Version Control Workshop: Git and GitHub (Day 1)

Cyrus Vandrevala [1]
Nicolás Guarín-Zapata[2]
[1] Physics Department
[2] Civil Engineering Department

October 30-31, 2014

Overview

We Encourage Participation!

- Post Questions That You Might Have in the Repo
- Recommend Other Sources That You Found Useful
- Remember, We Do Not Know Everything!

What is Version Control?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

- *Pro Git*, Chapter 1

What is Version Control?

It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

- *Pro Git*, Chapter 1

Why is Version Control Important?

1. Keep Track of Code History
2. Concurrent Teamwork
3. Coordinate Coding Environments
4. Due Diligence Checks
5. Share Code

Everybody Should Use Version Control!

Why is Version Control Important?

1. Keep Track of Code History
2. Concurrent Teamwork
3. Coordinate Coding Environments
4. Due Diligence Checks
5. Share Code

Everybody Should Use Version Control!

Why is Version Control Important?

1. Keep Track of Code History
2. Concurrent Teamwork
3. Coordinate Coding Environments
4. Due Diligence Checks
5. Share Code

Everybody Should Use Version Control!

Why is Version Control Important?

1. Keep Track of Code History
2. Concurrent Teamwork
3. Coordinate Coding Environments
4. Due Diligence Checks
5. Share Code

Everybody Should Use Version Control!

Why is Version Control Important?

1. Keep Track of Code History
2. Concurrent Teamwork
3. Coordinate Coding Environments
4. Due Diligence Checks
5. Share Code

Everybody Should Use Version Control!

Why is Version Control Important?

1. Keep Track of Code History
2. Concurrent Teamwork
3. Coordinate Coding Environments
4. Due Diligence Checks
5. Share Code

Everybody Should Use Version Control!

What Options Are Available?

Option #1: Client-Server Version Control Systems

Advantages

1. A Single Admin Keeps Track of the Project
2. There is a Single Master Version of the Code
3. It is Relatively Easy to Learn

Disadvantages

1. There Is Only One Admin/Server
2. You Need a Network Connection to Work
3. Operations Can Be Slow

Examples include Concurrent Versions System (CVS) and Subversion (SVN).

What Options Are Available?

Option #2: Distributed Version Control Systems

Advantages

1. You Don't Need a Network Connection
2. Multiple Coding Environments
3. It Encourages Collaboration and Modularity

Disadvantages

1. Can Be Difficult to Learn
2. Teams Need to Talk About Conventions
3. It is Really Easy To Create Unorganized Code

Examples include Git, Mercurial, and Bazaar.

Why Git and GitHub?

1. It Keeps Track of Detailed Metadata (More Than Others)
2. Branching is Encouraged (Which Modularizes Development)
3. Most Operations in Git are Local (Which Increases Speed)
4. GitHub Has a Great Social Community

Why Git and GitHub?

Full Disclosure...

1. It Isn't the Best for Binary Files
2. GitHub Distinguishes Between Public and Private Repos

Version Control in Academia

1. It Creates Reproducible Research
2. It Helps Train New Group Members
3. It Encourages Collaboration
4. It Encourages Good Code Practices

More about reproducible research

This is a different topic. Nevertheless, is worth mentioning that Purdue is already taking action in this subject

