Daily workflows with Git + GitHub + RStudio

What did we do previously?

Confirmed your setup 🎉

New/Existing repo, GitHub first
- Made several successful roundtrips
- Importance of viewing diffs and commits

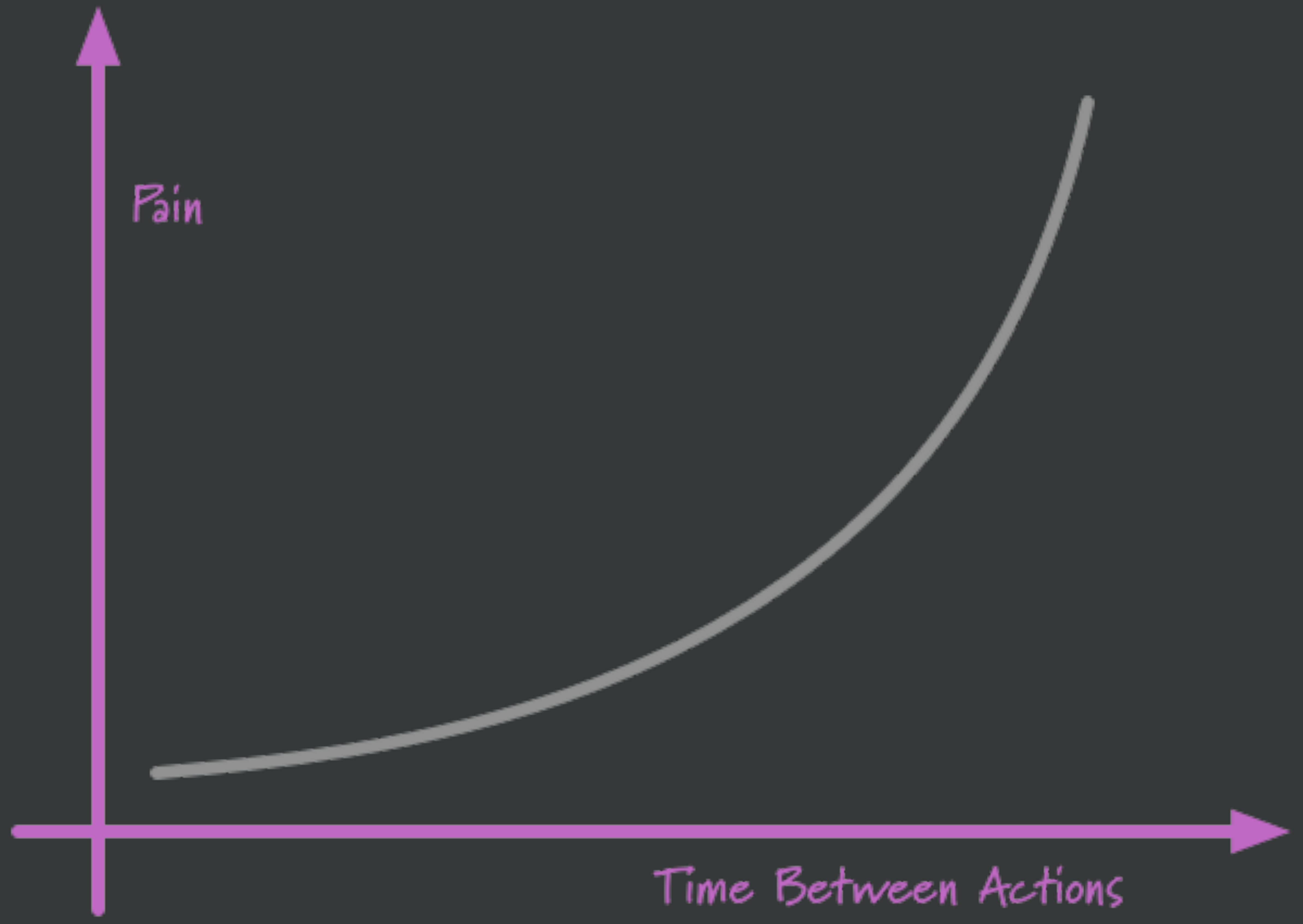
Special R + GitHub stuff:
- R → md is easy, high payoff

Got out of some uncomfortable situations

Today we'll preview some intermediate workflows you'll "enjoy" soon.

Deep Thoughts

"If it hurts, do it more often."

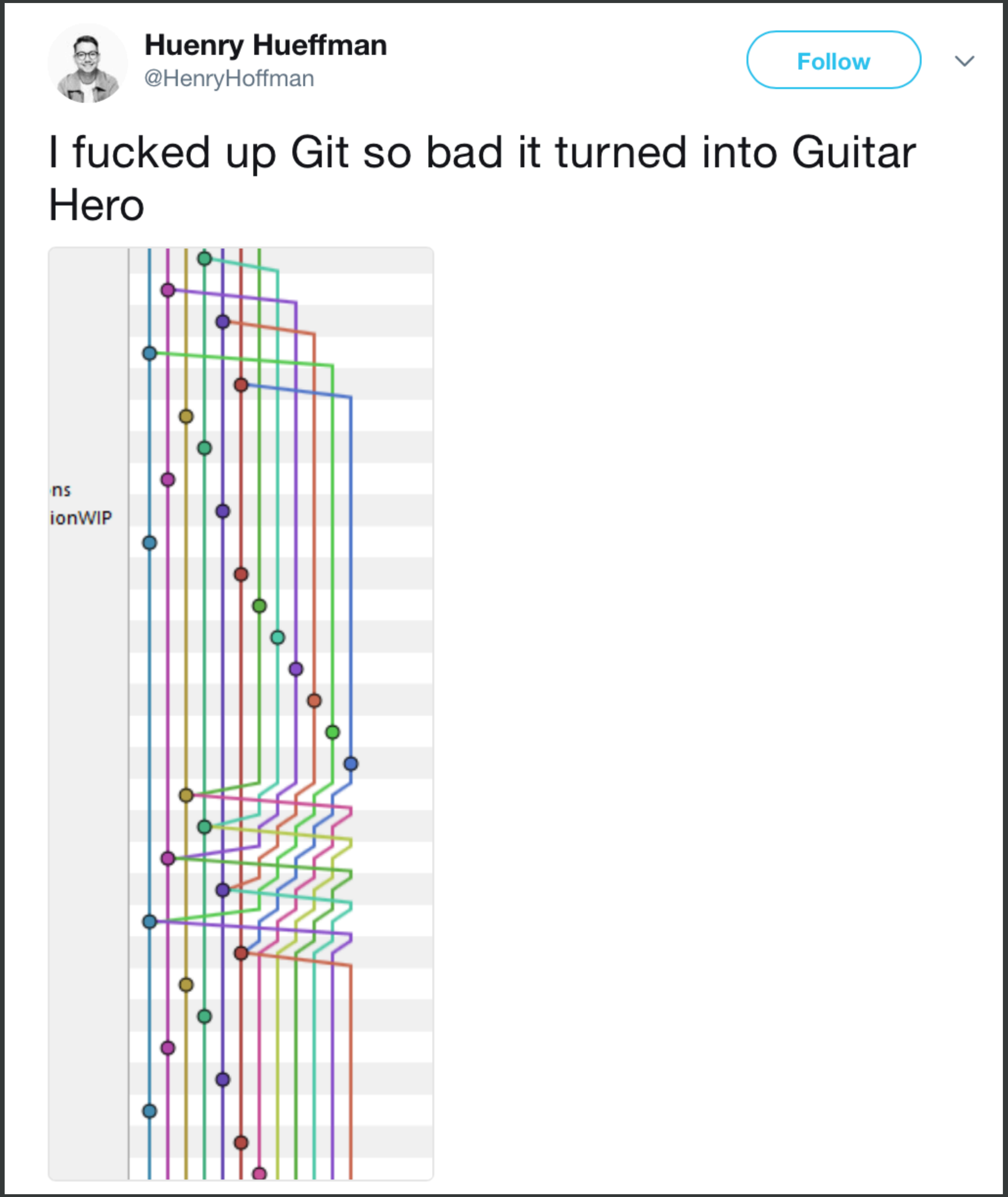


https://martinfowler.com/bliki/FrequencyReducesDifficulty.html

"If it hurts, do it more often."

Apply this to git commit, pull, merge, push.
(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.

