



Python Deeper Dive Intro to Git

Data Boot Camp
Lesson 3.3



Version Control Systems in General

Git

Commits

- Big picture
- How-to

Branching

- Big picture
- How-to

Remote Repositories

- Big picture
- How-to

Version Control Systems

The Problem: Keeping Track of Work History Is Hard!

Worse when you consider that:



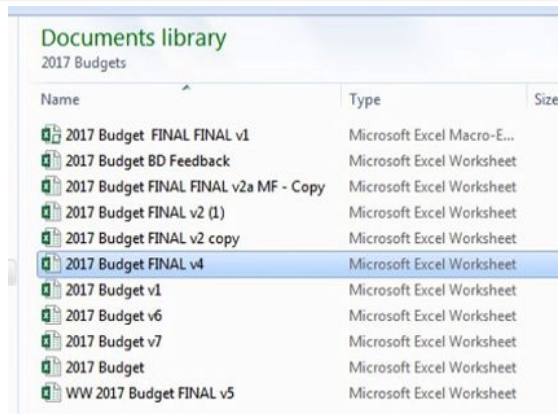
People may need to work on something at the same time



And often in different locations



Over the lifetime of a project, it may not even be the same people



Documents library

2017 Budgets

Name	Type	Size
2017 Budget FINAL FINAL v1	Microsoft Excel Macro-E...	
2017 Budget BD Feedback	Microsoft Excel Worksheet	
2017 Budget FINAL FINAL v2a MF - Copy	Microsoft Excel Worksheet	
2017 Budget FINAL v2 (1)	Microsoft Excel Worksheet	
2017 Budget FINAL v2 copy	Microsoft Excel Worksheet	
2017 Budget FINAL v4	Microsoft Excel Worksheet	
2017 Budget v1	Microsoft Excel Worksheet	
2017 Budget v6	Microsoft Excel Worksheet	
2017 Budget v7	Microsoft Excel Worksheet	
2017 Budget	Microsoft Excel Worksheet	
WW 2017 Budget FINAL v5	Microsoft Excel Worksheet	



Version control systems are software that help you track changes you make in your code over time. As you edit your code, you tell the version control system to take a snapshot of your files. The version control system saves that snapshot permanently so you can recall it later if you need it.

—Microsoft

GitHub Commits

Git works in commits

Git thinks of project history as a series of snapshots or checkpoints. The Git term for this is **commits**.

When you're ready to save your work, it's said that you can "make a commit," "commit your files," or just "commit." Think of this as taking a snapshot of your entire project, aka *repository*.

In the future, you can always go back to previous commits, no matter what you've done to your code in the meantime.

