Git is a version control system

Working directory vs Git repository:

Git repositiory is .git folder which contain stash file and commit object folder

Working directory is the directory which contain all your coding files and Images.

**Git Init :** create Empty git repository

**Git Status:**command **displays the state of the working directory and the staging area**.

**Git Log**: to see Commit history

**Git add filename OR Git add .** use to add files in staging Area.

**Git commit –m “commit mssg”** will commit files from staging area.

Note : On Initial commit only git Store/Save the files, Then it only track Changes it don’t store every snapshot.

**Git ls-files**: will display list of all files in Staging Area.

Undoing Unstage Changes:

**Git restore filename** to Undo/Discard all Changes of specific filename or **git restore .** to undo/Discard all untrack files.

**Git reflog** this command will give Hashcode which help to bring back lost information like (reset commit) and deleted branches.

**Git restore –staged filename** is used to Undo/Discrad all changes of FileName from staging Area.

**Git restore –staged .** is used to Undo/Discrad all changes from staging Area.

***Git reset is Used to remove the latest commit and move the head to specific Commit*.**

**Git reset –soft head~1: this will remove the commit only from git Log, but changes of that particular commit will be present in both working directory and Staging area . ~1 Indicated move back 1 commit from latest.**

**Git reset Head~1 :** this will remove commit from git log as well as Staging Area but changes f that particular commit stay in working Directory.

**Git reset –hard head~2:** this will remove commit from all, i.e from git log, staging area as well as working directory

**Git branch –d brachName:** this will delete branch if this already merge to master branch.

**Git branch –D brachName:** this with Capital D will force to delete branch if merge or not Merge.

**Git clean –df** : this will force to delete the extra files which were added and Untracked.

**To switch Between Multiple Commits use :**

git checkout commitHash: this is used to switch to specific commit. you can use git log to get commithash.

Note: git checkout will remove the leading commits from git log, but in order to get back in leading commit use git checkout branchName.

**Git branch** : this command will display all branches

**Git branch branchname** : this will create **new Branch which also having all commits of master branch.**

**Git checkout branchname / Git Switch branchname will help to switch between multiple branches.**

**Git checkout –b NewbranchName / git switch –c newBranchname will create new branch and checkout to new Branch**

**Git merge branchName: This will merge your current branch which u are present on with the respective BranchName.**

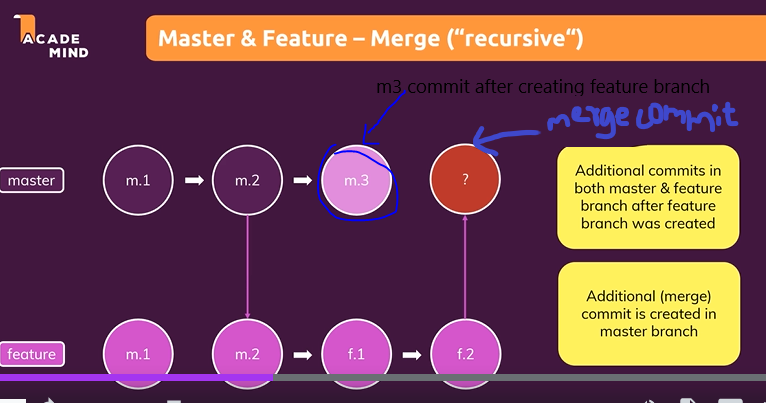
**It will also bring the all commit of respective branchName on the current branch.**

