**SSH Connectivity**

1. **Generate SSH Keys**: Use ssh-keygen to create a **private and public key pair on your PC** or EC2 instance.
2. **Add SSH Key to GitHub**:
   * Copy the contents of id\_rsa.pub (the public key).
   * Go to GitHub → **Settings** → **SSH and GPG keys** → **New SSH Key**, and paste the key.
3. **Clone Repository**: Use the **SSH Git URL** from GitHub to clone the repository securely without repeatedly entering credentials.

**Git Commands**

**1. Git Revert**

* **Purpose**: Creates a new commit to **"undo" a previous commit** by applying the reverse changes.
* **Effect**: Does not alter existing commits, preserving history and making it safe for team workflows.
* **Example**: git revert <commit-hash> (e.g., git revert 98bd832)

**2. Git Reset**

* **Purpose**: Moves the HEAD pointer to a specified commit, changing the branch’s commit history.
* **Effect**: Depending on the reset type, it can stage, unstage, or discard changes.

**Reset Types**:

* **Soft Reset** (git reset --soft <commit>): Moves HEAD without modifying the working directory or staging area. Changes after the reset point stay staged.
  + Useful for undoing the last commit but keeping the changes staged.
  + **Example**: git reset --soft HEAD~1 or git reset --soft <commit-hash>.
* **Mixed Reset** (git reset or git reset --mixed <commit>): Moves HEAD and unstages changes, but keeps them in the working directory.
  + Useful for uncommitting changes to modify them before re-committing.
  + **Example**: git reset HEAD~1.
* **Hard Reset** (git reset --hard <commit>): Moves HEAD, and discards all changes in the working directory and staging area after the reset point.
  + **Use with caution** as it can permanently delete uncommitted changes.
  + **Example**: git reset --hard HEAD~1.

**Pull Requests**

* **Purpose**: Allows developers to propose and discuss changes before merging them into the main codebase.
* **Process**: Developers submit pull requests for code review and feedback.

**1. Add Collaborator**

* Go to your repository on GitHub.
* Navigate to **Settings** → **Manage access**.
* Click on **Invite a collaborator**, enter their GitHub username, and select **Add**.
* The collaborator will receive an invitation they must accept to gain access.

**2. Set Branch Protection Rule (Optional but recommended for production/main branches)**

* Go to **Settings** → **Branches**.
* Under **Branch protection rules**, click **Add rule**.
* **Branch Pattern**: Specify the branch you want to protect (e.g., main or master).
* **Select options** for protection:
  + **Require pull request reviews before merging**: Ensures PRs are reviewed.
  + **Require status checks to pass**: Only allows merging if tests or checks pass.
  + **Require signed commits**: Optional but good for security.
* Click **Create** to save the protection rule.

**3. Create a New Branch**

* Open your terminal and clone the repository locally if you haven’t already:

bash

Copy code

git clone <repository-url>

* Change into the repository folder:

bash

Copy code

cd <repository-name>

* Create and switch to a new branch for your changes:

bash

Copy code

git checkout -b <new-branch-name>

**4. Make Changes and Commit**

* Make the necessary code changes in your new branch.
* Stage the changes:

bash

Copy code

git add .

* Commit the changes with a descriptive message:

bash

Copy code

git commit -m "Description of the changes"

**5. Push the Branch to GitHub**

* Push your new branch to GitHub:

bash

Copy code

git push origin <new-branch-name>

**6. Open a Pull Request (PR)**

* Go to your repository on GitHub.
* You will see an option to **Compare & pull request** next to your recently pushed branch; click it.
* **Fill out the PR details**:
  + **Title**: Clear and concise title for the PR.
  + **Description**: Provide details on the changes made, why they are needed, and any additional context.

**7. Assign Reviewer(s)**

* In the PR page, locate the **Reviewers** section on the right.
* Select the collaborator(s) who should review the PR.
* They will receive a notification to review and provide feedback.

**8. Add Labels and Assignees (Optional)**

* **Labels**: Use labels like bug, enhancement, documentation, etc., to categorize the PR.
* **Assignees**: Assign yourself or the responsible team member(s) to the PR.

**9. Discuss and Address Review Comments**

* Reviewers will examine the code and may leave feedback or request changes.
* Address any comments or requested changes locally, commit them, and push to the same branch.
* GitHub automatically updates the PR with new commits pushed to the branch.

**10. Pass Required Checks**

* If you enabled **branch protection rules** with required checks, make sure all checks (tests, code coverage, etc.) pass.
* If any checks fail, fix the issues and push the changes to the branch until all checks are successful.

**11. Merge the Pull Request**

* Once approved and all checks have passed, you can merge the PR:
  + Click **Merge pull request**.
  + Choose a **merge method** (depending on the repository settings):
    - **Merge commit**: Creates a merge commit.
    - **Squash and merge**: Combines all commits from the branch into a single commit.
    - **Rebase and merge**: Rewrites the commit history to create a linear path.
* Confirm the merge.

**12. Delete the Branch (Optional)**

* After merging, you’ll see an option to **Delete branch** in the PR page.
* Click it to delete the branch from GitHub (recommended to keep the repository clean).