Git and GitHub

2024.03.07

SWPP Practice Session

Seunghyeon Nam (with lots of derived works)

Git and GitHub

- Git is a VCS *version control system*
- GitHub is a web service that hosts git repositories + @
  - Helps collaborative development
    - Issue tracker, pull request, access control, wiki, …
  - Offers automated testing and distribution environment
    - Actions

Git Workspace Structure

Working Directory

- Files (on disk)
  - Including unchanged/unfinished works

Staging Area

- Staged files
  - Changes ready for commit

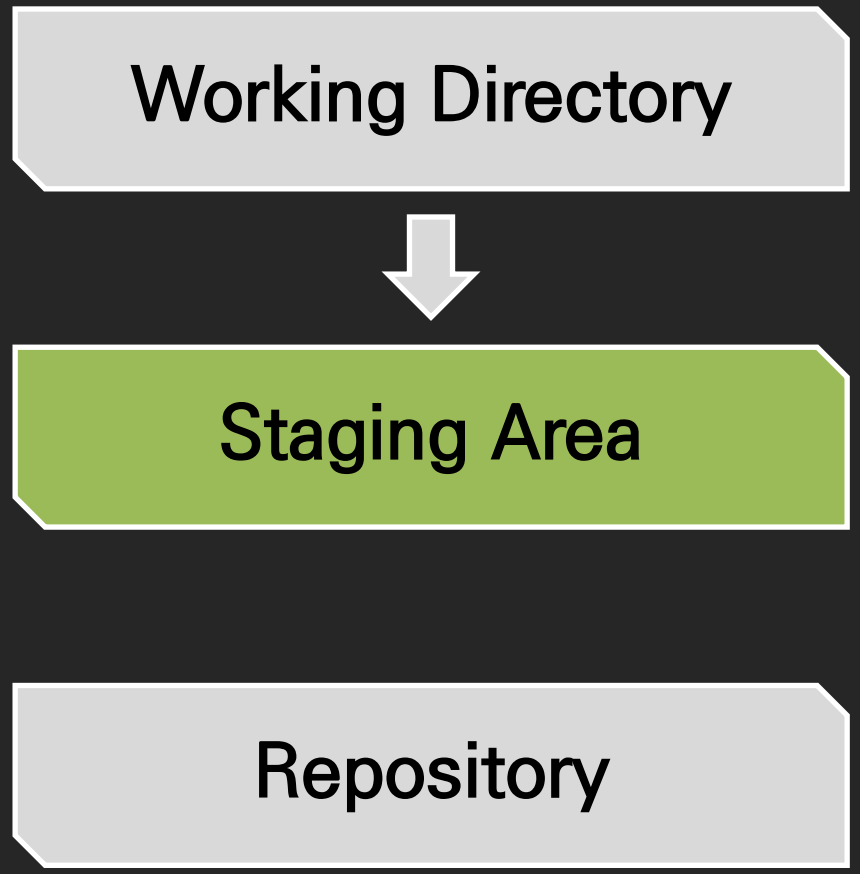
Repository

- Commits
  - Unit of implementation

Git Workspace Setup

- `git init <directory>`
- Creates a local repository of directory name
- Omit the directory name to create one in current directory
  - Initially, all the files are untracked

Staging File



- `git add <file>`
- Marks the file as tracked

  (if it was marked as untracked)
- Caches the changes so far

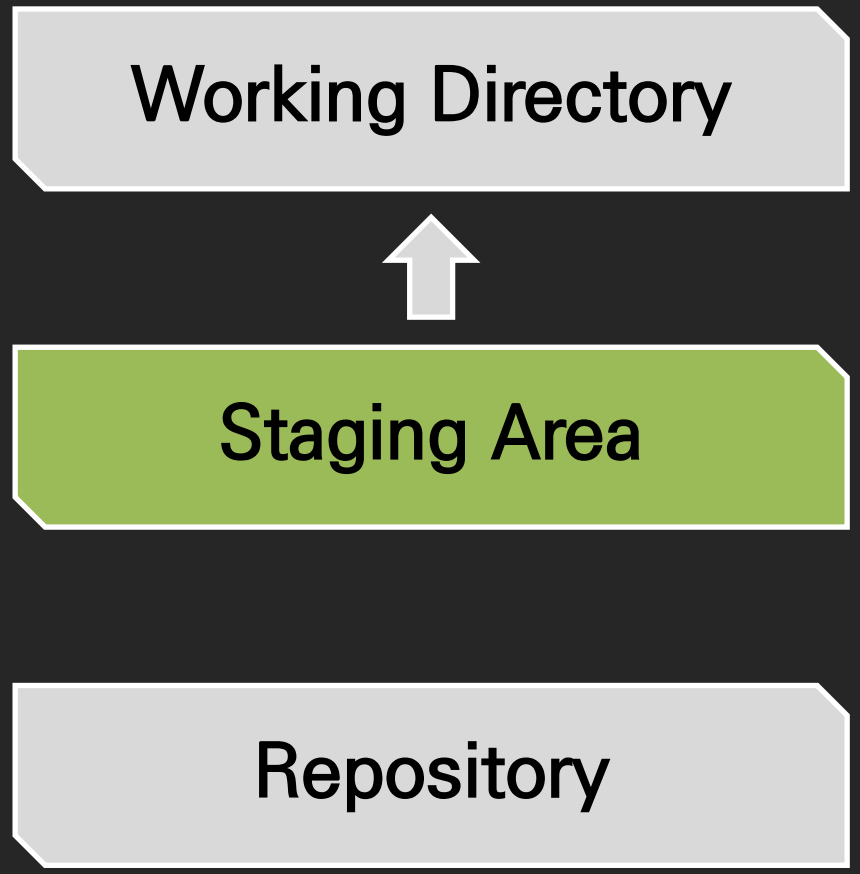
Checking File Status

- `git status`
- Files are categorized into:
  - Untracked files
  - Tracked files with unstaged changes
  - Tracked files with staged changes

.gitignore

- Not a command, but a file!
- List of files/extensions/folders/etc to ignore
- Does not show up in `git status`

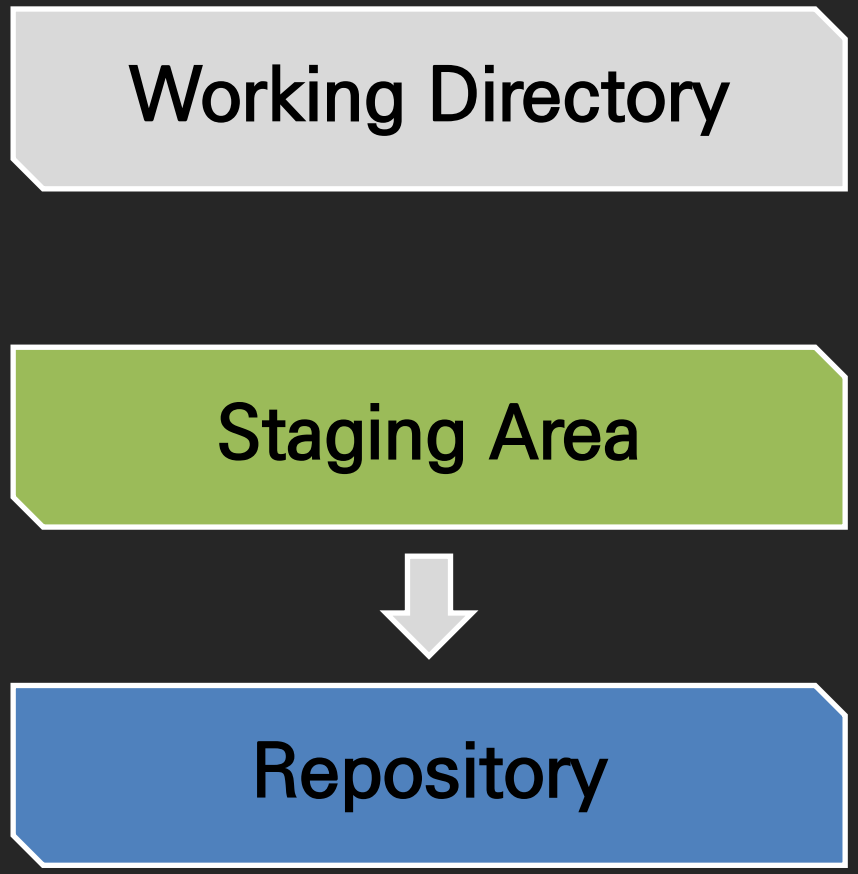
Unstaging File



- `git restore --staged <file>`
- Discards cached changes

- `git rm --cached <file>`
- Discards cached changes
- And marks the file as untracked

Committing Changes

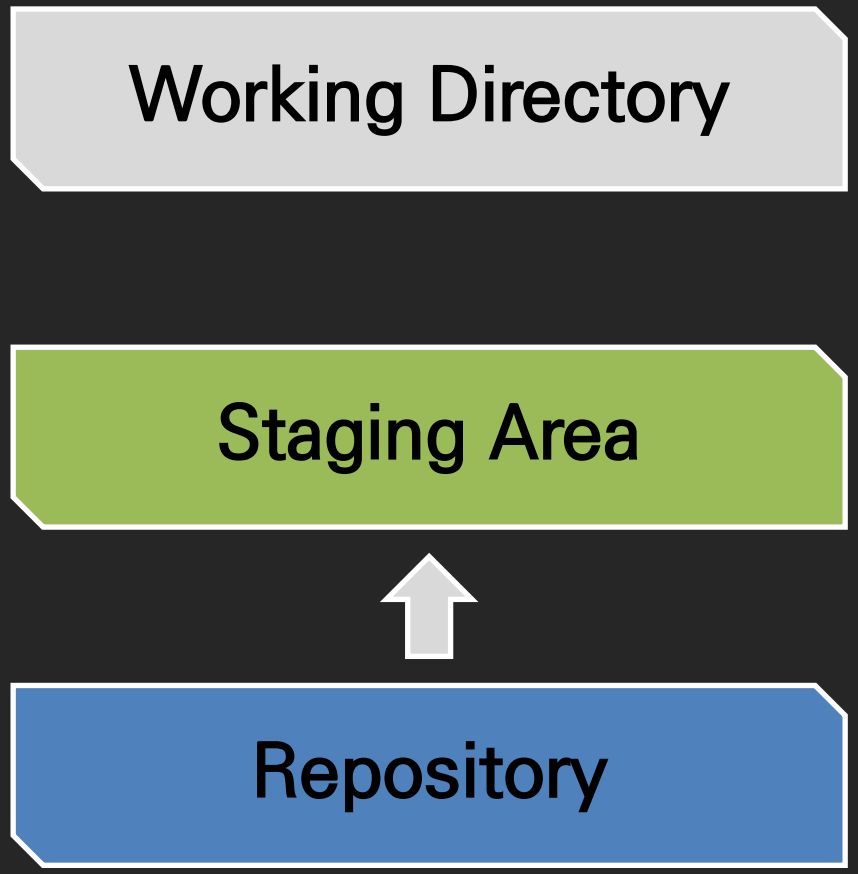


- `git commit -m “commit message”`
- Push staged changes into repo with given message
- Commit message is mandatory!
- Add `--amend` to append changes into the last commit

Checking Commit History

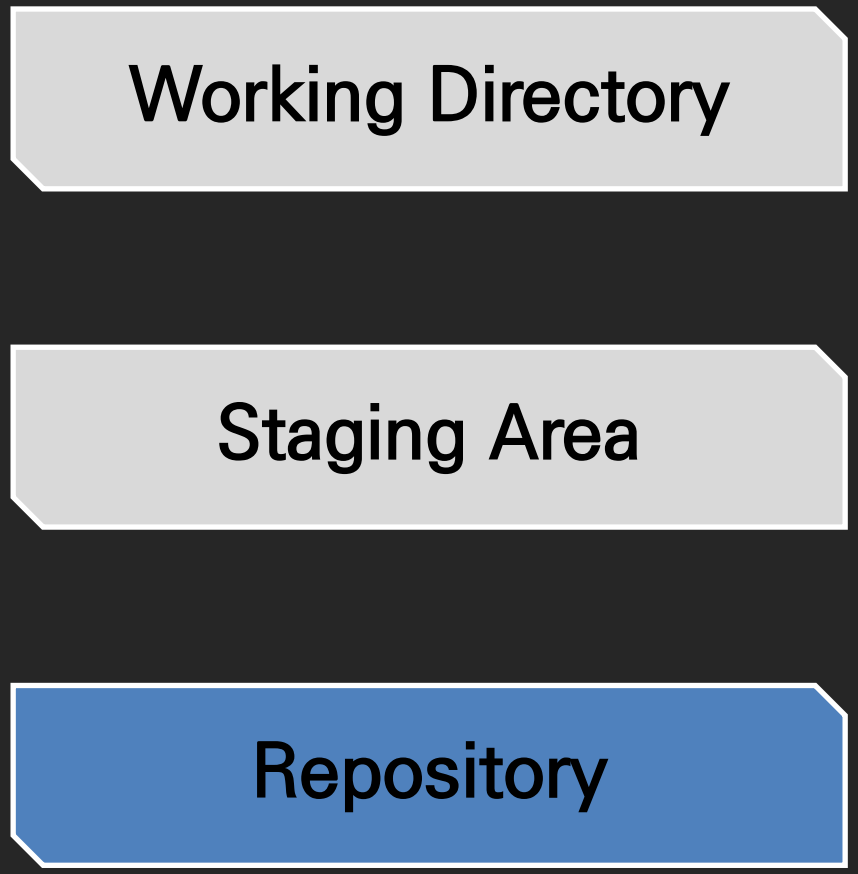
- `git log`
- View history of previous commits with timestamp and commit hash
- Each commit can be identified using commit hash
- Press q to exit