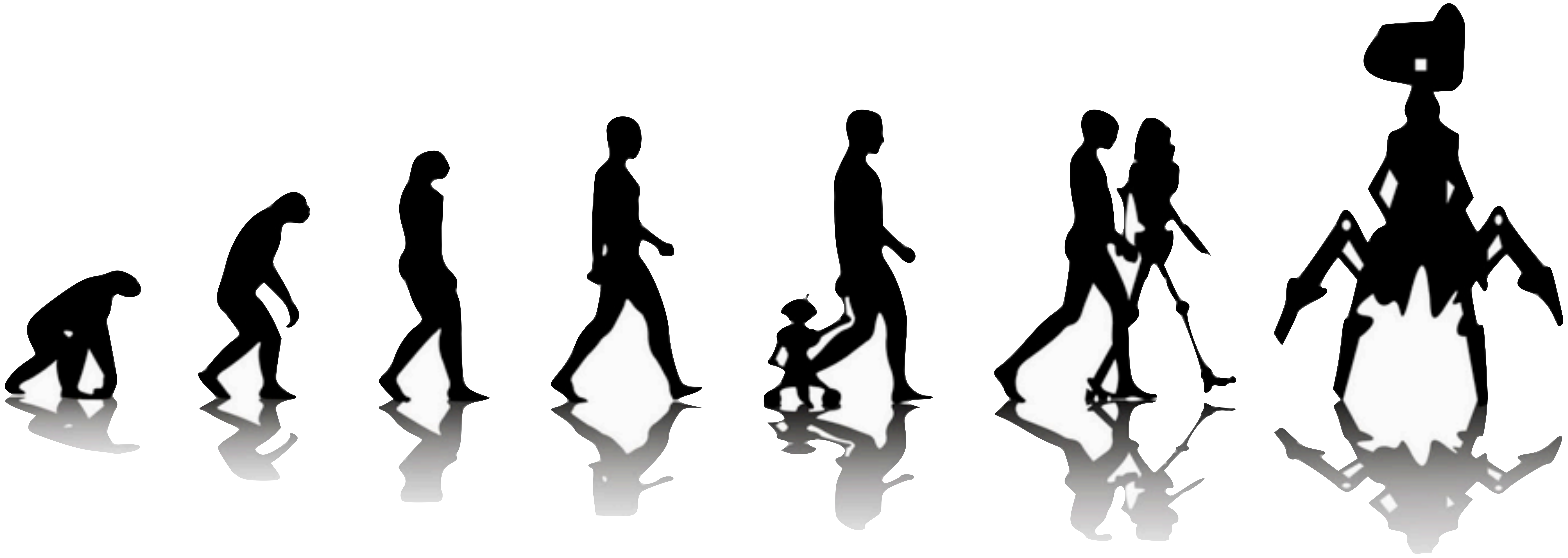

Huenry Hueffman
@HenryHoffman

I fucked up Git so bad it turned into Guitar Hero

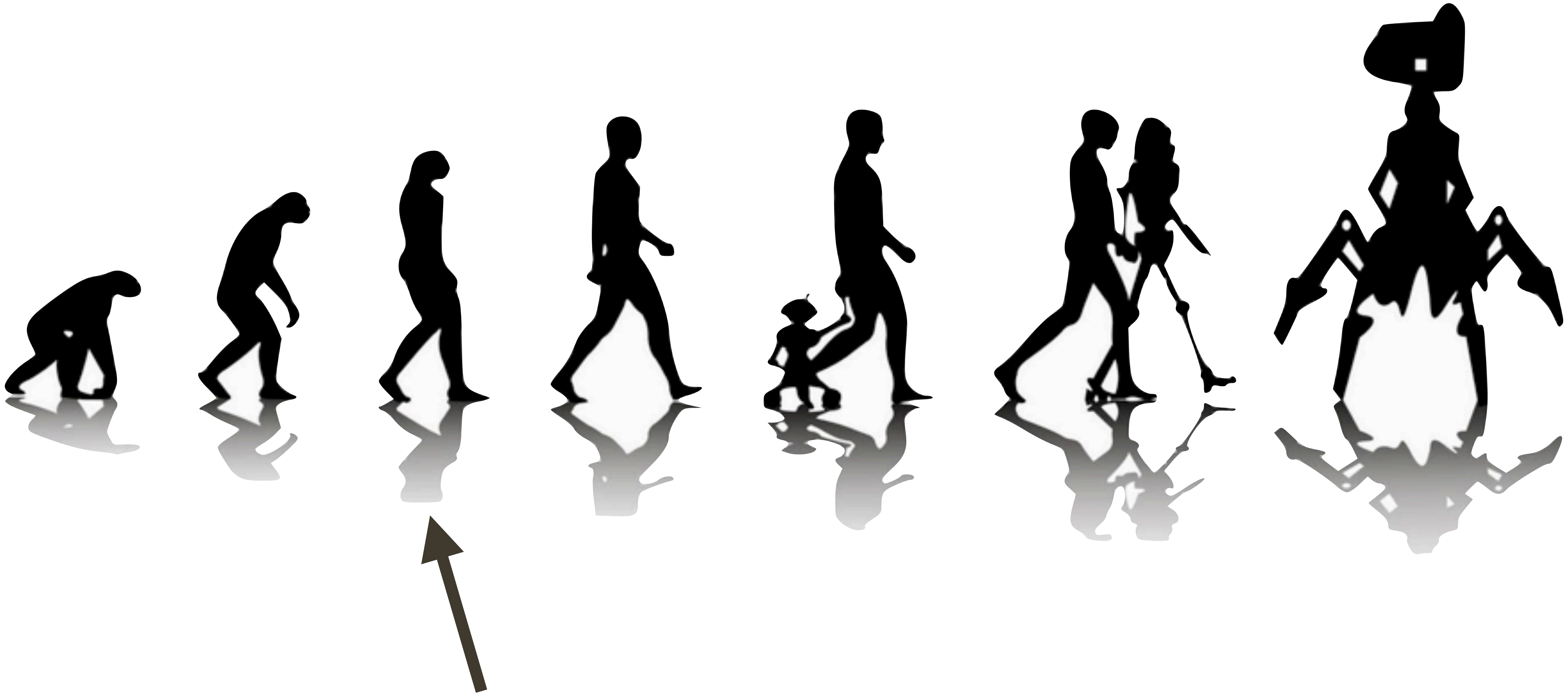
https://twitter.com/henryhoffman/status/694184106440200192

You do NOT want "Guitar Hero" Git history.

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



"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.

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* 🐲

"I just need to see the past."

https://github.com/rstats-wtf/wtf-ascii-funtimes

Go visit this in a web browser.

What's in this repo? What's in the files?

How many commits have been made?

Which commit introduced the bunny?

How many times has the truck been changed?

Which file(s) was/were most recently changed?

"I just need to see the past."

GitHub (or any modern git remote) is the easiest way to navigate project history.

"Why is this thing the way it is? How did we get here?"

This (+ burn it all down) make a remote repo extremely valuable, even for private solo work.

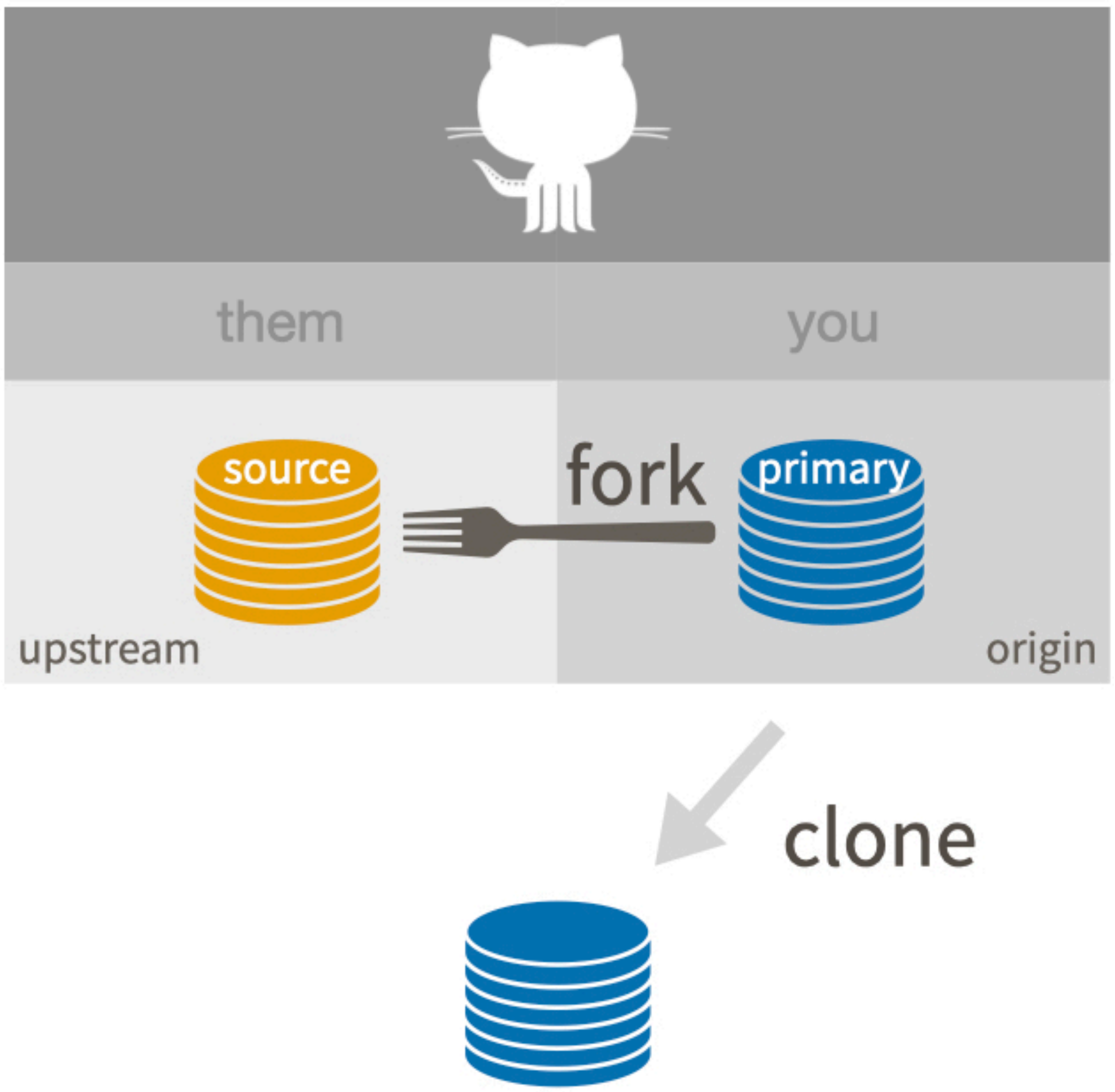
Levels of Git Time Travel

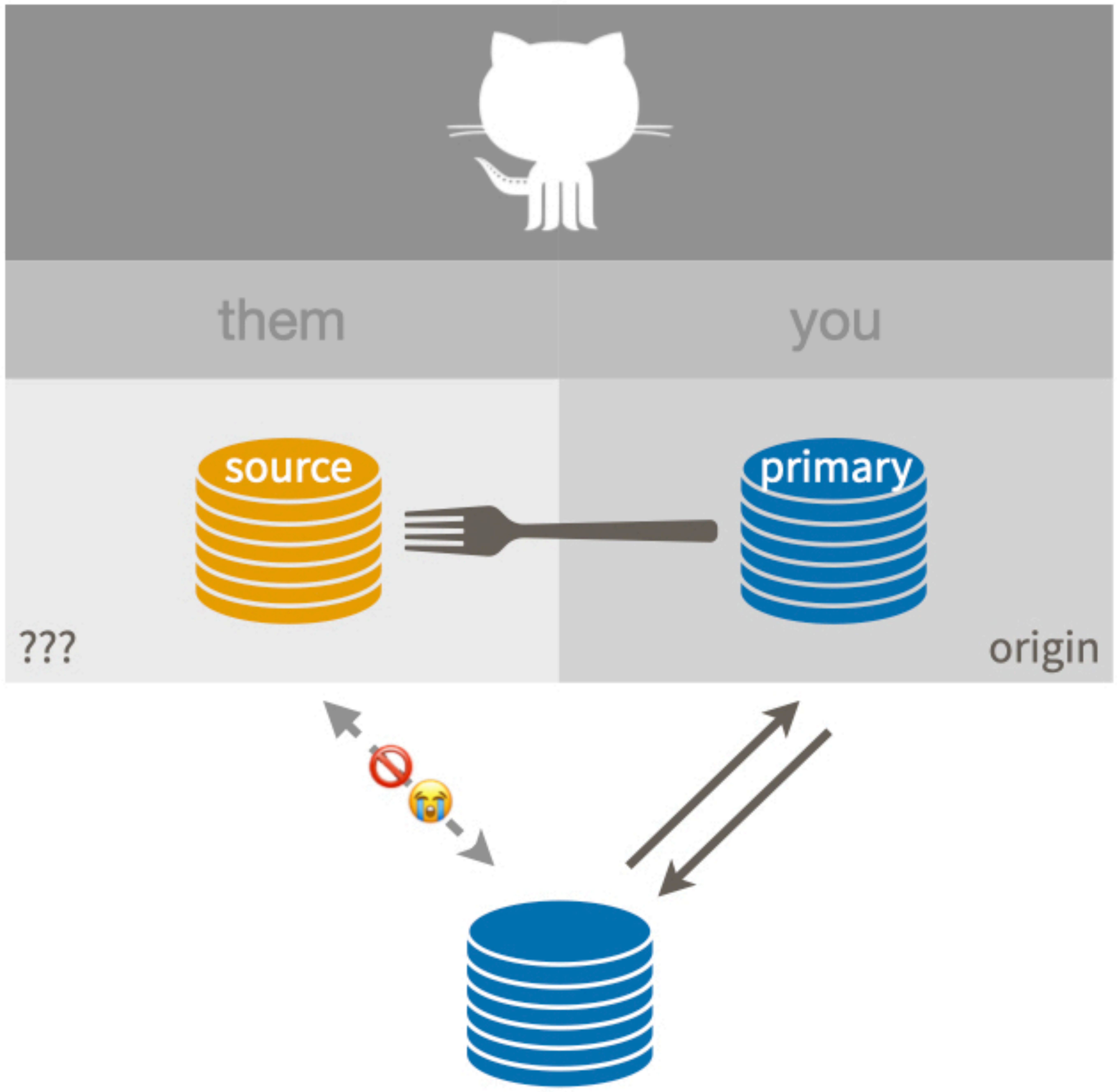
"I just need to see the past."	Browse & search on GitHub.
"I need to visit the past."	Create and checkout a branch.
"I want to return to the past."	Revert or reset.
"I had a great cookie last October."	Cherry pick or checkout a path.
"I want to change the past."	🐲 *there be dragons* 🐲

