrelated-histories
# Then push
  git push -u origin master
```

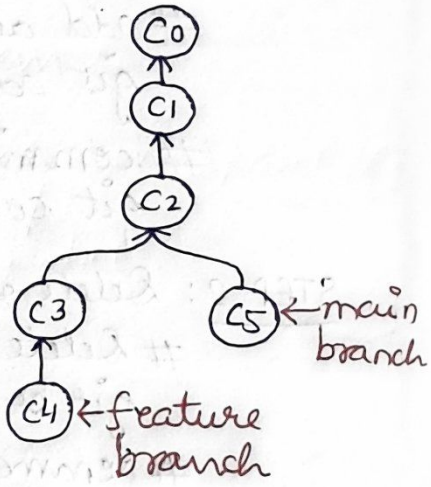
[This flag is needed if the local & remote histories don't share a common base]

[OPTION-2]

```
# Force Push
  git push -u origin master --force
```

→ Branches in Git ↴

```
git branch feature
git checkout feature
# Merging branch to main
git merge feature
```




→ Working with existing Projects ↴

Why Fork & How to Fork?

→ Cloning the forked project to local

```
git clone https://github.com/Priyanka18/commclass.git
cd commclass/
ls
cat README.md
```

→ From where you've forked this project i.e, known as Upstream url by convention, Adding it to local ↴

```
git remote add upstream https://github.com/commclass/commclass.git
```

→ What is a Pull Request?

```
git branch priyanka
git checkout priyanka
git branch ⟶ [ main
               *priyanka ]
git add .
git commit -m "changes made"
git log
git push origin priyanka .
```

→ Removing a commit from the pull request by force pushing to it.

```
git reset a4e3d4456f4320abc9a0
git status
git add .
git stash
git log
```

⇒ `git push origin priyanka -f`

→ Merging a Pull Request

→ Making forked project even with main project ↴

```
git checkout main
git status
#STEP-1 [fetched all the changes]
git fetch --all --prune
#STEP-2 [Reset the main branch of my origin
  to the main branch of upstream
git reset --hard upstream/main

#STEP-3
git push origin main
```

OR.

⇒ `git pull upstream main`

→ Squashing commits ↴

If you've a lot of commits that you're working on & if you want to merge them into one commit.

NOTE:

Never commit on main branch ∵ imagine you are working on 10 projects of 10 features & for every feature you'de created only 1 Pull request then it'll be difficult to review the code i.e; for every bug/issue create a diff. Pull Request i.e; 1 branch == 1 Pull Request

git branch temp
git checkout temp

NOTE: Whenever you create a new branch, it'll get created from head so make sure whenever you create a new branch, your main branch is up to date.

touch 1
git add .
git commit -m "1"
} → add 3 more files
git log

git rebase -i de346ac4543096abca43

[Last commit id in which we want to squash all the commit that has been made after this]

→ Merge conflicts & how to resolve them?

A Merge conflict occurs in Git when:

- Two branches have made changes to the same part of file
- Git doesn't know which version to keep.

Eg → Let's say we have a file: greeting.txt

Content in main branch
Hello from main branch!

Content in feature branch
Hello from feature branch!

Now you try to merge feature into main
git checkout main
git merge feature

→ Git will detect a conflict : both branches changed same line.

[RESULT]: Git will stop merge & mark conflicting area in file like ]

<<<< HEAD
Hello from main branch!
====
Hello from feature branch!
>>>> feature

→ How to Resolve the conflict ↴

Edit manually

→ Choose one version or combine both
→ Save file after editing
→ Mark conflict as resolved

git add greeting.txt

→ Complete the merge

git commit -m "Resolved merge conflict"