**GIT – Where Everything tracks in the project.**

**Git init 🡪 It will install a empty repo.**

**Git - - version 🡪 To know the version installed on local machine.**

**Git confi - -global user.name ‘vinodh’**

**Git config - -global user.email ‘vinodh@gmail.com’**

**Why?**

**To track who made the changes in the projects.**

**Repo or Repository ;**

**Git - -init**

**First create new directory**

**Mkdir mastering\_git/**

**Now use the init command**

**Git config –global init.defaultbranch Main**

**Master is the default branch name of the repo created by git. Master 🡪 main**

**branch is nothing but a parallel version of ypur project**

**creating new file with code**

**test.js**

**creating another new file called**

**readme.md**

**git status**

**it swill show now youre on main branch**

**no commits yet**

**git add readme.md  
why add?  
before commiting , need to be added to track by using add.**

**we need commit it**

**why commit?**

**To track the changes of what happened at what time . if anything happens in future ,so that you can rollback to pv**

**Git commit –m ‘add readme.md file’**

**Git status**

**Output**

**./idea**

**test.js**

**hello.js**

**git add ./ 🡪 .[dot] after git will tell the git to add, modify,delete to the tracking**

**git commit –m ‘add hello and test files’**

**git log 🡪 git history**

**how do we switch to the older version?**

**First cp the commit hash code**

**Now**

**Git checkout <commit-hashcode>**

**In git there is concept HEAD**

**Which refers to the pointer pointing to the lateset commit you’ve created**

**Head 🡪 readme.md file 🡪 add hello and test files**

**Now the head has been moved to the lastest commit.**

**There is some buggy in the latest version how do we move to the previous version without deleting any?**

**Git checkout <commit-hascode>**

**So now the head will be moved to the previous version**

**Git checkout main 🡪 now this cmmnt will go back to the main**

**Git 🡪 local = laptop or local machine**

**Github 🡪 remote/cloud 🡪 it is used to collab with the team globally**

**When you clone a repository from github, git named as ORIGIN. Default name**

**Origin = github URL**

**We can have multiple origins**

**Origin/main🡪 main is the repo**

**To switch to the default master branch name to Main**

**Git branch –M main**

**To add a origin**

**Git remote add <origin or anyname > <URL>**

**Wait wait**

**How this is enough for communicating with remote profile?**

**No. We need SSH for safe communication. So create a SSH\_KEYGEN, private and public key.**

**Branching and Merging:**

**Main O – O – O – O – O Main is like the source code  
 feat – O – O – O – O Feat is like copy of source code**

**Why do we need feat?**

**Because it wont affect the main or source code, if anything happens in the code or in the production can easily go to the main code.**

**Git branch branch-name**

**Git checkout branch-name 🡪 switch to branch (branch-name)**

Main

Head

Head

**git checkout –b feature-branch 🡪 it will create a branch from the main branch.**

**Git branch new-branch-name source branch 🡪 it will create new branch from the source branch, you can do this action from any branch.**

**Git add ./**

**Git commit –m ‘modified readme’ 🡪 inside the single quotation it should be easy to read.**

**Git push –set—upstream origin feature-branch 🡪it will sync the branch with remote repo**

**Git pull 🡪 this command will pull the changes remote to local repo.**

**Pull Request:**

**What is pull request?**

**Pull request lets share your with your teama for review and feedback. Once approved and merged your changes became a part of the main branch.**

**Keeping the codebase stable and organized.**

**Main O – O – O – O – O**

**BUG – O – O – O**

**DEV – O – O – O**

**MAKE A PR AFTER IT WILL SHOW**

**SUMMARIZE ;**

1. **Clone a repo**
2. **Create a new branch from the main branch or other branch**
3. **Make your changes**
4. **Push the branch to the remote repo**
5. **Open a pull request**
6. **Merge the changes**
7. **Pull the merged changes into your local main branch.**
8. **Repeat from repo**

**Merge Conflict:**

**Why do happen?**

**Because git is unsure which line of code to merge.**

**When you and your other team member is working on same line of code, you and other into the remote repo, now the pull requested has to be accepted to merge. Your TL or PM has accepted the PR, now the same PM is trying merging the your changes, it will show the conflict!! If its happen we have to manually work on the codebase.**

**In simple words, git will get confused.**

**First,  
git checkout main**

**Git pull**

**Git checkout dev-ops**

**Now we have to resolve the conflict, so do this**

**Git merge main**

**Git add ./**

**Git push**

**To remove the commits, but to keep the changes we made**

**Ex:**

**O – O – O – O – O**

**There are 6 commits, but I wanna to go to the 3rd commit and also I wanna delete the commits from 4 to 6, by keeping the changes { its upto the person}**

**Git reset**

**Git reset - -soft <commit hash>**

**Mixed reset  
it will be in the unstaged to the working directory.**

**Hard reset :**

**It will entirely delete the comments**

**Git reset - -hard <commit hash>**

**Git revet ;**

**lets say you broke the production, want to undo it effects without losing the commit history you want to the locks to be there but you want to revert to an old commit.**

**What is staging?**

**Staging is nothing but adding to the track 🡪 git add ./**

**Git stash:**

**Lets temporarily save your uncommitted changes both staged and unstaged without actually committing them.**

**Staged – ready to commit**

**Unstaged – Ready to commit.**

**Git stash**

**It will save the changes to working directory and on dev-ops((branch name)**

**Git add ./ 🡪 git commit –m ‘save the day’ 🡪 git push**

**Git stash list /**

**To get all the list of stashes**

**Ex: stash@{0}WIP on dev-ops:217cdd09 revert” …**

**Git stash apply stash@{0}**