# What They Forgot to Teach You About R

RStudio

2018 October Seattle

Jennifer (Jenny) Bryan
RStudio (& UBC)

@jennybc
@JennyBryan

Jim Hester
RStudio

@jimhester
@jimhester_

Alison Hill
OHSU (& special friend of RStudio)

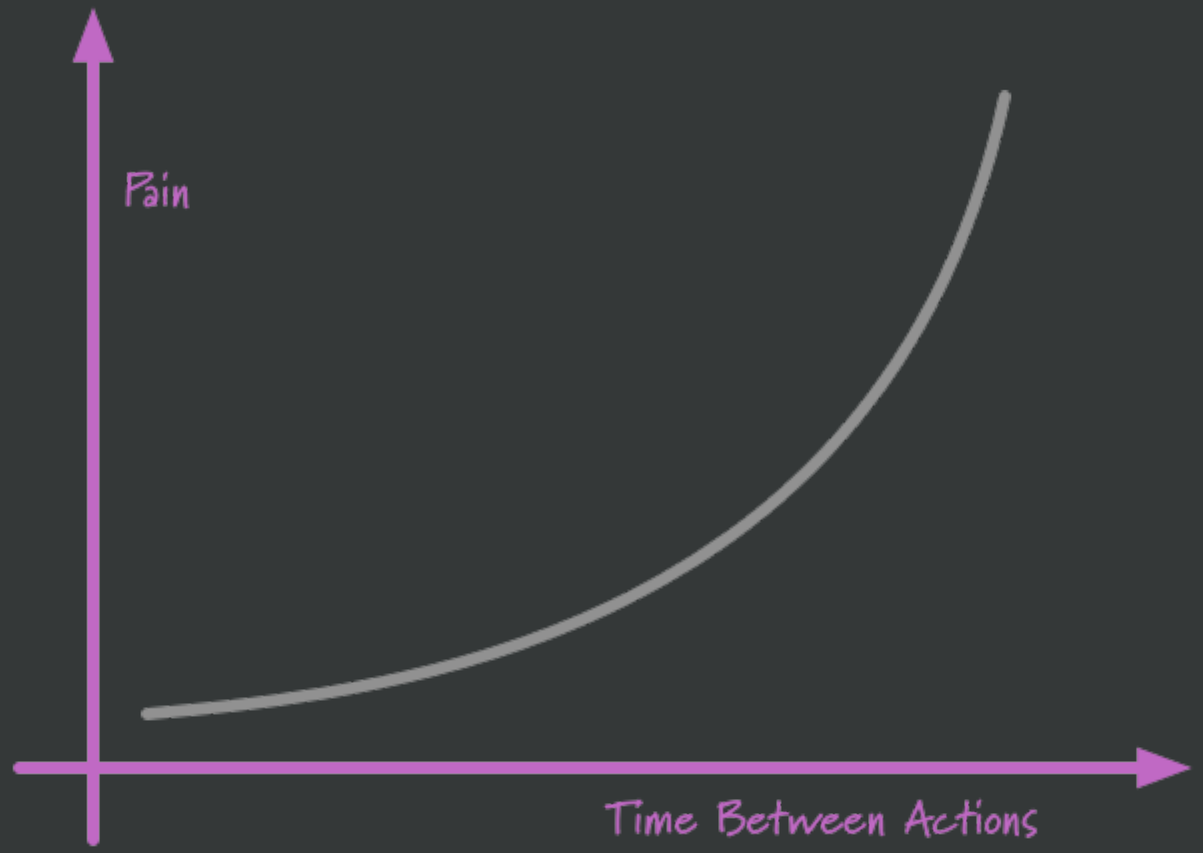
apreshill
apreshill

rstd.io/wtf-seattle

Let's deal with all the accumulated Git dilemmas and preview some intermediate workflows you'll need soon.


