1 Basic Commands

- `git clone <url>` → Bring a repository that is hosted somewhere like Github into a folder on your local machine
- `git add <file>` → Track your files and changes in Git and stage the changes in git
- `git commit -m "<messege>" <file>` → Save your files in Git. You can commit like this

  `git commit -m "messege" <file> -m "<description>"`

  which also gives some description
- `git push origin <branch-name>` → Upload Git commits to a remote repo, like Github
- `git pull` → Download changes from remote repo to your local machine, the opposite of push
- `git status` → Shows all the files that are created and commited and haven't been commited

2 Initialize a Repo

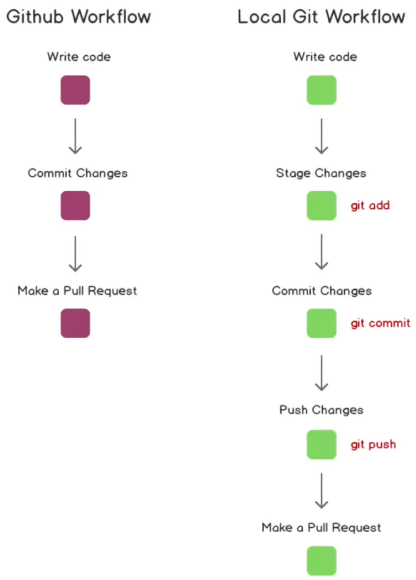
- Create a folder let `repo`
- Create a `README.md` file
- Run the command `git init`
- Then add the file by

  `git add README.md`

  `git commit -m "Created readme"`

  The command `git commit -am "<messege>"` adds and commits but it only works for modified files not newly created files
- Now we can not just `git push origin master` since git does not know where to push this since there is no connection. We have to create this connection
- Go to github and create an empty repository. Copy the given link
- Then run `git remote add origin <copied-url>`

- Then run `git push origin master`
- Later we will use only `git push` but we will use something called upstream meaning this is where i want to push by default by `git push -u origin master`

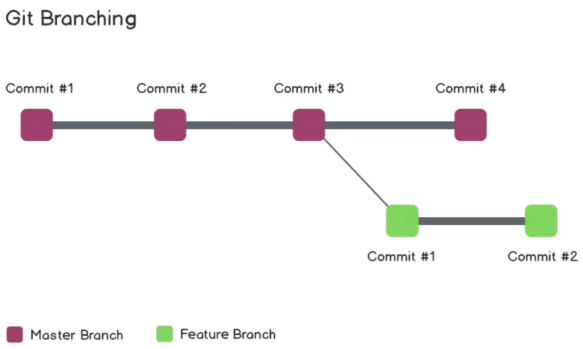


3 Git Branching

By defautl it is the *Master* branch now *Main* branch. We can also create other branches. The more branches you have it more an more looks like a tree.

- Suppose we create another branch *Feature Branch*
- At first the code on both branches will be exactly same.
- As you make changes in *Feature Branch* those changes are only visible in *Feature Branch*. Not visible if you swith to *Master Branch*. Same for *Master Branch* changes
- Then if you want you can merge it back to Master branch

- If you run `git branch` it shows and the branch we are currently in is marked by a *
- With `git checkout` you switch between branches. Withour commiting the changes or stash you can not change branch otherwise the changes will be overwritten



- To create new branch you run

  `git checkout -b <branch-name>`

  and it then switches to that newly created branches
- `git diff` shows all the changes that havent been committed since last commit. But `git diff <branch>` shows all the difference between current branch and the specified branch
- Now you can merge the specified branch with the current branch by `git merge <branch>`
- But more common practice is pushing the changes upto github and make a *Pull Request*
- While pushing if we are on a different branch in the git push command the branch name will be changed.

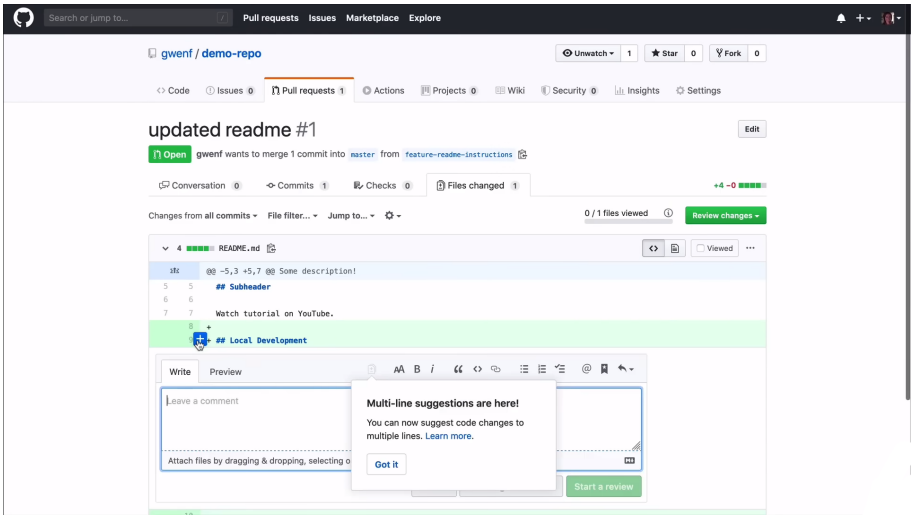
For `git merge` if you are on branch A and you have another branch B then `git branch B` makes the merge direction $A \leftarrow B$

4 Pull Request (PR)

- So will make a PR from the Feature branch to Master branch.
- Once you do a PR ayone can comment on that PR.
- After making a PR you can also update the code and making commits and pushing to the github as long as its on the same branch that you are making th PR with.
- Once the PR is merged you generally delete the Feature branch and switch back to the master branch and if you want to make additional coding changes then create new branch adn start the process again

- You can merge the PR in github page
- Now locally we dont have the changes.
- We need to pull them down to local. To get the changes in our local Master branch from origin we will do `git pull origin master`
- We can always remove the branch name from `git pull origin master` by setting the upstream
- We still have the branch even after the PR merge. So we will delete the Feature Branch by `git branch -d <branch>`

You can also comment on a specific line change in a PR by hittiing the '+' button In real life



it is not easy merging. There is something called **Merge Conflict**. You building your own code on your own branch. Other people are writing their own code in their own branches. And master is getting updated from multiple different places. SO its possible for multiple people to change the same file. So git doesn't know which code is to keep which code is redundant which code you want to get rid of. You have to manually do that.

To fix merge conflict github gives an interface to fix them. Also you can change with terminal. Then you have to commit the changes then `git merge` .

5 Undoing

- After staging a file if you want to undo then do `git reset [<file-name>]` Then the file is no longer staged
- After commiting if you want to undo the commit do

`git reset HEAD~1`

Here the `HEAD` is a pointer to the last commit. So i am telling it to do something with the last commit. Then `~1` tells git to instead of pointing to the last commit go one commit further that will completely undo the last commit

- Instead of last commit if you want to go some specific commit you can get the hash of that commit from `git log` and then use instad of `HEAD`. But the changes will be there but not saved in git
- To grt rid of all changes from some commit then copy the hash of that commit then do `git reset --hard <commit-hash>`

6 Forking

Another person's repo you can make a copy for yourself and your updates by *Forking*. After making all your updates you can then do a PR