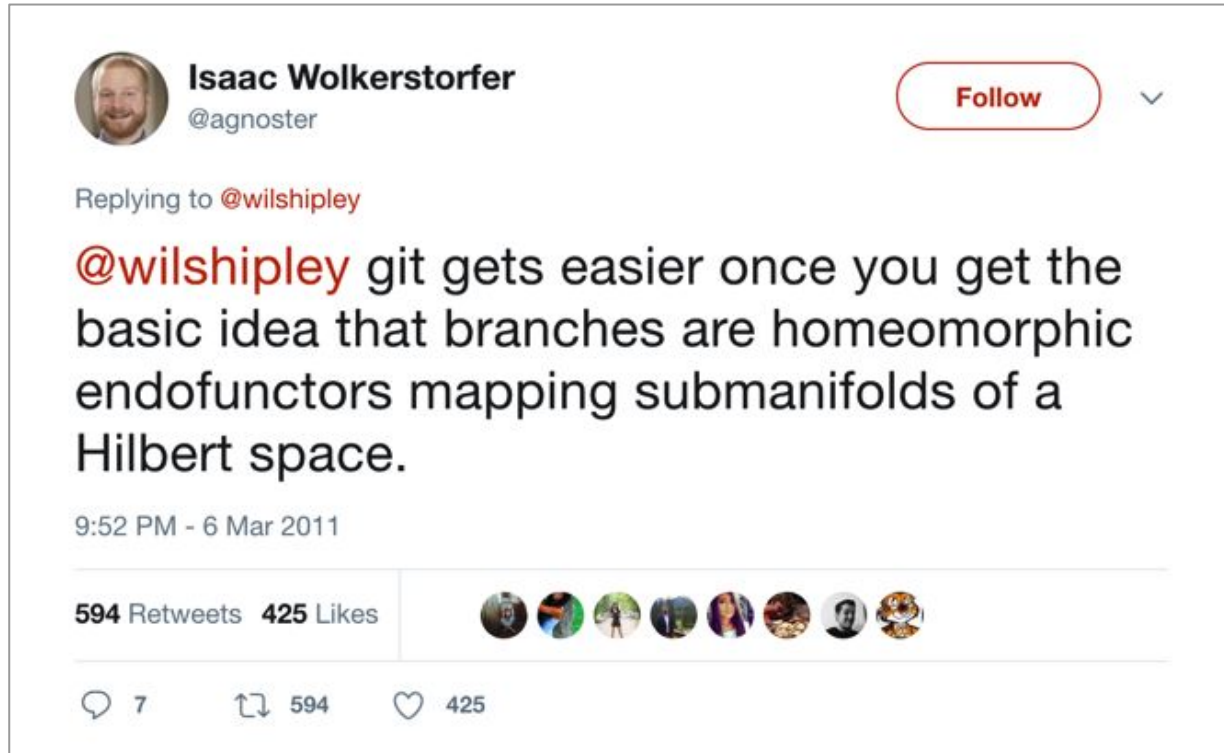


GitHub

Git is a version control system. For our class, we will be using **GitHub** as the hosting service for our Git repositories.

Using Git on GitHub

It's not this bad, but Git *is* known for having a bit of a learning curve.



GitHub Commits

GitHub Commits

Other metaphors

Reaching a checkpoint in
a video game



Time Machine on Mac
(or other backups)

Saving a v37 (except
more sophisticated)

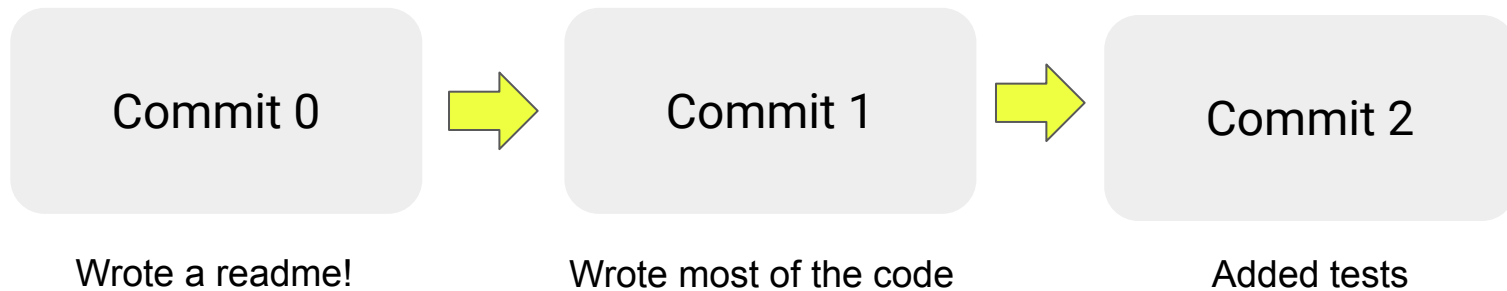
GitHub Commits

Another way to think about a commit



GitHub Commits

A project is just a series of commits



GitHub Branching

GitHub Branches

All commits live on branches



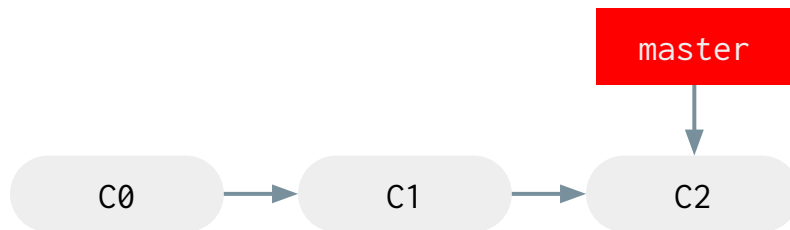
The default, or main, branch is typically called the **master** branch.



