# CSE 403

Software Engineering

**Version control and Git**

# Today

- Version control: why, who, how?
- Git: concepts and terminology

# Why use version control?

Common App
Essay

**11:51pm**

# Why use version control?


Common App Essay


Common App Essay EDITED FINAL


Common App Essay FINAL FINAL


Common App Essay FINAL REVISED


Common App Essay FINAL

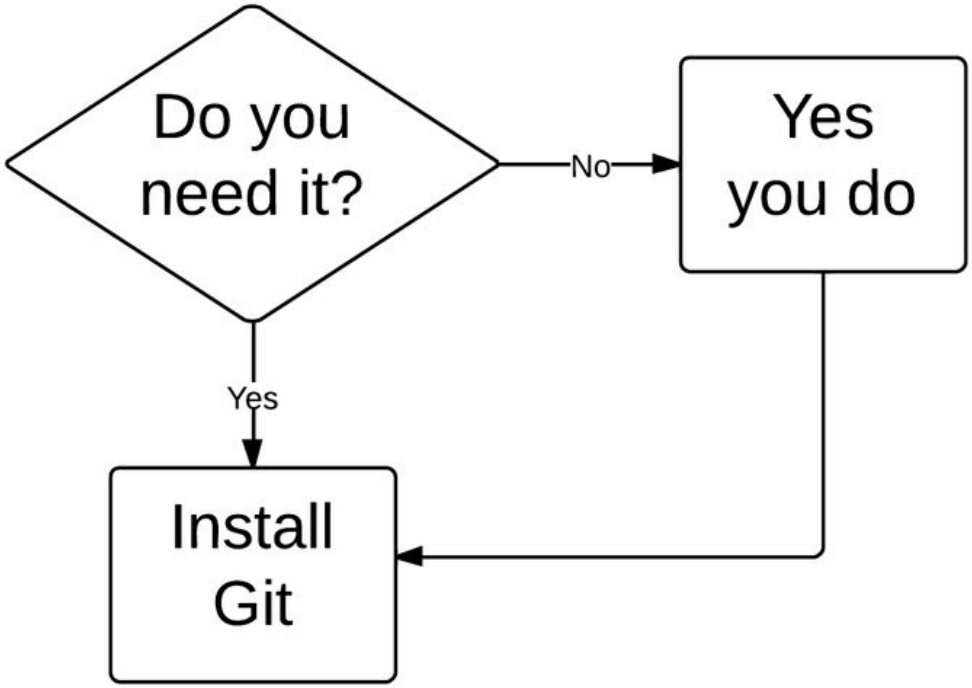

Common App Essay OKAY THIS IS THE FINAL ONE


Common App Essay REVISED FINAL


Common App Essay REVISED

# Version control

Version control records changes to a set of files over time.
This makes it easy to review or obtain a specific version (later).



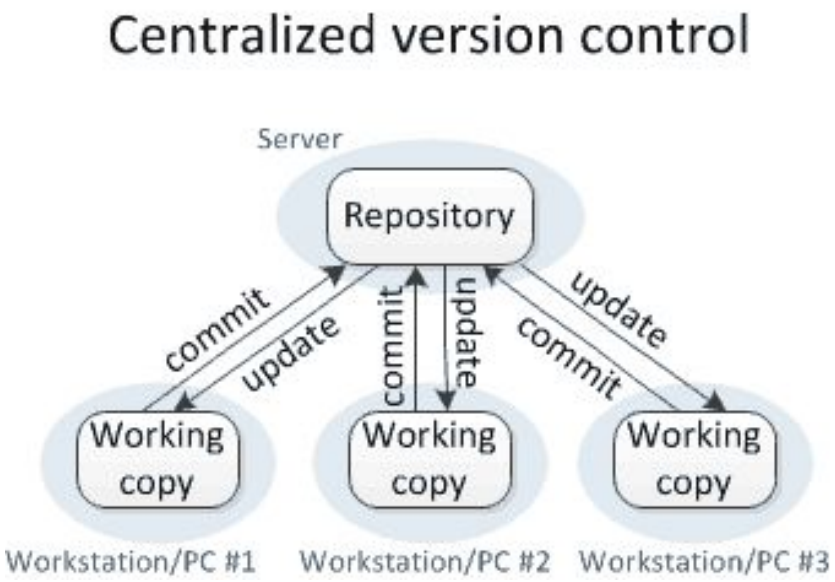
# Who uses version control?