Deep
Thoughts

# "If it hurts, do it more often."

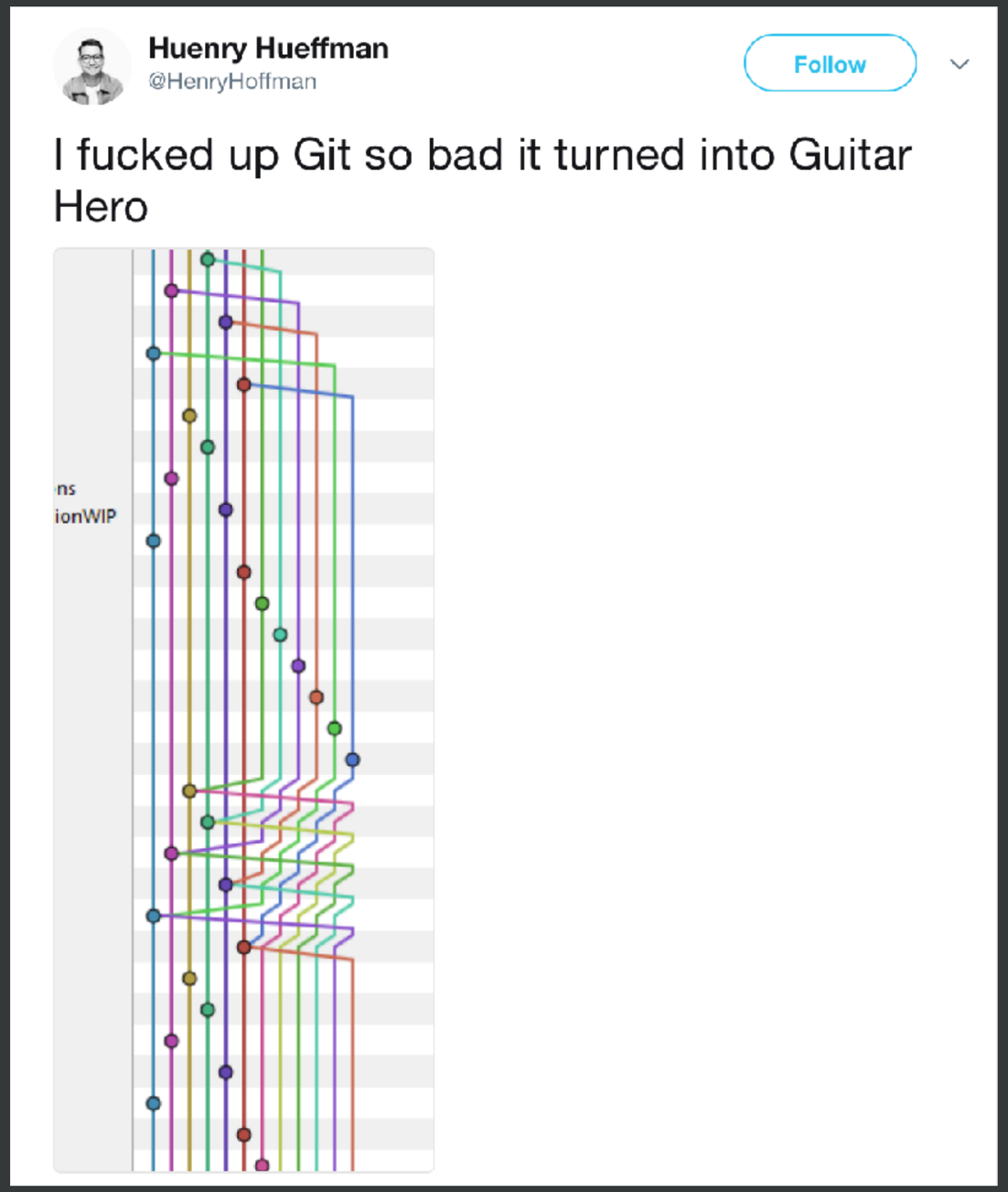



https://martinfowler.com/bliki/FrequencyReducesDifficulty.html

"If it hurts, do it more often."

Apply this to git commit, pull, merge, pull.
(and restarting R, re-running your scripts)

Why?
Take your pain in smaller pieces.
Tight feedback loop can reduce absolute pain.
Practice changes what you find painful.




https://twitter.com/henryhoffman/status/694184106440200192

You do NOT want "Guitar Hero" git history.

The longer you wait to integrate, the harder it will be.


Deep
Thoughts

# Recovering from Git(Hub) failure

Scenario: You have a huge mess you cannot fix.

Official answer: git reset.

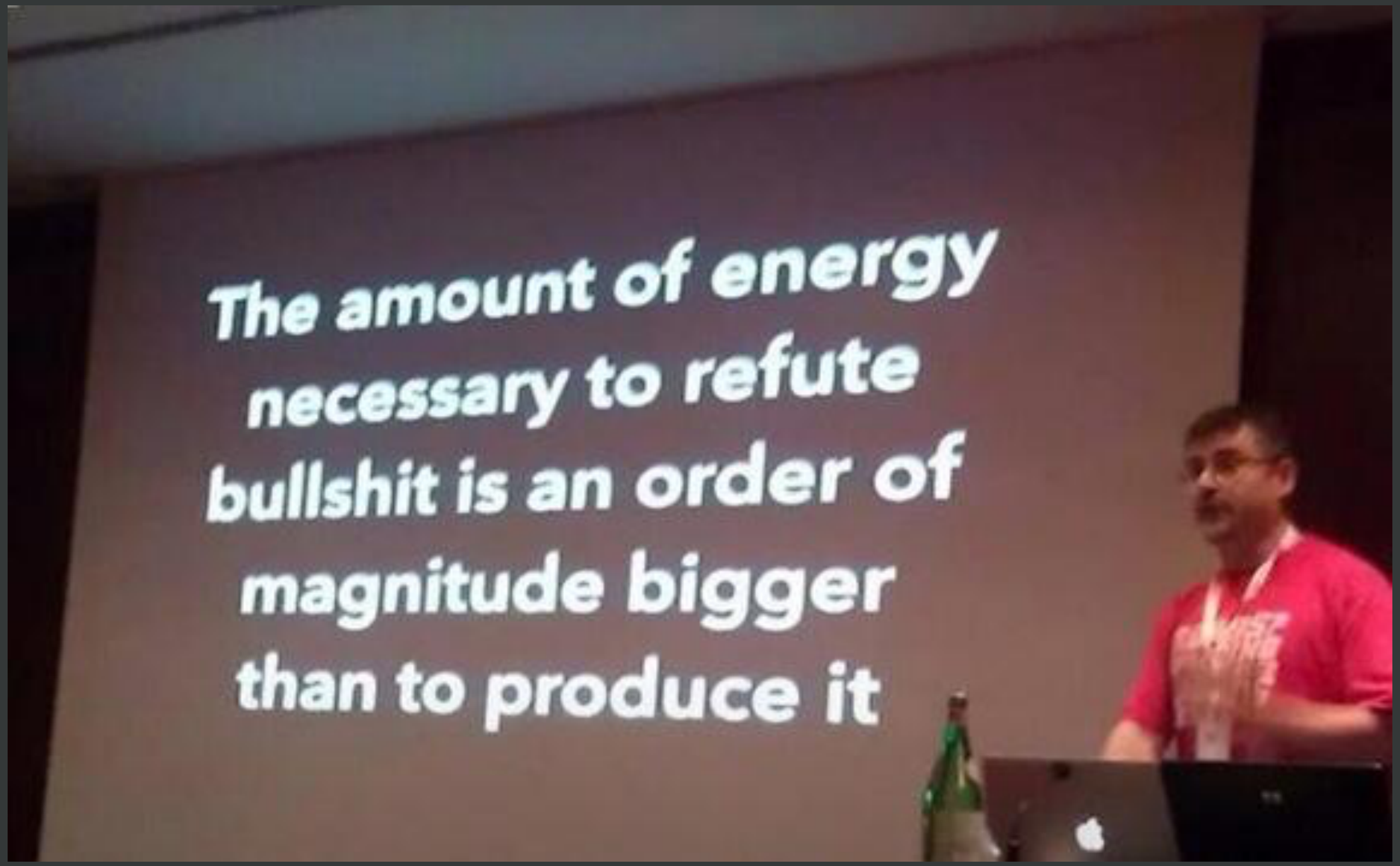
Unofficial answer: burn it all down 🔥

So I can face Jim Hester when he sees this:

`git reset` (mixed and hard) is genuinely worth learning.

SourceTree, for example,  makes it easy to do hard or mixed resets to previous states.

After you reset to a non-broken state, have another go at whatever you were doing.




- Alberto Brandolini

The amount of Git skilz necessary to fix a borked up repo is an order of magnitude bigger than to bork it.

- Me


BURN IT ALL DOWN

🔥 requires you have a remote repo in a decent state!
Commit early, commit often! And push! It's your safety net.

