Git & GitHub

1. Setup & Basics

- **git init**
  When: Start tracking a new project (creates a .git folder).
- **git clone <repo-url>**
  When: Copy an existing repository to your local machine. (used to contribute in Open source Projects)
- **git config --global user.name "Your Name"**
  When: Set your name/email (do this once per machine).

---

2. Making Changes

- **git status**
  When: Check which files are modified/staged/untracked.
- **git add <file>**
  When: Stage changes for committing (use git add . for all files).
- **git commit -m "message"**
  When: Save staged changes to history.
- **git diff**
  When: See **unstaged** changes (before git add).
- **git diff --staged**
  When: See **staged** changes (before git commit).

---

3. Branching

- **git branch**
  When: List all branches (current branch has a *).
- **git branch -all**

  When: use to see all the branches including **remote branches**.
- **git branch <branch-name>**
  When: Create a new branch. **But not switched , its in current branch.**
- **git checkout -b <branch-name>**

  When: Create a new branch and switched to that new branch.
- **git checkout <branch-name> (older one)**
  When: Switch to another branch. Also used to time travel to previous commits.
- **git switch <branch-name> (newer one)**

  When: Switch to another branch.
- **git pull origin <branch-name>**

  When: used to pull the remote repo to local.

- **git push origin <branch-name>**

  When: use to push the local repo to remote.

- **git merge <branch-name>**
  When: Combine changes from another branch into your current branch. (It shows the branches)

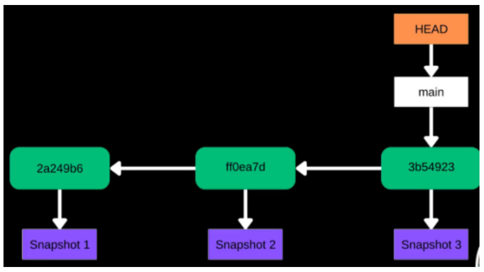
- **git rebase <branch>**
  When: Move your branch's commits to the tip of another branch (cleaner history). (It shows linearly)

- **git branch -d <branch-name>**

  used to delete branch.

This is how Branching works.

To see the commit graph use git log --graph command.



How Branches Work

Step 1: The Default Branch

- Every Git repo starts with a default branch, usually called main or master.
- This is your "source of truth" (the official version of your project).

Step 2: Create a New Branch

- When you create a new branch (e.g., git branch login-feature), Git makes a copy of the current code (main) at that exact moment.
- Think of it as forking off a new timeline. You can now edit files freely here.

Step 3: Switch to the Branch

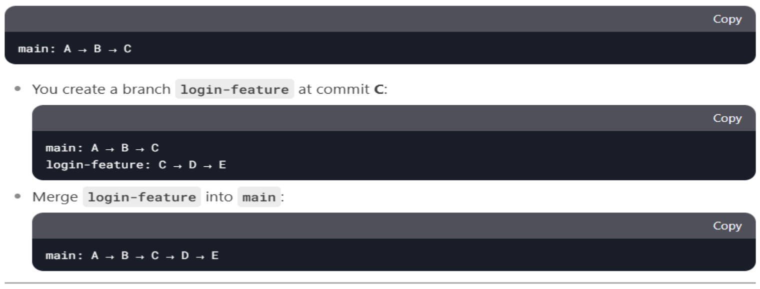
- Use git checkout login-feature to move to your new branch. Now, any changes you make happen *only in this branch*.

Step 4: Commit Changes

- As you edit files and commit (git commit), the branch grows independently of main.
- The main branch remains unchanged.

Step 5: Merge When Ready

- Once your login-feature works perfectly, merge it back into main.



```
main: A → B → C
```

- You create a branch `login-feature` at commit **C**:

```
main: A → B → C
login-feature: C → D → E
```

- Merge `login-feature` into `main`:

```
main: A → B → C → D → E
```

The above is visual example.

4. Syncing with Remote

→ when you want to share your code in Internet, or sometimes it may be private or it may be public. Simply known as "Repository."

- **git remote -v**
  When: Check linked remote repositories (like GitHub).

  command **used to see** which is our **origin.**

- **git fetch**
  When: Download updates from remote **without merging**.

- **git pull**
  When: Fetch + **merge** remote changes into your branch.

- **git push origin <branch>**
  When: Upload your local commits to a remote repository.

- **git tag -a <give_VersionName>:**

  When: use **git tag -a <version name>** to naming the version.

- **git show <version>:**

  When:use **git show <version>** to see details about particular version.

---

5. Undoing Mistakes

- **git reset <file>**
  When: Unstage a file (keep changes in working directory).

- **git reset --hard HEAD**
  When: Discard **all** unstaged changes (⚠ use with caution!).

- **git revert <commit-id>**
  When: Undo a specific commit by creating a new commit.

- **git checkout -- <file>**
  When: Discard changes to a specific unstaged file.

- **git clean -fd**
  When: Delete untracked files/folders.

---

6. Checking History

- **git log**
  When: View commit history (press q to exit).

- **git log --oneline**
  When: Compact commit history (shows short commit IDs).

- **git reflog**
  When: Track **all** actions (even deleted commits).

---

7. Remove/Delete Files

- **git rm <file>**
  When: Delete a file **from disk** and stage the deletion.
  Example: git rm old_file.txt → Deletes the file and stages the change.

- **git rm --cached <file>**
  When: Stop tracking a file but **keep it on your disk** (untrack it).
  Example: git rm --cached secret.txt → Removes it from Git history but leaves it locally.

