

# Version Control

EECS 2311 - Software Development Project

Click to edit Master slide styles

Second level

Third level

F

Fifth level

January 16, 2019

# TalkBoxConfiguration interface

- Your software must serialize and deserialize an object that implements the TalkBoxConfiguration interface
- The interface is posted on the course webpage (wiki) under **Project**
  - Read carefully under **Important Requirement**
- We will discuss it next week

# Scenario 1

- You finished the assignment at home
- You get to York to submit and realize you did not upload it
- Has this ever happened to you?

# Scenario 2

- Your program works pretty well
- You make a lot of improvements ...
  - ...but you haven't gotten them to work yet
- You need to demo your program *now*

# Scenario 3

- You are working on the 2.0 version of “your great app.” But 2.0 does not quite compile yet... and customer finds a critical bug in 1.0, which must be fixed ASAP.
- If you're smart, you have a copy of your 1.0 source. You make the change and release, but how do you merge your changes into your 2.0 code?
- If you're not so smart, you have NO source code saved. You have no way to track down the bug, and you lose face until 2.0 is ready.

# Scenario 4

- You change one part of a program - it works
- Your teammate changes another part - it works
- You put them together - it does not work
  
- What were all the changes?

# Scenario 5

- You make a number of improvements to a class
- Your teammate makes a number of *different* improvements to the *same* class
- How can you merge these changes?

# A poor solution

- There are a number of tools that help you spot changes (differences) between two files, such as diff
- Of course, they won't help unless you kept a copy of the older version
- Differencing tools are useful for finding a *small* number of differences in a *few* files
- A better solution...



# Version control systems

- Keep multiple (older and newer) versions of everything (not just source code)
- Request comments regarding every change
- Display differences between versions
- Allow merging of changes on the same file

# Centralized Version Control

- Traditional version control system
  - Server with database
  - Clients have a working version
- Examples
  - CVS
  - Subversion
- Challenges
  - Multi-developer conflicts
  - Client/server communication

# Distributed Version Control

- Authoritative server by convention only
- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial
- Examples
  - Git
  - Bitkeeper

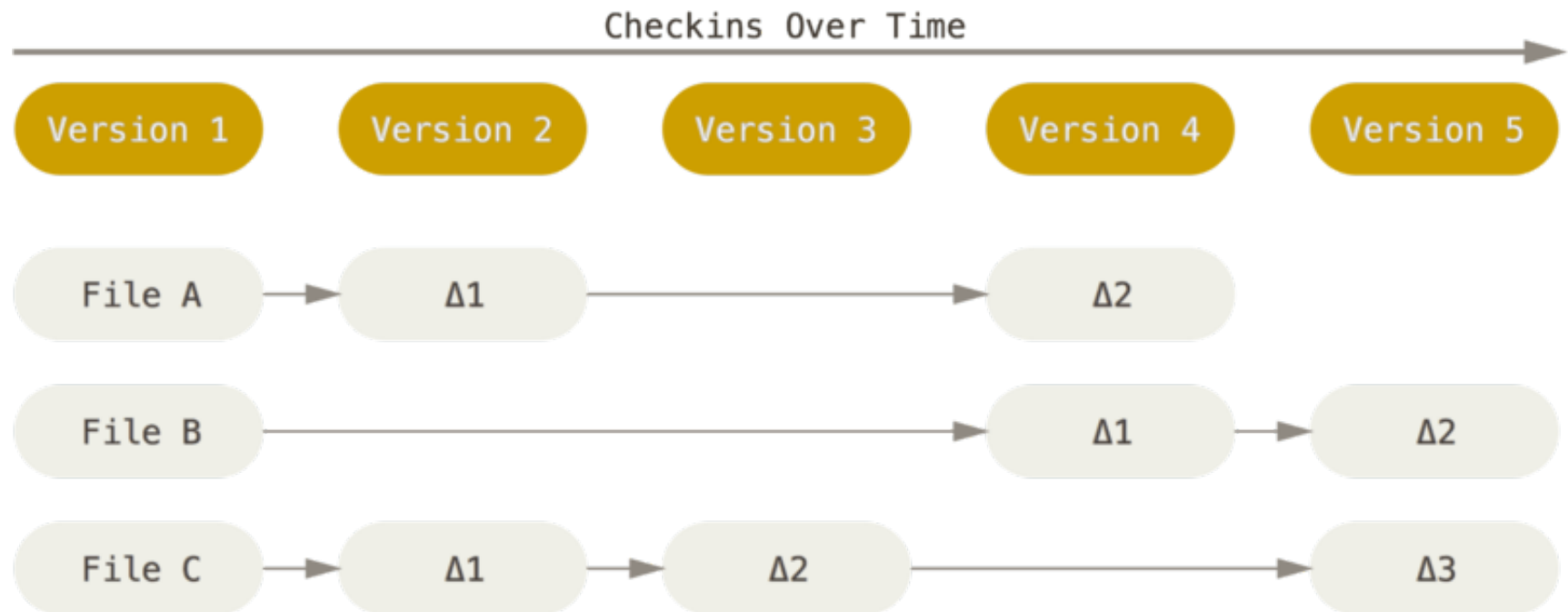
# Terminology

- A **repository** contains several branches
- The main branch is called the **master**
- **Branches** break off from the master to try something new, e.g. a new feature, code restructuring etc.
- Branches can be merged with other branches or into the master
- **Tags** are usually official releases that have to be supported

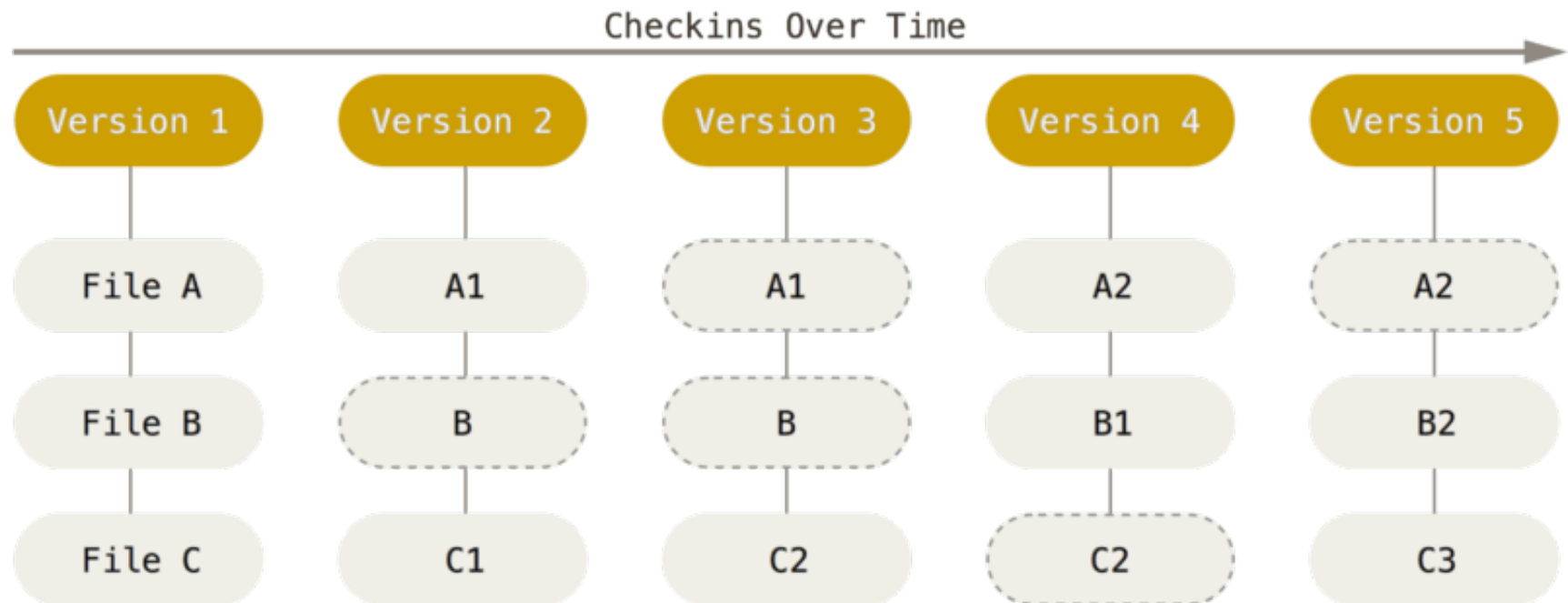
# Git

- Developed by Linus Torvalds and the Linux community starting in 2005
- Goals
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Can handle large projects like Linux
- The rest of these slides are based on the excellent Pro Git book ([link on course website](#))