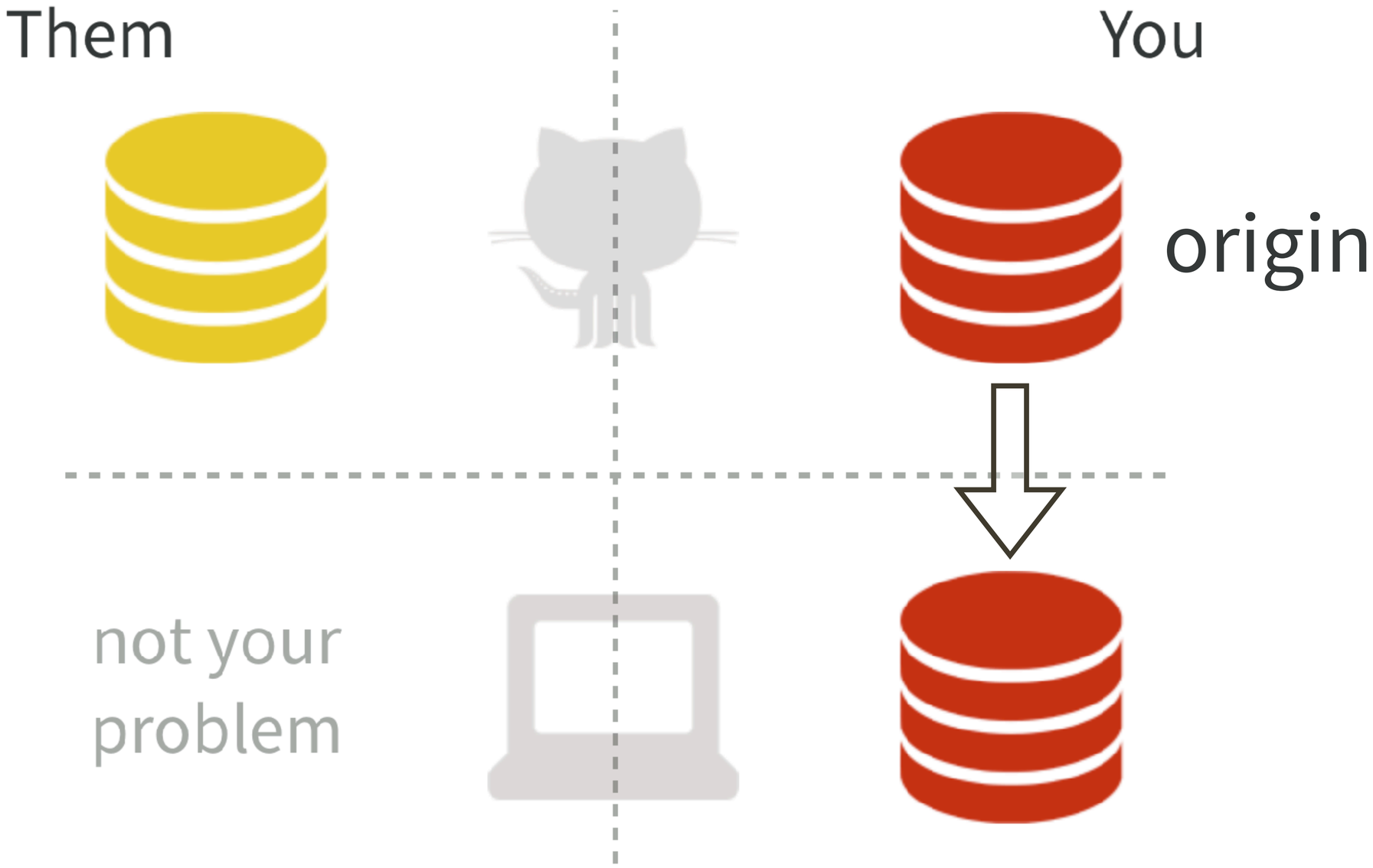
Rename local repo to, e.g. "foo-borked".

Re-clone to a new, clean local repo, "foo".

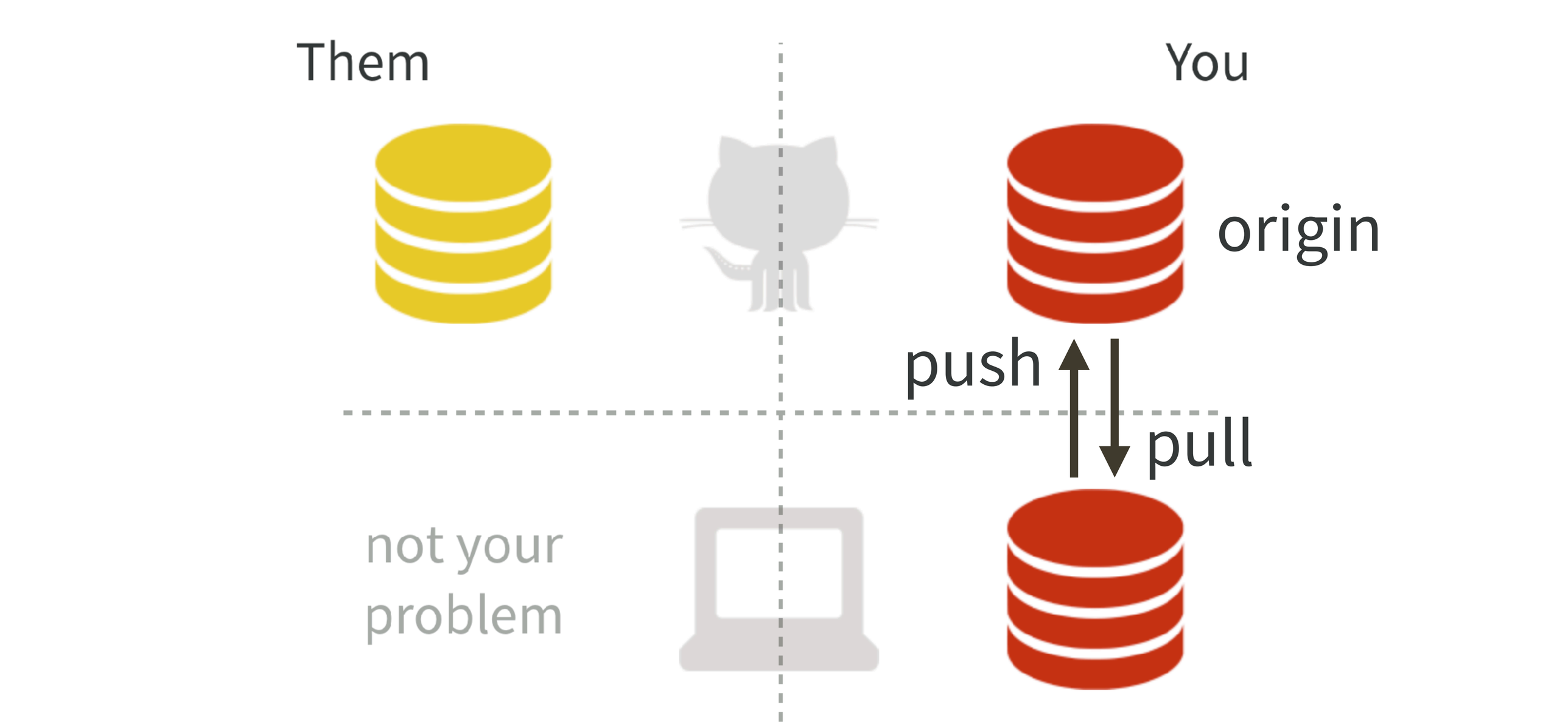
Copy any files that are better locally from "foo-borked" to "foo".
Commit. Push. Carry on.


