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

Reproducible Research

Purdue is already taking action in this subject:

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

We can create a DOI for a GitHub repository using `zenodo.org`:

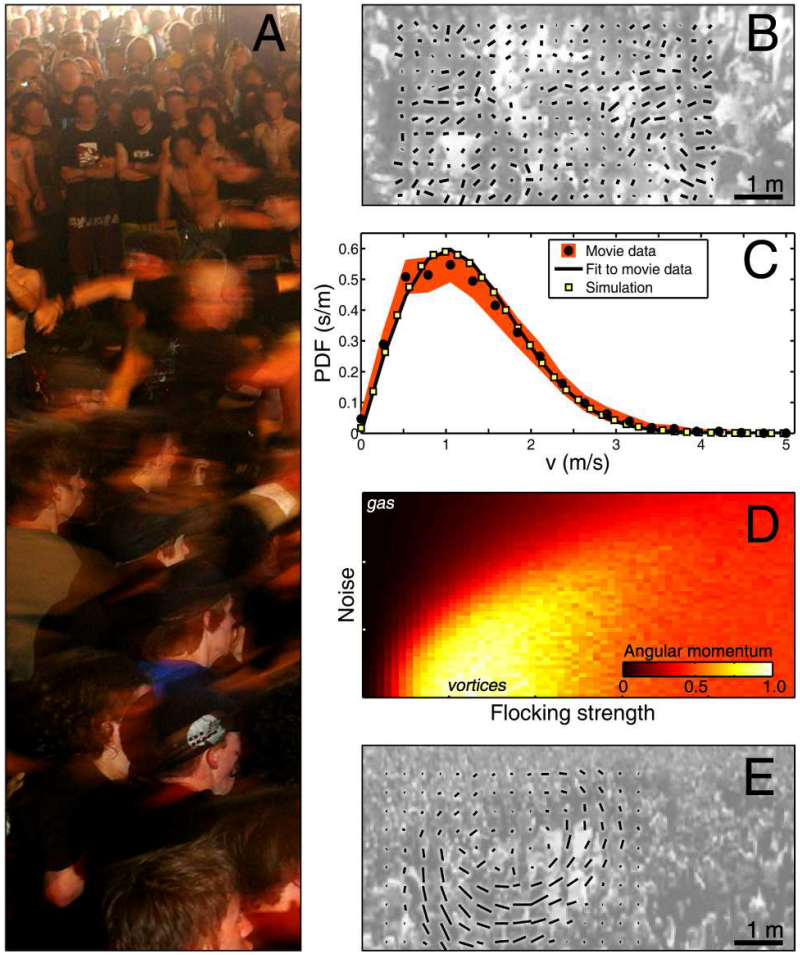
- `https://guides.github.com/activities/citable-code/`

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`



Some Useful Skills

Some Useful Skills That You Should Learn Are:

1. Bash
2. Markdown
3. Vim and/or Emacs

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. Run *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

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

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

1. Modified
2. Staged
3. Committed

The Basic Git Work Flow

In order to determine which files are in which state, you can use (most to least detail):

1. *git diff* (unstaged changes only)
2. *git status*
3. *git status -s*

The Basic Git Work Flow

In order to get a full history of your commits, you can use:

- *git log*

Every commit 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

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.

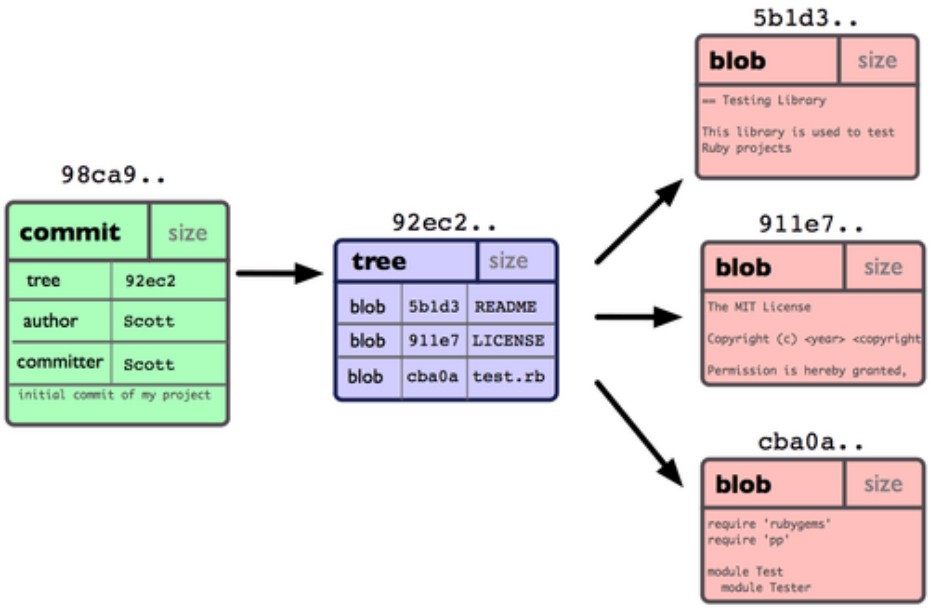
What is Branching?

