

# Principles of Software Construction: Objects, Design, and Concurrency

Managing change (2)

Charlie Garrod

**Bogdan Vasilescu**

# Administrivia

- Homework 6 checkpoint deadline (Monday, April 30th)
- Homework 6 due Wednesday, May 2nd
  
- Final exam Monday May 7th 5:30-8:30 PH 100
- Review session Saturday May 5th WH 5403

# Key concepts from Tuesday

# Components of Modern CM

Version Control: Branches/Forks/Workflows

Task and Build managers

Build machines, virtual environments (dev stacks)

Package managers

Containers, VMs, in the Cloud

Deployment – Infrastructure as Code.

Data migration

*Other issues:* orchestration, inventory, compliance

# Semantic Versioning for Releases

- Given a version number MAJOR.MINOR.PATCH, increment the:
  - MAJOR version when you make incompatible API changes,
  - MINOR version when you add functionality in a backwards-compatible manner, and
  - PATCH version when you make backwards-compatible bug fixes.
- Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

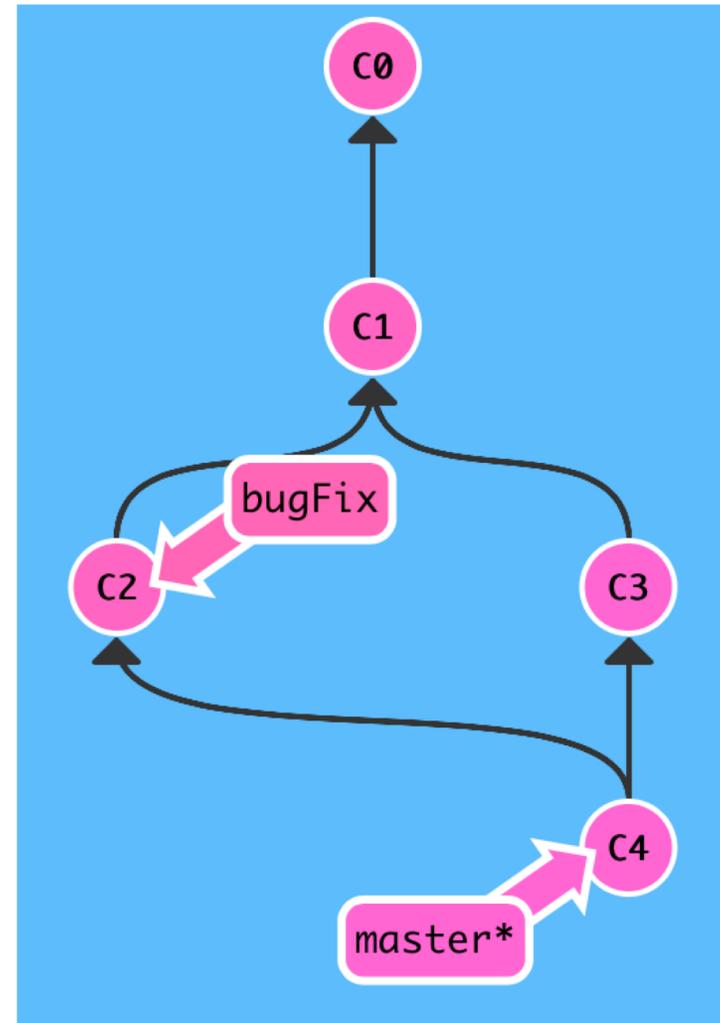
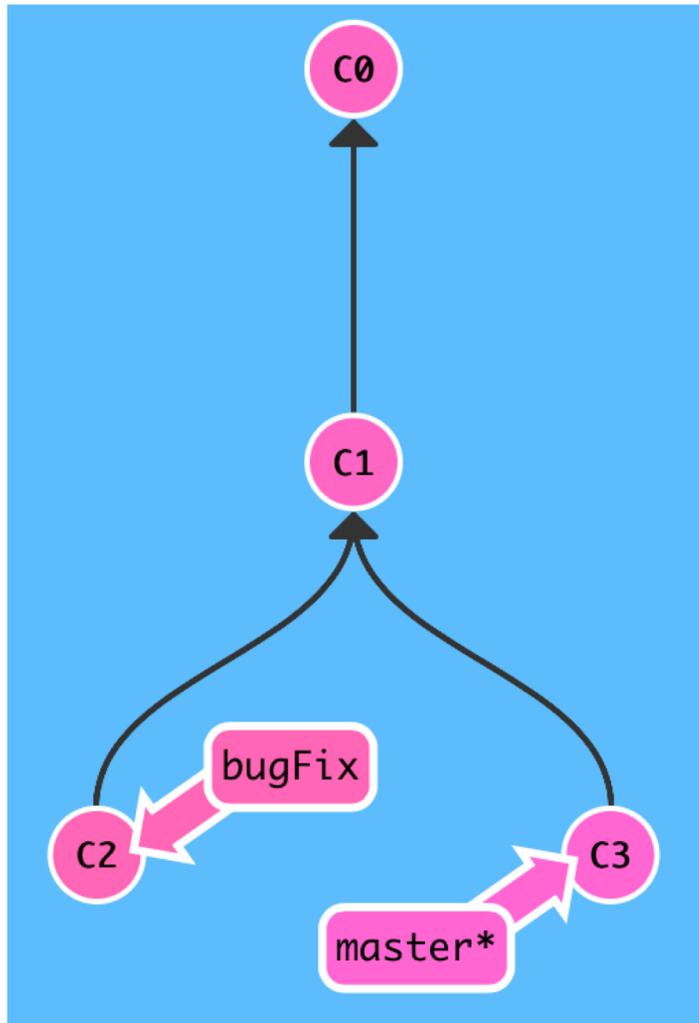
**<http://semver.org/>**

# GIT BASICS

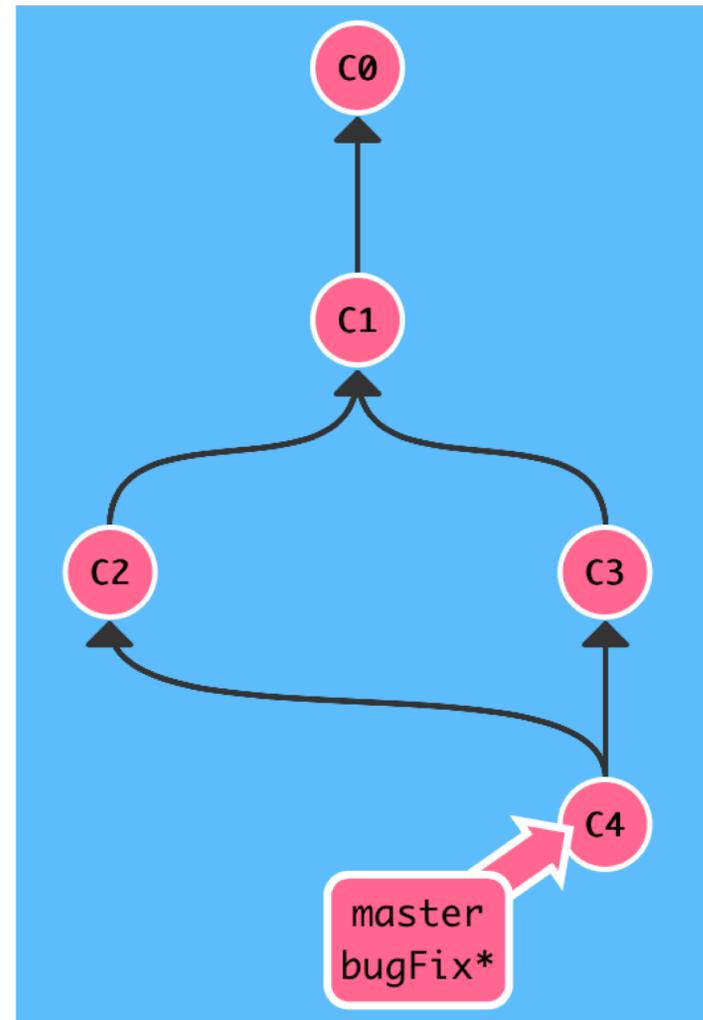
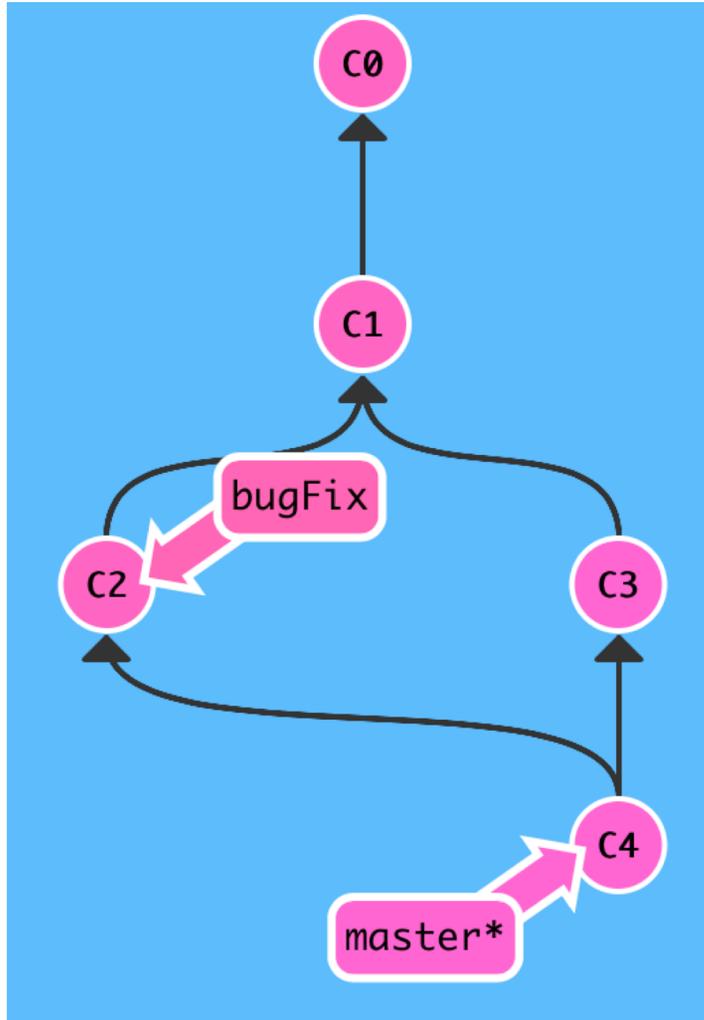
Graphics by <https://learngitbranching.js.org>