**Example application domains**

- Software development
- Research (infrastructure and data)
- Applications (e.g., (cloud-based) word processors)

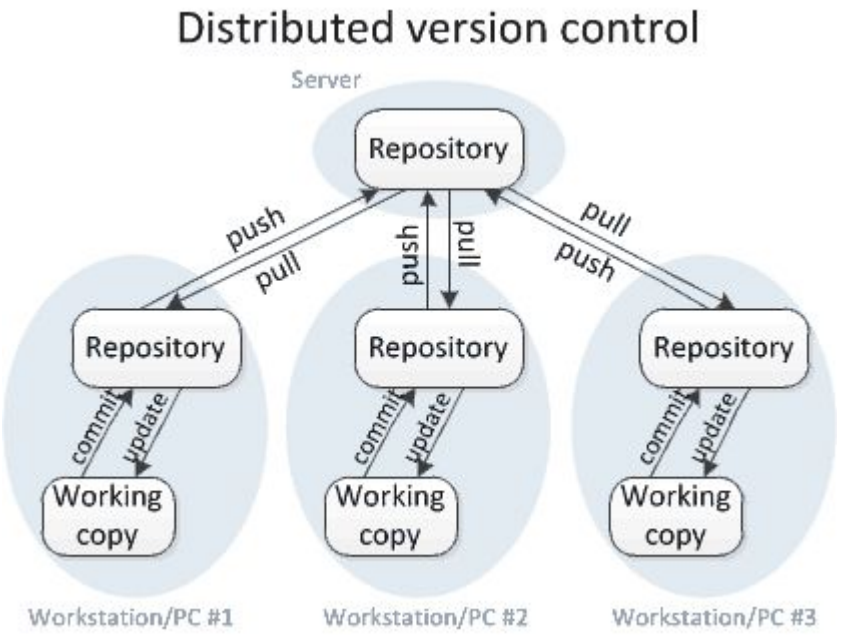
# Centralized version control

- **One central repository.**
- All users **commit** their changes to a **central repository**.
- Each user has a working copy. As soon as they commit, the repository gets updated.
- Examples: SVN (Subversion), CVS.



# Distributed version control

- **Multiple copies of a repository**.
- Each user **commits** to a **local** (private) repository.
- All committed changes remain local unless **pushed** to another repository.
- No external changes are visible unless **pulled** from another repository.
- Examples: Git, Hg (Mercurial).