Undoing Commit



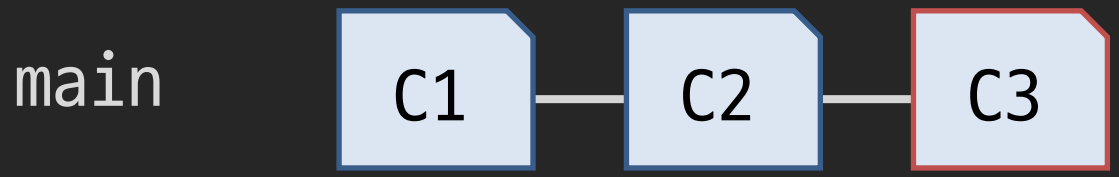
- `git reset --soft HEAD~1`
- Remove last commit from repo and restore changes into staging area
- Increase number after `HEAD~` to remove and restore more commits

Creating Branch

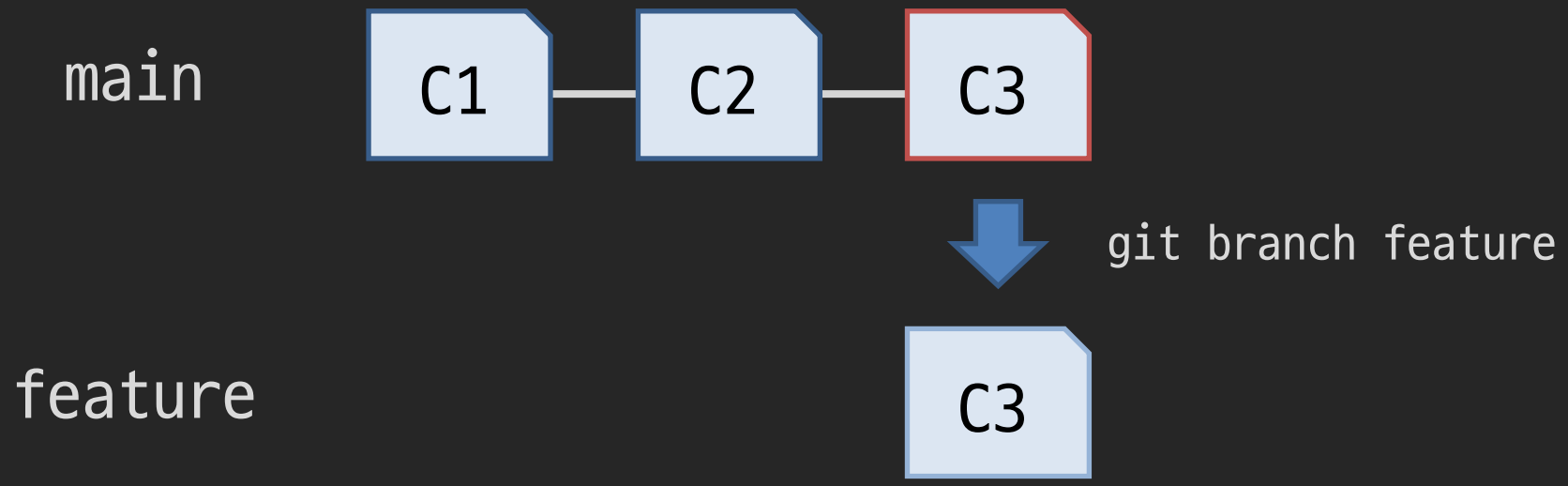


- `git branch <name>`
- Create a branch
  using current commit as parent
- Branches of duplicate names
  are not allowed

Creating Branch



Creating Branch



Managing Branch

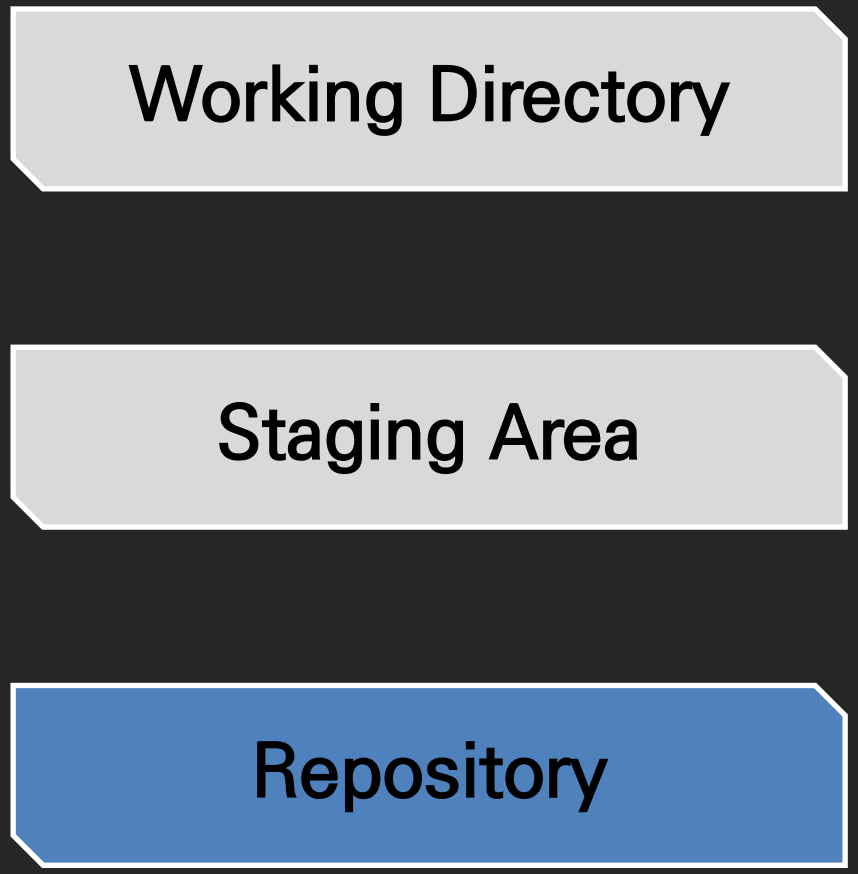
Working Directory

Staging Area

Repository

- `git branch`
- List the branches in the repo
- Press q to exit

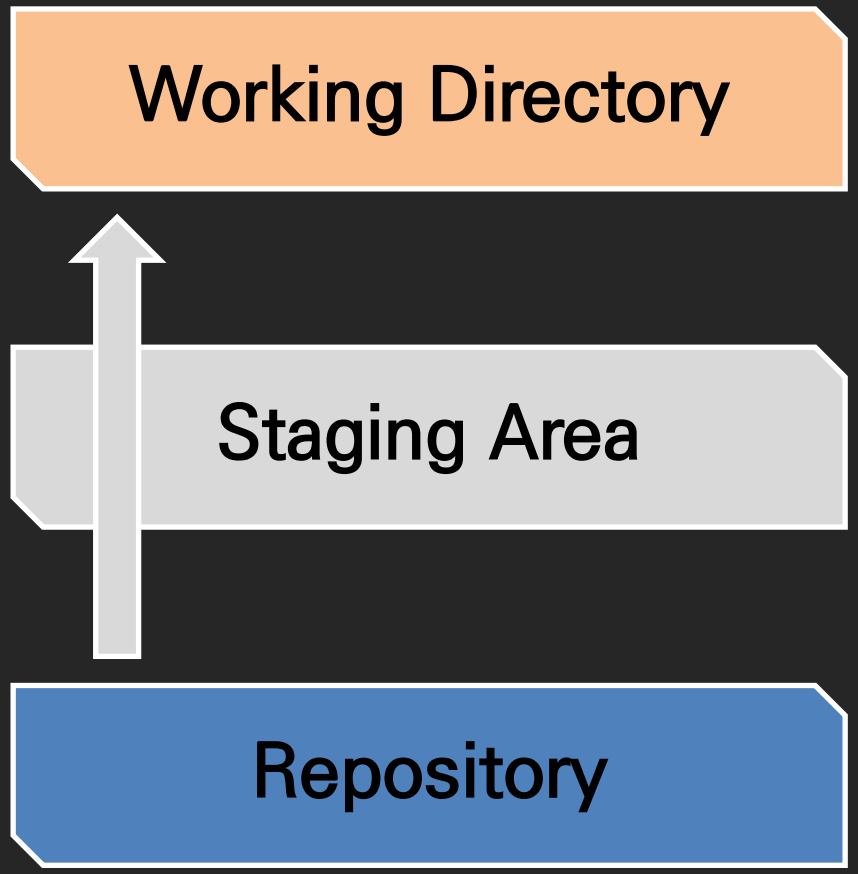
Managing Branch



- `git branch -D <branch>`
- Delete the branch

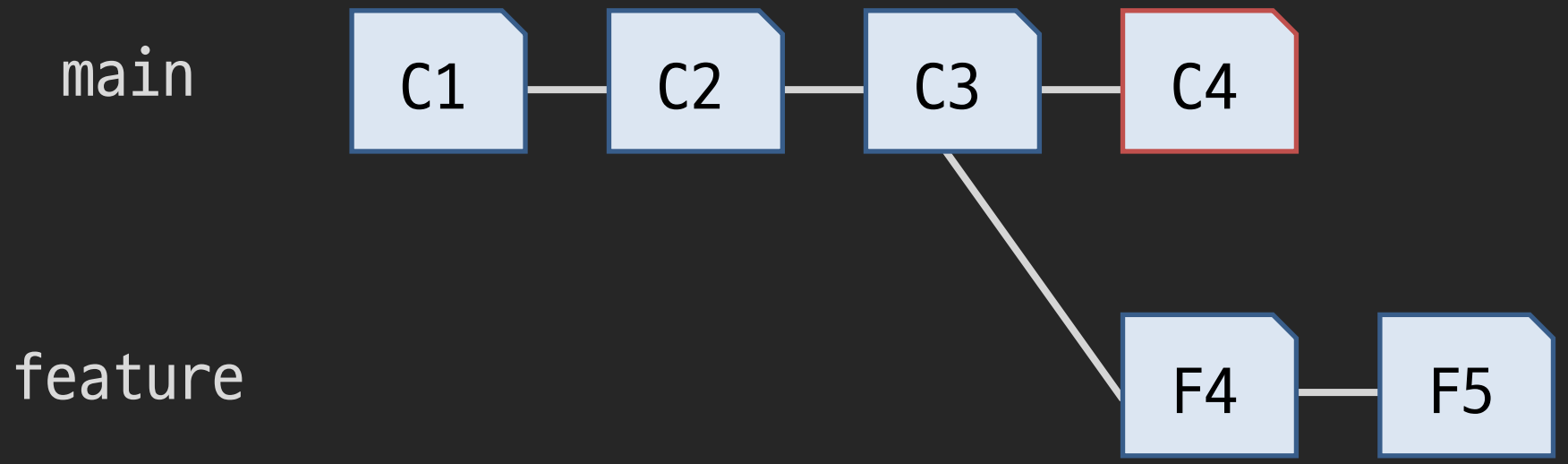
- `git branch -m <old> <new>`
- Rename the branch `old` to `new`

Switching Branch

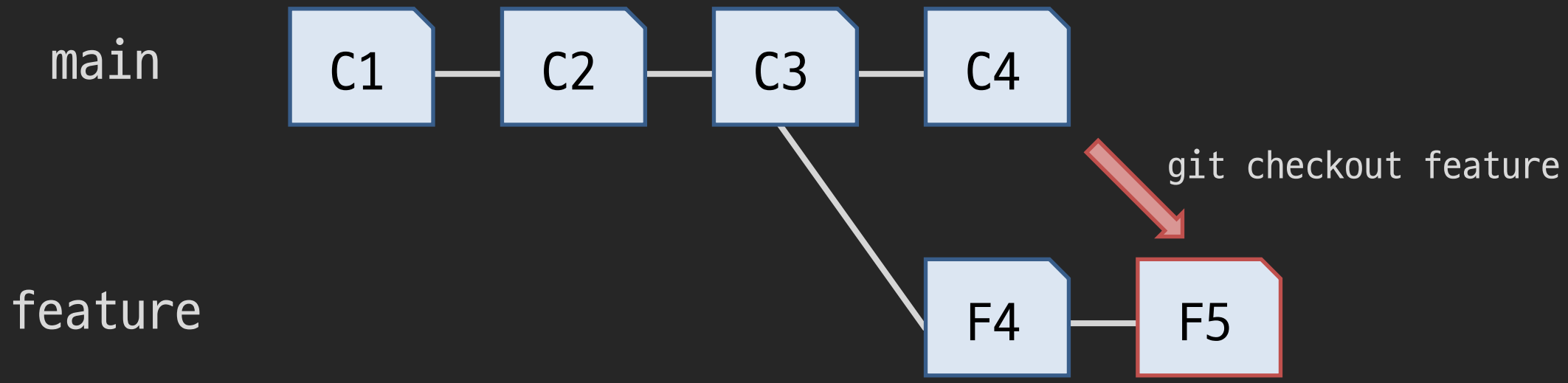


- `git checkout <branch>`
- Change to context of target branch
- Update working directory
- Staging area must be clean
  - Checkout will be aborted otherwise
  - Commit, unstage, or *stash*

