

# Daily workflows with Git + GitHub + RStudio



**rstudio::conf**

SAN FRANCISCO // JANUARY 27 - 30, 2020

from  RStudio

This work is licensed under a **Creative Commons Attribution-ShareAlike 4.0 International License**.

To view a copy of this license, visit  
<http://creativecommons.org/licenses/by-sa/4.0/>

# What did we do previously?

Confirmed your setup 🎉

## New repo, GitHub first, then RStudio

- Made several successful roundtrips
- Importance of viewing diffs and commits

## Special R + GitHub stuff:

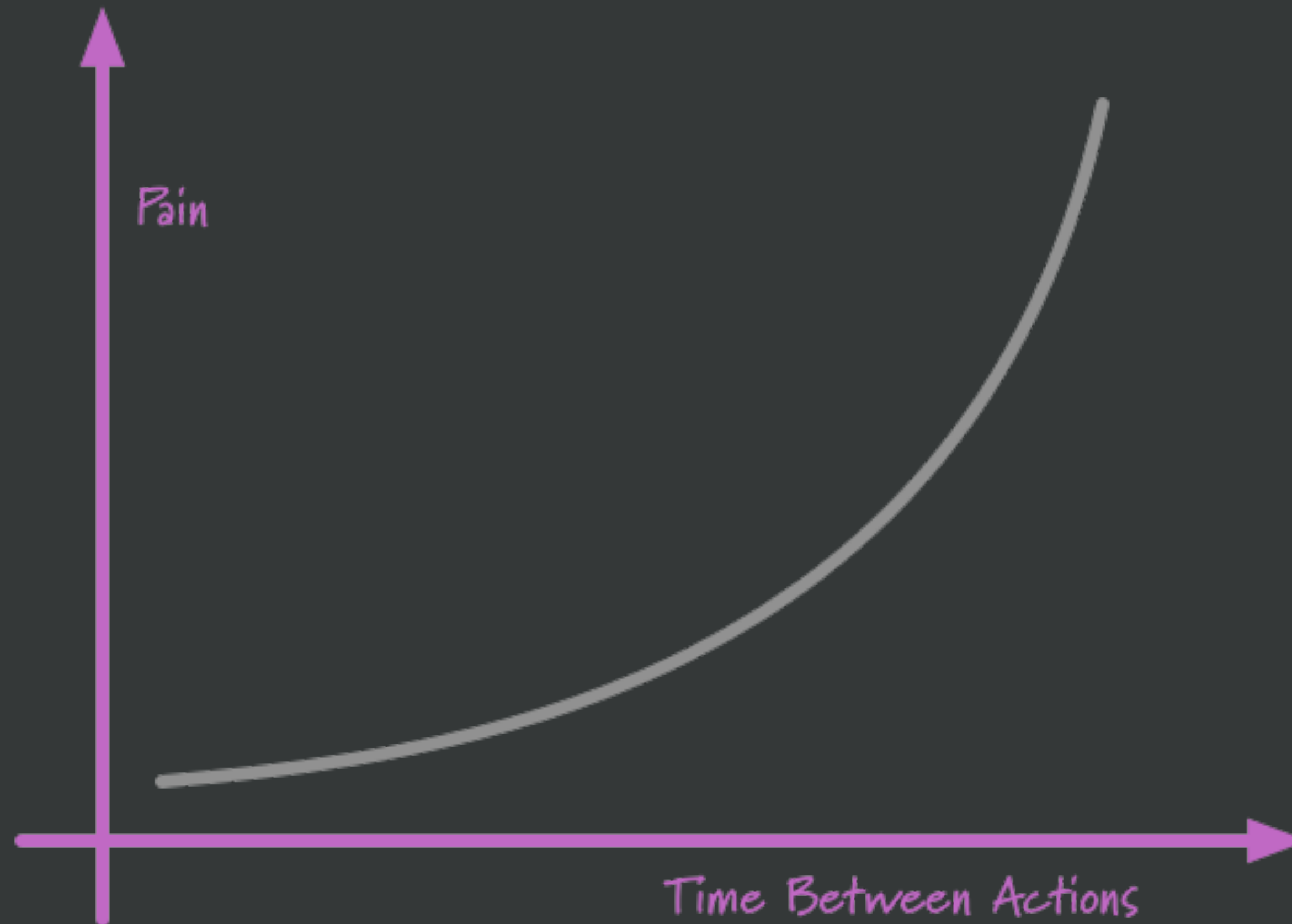
- R or Rmd → md is easy, high payoff
- (Did not see but check it out: GitHub Pages can use any md to make world's easiest website)