- 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?

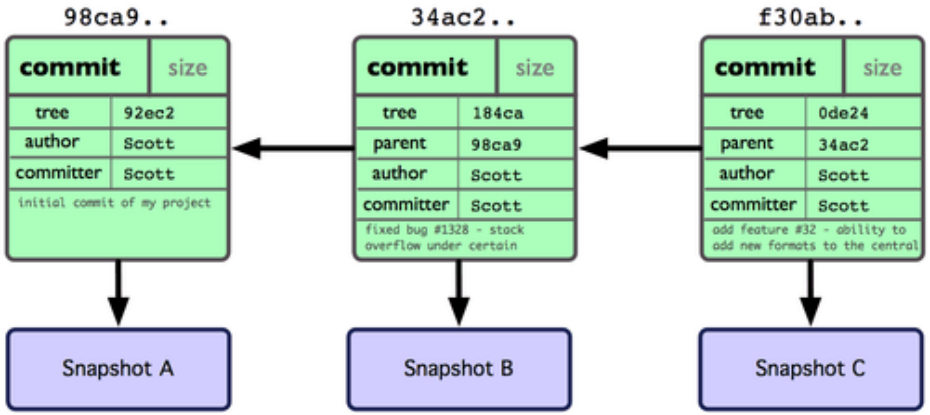
Let's look at a couple of examples from Pro Git (2nd Edition).

How Does Branching Work?



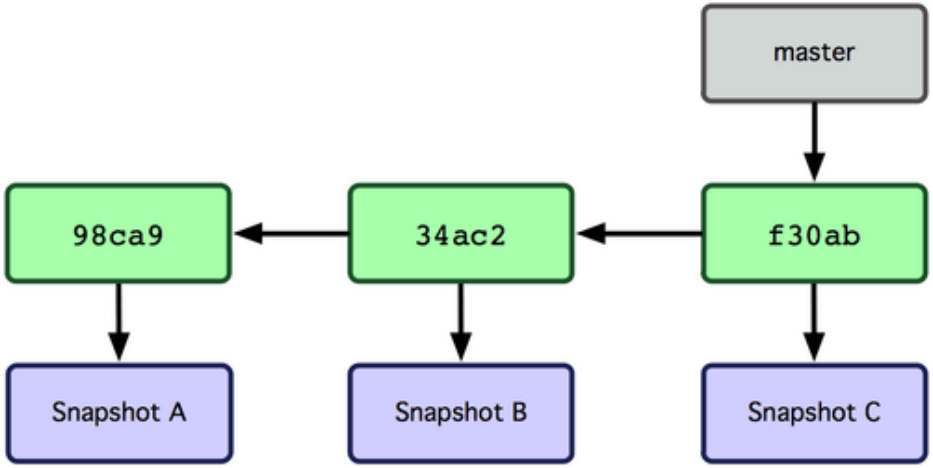
This is the structure of a commit.

How Does Branching Work?



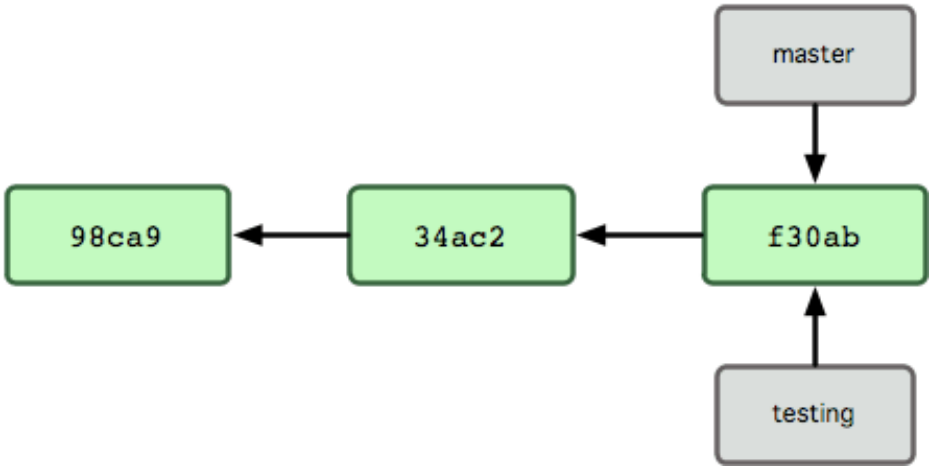

```
# Add code; git -a commit
# Add code; git -a commit
# Add code; git -a commit
```

How Does Branching Work?