# Three ways to move work around between branches

## 1) git merge bugFix (into master)

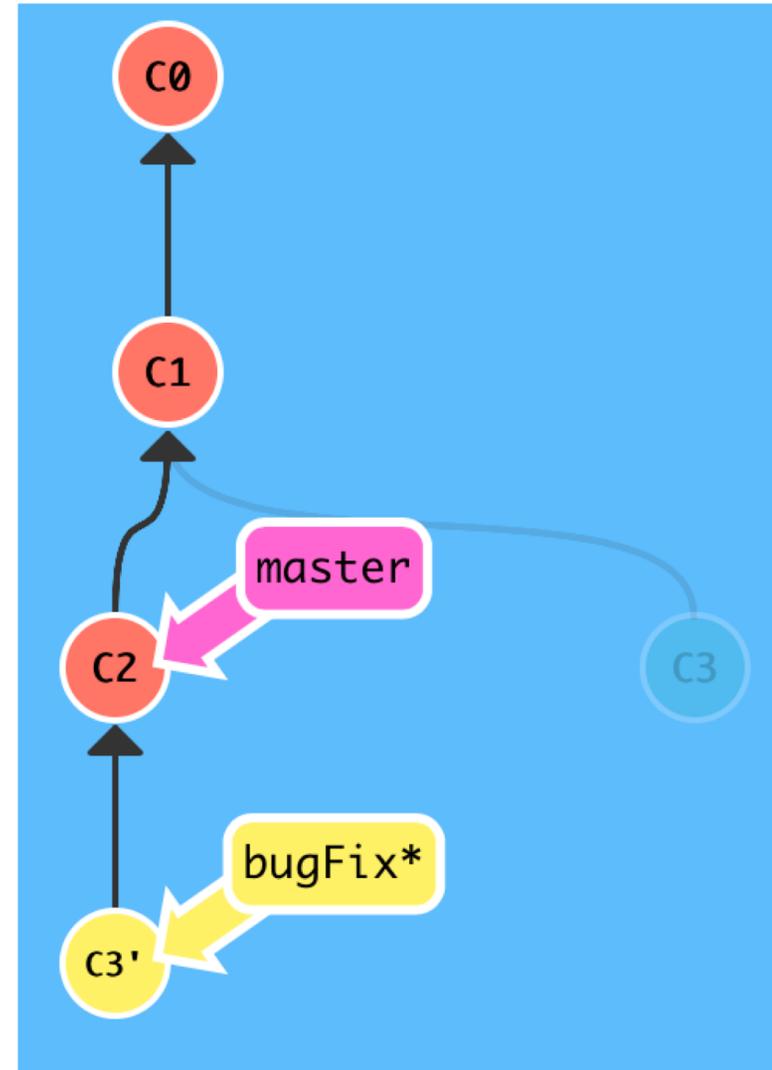
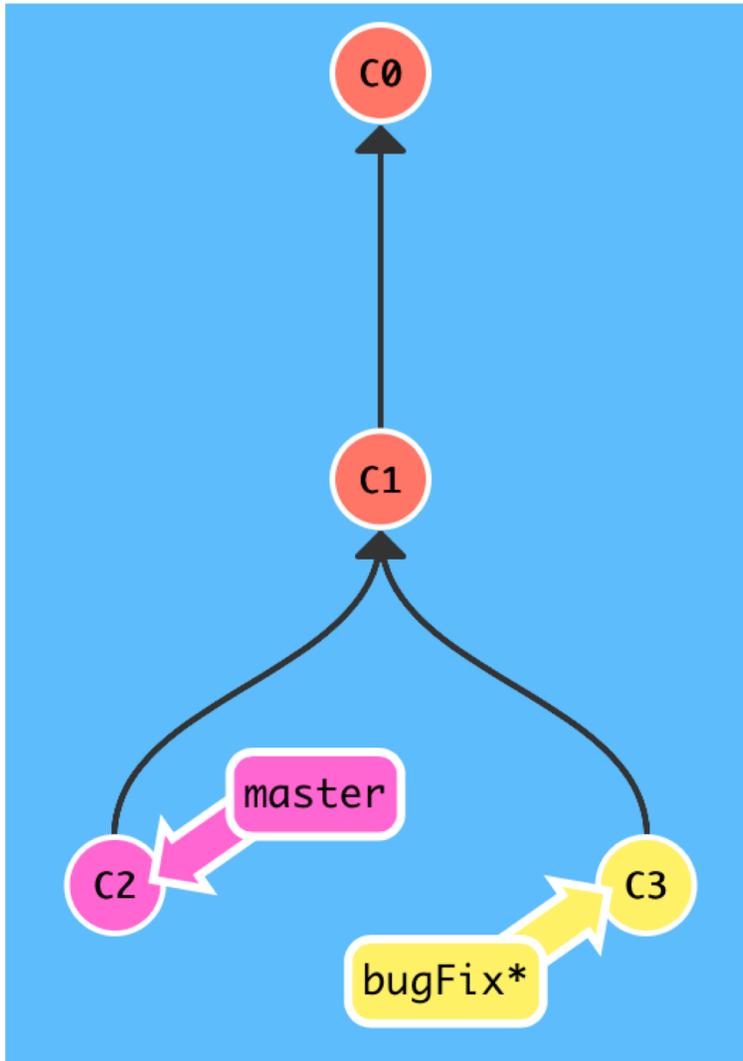


git checkout bugfix; git merge master (into bugFix)