# Before Git: Delta storage



# Git: Snapshot storage



# Git storage

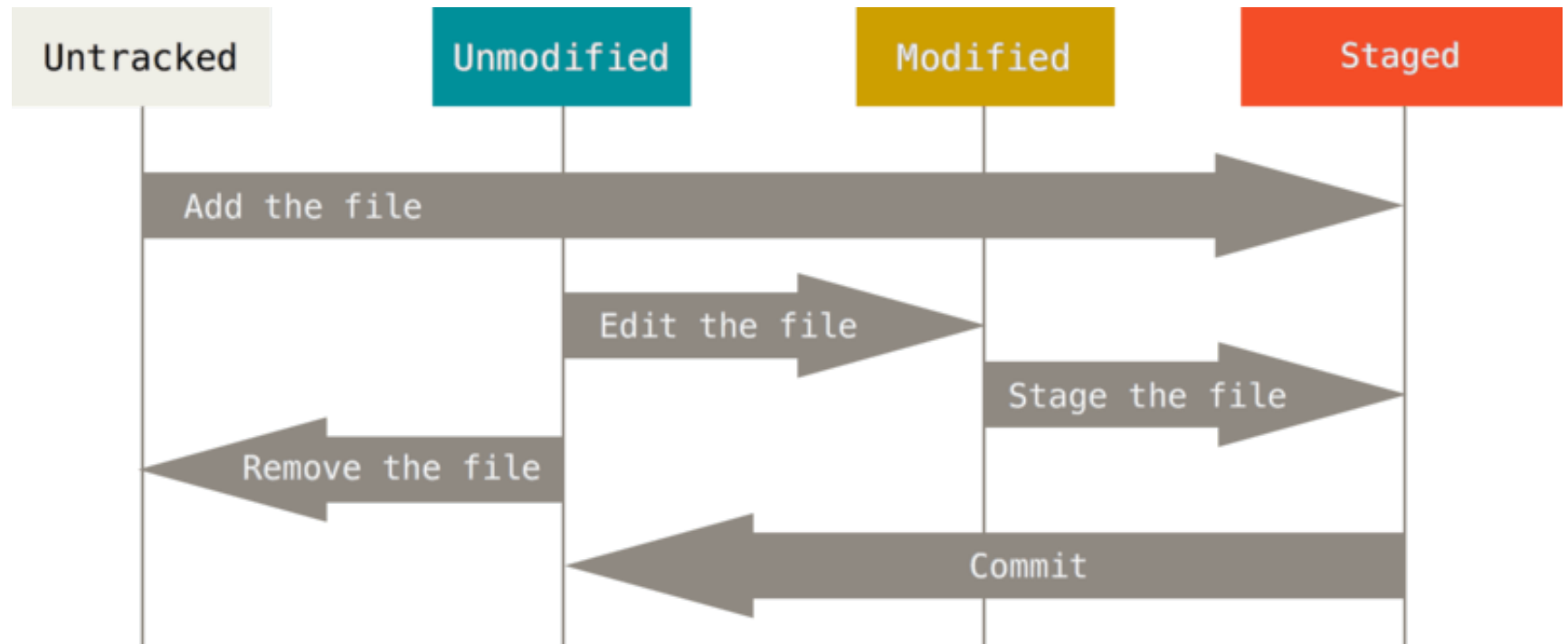
- More like a miniature filesystem
- Makes for some very fast operations
- Beneficial when we get to branching



# File States in Git

- **Committed** means that the data is safely stored in your local repository. Also called **Unmodified**
- **Modified** means that you have changed the file but have not committed it to your repository yet.
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
- **Untracked** means that Git will not include the file in any snapshot

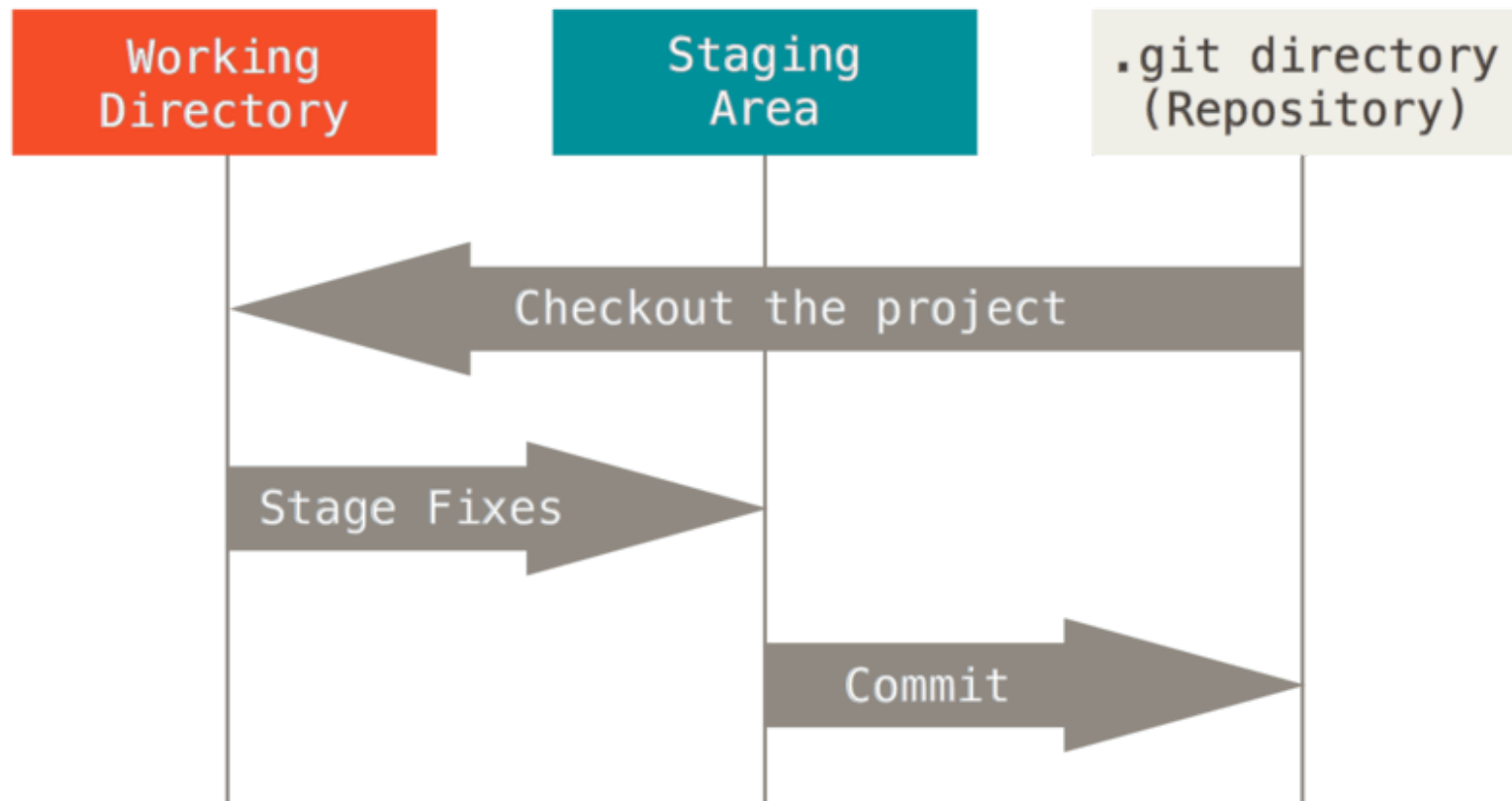
# File States in Git



# Git terminology

- **Working directory:** a single checkout of one version of the project
- **Staging area:** a file that stores information about what will go into the next commit. Also called **index**
- **.git directory:** The actual repository. Contains metadata and the object database of your project

# Basic Git workflow



# Basic Git workflow

- You **modify** files in your working directory.
- You selectively **stage** just those changes you want to be part of your next commit, which adds only those changes to the staging area.
- You do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your `.git` directory.