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Why do you have to care about remotes, eventually?

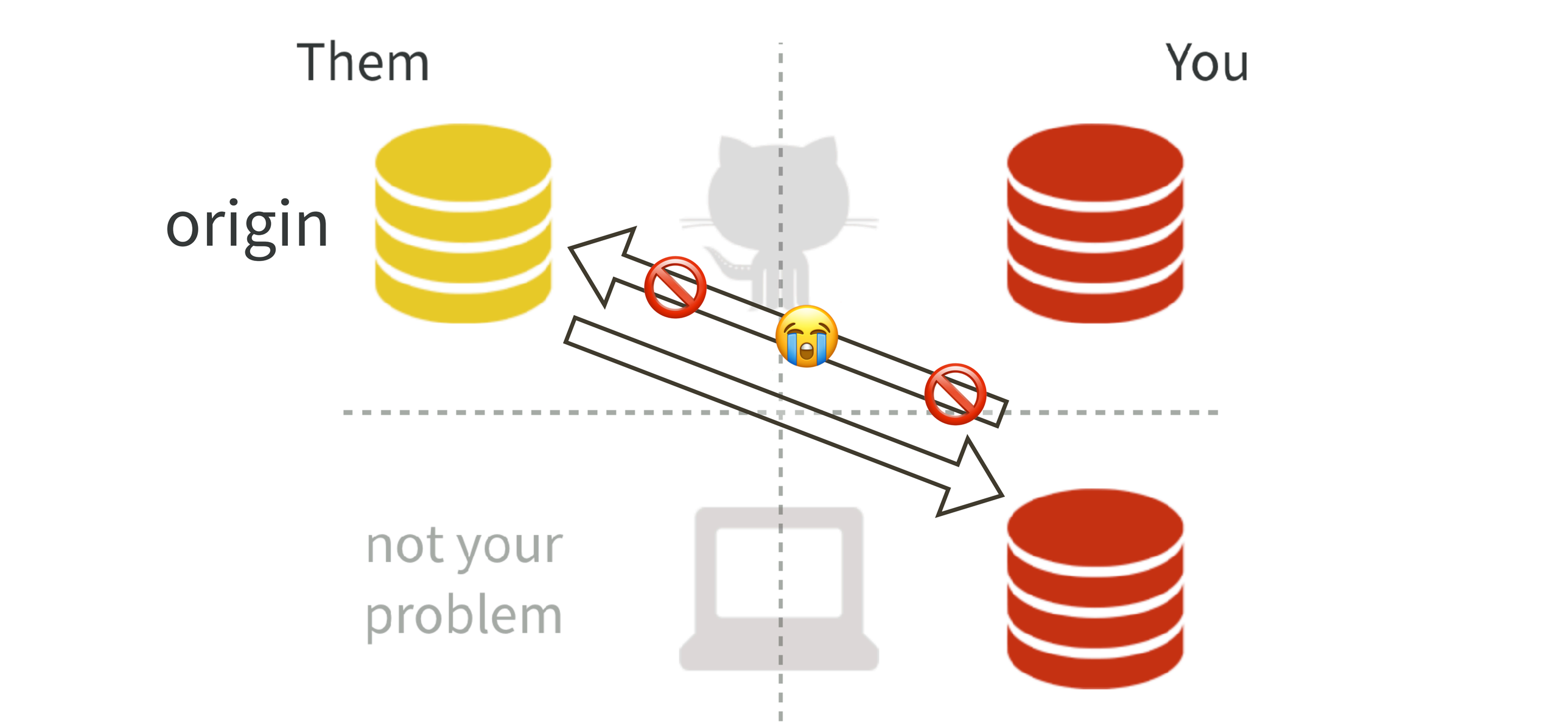



"clone"



