1. GIT PULL:

git pull = git fetch + git merge

It **downloads** the latest changes from the main branch of the remote (origin) AND **automatically merges** them into your current local branch.

➢ ***git pull origin main*** # use this to pull data

Note: used to fetch and download content from a remote repo and immediately update the local repo to match that content

2. GIT FETCH:

It **downloads** all the latest changes from the remote repository, but it **does not merge** them into your current work.

➢ ***git fetch origin***

✅ **Common Usage Flow**

git fetch origin # Download changes but don't merge

git status # Check if your branch is behind

git diff origin/main # See what's changed

git merge origin/main # Manually merge changes if needed

3. GIT REMOTED:

WE ALREADY KNOW LITTLE BIT...ABOUT THIS LIKE

When you **start a project on your local computer**, and then want to **push it to GitHub**, you use a **remote** to tell Git **where** to send your code.

LET'S KNOW MORE..:

- **1. What is a Remote?**

A remote in Git is a pointer or reference to a version of your repository that is hosted elsewhere, usually on platforms like GitHub, GitLab, or Bitbucket.

- **2. Why Do We Use a Remote?**
  - To connect your local Git repo with an online version.
  - To push your local commits to GitHub.
  - To pull others' commits into your local project.
  - To **collaborate** with a team.

- **3. Common Remote Name: origin**
  - origin is the default name Git gives when you clone or add a remote.
  - It is just a nickname for the GitHub URL.

- **4. Basic Workflow Example**

👉 When You Start Locally:

git init # Create a local git repo

git remote add origin https://github.com/user/repo.git

git push -u origin <branch name> # also use git push origin <branch name>

👉 **When You Clone:**

git clone https://github.com/user/repo.git

 # Git automatically sets the remote to 'origin'

🔹 **5. Important Git Remote Commands**

Command	Description
git remote -v	Show all remotes (with URLs)
git remote add <name> <url>	Add a remote (usually called origin)
git push <remote> <branch>	Push branch to remote
git pull <remote> <branch>	Pull branch from remote
git remote remove <name>	Remove a remote
git remote rename <old> <new>	Rename a remote
git remote show <name>	is used to display detailed information about the remote named origin.
git remote update	Fetches remote branches without merging

WE DON'T USE MERGE WE JUST PULL LATEST CODE AND MAKE CHANGES ACCORDING TO US AND AFTER THAT WE SIMPLY ADD, COMMIT, PUSH. IF CONFLICT APPEAR SIMPLE ADD FILE MANUALLY KEEP WHATEREVER YOU WANT OR CHANGE AND AFTER COMPLETING THAT SIMPLY ADD, COMMIT AND PUSH.

4. GIT CONFLICT:

- **When Do Conflicts Happen?**

  Conflicts usually occur during:

  - git merge
  - git rebase
  - git pull (if it does a merge inside)

- **Conflict Resolution Flow (Visual):**

```
git merge feature
        |
        V
Conflict Detected
        |
        V
Manually Edit File
        |
        V
git add <file>
        |
        V
 git commit
```

5. GIT FORK:

In **Git**, a **fork** is a **copy of a repository** that is made under your own GitHub (or GitLab, Bitbucket, etc.) account. Forking is commonly used when you want to:

- **Contribute to someone else's project**
- **Experiment** with a project without affecting the original
- **Make changes independently**, especially in open-source collaboration

🧭 Visual Flow Diagram

Original Repo (Upstream)

↓ Fork

Your GitHub Repo (Forked Copy)

↓ Clone

Your Local Machine

↓ Create branch → Code → Commit

↓ Push ***# git push origin your-feature-branch***

Your GitHub Fork

↓ Pull Request (PR)

Original Repo (Upstream)

🔹 1. Fork the Repository:

- Go to the GitHub repository you want to contribute to.
- Click the **"Fork"** button (top-right corner).
- This creates a **copy in your GitHub account**.

- **2. Clone Your Fork Locally:**

  git clone https://github.com/your-username/forked-repo.git

  cd forked-repo

- **3. Set the Original Repository as upstream**

  To keep your fork updated with the original repo:

  git remote add upstream https://github.com/original-user/original-repo.git

  Check remotes:

  git remote -v

- **4. Create a New Branch for Your Changes**

  git checkout -b your-feature-branch

- **5. Make Changes and Commit**

  # Make your changes using an editor

  git add .

  git commit -m "Describe your changes"

- **6. Push to Your Fork**

  git push origin your-feature-branch

🔹 7. Create a Pull Request (PR)

- Go to your fork on GitHub.
- Click **“Compare & pull request”**.
- Write a meaningful description and **submit the PR** to the original repo.

🔹 8. Sync with Original Repo (Optional but Important)

To keep your fork up-to-date:

git checkout main

git pull upstream main

git push origin main

6. GIT SQUASH:

Git Squash means **combining multiple commits into one single commit**.
It helps keep your Git history **clean and organized**, especially before merging a feature branch.

- **Git Squash Workflow (Step-by-Step):**

You made 4 commits in your branch:

* c4e1: fix spelling

* b2f3: update button color

* a1d2: add login form

* 91e3: initial setup

You want to squash the **top 3 commits** into one (keep the initial commit).

- **Step 1: Start Interactive Rebase:**

  # This command opens the last 3 commits in your default text editor.

  git rebase -i HEAD~3

- **Step 2: Edit the Commit List**

  You'll see something like this:

  pick a1d2 add login form

  pick b2f3 update button color

  pick c4e1 fix spelling

➞ Change it to:

pick a1d2 add login form

squash b2f3 update button color

squash c4e1 fix spelling

✅ Only the **first** commit stays pick, others become squash (or s)

🔹 **Step 3: Edit the Combined Commit Message**

Git will then show:

Commit message for your squash:

add login form

The following commits will be merged:

update button color

fix spelling

➞ Change it to a single message like:

add login form with styled button and typo fix

🔹 **Step 4: Push the Squashed Commit**

If you already pushed the original commits before:

git push --force origin your-branch-name

⚠️ Use --force carefully: only on personal or feature branches, **not on main/shared branches**.