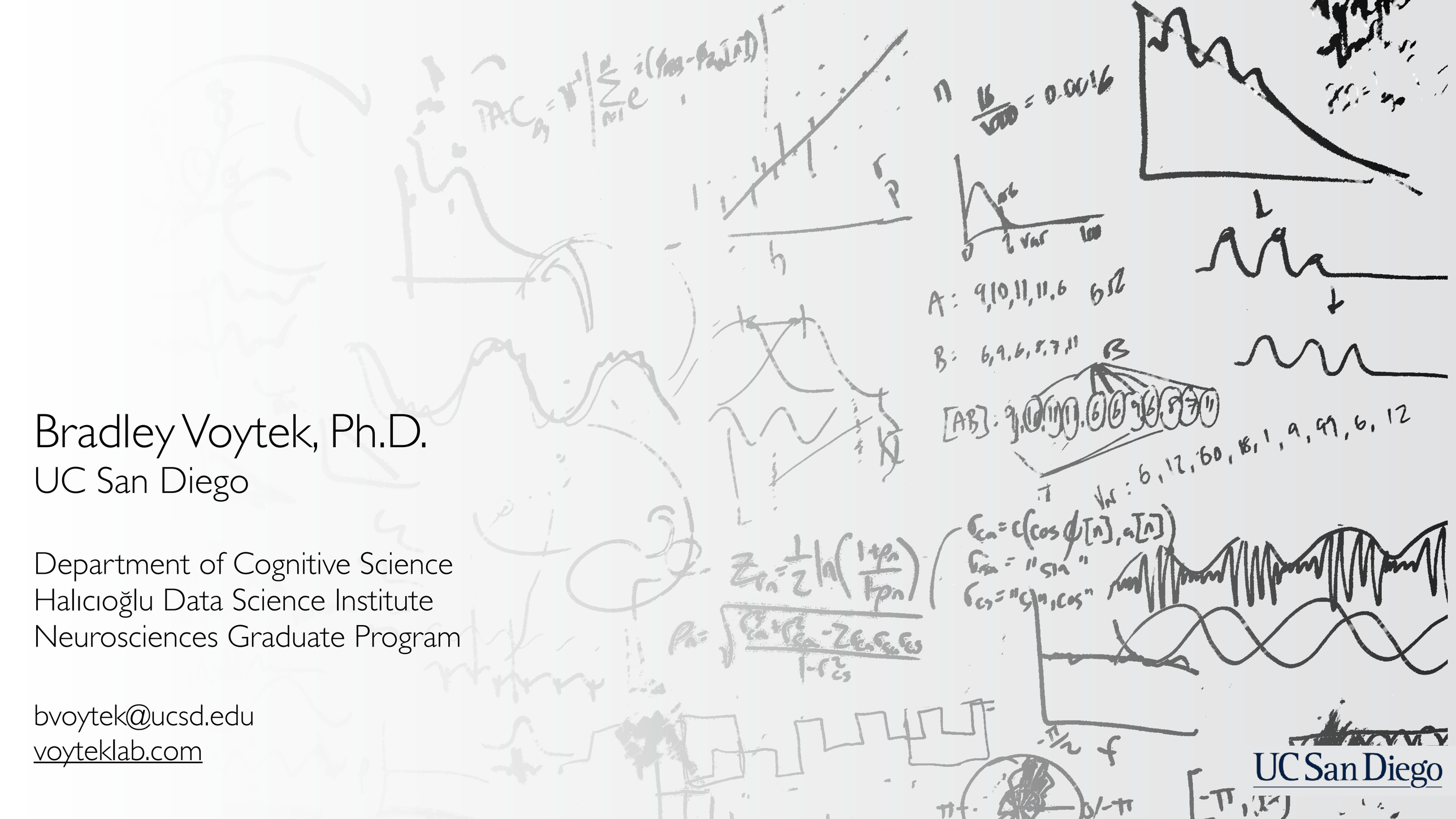
Bradley Voytek, Ph.D.

UC San Diego

Department of Cognitive Science
Halıcıoğlu Data Science Institute
Neurosciences Graduate Program

bvoytek@ucsd.edu
voyteklab.com



UC San Diego

VOYTEKlab

Cognitive Science

Eena Kosik

Quirine van Engen

Dillan Cellier

Postdocs

Pamela Riviere Ruiz

Christian Cazares

Data Science

Ryan Hammonds

Neuroscience

Andrew Bender

Blanca Burgos-Martin

Morgan Fitzgerald

Trevor McPherson

Michael (MJ) Preston

Sydney Smith

I care about how we measure things.

More specifically…

How we transform our

data

into

numbers

that we perform

statistics

on is very important.

So we write (Python) code
and
develop algorithms

to understand
(and help others understand)

“how the brain works”

But to do that, you need to learn some technical skills.

Git and **GitHub**

This sucks

Voytek-NatNeuro-Craniectomy_2008Dec14.doc	Dec 16, 2008, 4:22 PM
Voytek-NatNeuro-Craniectomy_2008Dec26.doc	Jan 14, 2009, 5:26 PM
Voytek-NatNeuro-Craniectomy_2008Feb23.doc	Feb 23, 2009, 1:05 PM
Voytek-NatNeuro-Craniectomy_2008Jan14.doc	Jan 19, 2009, 9:54 PM
Voytek-NatNeuro-Craniectomy_2008Jan19.doc	Jan 20, 2009, 7:06 PM
Voytek-NatNeuro-Craniectomy_2009Mar4.doc	Mar 4, 2009, 10:03 PM
Voytek-NatNeuro-Craniectomy_2009Mar12.doc	Mar 12, 2009, 1:58 PM
Voytek-NatNeuro-Craniectomy_2009Mar16-FINALtoBOB.doc	Mar 16, 2009, 4:41 PM
Voytek-NatNeuro-Craniectomy_2009Mar20.doc	Apr 3, 2009, 6:08 PM

So does this

[redacted] Feb 26

Hi Brad (and everyone),

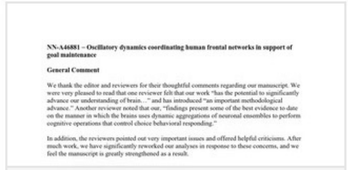
[redacted]

Brad, did you want to take a look through all the comments & make sure you agree with them? Maybe after you've had a chance to look, we can hammer out the answers to 1 & 2 above, if the way forward isn't already clear.

3 Attachments

Voytek-Hierarchy-2015Feb20_mddbak.doc
6.4 MB

Voytek-Hierarchy...



Voytek-Reviewer...

UC San Diego

And so does this…

[redacted] May 11

Hi Brad,

Thanks for chatting with me earlier today. I added the link to the visualization project into my resume and attached the resume. Thanks for any connections you can make for me. I'd love to know where you send it, so I can keep track of that. Thanks again!