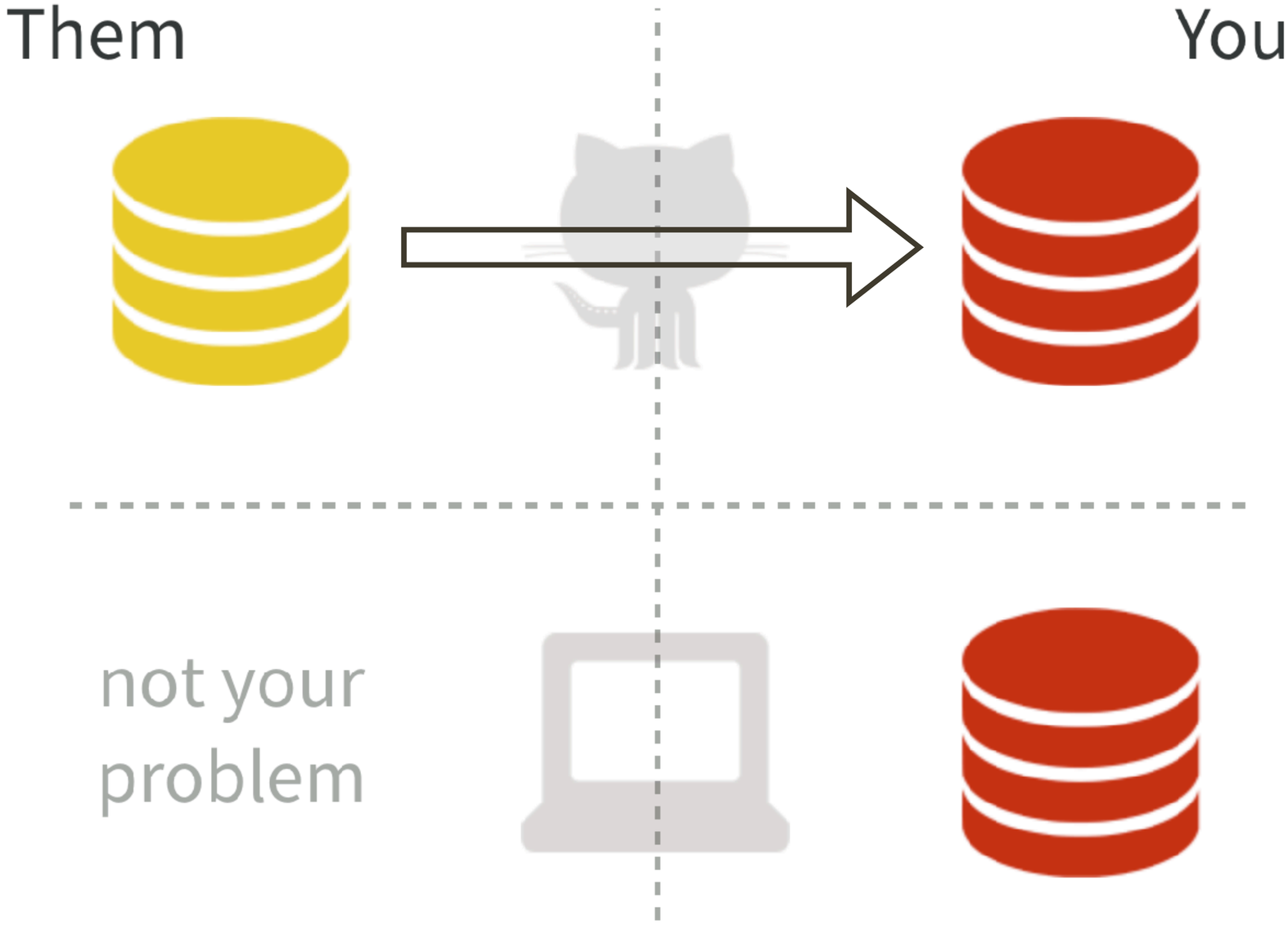

# daily work, your stuff


"clone" *not as useful as you might think because you can never send a PR

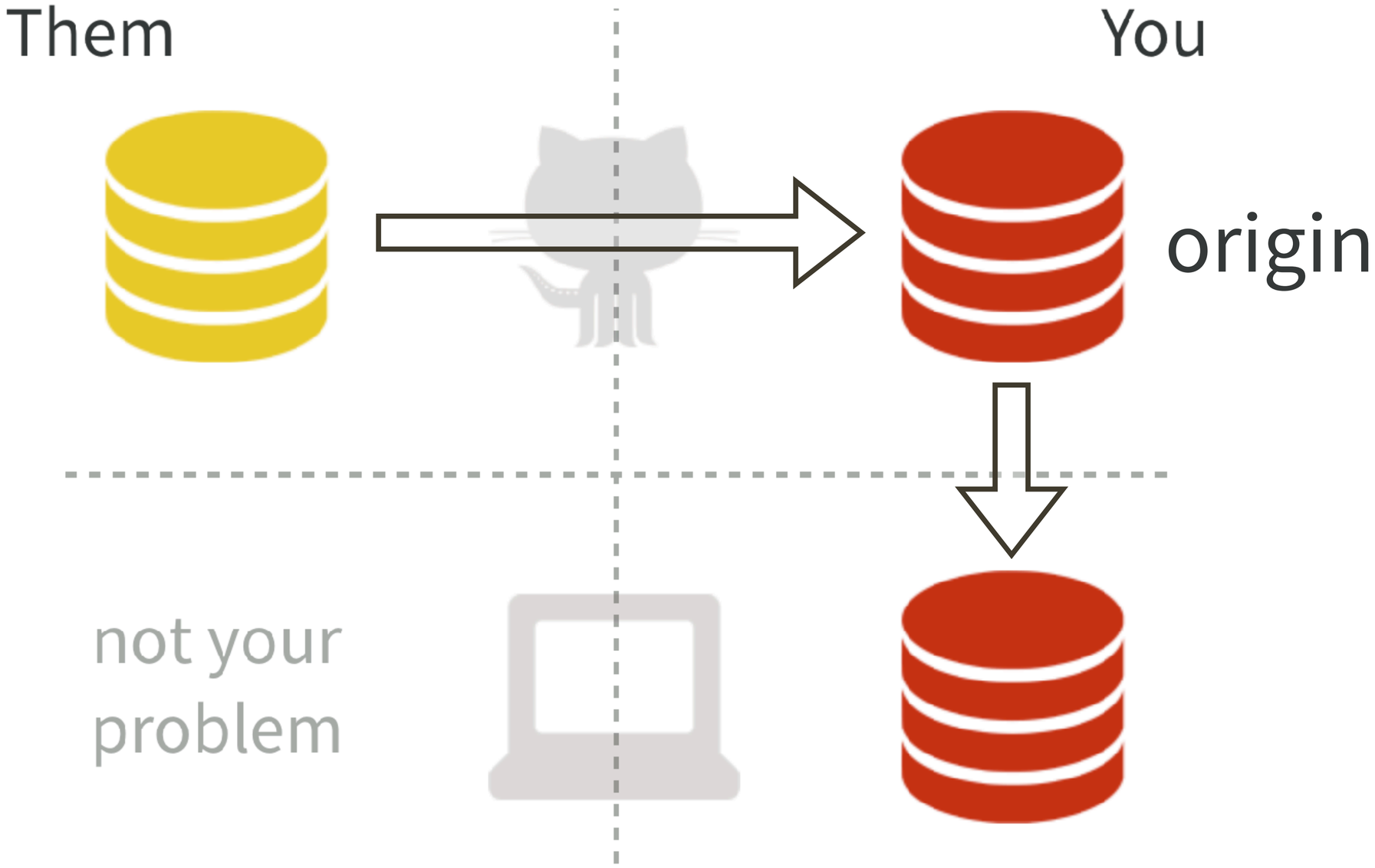