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.



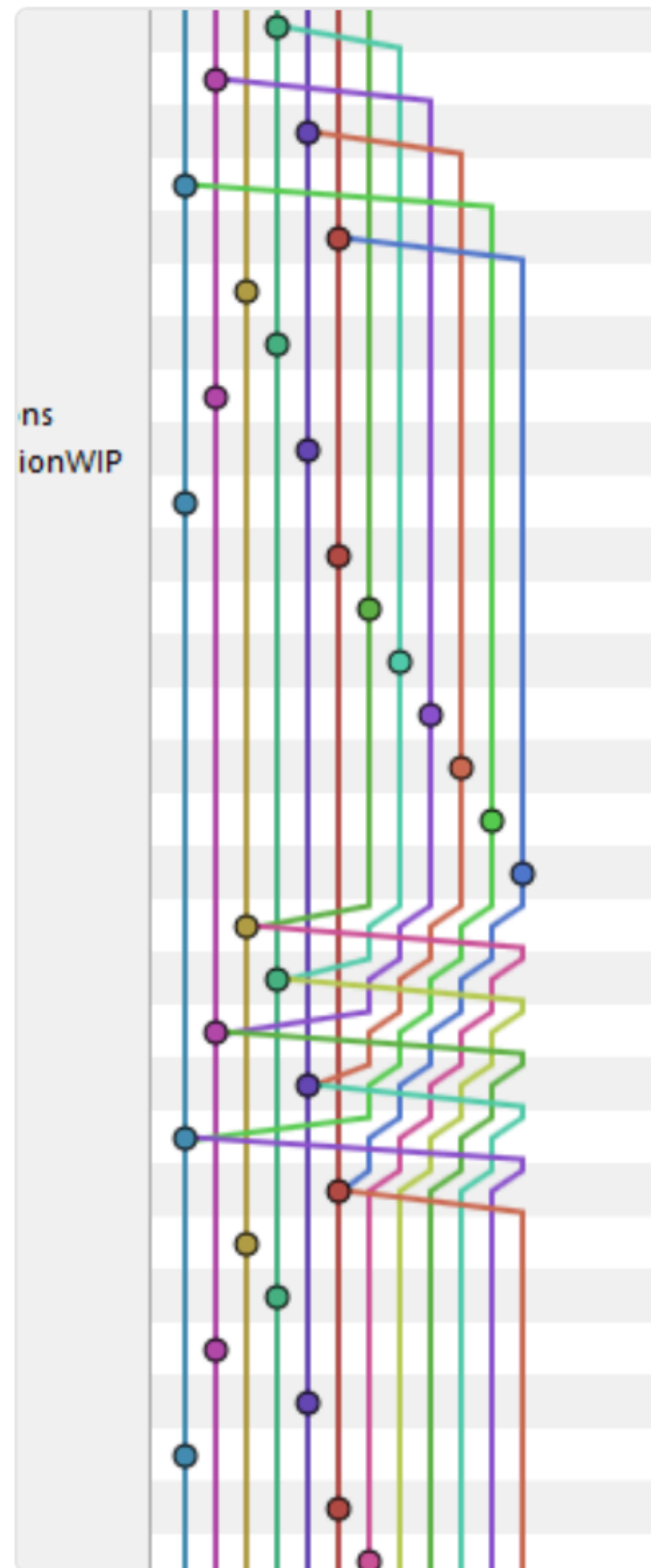
Huenry Hueffman