# Remote Repositories

- So far, everything has been local to your computer
- To collaborate with others (and to ensure backup), you need to set up a **remote repository**
- When we cloned your Github repository in Eclipse, we established one such remote repository
- Your default remote repository is referred to as **origin**
- It's possible to have multiple remote repositories

# Remote Repository Operations

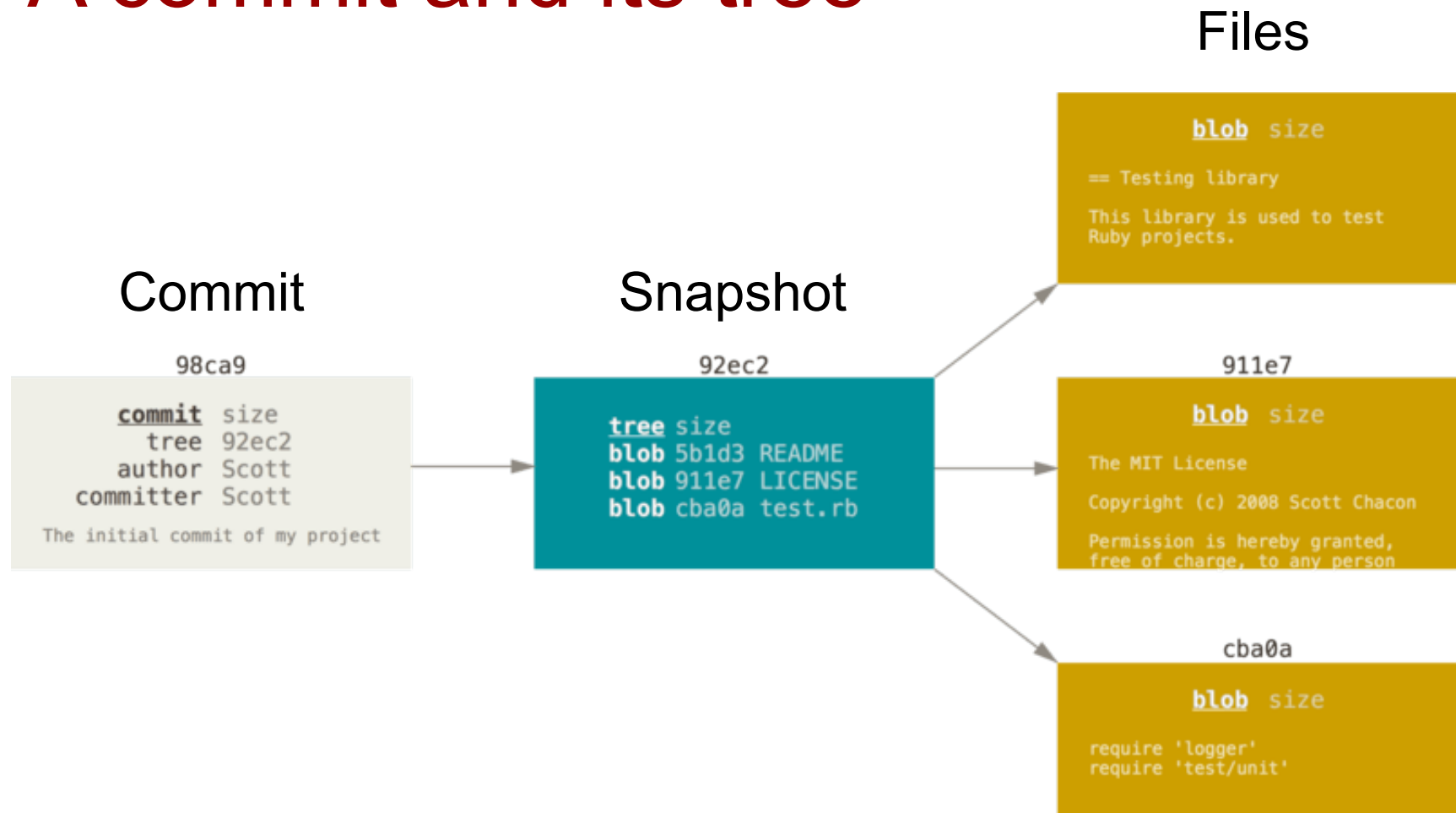
- **Fetch:** Downloads data from the remote repository, i.e. any changes your teammates have uploaded. Does not merge with your local repository.
- **Pull:** Fetches and then merges with your local repository. In many cases, this is all you need.
- **Push:** When you have a commit in your local repository that you would like to share, use Push to upload your code to the remote repository.

# Git branching

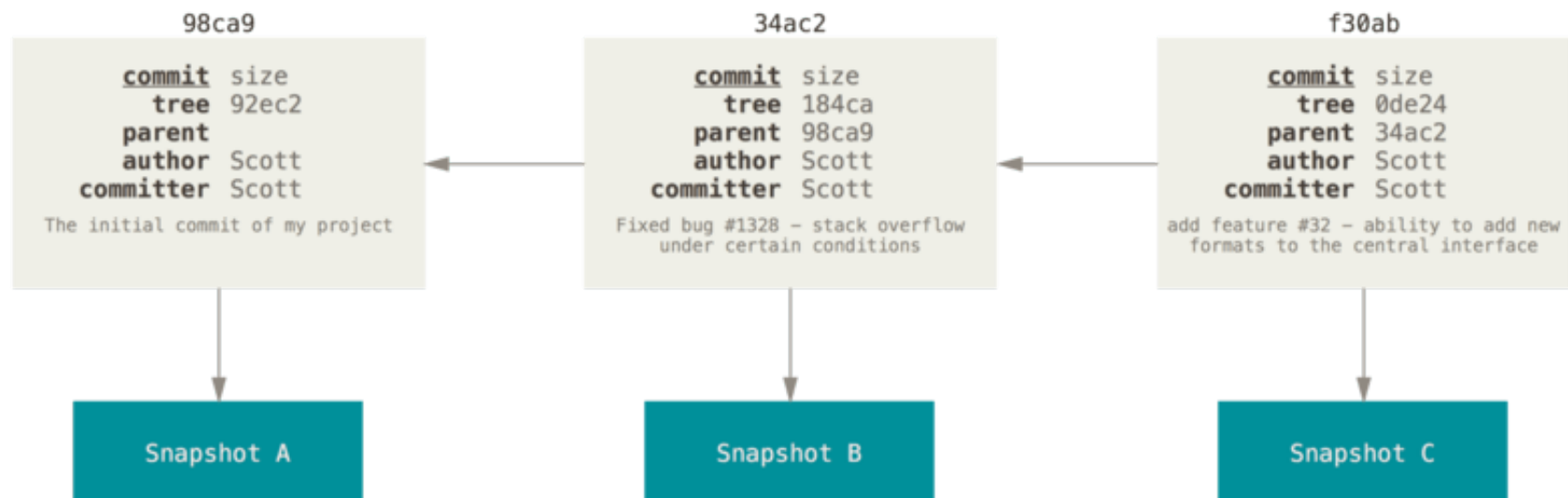
- Your main line of development in your project is called the **master** branch
- You can create other branches to try out an idea without affecting your teammates
- If the idea works out, you can **merge** your branch back into the master branch
- Git provides powerful support for this process



