Name : Santanu Kumar Nayak Regdno:2141014096

GIT TUTORIAL

- Git is a version control tool .
- It provides total development freedom and make save points that saves your project.
- Github is a service that hosts git repositories collaboration.

Creating a Git Repository

1. Git init :

- *Initializes a new Git repository in the current folder.*
  It creates a hidden .git folder that tracks changes.

  git init

Viewing Files and Status

1. ls-al :

- *Lists all files and folders, including hidden ones (.git).*
  Useful to confirm that .git has been created.

  ls -al

2.git status:

- *Shows the current status of the working directory and staging area.*
  Tells you which files are staged, unstaged, or untracked.
- **git status**

Cloning a Repository :

- git clone <repository-url>
- *Copies an existing Git repository (usually from GitHub) to your local machine.*
- **git clone https://github.com/username/repo-name.git**

Viewing Commit History :

- git log
- *Shows a list of commits with details like author, date, and commit message.*
- ***git log***

Review a Git Repository – History Commands

These commands help you look into the commit history, changes made, who made them, and how files evolved over time.

1. **git log :**
    - Shows the commit history in reverse chronological order (latest first).
    - **Includes:** Commit ID, Author , Date , Commit message .
    - **git log**

2. **git log -p :**
    - Shows the patches (diffs) — i.e., the actual lines added or removed in each commit.
    - Useful when you want to see what changed in each commit
    - git log -p

3. **git log –stat :**
    - Shows the **summary of file changes** in each commit.
    - **Includes :** Files changed , Insertions and deletions count (in numbers and bars)
    - **git log --stat**

4. **git show :**

Displays the details of a specific commit (by default, the latest one).

Includes: Commit metadata ,Patch (actual changes),Useful for inspecting a single commit

git show

** To show a specific commit

git show <commit-hash>

Let's Make a Commit – Step-by-Step

1. **git add :**
    - Purpose: Stages changes (files or folders) for the next commit.
      Without this, Git won't include the changes when you commit.
    - **git add <filename>**
    - # or stage everything:
    - **git add .**

2. **git commit :**
    - Purpose: Commits the staged changes with a message.
      This creates a permanent snapshot in your local Git history.

git commit -m "committed successfully"

** git commit -am "message"

- Stages AND commits *already tracked files* in one step.
  (Note: This does not include untracked/new files.)

git commit -am "Fixed bug in login"

3. **git diff :**
    - **Purpose:** Shows the differences between:
    - Unstaged changes vs. last commit
    - Or between any two commits, branches, etc.
    - **git diff** # See unstaged changes
    - **git diff --staged** # See staged changes

4. git restore
    - Purpose: Discards changes in the working directory.
    - Use when you want to undo edits before staging.
    - **git restore <filename>**
    - You can also use it to unstage :
    - **git restore --staged <filename>**

**** .gitignore :**

* **Purpose:** Tells Git which files/folders to **ignore** — i.e., not track or commit.

* Commonly used for:

Log files

Node modules

Build folders

Secrets and config files

* Example .gitignore file:

```
node_modules/
.env
*.log
dist/
```

**** Just create a .gitignore file in your repo's root and list what you want to ignore.

Branching, Tagging & Merging in Git:

1. git branch :
- Lists all branches in your repository.
  Also shows the current branch (marked with a *).
- **git branch**

2. git branch -d <branch-name> :
- Deletes a local branch (after it has been merged).
  Use -D to force-delete.

  git branch -d feature-branch

3. git checkout <branch-name>
- Switches to another branch.

  git checkout main

4. git checkout -b <branch-name>
- **Creates a new branch and switches to it** immediately.

  git checkout -b feature-login

** **git merge <branch-name> :**

- Merges another branch into your current one.
  Used to combine work from different branches.

  git merge feature-login

Tagging in Git :

1. **git tag :**

- Lists all tags in the repository.

git tag

2. git tag -a v1.0 -m "Release version 1.0"

- Creates an annotated tag with a message.

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

3. git tag -d <tag-name> :

- Deletes a local tag.

git tag -d v1.0

** Stashing Changes

1. **git stash :**
   - Temporarily saves uncommitted changes so you can switch branches safely.

   git stash

2. **git stash list :**
   - Shows a list of all stashed changes.

   git stash list

3. **git stash apply :**
   - Re-applies the most recent stash.

   git stash apply

** Working with Remote Repos :

1. **git pull :**
   - Fetches changes from the remote repo and merges them into your local branch.

   git pull

3. **git push -u origin master :**
   - Pushes your branch to the remote and sets the upstream (tracking) branch.

   git push -u origin master

** Undoing Commits in Git :

1. **git commit –amend :**

- Use when: You want to edit the last commit (e.g., to fix the message or add a forgotten file).
  It replaces the previous commit with a new one.

  git commit --amend

  Useful for fixing small mistakes without making a new commit.

2. **git revert <commit-id>**
   - Use when: You want to undo a specific commit, but keep history clean.
     It creates a new commit that reverses the changes of the given commit.

     git revert abc1234

     Safe to use in shared branches (like main)

3. **git reset**
   - Use when: You want to move your HEAD (current branch pointer) back to a previous commit.
4. **git reset --soft <commit-id>**
   - Moves HEAD to the chosen commit.
     Keeps changes in the staging area (ready to commit again).

     git reset --soft abc1234

5. **git reset --mixed <commit-id> (default)**
   - Moves HEAD back, and
     Keeps changes in the working directory,
     But unstages them.

     git reset --mixed abc1234

6. **git reset --hard <commit-id>**

   - Danger Zone:
     Completely resets everything —
     Moves HEAD,
     Deletes staged + working directory changes.

     git reset --hard abc1234

❗ This is irreversible unless you've backed up or have reflog.

- :wq
- This is a **Vim command**, not Git.
  You use it when editing a commit message in the terminal.

- :w = write (save)

- :q = quit
- So :wq means "save and exit" in Vim.