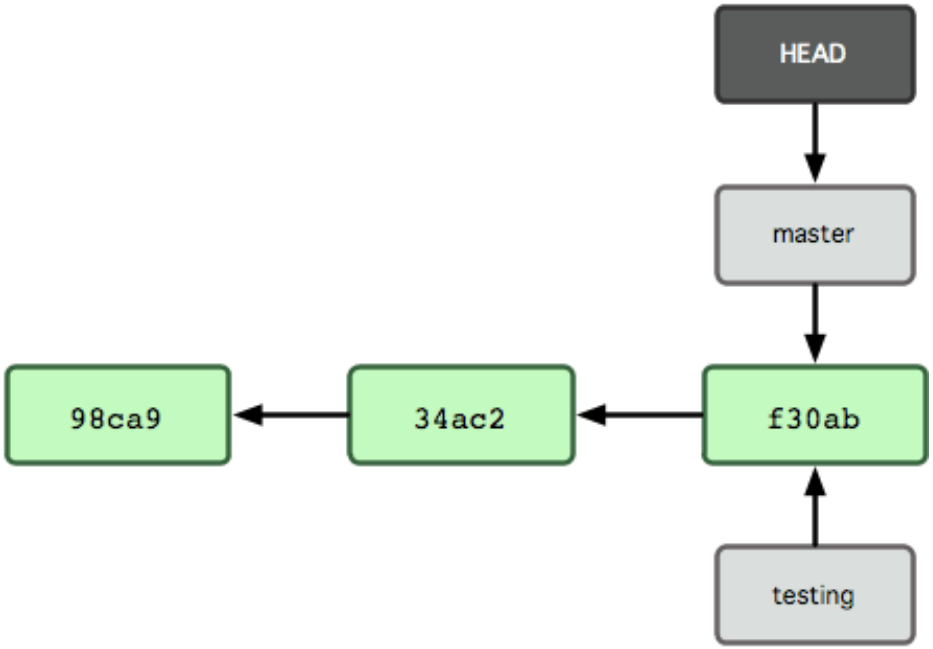
Every project starts off with a master branch.

How Does Branching Work?



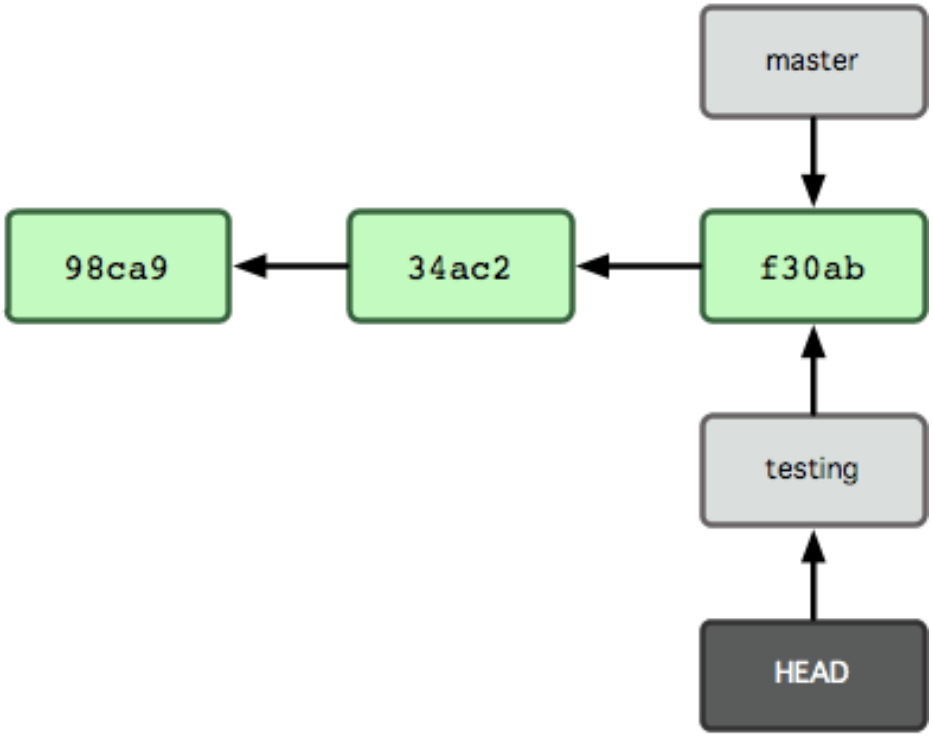
git branch testing

How Does Branching Work?