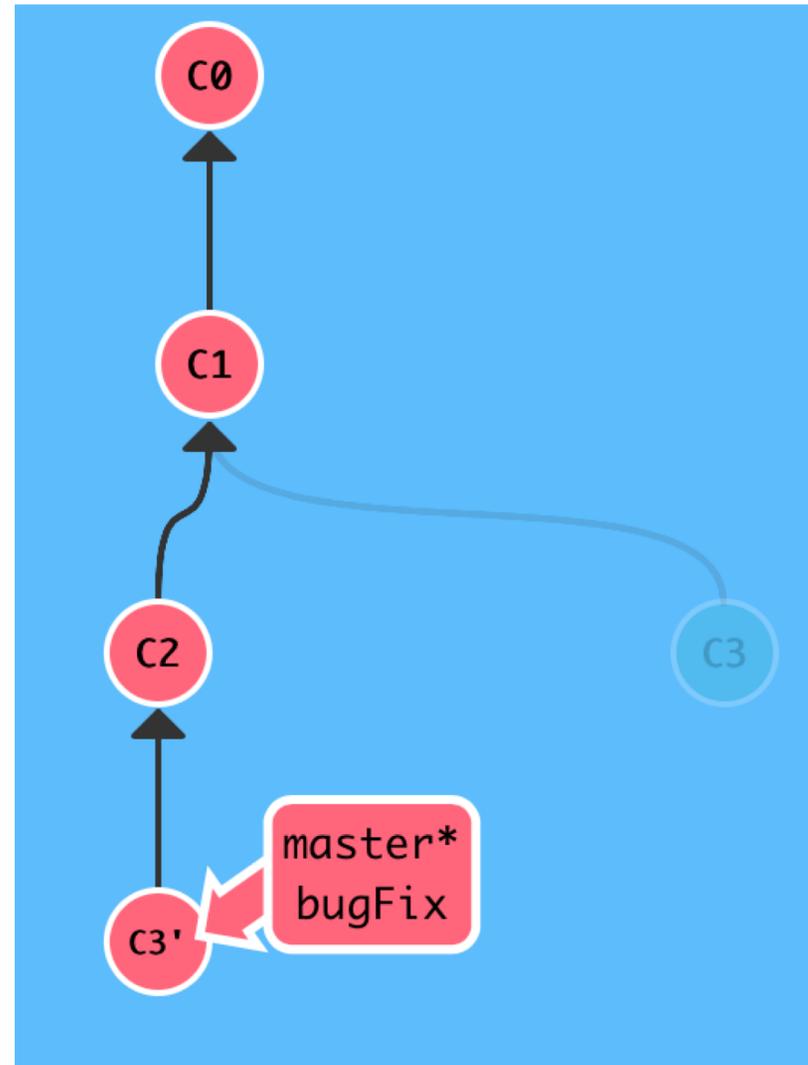
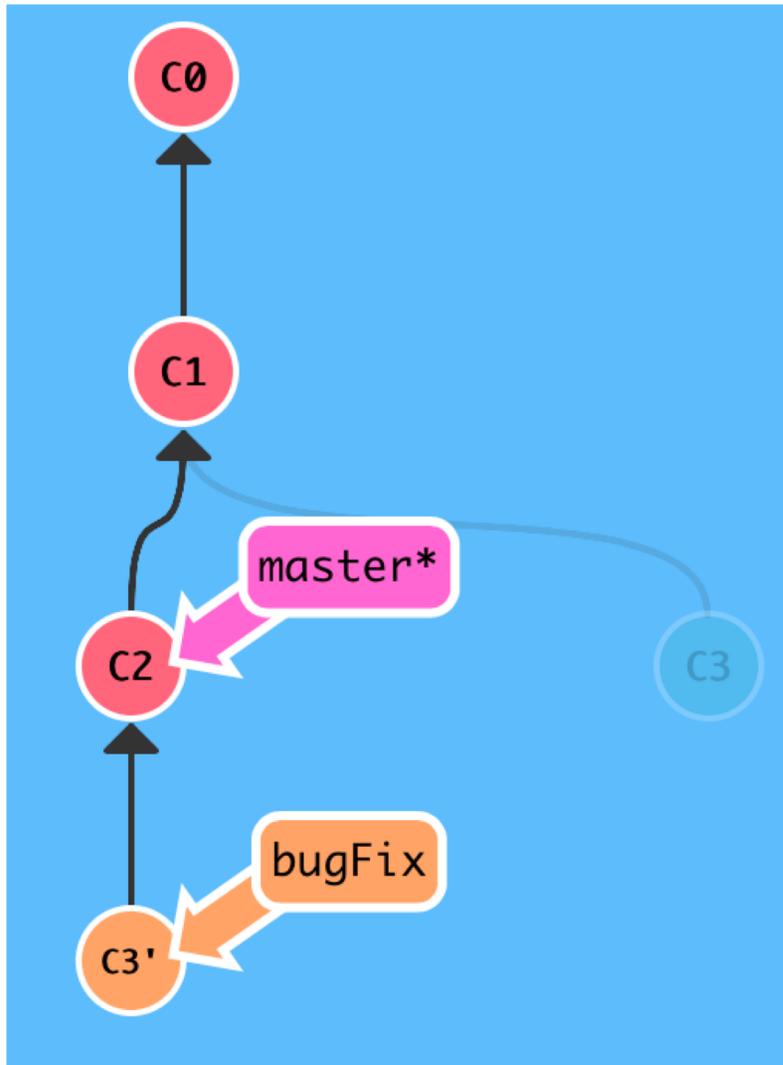
Move work from bugFix directly onto master

## 2) git rebase master



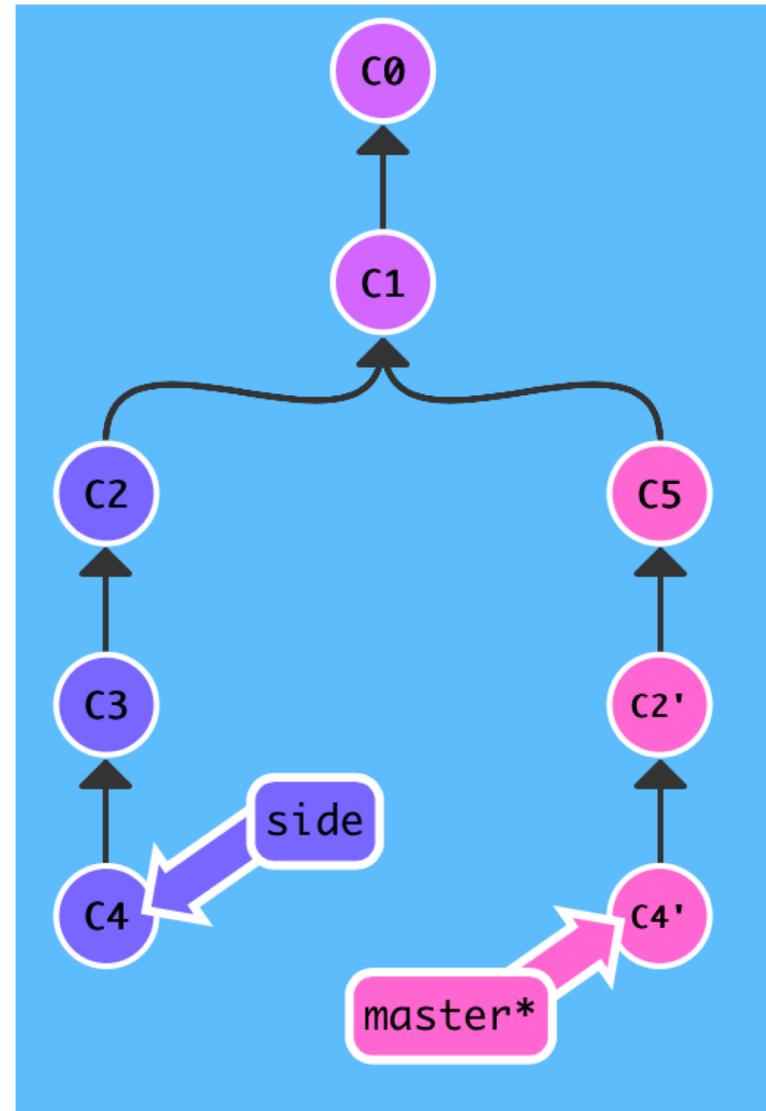
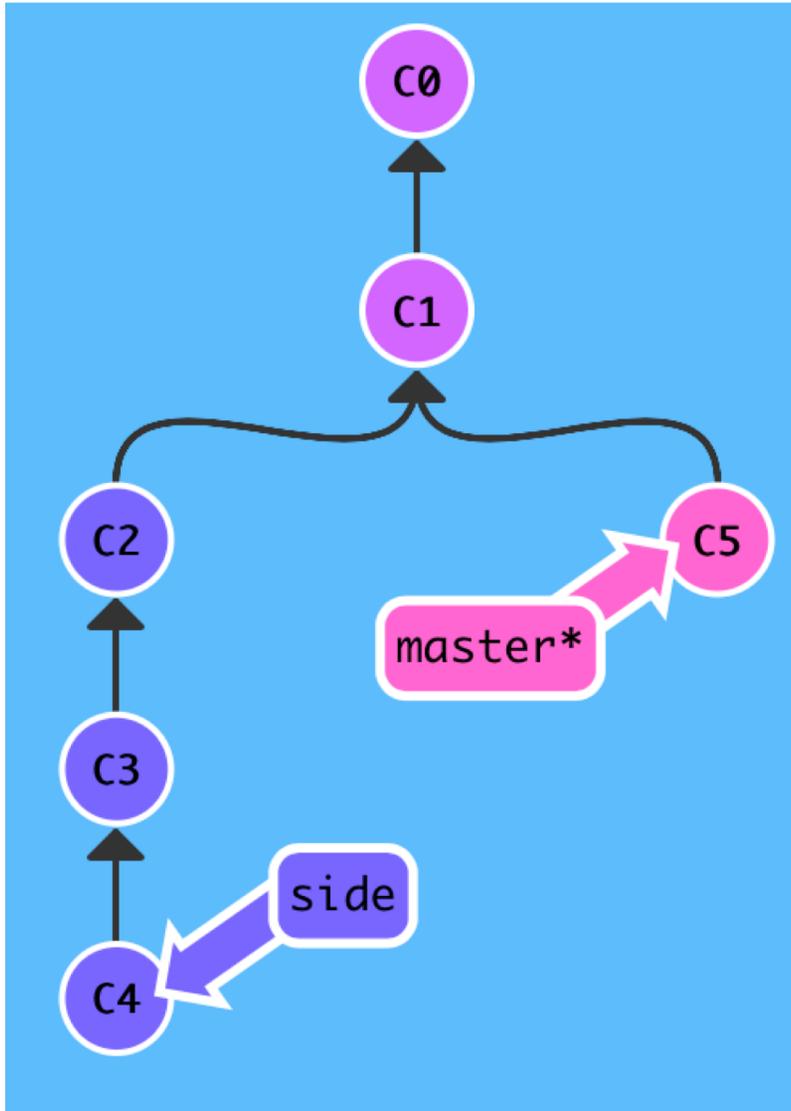
But master hasn't been updated, so:

```
git checkout master; git rebase bugFix
```