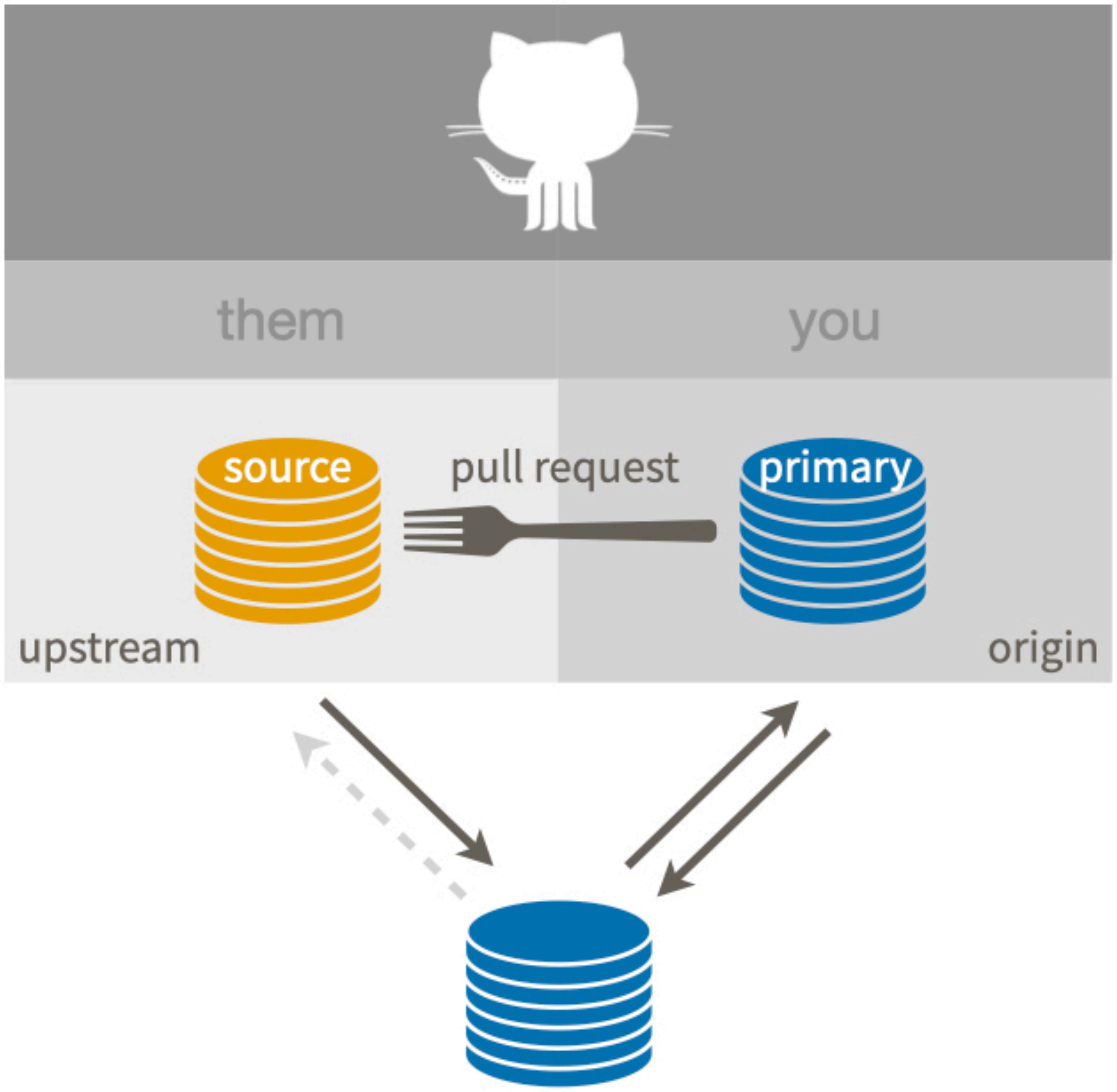
get your own copy of
wtf-ascii-funtimes!

fork and clone

Why do you have to care about remotes, eventually?


them
you
source
fork
primary
upstream
origin
clone


them
you
source
primary
???
origin


them
you
source
pull request
primary
upstream
origin

```
create_from_github(
  "https://github.com/rstats-wtf/wtf-ascii-funtimes",
  destdir = "???"
)
```

In Happy Git:

Fork and clone

https://happygitwithr.com/fork-and-clone.html

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* 🐲

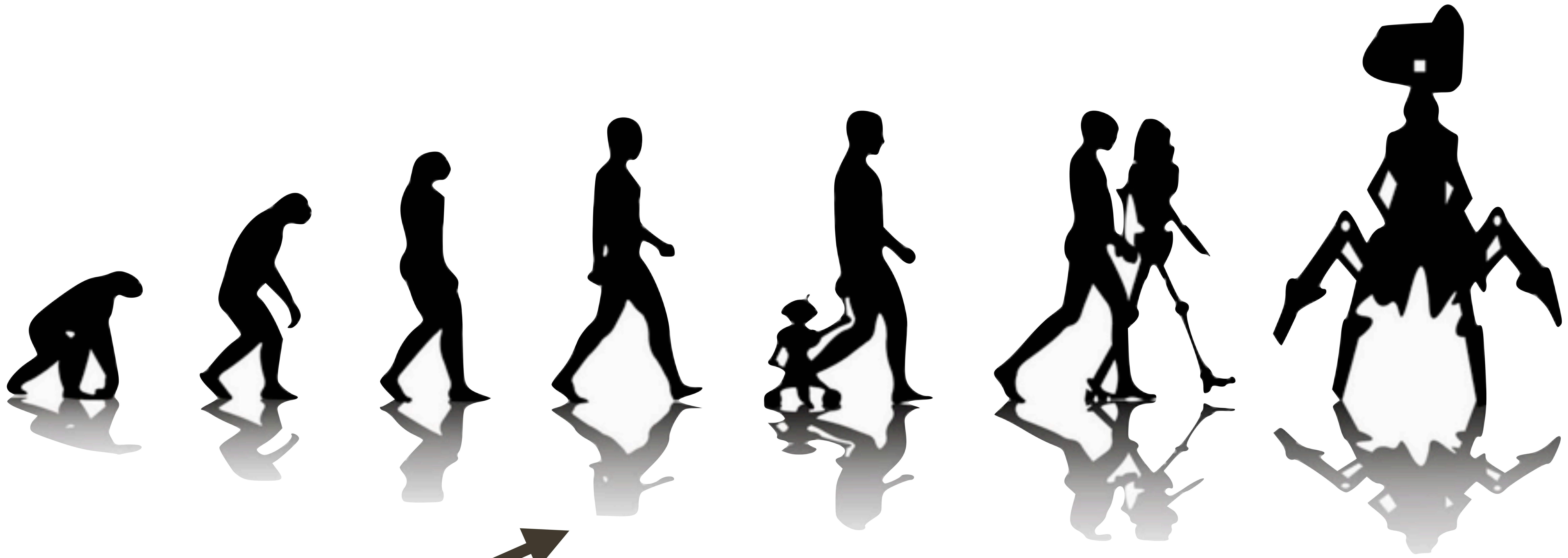
Why must we talk about SHAs (commits) and branches?

Because that is how we talk, precisely, about "time", with Git.

It's how we address a specific diff or state.

In Happy Git:

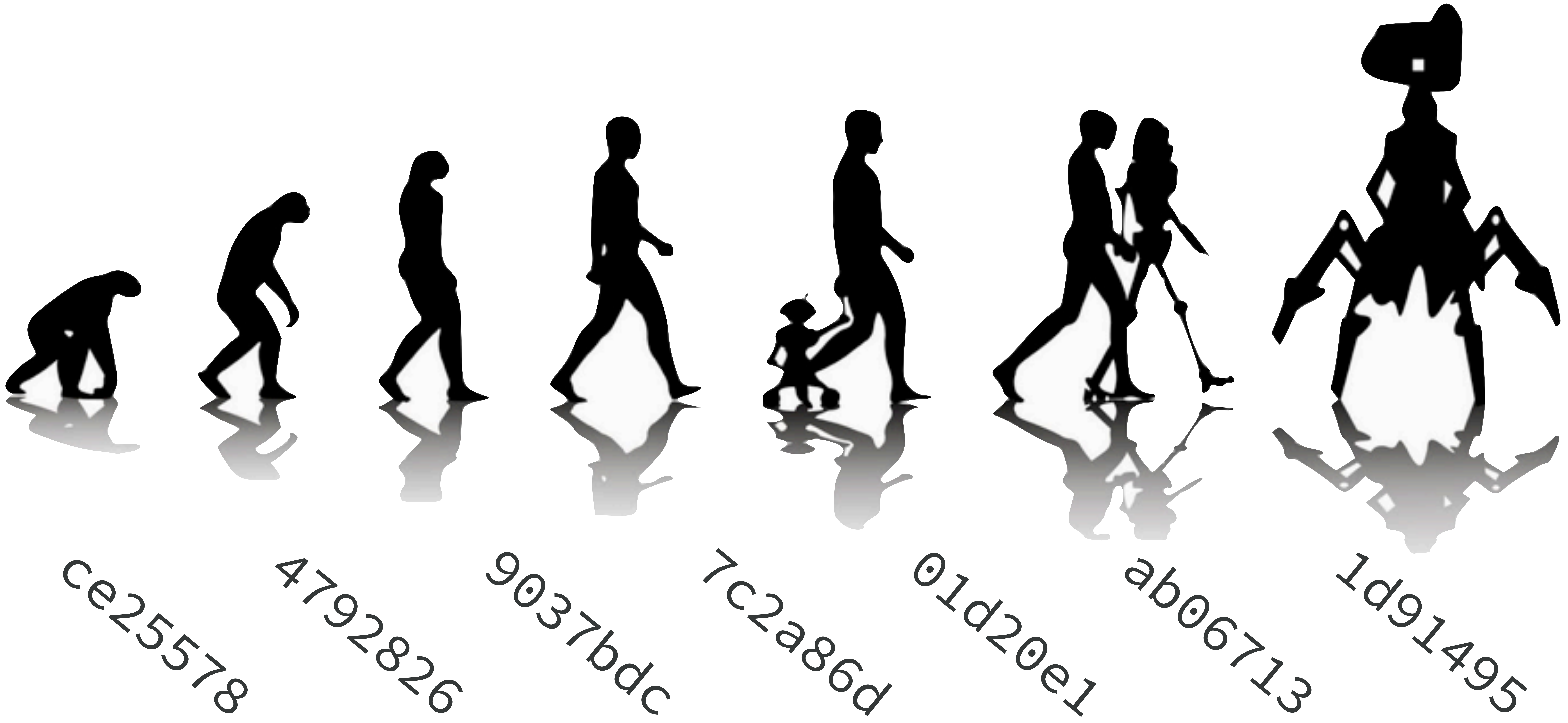
https://happygitwithr.com/git-refs.html



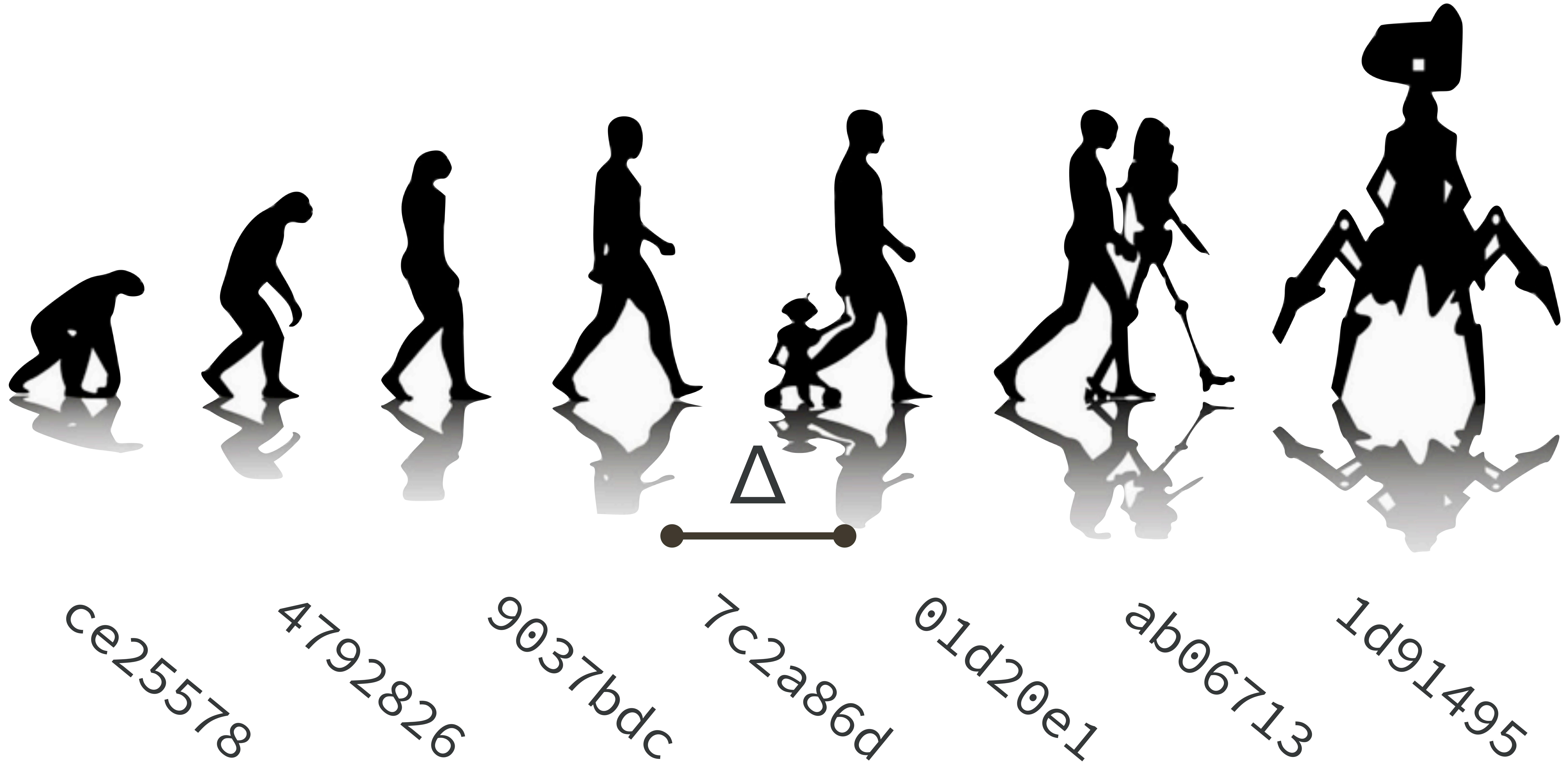
"commit"

a file or project state that is **meaningful to you**

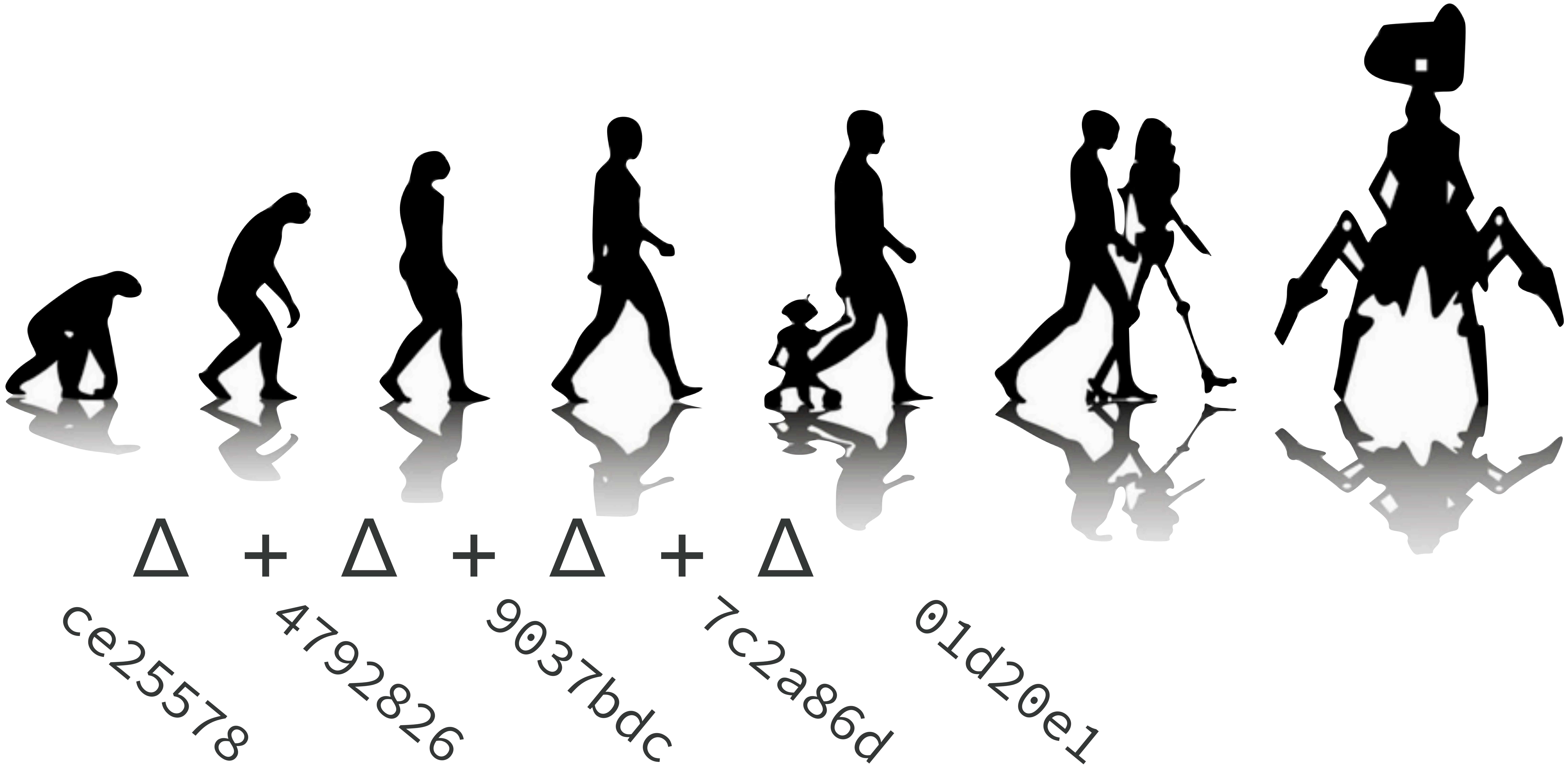
for inspection, comparison, restoration



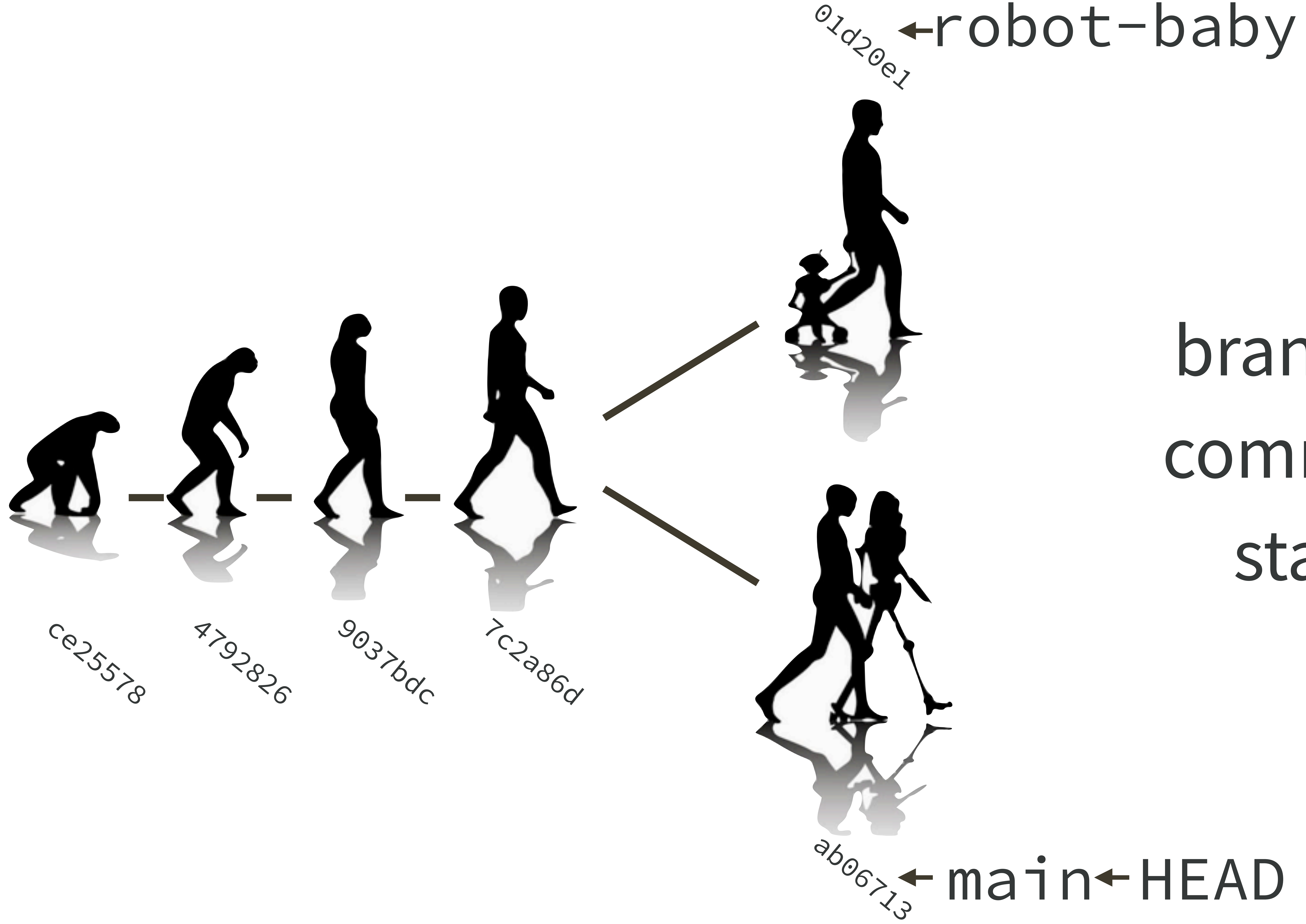
commit ≈ SHA-1 hash ≈ SHA ≈ 40 chars ≈ 1st 7 chars



sometimes, a commit ≈ a diff(erence)