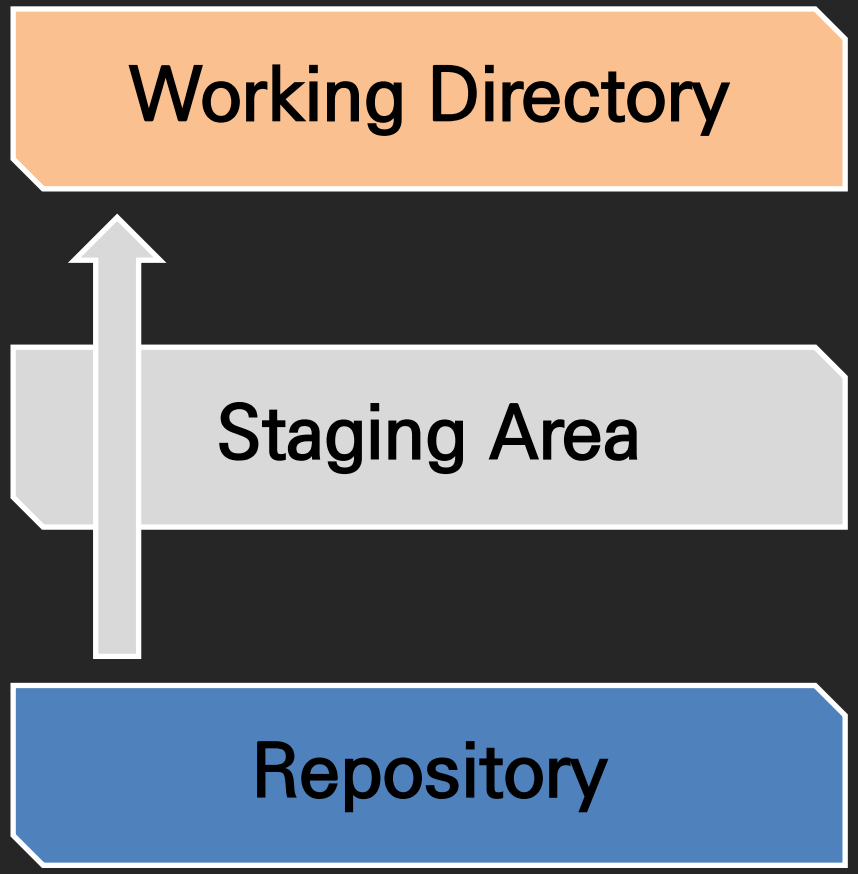
Switching Branch



Switching Branch

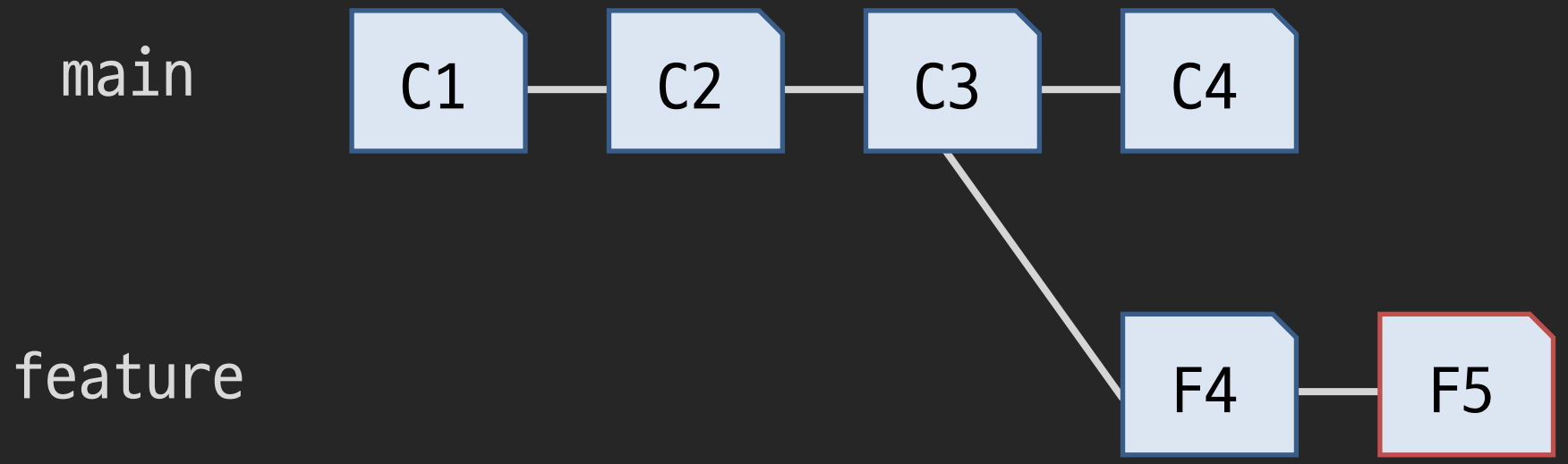


Applying Changes From Other Branch

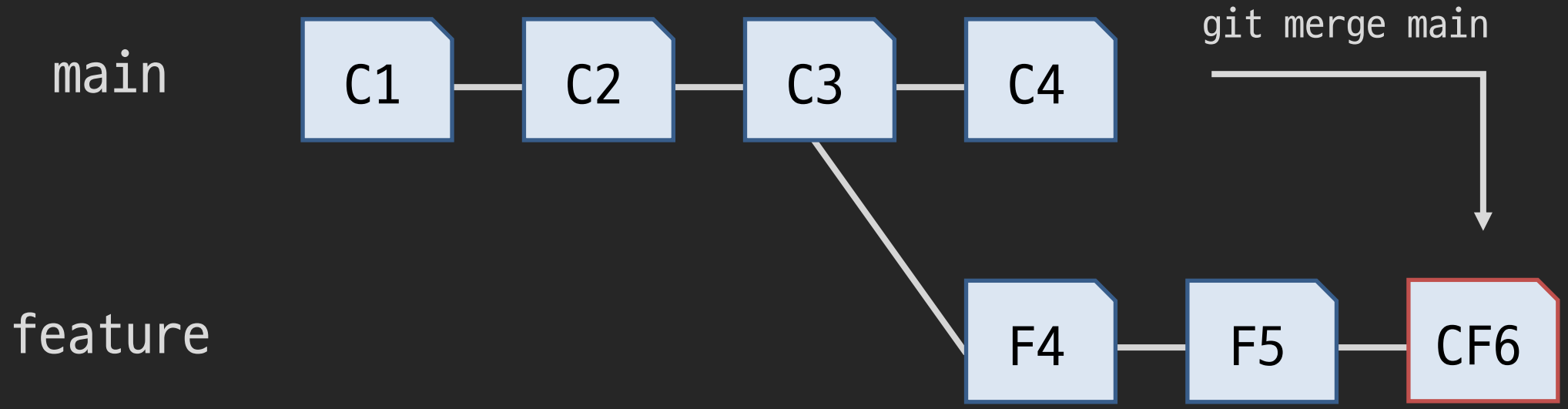


- `git merge <branch>`
- Collapse the changes in the other branch since the last shared commit into a merge commit
- Merge conflict may occur, which should be resolved manually

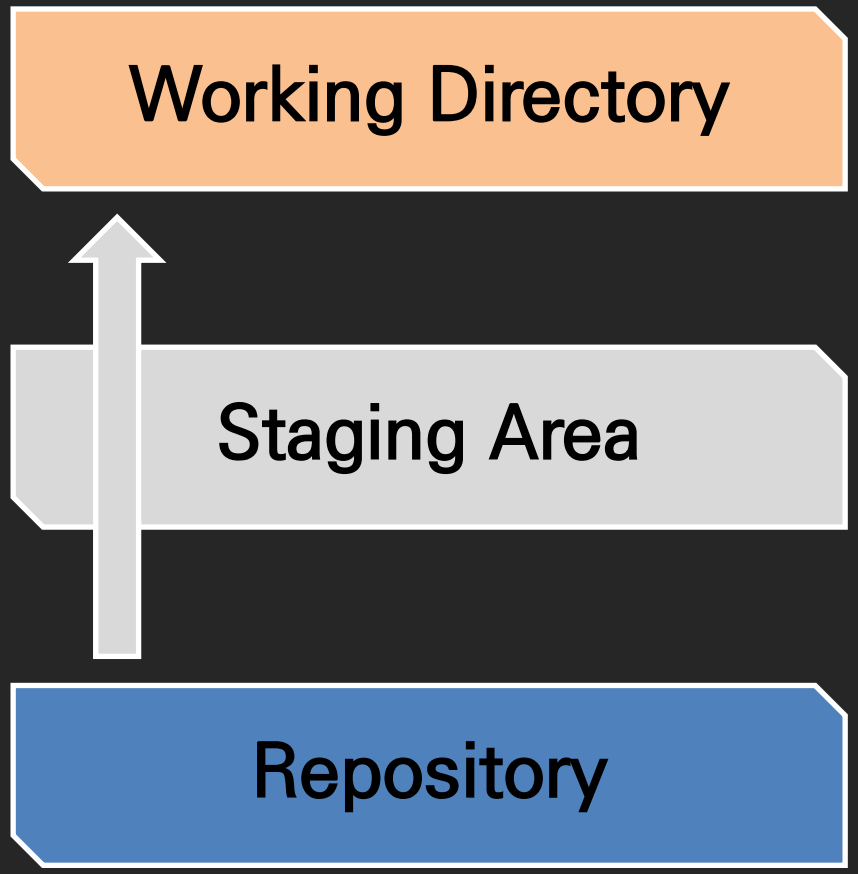
Applying Changes From Other Branch



Applying Changes From Other Branch

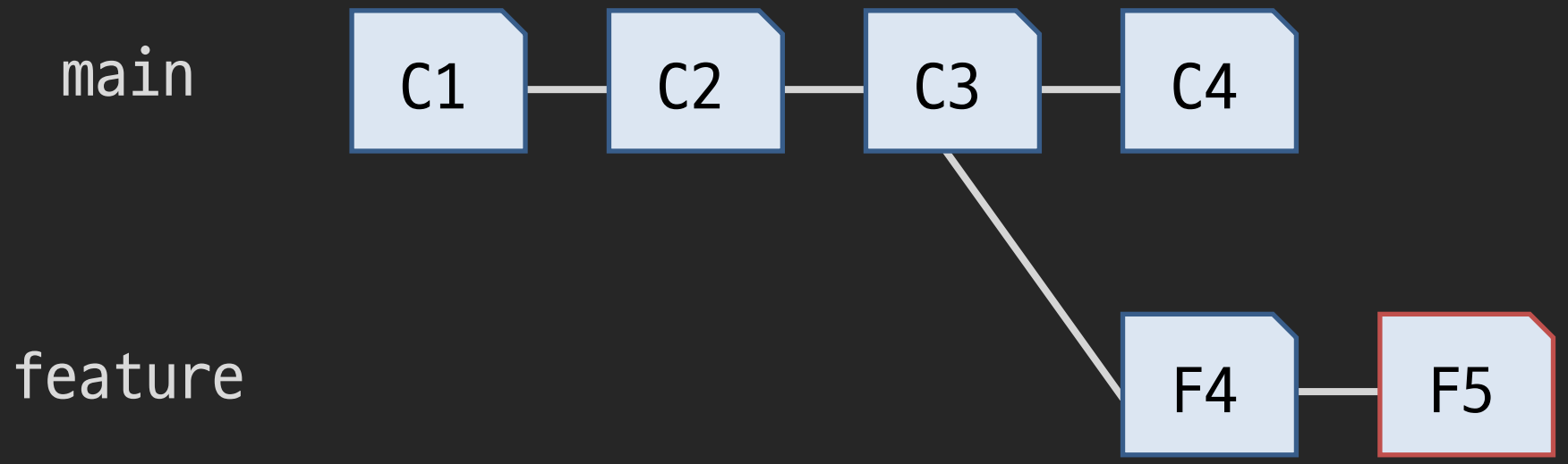


Changing Parent to Other Branch

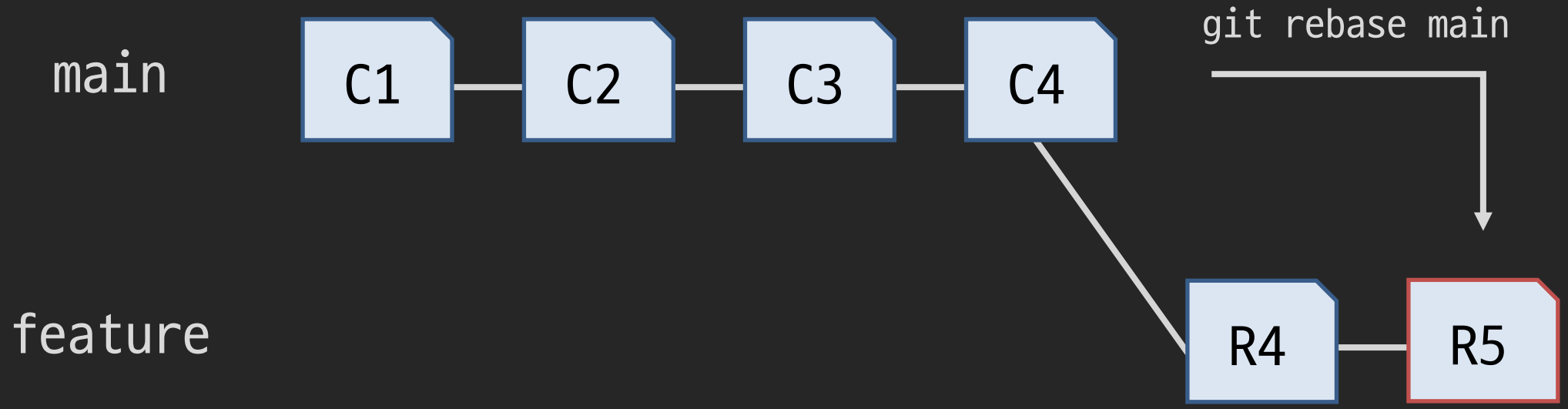


- `git rebase <branch>`
- Change the parent of current branch to the other branch's commit
- Merge conflict may occur, which should be resolved manually

Changing Parent to Other Branch



Changing Parent to Other Branch



Merge or Rebase?

