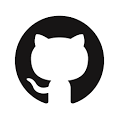
****

**TWO TYPES OF VERSION CONTROL SYSTEM**

**DISTRIBUTED VERSION CONTROL SYSTEM**

**CENTRALIZED VERSION CONTROL SYSTEM**

1. to the initiate the git repo : **git init**
2. to add the file in stage : **git add**
3. to unstage the commit : **git rm –cached <commit id>**
4. to undo the last commit in file : **git restore <filename>**
5. add user in git locally : **git config --global user.name "username"**
6. add user in git locally : **git config --global user.name "useremail"**
7. to access the git hub repo and link the local and remote : **git remote add <url of git account>**
8. to clone the complete project from remote to local machine: **git clone <url of github repo>**
9. git fetch
10. types of cloning in git : HTTPS,Github cli,SSH
11. SSH cloning : **ssh-keygen -t ed25519 -C "Skrishnan586@gmail.com", -> cd .ssh -> ls -> cat it\_ed25519.pub -> copy code and paste in git hub account in settings->ssh->add keygen->save**
12. account to account data copy : **git fork**
13. pulls the code from remote to local repo and merge the code in local repo : **git pull origin main**
14. HARD remove commit from VCS delete data : **git reset --\*hard <commit id>**
15. SOFT remove from VCS but data exists we can add,modify and commit : **git reset <commit id>**
16. **PULL Request** : once developer completes and code and verified to merge with client machine can give request to client copy the code
17. MERGE REQUEST :once developer complete the changes in code and wants to merge with prod branch
18. CHANGE REQUEST : update request from devoper/client
19. to hide the commit and saves in virtual space and by stash pop add in same stage area : .**git stash,save,pop,list,drop**
20. used to merge the branches to update the codes/data : **git merge <BRANCH NAME>**
21. to switch to existing branch to another branch : .git checkout to branch <name>
22. used to update the branches all data will be completely copied without any error and conflicts : **git rebase <commit id>**
23. used to pick and copy the particular commit by commit id from another branch : **git cherry-pick <commit id>**
24. used to remove the last commit from the file and creates a new commit using log reference of delete the commit : **git revert --commit <commit id>**
25. to view the complete work history in git : **git reflog**
26. to restore the deleted branch using commit id : **git branch <branch name> <commitid>**
27. to delete the branch : **git branch -D <branch name>**
28. to rename a branch : **git branch -M <old br name> <new br name>**
29. used to find the error in multiple commits after breaking the pipeline : **git bisect start <successful start commit id > <last fail commit id >**
30. used to commit multiple commits as single commit : **git squash-->not command to use git rebase -i commit id**
31. how will you secure your github account : **to update pwd frequetly-> enabling two factor auth->Keep software up-to-date->Review and manage your connected apps**
32. command to check merged branches in master : **git branch –merged**
33. how to restrict commit on git hub : **settings->branches->click branch to protect->require PR->setting branch protection rule**
34. to rename the commit message : **git commit --amend -m "message name"**
35. disable fork in GIT : **GitHub and click on "Settings->Danger Zone" section and click on "Options->Disable fork" checkbox and click on "Save**
36. git local vs remote **: git is SCM local repo where we can write/modify code -> whereas remote is HUB for acoount management**
37. git config --global core.excludesfile ~/.gitignore\_global: gitignore public repository. You can also use gitignore.io to create a .gitignore file

Git branching strategy refers to the approach and guidelines followed by a development team when creating and managing branches in a Git repository. It helps organize and streamline the development process, enabling parallel work on multiple features, bug fixes, and releases. Here's an example of a simple Git branching strategy:

1. Main Branch:
   * Start with a main branch (often named **main** or **master**) that represents the stable codebase.
2. Feature Branches:
   * When working on a new feature, create a new branch from the main branch:

**git checkout -b feature/my-feature main**

* + Make changes, commits, and iterations on the feature branch.
  + Push the feature branch to the remote repository:

**git push origin feature/my-feature**

1. Pull Request:
   * Once the feature is complete, create a pull request (PR) to merge the feature branch into the main branch.
   * Team members can review the changes, provide feedback, and discuss any necessary modifications before merging.
2. Merge:
   * After the PR is approved, merge the feature branch into the main branch.
   * Update your local repository with the latest changes from the main branch:

**git checkout main git pull origin main**

* + Merge the feature branch into the main branch:

**git merge feature/my-feature**

* + Push the updated main branch to the remote repository:

**git push origin main**

1. Release Branches:
   * Create a release branch from the main branch to prepare for a new release:

**git checkout -b release/v1.0 main**

* + Perform any necessary release-specific tasks (e.g., version updates, bug fixes).
  + Merge the release branch into the main branch and create a tag for the release:

**git checkout main git merge release/v1.0 git tag v1.0**

* + Push the updated main branch and the new tag to the remote repository:

**git push origin main --tags**

1. Hotfix Branches:
   * If a critical bug is discovered in the released code, create a hotfix branch from the main branch:

**git checkout -b hotfix/bug-fix main**

* + Fix the bug on the hotfix branch and merge it into the main branch.
  + Merge the hotfix changes into the release branches as needed.
  + Push the updated branches to the remote repository.

This example represents a simplified branching strategy, but in real-world scenarios, more branches and variations may be involved based on the project's complexity and requirements. The key idea is to have a structured approach to manage parallel development, isolate changes, facilitate collaboration, and ensure the stability of the main branch.