Git workflow git-flow

- When working in team we want follow some rules and style to work on project.
- For this we should follow branching model in where there should be specific branches for specific purpose

Git workflow
git-flow

- Branches (example)
  - master
  - development
  - feature/rss-feed
  - hotfix/missing-link

Git workflow git-flow

- When working in team we want to review code of team members and merge code in main branch only after that.
- For this purpose github provide pull request feature.
- In professional projects you will see the very often

Demo

Social Coding – Forking a repository

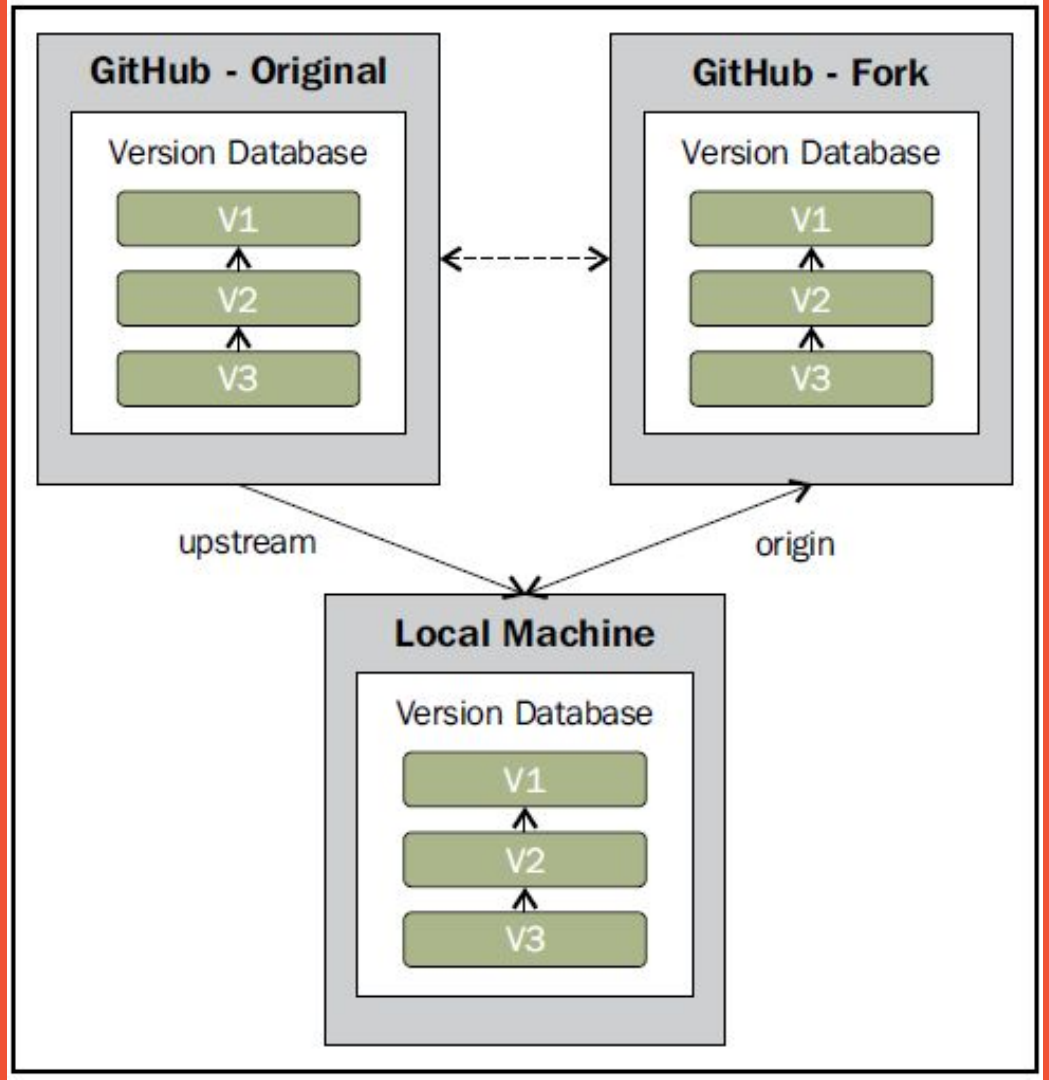
- First, remember that fork is not a Git feature, but a GitHub invention
- When you fork on GitHub, you get a server-side clone of the repository on your GitHub account

Social Coding – Forking a repository

- When you want to work on someone else's repository on which you don't have write access, you create fork of it
- This is mostly done with public projects, where community contributes

Social Coding – Forking a repository

- When you fork that means you are creating copy of someone else's repo on github into your github account.
- This happens on server side directly on github
- Then you can clone the repo from your account


GitHub - Original
Version Database
V1
V2
V3
GitHub - Fork
Version Database
V1
V2
V3
upstream
origin
Local Machine
Version Database
V1
V2
V3

Social Coding – Forking a repository

- After clone you can work on repo as it is your repo
- Make changes, commit push/pull all happens in your repo
- Only when you want to contribute your changes to original repo you can create pull request

Social Coding – Forking a repository

- A pull request is a way to tell the original author, "Hey! I did something interesting using your original code. Do you want to take a look and integrate my work, if you find it good enough?"