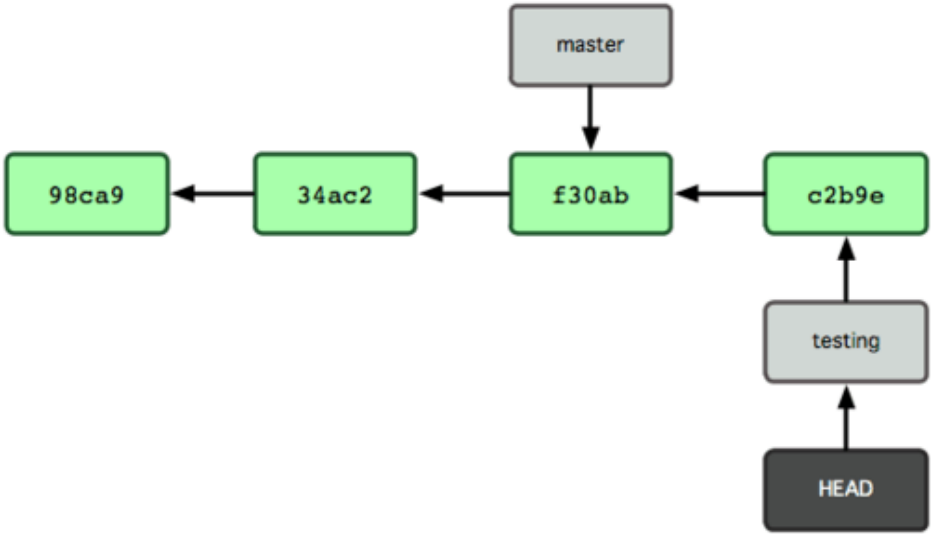
HEAD is still on the master branch.

How Does Branching Work?



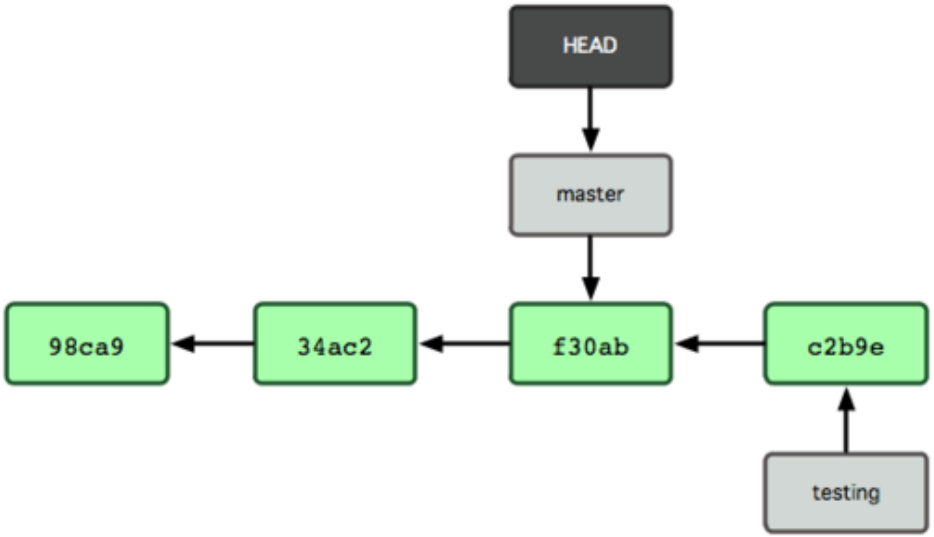
git checkout testing

How Does Branching Work?