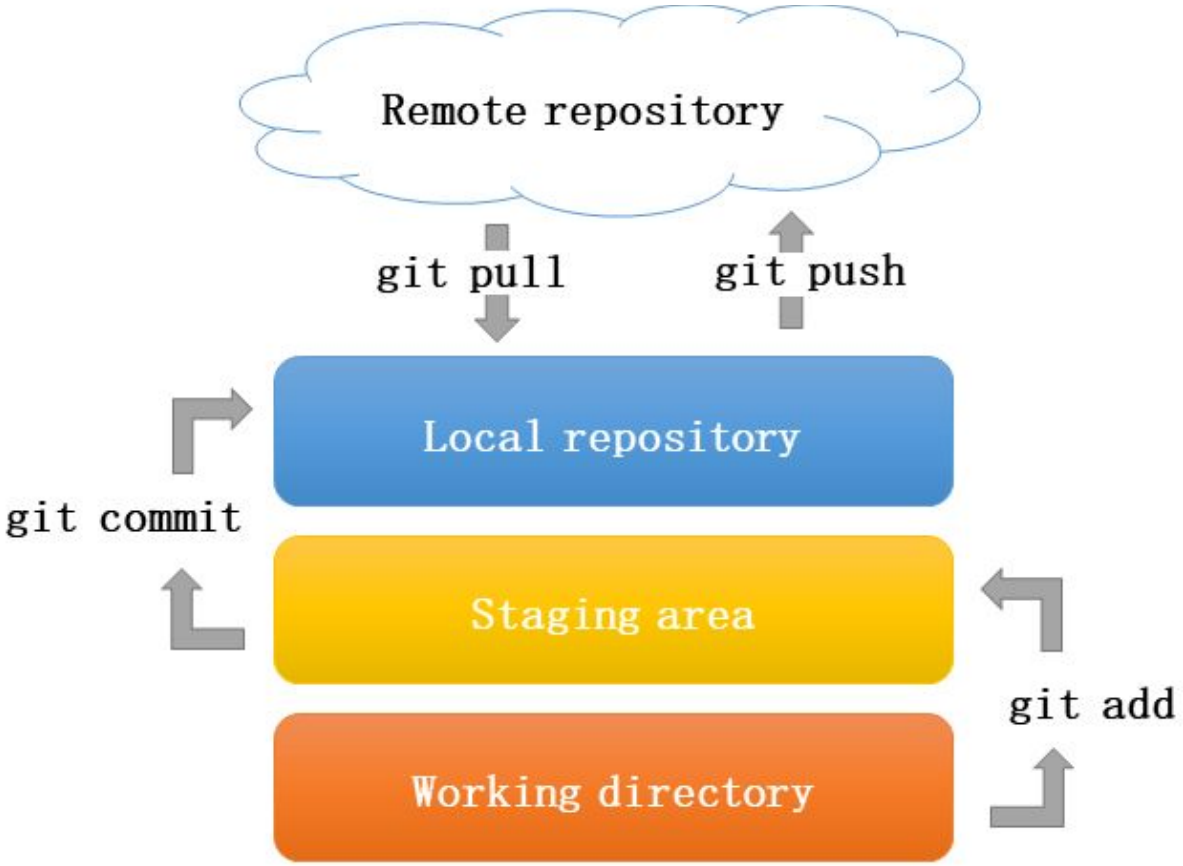


# A Fun Git Quiz

https://forms.gle/vG5yJo5LVAbA8fHQ7

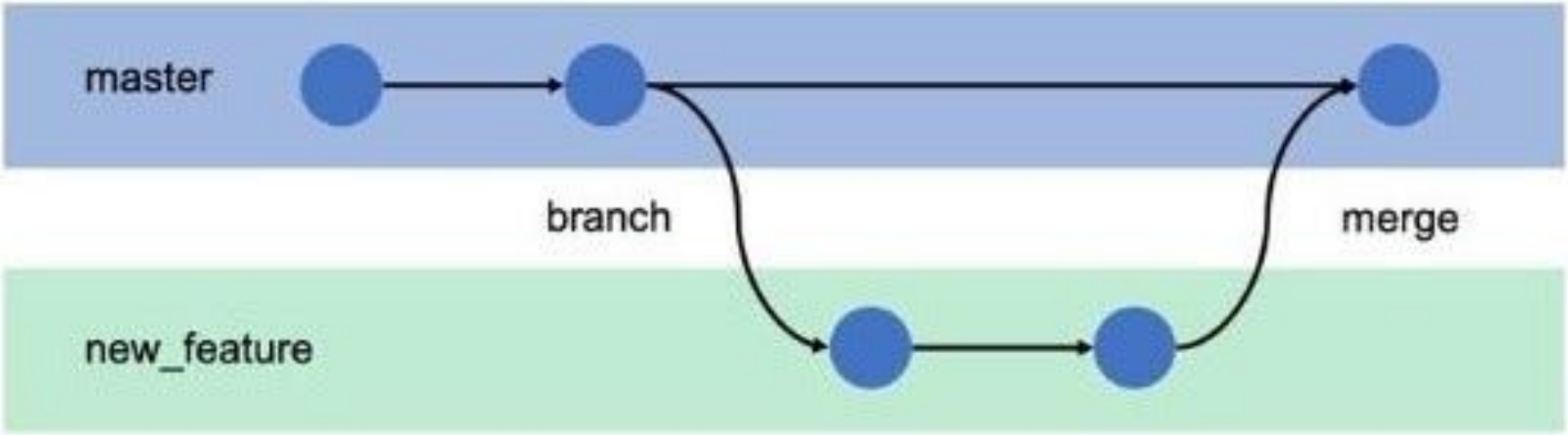
# Git commands [Demo]

- **clone**: linked copy of repo
- **fork**: independent copy of repo
- **add → commit → push**