@HenryHoffman

Follow



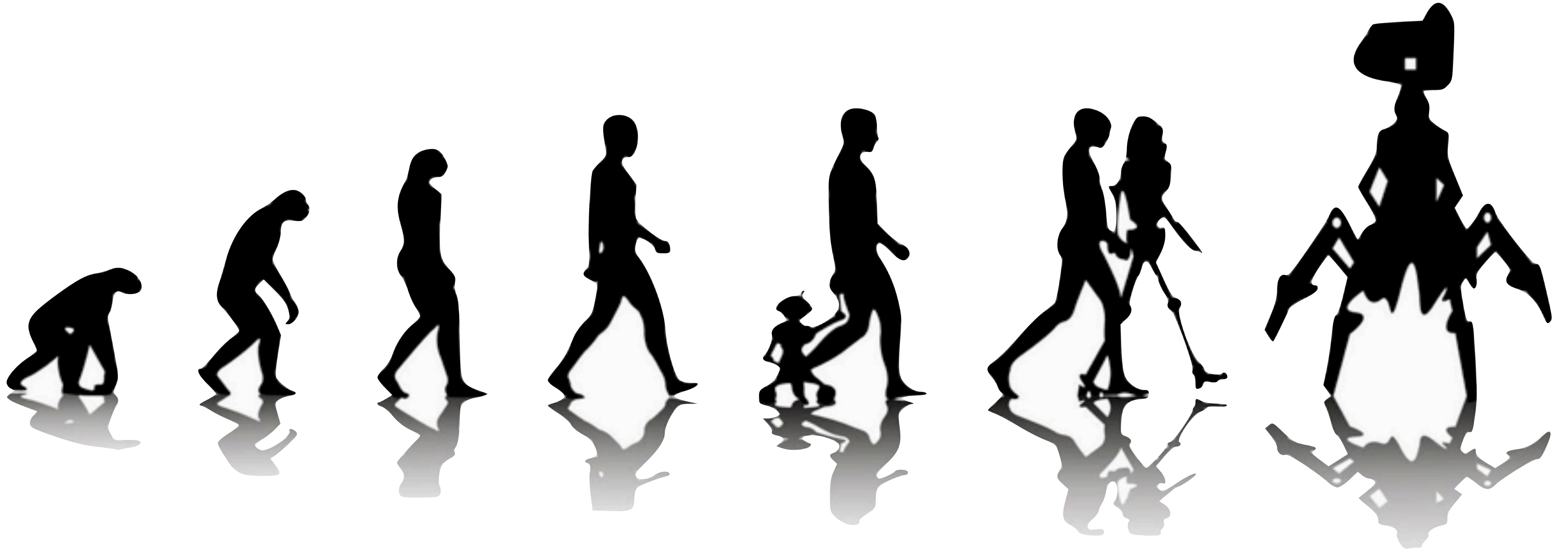
I fucked up Git so bad it turned into Guitar Hero