- Purdue University Research Repository (PURR): `https://purr.purdue.edu/`

In the case of GitHub repositories, we can create DOI using `zenodo.org`

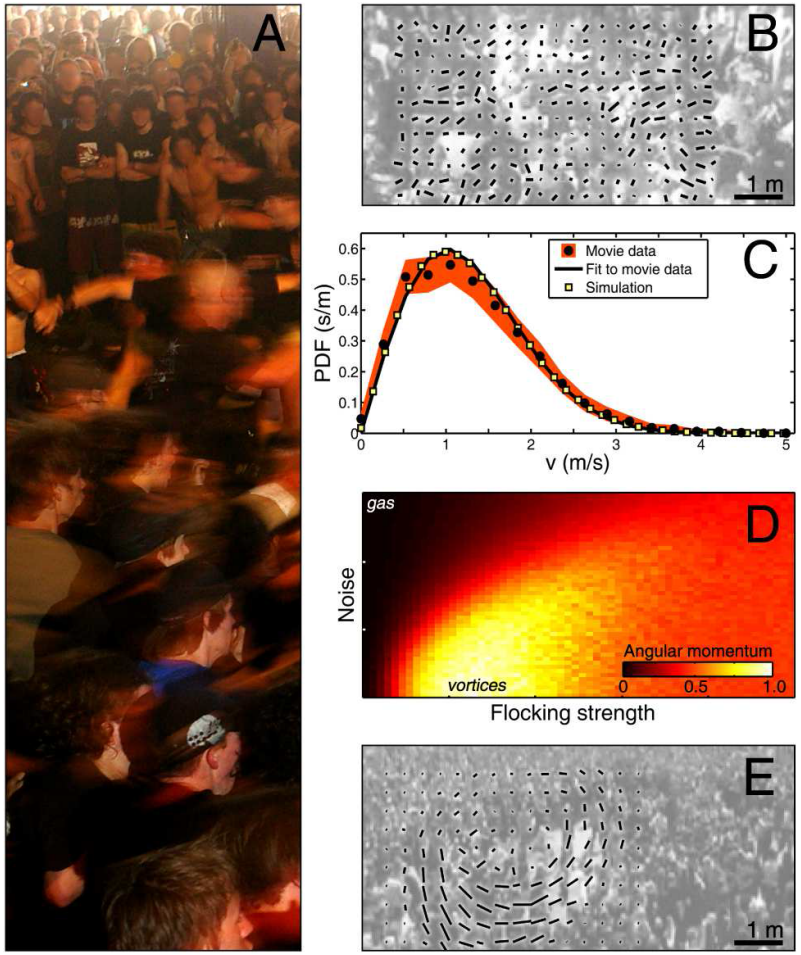
- A guide here: `https://guides.github.com/activities/citable-code/`

More about reproducible research

A cool example

Silverberg, Jesse L., et al. "Collective motion of humans in mosh and circle pits at heavy metal concerts." Physical review letters 110.22 (2013): 228701.

Repo: `https://github.com/mattbierbaum/moshpits`



Version Control in Academia

Some Useful Skills That You Should Learn Are:

1. Bash
2. Markdown

Setting Up Git - Linux

You can use the package management tool that comes with your distribution (use sudo):

1. yum install git
2. apt-get install git

Setting Up Git - Mac

There are three main ways to install Git:

1. Install the Xcode Command Line Tools and Type "git" Into the Terminal
2. Binary Installer: http://git-scm.com/download/mac
3. Git/GitHub GUI: https://mac.github.com/

Setting Up Git - Windows

There are three main ways to install Git:

1. Binary Installer: http://git-scm.com/download/win
2. msysGit: http://msysgit.github.io/
3. Git/GitHub GUI: https://windows.github.com/

Setting Up Git - Installing From Source

You can also install GitHub from source. See the Git website for full instructions on how to do that.

Setting Up Git - Config File

Git stores user information in */etc/gitconfig*, */.gitconfig*, and */your-project/.git/config*. To set up your information:

- *git config --global user.name "Cyrus Vandrevala"*
- *git config --global user.email cyrus.vandrevala@gmail.com*
- *git config --global core.editor vim*

Setting Up Git - Config File

You can double check the information you entered by using:

- *git config --list*

Setting Up a New Git Repo

1. Create a New Directory (mkdir my-awesome-directory)
2. Navigate Into the Directory (cd my-awesome-directory)
3. Initialize the Directory (git init)

The git init command creates a hidden directory called .git that contains all of the metadata for the project. *You should never change anything in .git directly!*

Retrieving an Existing Git Repo

1. Navigate to the Directory Where You Want to Store the Project
2. *git clone https://mydirectory.com/*

- Git supports many transfer protocols (including SSH)
- Remember, you are creating a standalone copy of the entire project.

The Basic Git Work Flow

Files in your project can be in one of three states:

1. Modified
2. Staged
3. Committed

The Basic Git Work Flow

1. Synchronize Your Repo (git pull)
2. Make Changes to Your Code
3. Stage Changes for Commit (git add)
4. Commit Changes Locally (git commit)
5. Push Changes to Origin (git push)

The Basic Git Work Flow

In order to determine which files are in which state, you can use (most to least detail):
git diff (unstaged changes only)
git status
git status -s

The Basic Git Work Flow

In order to get a full history of your commits, you can use:
git log Everything is labeled with a SHA-1 checksum.

The Basic Git Work Flow

In order to ignore certain files in your commits, you can change: *.gitignore* There are lots of .gitignore templates online at https://github.com/ github/ gitignore Standard glob patterns work.

The Basic Git Work Flow

Shortcuts:
git commit -m "My message" Commit with a message. *git commit -a -m "My message"* Commit without staging with a message.

What is Branching?

- Pretty much every version control system has some form of branching. This means that you diverge from the main line of development and continue to do work without changing the main line.
- Usually this is an expensive process because you have to copy all of the source code in the directory into a new branch.
- However, branching is where git truly shines. The git branch is extremely lightweight. This encourages branching in order to add new features.

What is Branching?