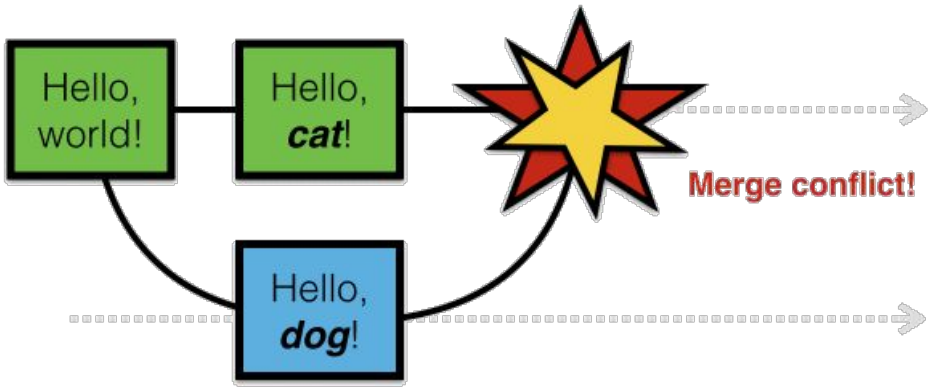


# Branches [Demo]

- **git branch <branch_name>**: create a branch
- **git checkout <branch_name>**: go to branch
- **git checkout -b <branch_name>**: both at same time
- **git merge <branch_name>**: bring all of `branch_name`s changes into current branch
- **Pull Request for main ← <branch_name>**: request to merge `branch_name` into main and see **diff** between them (*GitHub* operation)



# Conflicts




- **Conflicts** arise when two users **change the same line** of a file.
- When a conflict arises, the last committer needs to resolve it.

# Git pull/fetch [Demo]

- **git fetch**: downloads changes from remote repo without modifying working copy
- **git pull = git fetch + git merge**: downloads and merges changes from remote repo into working copy

*How to avoid conflicts?*

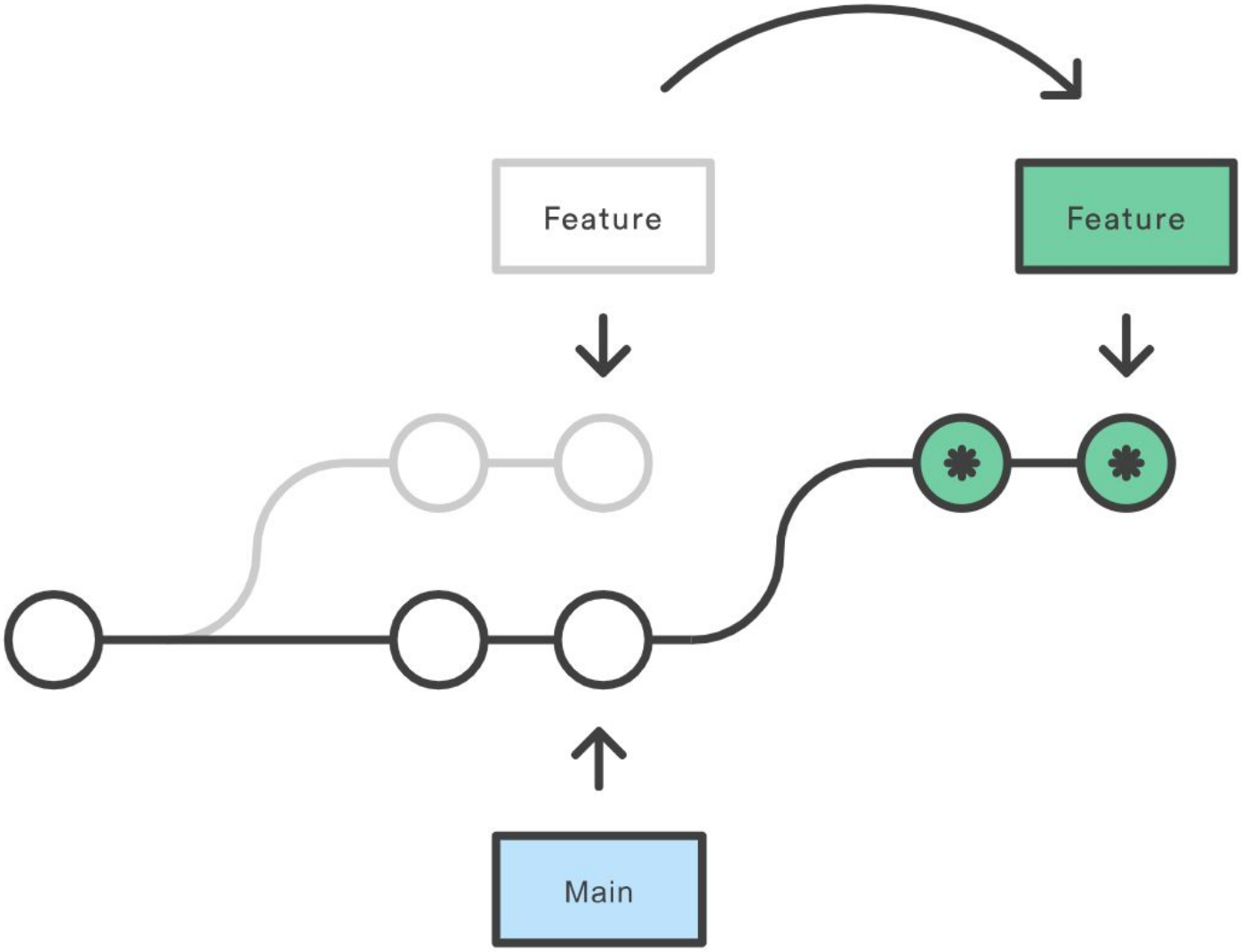
- Do git pull often before pushing
- If you are on a feature branch, do **git pull origin/main** before pushing so that merging feature branch → main is easier
  - Don't have long-running feature branches, otherwise feature branch will get too outdated from main