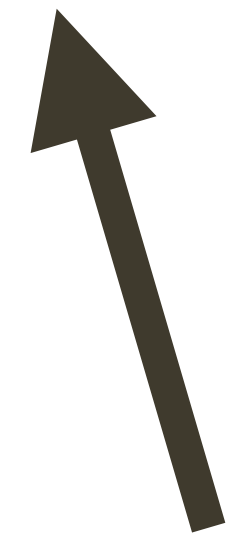
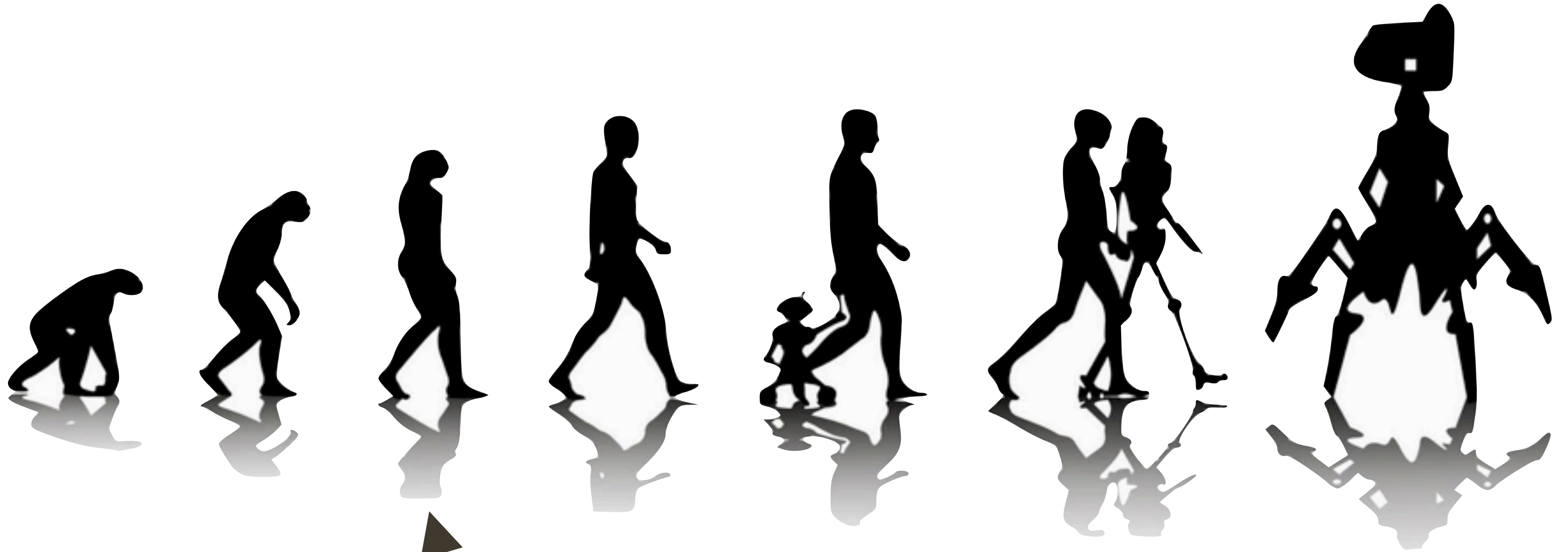
You do NOT want "Guitar Hero" Git history.

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."

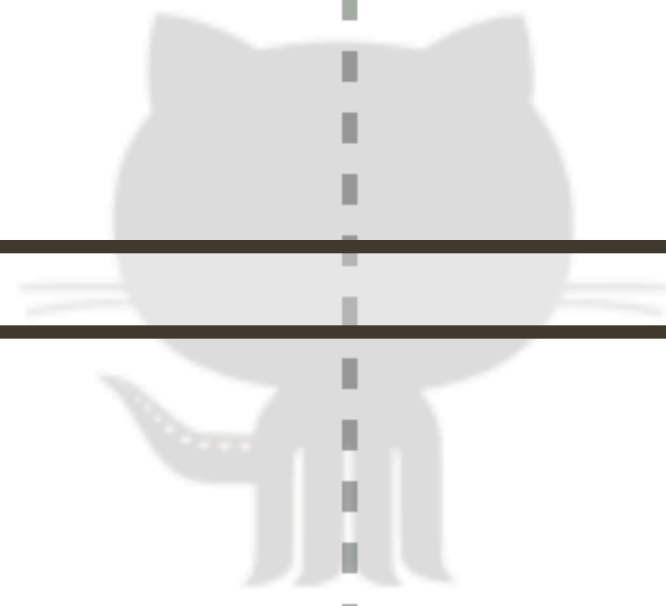
 *there be dragons* 

get your own copy of  
wtf-ascii-funtimes!

fork and clone

Why do you have to care about  
remotes, eventually?