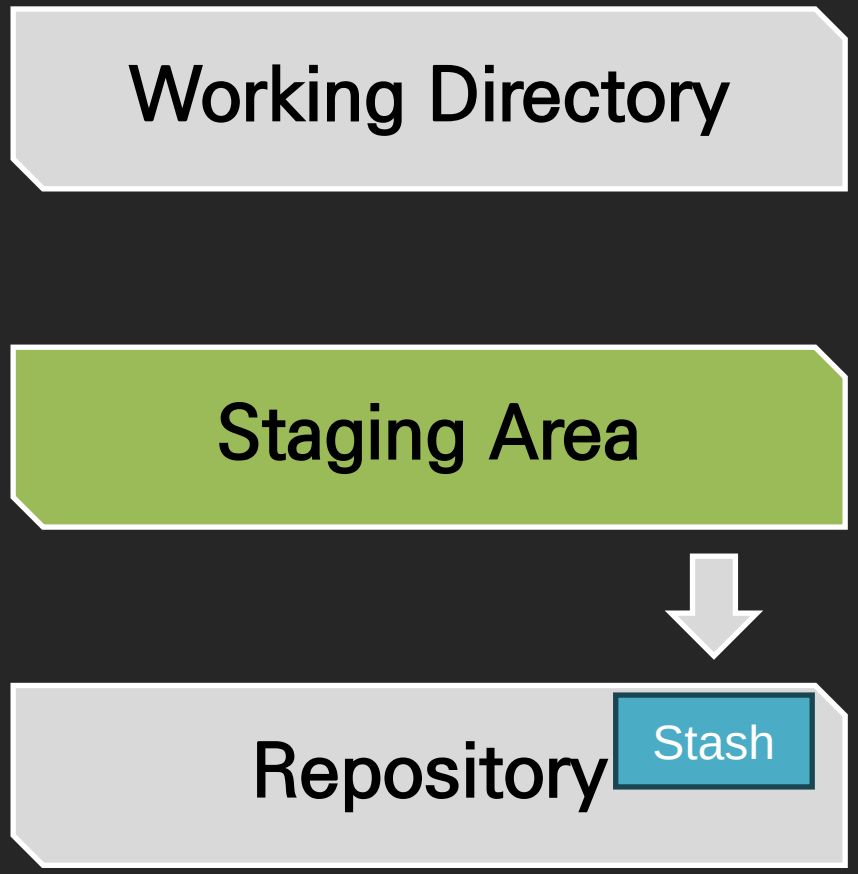
Merge

- Branch has two parents
  - Complicates commit history
- Preserves commit history
- Updating published branch

Rebase

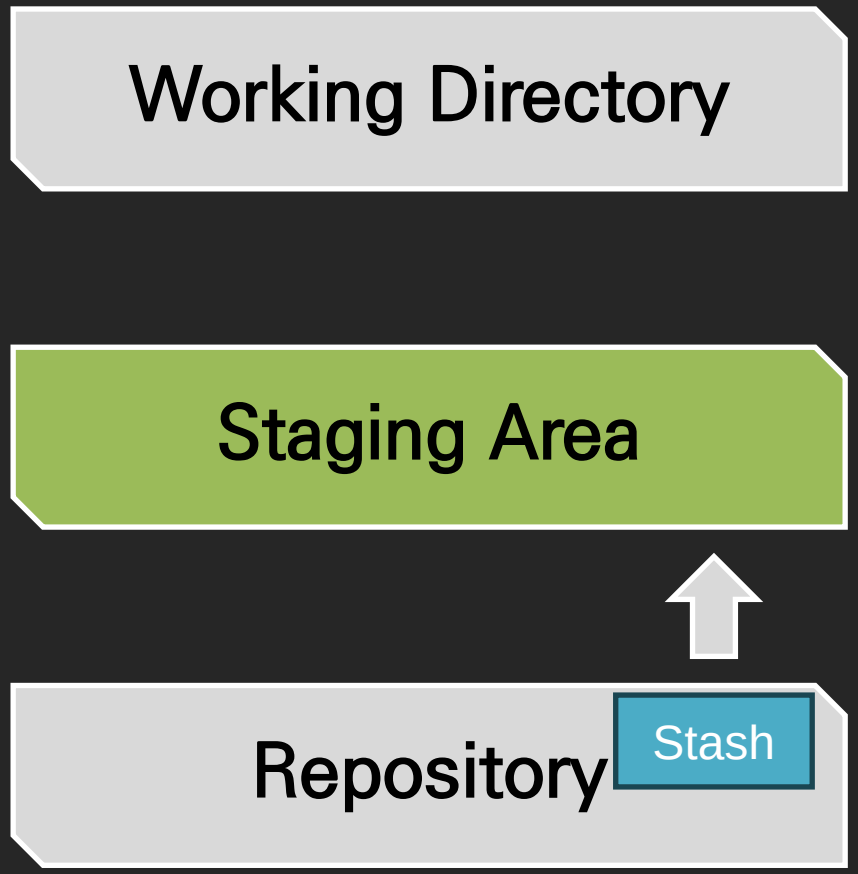
- Branch has one parent
  - Simplifies commit history
- Rewrites commit history
  - Breaks sync with collaborator
  - Never use on published branch!
- Updating local branch

Stashing Undone Works



- `git stash`
- Caches changes in all tracked files as a stash
- Adds the stash into stash list
- Discards all changes in working directory and staging area

Restoring Stashed Works



- `git stash apply`
- Restores last stashed changes

- `git stash pop`
- Restores last stashed changes
- And removes that stash from list

Remote Repository

- Hosts a repository online
- Fundamentals of collaborative development using git
- May be public, under restricted access, or private
- GitHub is just one way of hosting remote repositories

Downloading from Remote Repository

- `git clone <repository>`
- Downloads entire repository
- Repository can be really big if it has long commit history
  - Add `--depth N` to restrict the number of commits to clone

Managing Remote Repository

- `git remote add <name> <repository>`
- Connect to remote repository

- `git remote remove <name>`
- Disconnect from remote repository

Interacting with Remote Repository

- `git fetch`
- Updates remote changes

- `git pull`
- Updates remote changes
- And apply the changes to working directory

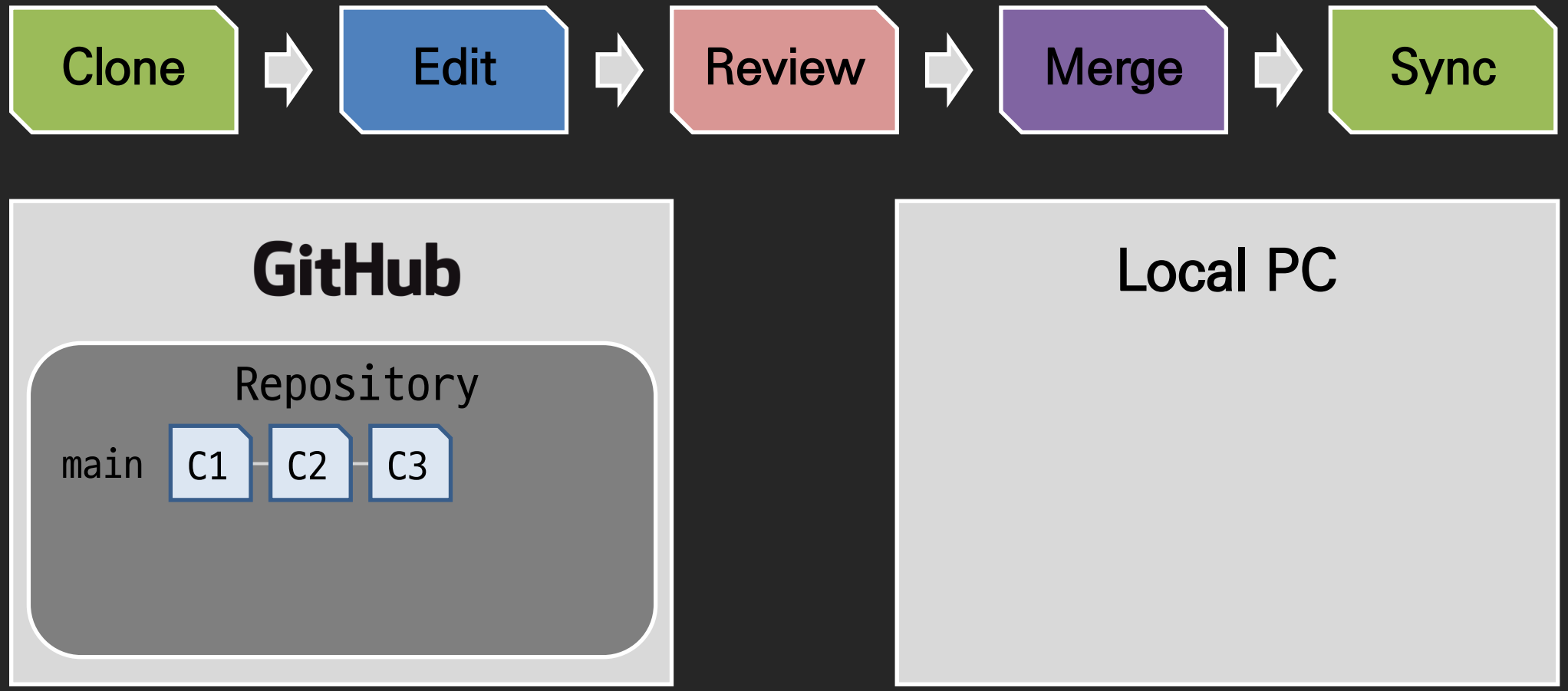
- `git push`
- Upload local changes
- Fails if local changes diverged from remote
  - Don’t force push!
  - It may overwrite others’ works
  - Rebase or merge instead

Branch and Remote Repository

- `git checkout -t <remote branch>`
- Copy remote branch into local repository

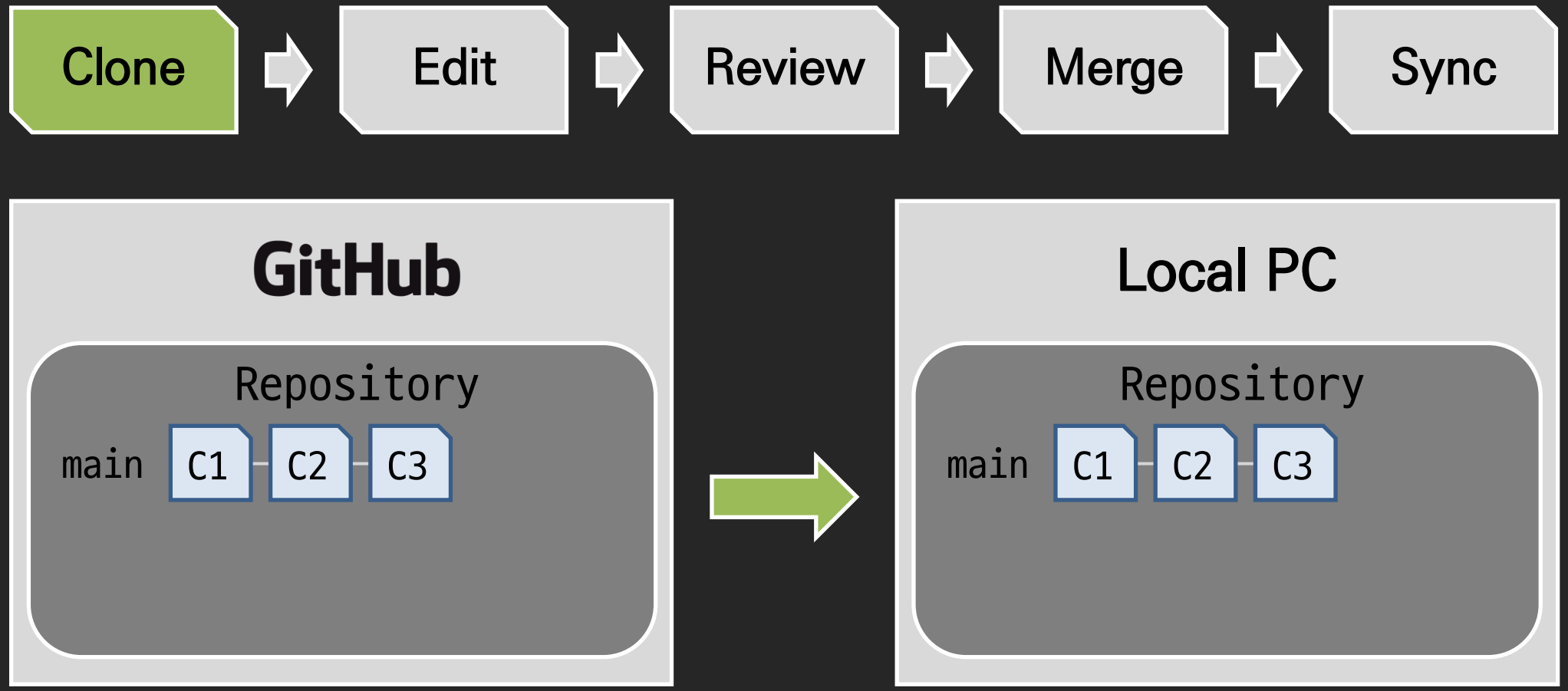
- `git push <remote> <branch>`
- Upload local branch to remote repository