**Two types of git Merge : 1. Fast Forword Merge 2. Recursive merge**

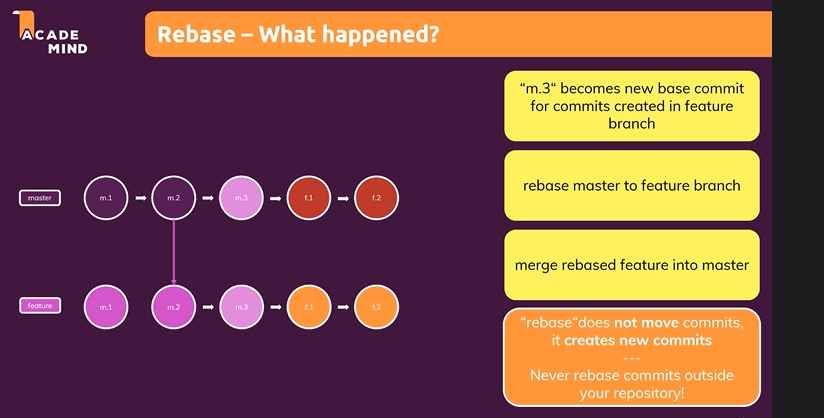
**1. Fast Forward Merge: this will merge for e.g if feature branch created from master branch then it will merge feature with master if no additional commits made on the master branch.**

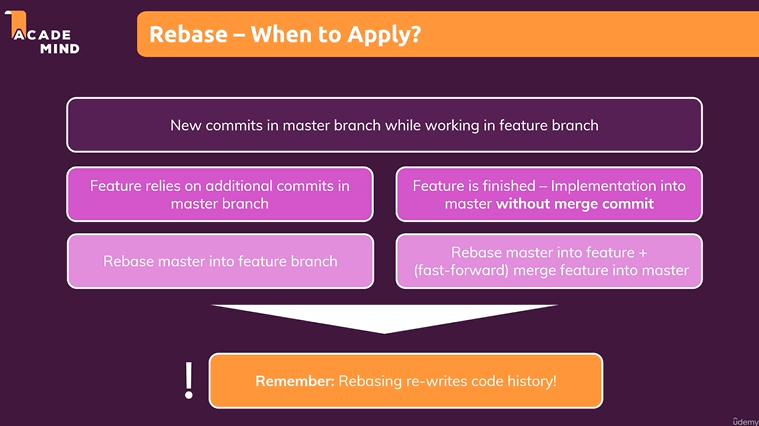
**2. recursive Merge : this will merge for e.g if feature branch created from master branch then it will merge feature with master even after additional commits made on the master branch.**

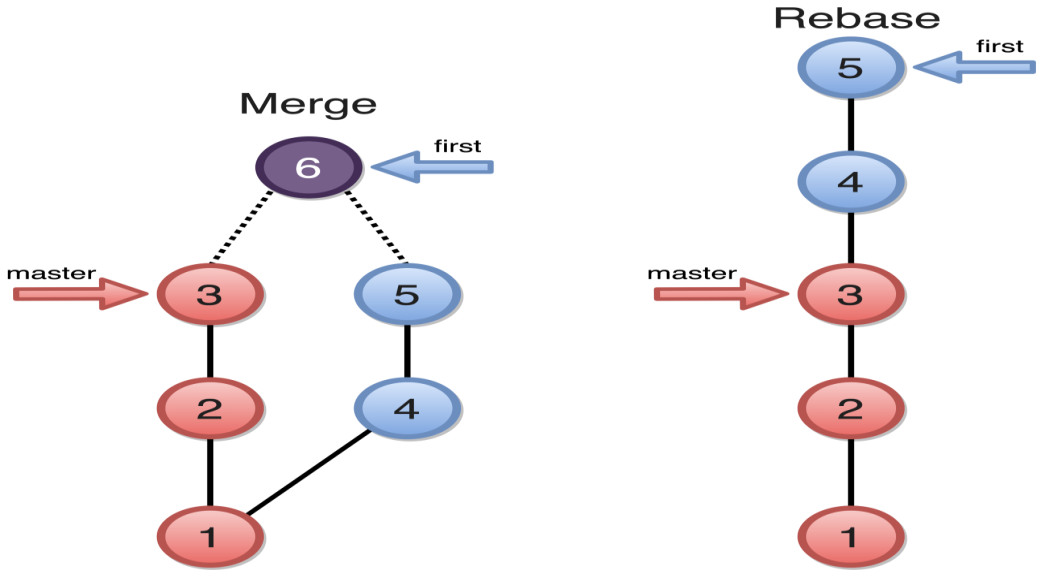
**After merging it will automatically create additional commit like two branches merged.**

****

**Rebase : rebase will merge the branches while adding commits of feature branches upon the latest commit of master branch.**