# A commit and its tree



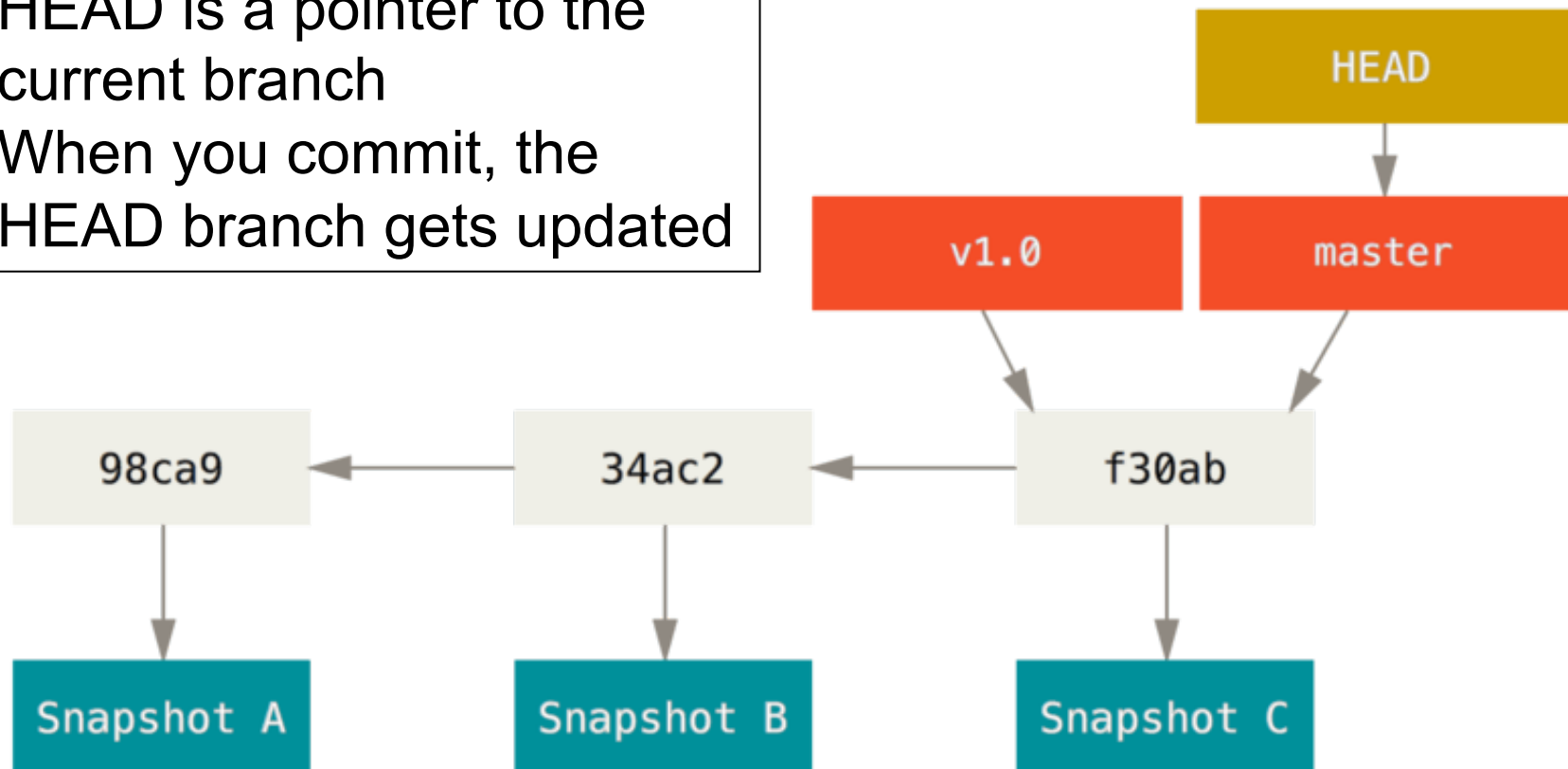
# A series of commits



Each commit has a pointer to the previous commit

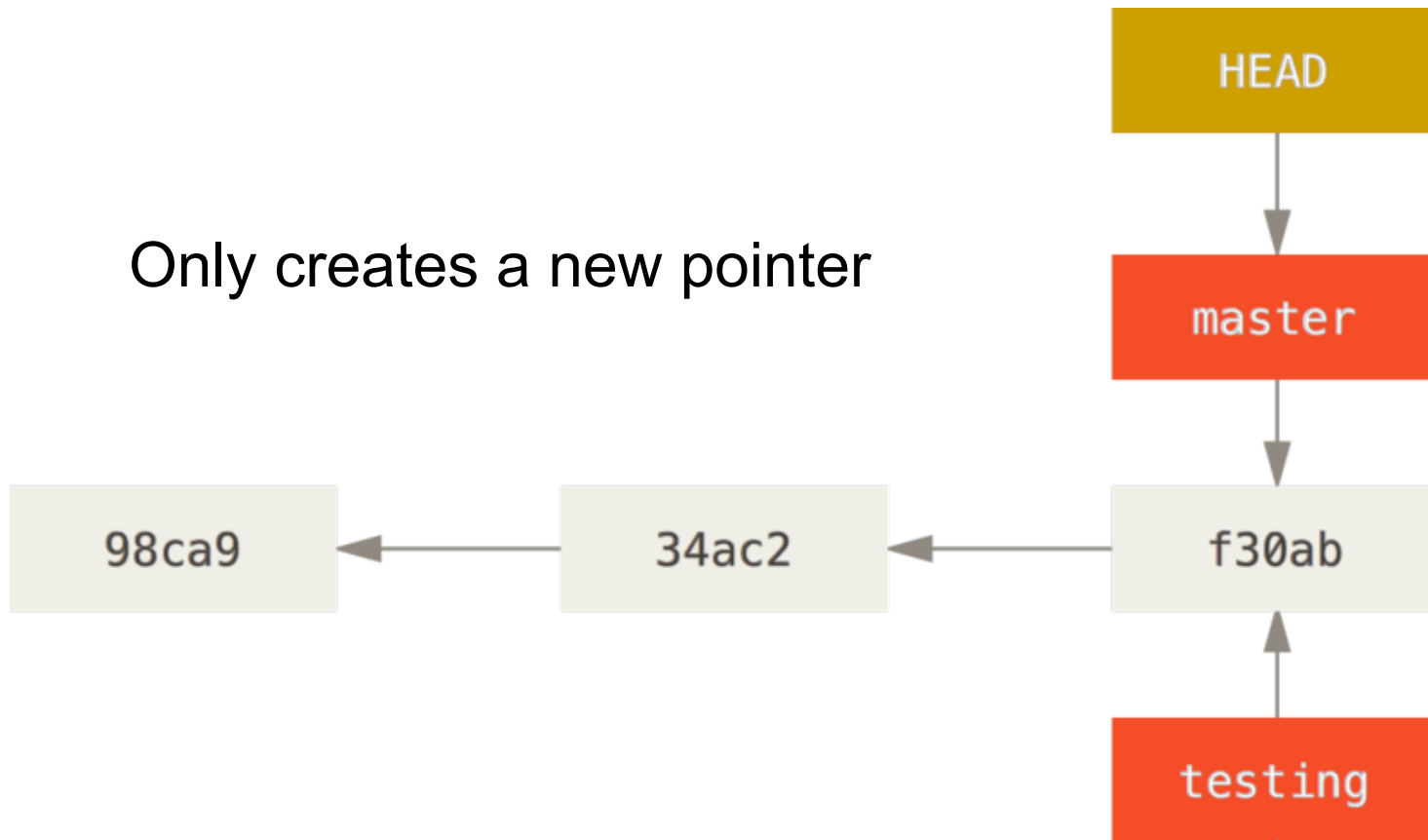
# A branch is only a pointer

HEAD is a pointer to the current branch  
When you commit, the HEAD branch gets updated

