#GIT(Version Control System)

#Difference between git and github.

- **Git**: Think of it as a tool you use on your computer to keep track of changes in your code. Git is like a personal journal or diary where you write down your thoughts and ideas
- **GitHub**: Picture it as a website where you can store your code (using Git) and work together with others on it. It's like a social network for coding. GitHub is like a library where you can store your journal (Git repository) for others to see and collaborate on.

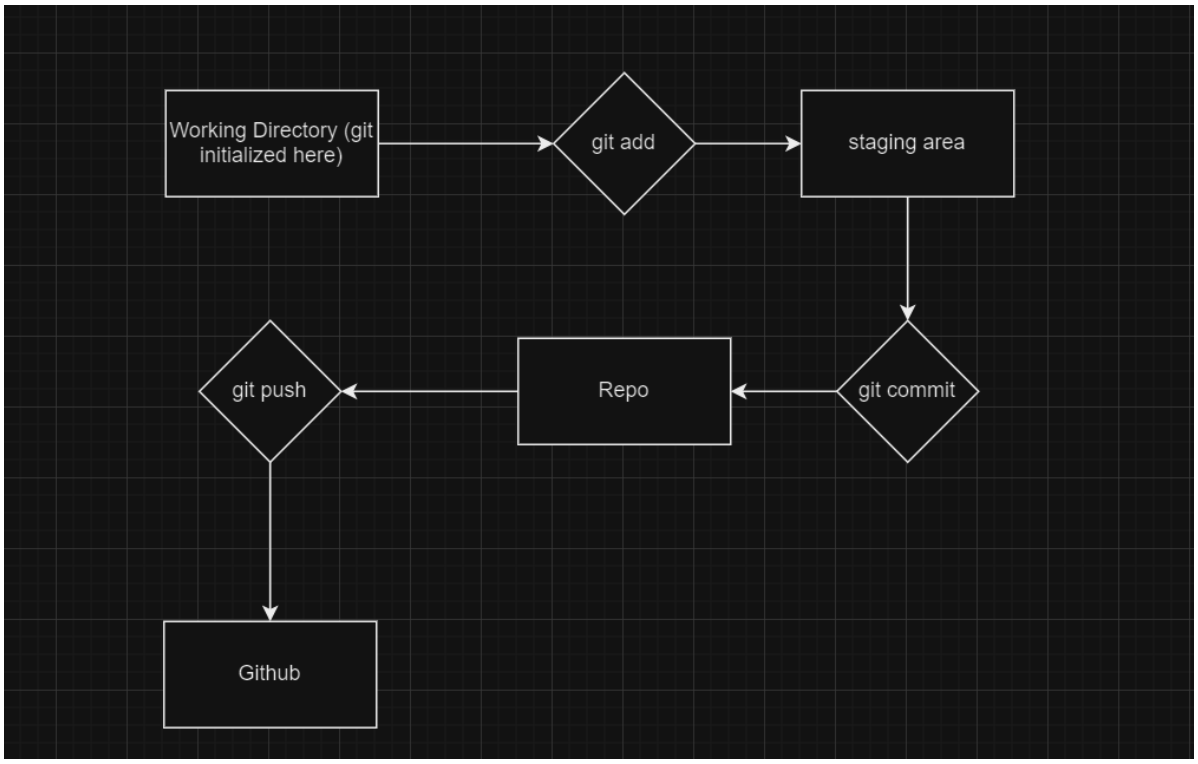
#What is Repo?

In Git, a "repo" is short for repository. Think of it as a folder or a directory that contains all the files and the complete history of changes for a project.

#Getting Started With Git

1. `git –version` → To get the installed version of git on your machine.
2. `git status` → To get the status of Your tracked and untracked files and Folders.
3. `git-init` → When you run `git init` in a folder, Git creates a new repository, or "repo," in that directory. This sets up all the necessary files and directories that Git needs to start tracking changes in your project. `.git` **directory** is where Git stores all the information about your project's history, branches, and configurations.

4. `git add file_name` → To add your files into the staging area (intermediate zone before final commit).
5. `dir /a` → To see hidden folders.
6. `mkdir  folder_name` → To create a Directory.
7. `cd path_name` → To move to directories.
8. `dir` → to see various files and folders in current directory.
9. `git commit -m “message of commit”` → Just like checkpoint in a game which is used to save current changes or process of your files. Commit messages should be done in Imperative manner like “add footer to main file”.
10. `git log` or `git log -oneline` :- gives you details about your commits , author , date etc...



#Configuration Files

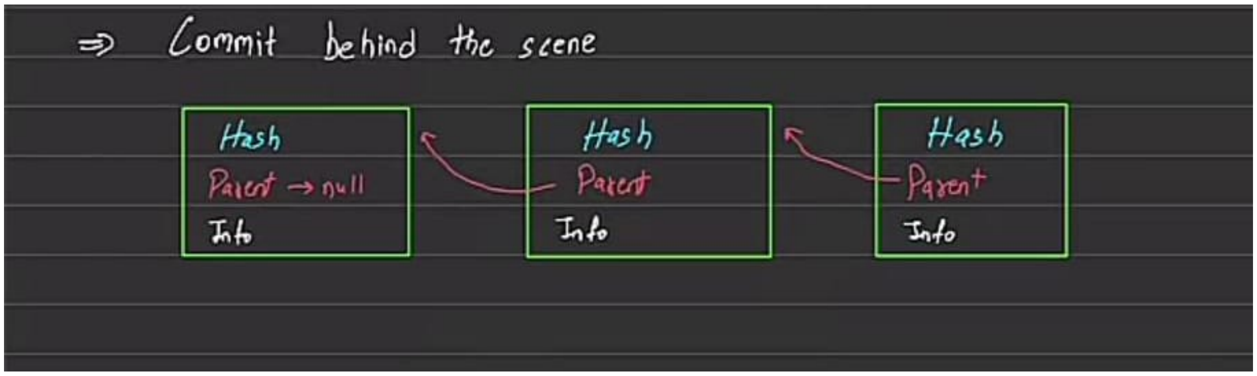
→ `git config –global user.name “...”`

→ `git config –global user. email "..."`

→ `git config --global core.editor "code --wait"` → To set up vs-code default Code editor.