Best,

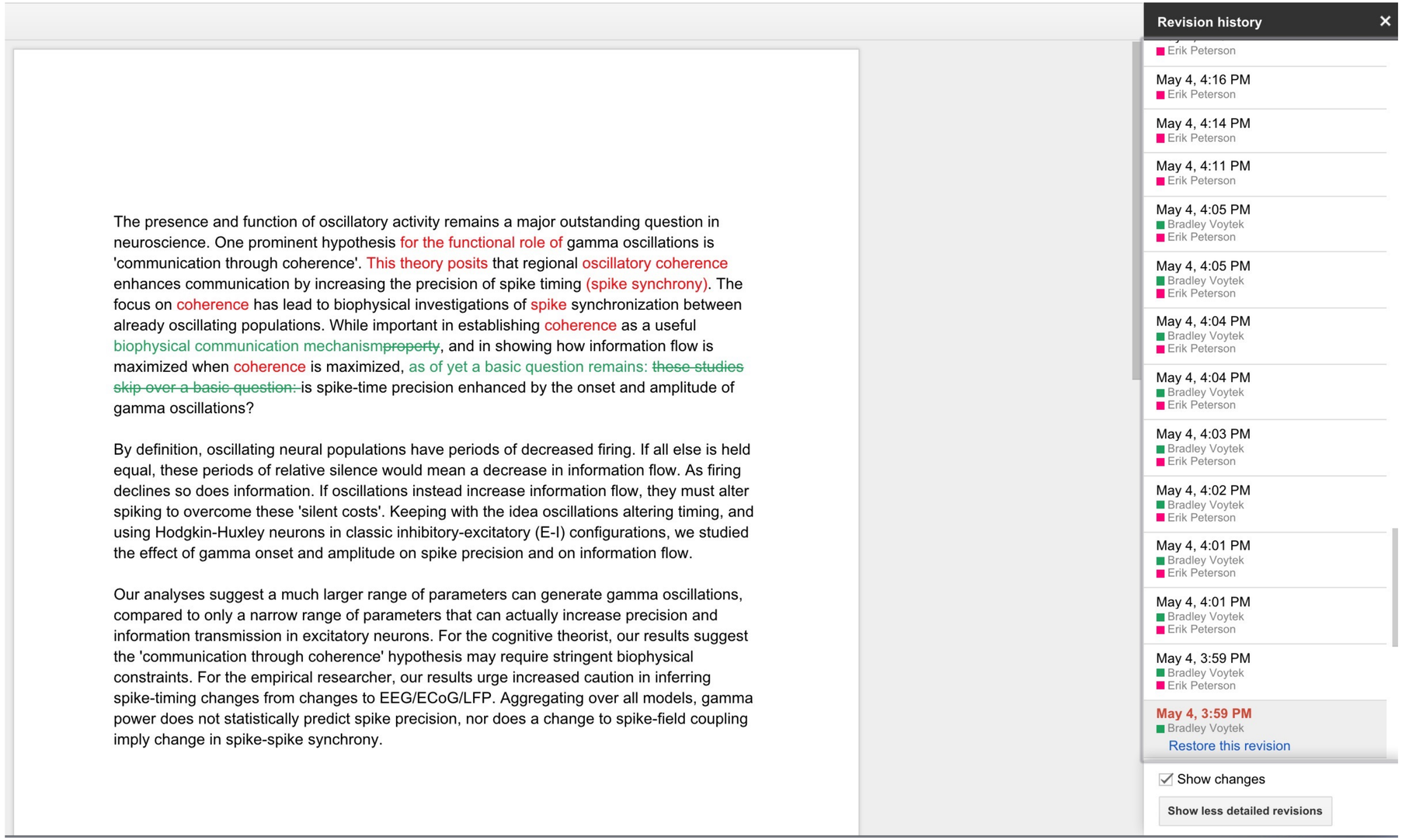
[redacted] May 11

Actually, please use this one. I fixed a typo that was previously missed. Thanks!

[redacted] May 11

Final copy, I swear. Thanks for helping out.

This is better



The presence and function of oscillatory activity remains a major outstanding question in neuroscience. One prominent hypothesis for the functional role of gamma oscillations is 'communication through coherence'. This theory posits that regional oscillatory coherence enhances communication by increasing the precision of spike timing (spike synchrony). The focus on coherence has lead to biophysical investigations of spike synchronization between already oscillating populations. While important in establishing coherence as a useful biophysical communication mechanism~~property~~, and in showing how information flow is maximized when coherence is maximized, as of yet a basic question remains: ~~these studies skip over a basic question:~~ is spike-time precision enhanced by the onset and amplitude of gamma oscillations?

By definition, oscillating neural populations have periods of decreased firing. If all else is held equal, these periods of relative silence would mean a decrease in information flow. As firing declines so does information. If oscillations instead increase information flow, they must alter spiking to overcome these 'silent costs'. Keeping with the idea oscillations altering timing, and using Hodgkin-Huxley neurons in classic inhibitory-excitatory (E-I) configurations, we studied the effect of gamma onset and amplitude on spike precision and on information flow.

Our analyses suggest a much larger range of parameters can generate gamma oscillations, compared to only a narrow range of parameters that can actually increase precision and information transmission in excitatory neurons. For the cognitive theorist, our results suggest the 'communication through coherence' hypothesis may require stringent biophysical constraints. For the empirical researcher, our results urge increased caution in inferring spike-timing changes from changes to EEG/ECoG/LFP. Aggregating over all models, gamma power does not statistically predict spike precision, nor does a change to spike-field coupling imply change in spike-spike synchrony.

Revision history

Erik Peterson

May 4, 4:16 PM
Erik Peterson

May 4, 4:14 PM
Erik Peterson

May 4, 4:11 PM
Erik Peterson

May 4, 4:05 PM
Bradley Voytek
Erik Peterson

May 4, 4:05 PM
Bradley Voytek
Erik Peterson

May 4, 4:04 PM
Bradley Voytek
Erik Peterson

May 4, 4:04 PM
Bradley Voytek
Erik Peterson

May 4, 4:03 PM
Bradley Voytek
Erik Peterson

May 4, 4:02 PM
Bradley Voytek
Erik Peterson

May 4, 4:01 PM
Bradley Voytek
Erik Peterson

May 4, 4:01 PM
Bradley Voytek
Erik Peterson

May 4, 3:59 PM
Bradley Voytek
Erik Peterson

May 4, 3:59 PM
Bradley Voytek
Restore this revision

Show changes

Show less detailed revisions

This is even better

Git is a system for

keeping track of who

changed which files, in which ways, and when.

GitHub is platform / website

that makes that easy.

LINGO

Git is the software used to track edits to files.

When a tracked file is edited and saved (**git commit**), the edit is given a unique identifier.

That's how Git tracks edits: modified file(s) are selected (**git add**), and snapshots are created (**git commit**) with a unique identifier to track it.

GitHub is a platform / website where all those edits can be hosted in the cloud, instead of just living on your computer.

This makes it easier for others to edit them, too.

Let's say you're working on a group project, and that project has 3 different files.

Those 3 files live in a **repository** (or "repo") on GitHub.

A repo is just a folder containing all the files for your project.

It contains all the *history* of your project, too.