However, you can have many branches.

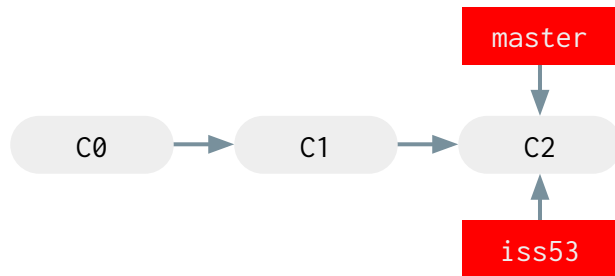


For example, you might start a **feature** branch in order to work on a new experimental feature. If we're working on issue #53 in our backlog, let's call the branch "**iss53**".

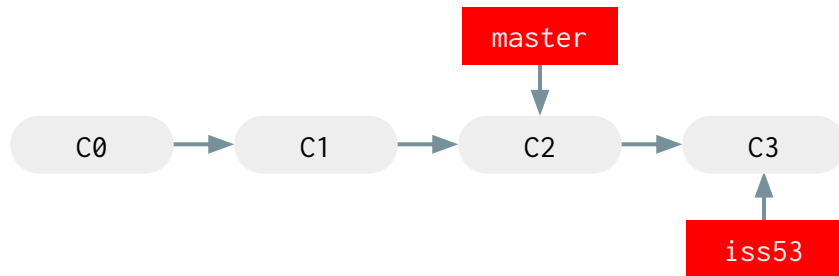


GitHub Branches

At first, our branch and the **master** branch look exactly the same:

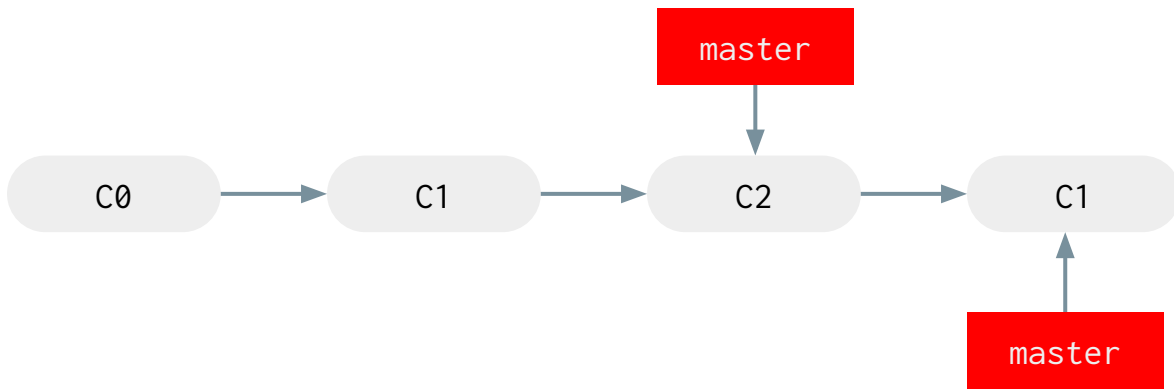


Then, maybe we'll do some work and make a commit on the “**iss53**” branch. Then it will look like this:



GitHub Branches

In simple cases, once we've finished working on the feature branch, we'll be ready to merge it back into the master branch. Then we can get rid of the feature branch because the master branch will now contain all of the commit information.