**Git cherry-pick: Suppose If you wanted to merge only specific commit from other branch to master branch, Then from masterbranch use cherry-pick commitHashCode(of that specific commit) this will merge only that specific commit content in master branch.**

**Merge Conflict : when two persons working on different branches but on same file or same line of file then done commits on there respective branches, then on merge there will be merge conflicts.**

**Resolving merge conflicts :**

**1.Accept current changes : accept the changes of respective branch (where u present) while having merge.**

**2. Accpet Incoming changes : Accept the Incoming changes from other branch.**

**Git Tag : You This help to track certain milestone or Released Version( e.g v1.0) with tag name instead Of Using CommitHash Code.**

**git tag tagname commithashcode(Light weighted Tag, refers any specific commit), other is git tag –a tagname(Annotated tag, refers only branch Head tag, Majorly used, also shows TaggerName and Date Information).**

**git tag Command show all tags**

**git show tagname Command show commit details using TagName**

**Git Detached Head : You may have common commit from Different branches, which means commit with same hashCode present in Multiple Branches, so if u use git checkout commitHashCode , then u will go back to specific Instance in the history but will not refereeing to any branch at that moment. In short u will lose the track of head from all branches.**

**If u want to change Something in that specific commit then do the Respective Change and Commit that changes, that Commit will refer to dethached Head only (not any specific branch) then Make new temporary Branch like using git branch DetachedHeadBranch new then switch brack to master branch and merge master branch with temporary DetachHeadBranch, Now that respective commit change will be in Master Branch and ,U can delete that temporary detachedHeadBranch.**

**Use Git checkout AnyBranchName to get back to specific branch and if u don’t want to stay in detached head Condition.**

**Git STash is use to stash your current changes in some memory and get back you to latest commit.**

**Git STash apply will get back u to the code which u have stashed**

**Git STash list can give u all stash list**

git commit --amend --no-edit: this is used to do changes in the previous commit, if you dont want a new commit

git checkout filename: this is used to discard all changes in working area and switch to previous commit.

Note: This command will only discard the changes in the working area. If you have already staged the changes, you need to use reset command.

git checkout commitHash: this is used to swith to specific commit. you can use git log to get commithash.

git reset filename : this will move the file from stagging area to working directory.

$ git revert --no-commit HEAD HEAD~1 HEAD~2

$ git revert HEAD HEAD~1 HEAD~2 use this..

$ git revert --no-commit HEAD~3..HEAD

GITHUB (cloud Platform to store code)

We store our project code into repositiories present on github know as remote repositories.

GIT is local repository and GITHUB is remote repository

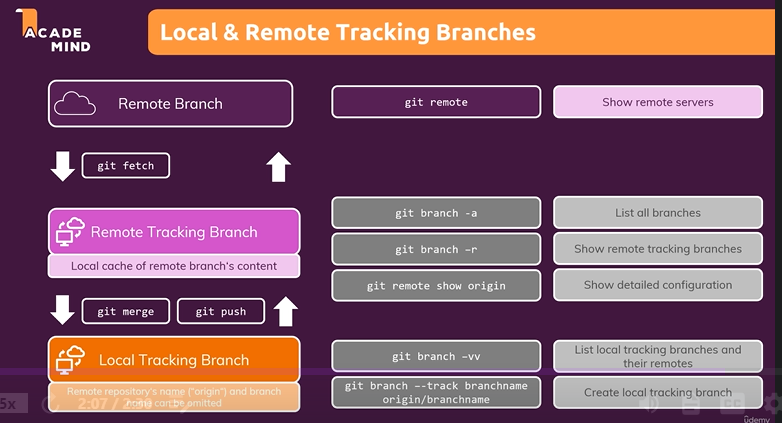
Connecting local repo and remote repo :

git remote add origin “Repository Link from gitHub”

git push origin master(branchName) command used to push the commit/code to remote repo.

Git pull origin master(branchName) command used to pull the code to local repo.

Git pull = git fetch + git merge



Git branch –a command will show all local branches and remote branches.