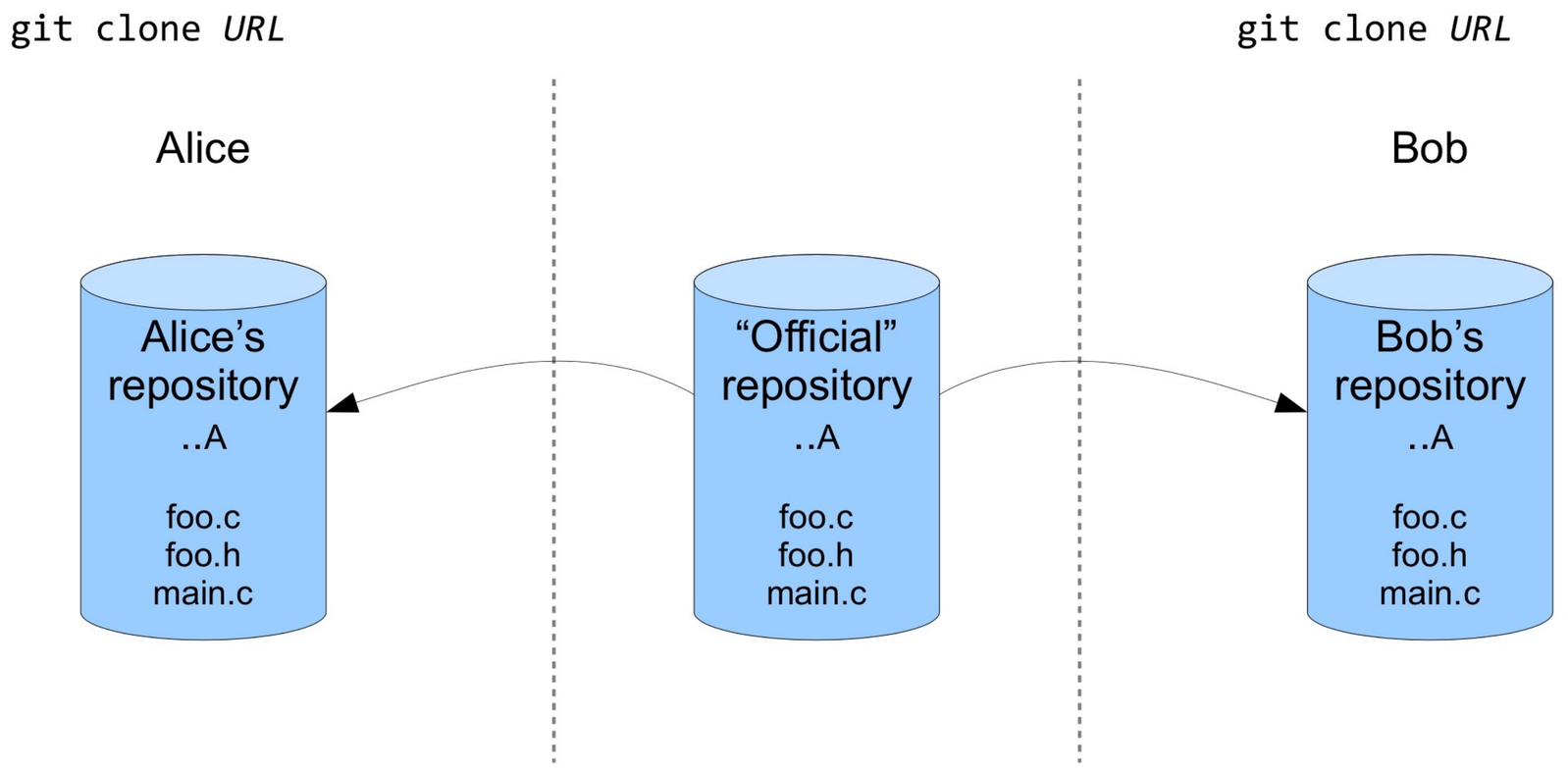
Normally when you start a project you'd create a new empty repo, with no history or files.

To join a project is already in progress you will instead **fork** or **clone** the existing repo.

A **fork** creates your own copy in GitHub.

A **clone** creates your own local copy.

Git

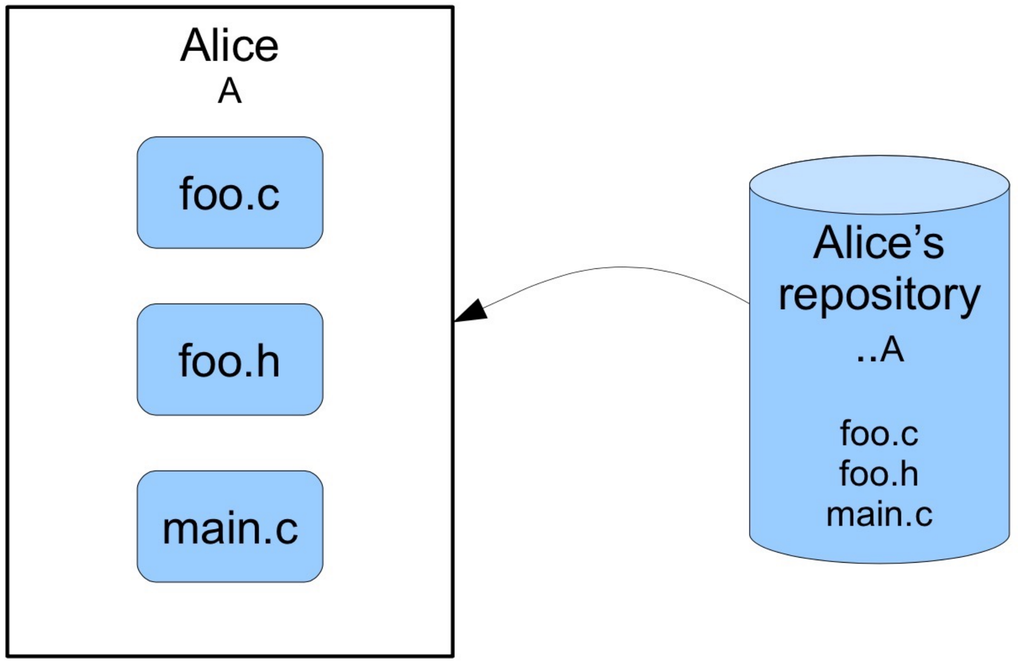


- First step is to *clone* the repository, not check it out.
- This gives you a local clone of the *entire* repo!

Cloning means you copy everything in that GitHub repo – all the files and their history – onto your computer.

This version that you now have is called your **local copy** or **working copy**.

Git



- Alice can now work locally (and offline), without worrying about other repositories.

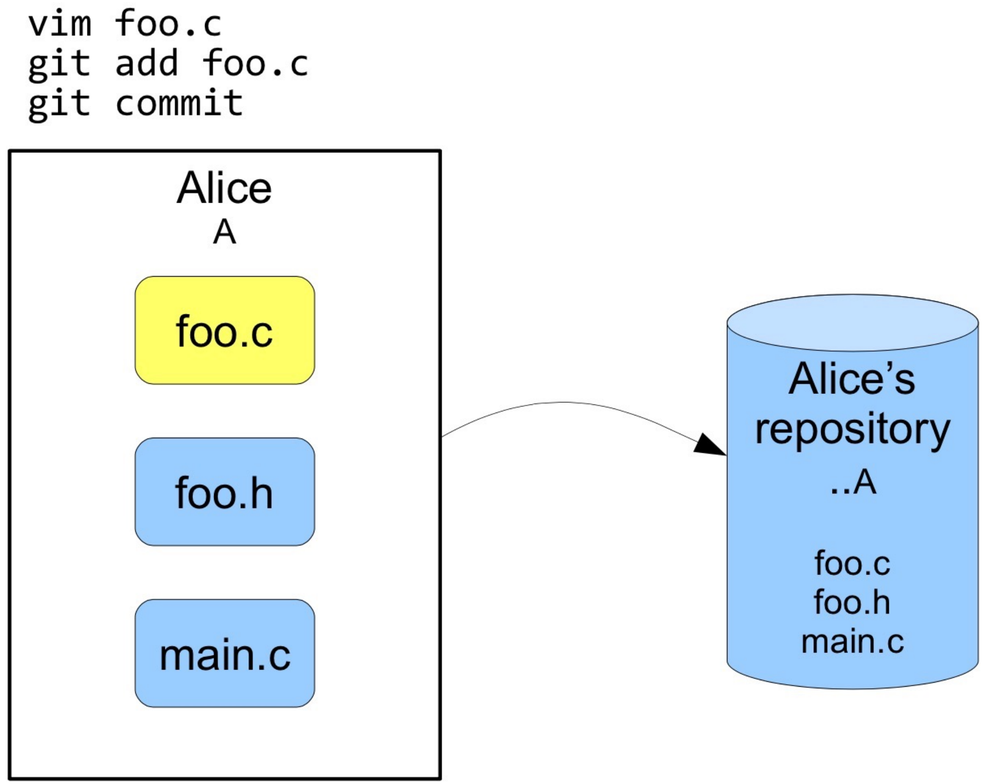
That's how Git tracks edits: modified file(s) are selected (**git add**), and snapshots are created (**git commit**) with a unique identifier to track it.