→ `.gitignore` file use to hide important files such as .env files containing api key etc.. just go inside the .gitignore file and mention those files you don't want to keep track of. You can use gitignore generator on google to get info about files to be ignored.

→ `cat .gitconfig` → To get full details about name , email ,sign-in key go to the home diretory such as C in windows and run this command.



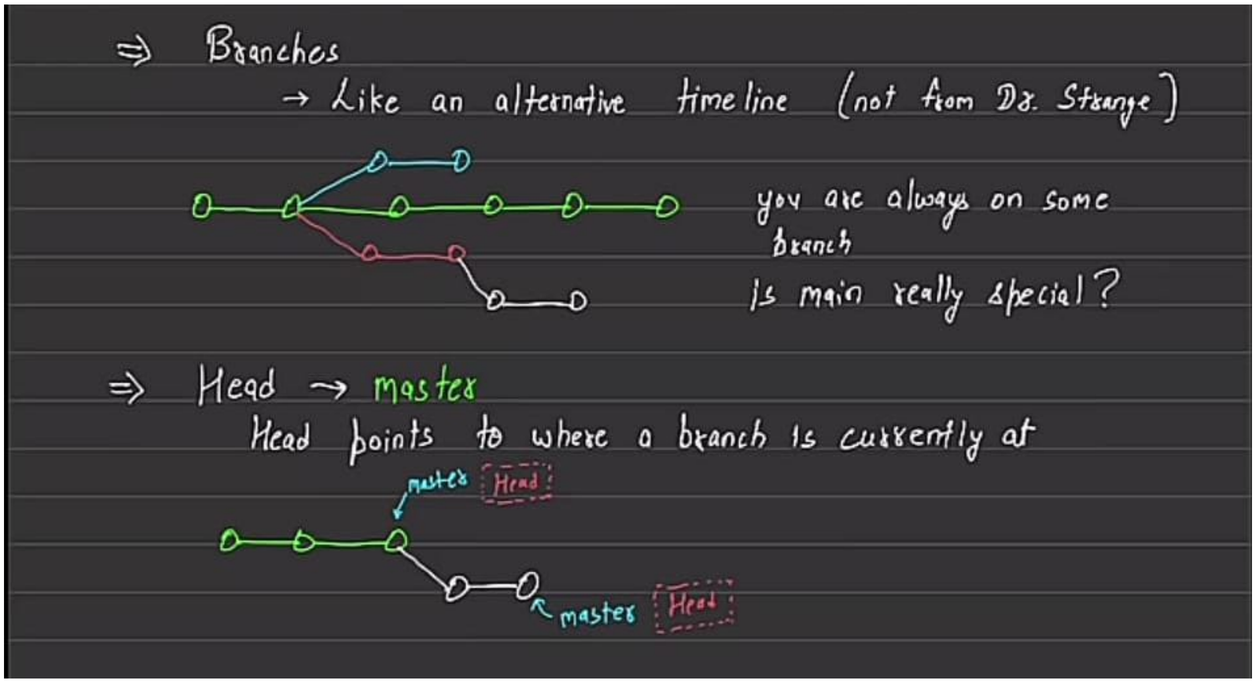
#Branches

Branches are essentially pointers to a specific commit in the repository's history. They allow you to diverge from the main line of development and work on different features, fixes, or experiments without affecting the main codebase until you're ready to merge your change.

→ `git branch` → will tell you different branches that exists and current pointer is pointing to which branch .

→ `git branch branch_name` → To create a new branch.

→ `git checkout branch_name` → To move or switch to other branch.



⇒ Branches

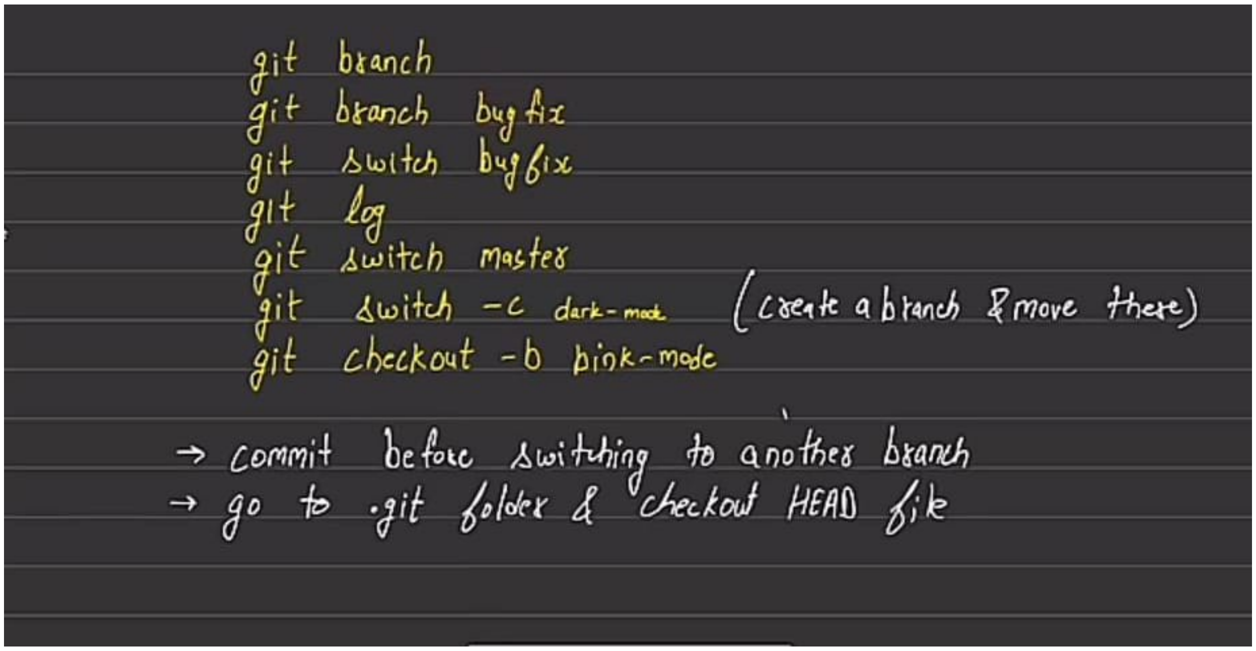
→ Like an alternative time line (not from Dr. Strange)

you are always on some branch

Is main really special?