# Merge vs. Rebase

- Goal of rebase: prepare your work before delivery
  - “Let me clean up my feature branch so it's easier to read and merge.”
  - Action: copies all of feature branch’s commits on top of main (removes old commits)
  - Result: Your feature branch is now based on the latest main, and has a clean linear history.

- Command: On feature-branch, do **git rebase main**

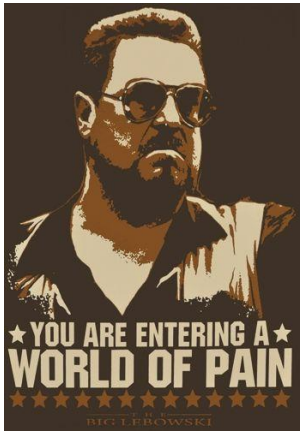
# Git rebase



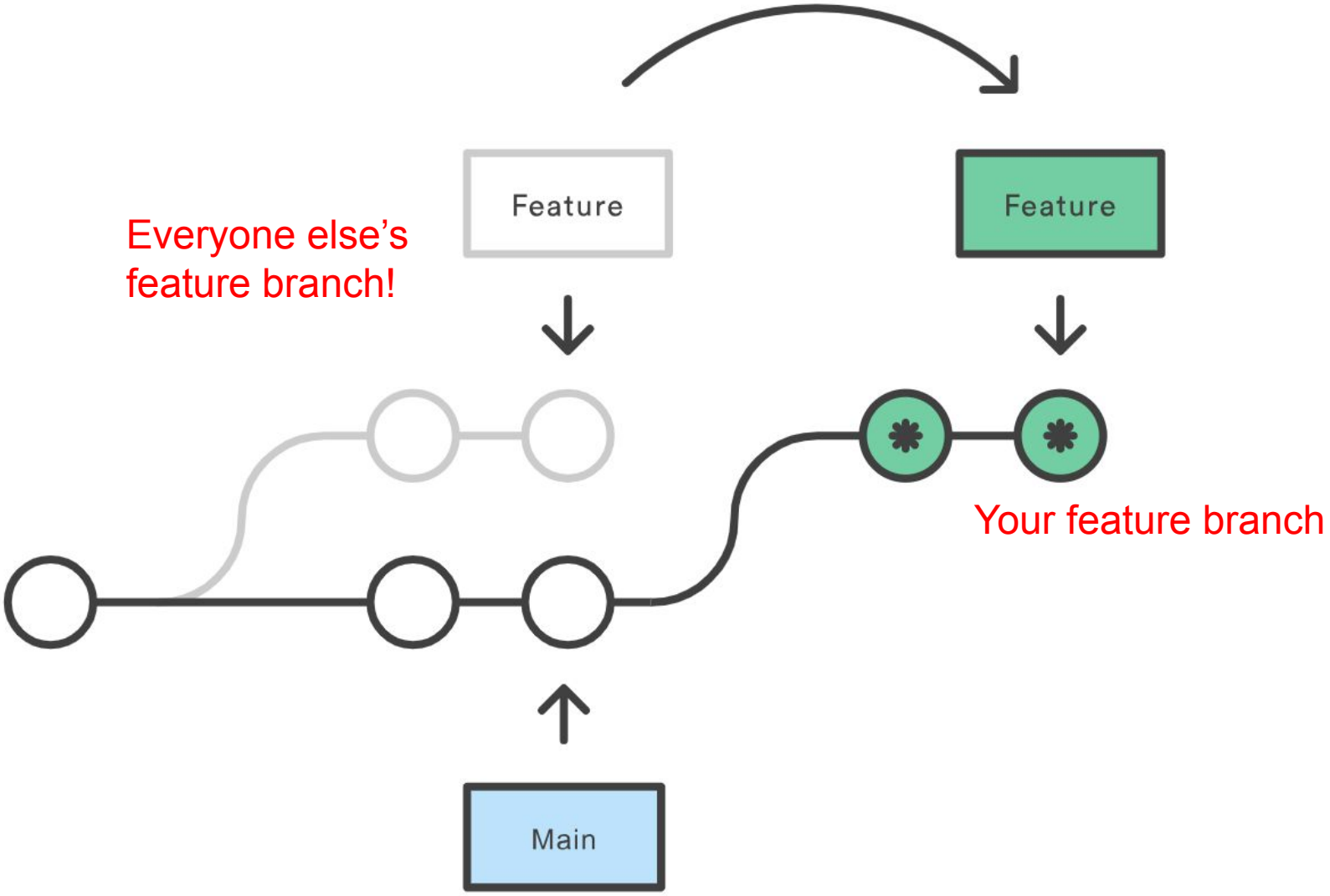
# Rebase: a powerful tool, but …

- **Changes the commit history!**
- Anyone else on the feature-branch will have the old commits, so it will be **really hard** for them to push to that branch!!