If you edit files in a project, Git will see that you've changed your working copy files.

But it's only noticed that the files have changed, it hasn't done anything with that information yet.

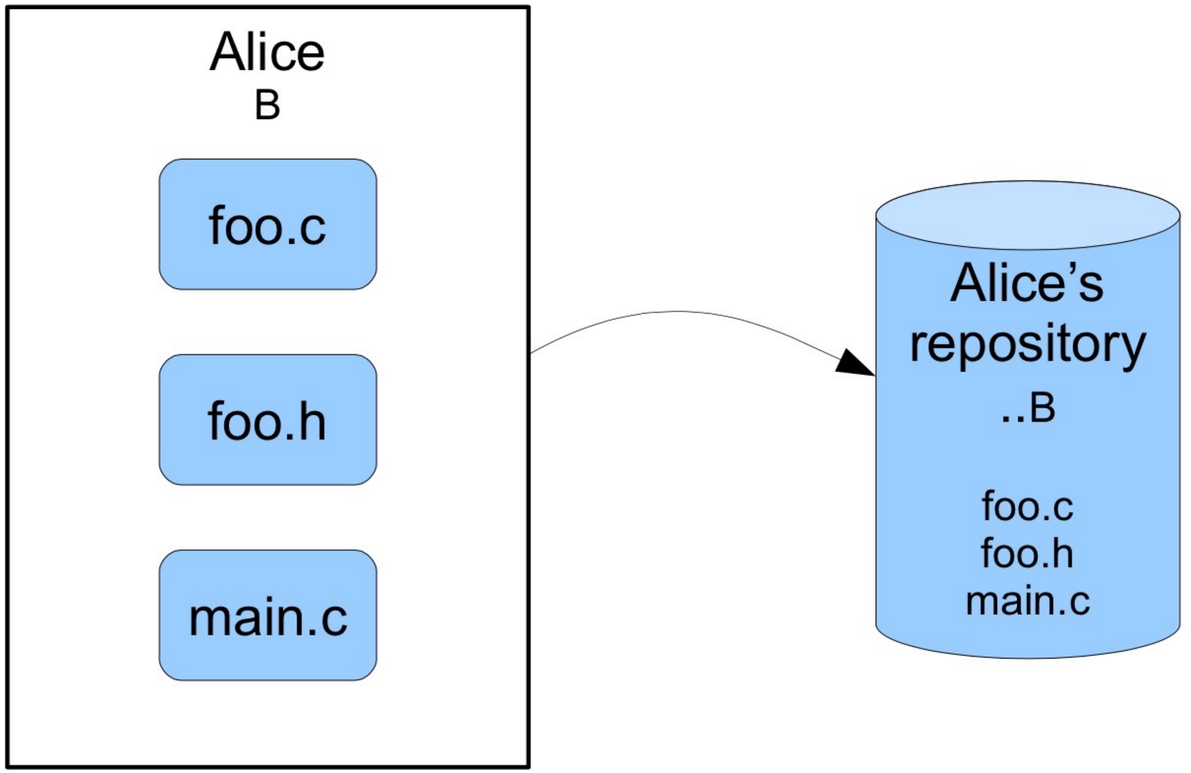
In order for Git to save a snapshot of all the changes, you have to **commit** those edits.

Git



- Alice edits foo.c as usual, adds the changes to her commit, and commits.

Git



- Revision B, based on A, is now in Alice's repository.

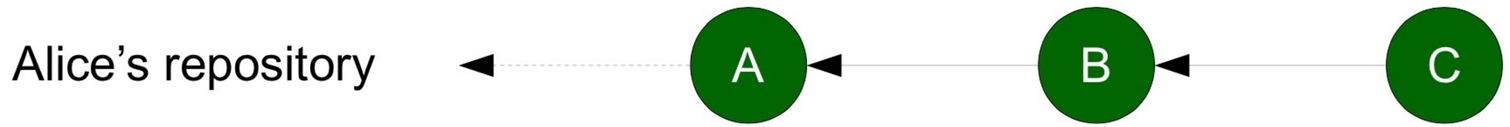
Git

- In our Git example, Alice has committed revision B "onto" revision A.

A **revision** is the state of the repository at a certain point in time.

You can always roll back (**git reset**) to a prior revision (like, if someone breaks everything…)

Git



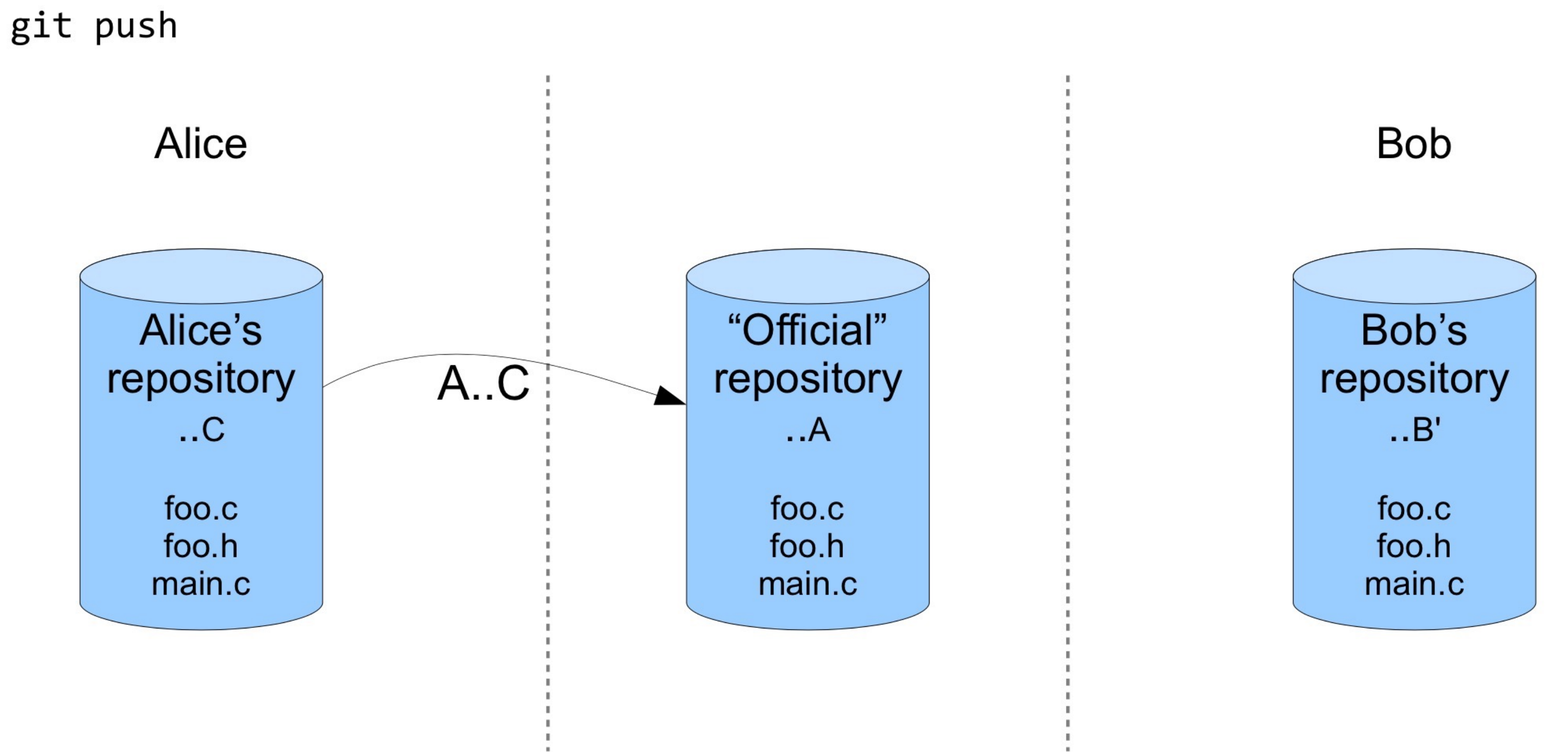
- In our Git example, Alice has committed revision B "onto" revision A.
- She can then commit another revision C onto that.