⇒ Head → master

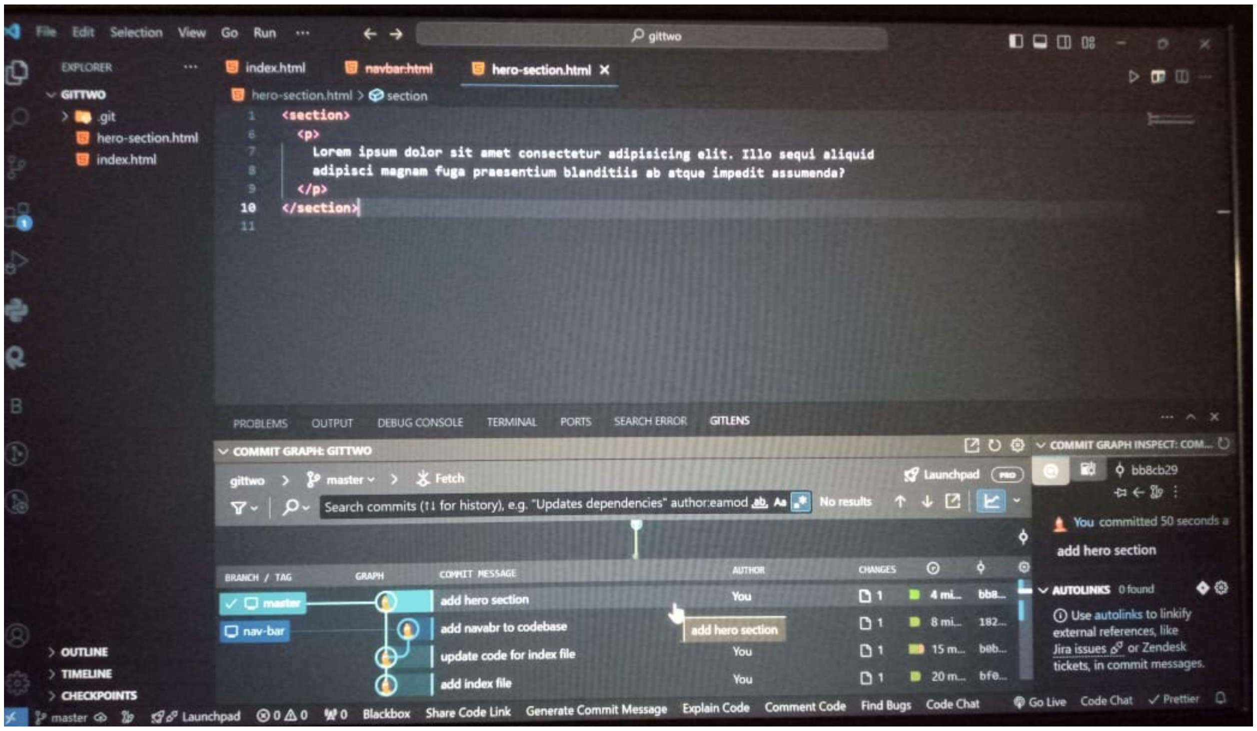
Head points to where a branch is currently at



```
git branch
git branch bugfix
git switch bugfix
git log
git switch master
git switch -c dark-mode      (create a branch & move there)
git checkout -b pink-mode
```

→ commit before switching to another branch

→ go to .git folder & checkout HEAD file



```
<section>
  <p>
    Lorem ipsum dolor sit amet consectetur adipisicing elit. Illo sequi aliquid
    adipisci magnam fuga praesentium blanditiis ab atque impedit assumenda?
  </p>
</section>
```