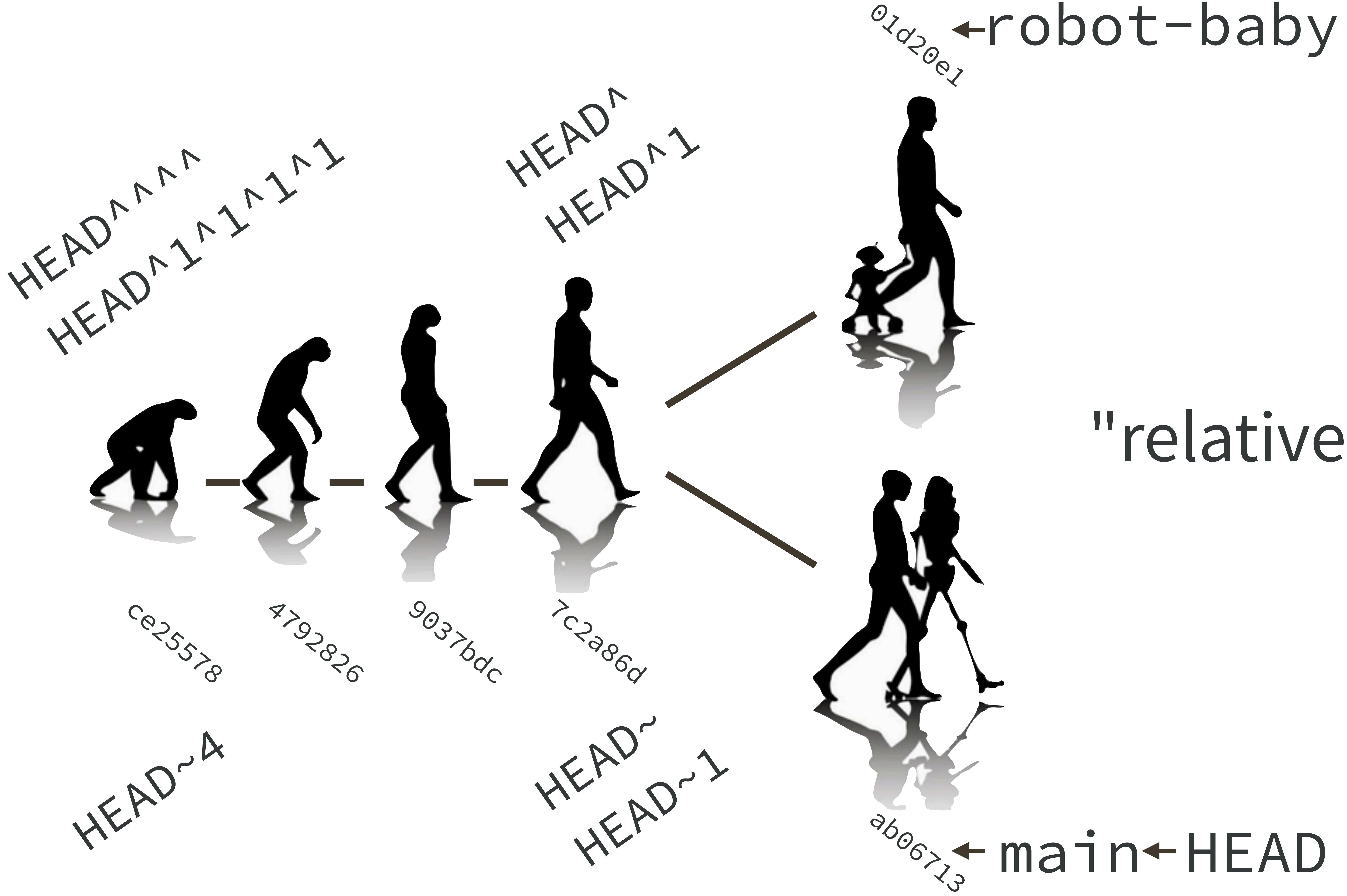


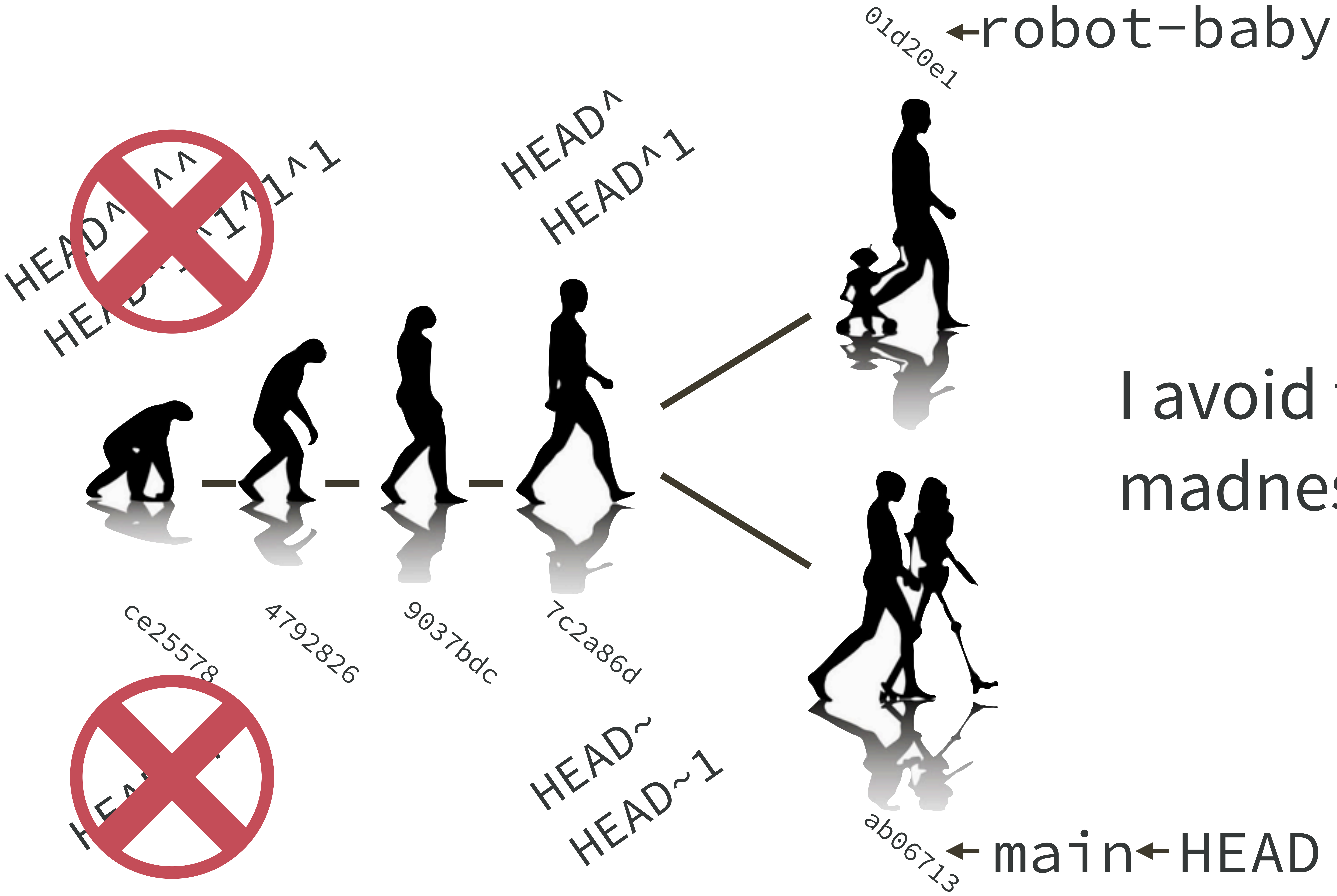
sometimes, a commit ≈ a state



branch ≈
commit ≈
state

"relative" refs





I avoid this madness

"I need to visit the past."

Create and checkout a branch at a specific commit ≈ state.

Then return to present ≈ checkout main.

create & checkout
a branch

at a specific state

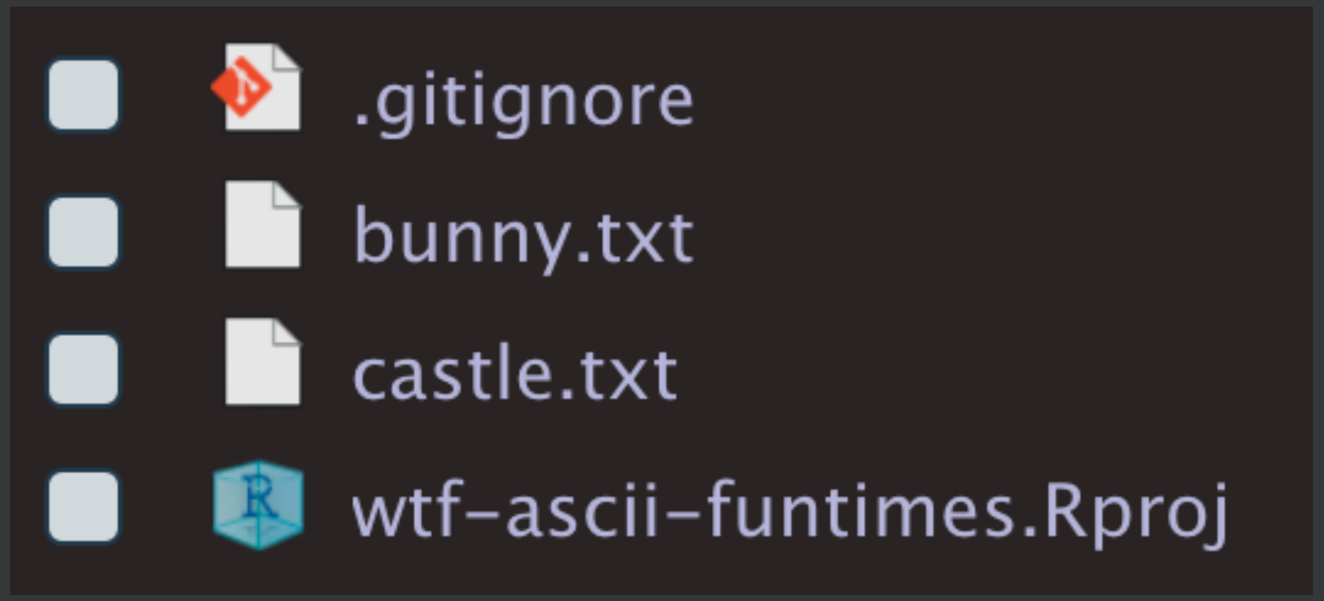
```
git checkout -b branchname <sha1-of-commit or HEAD~3>
```

```
git checkout main
```

return to present

Reset your local files to this state:

- Castle exists
- Bunny exists
- Truck does NOT yet exist

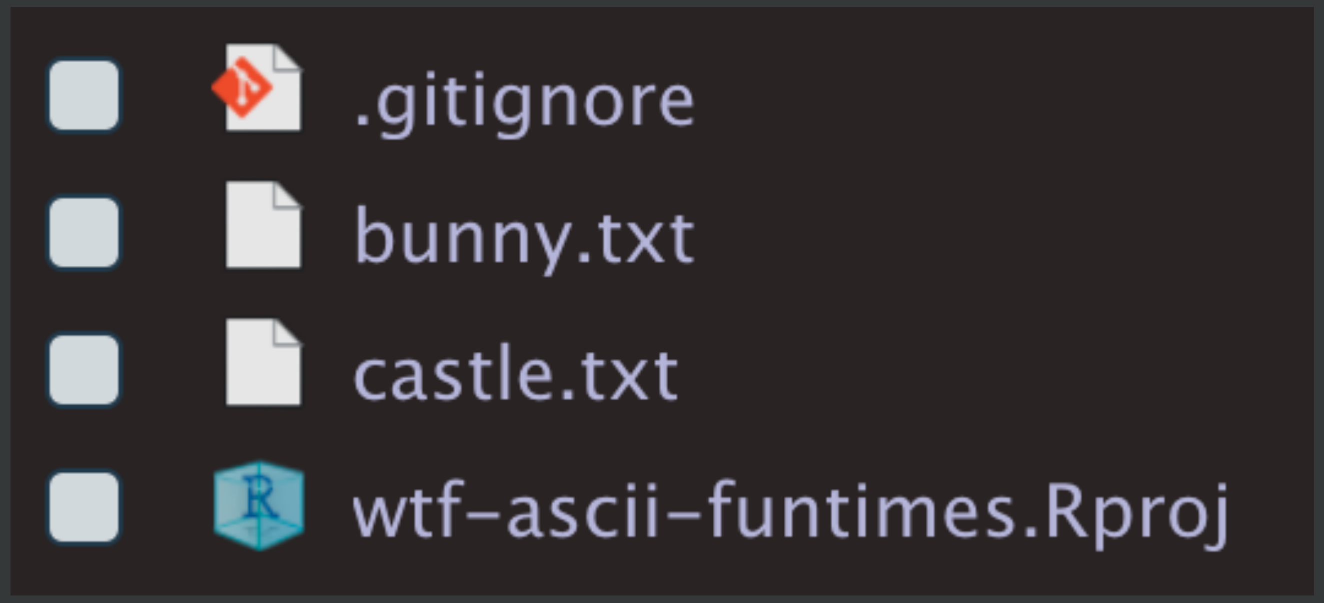


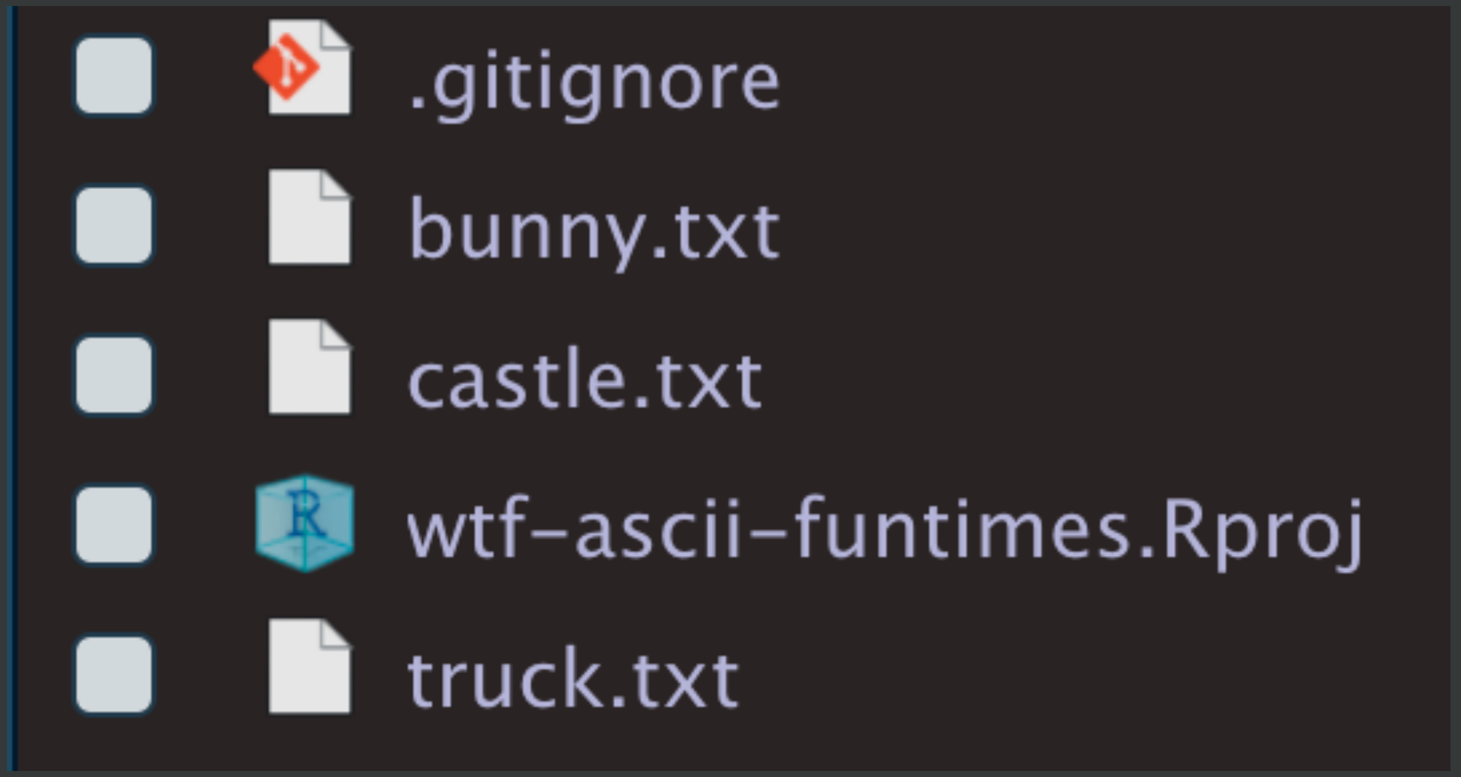
.gitignore
bunny.txt
castle.txt
wtf-ascii-funtimes.Rproj

```
git checkout -b time-travel ???
```

Reset your local files to the present

```
git checkout main
```


.gitignore
bunny.txt
castle.txt
wtf-ascii-funtimes.Rproj


.gitignore
bunny.txt
castle.txt
wtf-ascii-funtimes.Rproj
truck.txt

* or use RStudio or GitKraken to switch back to main

What about all these old branches lying around?

```
git branch -d localBranchName
git push origin --delete remoteBranchName
git prune
```

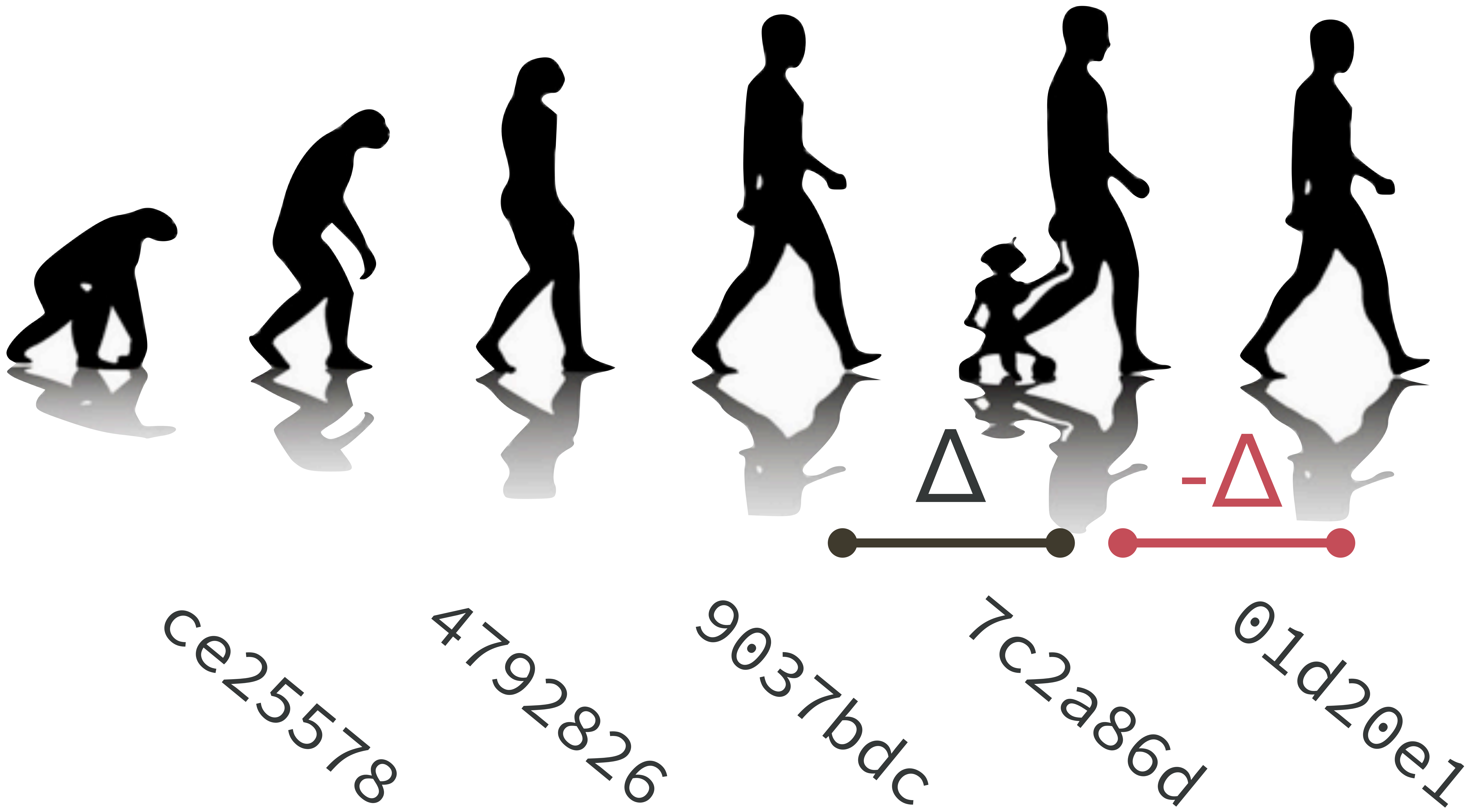
* I usually do this via GitKraken or GitHub

"I want to return to the past."