Once you've committed, you can then choose to share those files with the world, and get those edits/commits off of just your computer and put them onto GitHub.

To do that, you **push** those commits to GitHub,
saving your changes back to the repository

in the **CLOUD**.

Git



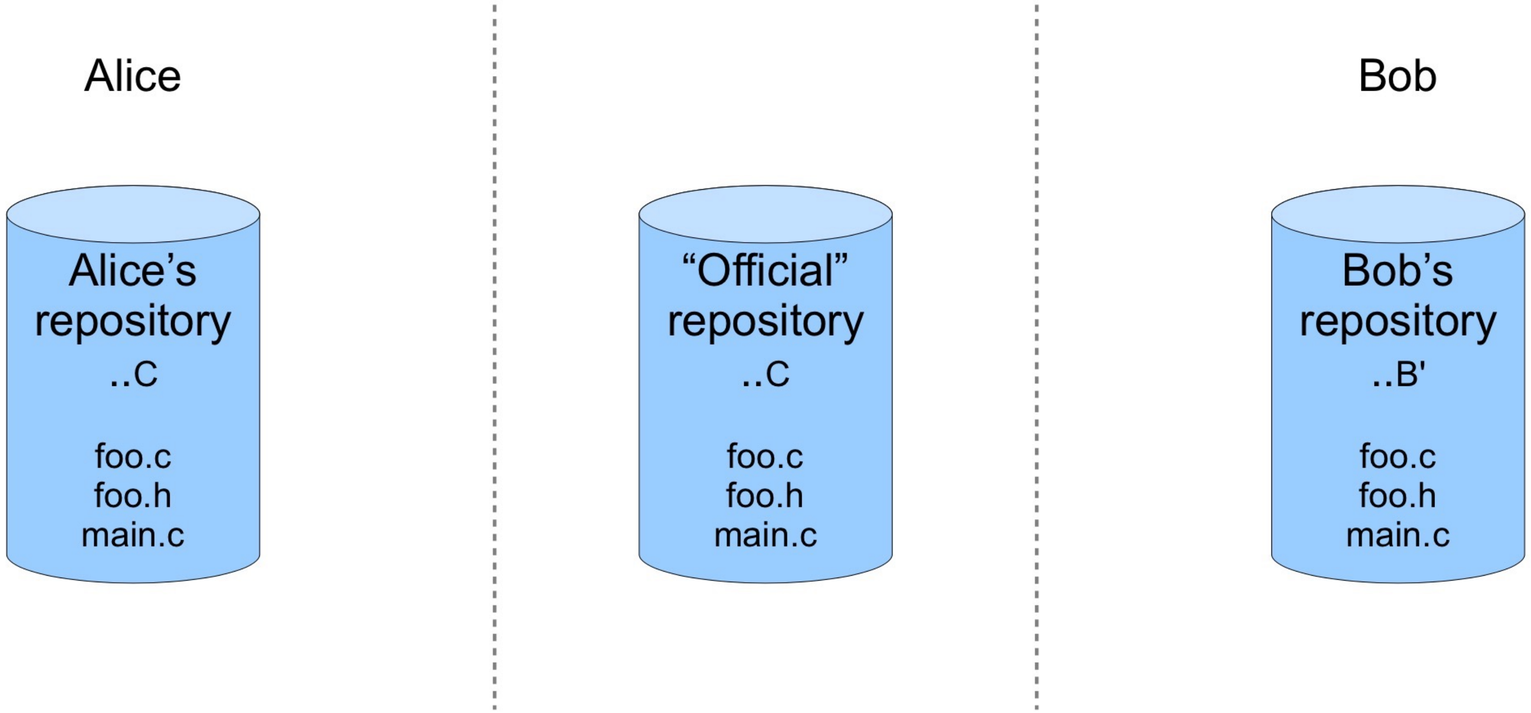
- Alice can *push* her new commits to another repository, such as the "official" one.
- The commit being pushed must be a *descendant* of the remote one.

When you push changes, the new version, *C*, is created, which is a **child** of both *B* and *A*.

This idea of being a "child of" is part of what's called a **dependency graph**, where you can only push code that is a **direct descendant** of the official repo.

Usually this occurs without any problems.

Git

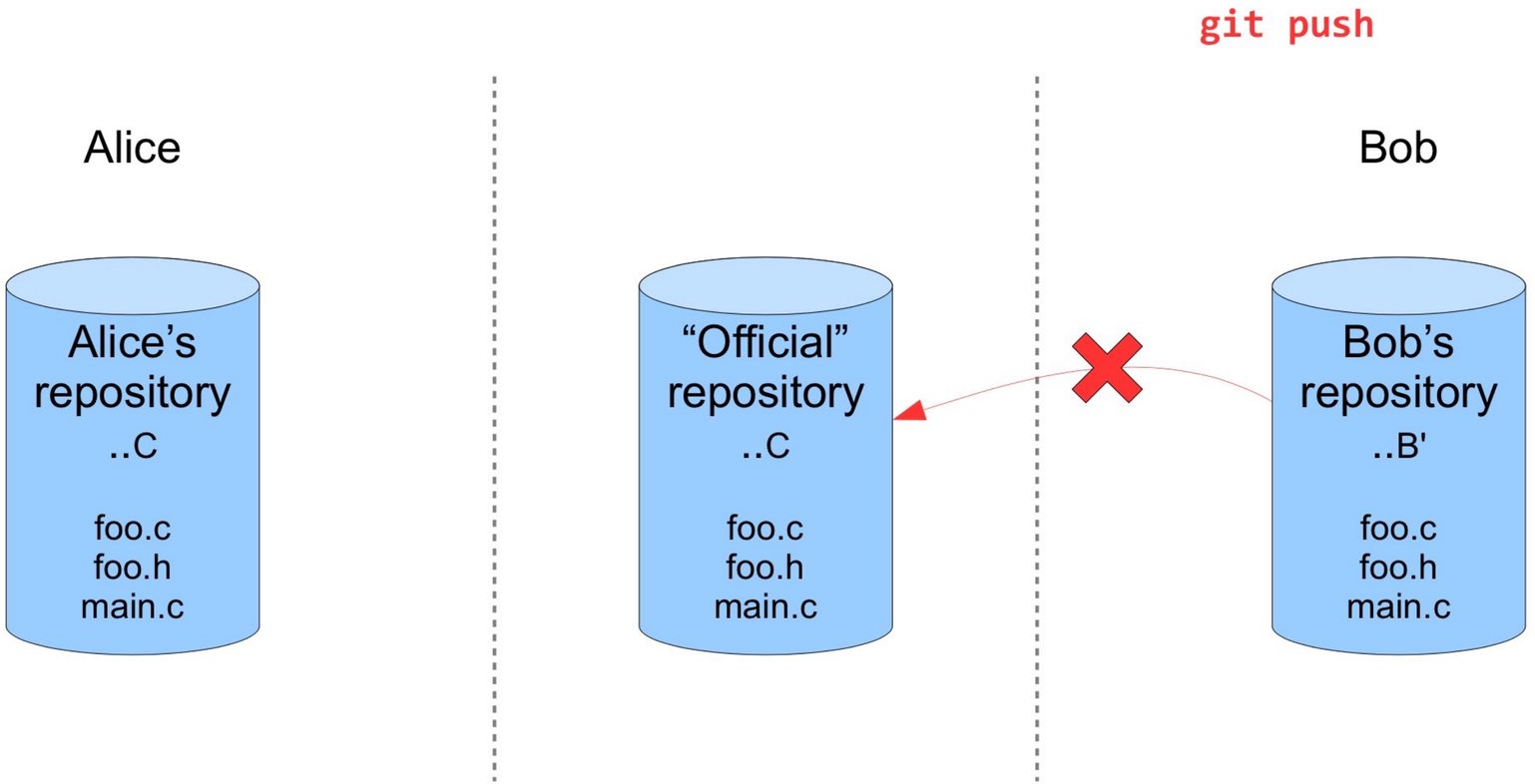


- The official repository now contains Alice's commits.
- Notice Bob has also committed B' but not yet pushed!