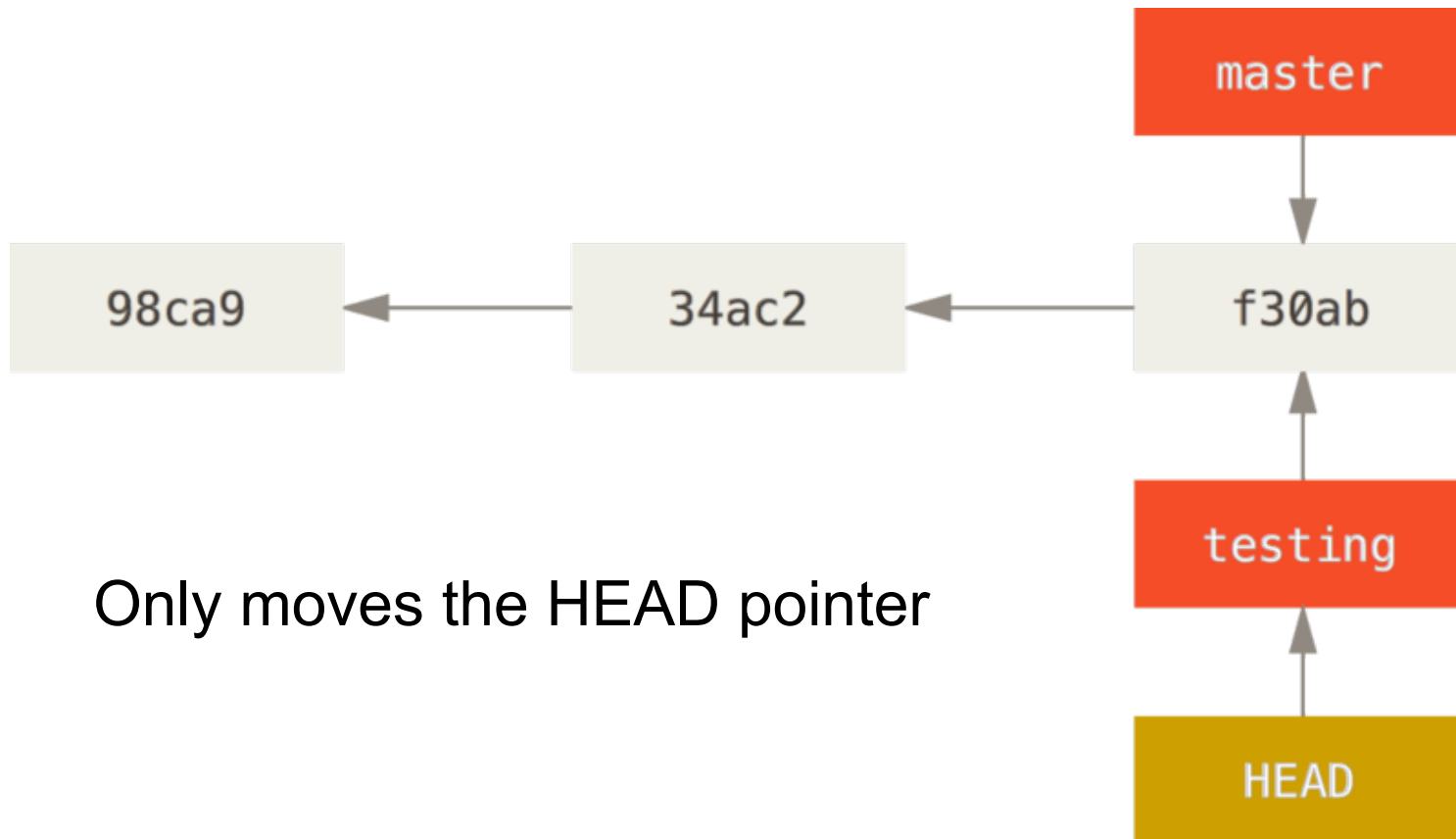


# Create a new branch

Only creates a new pointer

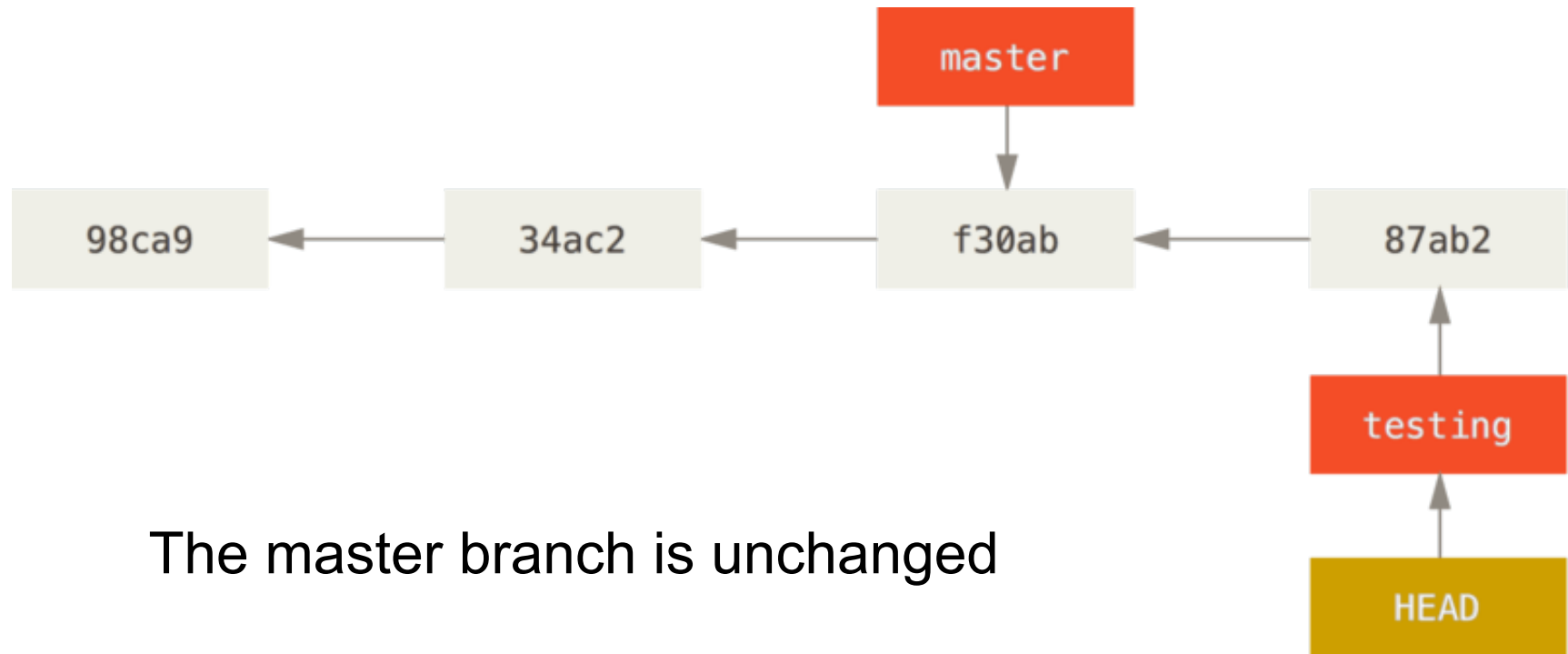


# Switch to the new branch



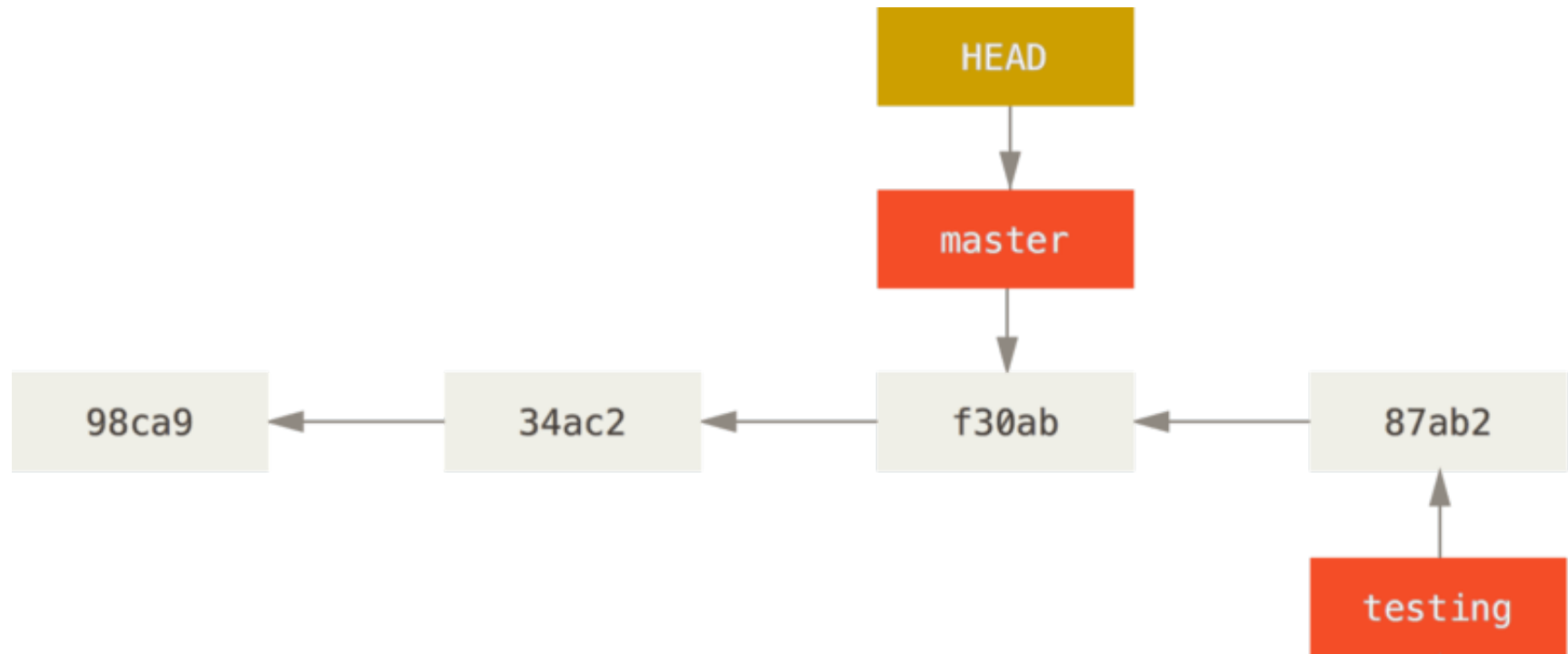
Only moves the HEAD pointer

# Commit to the new branch



The master branch is unchanged

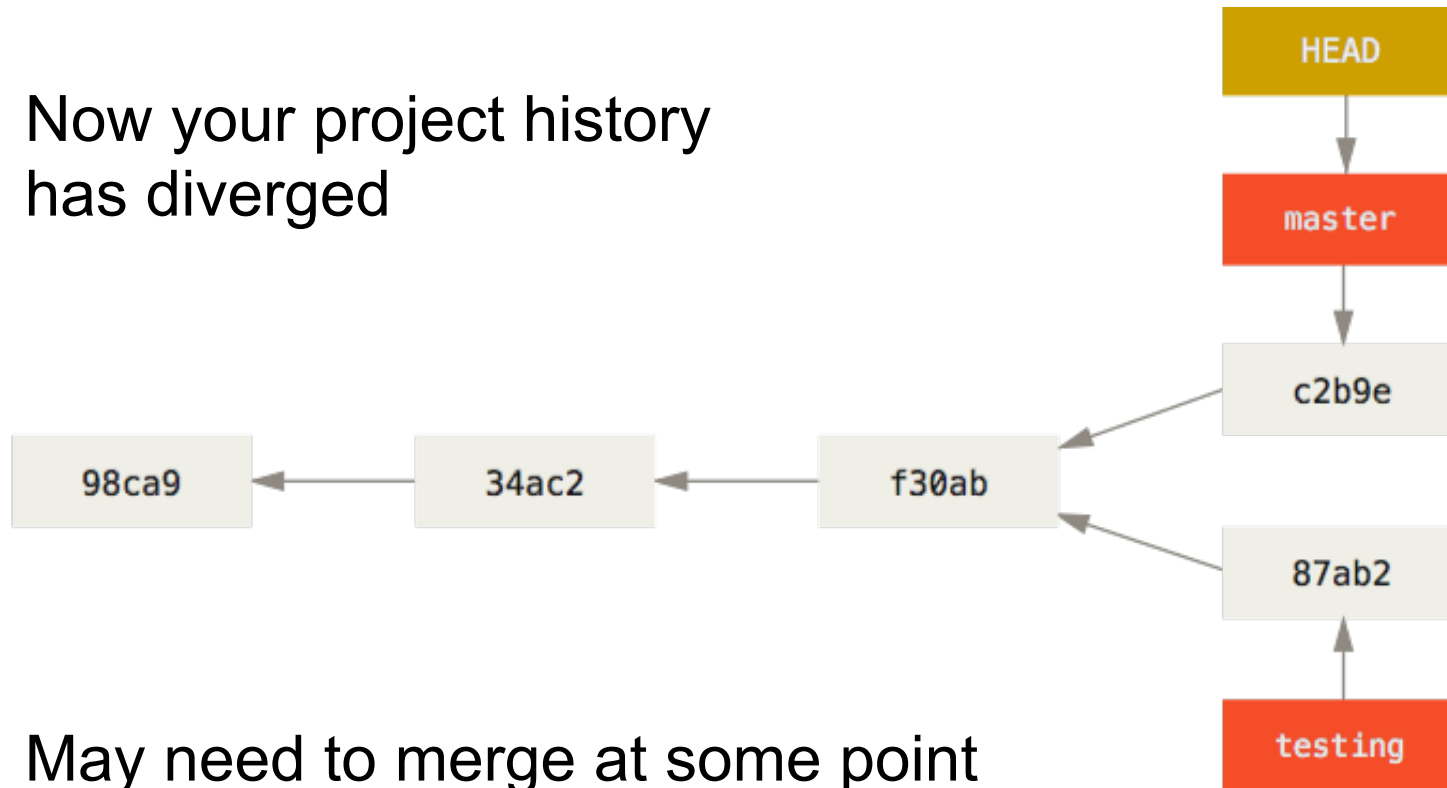
# Switch back to the master branch



Only moves the HEAD pointer

# Commit to the master branch

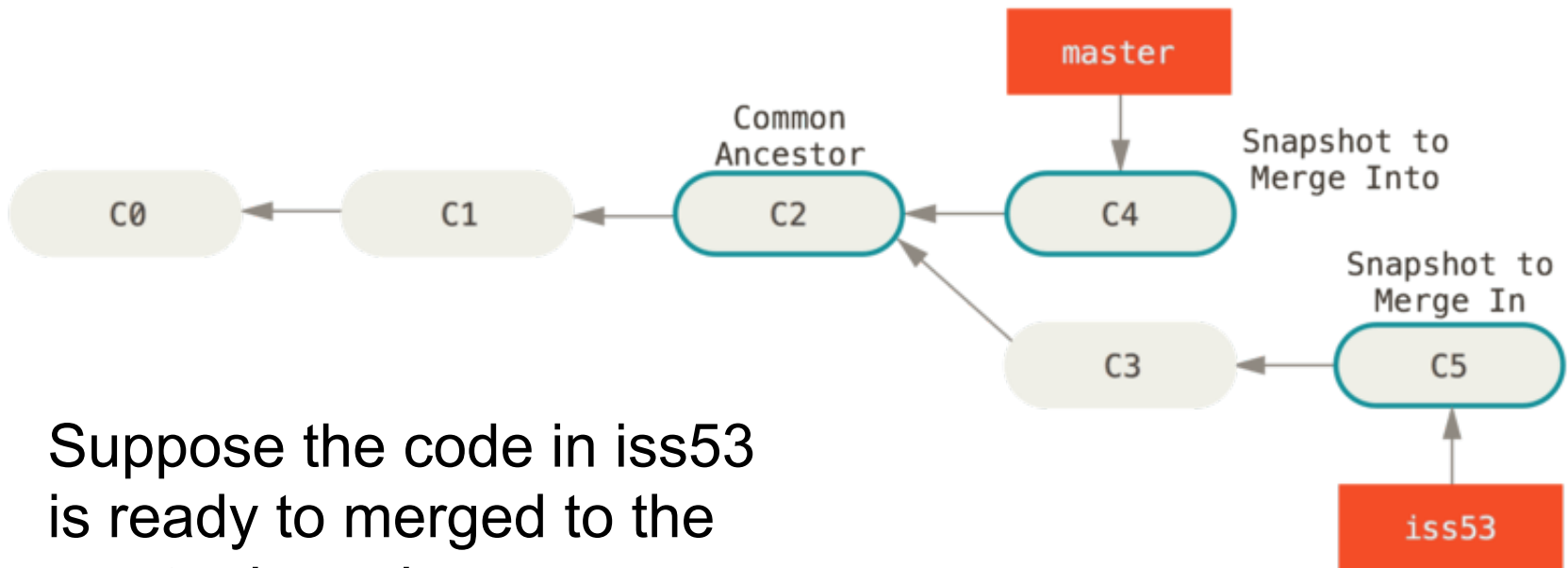
Now your project history has diverged



May need to merge at some point