Collaborative Development Steps

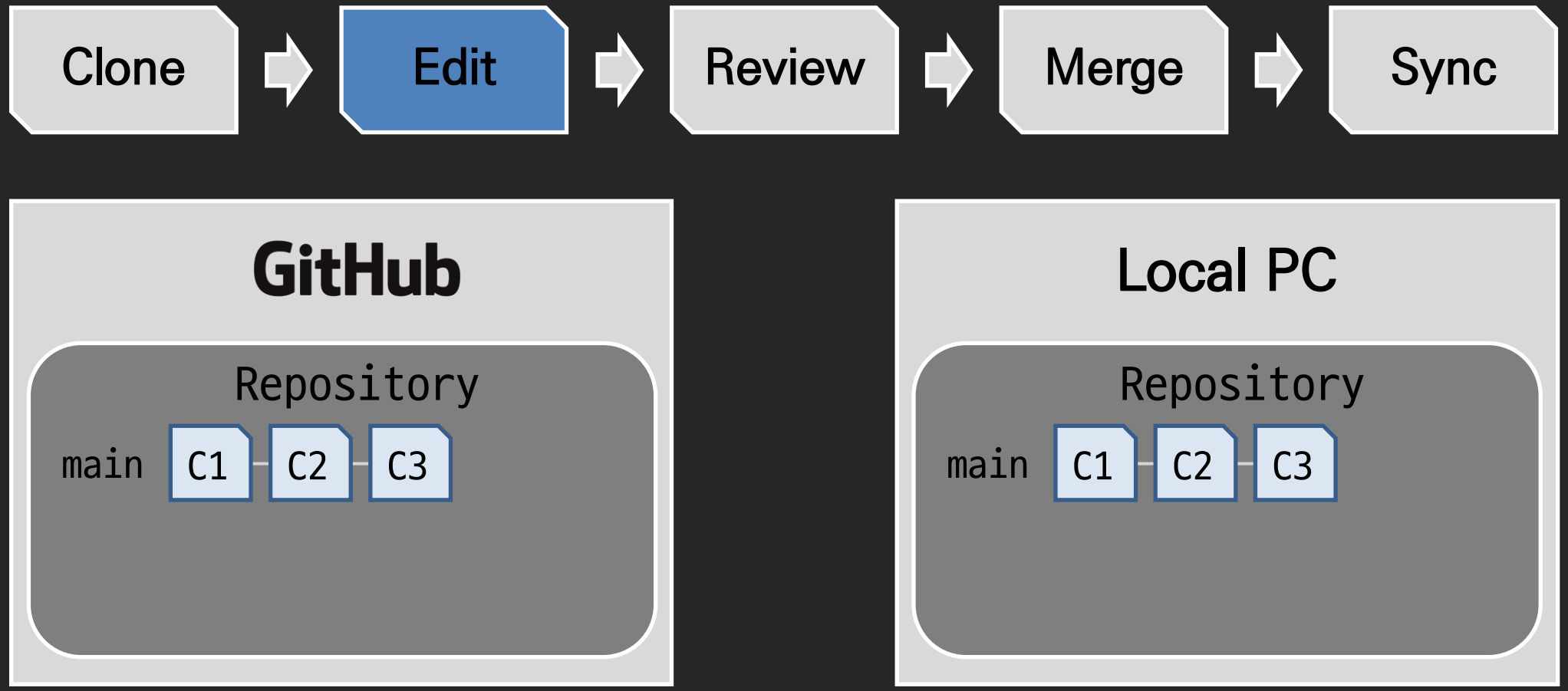


Collaborative Development Steps

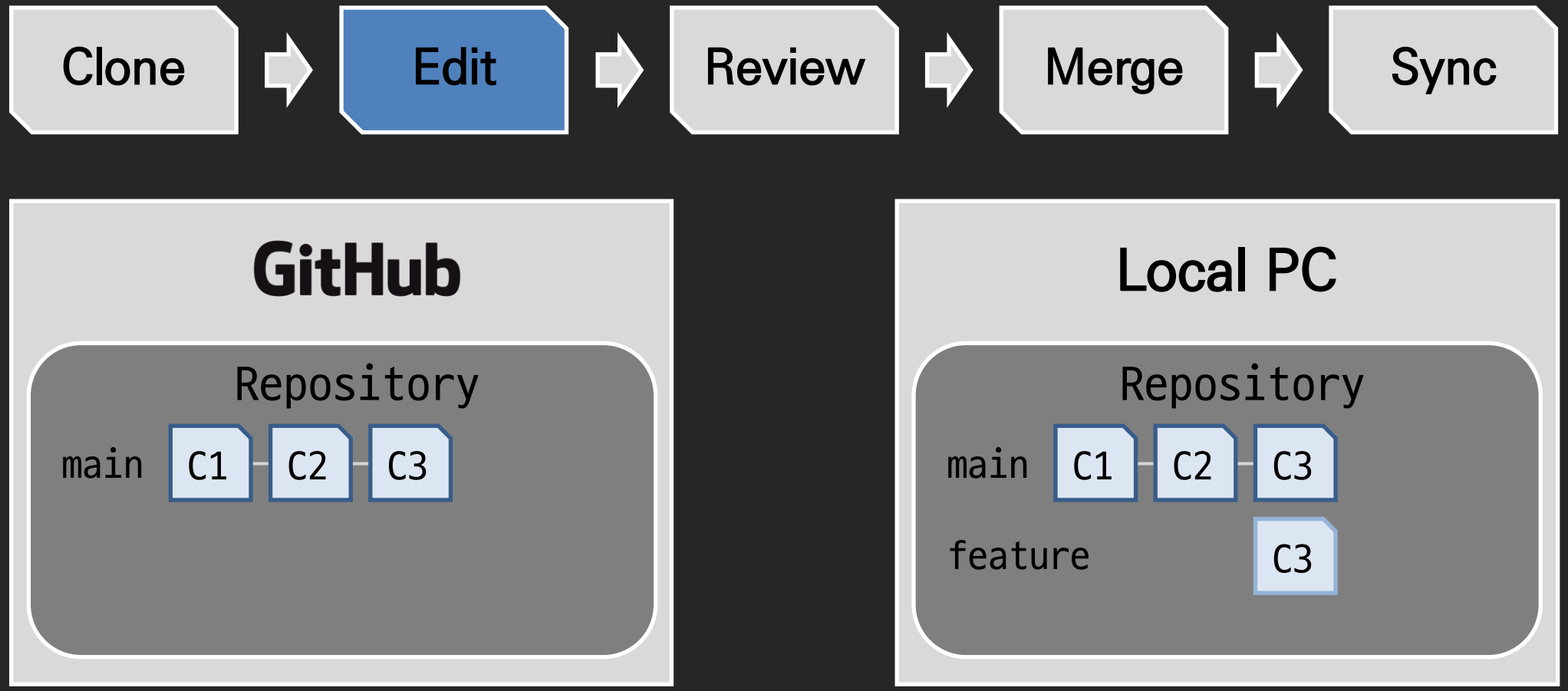
Collaborative Development Steps



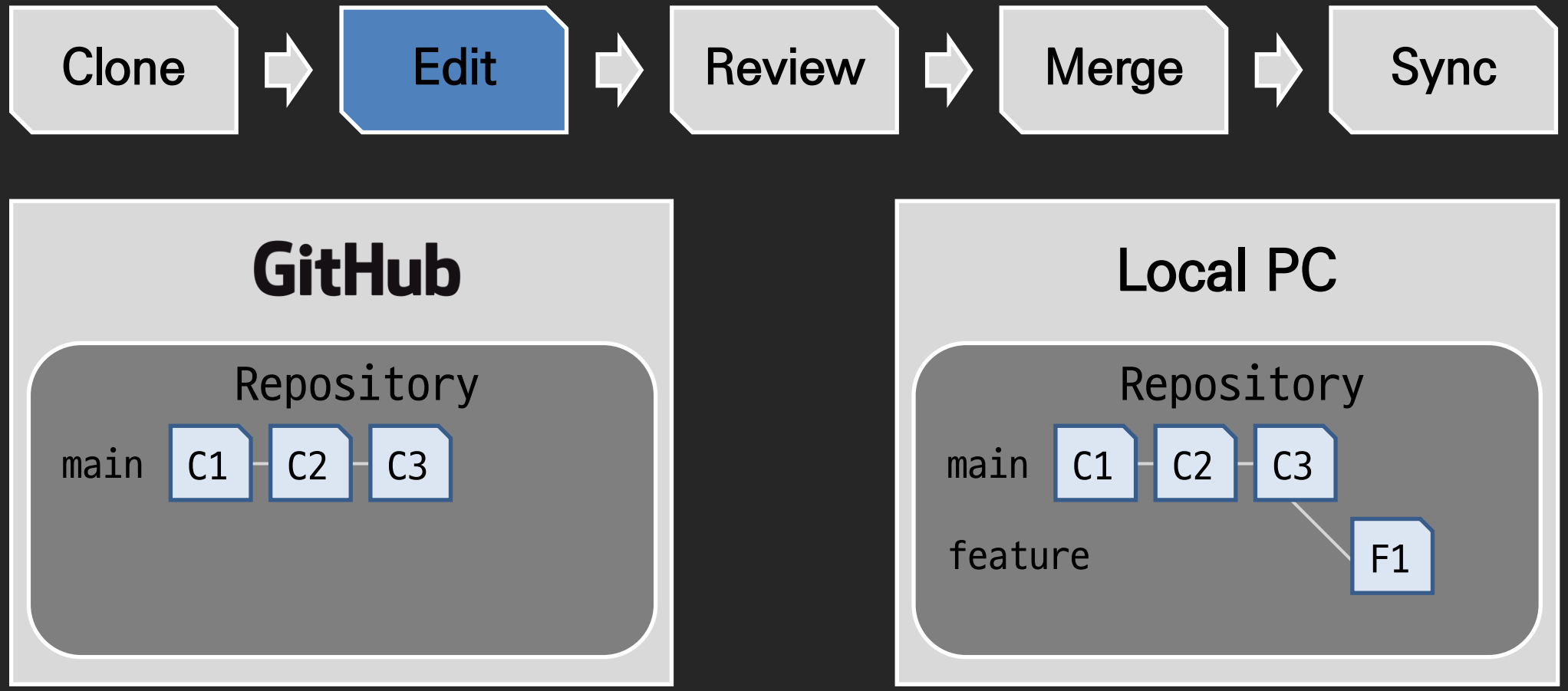
Collaborative Development Steps



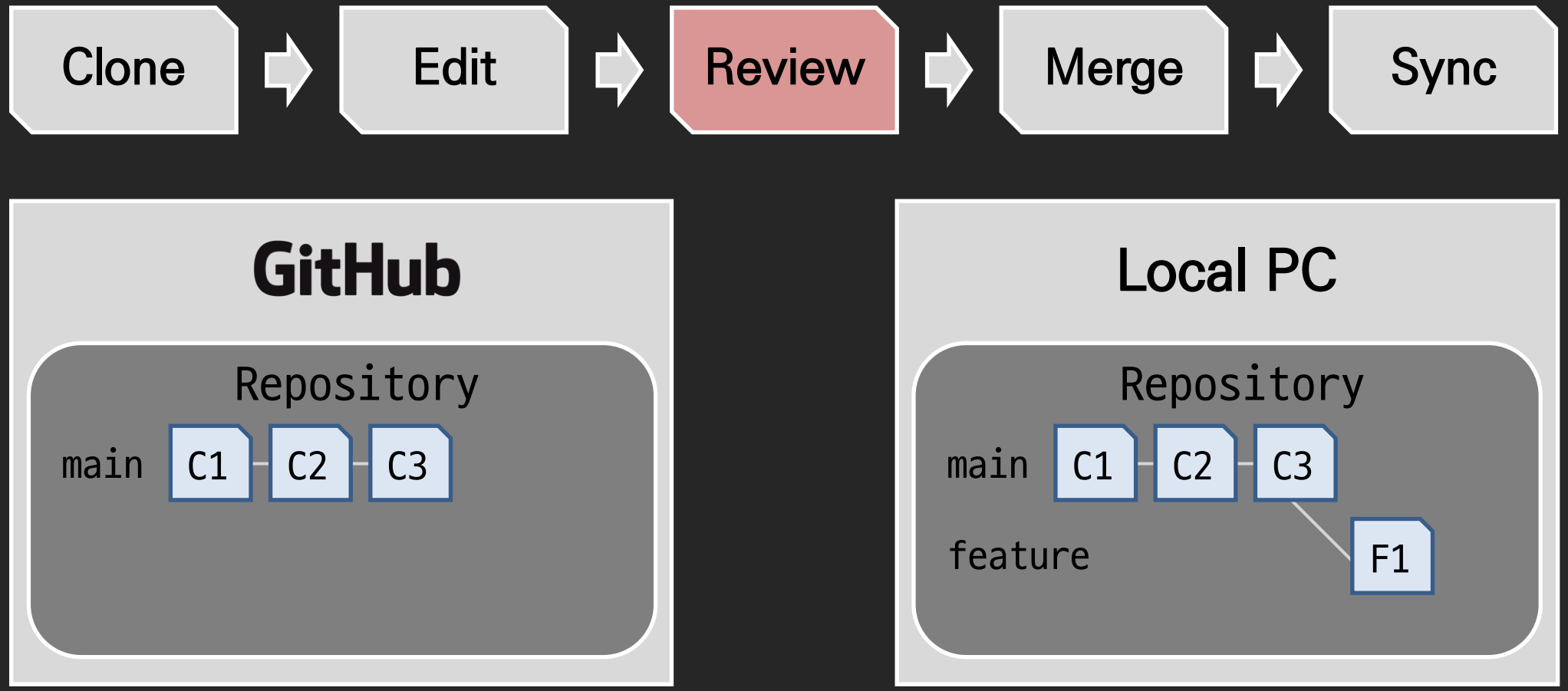
Collaborative Development Steps



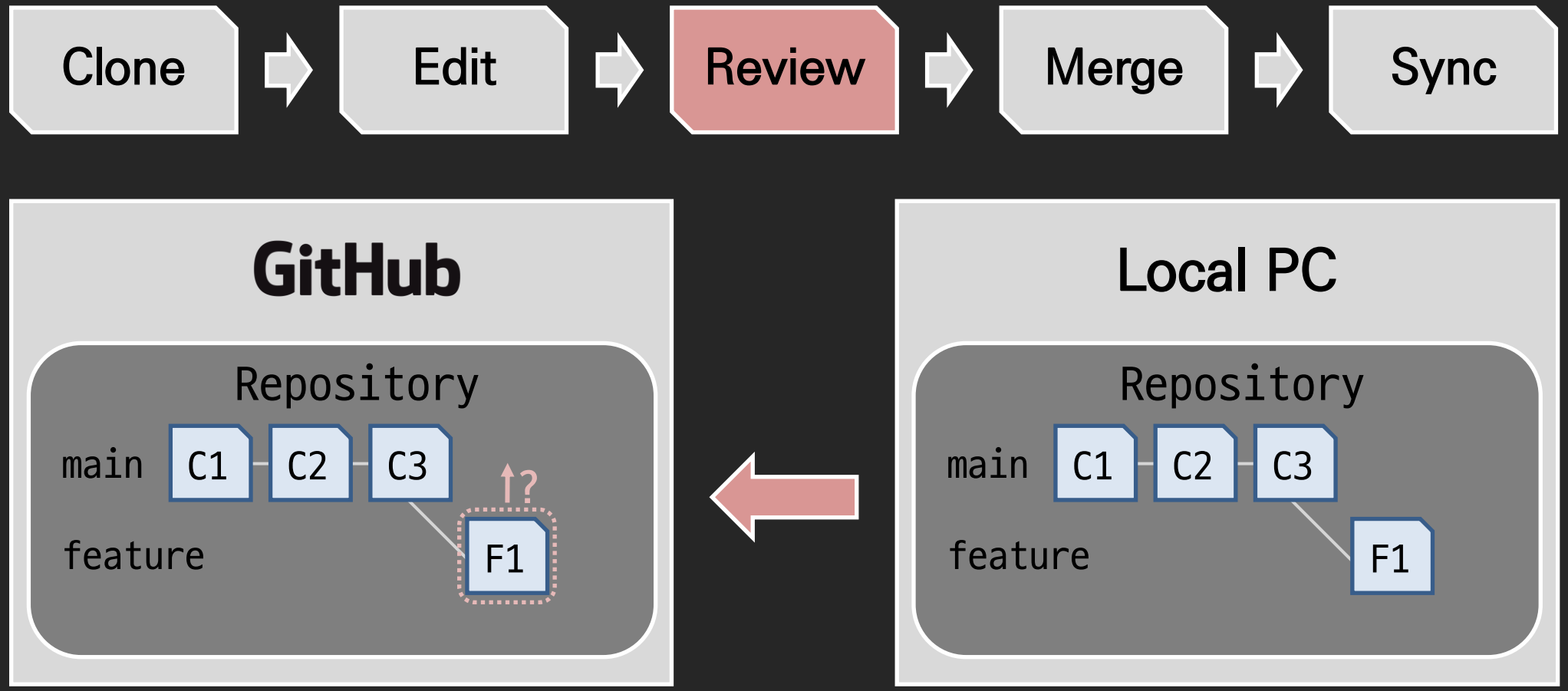
Collaborative Development Steps



Collaborative Development Steps