Git

- Bob's commit graph has his new revision, but none of Alice's.

Git



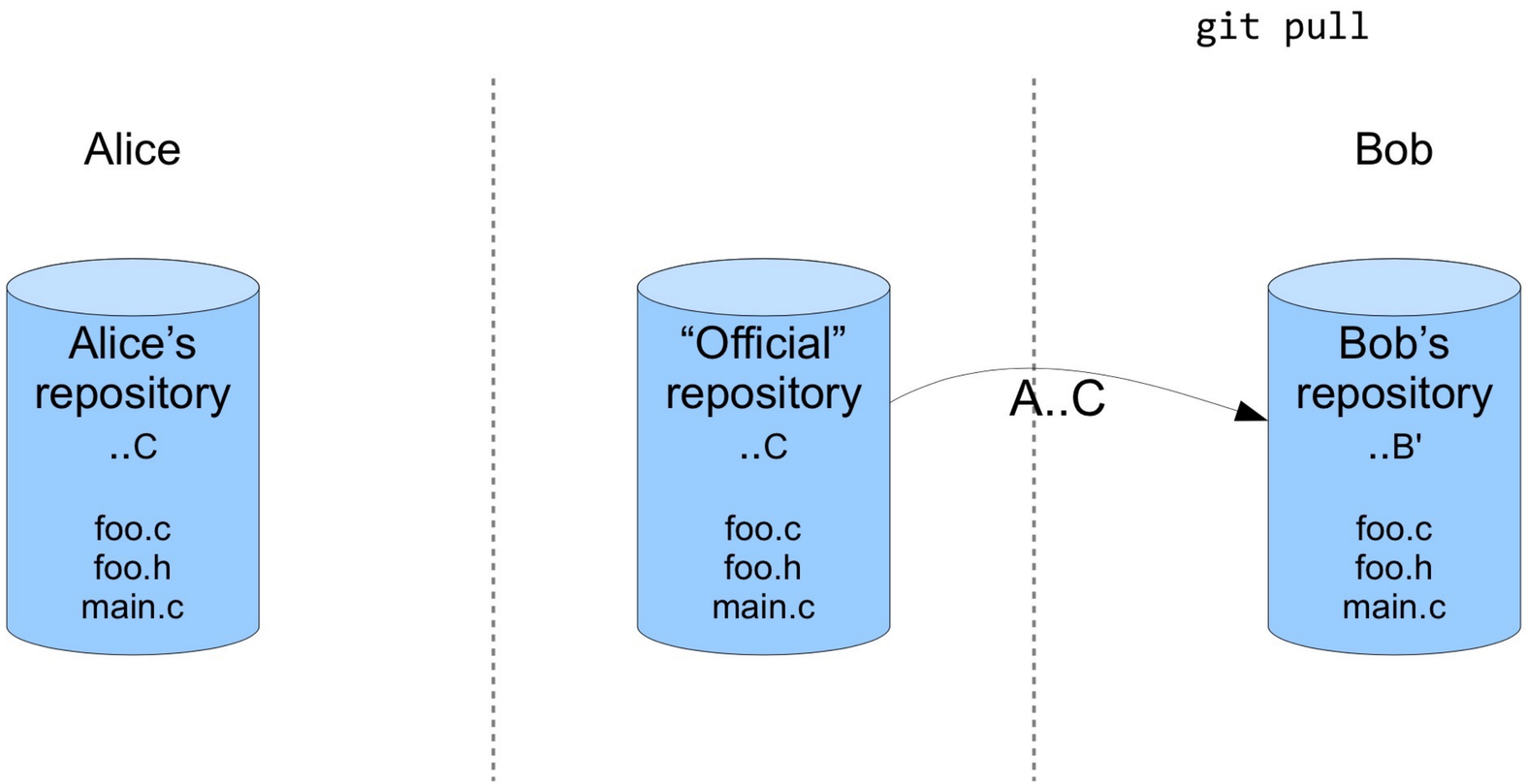
- Bob cannot push his changes yet, because B' is not a descendant of C.

Bob had *A*, but changed *A* to become B′.

His edits are not a direct descendant of the official repo version (now *C*).

Here you have to pull the new revision *C* from the GitHub repo, which then tries to merge the B′ edits with the latest version on GitHub.

Git



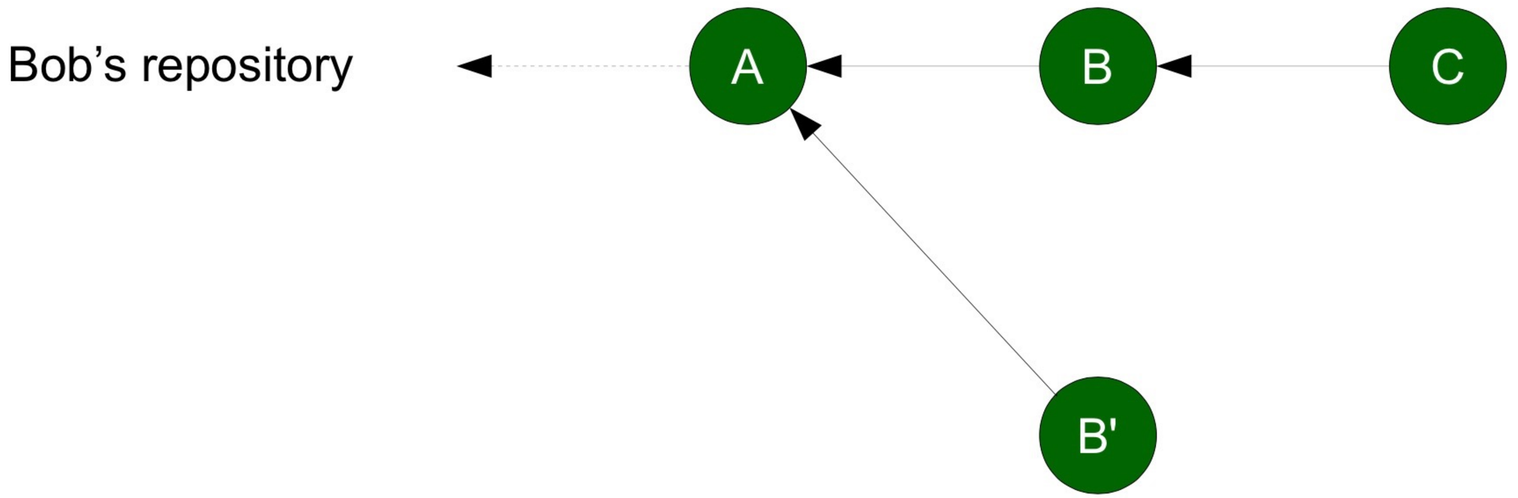
- Bob needs to get Alice's new changes and merge them.
- First he *pulls* the changes from the official repo...

That is, you combine two sets of changes to the files in your project. (Note that **pull** grabs the files and tries to **merge** them, in one step.

