**GIT**

GIT is a version control tool is used maintain the history of the projects.

Suppose you made changes to you code and the application breaks because of the new feature and you wanted to go back in time and access previous code where that application is working fine. GIT will help you achieve that.

And 100 developers are working on the same projects GIT will be used to maintain the project with GIT we can see at what time which developer made what change to the code base and also with GIT we can set approvers so that only when the code is approved by reviewers the code can be merged into main code base.

GIT is a service provider like email just like email has different applications like Gmail and Outlook GIT also has different applications to provide version control service like GitHub, GitLab

**GIT commands:**

|  |  |
| --- | --- |
| Command | Description |
| git init | creates .git folder where all history of current folder is being saved |
| git status | to know what changes are made in our project |
| git add . | Adds all the files that are untracked to the staging area |
| git add filename.txt | adds specific files to staging area |
| git commit -m “add text here” | to commit the staged changes |
| git restore --staged filename.txt | to restore staged files/ to unstage staged files |
| git log | History of all the commits |
| git reset hashcode (eg of hashcode afgdtehvbchdu) | Removes all the commits above the hashcode given after git reset command and the above commits are sent to unstaged state |
| git stash | Helps if we don’t want to commit the changes right now but use at a later point of time |
| git stash pop | All the commits in the stash area will be applied |
| git stash clear | All the changes in the background will be cleared |
| git remote add origin https://websitename | Add our folder to github repo |
| git remote remove origin | Remove the origin website |
| git remote -v | Displays the origin URL |
| git push origin main | Pushes our changes to main branch |
| git pull --rebase origin main | If any ref errors occur |
| Git branch featurebranchname | Creates a new branch |
| Git checkout featurebranchname | Head is pointing featurebranch and any commits will be added to featurebranch |
| git clone http:// website | clones the forked branch to local |
| git remote add upstream http:// website | Add upstream website to which we want to contribute |
| Git pull upstream main | Pulls upstream main branch |
|  |  |

**Some Basic Linux Commands**:

* ls : list all things in current folder
* ls – a : list all the files including hidden files
* touch : create a new file eg: touch file.txt
* cat : open content of file

* Head in GIT: head points to the origin or branch to which out next commits are added. It changes form main branch to feature branch we have created once we create a branch and checkout from feature branch.
* We can only create one pull request per branch any other commit on the same branch will result as commit in the same branch and a pull request cannot be created for that feature.

Suppose we are working on feature f1 we have created branch b1 and committed the feature f1 and created the pull request for it.

Now we are working on feature f2 if we commit the f2 on branch b1 the discussions for feature f1 and f2 has to happen on the same pull request

If that happens ambiguity arises between f1 and f2 since the discussing 2 features on a single pull request is not advisable

So now we have added an extra commit as f2 to delete the commit in f2 we use git reset hashcode and the commit f2 will be unstaged now we will stage this using git add. and using git stash we will stash the commit. So that the local forked have f1 commit but upstream commits have both f1 and f2 commits so we have to use git push origin feature branch -f which means we have to force our f1 commit as f2 is stashed.

* git clone <https://github.com/prudhvi1836/git-basics-1.git> clones the forked branch to local
* To set upstream we can do git remote add upstream http:// website to which we want to contribute
* After pull request approval the changes in the project which we are going to contribute will not be able to reflect in our forked repository

Consider a scenario where you are working on a feature and the main branch of the project to which we are contributing got updated in order to get those updated changes to our forked repo we need type below command

First change from featurebranch to main branch using

Git checkout main

Then pull upstream main using

Git pull upstream main

Now push these changes to origin main using

Git push origin main

* **git rebase** is used to merge commits there are 2 ways you can do it

1st way: use git reset hash number then all the commits above the hashcode will be unstaged then we can stage them using command **git add .** and commit all of them at once

2nd way : using command **git rebase -i hashcode** it will enter interactive mode

In that we get something like below

Pick hdhdd commit message

Pick fhfhfh commit message

Pick dbcgfv commit message

Pick fbvghr commit message

If we want to merge all the commits we can do

Pick hdhdd commit message

S fhfhfh commit message

S dbcgfv commit message

S fbvghr commit message

All the squash(S) here will be added to the pick above them

Suppose we want to add middle 2 to the above commit we can do it in below way

Pick hdhdd commit message

S fhfhfh commit message

S dbcgfv commit message

pick fbvghr commit message

here the s – squash will be added to the pick above i.e, fhfhfh , dbcgfv commits will be added to hdhdd commit

use :x to escape interactive mode

**merge conflict :**

when two persons try to change the same line git will get confused it will ask us to resolve the conflict my manually selecting what changes we want

--------------------------------------------------------------------------------------------------------------------------

**Note: Use *My Account* → *Settings* → *Developer settings* → *Personal access tokens* → *Generate new token*.**

**Command:**

**git remote set-url origin** [**https://<token>@github.com/<username>/<repo**](https://%3ctoken%3e@github.com/%3cusername%3e/%3crepo)**>**