1. **Pull Request:**

* A pull request occurs when a developer asks for changes committed to an external repository to be considered for inclusion in a project’s main repository.
* A common workflow for submitting a pull request with GitHub would look like this:

1. Create/Log in to your GitHub account
2. Go to the page for the code respository you want to contribute to (the “upstream”)
3. “Fork” the repository (this creates a clone to your GitHub account)
4. Create a local clone of your fork with git clone
5. Create a local branch for your changes
6. Make your changes and commit them to your local branch with git commit, ensuring to include a descriptive commit message
7. Push the branch to your GitHub fork using git push
8. Go to the page for the upstream repository go to the pull requests tab
9. Click the “New Pull Request” Button
10. Select the branch you want to submit, and write a summary of what your change explaining what it is intended to do and how it is implemented

**Git restore:**

* The "restore" command helps to unstage or even discard uncommitted local changes.
  + - On the one hand, the command can be used to undo the effects of git add and unstage changes you have previously added to the Staging Area.
    - On the other hand, the restore command can also be used to discard local changes in a file, thereby restoring its last committed state.

**Git reset:**

* + - git-reset - Reset current HEAD to the specified state
    - The git reset command is a complex and versatile tool for undoing changes. It has three primary forms of invocation. These forms correspond to command line arguments --soft, --mixed, --hard. The three arguments each correspond to Git's three internal state management mechanism's, The Commit Tree (HEAD), The Staging Index, and The Working Directory.
* **Git revert:**
* The git revert command can be considered an 'undo' type command, however, it is not a traditional undo operation. Instead of removing the commit from the project history, it figures out how to invert the changes introduced by the commit and appends a new commit with the resulting inverse content. This prevents Git from losing history, which is important for the integrity of your revision history and for reliable collaboration.
* Reverting should be used when you want to apply the inverse of a commit from your project history. This can be useful, for example, if you’re tracking down a bug and find that it was introduced by a single commit. Instead of manually going in, fixing it, and committing a new snapshot, you can use git revert to automatically do all of this for you.

****