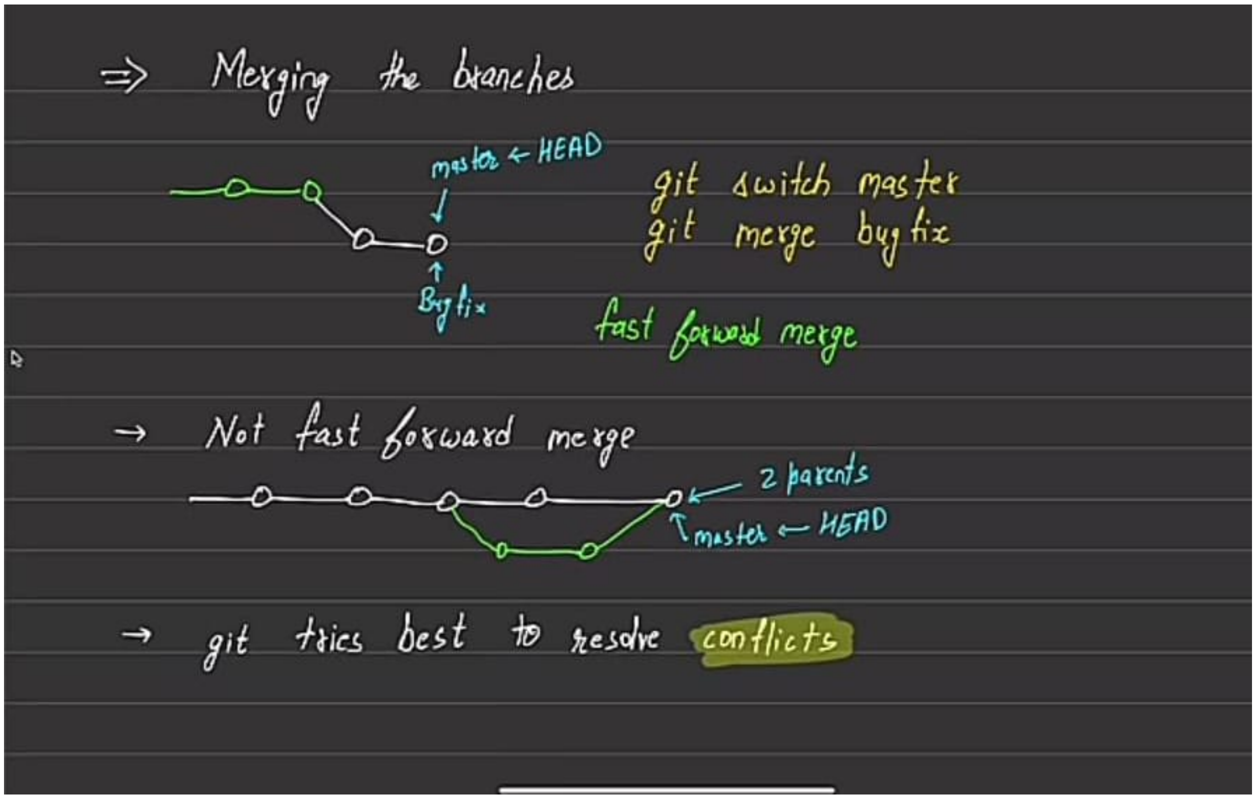
BRANCH / TAG	COMMIT MESSAGE	AUTHOR	CHANGES		
✓ master	add hero section	You	1	4 mi...	bb8...
nav-bar	add navabr to codebase		1	8 mi...	182...
	update code for index file	You	1	15 m...	beb...
	add index file	You	1	20 m...	bfe...

#Merging Branches

1. Fast Forward Merging → The main branch doesn't do much , you work entirely on separate branch then you merge that branch into the main.
2. Not fast Forward Merge → Main branches and other branches are working and eventually at a particular time you want the perform merging. in order to perform merging eg nav-bar branch to master branch you need to be on master branch.

→ `git merge branch_name` → To MERGE x branch with master branch.

→ `git branch -d branch_name` → To delete a branch



⇒ Merging the branches

```
git switch master
git merge bug fix
```

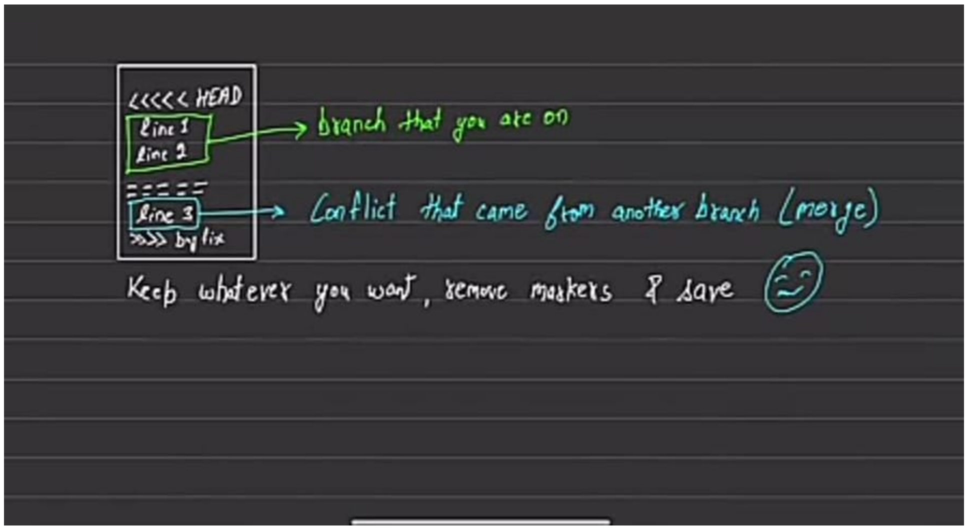
fast forward merge

→ Not fast forward merge

→ git tries best to resolve conflicts

Conflicting condition :- If a particular guy is working on different branches or not on separate branch.

“it means suppose you have a master branch containing index.html file and while on this master branch you edited index.html with some code now you switch on to the footer branch and again inside index.html you added some code or again edited index.html now when you again switch to master branch you will see different index.html file . Now suppose you try to merge master with footer branch you will encounter the error of Automatic merge failed because now you have to decide whether you need to keep the code of master branch or footer branch inside index.html and that is the whole idea about conflict”



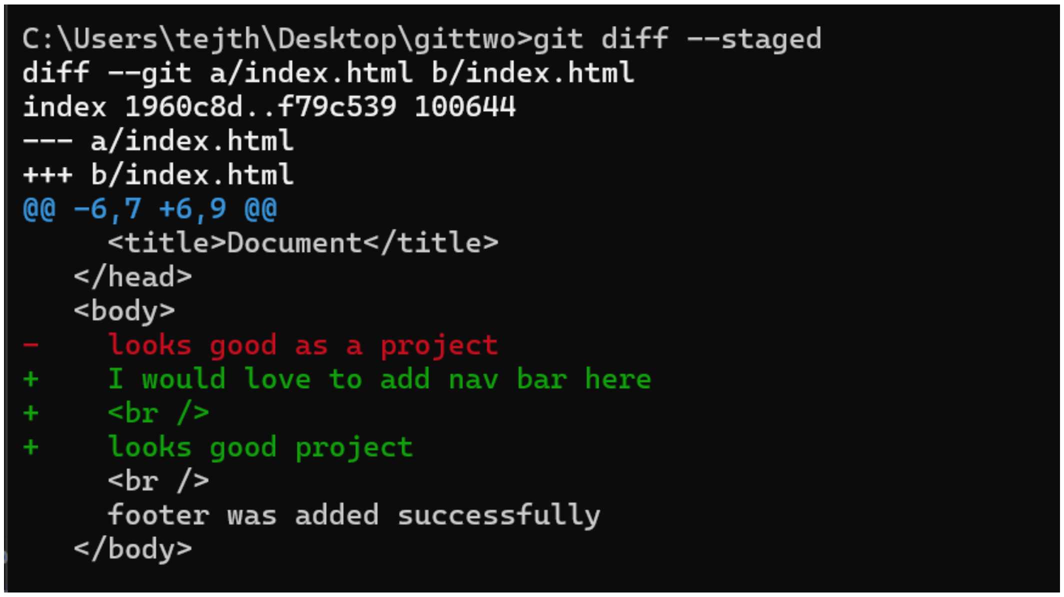
<<<<< HEAD
line 1
line 2
=====
line 3
>>>>> bugfix
branch that you are on
Conflict that came from another branch (merge)
Keep whatever you want, remove markers & save