Alternatively, you can **fetch** the files, which just downloads the new data, but doesn't **merge** (integrate) those data with your working copy.

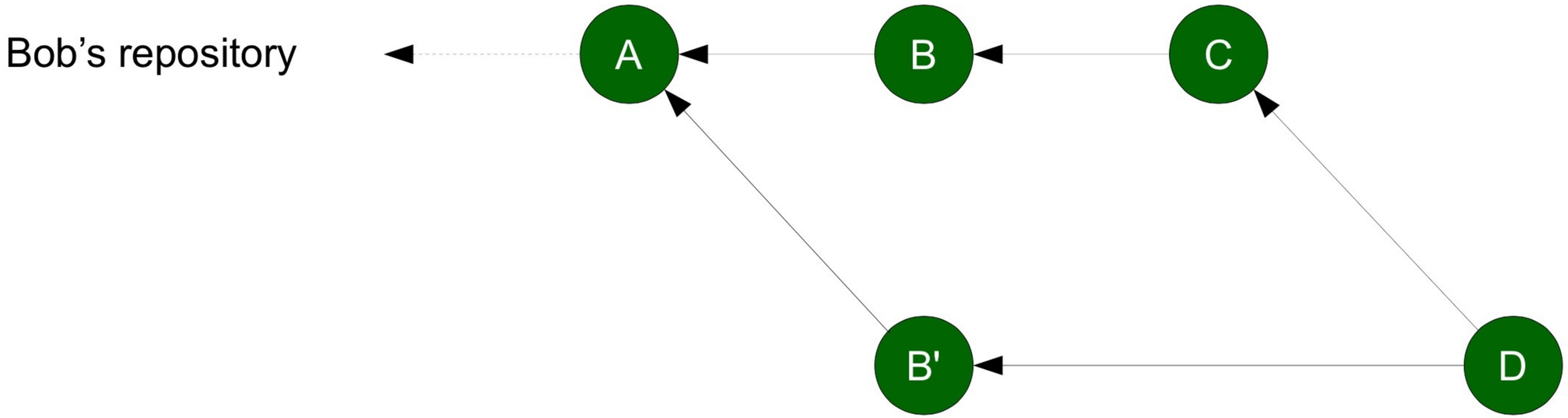
Some people prefer to fetch-then-merge instead of pull, but this is just a personal choice.

Git



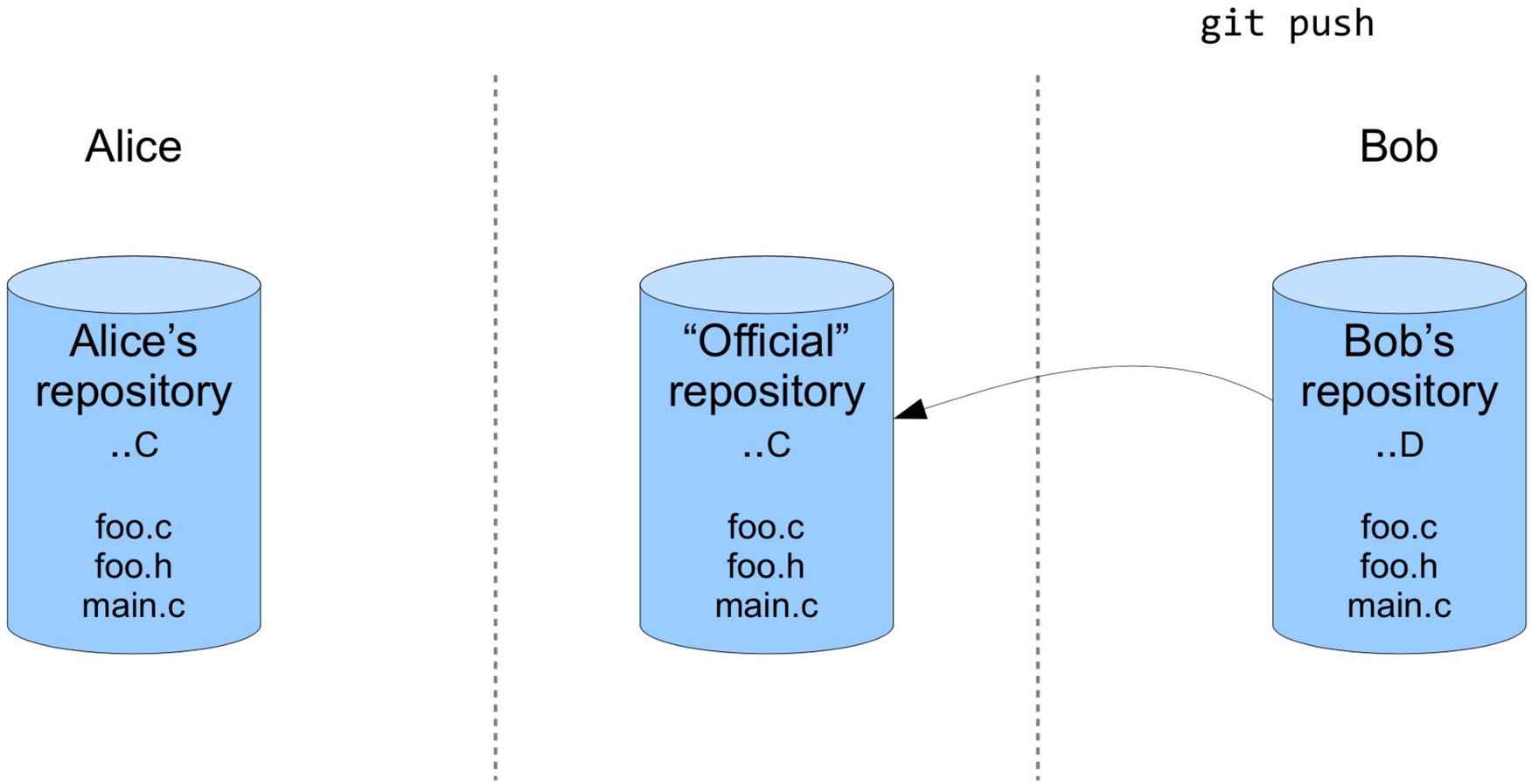
- ... and they are added to his repository.
- He can now *merge* Alice's changes with his.