- Pretty much every version control system has some form of branching. This means that you diverge from the main line of development and continue to do work without changing the main line.
- Usually this is an expensive process because you have to copy all of the source code in the directory into a new branch.
- However, branching is where git truly shines. The git branch is extremely lightweight. This encourages branching in order to add new features.

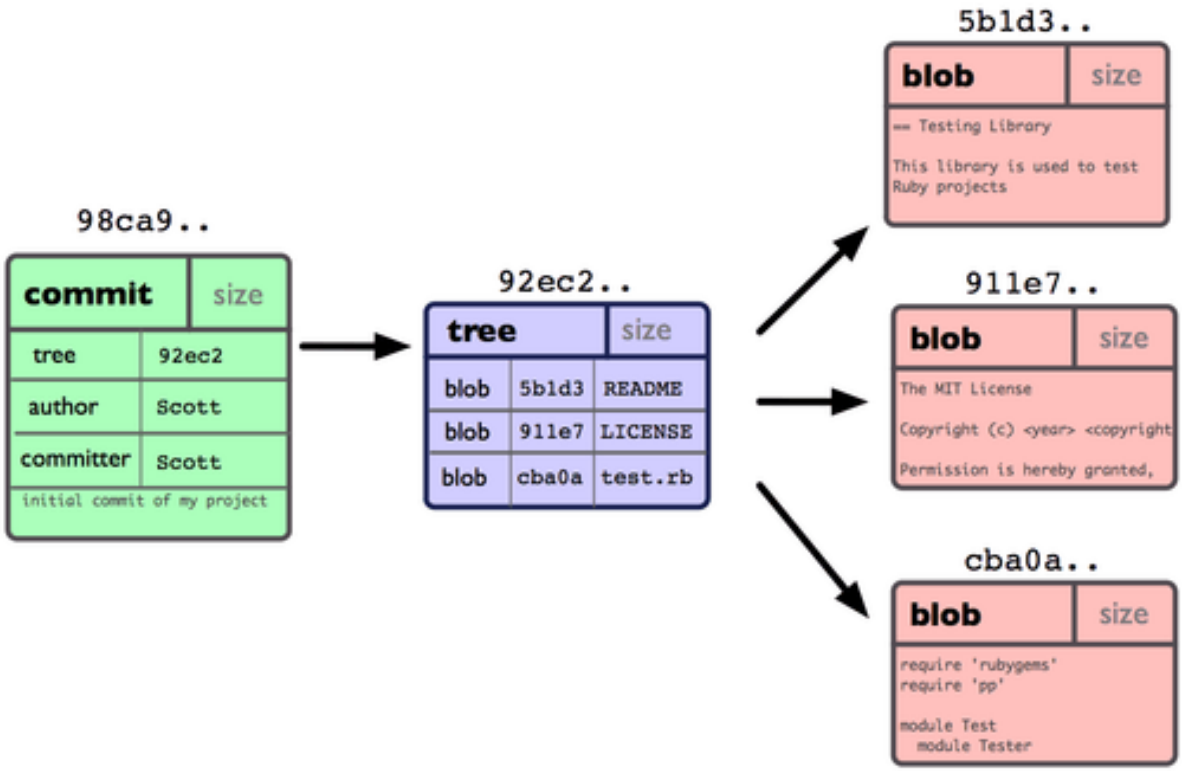
What is Branching?

- Pretty much every version control system has some form of branching. This means that you diverge from the main line of development and continue to do work without changing the main line.
- Usually this is an expensive process because you have to copy all of the source code in the directory into a new branch.
- However, branching is where git truly shines. The git branch is extremely lightweight. This encourages branching in order to add new features.