# "clone"

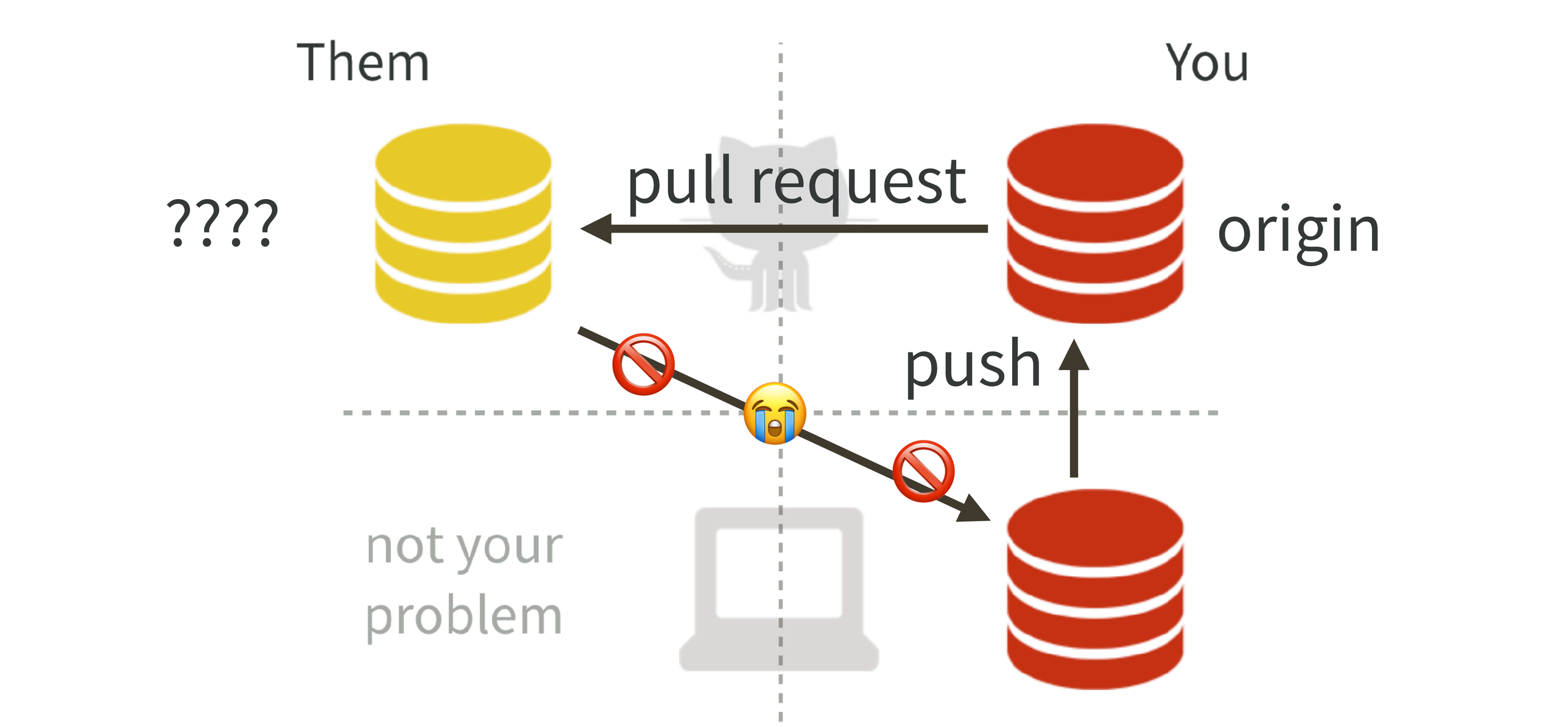
*not as useful as you might think because you can never send a PR



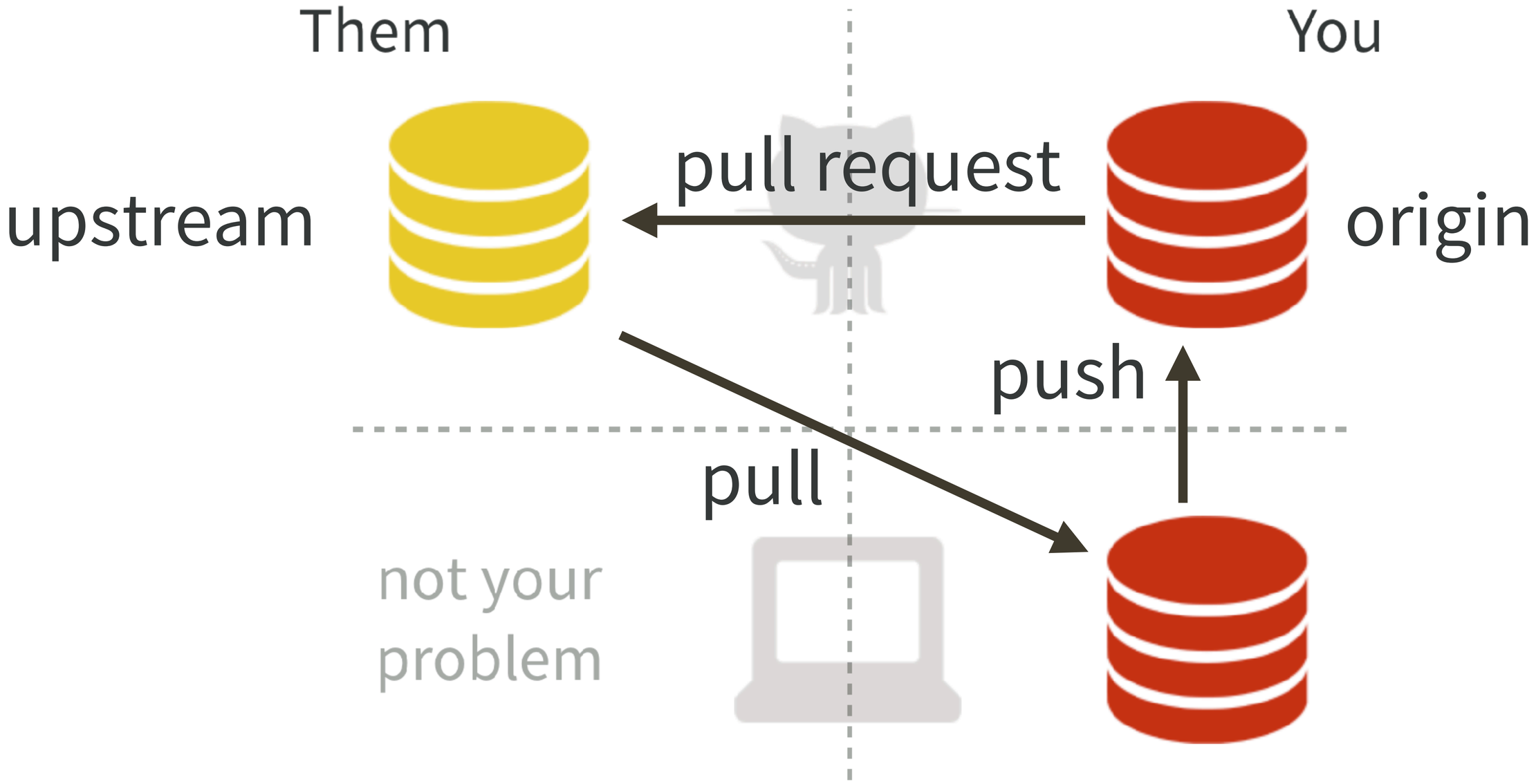

"fork"