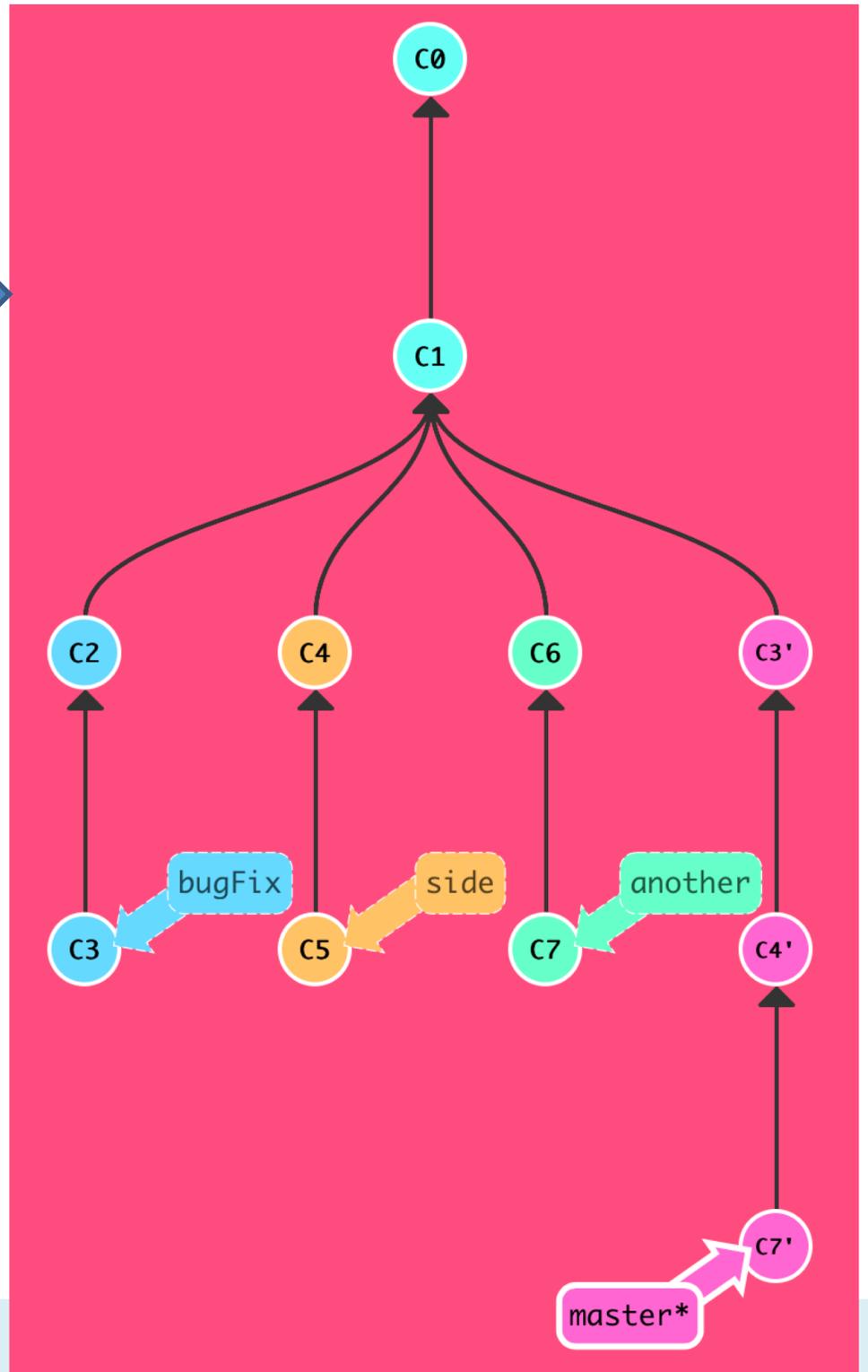
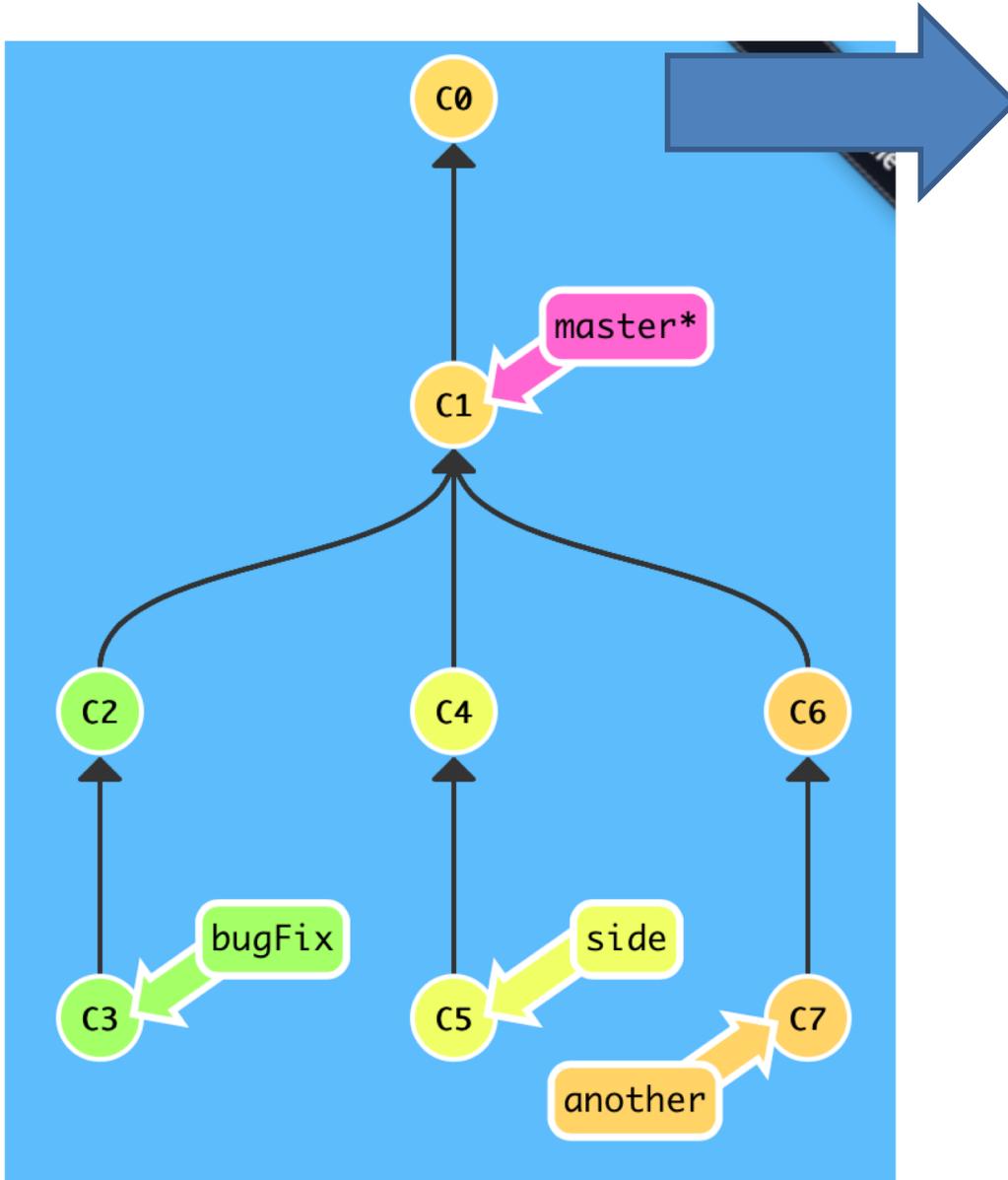