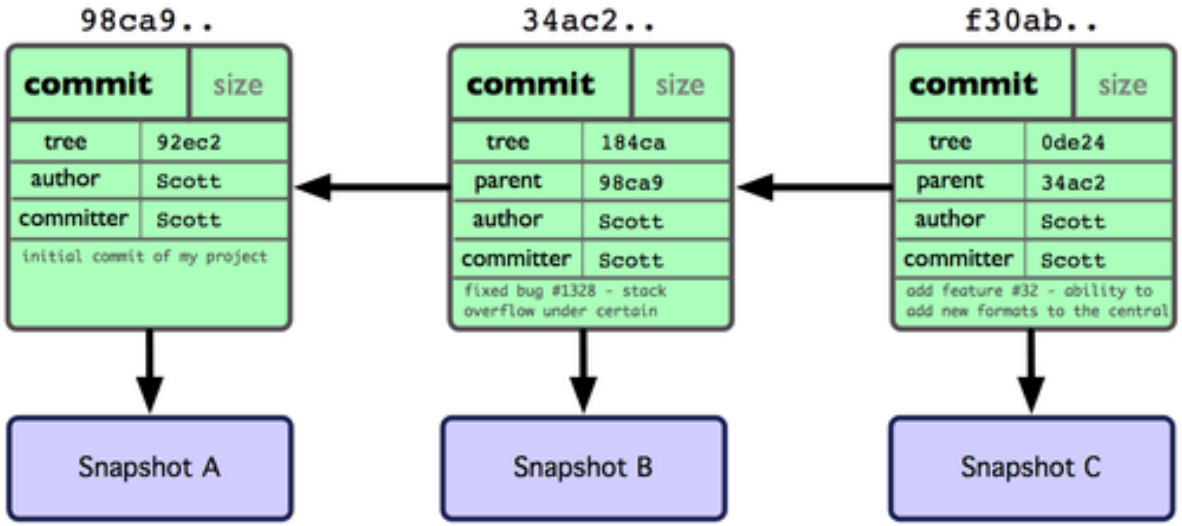
How Does Branching Work?

Let's look at a couple of examples from Pro Git (2nd Edition).

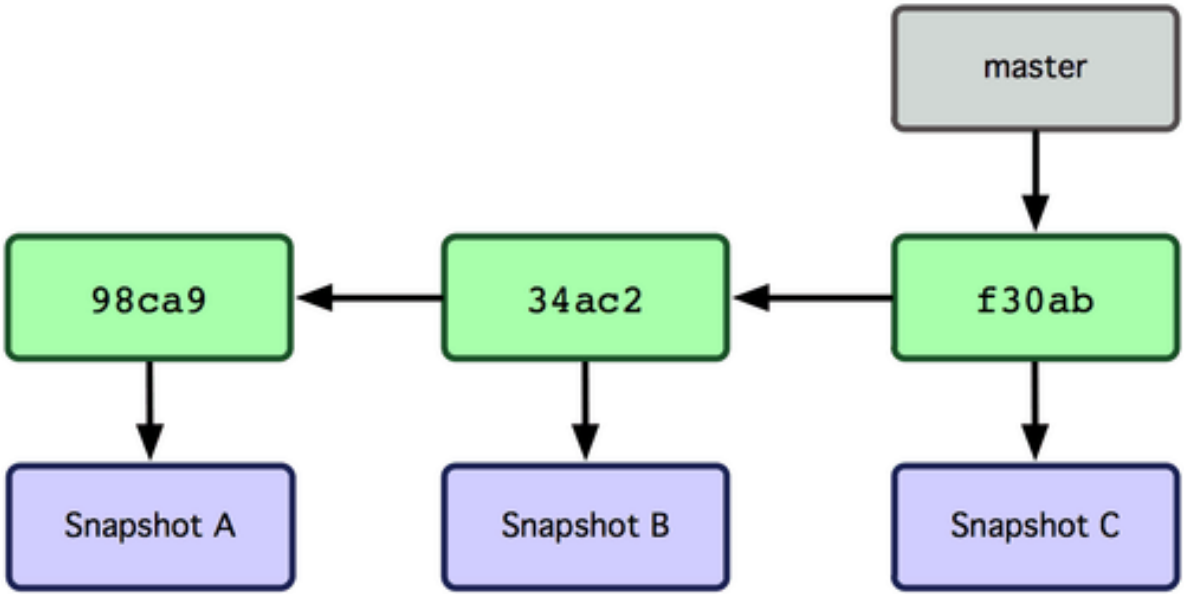
How Does Branching Work?