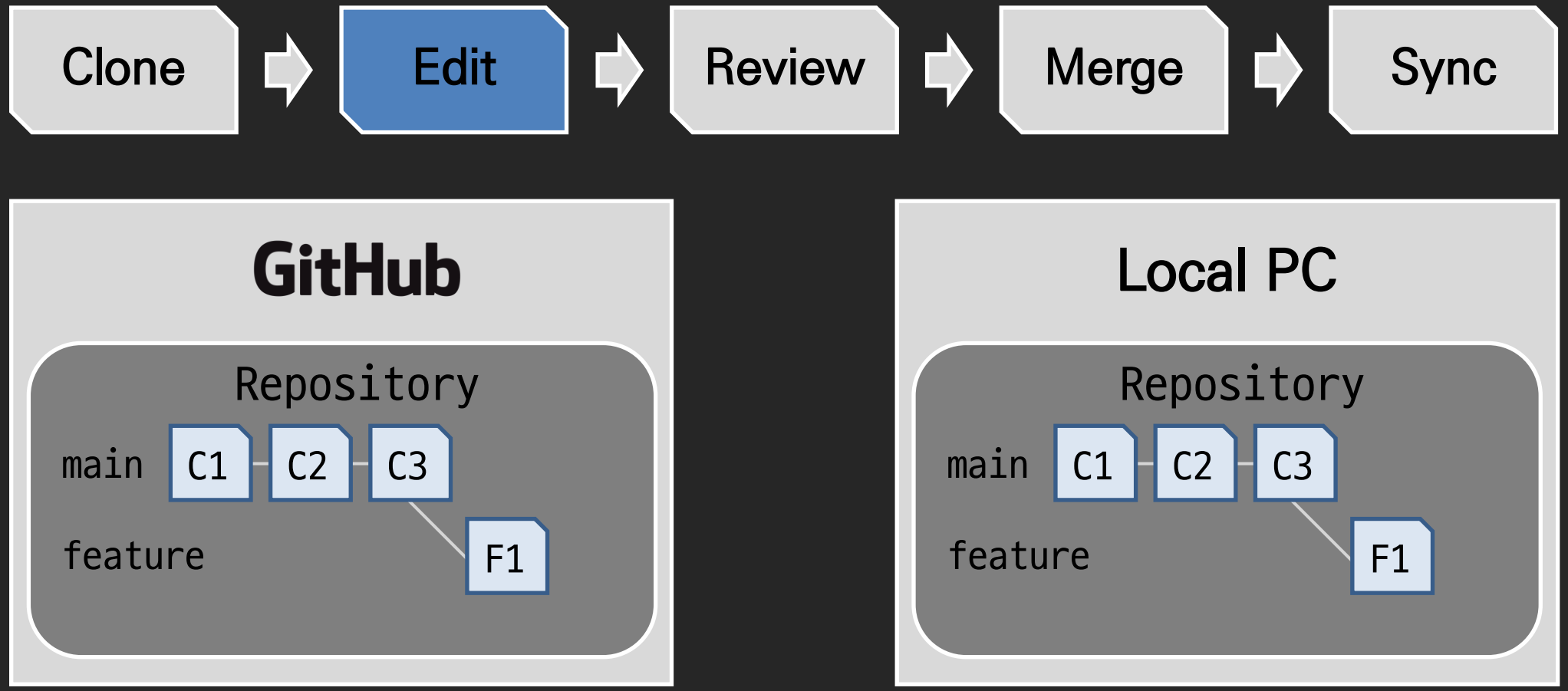
Collaborative Development Steps



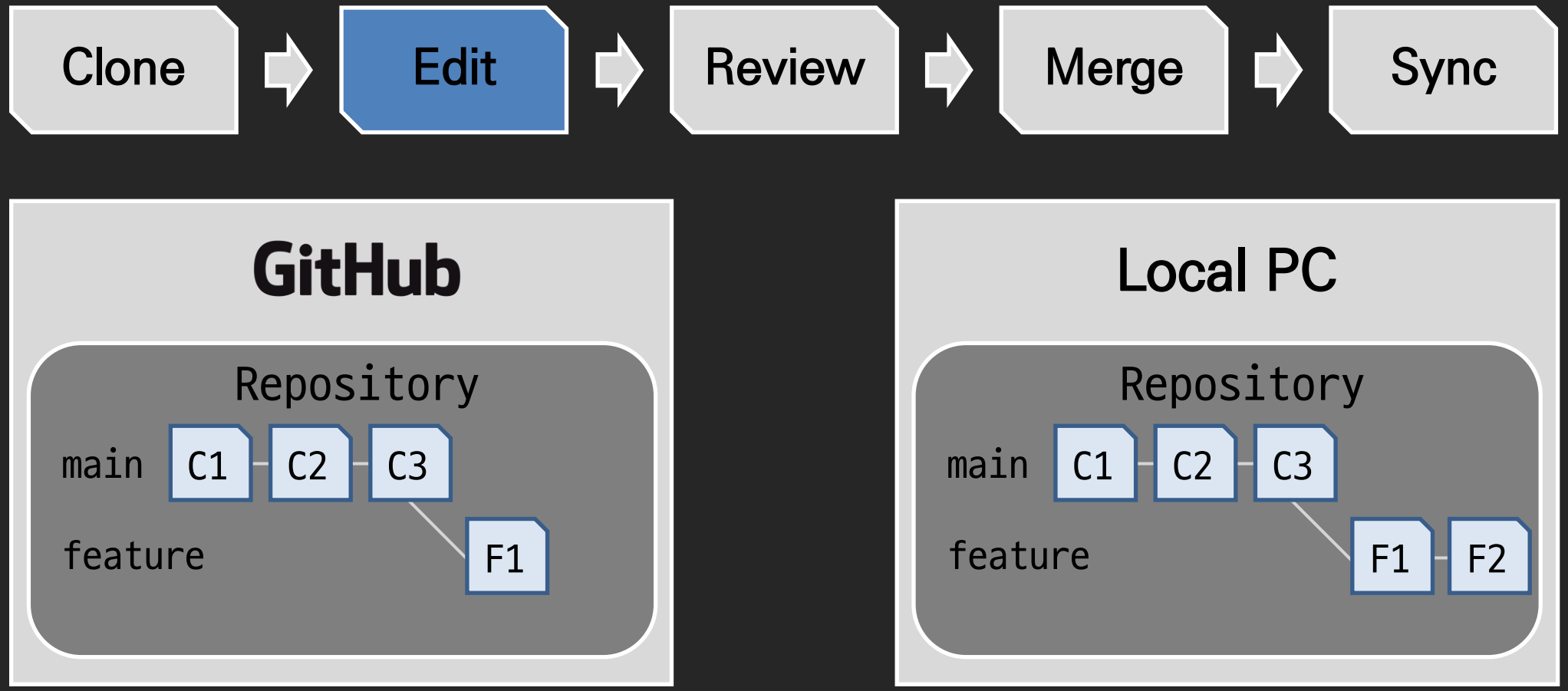
Collaborative Development Steps

Clone → Edit → Review → Merge → Sync

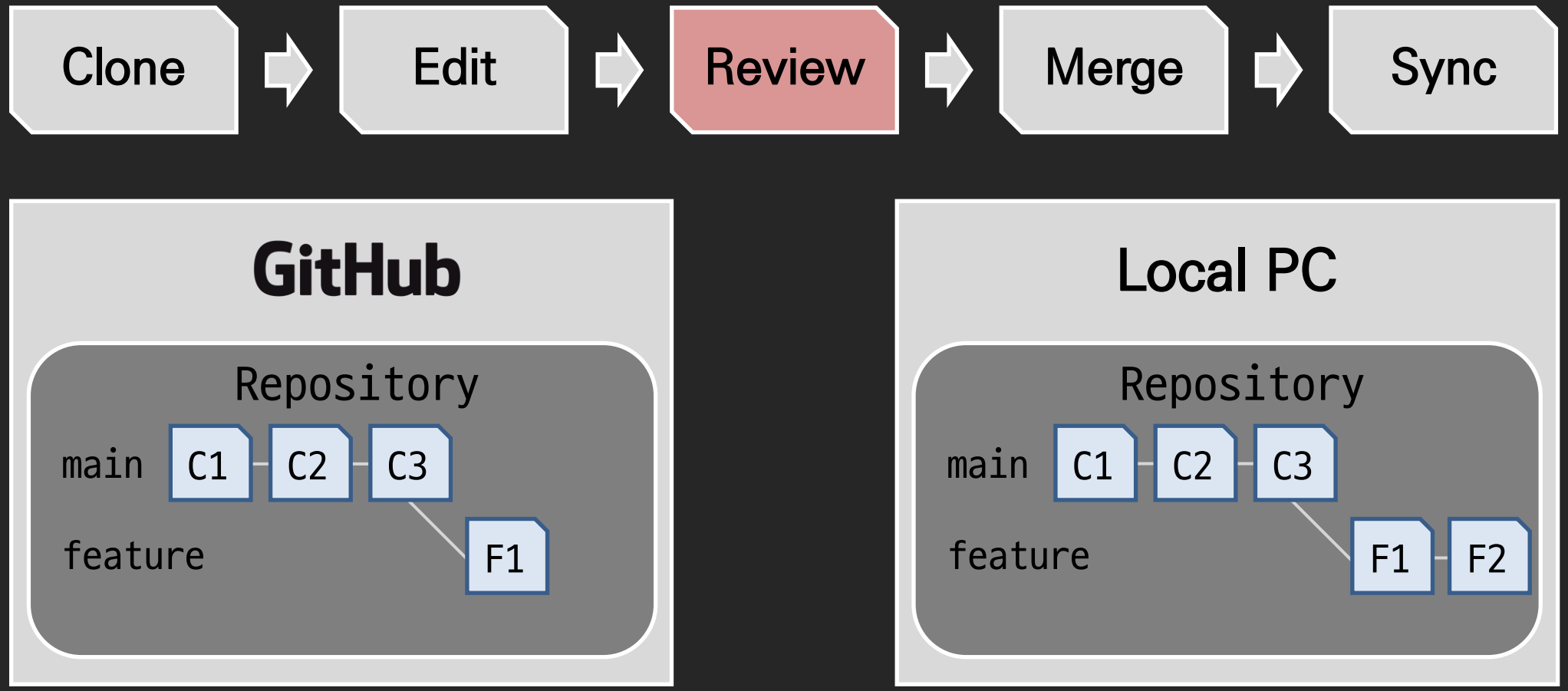
Collaborative Development Steps



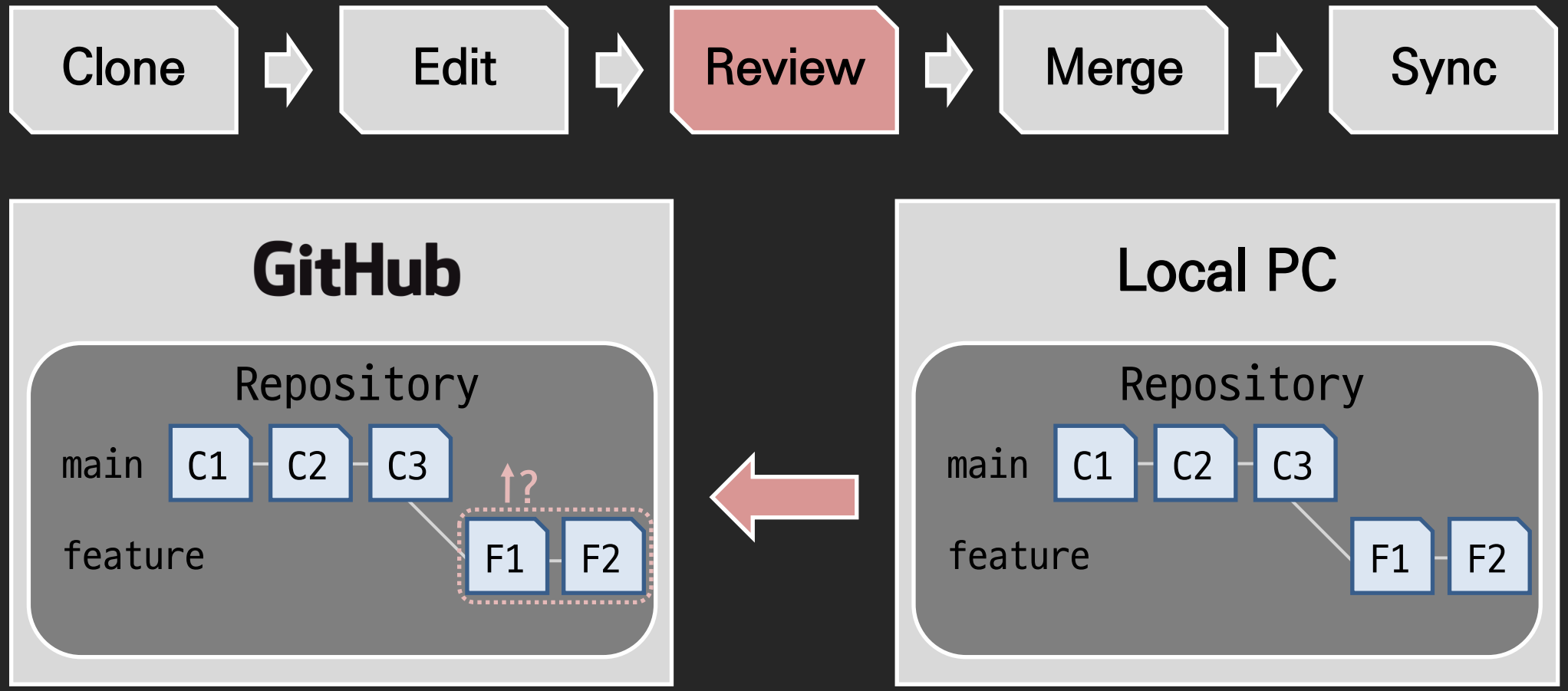
Collaborative Development Steps



Collaborative Development Steps

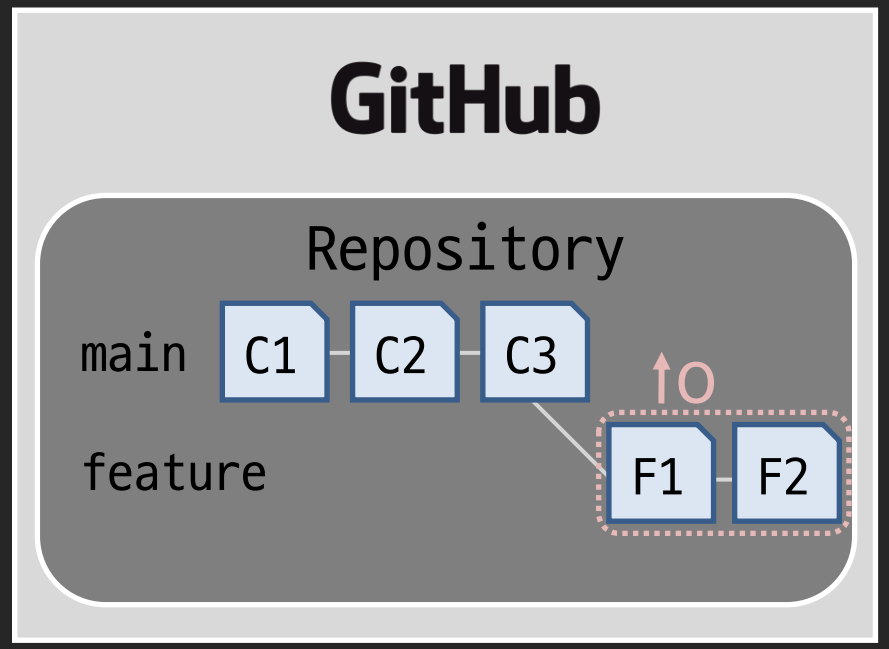


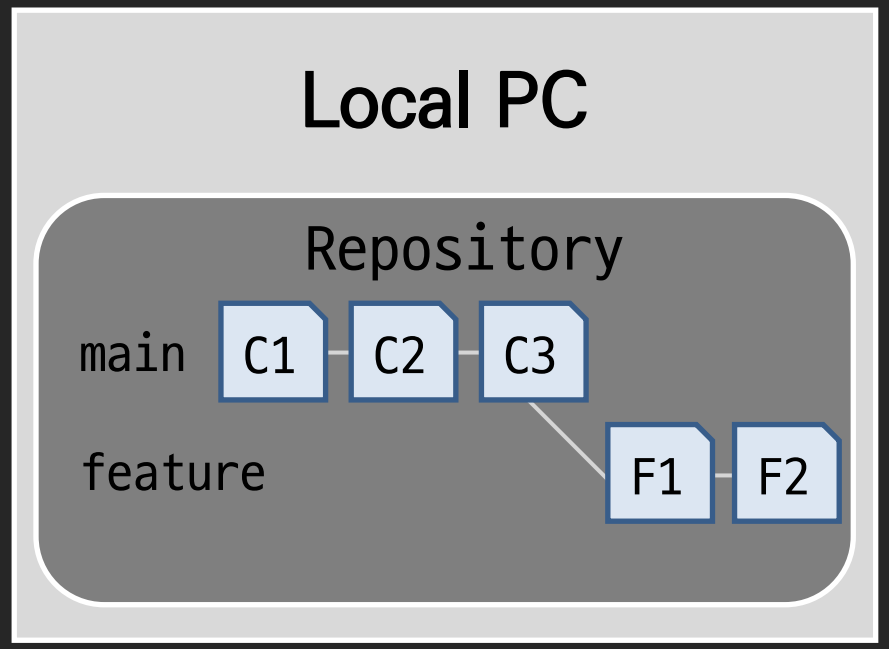