Copy a series of commits below current location

3) `git cherry-pick C2 C4`

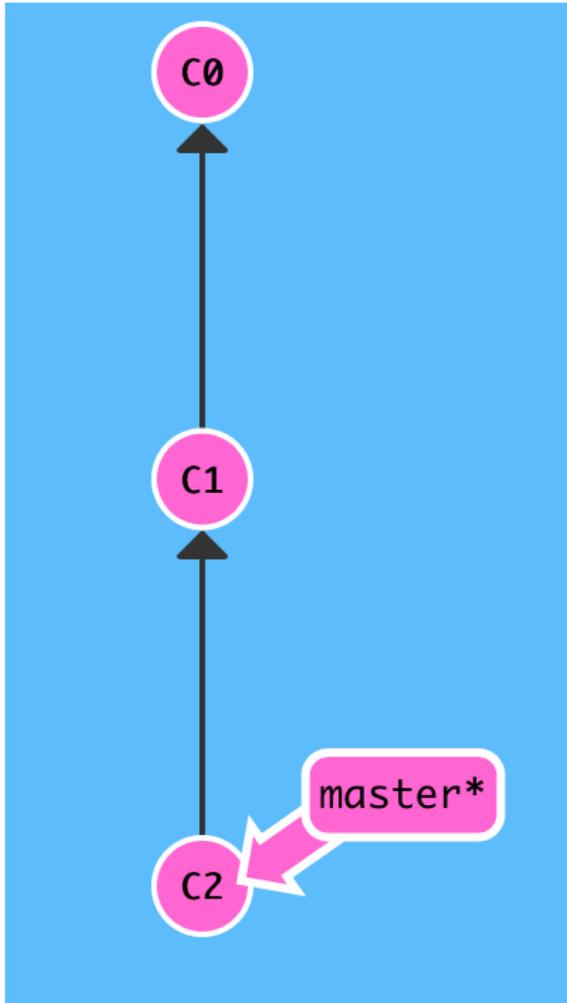


# Activity:

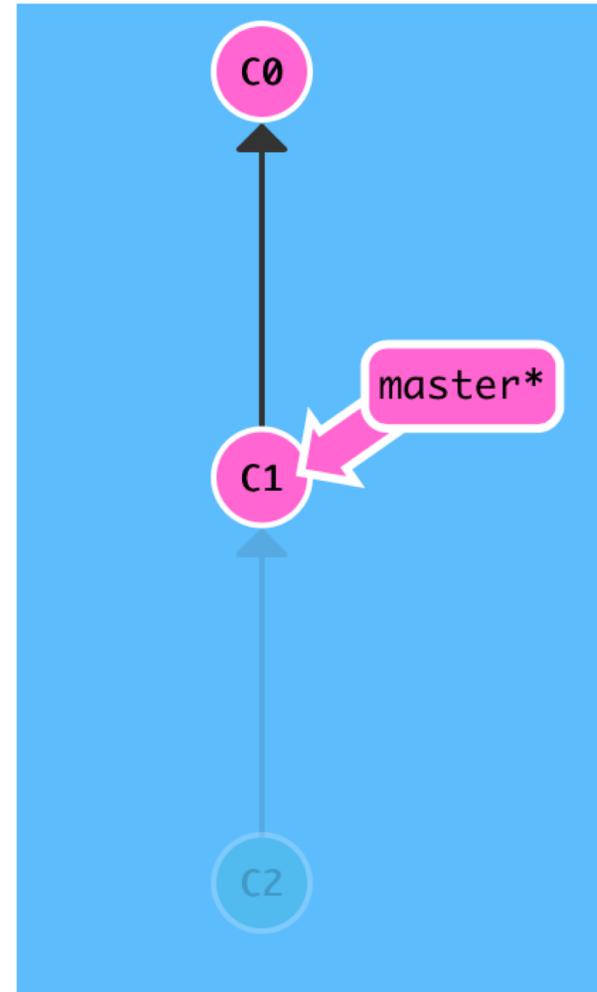


Ways to undo work (1)

```
git reset HEAD~1
```

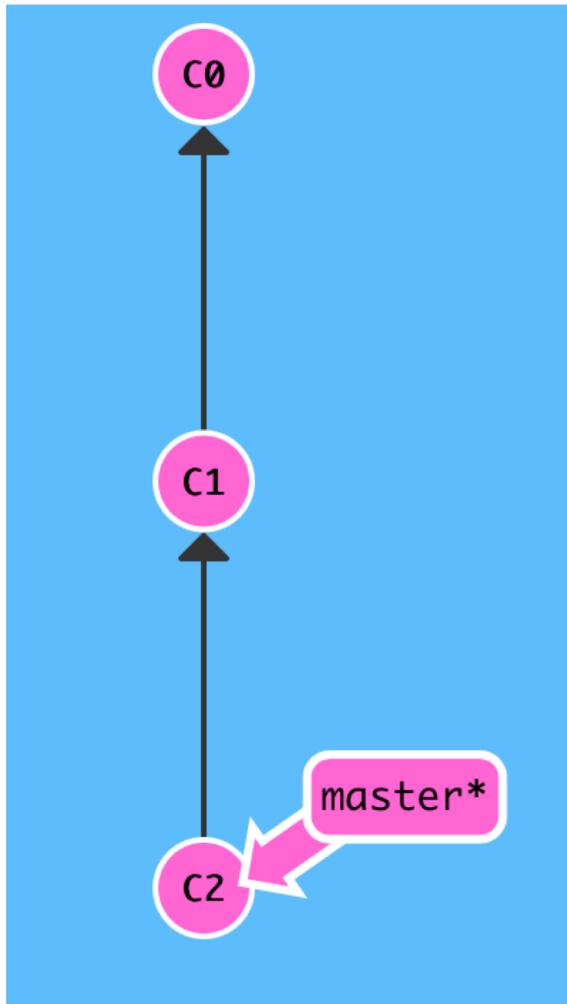


HEAD is the symbolic name for the currently checked out commit

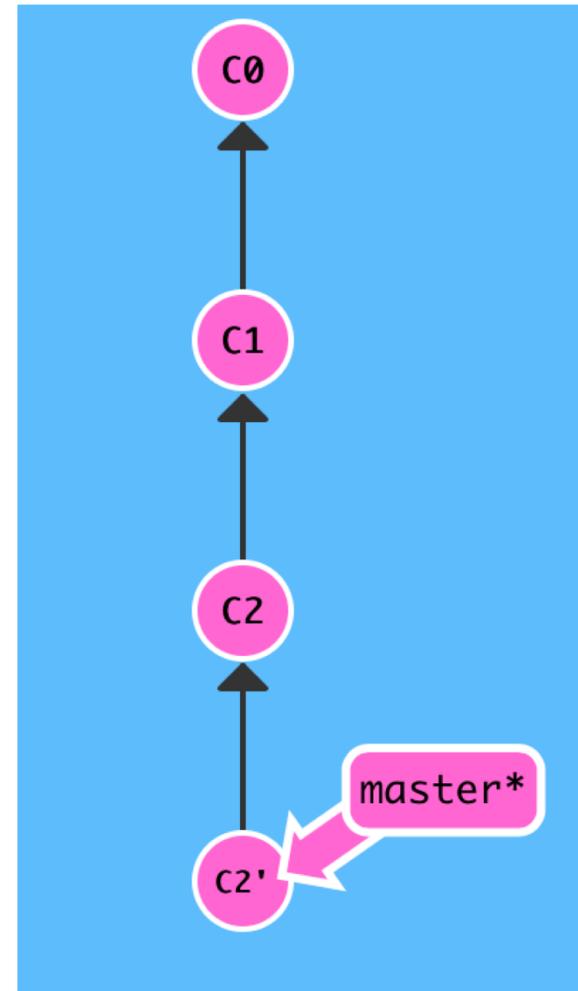


Ways to undo work (2)