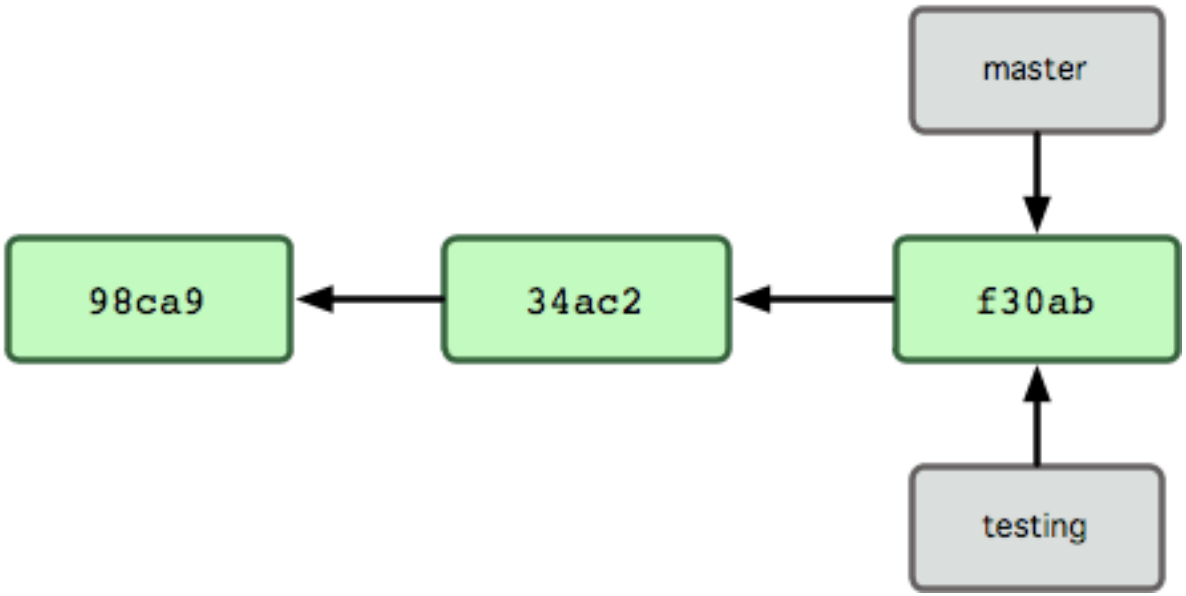
How Does Branching Work?



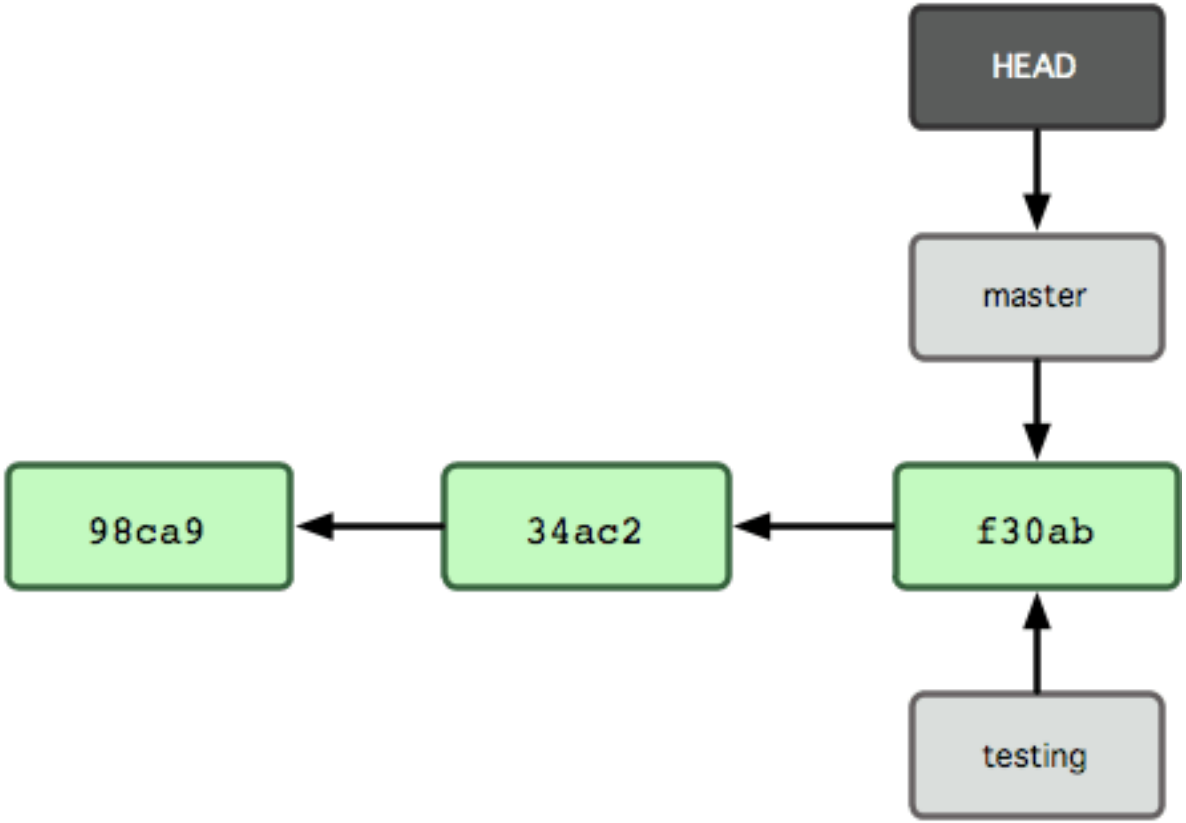
How Does Branching Work?