#Git-Diff

`git diff` command shows the differences between changes you've made in your files since the last time you saved those changes to your Git repository. It helps you see what's been added, removed, or modified.

→
`git diff –staged`

"—- indicates file before stagging and +++ indicated final file after staging "


```
C:\Users\tejth\Desktop\gittwo>git diff --staged
diff --git a/index.html b/index.html
index 1960c8d..f79c539 100644
--- a/index.html
+++ b/index.html
@@ -6,7 +6,9 @@
     <title>Document</title>
   </head>
   <body>
-    looks good as a project
+    I would love to add nav bar here
+    <br />
+    looks good project
     <br />
     footer was added successfully
   </body>
```

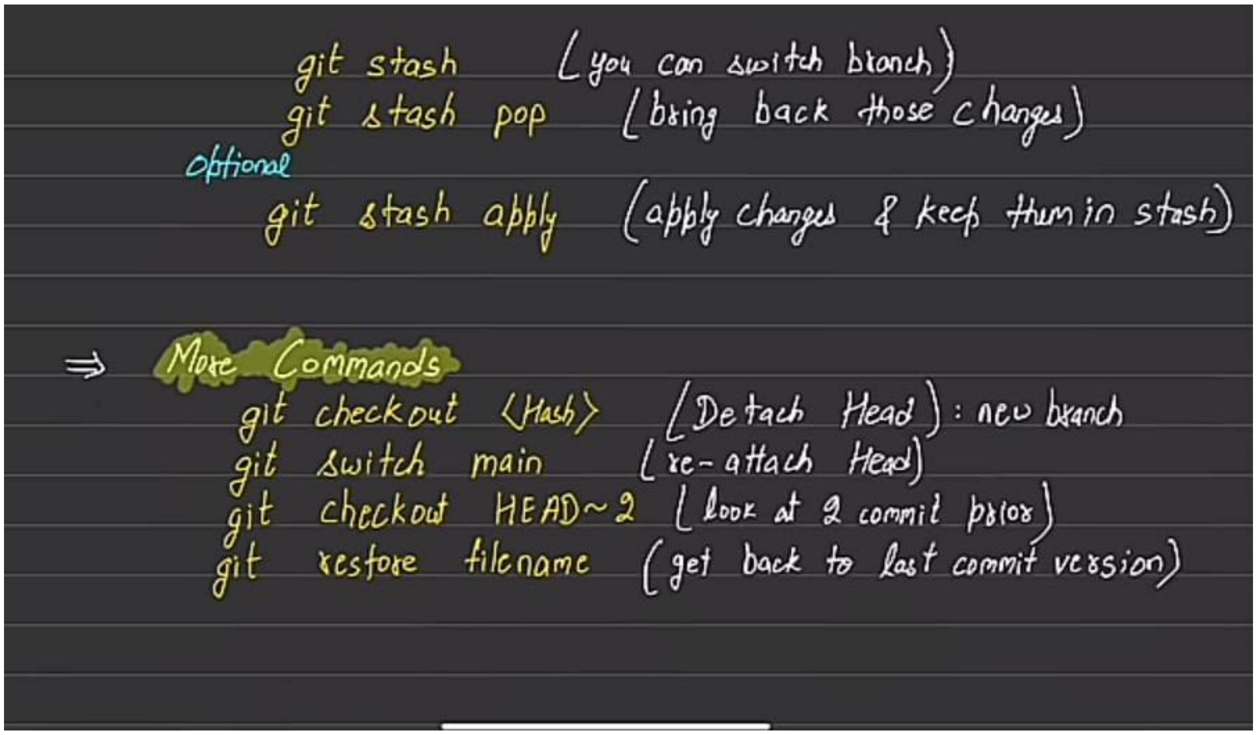
→ `git diff de2bcf3 3d0e639` (using diff by commit id’s)

#Git Stashing

The
`git stash` command helps you temporarily save changes you've made to your files, allowing you to switch to another branch or deal with something else without committing those changes. It's like putting your changes aside for later without making a permanent commit.

`git stash pop` command applies the changes you previously stashed (put aside) back onto your current working branch(any branch). It's like taking the changes you saved and putting them back where they were before you stashed them.