**Do not rebase public branches!**



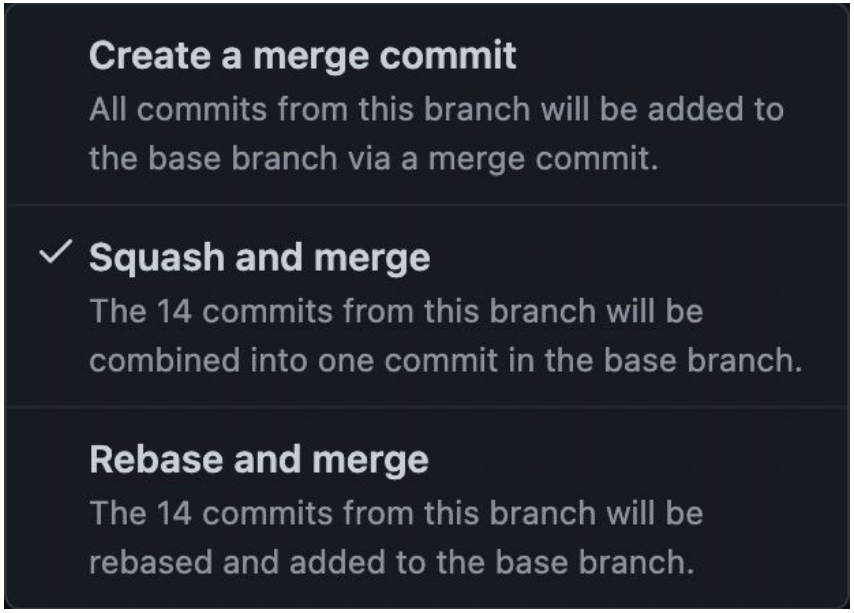
# Rebase: a powerful tool, but …



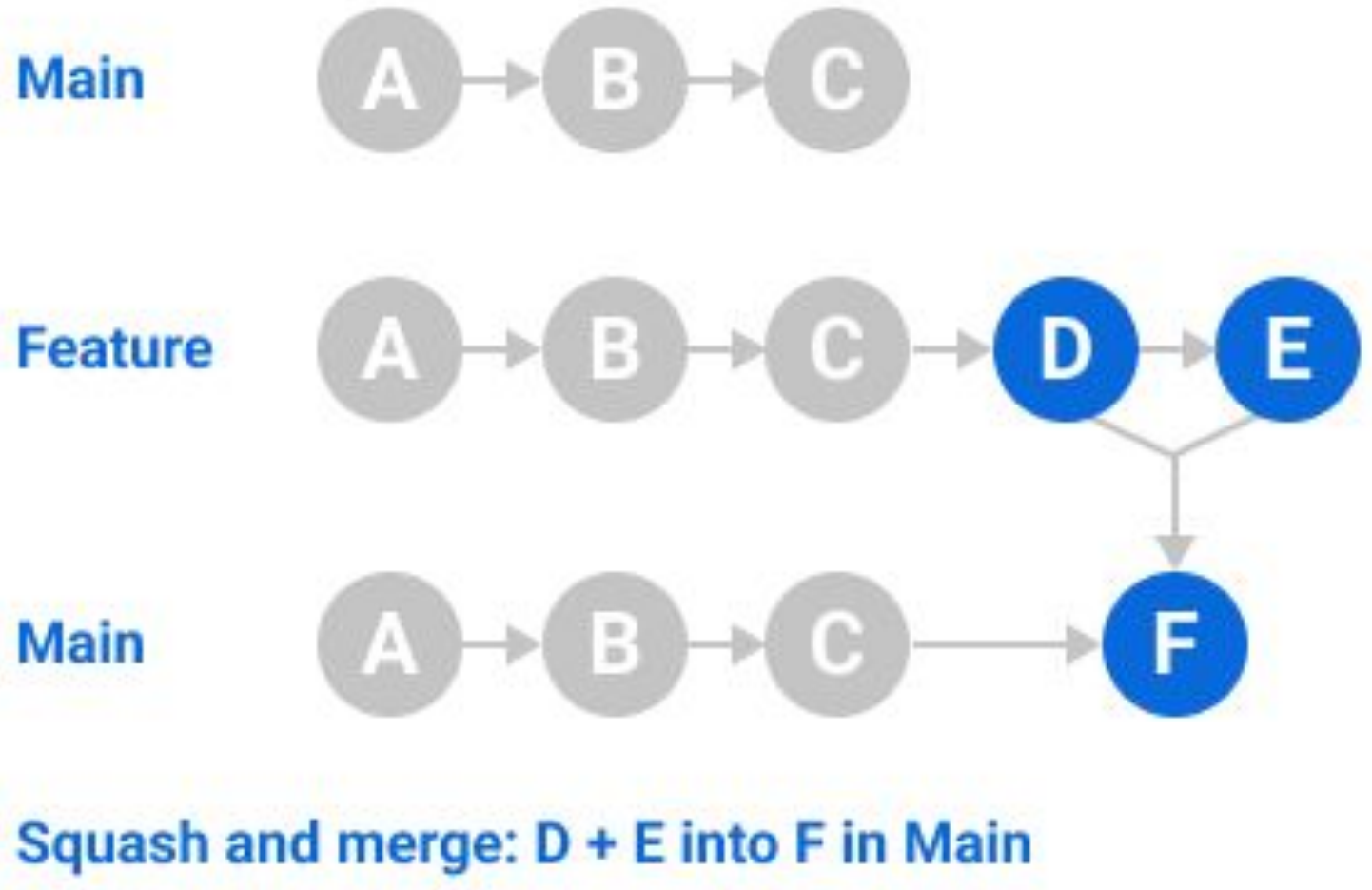
# Merge vs Squash & Merge

- Goal: **want a clean commit history** when merging
  - don’t want to keep history of all of feature branch’s commits!
  - Action: squashes all of feature-branch’s new changes into a single new commit and puts that onto main
  - Result: main has all of feature-branch’s changes with only 1 new commit

