* git reset --hard
  + Purpose: Moves the branch pointer to a specific commit and discards all changes.
  + Effect: It deletes uncommitted changes and commits after the specified commit.
  + Common Usage:
    - git reset --hard HEAD~3 → Moves the branch back 3 commits and deletes them.
    - git reset --hard origin/main → Forces local branch to match remote.
* git rebase
  + Purpose: Moves a branch to a new base commit, applying changes on top of another branch.
  + Effect: It rewrites commit history by reapplying commits on a new base.
  + Common Usage:
    - git rebase <branch> → Moves the current branch onto <branch>.
    - git rebase -i <commit> → Interactive rebase to modify history (squash, edit, reorder commits).
* git switch -c <branch-name> or git checkout -b <branch-name>
  + Creates and switches to a new branch <branch-name>.
  + This is the modern alternative to git checkout -b.
* git switch <branch-name>
  + This switches you back to the main branch.
* git push -u origin feature-branch
  + Push to specific branch
* git branch
  + view branches
* git clone –b <branch-name> <repo-url>
  + Clone a repo with specific branch
* git remote add <branch-name> <repo-url>
  + To Attached Local repo to remote github repo.
* git branch -d <branch-name> # Safe deletion (won't delete if unmerged)
* git branch -D <branch-name> # Force delete
* git push origin --delete <branch-name>
  + delete remote branch
* git add <file-name>
  + Add the Specific File
  + Example
    - git add file1.txt file2.js
    - git commit -m "Updated file1 and file2"
    - git push origin main

**SCENARIO:**

In a collaborative Git workflow, **Person A** is working on the main branch, and **Person B** is working on 123. Person B wants to keep their branch (123) updated with main and eventually merge changes. Here’s how the workflow should go:

### ****Step 1: Person A Pushes Code to**** main

1. **A works on main branch**
2. git checkout main
3. **A makes changes & commits**
4. git add .
5. git commit -m "A's changes"
6. **A pushes to main**
7. git push origin main

### ****Step 2: Person B Merges**** main ****into**** 123

Now, **Person B** wants to get the latest changes from main into their branch (123).

1. **Fetch latest changes from remote**
2. git fetch origin
3. **Switch to 123 branch**
4. git checkout 123
5. **Merge main into 123**
6. git merge origin/main
   * If there are **no conflicts**, the merge is successful.
   * If there are **conflicts**, resolve them manually, then:
   * git add .
   * git commit -m "Resolved merge conflicts"
7. **Push 123 to GitHub**
8. git push origin 123

### ****Step 3: Person B Creates a Merge Request****

Once 123 is ready, **Person B** needs to **merge it into main**.

1. **Go to GitHub**
2. **Open a Pull Request (Merge Request)**
   * Navigate to the repository.
   * Click **"Pull Requests"** → **"New Pull Request"**.
   * Select **base: main**, **compare: 123**.
   * Click **"Create Pull Request"**.
3. **Person A reviews & merges**
   * If everything looks good, Person A can **merge** the PR.
   * If changes are requested, Person B updates 123, commits, and pushes again.

### ****Step 4: Repeat Process****

Whenever **Person A** pushes new changes to main, **Person B** must repeat:

git fetch origin

git checkout 123

git merge origin/main

git push origin 123

This keeps 123 updated before merging again into main.

### ****TL;DR: Quick Commands****

# Person A (working on main)

git checkout main

git add .

git commit -m "A's changes"

git push origin main

# Person B (updating 123 with main)

git fetch origin

git checkout 123

git merge origin/main

git push origin 123

# Person B creates a Merge Request

# Person A reviews & merges it

This cycle continues as both developers keep updating and merging their work. 🚀