GIT COMMANDS: -

1. First, I want to create the git hub account.
2. Next create the ne git repository.
3. Next install the git in laptop
4. Next open the git bash and create the new folder [ mkdir newfolder].
5. Next instilities the git using the cmd as[git init].
6. Next add that in local repository using [git add .] .
7. Next access the git repository in central repo [ git remote origin “giturl”].
8. Next in case you can change the branch name use the [git remote set-head origin branchname ] . [git branch -m master main].
9. Next add the data in file using [ git commit -m “this my message”]
10. Next push the code in repository [ git push origin main]
11. Git getting any issue means use this cmd [ git pull origin main –-rebase]
12. Git branch push cmd [ git push origin -u PPD]
13. Creating git ne branch[ git branch -b branchname ]
14. Checking the correct reprository [ git remote -v ]
15. Present user name in git reprository [ git config –global user.name ]
16. [ git log - -oneline ] it can be show the git merged branches.
17. [ git log -3] it can show the how many commits are you need to see.
18. [ git log --graph] it can be show the branches and with commit .
19. [ git branch -D branchname ] it can be deleted the my branch in local Repository.
20. [ git push origin - -delete branchname ] it can be deleted the branch in central Repository.
21. [ git branch -r ] it can be show the after removing files.
22. [ git tag ] it can be show the in particular branch how many tags are there.
23. [ git tag <tag name> ] it can be create the new tag in particular branch.
24. [ git tag -a <tag name> -m “enter message” ] it can be created the annotated tag>
25. [ git push origin <tag name> ] it can be push and add the remote branch tag.
26. [ git push –tags ] it can push the all tags In particular branch.
27. [ git show <tag name> ] ]it can be show the details of the particular tag.
28. [ git tag -d <tag name> ] it can be deleted the local Repo Tag.
29. [ git push origin - - delete <tag name> ] it can be delete the Remot Repo tags.

Explanation: -

**1) Git Branches Explanation: -**

**Understanding Git Branches and Their Usefulness**

Git branches allow you to work on different features, fixes, or experiments independently without affecting the main codebase. Each branch represents an isolated version of the project, enabling better collaboration and development workflow.

**Uses of Git Branches**

1. **Feature Development**
   * Use a branch to develop a new feature.
   * Example: feature/login, feature/dashboard.
   * Purpose: Allows parallel development of features without interfering with the main branch.
2. **Bug Fixes**
   * Create a branch to fix bugs.
   * Example: bugfix/login-error.
   * Purpose: Keeps bug fixes separate until they are tested and ready to be merged.
3. **Release Branches**
   * Used for preparing production-ready code.
   * Example: release/v1.0.0.
   * Purpose: Allows testing and last-minute fixes before deploying to production.
4. **Hotfix Branches**
   * For urgent fixes on the production code.
   * Example: hotfix/critical-issue.
   * Purpose: Quickly patch production without waiting for other development work to finish.
5. **Experimentation**
   * Test new ideas without affecting the stable version.
   * Example: experiment/new-algorithm.
   * Purpose: Safely explore new concepts and decide whether to integrate them later.
6. **Staging**
   * Prepare changes for deployment to a staging environment.
   * Example: staging.
   * Purpose: Simulate production to ensure stability.
7. **Documentation Branch**
   * Create or update project documentation.
   * Example: docs/update-readme.
   * Purpose: Keeps documentation tasks organized and separate from code changes.

**How to Use Your 7 Branches Effectively**

1. **Main Branch (main)**
   * Stable production-ready code.
   * Use this branch for deployment.
2. **Development Branch (development)**
   * Consolidates all new features and bug fixes before they are merged into main.
3. **Feature Branches (feature/xyz)**
   * Each branch corresponds to a new feature.  
     Example: feature/login, feature/registration.
4. **Bugfix Branches (bugfix/xyz)**
   * Address specific bugs independently. Example: bugfix/ui-issue.
5. **Hotfix Branch (hotfix/xyz)**
   * For critical fixes in production.
6. **Release Branch (release/vX.Y.Z)**
   * Used for final testing and preparing a version for production.
7. **Experimental Branch (experiment/xyz)**
   * For trying out new ideas or tools.