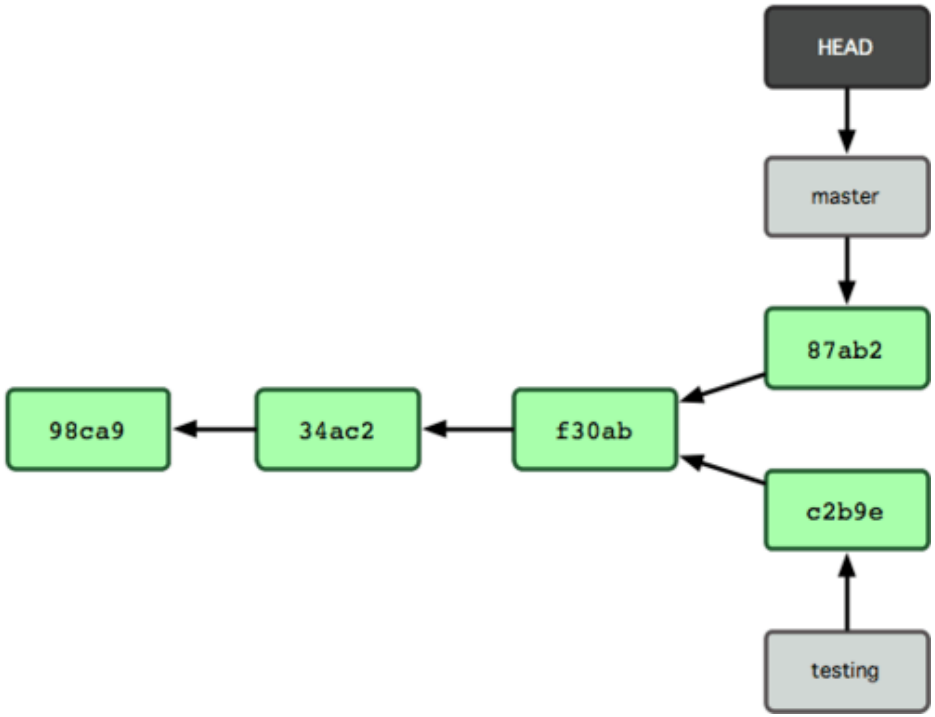

```
# Add new code to testing
git -a commit
```

How Does Branching Work?



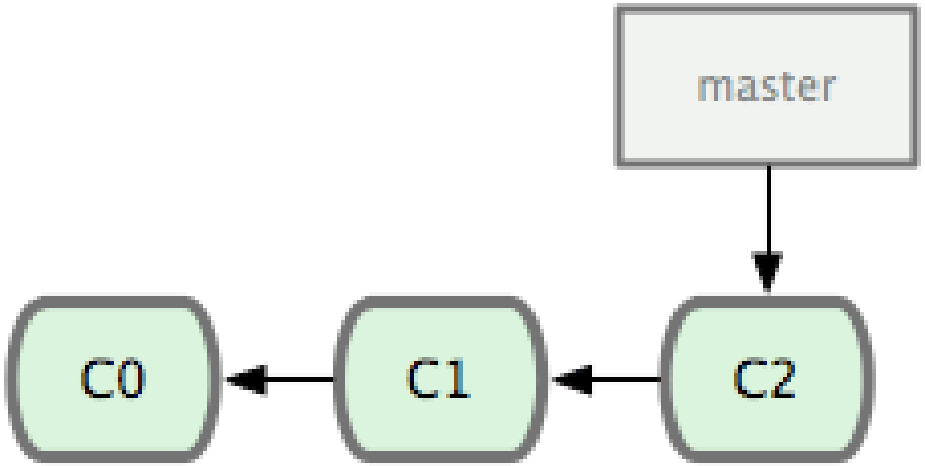
git checkout master

How Does Branching Work?



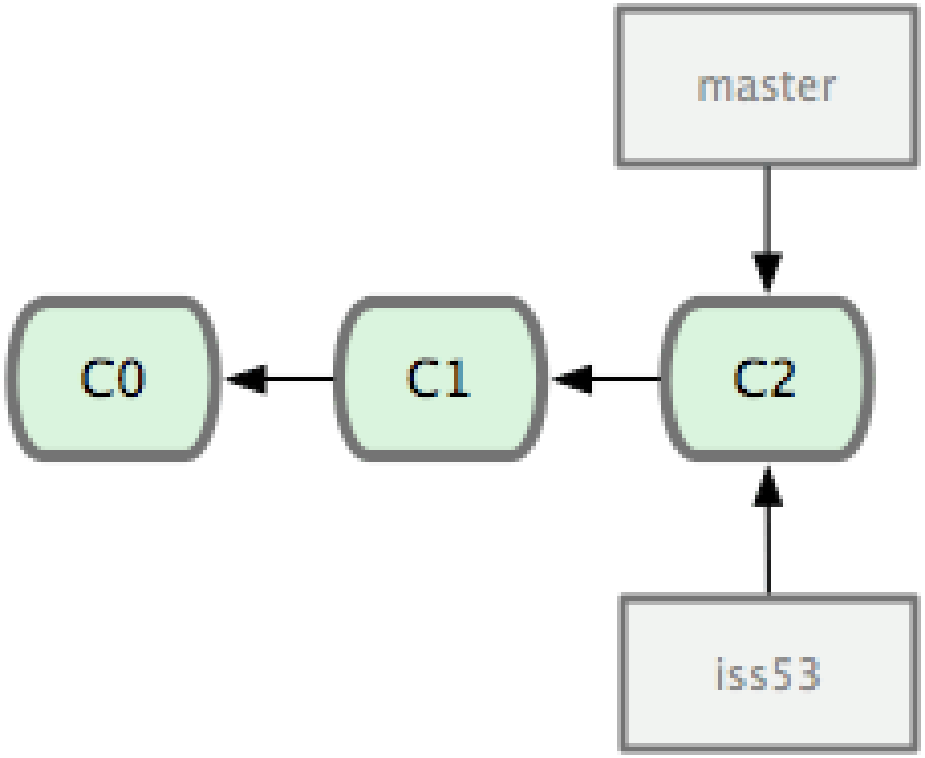
Add new code to master
git -a commit

How Does Merging Work?