Them



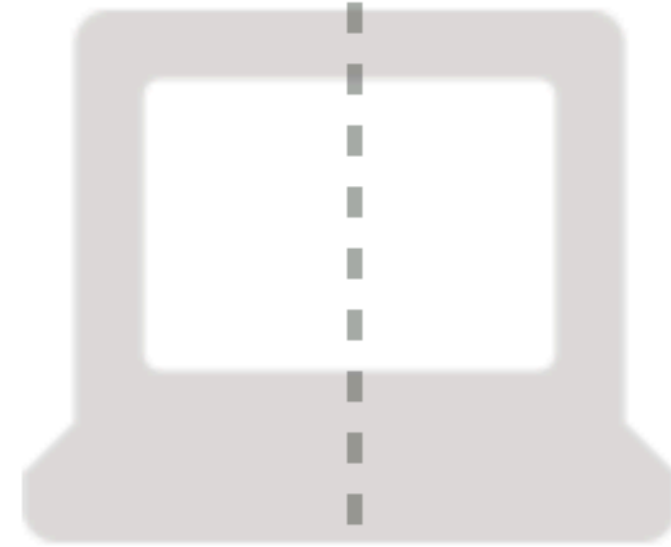
You



origin

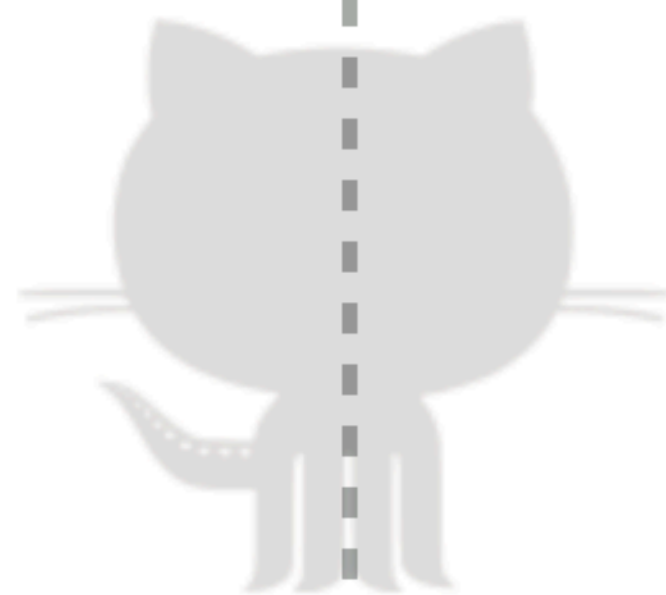


not your  
problem



"fork and clone"