**Branch Workflow Example**

1. **Create a new branch**

git checkout -b feature/new-feature

1. **Make changes and commit**

git add .

git commit -m "Added new feature"

1. **Merge back into development**

git checkout development

git merge feature/new-feature

1. **Push changes to remote**

git push origin development

**Benefits of Using Branches**

* **Isolated Workflows:** Work independently without affecting others.
* **Collaborative Development:** Team members can work on different features simultaneously.
* **Safe Code Integration:** Test and review code before merging.
* **Version Control:** Easily roll back changes if needed.

Let me know if you'd like guidance on specific Git workflows! 😊

If you've deleted branches in your local repository but they still exist in the remote (central) repository, you need to delete them explicitly from the remote repository as well. Here's how you can handle this:

**Steps to Delete Remote Branches**

1. **Check the Remote Branches**  
   Run the following command to see all remote branches:

git branch -r

1. **Delete a Remote Branch**  
   To delete a specific branch from the remote repository, use this command:

git push origin --delete <branch-name>

Replace <branch-name> with the name of the branch you want to delete.

**Example**:  
If the branch is named feature/login:

git push origin --delete feature/login

1. **Verify the Deletion**  
   After deleting the branch, check if it’s removed from the remote repository:

git branch -r

git fetch --prune

**Clean Up Local References to Remote Branches**

Even after deleting a remote branch, you may still see references to it in git branch -r. To clean up stale references, run:

git remote prune origin

**Summary of Commands**

1. Delete a local branch:

git branch -d <branch-name>

1. Delete a remote branch:

git push origin --delete <branch-name>

1. Clean up stale remote branches:

git remote prune origin

Let me know if you need further clarification! 😊

**2) Git Tags Explanation: -**

### ****What Are Git Tags?****

Git tags are used to mark specific points in your project's history, often for releases (e.g., v1.0, v2.0). They act as **milestones** and provide a way to easily reference a particular commit.

### ****Usefulness of Git Tags****

1. **Mark Releases**
   * Tags are commonly used to label a specific version of the project.  
     Example: v1.0, v1.1-beta.
2. **Easier Rollbacks**
   * Tags help quickly identify and roll back to a previous stable state.
3. **Reference Key Points in History**
   * Use tags to reference important commits, like the start of a major feature or a stable release.
4. **Improved Collaboration**
   * Tags make it easy for team members to work with specific versions of the project.
5. **Deployment**
   * Automate deployment processes by referencing tags in CI/CD pipelines.

### ****Types of Git Tags****

1. **Annotated Tags**
   * Includes metadata like the tagger’s name, email, date, and a message.
   * Suitable for public releases.
   * Example: v1.0.0
2. **Lightweight Tags**
   * Simple pointers to a commit, like a branch but immutable.
   * Suitable for temporary or internal use.

### ****How to Use Git Tags****

#### 1. **List All Tags**

git tag

#### 2. **Create a Lightweight Tag**

git tag <tag-name>

Example:

git tag v1.0

#### 3. **Create an Annotated Tag**

git tag -a <tag-name> -m "Tag message"

Example:

git tag -a v1.0 -m "Initial release"

#### 4. **Push Tags to Remote**

By default, tags are not pushed to the remote repository. You can push them with:

git push origin <tag-name>

Push all tags:

git push –tags

#### 5. **Check Tag Details**

git show <tag-name>

#### 6. **Delete a Tag Locally**

git tag -d <tag-name>

#### 7. **Delete a Tag from Remote**

git push origin --delete <tag-name>

### ****Example Workflow****

1. **Create a release tag:**

git tag -a v1.0 -m "First stable release"

1. **Push it to the remote:**

git push origin v1.0

1. **Collaborators can check out the tag:**

git checkout v1.0

### ****When to Use Git Tags?****

* Before releasing a new version.
* To mark a milestone in development.
* To signify a stable point in the project’s history.