Git branch –r command will show all remote branches

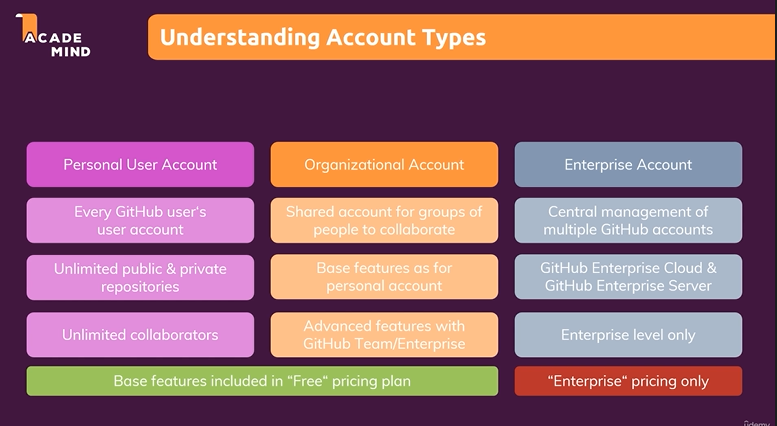
Git branch --track branchname origin/branchname command will create local tracking branch.(Note both remote and local branchName should be Same) this will create branch in local which is link to respective remote branch. Now u can use only git push/git pull as this branch already linked.

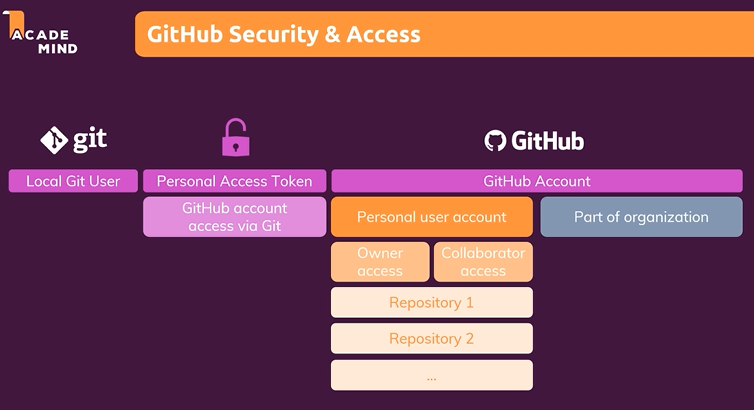
Git branch –vv : this command show all the remote and local linked branches.

Git push –u origin branchName : this command will automatically create local tracking branch(i.e automatically linked the local branch and remote branch)

Git fetch origin : this will fetch the remote branches into remote Tracking branch environment which was created directly on gitHub.

If you delete commit in local then after u push u will get warning because that deleted commit is present in remote, so should do git –force push





Note : To connect your git to github you need personal Access Token to push something to gitHub.

Public repositories: Visible to everyone also who don’t have github account, [Note: but can’t collaborate without personal access token]

Private repositories: Not visible to everyone.

Collaborate in public repositories:

If someone want to collaborate in your project and he also having personal gitHub account like You, then instead of sending your personal Token to him, You can add colloborater in your github Accnt, that will send invite to other person, then he can accept the invite. If he wants to push something in repository then he can generate token from his own account ad use it.

To generate token go to setting -> developer settings -> generate Token.

Collaborate in private repositories:

there is no difference in collaboration in private repo, its same as we add collaborator in public repository.

Teams Collaboration Inside Organizaton accounts:

Inside organization account, you can create team, so it will provide flexibility to provide different type of access to different repositories for different members of a team.

Git Fork and Pull request:

Git fork help to copy repository from 1 account to another account.( Note Fork is not a git command)

While working on Open source Project, you don’t directly clone the public repository, you first fork/copy the Project in your own remote repository then clone to your local repository-> do some changes-> push to your remote repository-> make a pull request to Project Owner repository, then if code is correct -> owner will accept pull request and Merge with its original code.

Issues: Issue tab in github dashboard help us to identify and create major Issues inside projects. E.g While working on open source project we focus on Issues.

Projects: Project tab help us to Keep the track of Issues/Task , like pending Issue/ InProgress / Completed Task like this…