`git checkout master` moves you back to the main branch



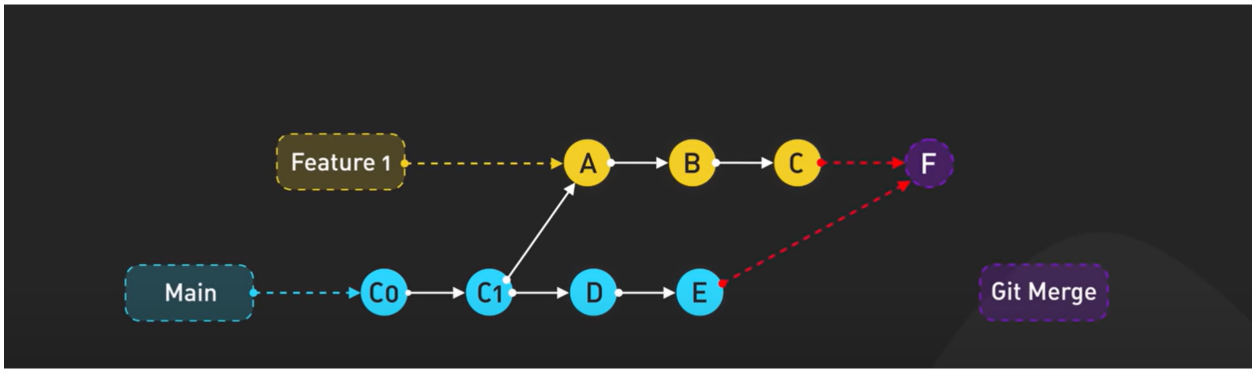
```
git stash            (you can switch branch)
git stash pop        (bring back those changes)
optional
git stash apply      (apply changes & keep them in stash)

⇒ More Commands
git checkout <Hash>      (Detach Head) : new branch
git switch main          (re-attach Head)
git checkout HEAD~2      (look at 2 commit prior)
git restore filename     (get back to last commit version)
```

#Git rebase

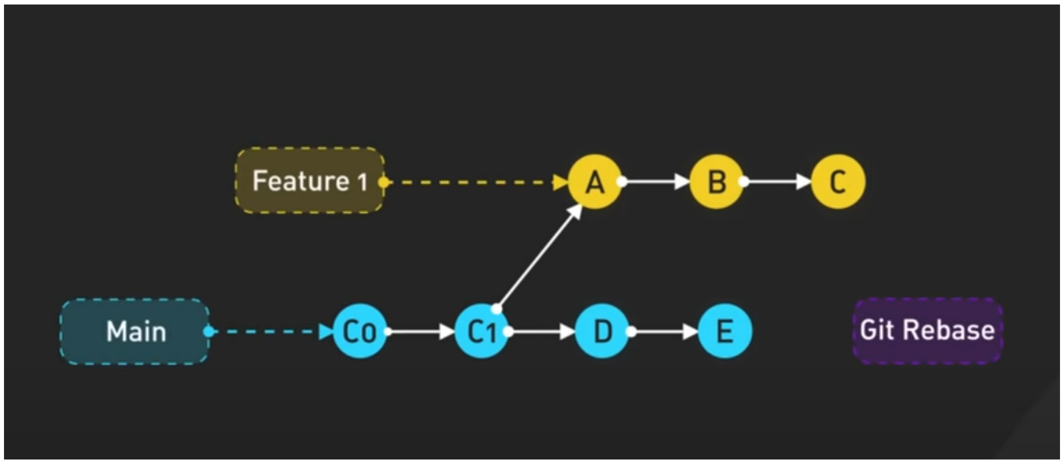
Git rebase is used to integrate changes from one branch into another by moving or combining a sequence of commits onto a new base commit. This can help maintain a cleaner and more linear project history compared to merging.

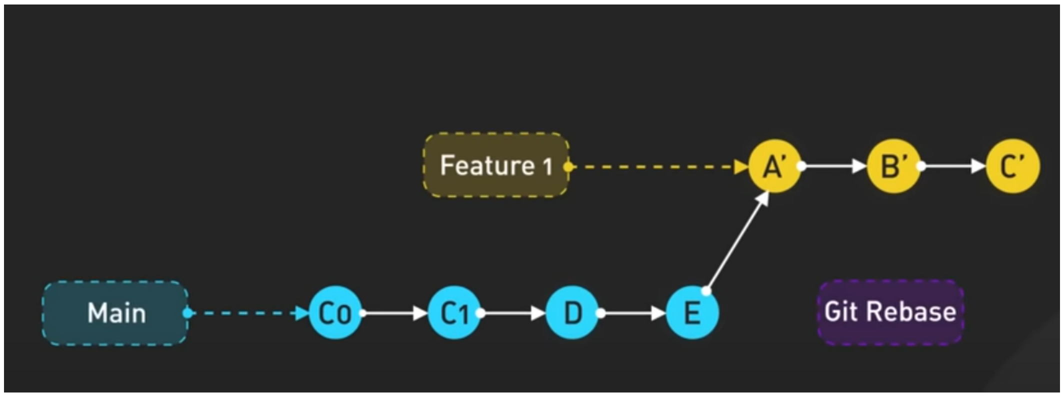
“Merges all the branch into the main branch(to remove unnecessary commits”

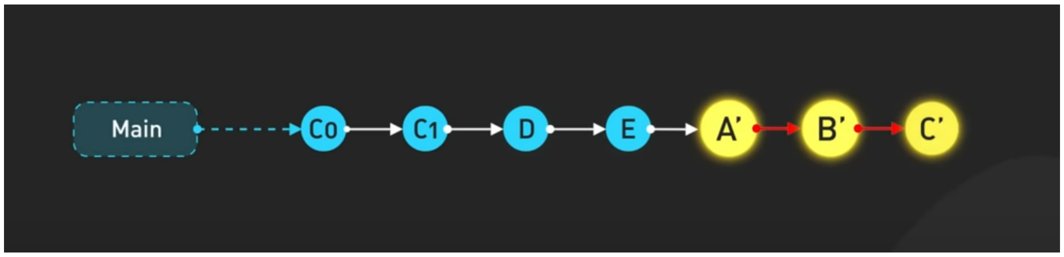
Git merge pulls in the latest changes from main into the feature branch creating a new merge commit in the process its like joining or tying two branches with a not. This can result in too many nodes and makes graph clumsy.



Git rebase changes the base of feature branch to the latest commit on main branch.Resulting clean git graph.