# "fork and clone"




# get changes from the main repo




# get changes from the main repo

Scenarios when you need to add a remote:

Add the main repo as a second remote, typically nicknamed "upstream" (fork and clone workflow, 2 months later, you need to re-sync)

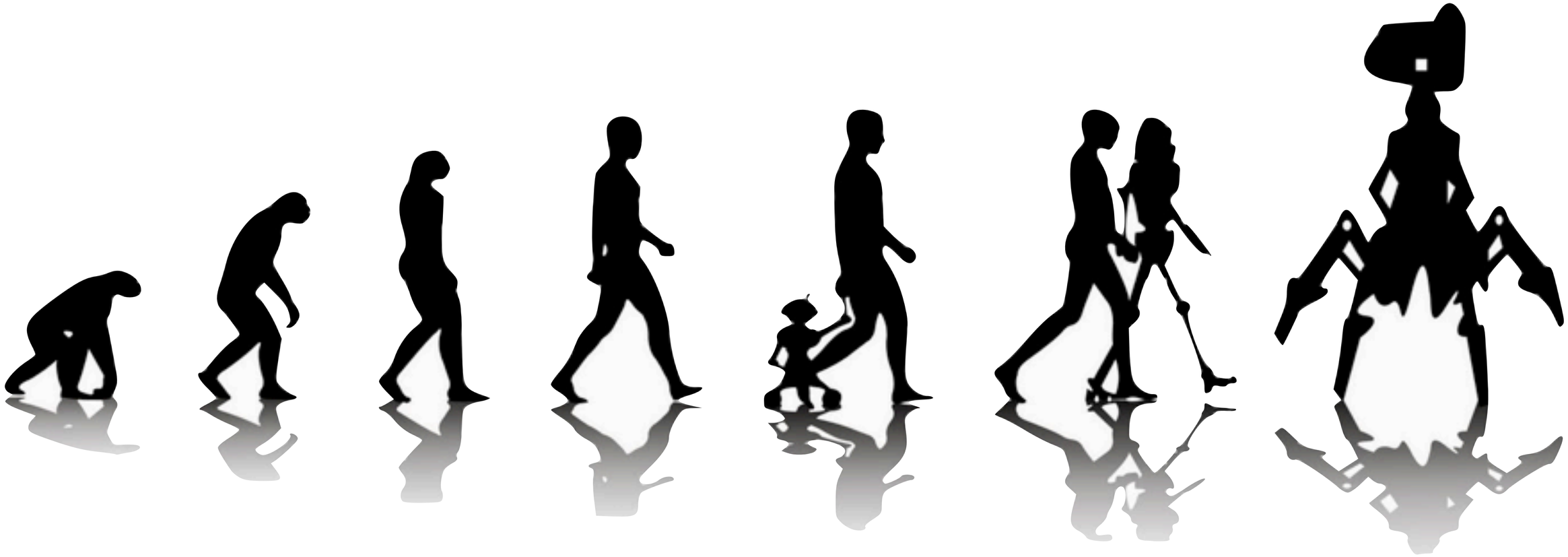
Add your fork as a second remote, when you did "clone" and, in hindsight, you wish you'd done "fork and clone"

"Burn it all down", the Fork version 🔥

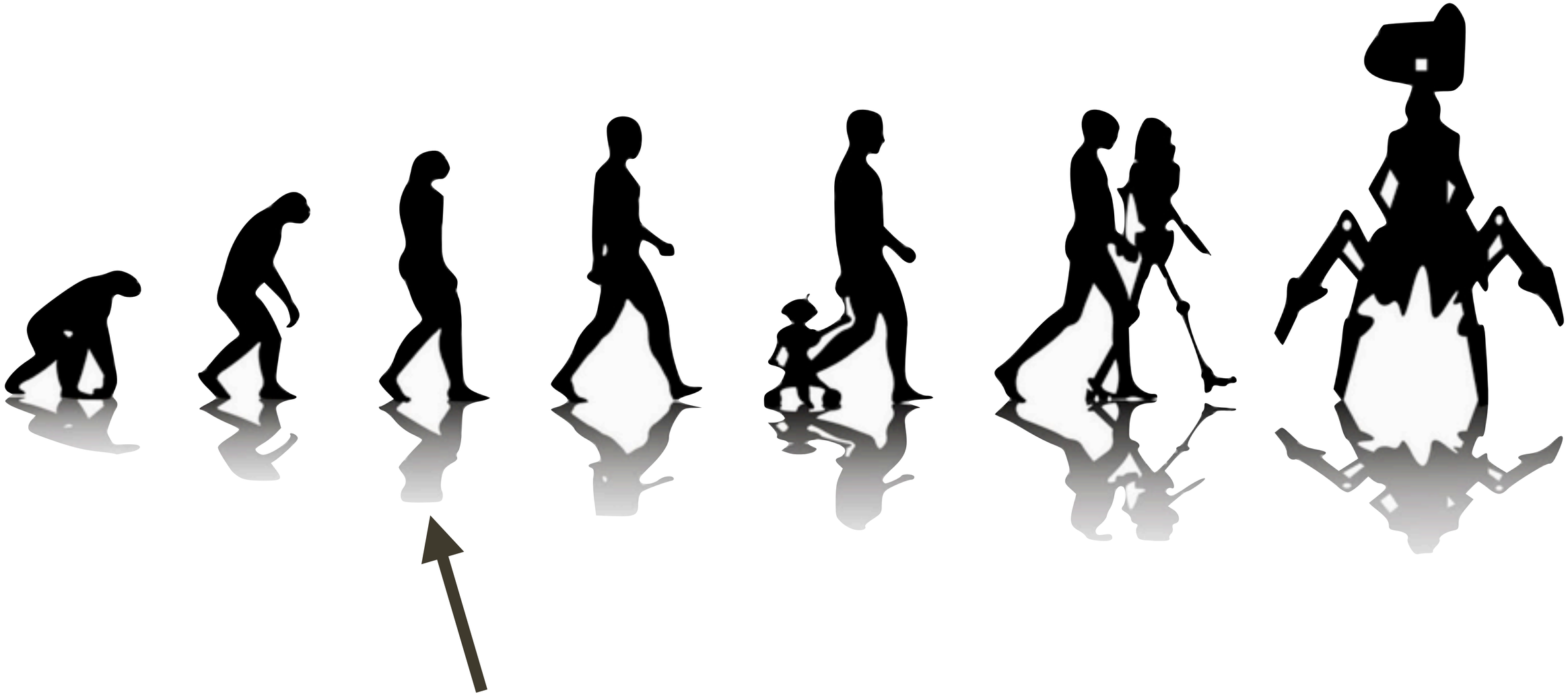
If you contribute to a repo once every 4 years, you can also just delete your (old) fork and your (old) local repo and start over (fork, clone, edit, push, PR).