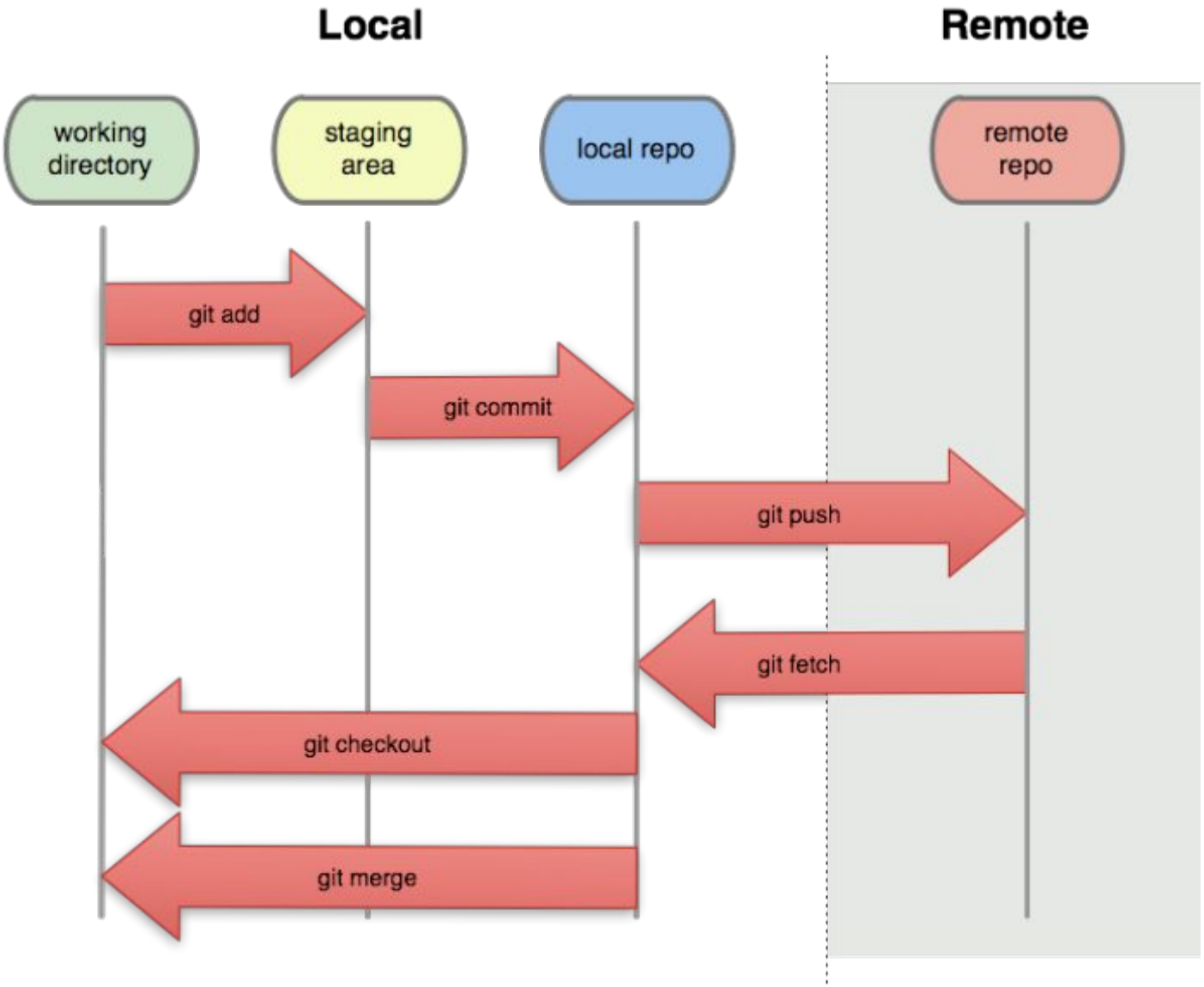
*Not a Git command, only on GitHub*



# Squash & Merge



# Summary of Git concepts



# Conventions when using Git

- Use feature branches, don’t modify main directly
- Don’t merge to main directly: make a PR and request code reviews from colleagues before merging to main
- Commit & push often
- Have good commit messages
- No long-running feature branches
- Do **git pull** often to stay up-to-date with changes
- Have good communication with your colleagues to prevent merge conflicts