Git



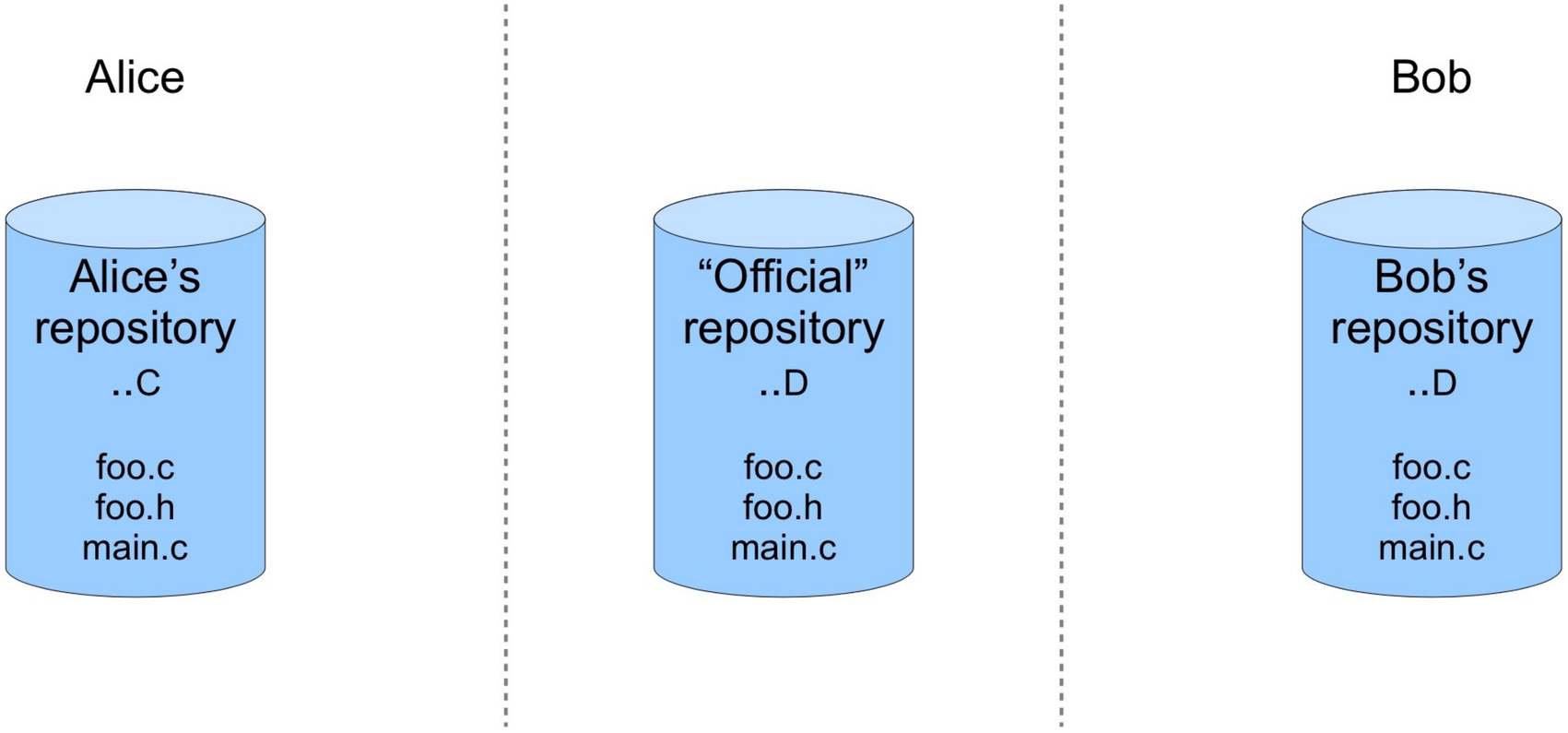
- This creates a new *merge revision* D which is a child of both B' and C.

Git



- Since D *is* a descendant of C, Bob can now push!

Git



- The official repository now has revision D, which contains both Alice's and Bob's changes.

If you happened to edit the same exact lines of the same exact files as the latest revision, this creates a **merge conflict**, and someone has to step in and look at what those differences are and decide how to combine them together.

If you want folks on your team to look at your edits, make comments on them, review your code, etc. you can initiate a **pull request** on GitHub, which tells people:

"hey, pull this new revision, test it out, and let me know what you think."

If you want to try out some side experimental thing – like adding some new feature – without breaking the official code base for everyone else, you can create a **branch** off of the repo at some point in its revision history.

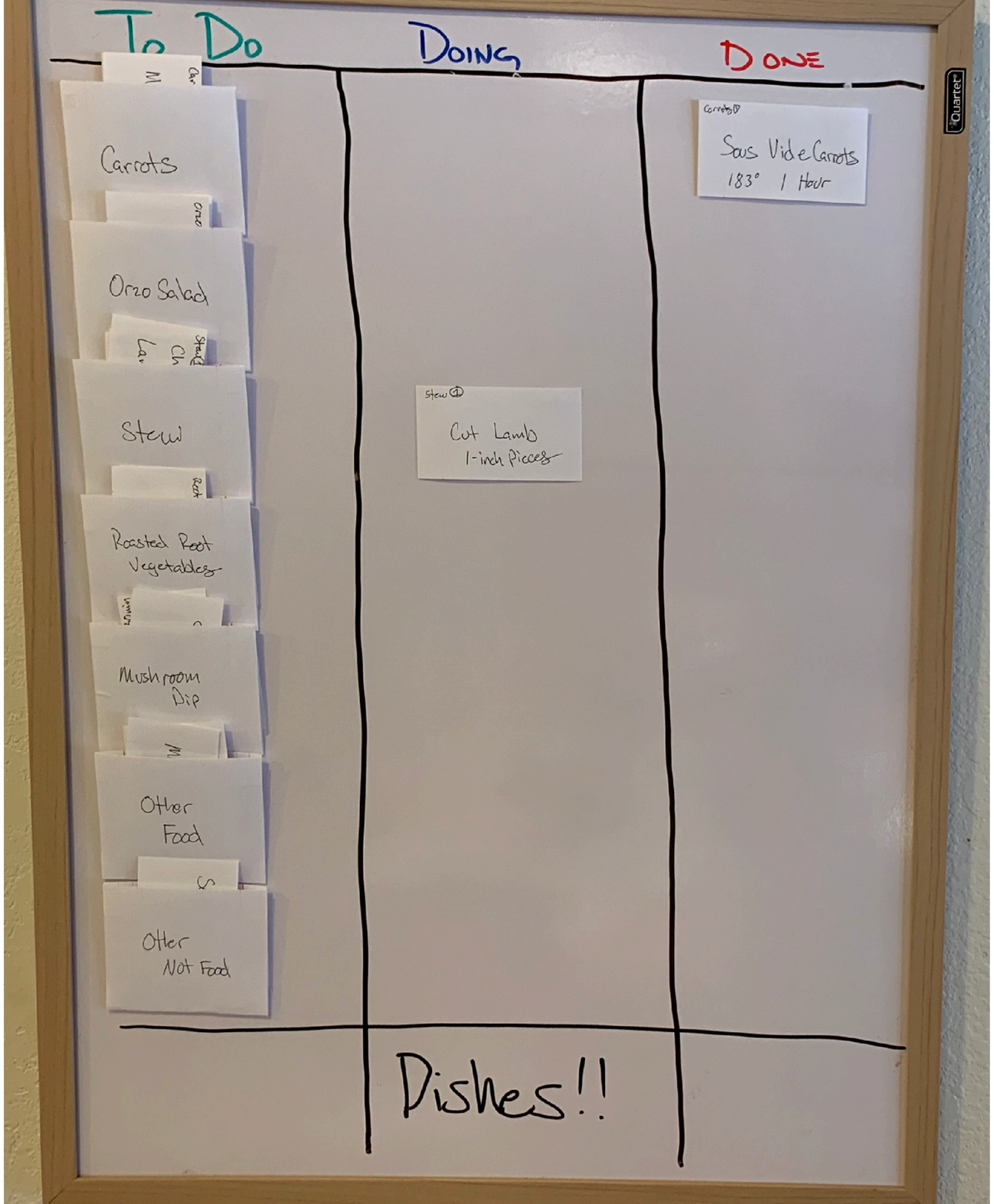
This lets you and your team mess around and try things out without breaking the official version.

Once you all are happy with things, you can then merge that branch back onto the main branch to fold in all of that branch's changes, history, etc. into the main repo.

Project Management

AndrewBender.md	Complete Andrew's project list
BlancaMartin.md	Update BlancaMartin.md
BradleyVoytek.md	Update BradleyVoytek.md
ChristianCazares.md	Update ChristianCazares.md
DillanCellier.md	Update DillanCellier.md
EenaKosik.md	Update EenaKosik.md
MichaelPreston.md	routine update to MichaelPreston.md
PamRiviere.md	added theta shape proj
Quirine-vanEngen.md	Update Quirine-vanEngen.md

Kanban Board!


To Do
Doing
Done
Carrots
Orzo Salad
Stew
Roasted Root Vegetables
Mushroom Dip
Other Food
Other Not Food
Cut Lamb 1-inch Pieces
Sous Vide Carrots 183° 1 Hour
Dishes!!