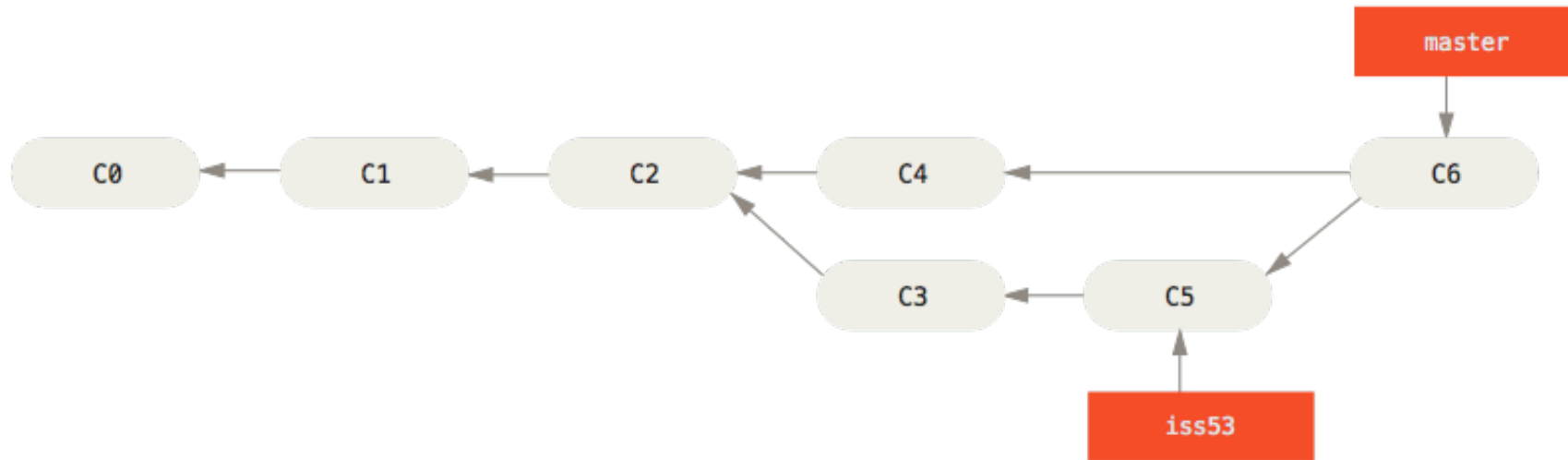
# Merging



Suppose the code in iss53 is ready to be merged to the master branch

# A merge commit

A merge commit has two previous commits

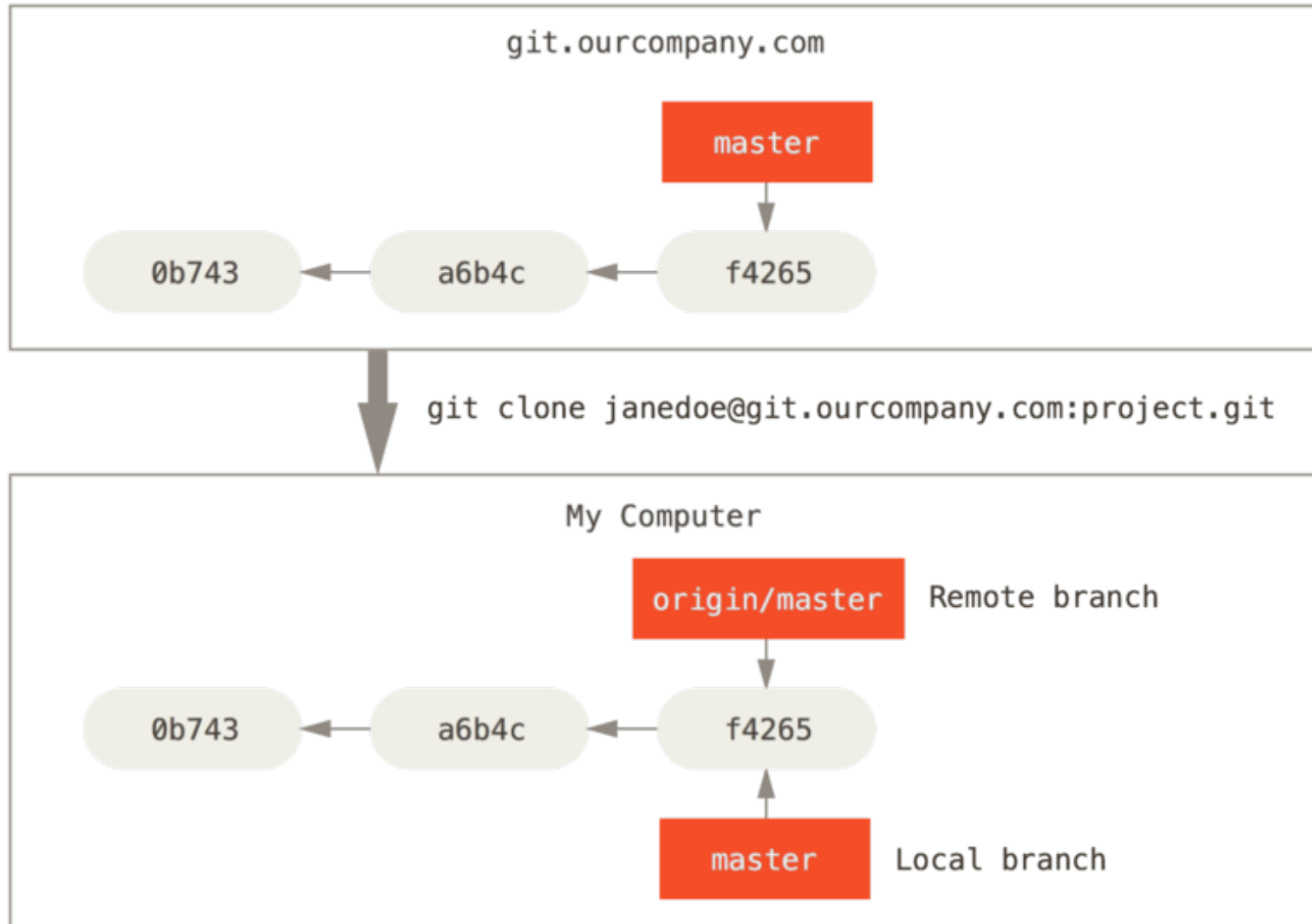


Branch iss53 can now be deleted

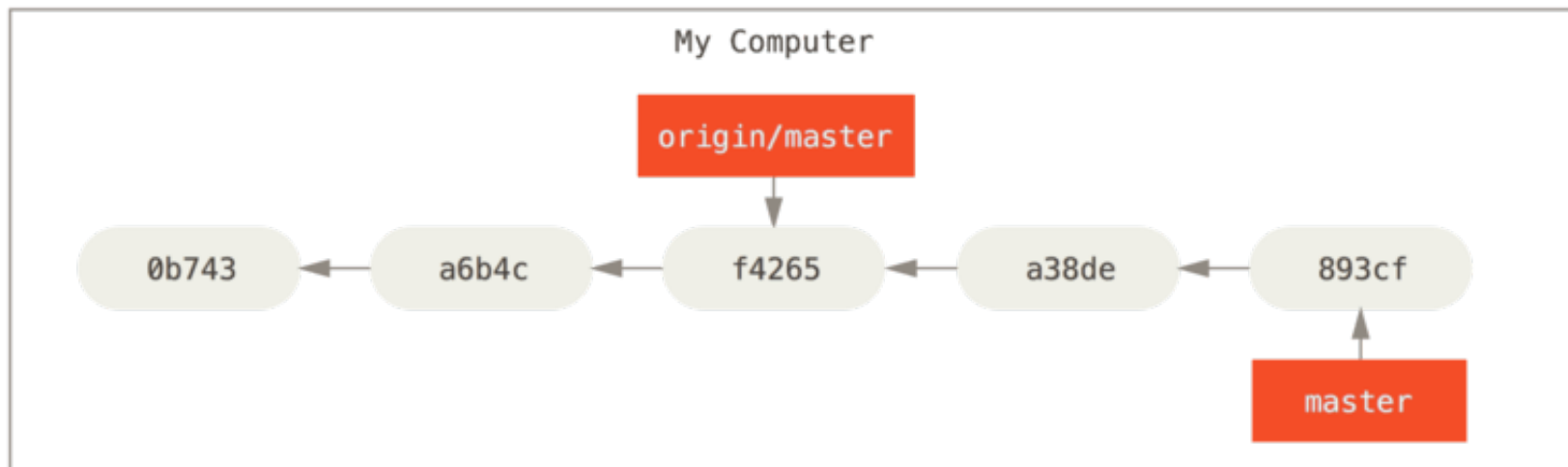
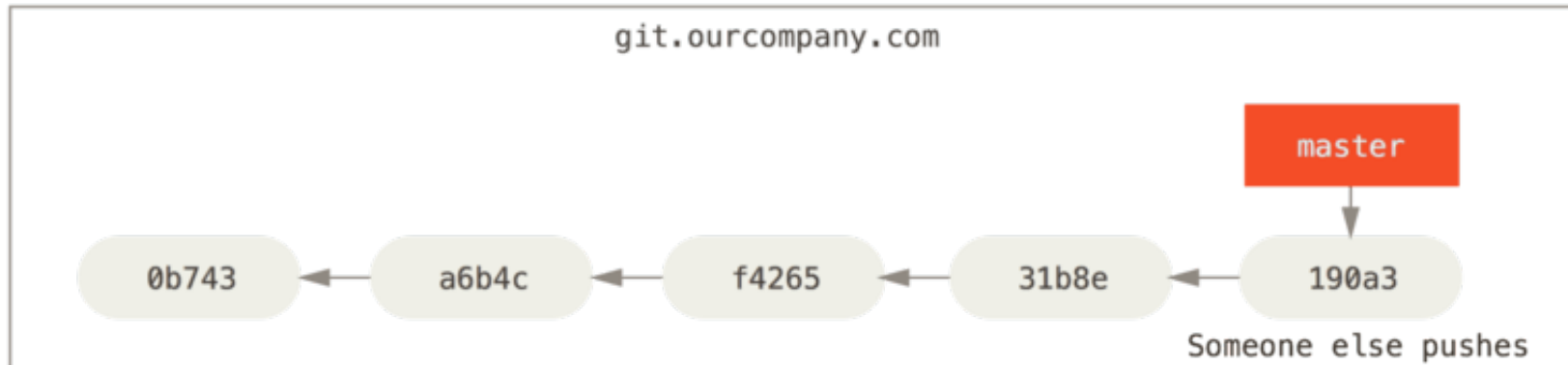
# Merge conflicts

- If git cannot merge the two branches because changes have been made to the same part of a file, it will present options to choose one of the two versions or even create a new version on the spot.