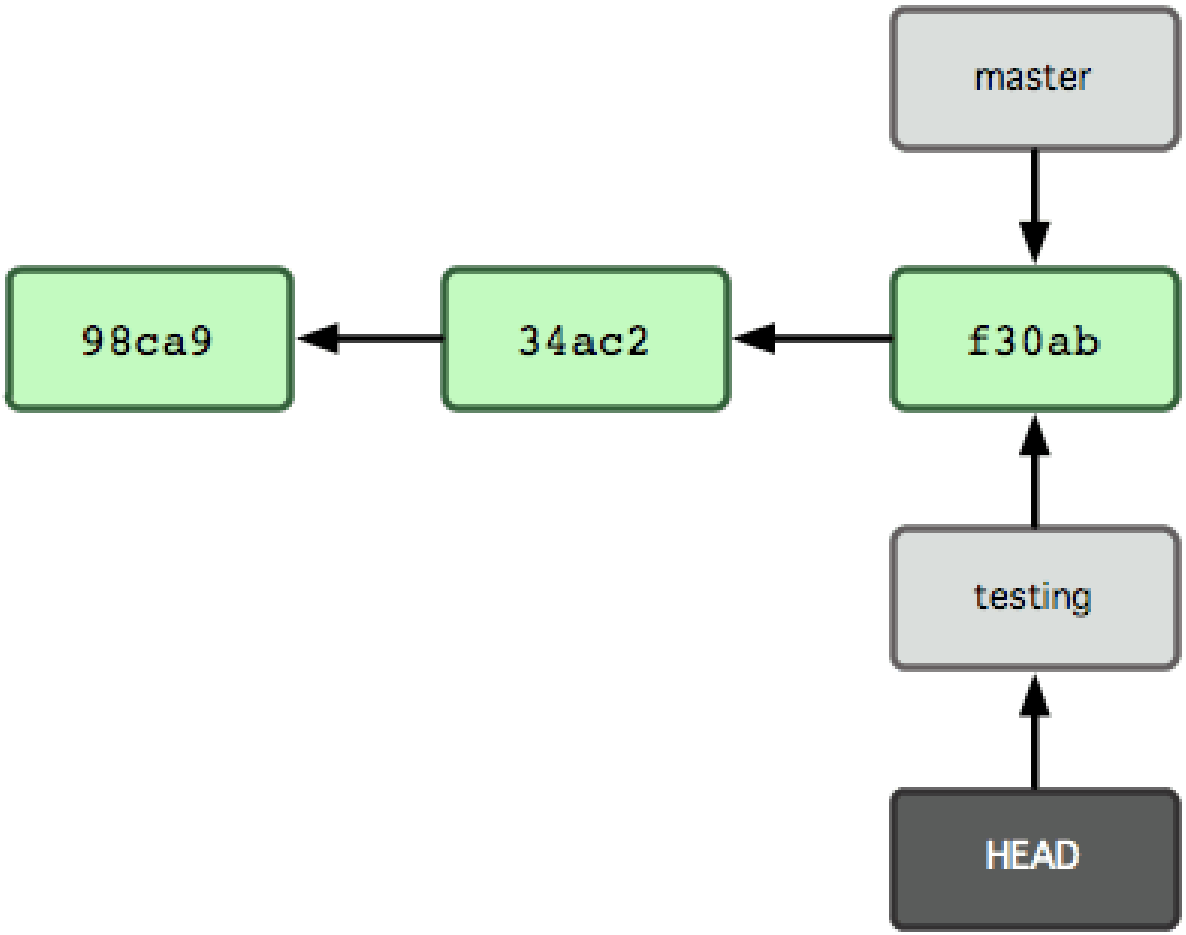
How Does Branching Work?