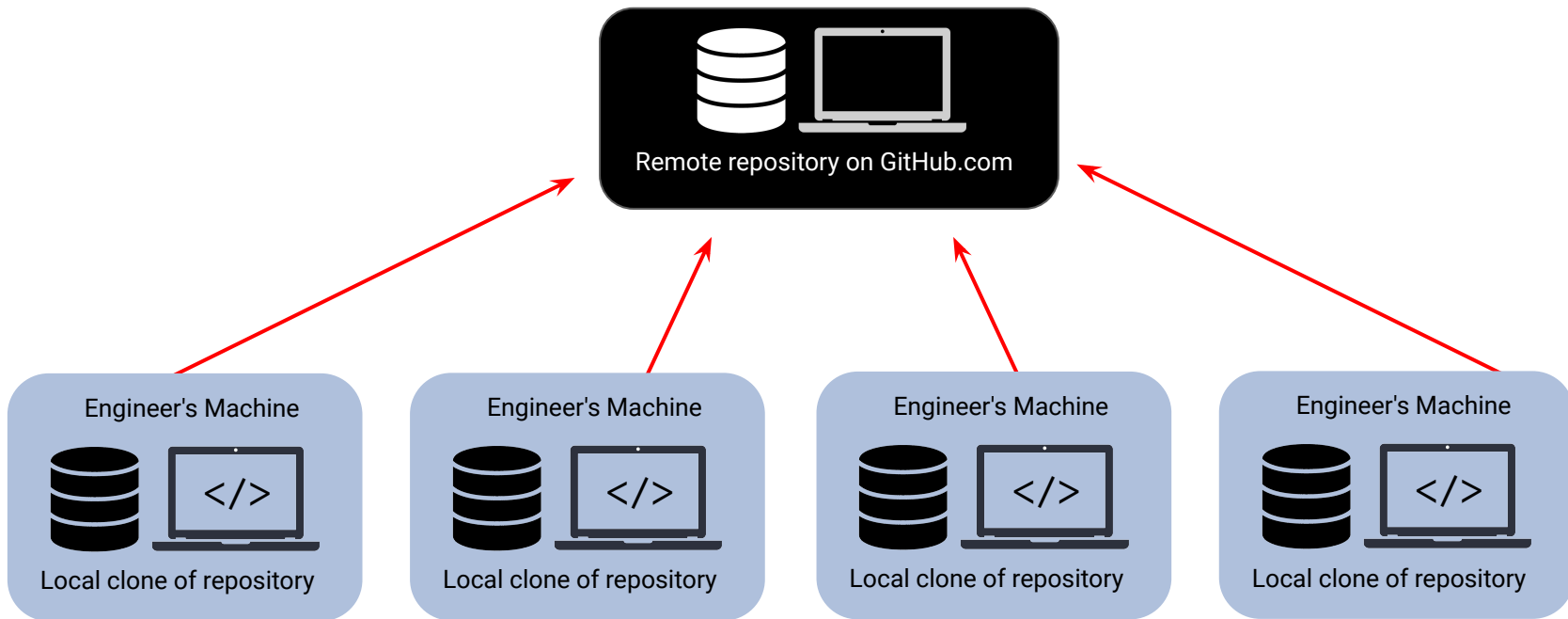


Branching may not seem that important now. However, it is foundational for more complex workflows and for working with other people on the same project. For now, **remember that branching is a useful way to work on separate parts of the project at the same time.**

GitHub Remote Repositories

GitHub Remote Repositories

So far, we've been talking only about the *local* environment—that is, a *local repository* on your computer





Git really starts to shine when you have multiple workers on their own machines trying to sync code to a **remote repository** that is hosted on another machine accessible to everybody (e.g., on the Internet or some other network).

GitHub Remote Repositories

In many workflows, the remote repository serves as an even-more-central master branch. A typical workflow consists of the following:

1. Sync your local repo with the remote repo to get the most up-to-date version of the code.
2. Create a feature branch.
3. Do some work.
4. Commit your changes.
5. Merge your feature branch into your local master.
6. *Push* your local master to the remote master.
7. Now everyone has access to the code you just wrote.



Additional Git and GitHub Resources

[Git-scm.com](https://git-scm.com)

[Recompilermag.com](https://recompilermag.com)

slideshare.net