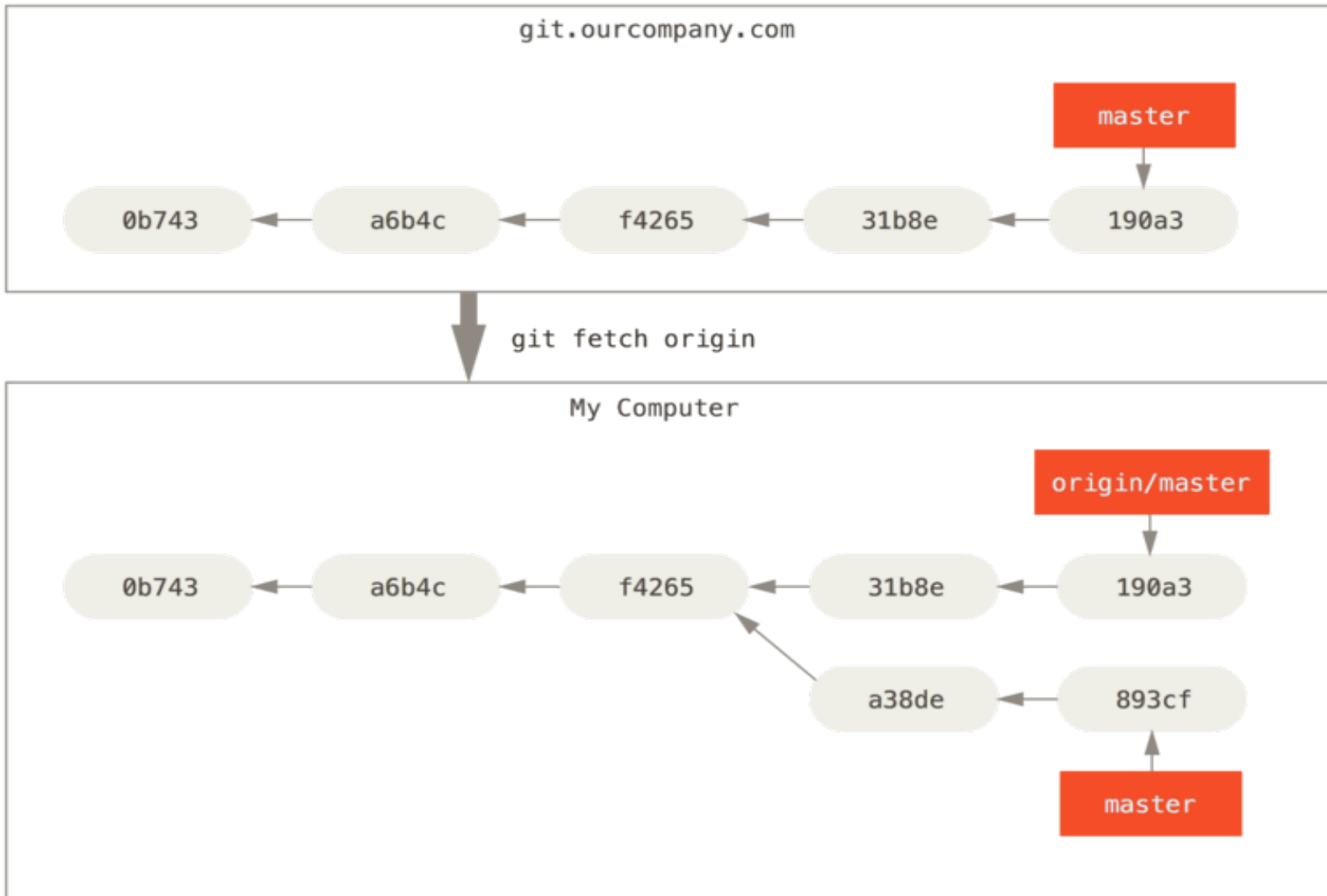
# Remote branches



# After some local work...



# When you fetch...



# Git vs. EGit

- Git has a powerful command-line implementation
- Can be used to apply version control to any files, not necessarily code
- Eclipse has EGit, a GUI implementation of git
- Not as powerful, but it provides most of the functionality of git
- See the documentation under Help -> Help Contents
- Tutorial link on Course Website

# EGit in-class demo

- See the history of a file
- Compare versions
- Create a new branch and make some changes
- Switch to master branch and make conflicting changes
- Attempt to merge and resolve conflict



# Lab Task

- Demonstrate to the TAs that you can create different branches, make changes, and resolve conflicts
- The conflicting versions must be created by two different team members and pushed to the team repository
- You must also demonstrate a first version of the TalkBox Simulator app that has functional buttons
  - When the buttons are clicked, an audio file is played
- Next week I will expect a first version for both apps, so get started!