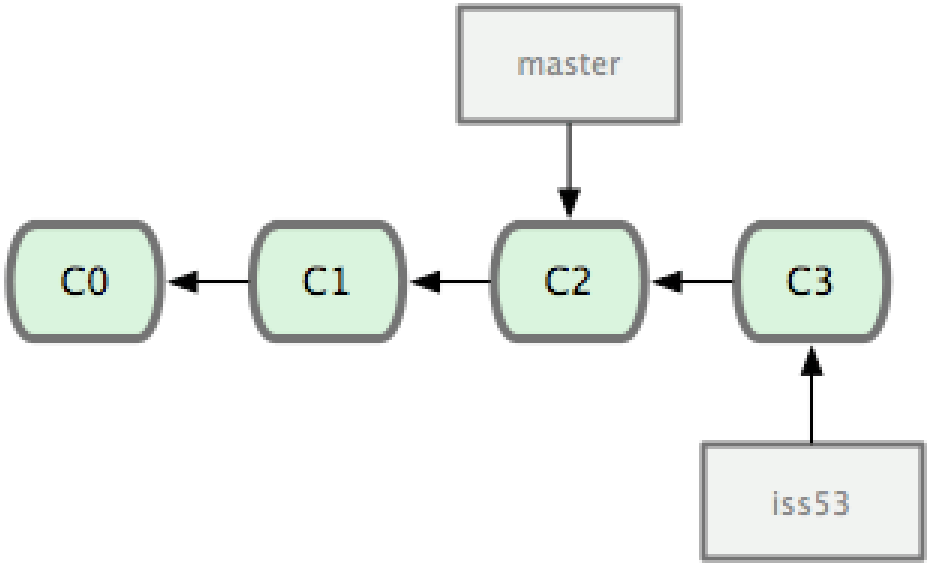
Suppose we have a project with a few current commits.

How Does Merging Work?



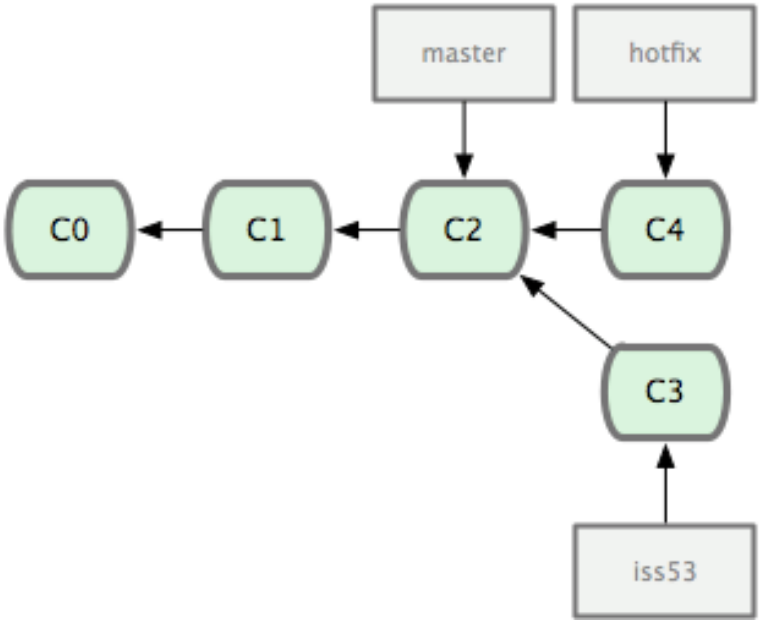
git checkout -b iss53
(git branch iss53; git checkout iss53)

How Does Merging Work?