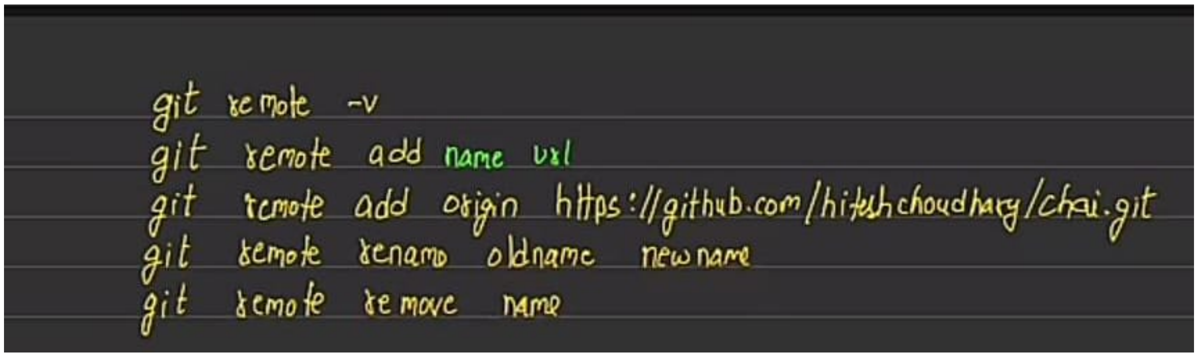


#Communication with Your Github Account

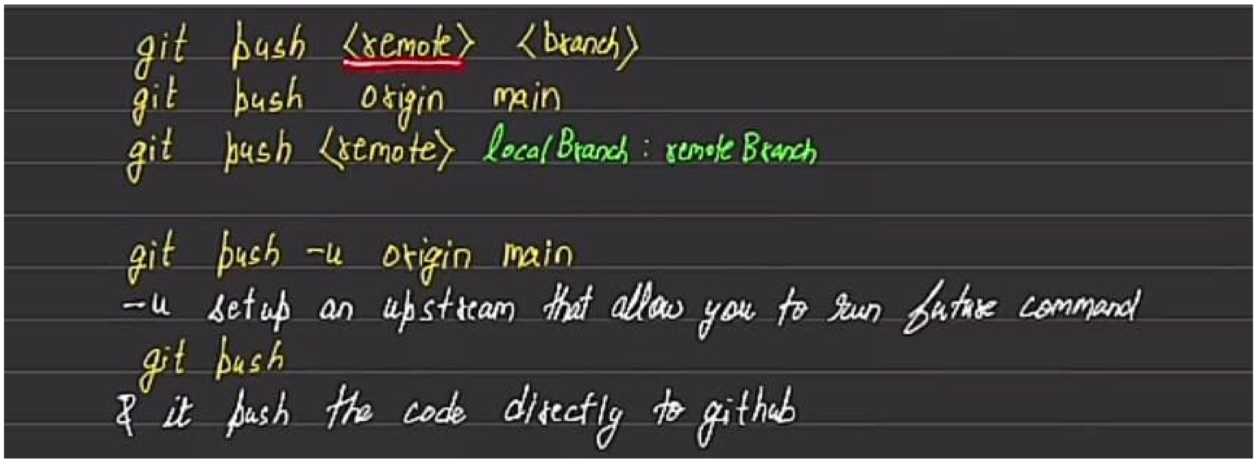
"Git is a software and github is a service to host git online"

1. Create Github Account using your email
2. generate Ssh key by following this documentation :- https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
3. Go to setting section in your github and paste the ssh key.
4. Create your project folder and add files inside it.
5. check `git status`
6. initialize git using `git init`
7. Add and commit necessary Files.
8. to check branch type git branch
9. create a new repository on your github.
10. You will get various commands after creating your repository such as renaming your branch etc..
11. `git remote -v` When you run `git remote -v`, Git shows you the names and web addresses of the remote places where your code is stored. It helps you keep track of where your code is coming from and where it's going when you share or collaborate with others
12. .The `git push -u origin main` command is used in Git to push your local changes to a remote repository, specifically to the branch named "main" on the remote repository called "origin". Here's a breakdown of the command:

- `git push`: This is the Git command used to upload local repository content to a remote repository.
- `u`: This flag sets the upstream branch for the current local branch. After this command, if you simply type `git push`, Git will know which remote branch to push changes to.
- `origin`: This is the name of the remote repository. By convention, "origin" is often used as the default name for the remote repository you cloned from.
- `main`: This is the name of the branch you are pushing to on the remote repository. In Git, "main" is often used instead of "master" as the default branch name.



```
git remote -v
git remote add name url
git remote add origin https://github.com/hiteshchoudhary/chai.git
git remote rename oldname newname
git remote remove name
```

