Corey Gumbs
Web 25
10/3/2019

Merge Conflict

A merge conflict is when you merge different branches that have commits that are different. They occur when conflicting changes are made on the same line, on the same file.

Git typically will resolve these issues on its own, however, when it can't, you will get a merge conflict warning. To fix a merge conflict, you have to manually fix the line/code that is in conflict. On my VSC editor, Git will actually add warnings to my lines of code when this happens. All changes have to be corrected locally before they can be pushed to the repository.

Pull

A pull request is used to update the local version of a repository from a remote repository or remote repository branch. It basically will update the local code of any changes made on the remote repo.

A git pull request will not update locally if there is any uncommitted code.

Rebase

Git Rebase is an alternative to Git Merge. It combines the changes of one branch into another branch. So rather than have seperate paths of commits when you merge branches, it combines a group of commit histories into a single path of commits, giving the commit history a more linear path of commit history succession.

Merge

A Git Merge is similar to rebase in that in joins the changes/commits of one branch into another. Merge takes the different branches (forks) that may be used in development and merges them into a separate branch. Merge does not affect the commit history of these branches or the branches themselves, unlike Rebase that combines them all together into one branch.

Reset

Git Reset is a tool used for undoing changes. It is comprised of a concept of “Three Trees”. They are:

HEAD: which is the pointer to the current branch reference (git commit)

INDEX: is the proposed next commit or the “staging area” (git add)

WORKING DIRECTORY: Which is sort of a sandbox or the place where git unpacks the files from HEAD and INDEX that were stored in git. (file)

When used git reset will move the head to the requested reference point. At this point it will “undo” the last commit and when you commit, create a new

commit reference to the head pointer you wanted. This process moves the branch back to where it was without changing the Index(“staging”) or Working DIrectory, allowing you create a new “amended” commit.

It uses 3 flags: --soft, --mixed, --hard

--soft flag: is the first thing reset will do and will stop there no matter the type of reset you do.

--mixed: is the default of reset. It will run if there are no flags added to the command. --mixed will update the Index with a snapshot of HEAD points to. (commit reference)

--hard: will update the Working Directory to match the Index .

1. **Move the branch HEAD points to (stop here if `--soft`)**
2. **Make the index look like HEAD (stop here unless `--hard`)**
3. **Make the working directory look like the index**

https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified

Revert

Git Revert is used to ‘undo’ the changes made to a repository’s commit history. But rather than ‘destroy’ the commit it creates a “inverted version” and appends it to a new commit. It does not remove the history of the commit. Unlike the reset, revert does not move the HEAD ref pointer.

Revert will take the specified commit from that pointer, take the inverted changes, and create a new commit. It will then update the pointers to point to the tip “end” of the branch path.

Clean

Git clean is different than reset or revert in that it does not deal with any tracked or committed files. Git clean works on untracked files, or any files not added to the Index via “git add” command.

It would basically remove any untracked files. However, as a safety precaution, it will return an error if the --force or -f flag does not accompany the command. This ensures you don’t accidentally delete any files in your working directory.