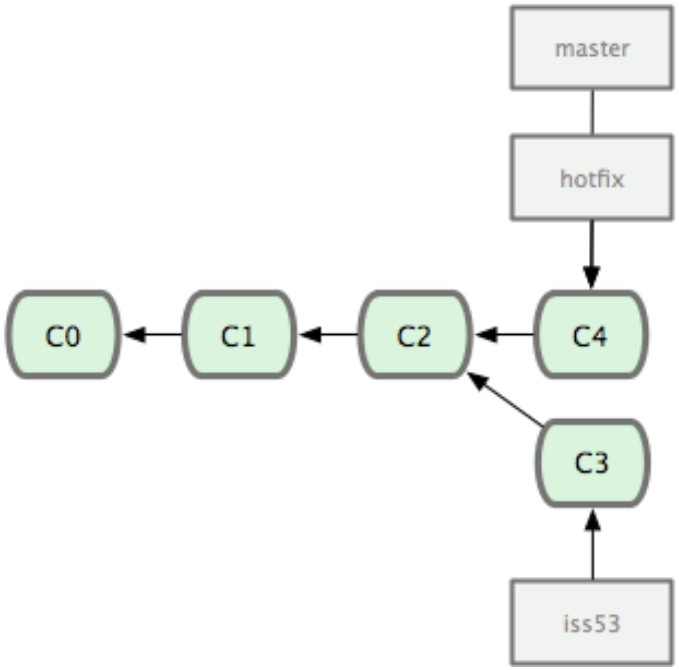

```
# Add code to iss53
git -a commit
```

How Does Merging Work?



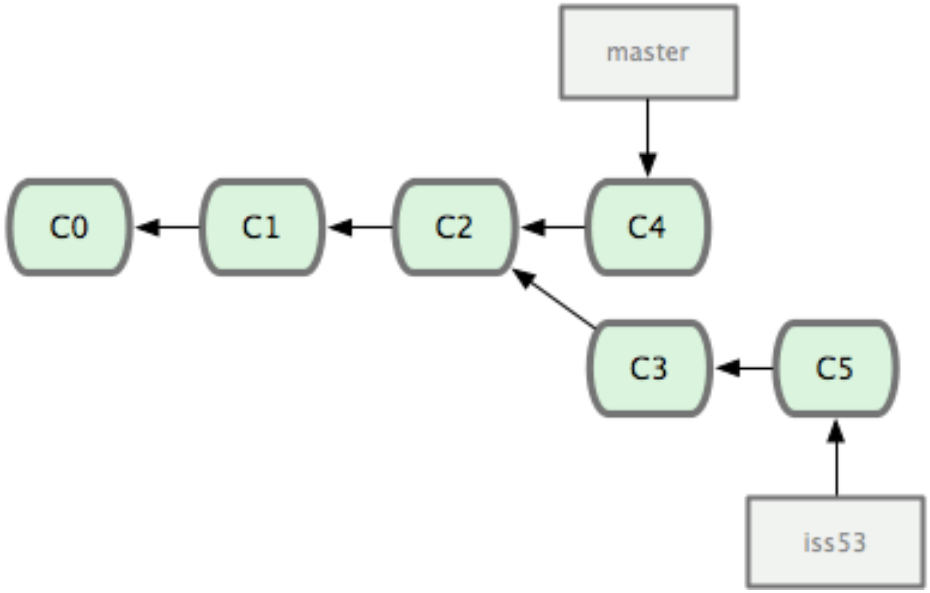
git checkout master
git checkout -b hotfix
Add code to hotfix
git -a commit

How Does Merging Work?