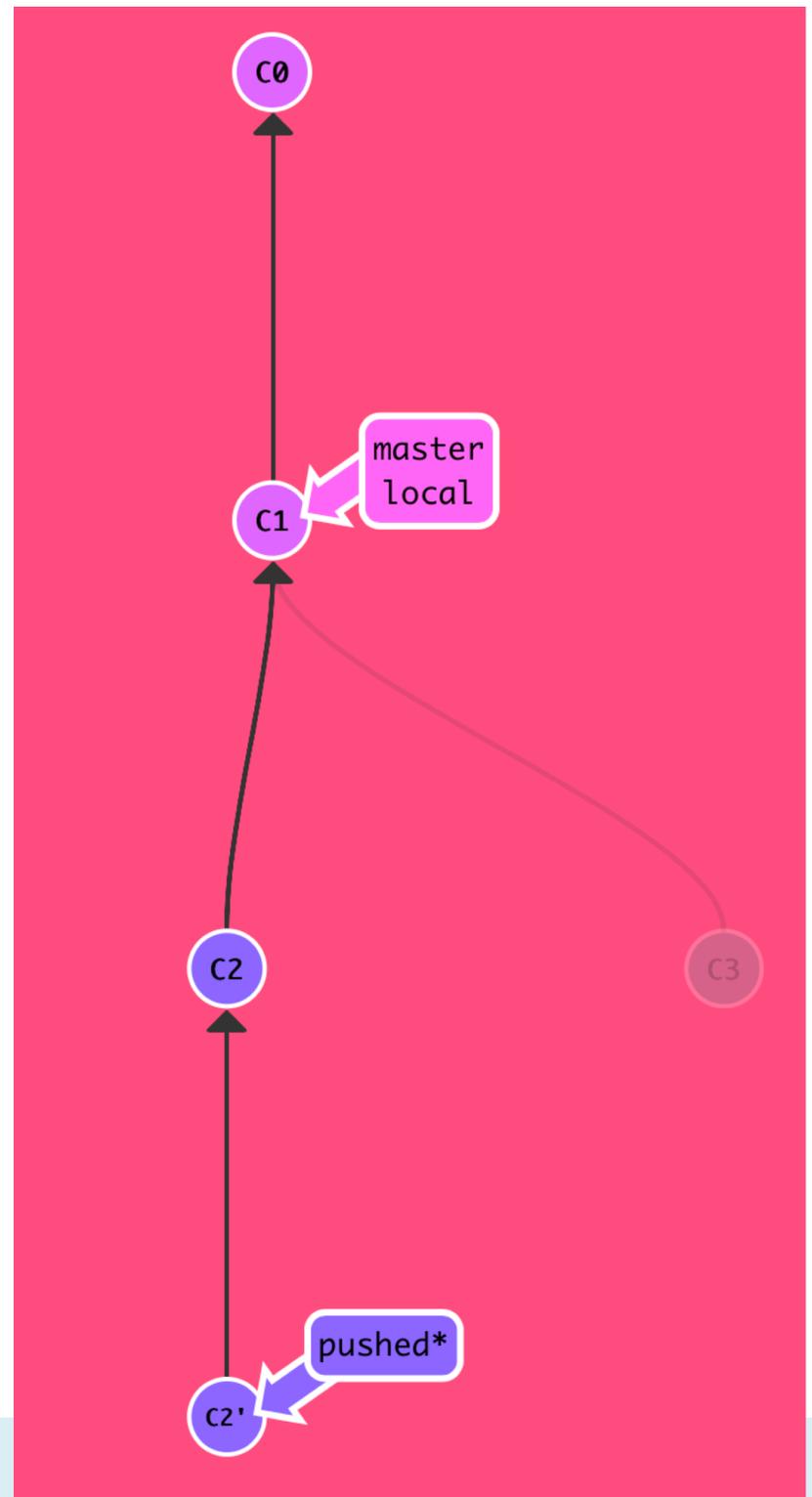
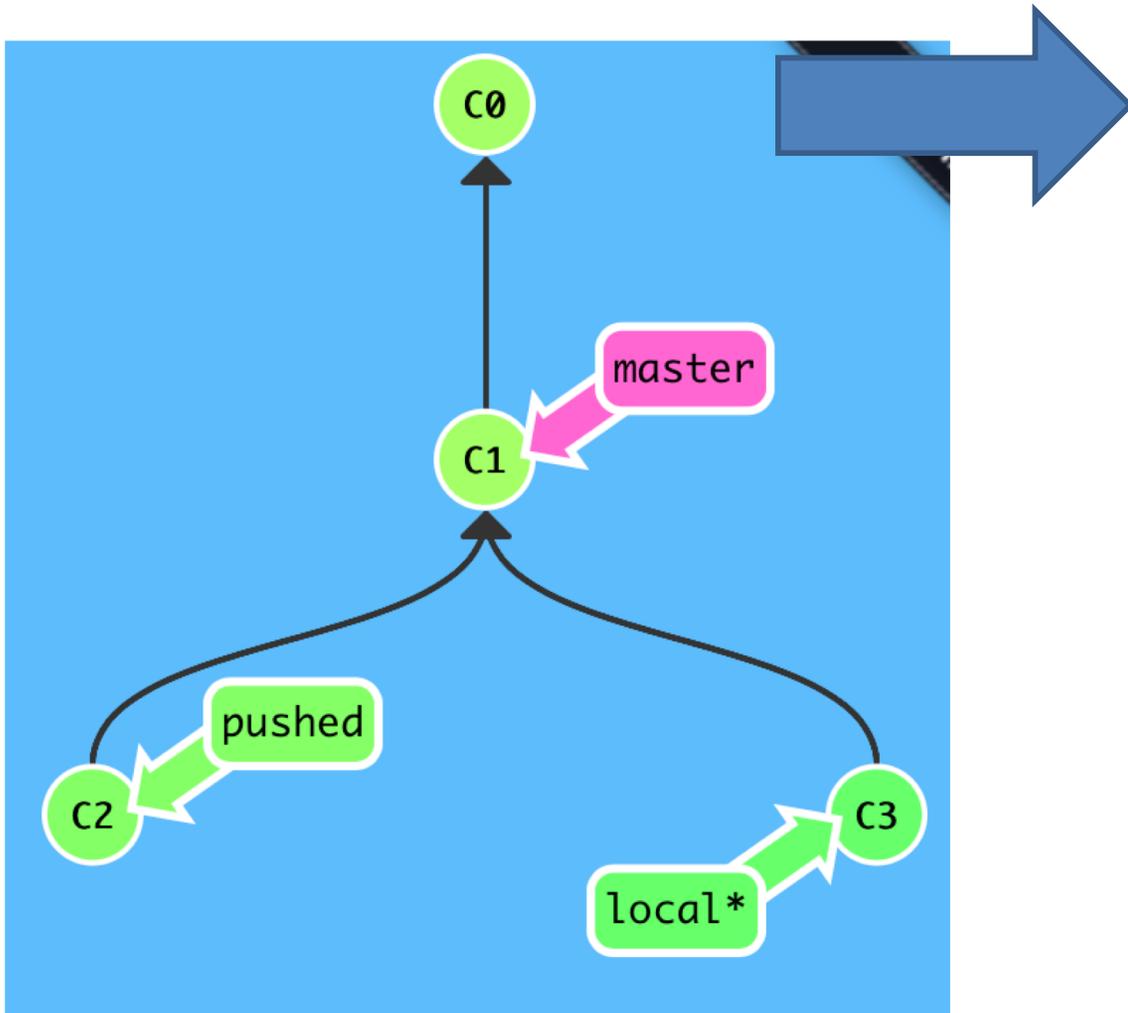
`git revert HEAD`