Revert = make a new commit that undoes a commit.

Reverses a specific change.

Do this to undo something that has been pushed.



```
git revert --no-edit 7c2a86d
```

Revert the commit where the bunny population went from 1 to 6.

accept the automatic commit message

```
git revert --no-edit ???
```

```
  \\
  __()
o(_-\_
```

* or use GitKraken to revert

Push the bunny birth control work to GitHub.

```
git push
```

Check in the browser to confirm you're synced up.

"I want to return to the past."

Reset returns repo to a previous state.

Safe only for work that has not been pushed.

Add a left-facing bunny.

Do a terrible job. Feel deep regret.

```
  \\     ///
  __() o(_-\_
o(_-\_ o(_-\_
```

Dismiss current uncommitted changes

```
git reset --hard HEAD
```

* or use "Discard All" in RStudio or "Discard file" in GitKraken

Add a left-facing bunny AGAIN.

Do a terrible job AGAIN.

Commit your awful bunny. DO NOT PUSH.

```
  \\     ///
  __() o(_-\_
o(_-\_ o(_-\_
```

Un-commit last commit, but keep the changes

```
git reset --mixed HEAD~
```

Un-commit last commit and discard the changes

```
git reset --hard HEAD~
```

* Or, frankly, I usually do this in GitKraken.

"I had a great cookie last October."

Bring a very specific thing from the past to the present:

- A whole commit = "cherry pick"
- The state of a specific file = "checkout (a specific filepath)"

Re-apply the commit where the bunny population went from 1 to 6.

```
git cherry-pick ???
```

```
  \\     \\     \\     \\     \\     \\
  __()   __()   __()   __()   __()   __()
o(_-\_ o(_-\_ o(_-\_ o(_-\_ o(_-\_ o(_-\_
```

* Or, frankly, I usually do this in GitKraken.

Re-store the short castle tower, by checking out castle.txt from a suitable state.

```
git checkout ??? -- castle.txt
```

```
  _   |~  _
[_]--'--[_]
|'|""`""|'|
| | /^\ | |
|_|_|I|_|_|
```

Commit this change.

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* 🐲

Push this your work to GitHub.

```
git push
```

Check in the browser to confirm you're synced up.

Branches as 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.

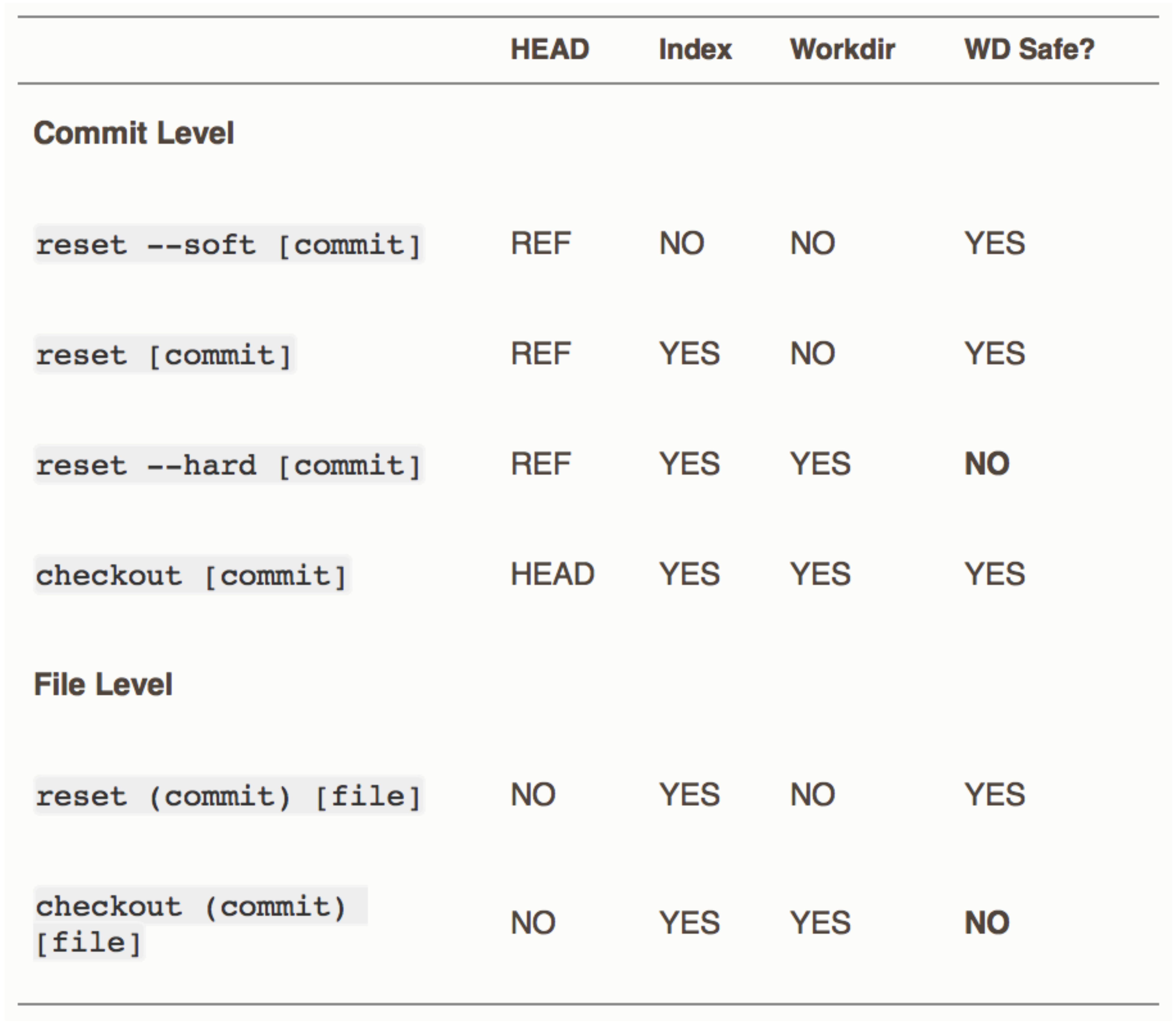
This can make it easier to back out of bad decisions.

Branches as 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.

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**

No more time travel to past

Two important techniques for moving forward:

1. Repeated amend
2. Merge (w/o and w/ conflicts)

The Repeated Amend

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

Rock climbing analogy → commit often!

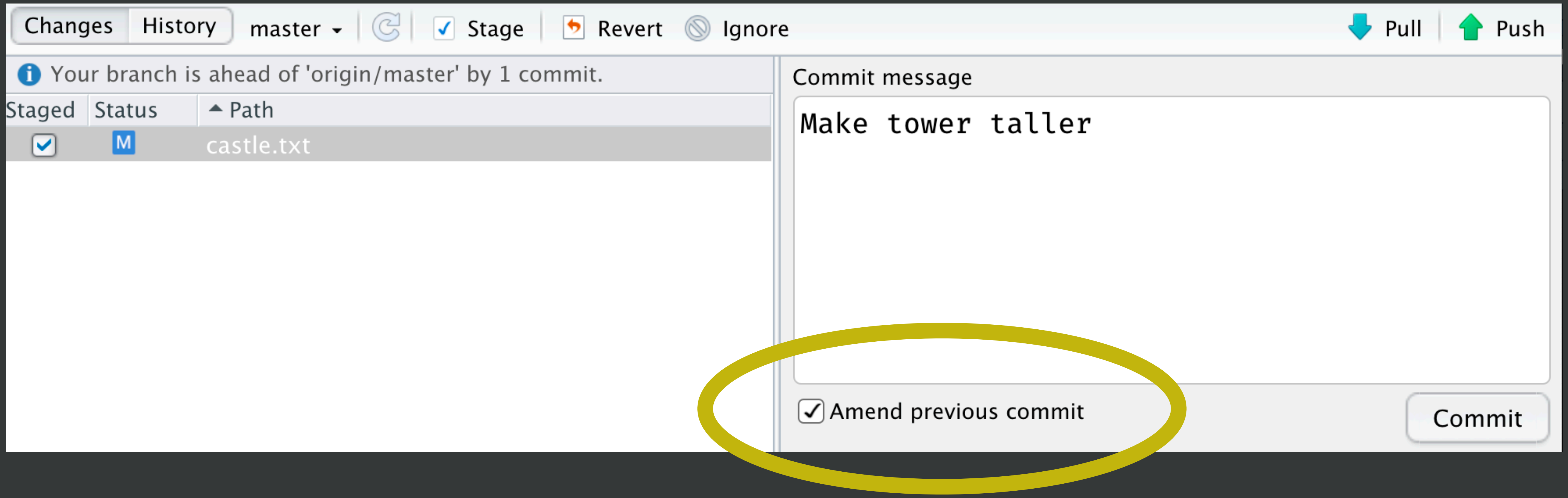
If you're embarrassed by the clutter and tiny steps, use git amend to slowly build up a "real" commit before you push it.

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

Make the tower tall again.
Add one layer at a time, using amend.

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

git commit --amend --no-edit
```



Your branch is ahead of 'origin/master' by 1 commit.

Staged	Status	Path
✓	M	castle.txt

Commit message

```
Make tower taller
```

[✓] Amend previous commit

* Or, frankly, I usually do this in RStudio or GitKraken.

Push this your work to GitHub.

```
git push
```

Check in the browser to confirm you're synced up.

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.

Edit truck.txt **on GitHub**.

Commit.

Edit castle.txt **locally**.

Commit.

Try to push. `git push`. NOPE.

```
>>> git push upstream HEAD:refs/heads/master
To github.com:rstats-wtf/wtf-ascii-funtimes.git
 ! [rejected]        HEAD -> master (non-fast-forward)
error: failed to push some refs to 'git@github.com:rstats-wtf/wtf-ascii-funtimes.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Do what Git says!