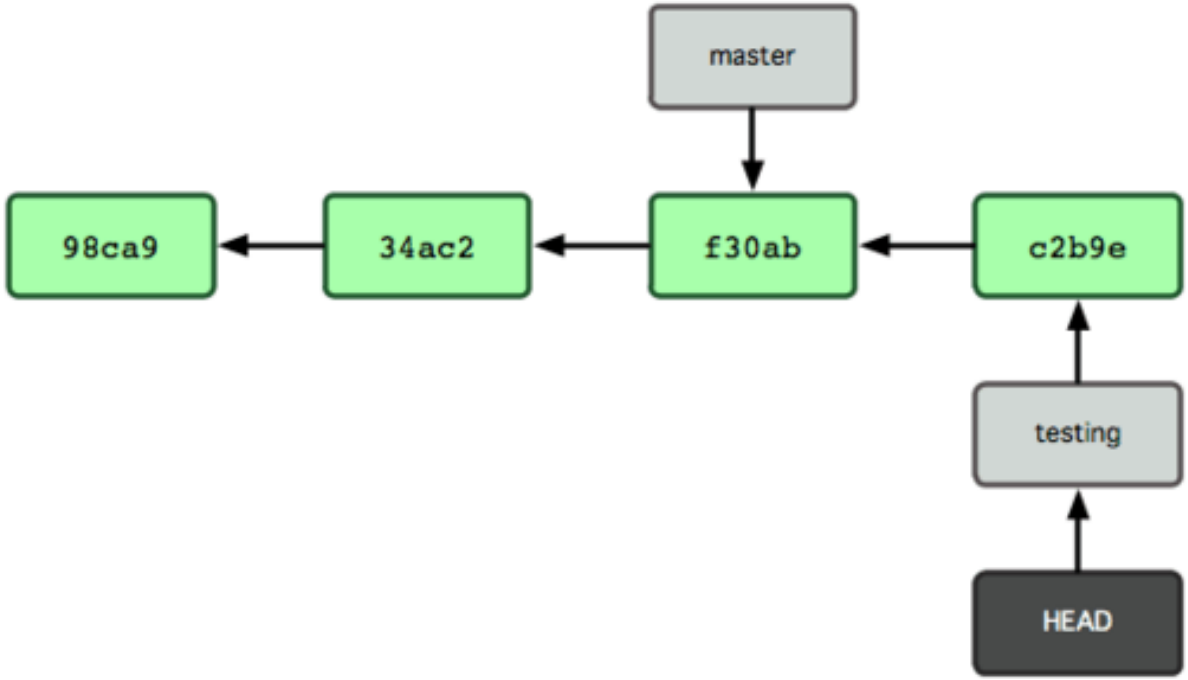
How Does Branching Work?



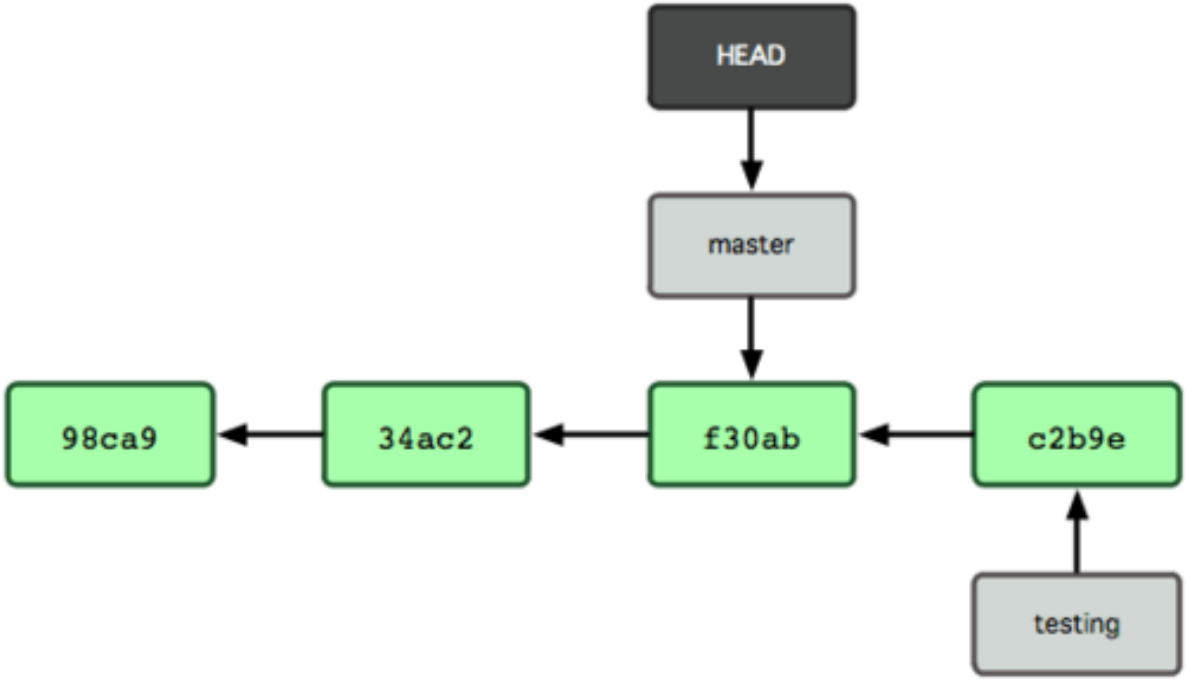
How Does Branching Work?