Collaborative Development Steps



Collaborative Development Steps

Clone → Edit → Review → Merge → Sync

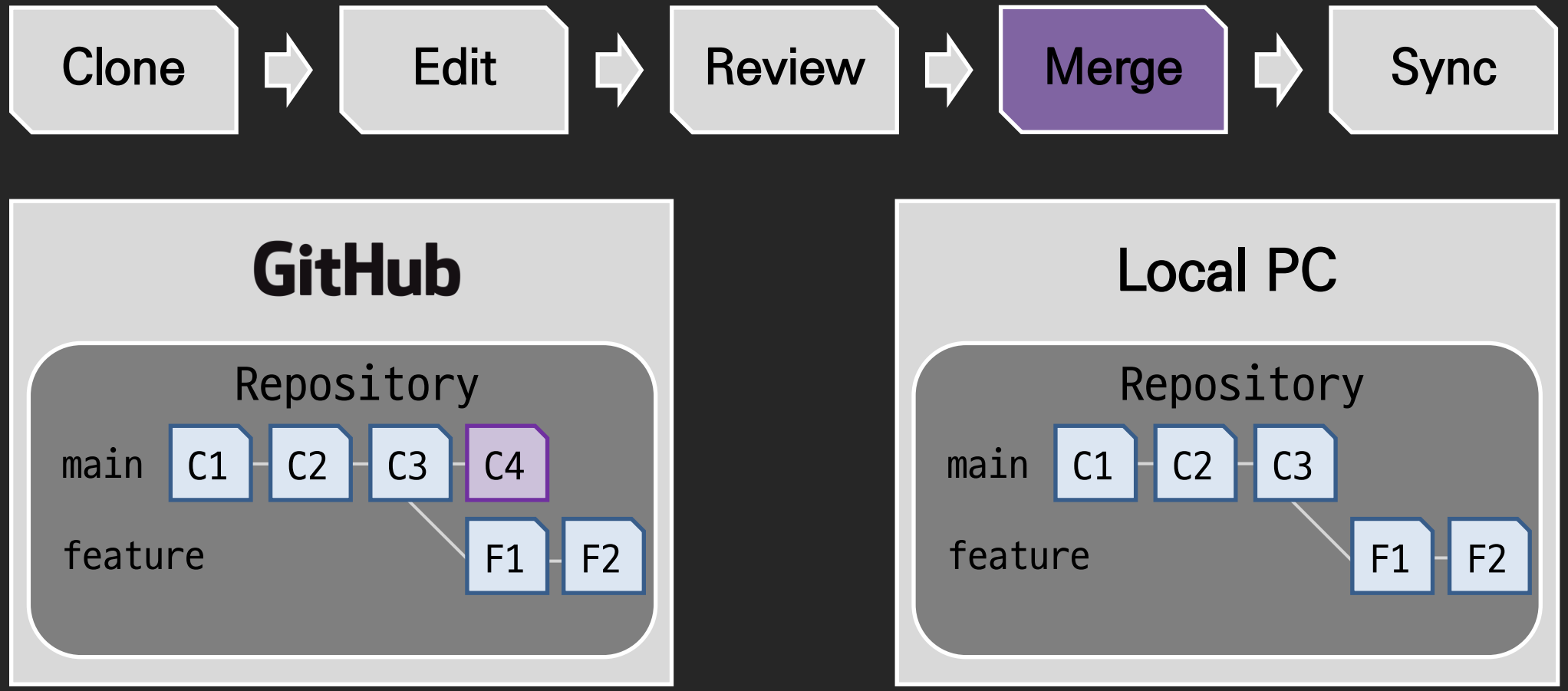




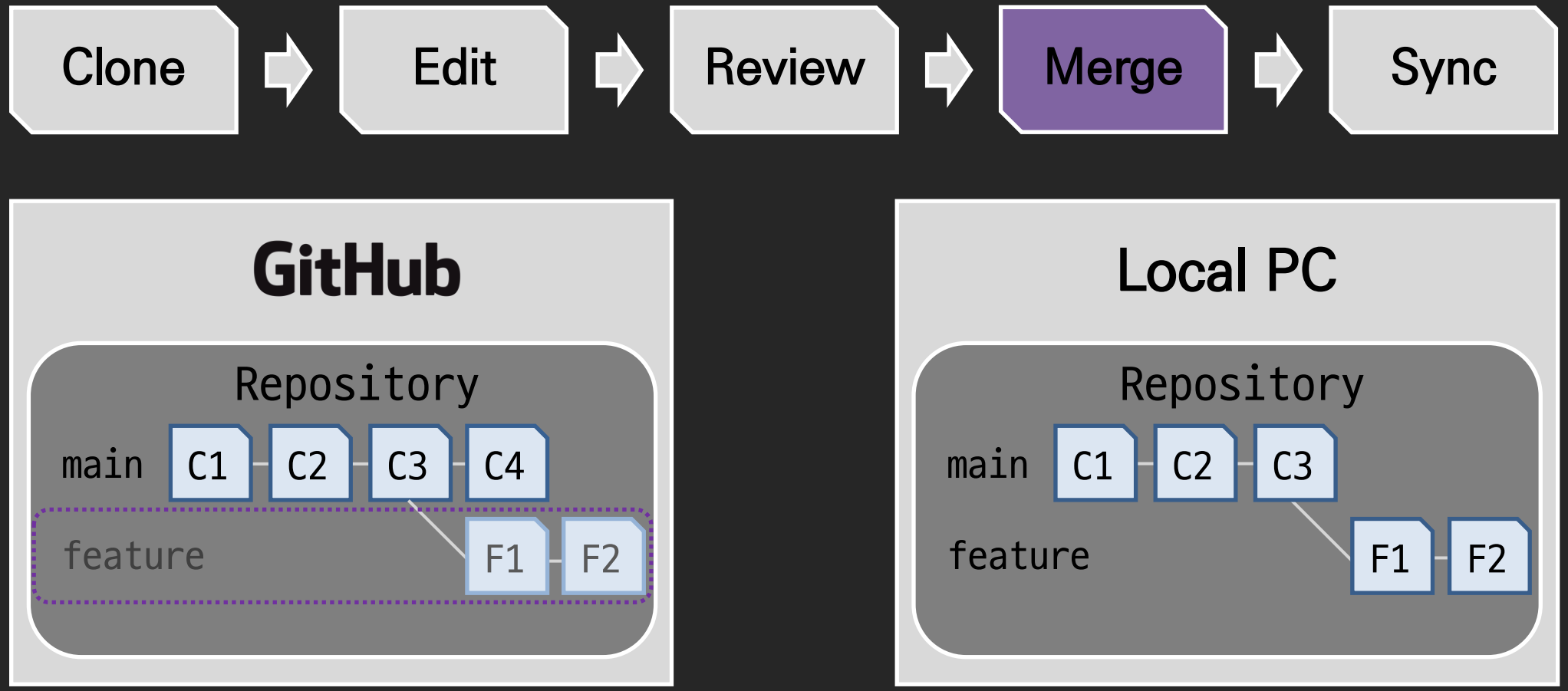
Collaborative Development Steps

Clone → Edit → Review → **Merge** → Sync

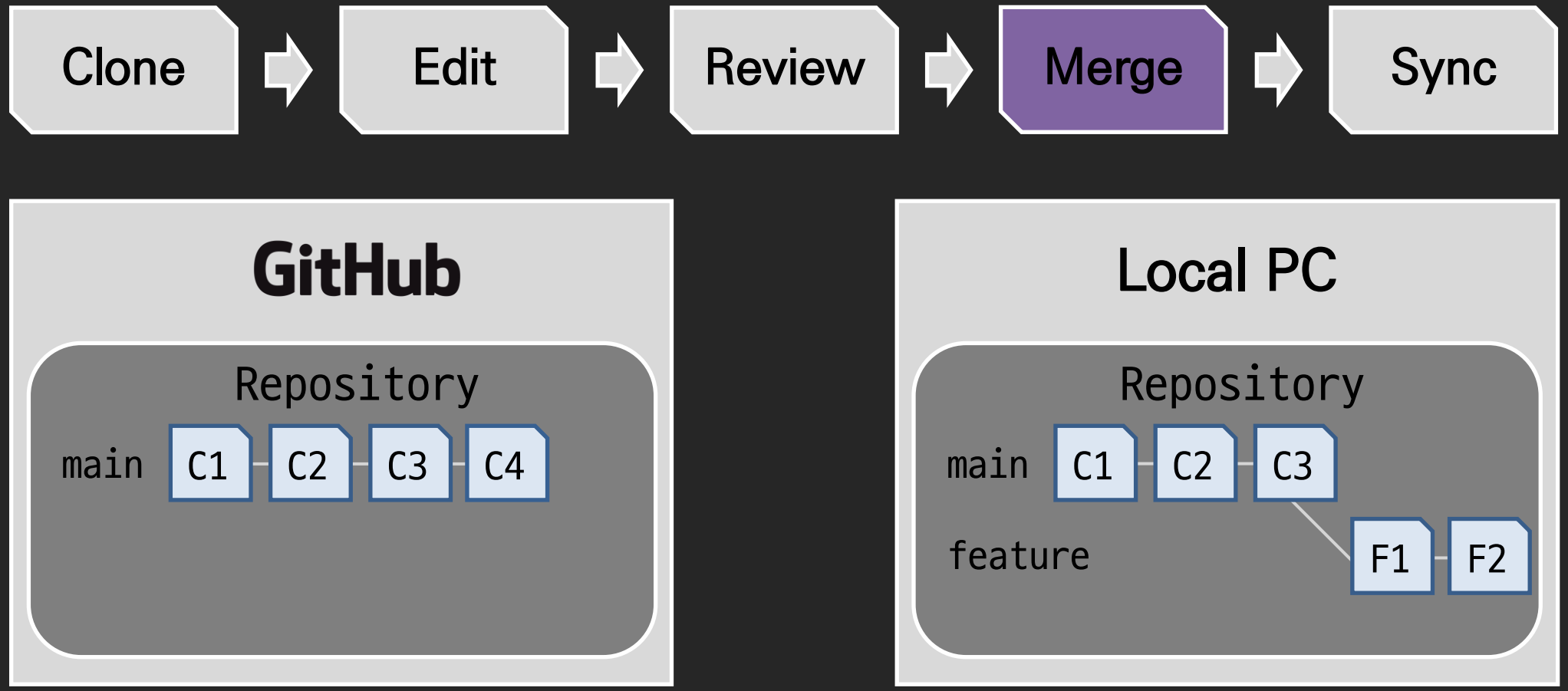
Collaborative Development Steps



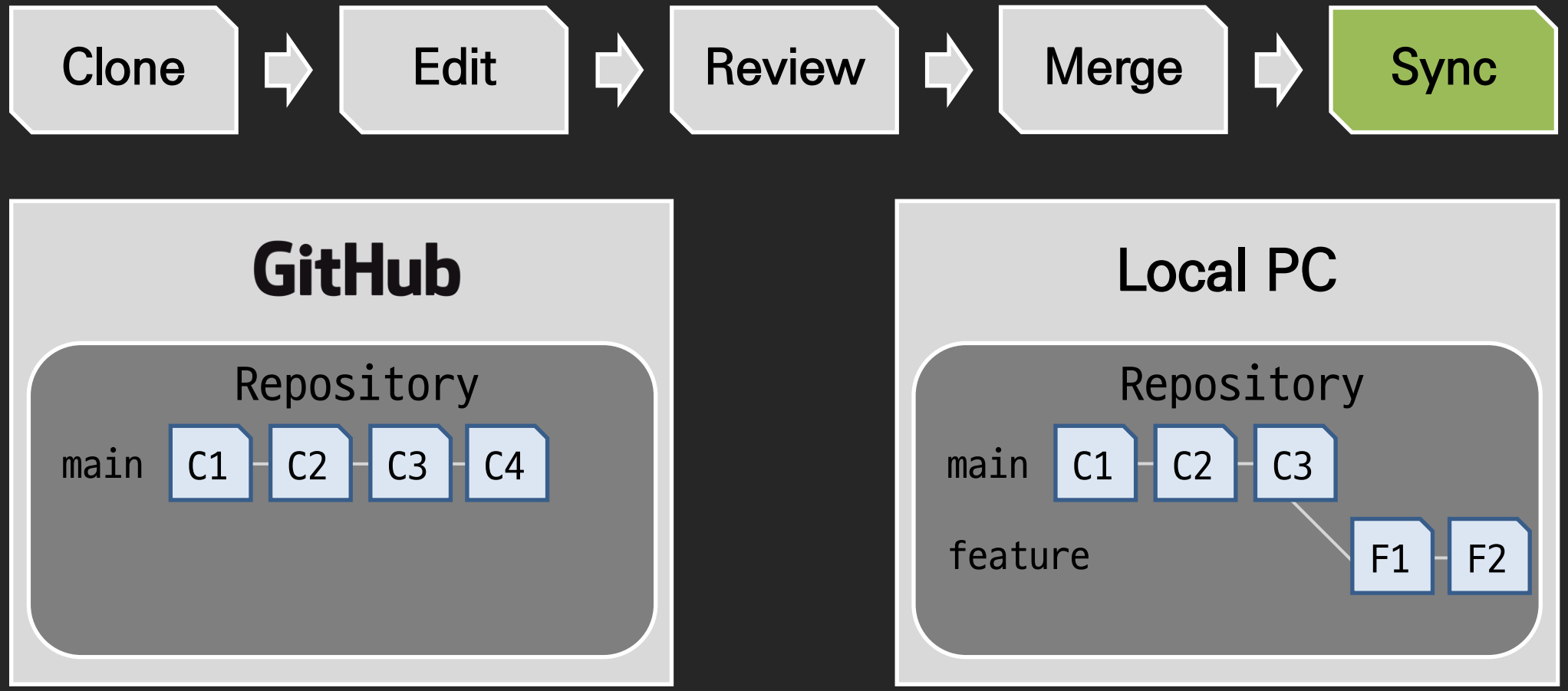
Collaborative Development Steps