`git reset` does not work  
for remote branches

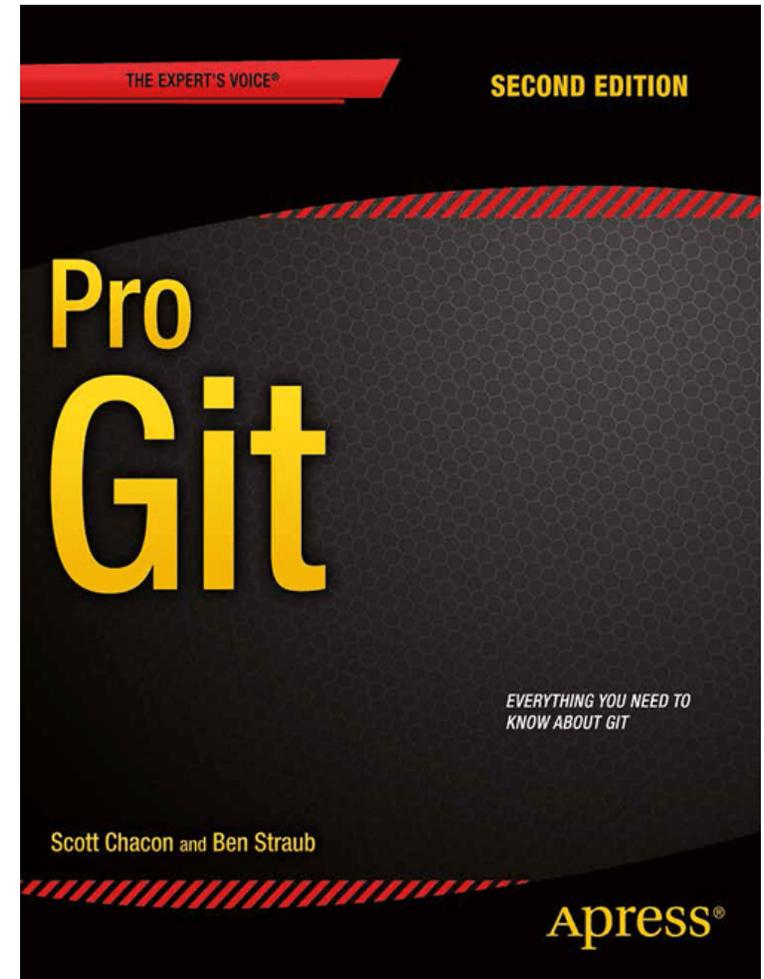


# Activity:



# Highly recommended

- (second) most useful life skill you will have learned in 214

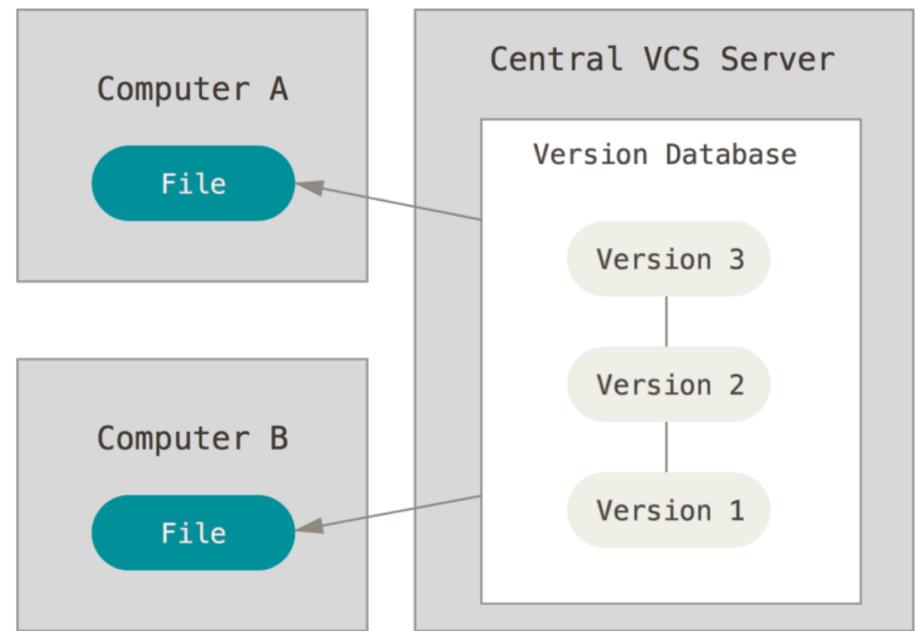


<https://git-scm.com/book/en/v2>

# TYPES OF VERSION CONTROL

# Centralized version control

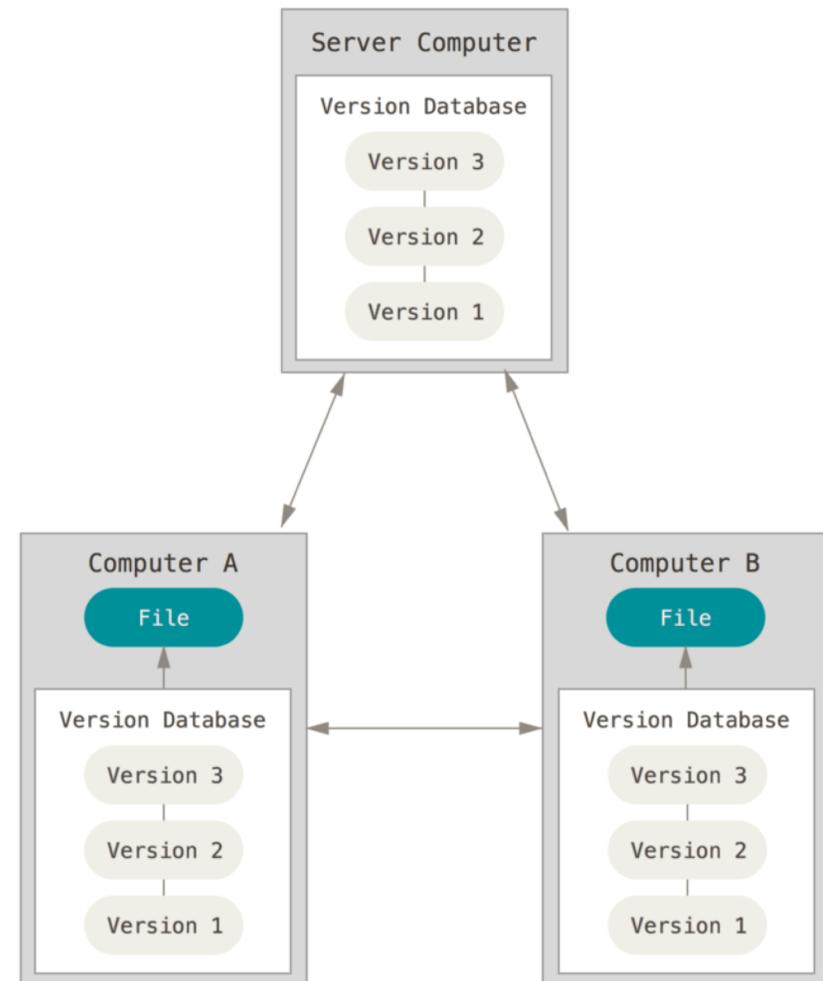
- Single server that contains all the versioned files
- Clients check out/in files from that central place
- E.g., CVS, SVN (Subversion), and Perforce



<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

# Distributed version control

- Clients fully mirror the repository
  - Every clone is a full backup of *all* the data
- E.g., Git, Mercurial, Bazaar

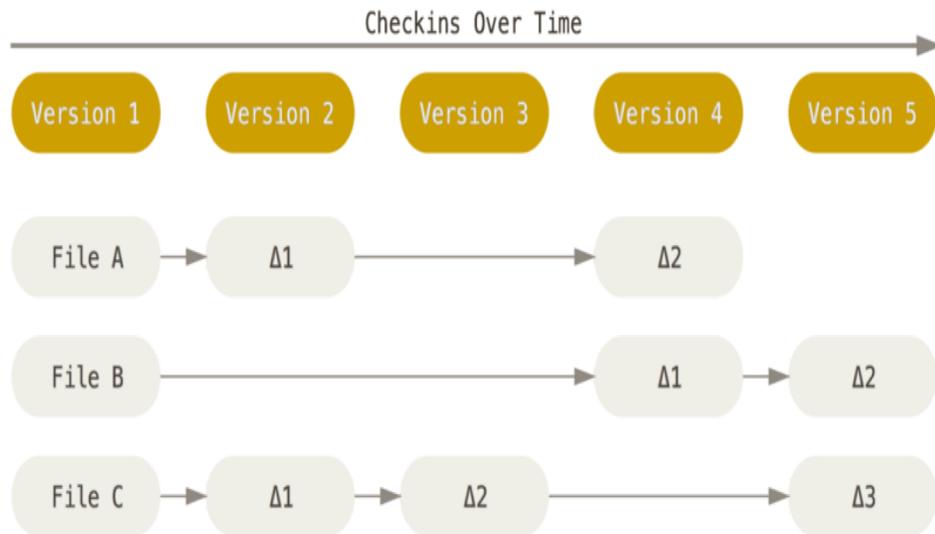


<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

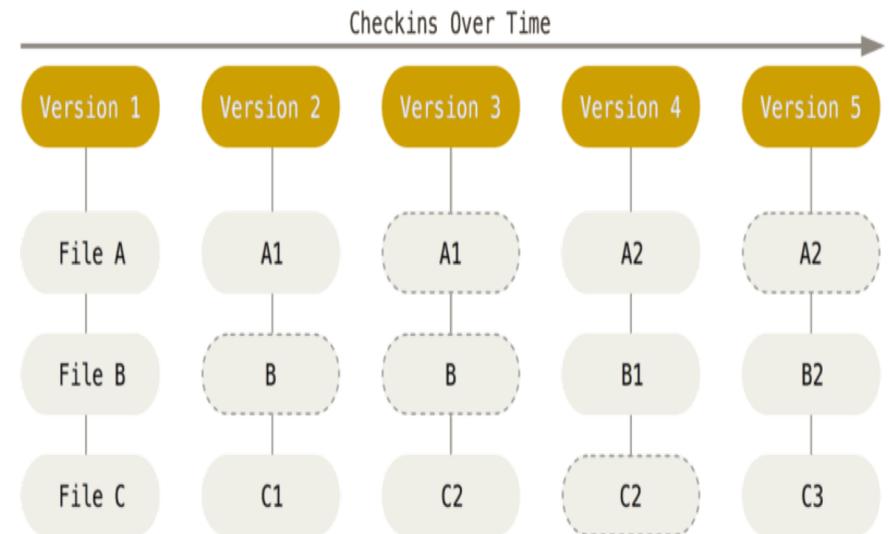
## Activity

- In pairs, discuss advantages and disadvantages of centralized (e.g., SVN) vs decentralized (e.g., git) version control