Obviously does not apply to a repo to which you regularly contribute.

Why do you have to care about branches, eventually?



"Git is great because you have the entire history of your project."



OK, but how do you *actually* go back in time?

# Levels of Git Time Travel

| | |
|---|---|
| "I just need to see the past." | Browse & search on GitHub. |
| "I need to visit the past." | Create and checkout a branch. |
| "I want to return to the past." | Revert (or reset). |
| "I had a great cookie last October." | Cherry pick or checkout a path. |
| "I want to change the past." | 🐲 *there be dragons* 🐲 |

# `git push --force`

For the purposes of this workshop, we consider this forbidden.

It can be useful -- we use it! -- but requires care.

Not a great idea for early days with Git and GitHub.

main source:

https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified

| | HEAD | Index | Workdir | WD Safe? |
|---|---|---|---|---|
| **Commit Level** | | | | |
| `reset --soft [commit]` | REF | NO | NO | YES |
| `reset [commit]` | REF | YES | NO | YES |
| `reset --hard [commit]` | REF | YES | YES | **NO** |
| `checkout [commit]` | HEAD | YES | YES | YES |
| **File Level** | | | | |
| `reset (commit) [file]` | NO | YES | NO | YES |
| `checkout (commit) [file]` | NO | YES | YES | **NO** |

"I just need to see the past."
Browse & search on GitHub.

live: browse history on GitHub
for a repo we created yesterday

"I need to visit the past."
Create and checkout a branch.

live: for a repo we created yesterday, locally, time travel by "create & checkout" of a branch

return to present: git checkout master

## Creates branch "branchname" at a certain state

```
git branch branchname <sha1-of-commit>
git branch branchname HEAD~3
```

## Creates and checks out branch! W00t.

```
git checkout -b branchname <sha1-of-commit or HEAD~3>
```

## Go back to master

```
git checkout master
```

https://stackoverflow.com/questions/2816715/branch-from-a-previous-commit-using-git

"I want to return to the past."
Revert = make a new commit that reverses a commit. Do this to undo something that has been pushed.

live: for a repo we created yesterday,
make a commit then revert it

```
git revert --no-edit <sha1-of-commit>
```

"I want to return to the past."
Reset. Safe only for work that has not been pushed.

live: for a repo we created yesterday,
make a change, don't commit, and dismiss it
make a commit then undo via reset

Dismiss current uncommitted changes

```
git reset --hard
```

Or, frankly, I always use "Discard All" in RStudio or "Discard file" in SourceTree

Un-commit last commit, but keep the changes

```
git reset HEAD^1
```

Or, frankly, I always use SourceTree to do this

"I had a great cookie last October."
Cherry pick a whole commit or
checkout a specific file from a specific commit.

live: for a repo we created yesterday,
make a branch and make a commit on it
go back to master
cherry pick that commit
pick an earlier commit and restore a specific file to that version

## apply a specific commit to current branch

```
git cherry-pick <sha1-of-commit>
```

## checkout a specific file from an earlier version

```
git checkout <sha1-of-commit> -- foo.R
```

# Safety nets

It is very hard to actually destroy data with Git.

You can almost always recover using the ref log.

But ... no one actually enjoys using the ref log.

Before doing something iffy, create a "safety net" branch.

This can make it easier to back out of bad decisions.

# Safety nets

If you have high confidence, create the safety net branch.
Then checkout master and have at it.
If things go poorly, reset master to the safety net state.

If you have low confidence, create the safety net branch.
Have at it.
If things go poorly, checkout master and carry on.

# The Repeated Amend

It is very hard to actually destroy data with Git.
Any commited state can be recovered.

Rock climbing analogy → commit often!

If you're embarrassed by the clutter and tiny steps, use git amend to slowly build up a "real" commit before you push it.

work, commit, work, amend, work, amend, work, amend, PUSH
work, commit, work, amend, work, amend, work, amend, PUSH

# The Repeated Amend

live: for a repo we created yesterday, locally, build up a commit with a few amends then push (prove the intermediate states do not show up)

Amend lets you update the message and/or the changes in the commit

```
git commit --amend -m "an updated commit message"
```

Amend is available in the usual RStudio commit interface, btw.

# Recovering from Git(Hub) failure

Scenario: You try to push and cannot

What's the problem?
There are changes on GitHub that you don't have.

Pull. If the gods smile upon you, merge works. Now push.

Let's create this situation.

Make sure local Git pane is clear.
Make sure local and remote are synced (push, pull).

Edit & commit to file A locally.
Edit & commit to file B remotely.

Try to push. You will fail.

```
jenny@2015-mbp bunny-scarf $ git push
To github.com:jennybc/bunny-scarf.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:jennybc/bunny-scarf.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Remedy? Do what it says!

pull, then push ... pull, then push ... pull, then push

Look at your Git history.

You will see a merge commit, where the local and remote changes were reconciled.

This is best case scenario and is likely with good Git habits (lots of small frequent commits and merges, no binary files in repo).

# Recovering from Git(Hub) failure

Scenario: You pull and get a merge conflict.

What's the problem?
GitHub can't figure out how to reconcile diffs.

Resolve the conflicts.
Or abort ... and come back later.

Let's create this situation.

Make sure local Git pane is clear.
Make sure local and remote are synced (push, pull).

Edit & commit to file A locally.
Make conflicting edit & commit to file A remotely.

Try to push. You will fail. Try to pull. You will fail. All is fail.

```
From github.com:jennybc/bunny-scarf
   958548f..3357952  master     -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.


<<<<<<< HEAD
Wingardium Leviosaaaaaaaaa

=======
Wing-GAR-dium Levi-O-sa
>>>>>>> 33579525d88af071268b0a0c64c54f357712589a
```

```
<<<<<<< HEAD
Wingardium Leviosaaaaaaaaa

=======
Wing-GAR-dium Levi-O-sa
>>>>>>> 33579525d88af071268b0a0c64c54f3577712589a
```

Git inserts markers at each locus of conflict and shows you both versions.

You must form a consensus version and delete the markers, at each locus. Commit. Push. Carry on.

```
<<<<<<< HEAD
Wingardium Leviosaaaaaaaaa

=======
Wing-GAR-dium Levi-O-sa
>>>>>>> 33579525d88af071268b0a0c64c54f3577712589a
```

If you're just not up for this right now, do
git merge --abort to back out.

You can keep working locally. But you must deal with this problem before you can resume syncing with GitHub.