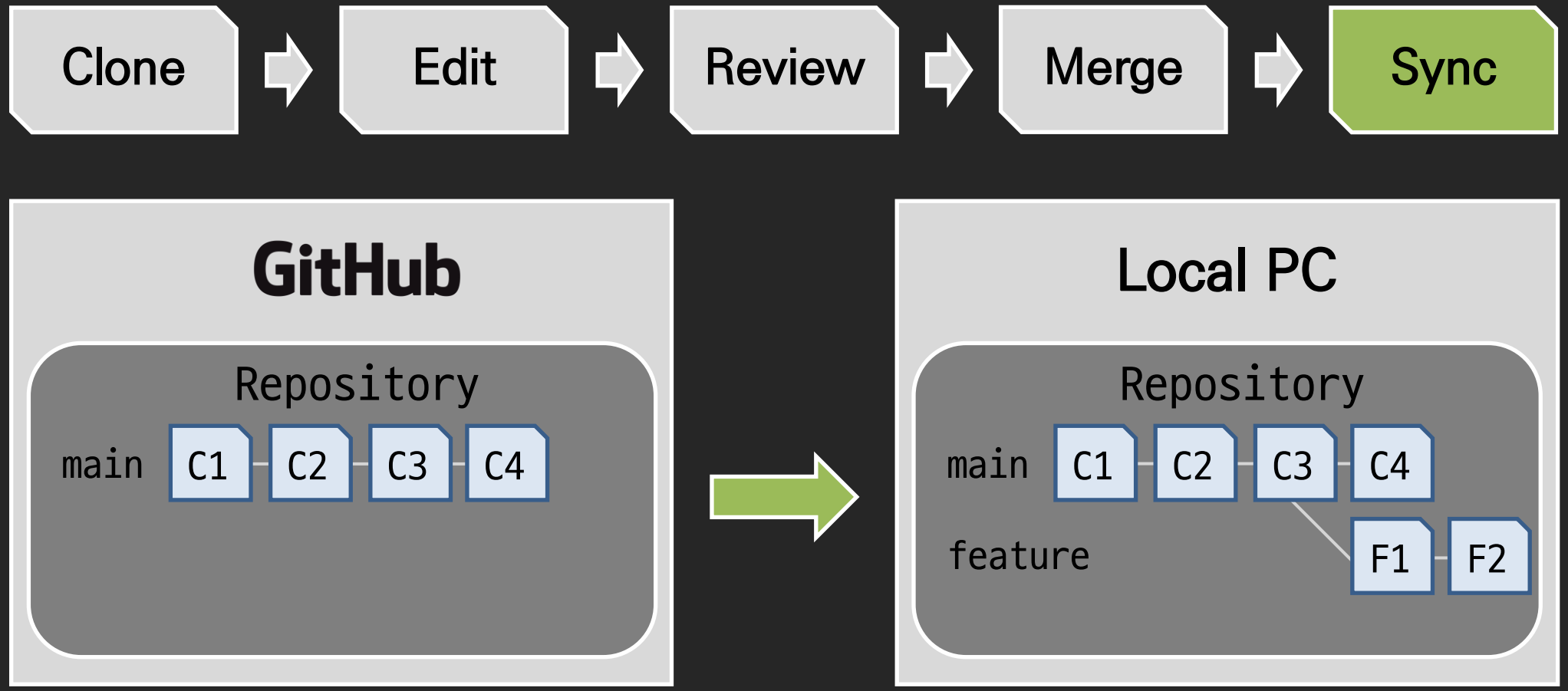
Collaborative Development Steps



Collaborative Development Steps



Collaborative Development Steps



Other Useful Commands

- `git diff`
- `git blame`
- `git bisect`
- `git cherry-pick`
- Google the arguments and usages by yourself!

vscode and git

- vscode has integrated git interface
- Often-used commands are easily accessible
  - Integrated GUI
  - Command palette
- Extensions offer even more convenient features!