Social Coding – Forking a repository

- As you are not contributor to the project you can only submit your work to original repo with pull request
- Author of original repo has rights to accept or reject or you changes

Demo

Deleting Branches

- When you are done with a branch and it is no longer needed then you can delete the branch
  - git branch -d contact-form
- Deleting remote branch, add “r” flag
  - git branch -dr origin/contact-form

Demo

Undoing Local Changes

- If you have local changes that are not committed and want discard change then you can use following commands
- Discard changes in single file
  - git checkout HEAD <file/to/restore>
- Discard all changes that are not committed (already learned previously)
  - git reset --hard HEAD

Demo

Undoing Committed Changes

Undoing Committed Changes

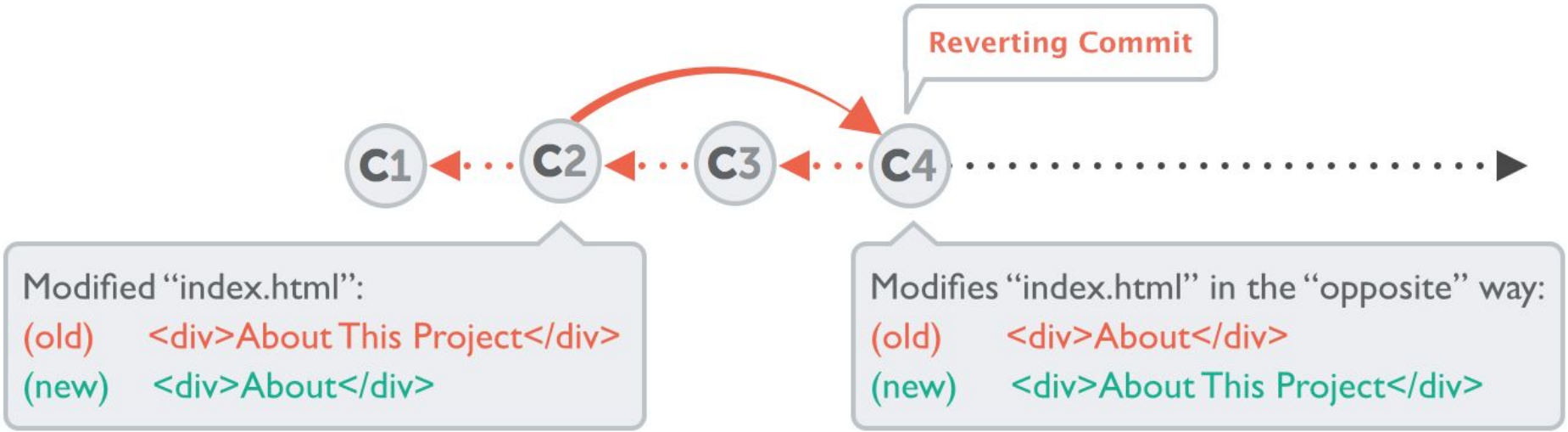
- Once code is committed and you feel your changes were wrong and you want to undo commits
- You can undo committed changes using following commands:
  - git revert <commitHash>
  - git reset --hard <commitHash>

git revert <commitHash>

- This command does not actually delete any commit.
- Instead it reverts the effects of a certain commit, effectively undoing it.
- It does this by producing a new commit with changes that revert each of the changes in that unwanted commit.
- For example, if your original commit added a word in a certain place, the reverting commit will remove exactly this word, again.

git revert <commitHash>

- For example: git revert 2b504be



git reset --hard <commitHash>

- It neither produces any new commits nor does it delete any old ones.
- It works by resetting your current HEAD branch to an older revision (also called “rolling back” to that older revision):
- If you call it with “--keep” instead of “--hard”, all changes from rolled back revisions will be preserved as local changes in your working directory.

git reset --hard <commitHash>

- For example: git reset --hard 2be18d9
- Preserved as local changes : git reset --keep 2be18d9