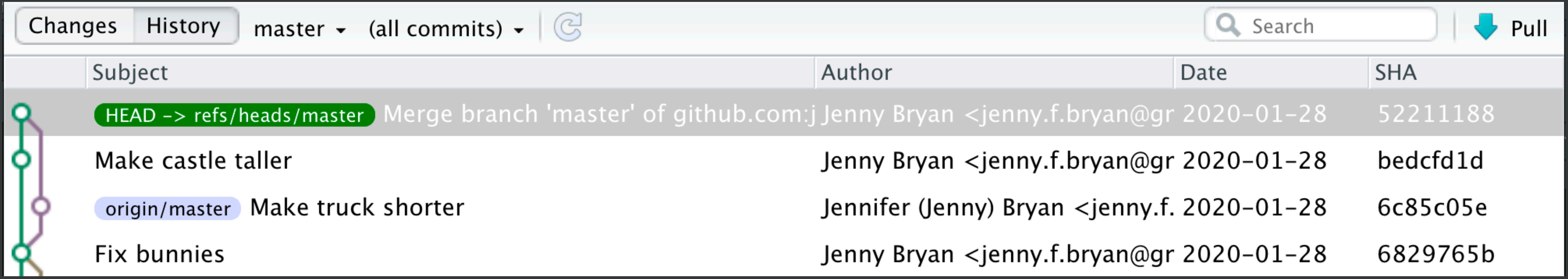
```
git pull
```

```
>>> git pull
From github.com:jennybc/wtf-ascii-funtimes
   9f4f288..42a6c97  master     -> origin/master
Merge made by the 'recursive' strategy.
 truck.txt | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
```

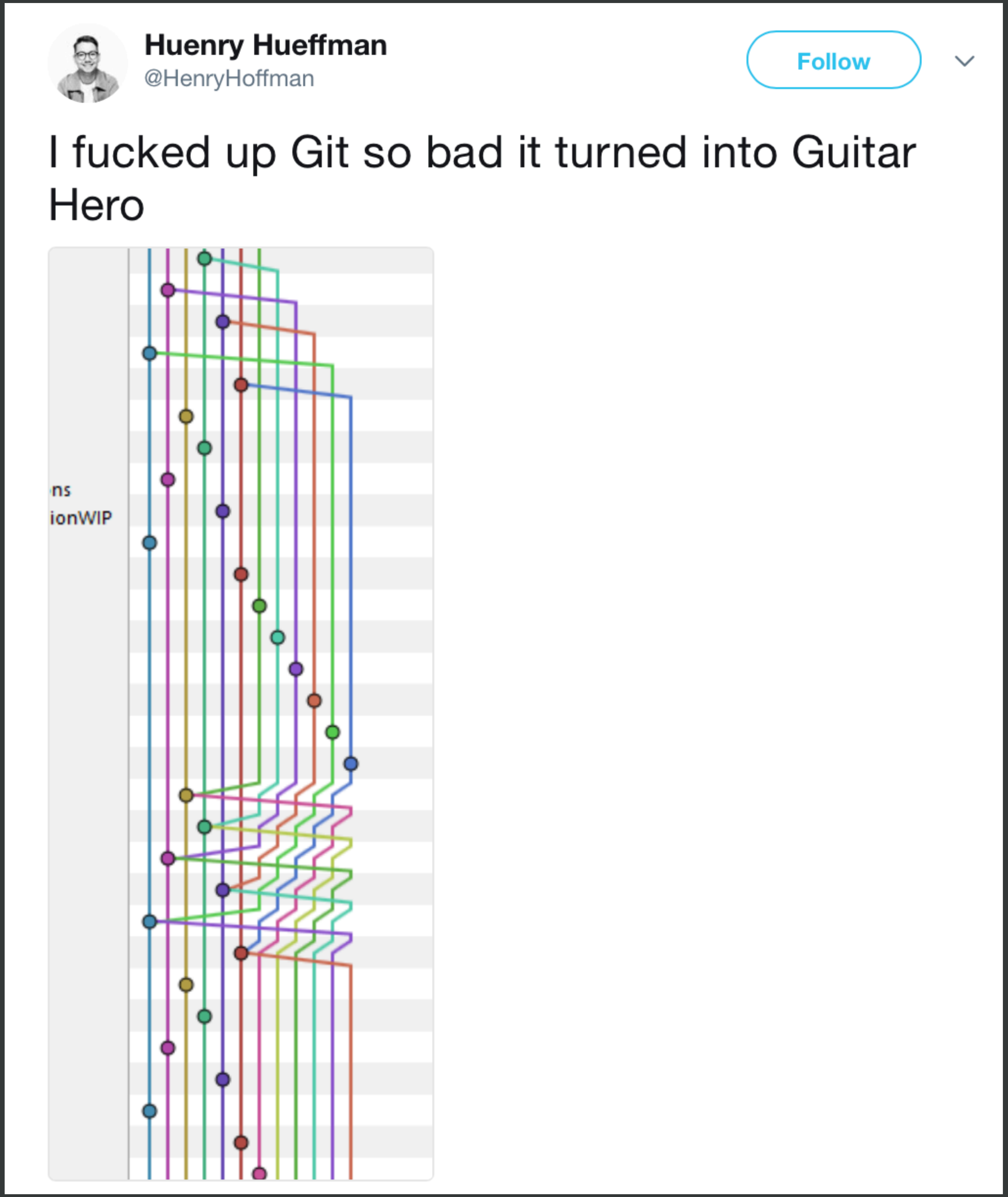
Look at your Git history.

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).



Subject	Author	Date	SHA
HEAD -> refs/heads/master Merge branch 'master' of github.com:j	Jenny Bryan <jenny.f.bryan@gr	2020-01-28	52211188
Make castle taller	Jenny Bryan <jenny.f.bryan@gr	2020-01-28	bedcfd1d
origin/master Make truck shorter	Jennifer (Jenny) Bryan <jenny.f.	2020-01-28	6c85c05e
Fix bunnies	Jenny Bryan <jenny.f.bryan@gr	2020-01-28	6829765b



Huenry Hueffman
@HenryHoffman

Follow

I fucked up Git so bad it turned into Guitar Hero

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.

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.

Push this your work to GitHub.

```
git push
```

Check in the browser to confirm you're synced up.

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.

Edit bunny.txt **on GitHub**.
Commit.

Edit bunny.txt **locally**.
Commit.

Make your edits contradictory.
Try to push. `git push`. NOPE.

```
>>> git push origin HEAD:refs/heads/master
To github.com:jennybc/wtf-ascii-funtimes.git
 ! [rejected]        HEAD -> master (fetch first)
error: failed to push some refs to 'git@github.com:jennybc/wtf-ascii-funtimes.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.
```

Do what Git says!

```
git pull
```

```
>>> git pull
From github.com:jennybc/wtf-ascii-funtimes
   1c4c06c..0d8f1e4  master     -> origin/master
Auto-merging bunny.txt
CONFLICT (content): Merge conflict in bunny.txt
Automatic merge failed; fix conflicts and then commit the result.
```

conflict markers

```
<<<<<<< HEAD
  \\     \\     \\
  __()   __()   __()
o(_-\_ o(_-\_ o(_-\_
=======
  \\     \\     \\     \\     \\     \\
  __()   __()   __()   __()   __()   __()
o(_-\_ o(_-\_ o(_-\_ o(_-\_ o(_-\_ o(_-\_

  \\     \\     \\     \\     \\     \\
  __()   __()   __()   __()   __()   __()
o(_-\_ o(_-\_ o(_-\_ o(_-\_ o(_-\_ o(_-\_
>>>>>>> 0d8f1e41d32342b2526a21f8a7c607bf32278efe
```

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.

When you're ready, git pull again and expect conflicts.

You must form a consensus version and delete the markers, at each locus.

Stage. Commit. Push. Carry on.

That is how we resolve merge conflicts!

* I resolve merge conflicts in GitKraken.

Bonus exercise:

Make non-overlapping, mergeable edits to the castle.

Flip flag direction

vs

Making door taller

This CAN auto-merge, even though affects the same file.

Deep Thoughts

Recovering from Git(Hub) failure

Scenario: You have a huge mess you cannot fix.

Official answer: git reset.

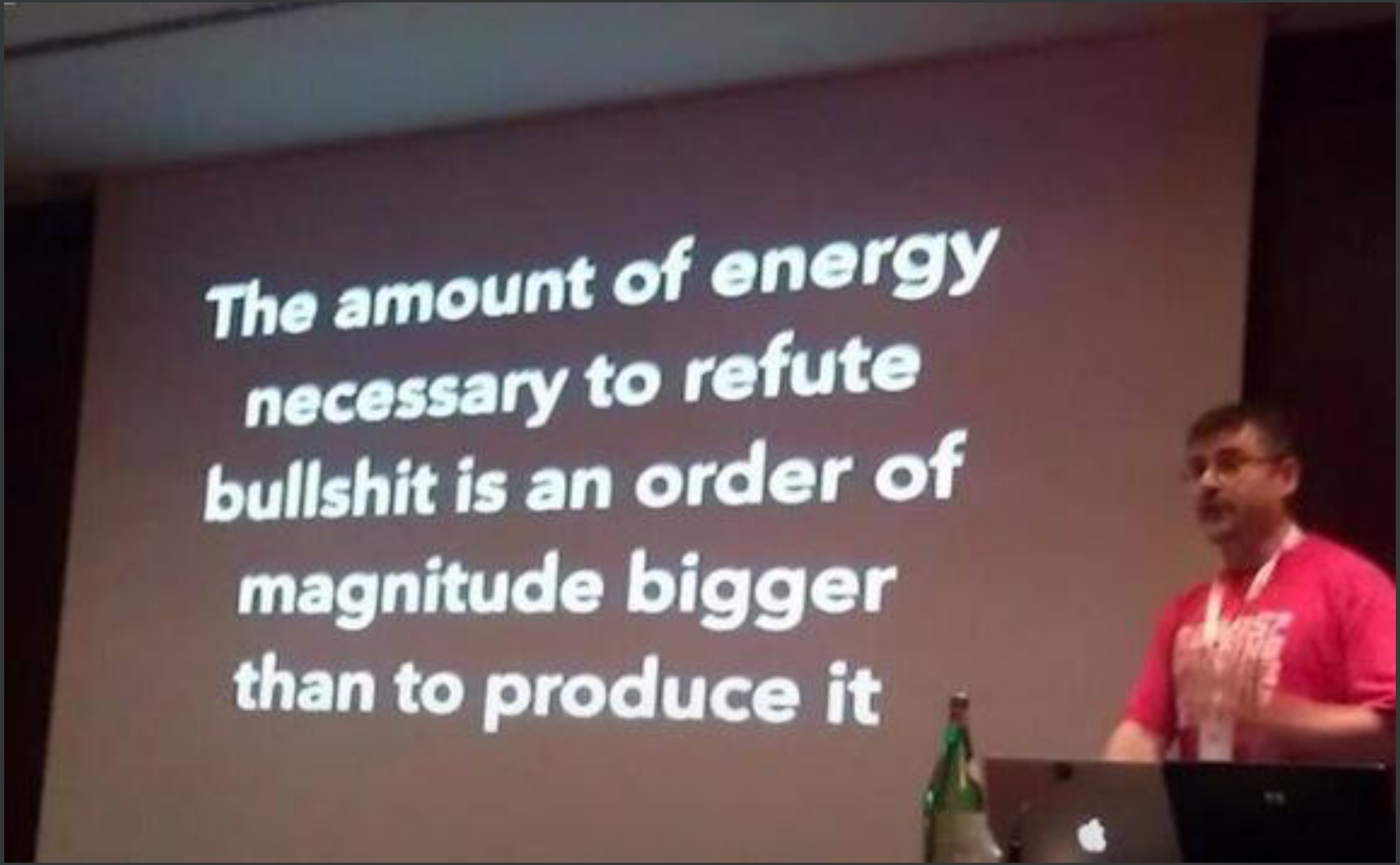
Unofficial answer: burn it all down 🔥

So I Jim Hester will still be my friend:

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

GitKraken, 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.



The amount of energy necessary to refute bullshit is an order of magnitude bigger than to produce it

- 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.