---

8. Remove Branches

- **git branch -d <branch>**
  When: Delete a **local branch** (safe: checks if merged).
  Example: git branch -d feature-bugfix

- **git branch -D <branch>**
  When: **Force-delete** a local branch (even if unmerged).
  Example: git branch -D experimental

- **git push origin --delete <remote-branch>**
  When: Delete a **remote branch** (e.g., on GitHub).
  Example: git push origin --delete old-feature

---

9. Remove Commits/Changes

- **git reset --hard HEAD~1**
  When: **Permanently delete** the last commit and all its changes.
  ⚠ *Warning*: Unrecoverable! Use HEAD~1 to remove the latest commit.

- **git revert <commit-id>**
  When: Undo a commit **safely** by creating a new commit that reverses it.
  Example: git revert abc123 → Good for shared branches.

- **git checkout -- <file>**
  When: Discard **unstaged changes** to a specific file.
  Example: git checkout -- broken.js → Restores the last committed version.

---

10. Remove Untracked Files

- **git clean -n**
  When: **Preview** untracked files that will be deleted (dry-run).

- **git clean -f**
  When: Delete **all untracked files** in the current directory.
  ⚠ *Warning*: Cannot undo!

- **git clean -fd**
  When: Delete untracked files **and directories**.

11.clone command :

- **git clone <link>**
  when: you want to clone a project use this in command prompt and paste the link of HTTPS of that project
  Ex:

  ```
  git clone https://github.com/torvalds/linux.git
  ```

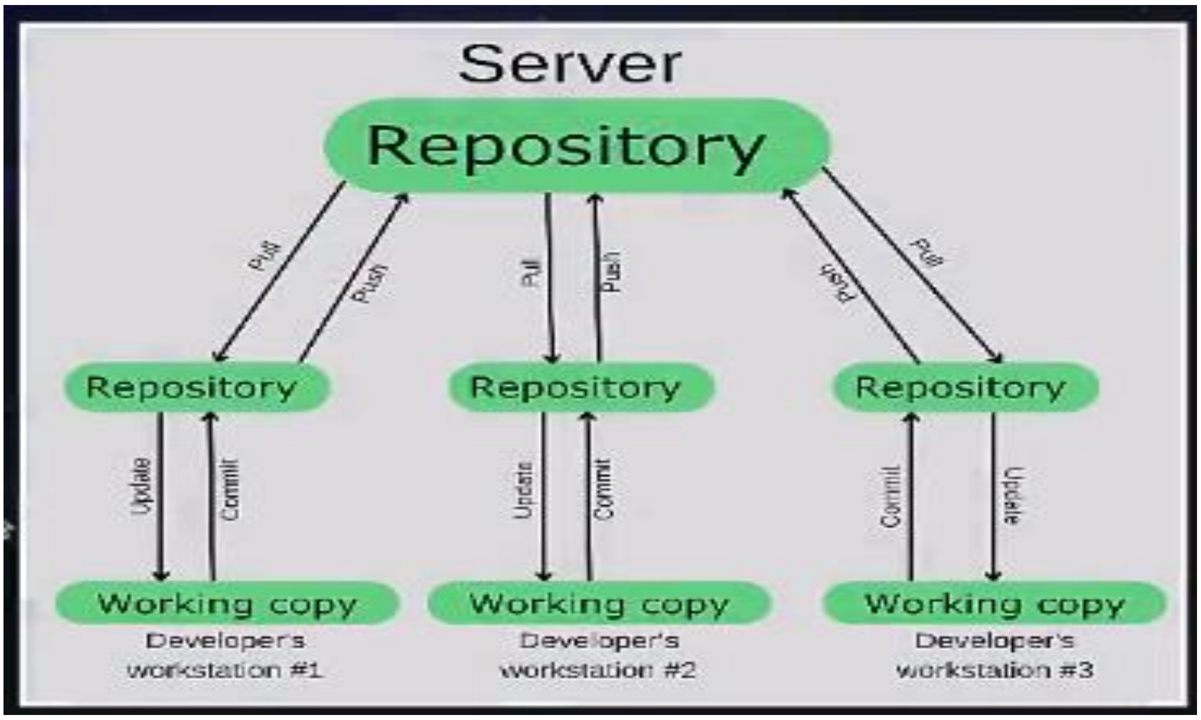
  . This is linux open source code's link.

12.SSH & HTTPS:

- **SSH:**
  Preferred for developers who push/pull code frequently.
  Better for managing multiple GitHub accounts (via multiple SSH keys).
- **HTTPS:**
  Easier for beginners (no key setup).
  More reliable in restrictive networks (e.g., public Wi-Fi, corporate firewalls).

Pro Tips to Remember

- **Always check git status** to know where you are.
- **Commit small chunks** (logical units) for easier debugging.
- **Use git commit -am "message"** to skip git add (only for modified files, not new ones).
- **Pull before you push** to avoid conflicts.
- Use git rm for tracked files (deletes from disk + Git).
- Use --cached to untrack a file without deleting it.
- **Avoid git reset --hard** unless you're sure—no undo!
- Prefer git revert for public/shared branches to preserve history.


Server
Repository
Pull
Push
Pull
Push
Push
Pull
Repository
Repository
Repository
Update
Commit
Update
Commit
Commit
Update
Working copy
Working copy
Working copy
Developer's workstation #1
Developer's workstation #2
Developer's workstation #3