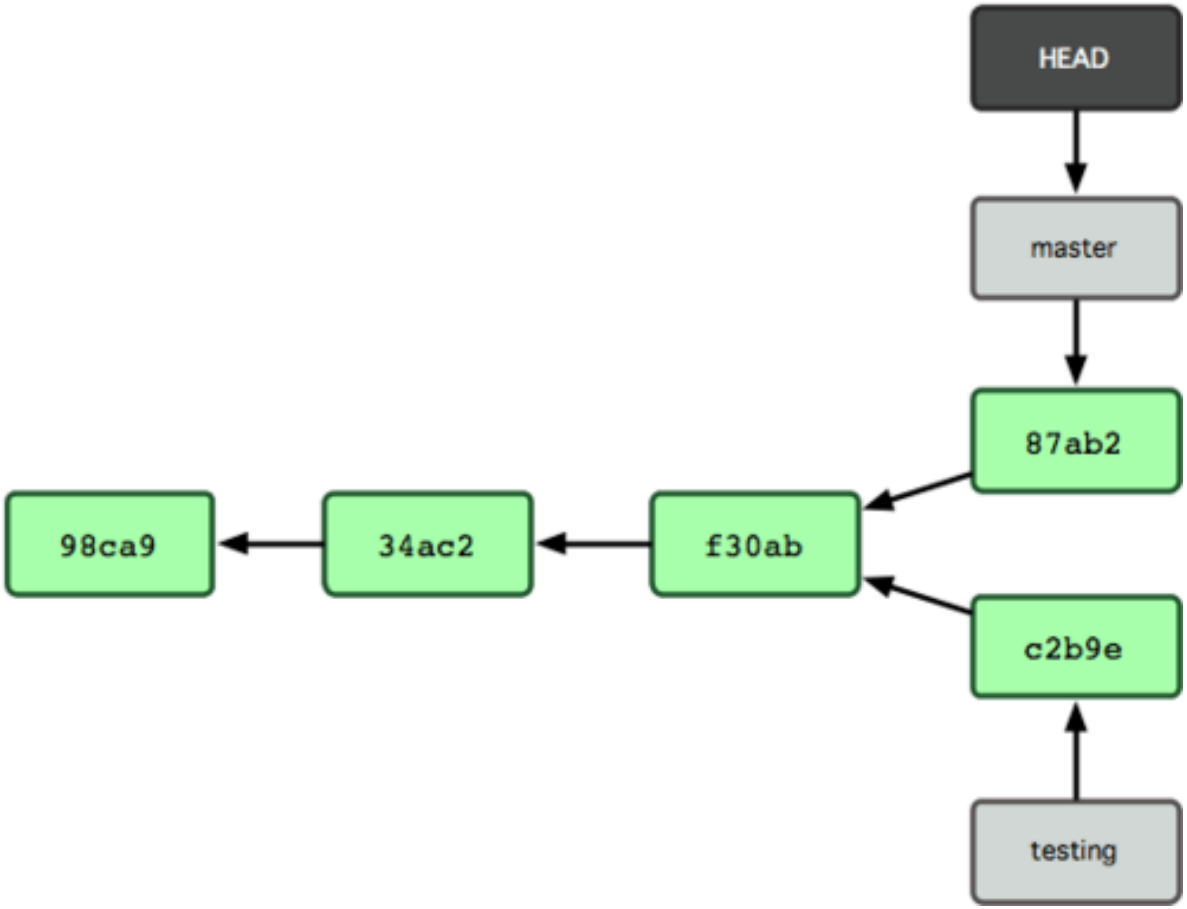
How Does Branching Work?



How Does Branching Work?



How Does Branching Work?



Delete a File (rm vs. git rm)

You can just delete a file in your filesystem, but this will need that you commit your changes with `git add file_removed`.
Instead, you can use `git rm file_name` to do these two things for you.[1]

[1] More discussion at: `http://stackoverflow.com/questions/7434449/why-use-git-rm-to-remove-a-file-instead-of-rm`

If you rm a file, it will delete it locally, but it will still exist in your git directory. In order to fully delete a file, you need to use git rm

Delete a File (rm vs. git rm)

If you want to delete a file that has been staged, but not committed, use git rm --cached

Delete a File (rm vs. git rm)

If you want to move a file, use git mv

Discard Changes to Unstaged Files

git checkout -- filename Branching is better practice

Amending Staged Files

In order to remove a file from the staged environment, use: git reset filename

Amending Commits

I purposely didn't add anything here. Don't do it...

Amending Commits

Ok, fine... git commit --amend

View All Remote Repositories

```
git remote -v
```

Add Remote Repositories

git remote add url

Pull From Remote Repositories

git fetch url git add git commit

Push to Remote Repositories

git push [remote name] [branch name] git add git commit

Public vs. Private Repositories

Bitbucket and GitHub

There is a lot more to learn! We did not discuss:

- Tagging
- Aliases

Thank you for your attention.