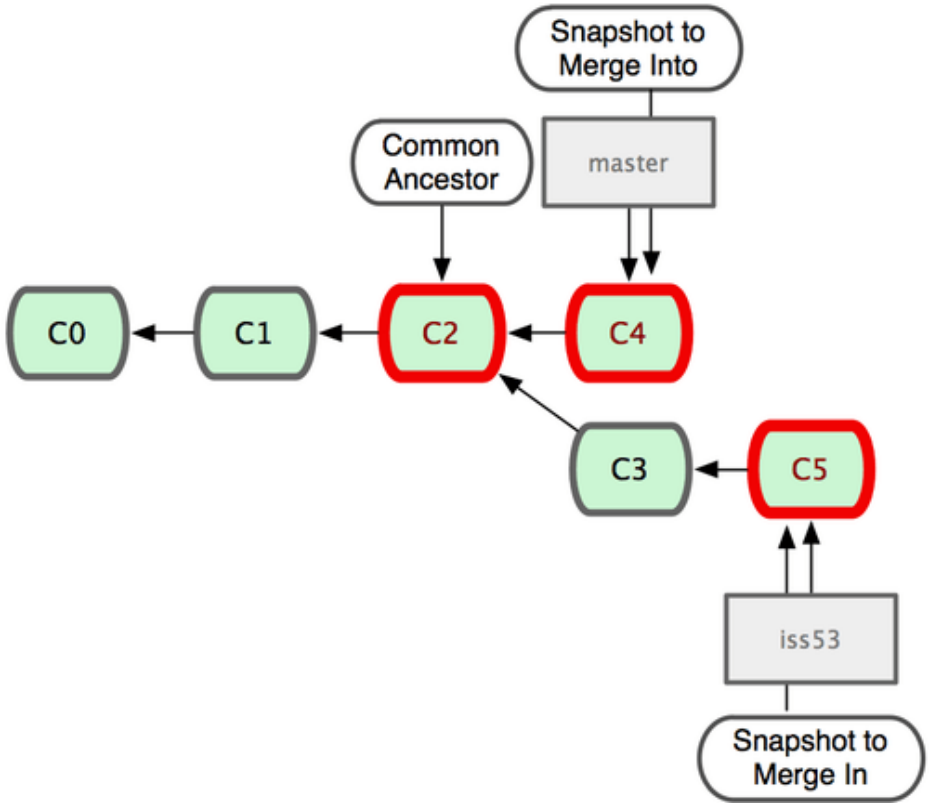
git checkout master
git merge hotfix

How Does Merging Work?



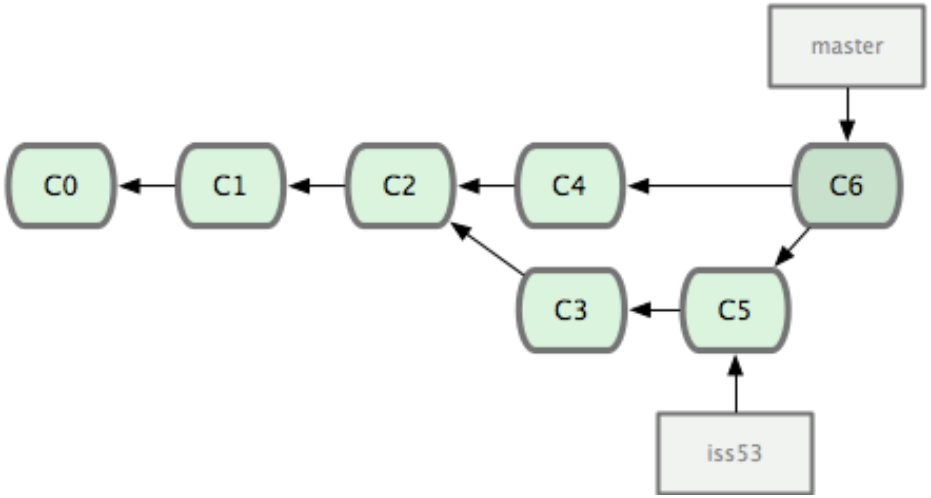

```
git branch -d hotfix
git checkout iss53
# Add code to iss53 branch
git -a commit
```

How Does Merging Work?



We want to merge iss53 to master

How Does Merging Work?



git checkout master
git merge iss53

Merge Conflicts

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge
conflict in index.html
Automatic merge failed; fix
conflicts and then commit the
result.
```

- Sometimes we run into merge conflicts
- *git status* is useful in these cases

Merge Conflicts

```
<<<<<<< HEAD:index.html
<div id="footer">contact :
email.support@github.com</div>
=======
<div id="footer">
 please contact us at
support@github.com
</div>
>>>>>>> iss53:index.html
```

The "=======" divides the two types of code.

Deleting a File (rm vs. git rm)

- If you delete a file in your filesystem, you still need to commit your changes with *git add file_removed*.
- Or you can use *git rm file_name*.

Deleting a File (rm vs. git 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*

Deleting a File

If you want to delete a file that has been staged, but not committed use:

- *git rm --cached*

Moving a File

If you want to move a file use:

- *git mv*

Discarding Changes to Unstaged Files

If you want to discard changes to unstaged files use:

- *git checkout -- filename*

Just keep in mind that branching is better practice...

Amending Staged Files

In order to remove a file from the staged environment use:

- *git reset filename*

Amending Existing Commits

So you say you want to amend an existing commit? Why?
I purposely didn't add anything here. Don't do it...

Amending Commits

Ok, fine...

- *git commit --amend*

But you are missing the point of version control...

Remote Repositories

- Part of the strength of Git is linking your repository with other remote repositories
- We will mostly talk about this in the context of Git and GitHub

View All Remote Repositories

In order to view all of the remote repositories for a project use:

- *git remote -v*

Add Remote Repositories

In order to add a remote repository use:

- *git remote add some_url*

Pull From Remote Repositories

In order to pull data from a remote repository use:

git fetch some_url
git -a commit

Push to Remote Repositories

In order to push data to a remote repository use:

git push remote_name branch_name

GitHub Demo

It's easier to demonstrate this than to write slides about it...

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

- Tagging
- Aliases
- Advanced Remote Control
- The --hard Option
- Custom Environments
- Scripting and Extending Git
- And Much More!

Thank You For Your Attention.