## Aside: Internals SVN (left) vs. Git (right)



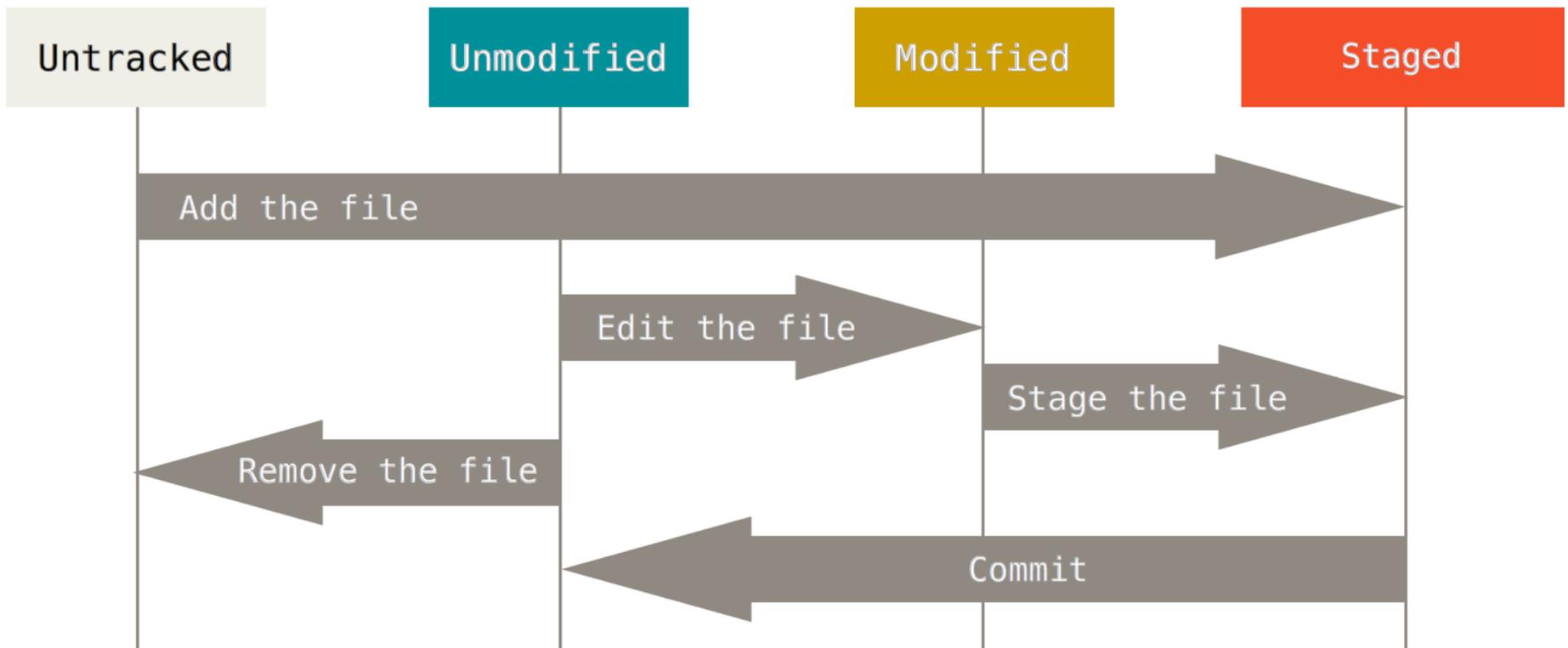
- SVN stores changes to a base version of each file
- Version numbers (1, 2, 3, ...) are increased by one after each commit



- Git stores each version as a snapshot
- If files have not changed, only a link to the previous file is stored
- Each version is referred by the SHA-1 hash of the contents

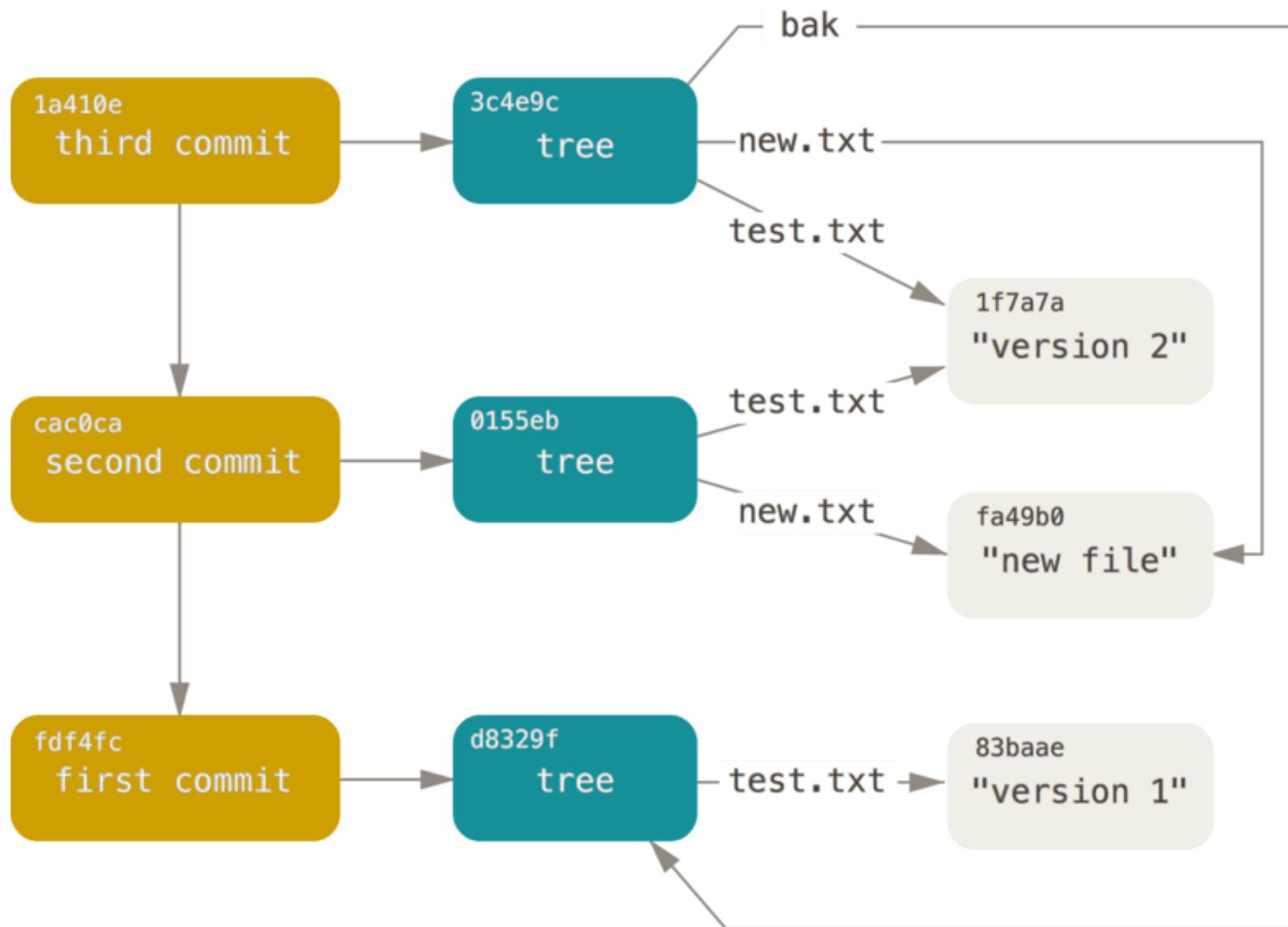
<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

# Aside: Git process



© Scott Chacon "Pro Git"

# Aside: Git object graph



© Scott Chacon "Pro Git"

## Aside: Which files to manage

- All code and noncode files
  - Java code
  - Build scripts
  - Documentation
- Exclude generated files (.class, ...)
- Most version control systems have a mechanism to exclude files (e.g., .gitignore)

# Summary

- Version control has many advantages
  - History, traceability, versioning
  - Collaborative and parallel development
- Locking vs. merging and merge conflicts
- Collaboration with branches
- From local to central to distributed version control