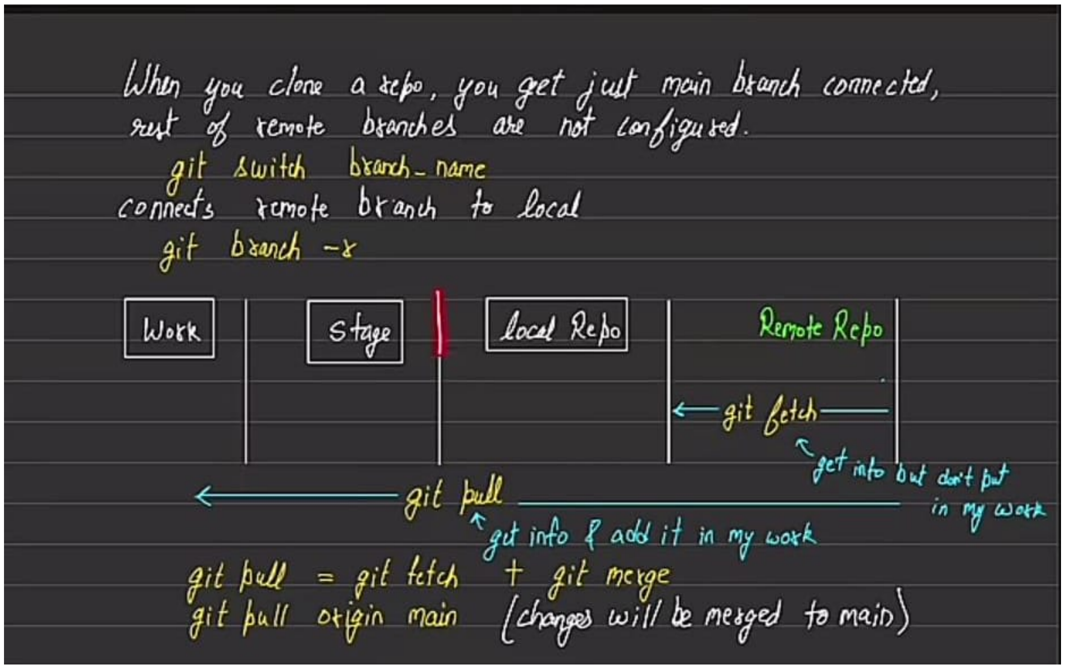


```
git push <remote> <branch>
git push origin main
git push <remote> localBranch : remote Branch

git push -u origin main
-u setup an upstream that allow you to run future command
git push
& it push the code directly to github
```

#OPEN SOURCE

Open source is a philosophy that software can be or should be distributed for free so that other programmers can save some time. open source project is a software project whose source code is made available to the public under a license that allows anyone to view, use, modify, and distribute the code. The concept of open source promotes collaboration, transparency, and community-driven development.

#Clone (Bringing code of other repo to your local storage)



When you clone a repo, you get just main branch connected, rest of remote branches are not configured.
git switch branch_name
connects remote branch to local
git branch -r

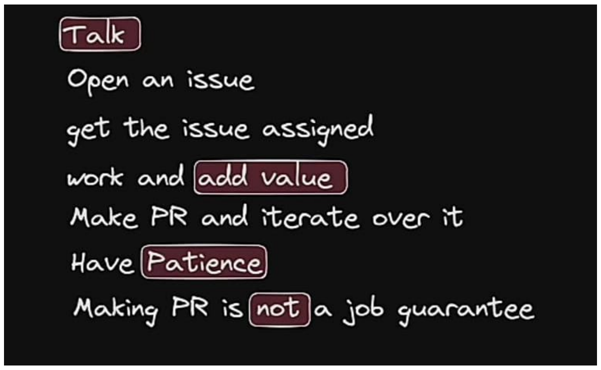
Work
Stage
local Repo
Remote Repo
git fetch
get info but don't put in my work
git pull
get info & add it in my work

git pull = git fetch + git merge
git pull origin main (changes will be merged to main)

→ `git clone <url>` command is used to create a copy of a repository, including all of its files, branches, and commit history. You need to replace "url" with the URL of the repository you want to clone.

→ `git fetch` The `git fetch` command is used to retrieve the latest changes from a remote repository without merging them into your local branches. It's a way to keep your local repository up-to-date with changes made in the remote repository without automatically integrating those changes into your current working branch.

→ `git pull` The `git pull` command is used to fetch and merge changes from a remote repository into your current branch. Essentially, it's a combination of two separate Git commands: `git fetch` and `git merge`.


Talk
Open an issue
get the issue assigned
work and add value
Make PR and iterate over it
Have Patience
Making PR is not a job guarantee

A pull request (often abbreviated as "PR") is a mechanism used in distributed version control systems, such as Git, to propose changes to a codebase hosted in a repository. It's essentially a request to merge one branch into another, typically from a feature branch into a main branch (like master or main). Pull requests are commonly used in open source projects but are also widely adopted in team-based development workflows.

GitHub

GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

`git config --global user.name “[firstname lastname]”`
set a name that is identifiable for credit when review version history
`git config --global user.email “[valid-email]”`
set an email address that will be associated with each history marker
`git config --global color.ui auto`
set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`
initialize an existing directory as a Git repository
`git clone [url]`
retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`
show modified files in working directory, staged for your next commit
`git add [file]`
add a file as it looks now to your next commit (stage)
`git reset [file]`
unstage a file while retaining the changes in working directory
`git diff`
diff of what is changed but not staged
`git diff --staged`
diff of what is staged but not yet committed
`git commit -m “[descriptive message]”`
commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`
list your branches. a * will appear next to the currently active branch
`git branch [branch-name]`
create a new branch at the current commit
`git checkout`
switch to another branch and check it out into your working directory
`git merge [branch]`
merge the specified branch’s history into the current one
`git log`
show all commits in the current branch’s history

INSPECT & COMPARE

Examining logs, diffs and object information

`git log`
show the commit history for the currently active branch
`git log branchB..branchA`
show the commits on branchA that are not on branchB
`git log --follow [file]`
show the commits that changed file, even across renames
`git diff branchB...branchA`
show the diff of what is in branchA that is not in branchB
`git show [SHA]`
show any object in Git in human-readable format

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`
delete the file from project and stage the removal for commit
`git mv [existing-path] [new-path]`
change an existing file path and stage the move
`git log --stat -M`
show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

`git config --global core.excludesfile [file]`
system wide ignore pattern for all local repositories

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

`git remote add [alias] [url]`
add a git URL as an alias
`git fetch [alias]`
fetch down all the branches from that Git remote
`git merge [alias]/[branch]`
merge a remote branch into your current branch to bring it up to date
`git push [alias] [branch]`
Transmit local branch commits to the remote repository branch
`git pull`
fetch and merge any commits from the tracking remote branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`
apply any commits of current branch ahead of specified one
`git reset --hard [commit]`
clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`
Save modified and staged changes
`git stash list`
list stack-order of stashed file changes
`git stash pop`
write working from top of stash stack
`git stash drop`
discard the changes from top of stash stack