Them

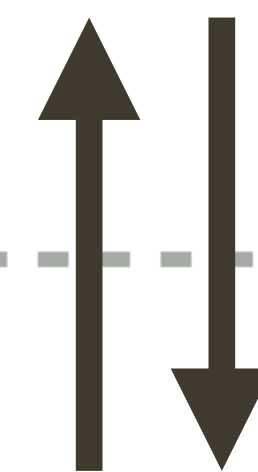


You



origin

push



pull

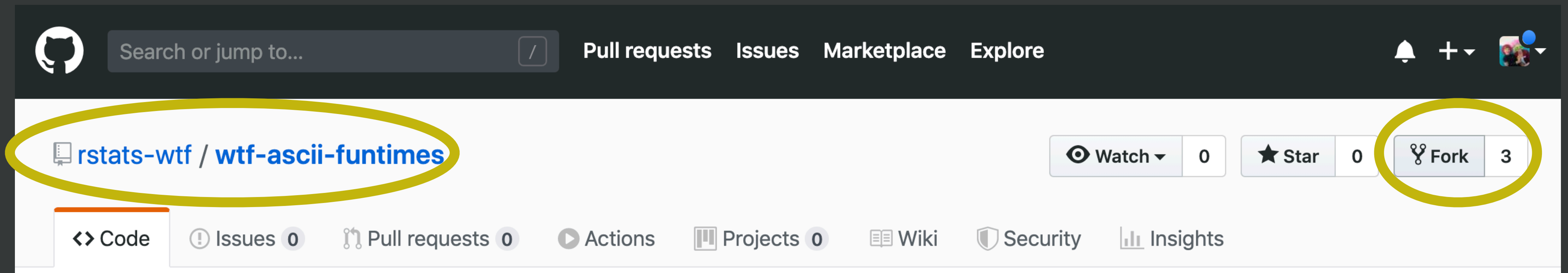


not your  
problem



daily work, your stuff

# Fork in the browser.



# Capture URL of your fork. ssh or https.

The screenshot shows the GitHub interface for a repository named 'wtf-ascii-funtimes' by user 'jennybc'. The repository name is circled in yellow. The repository is forked from 'rstats-wtf/wtf-ascii-funtimes'. The interface includes navigation tabs for Code, Pull requests (0), Actions, Projects (0), Wiki, Security, Insights, and Settings. A description area states 'No description, website, or topics provided.' Below this, statistics show 5 commits, 2 branches, 0 packages, 0 releases, and 1 contributor. A 'Clone or download' dropdown menu is open, showing options for 'Clone with HTTPS' (circled in yellow) and 'Use SSH'. The HTTPS URL 'https://github.com/jennybc/wtf-ascii-' is visible in the dropdown. Other options include 'Open in Desktop' and 'Download ZIP'.

**jennybc / wtf-ascii-funtimes**  
Forked from rstats-wtf/wtf-ascii-funtimes

Watch 0 Star 0 Fork 3

Code Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

No description, website, or topics provided. Edit

Manage topics

5 commits 2 branches 0 packages 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

This branch is even with rstats-wtf:master.

**jennybc** Moar bunnies! Make truck taller; Make castle tower taller

`.gitignore` Initial commit

`bunny.txt` Moar bunnies! Make truck taller; Make castle tower taller

`castle.txt` Moar bunnies! Make truck taller; Make castle tower taller 19 days ago

**Clone with HTTPS** Use SSH

Use Git or checkout with SVN using the web URL.

https://github.com/jennybc/wtf-ascii-

Open in Desktop Download ZIP

Clone **your fork** with RStudio.

*File > New Project ... > Version Control > Git*

then paste https or ssh URL, pick dest dir, *Create Project*



# How to know if you're done?

In R:

```
usethis::git_remotes()  
#> ✓ Setting active project to '/Users/jenny/tmp/WTF/wtf-ascii-funtimes'  
#> $origin  
#> [1] "https://github.com/jennybc/wtf-ascii-funtimes.git"
```

YOU, not rstats-wtf

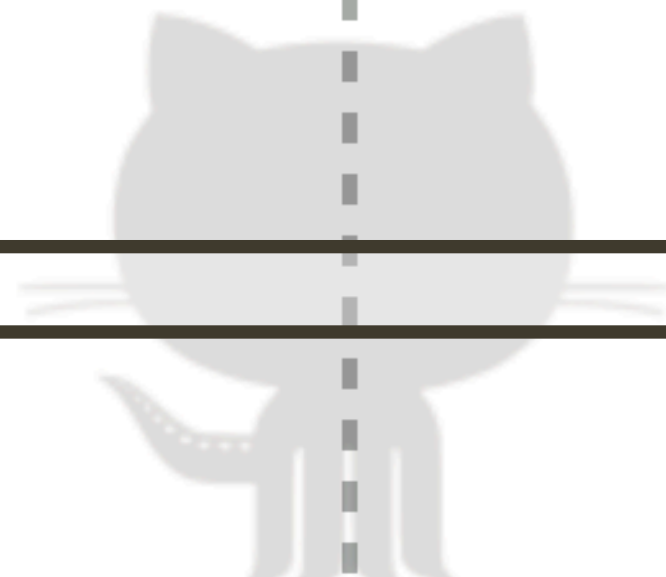


In shell:

```
jenny@2017-mbp wtf-ascii-funtimes $ git remote -v  
origin https://github.com/jennybc/wtf-ascii-funtimes.git (fetch)  
origin https://github.com/jennybc/wtf-ascii-funtimes.git (push)
```



Them



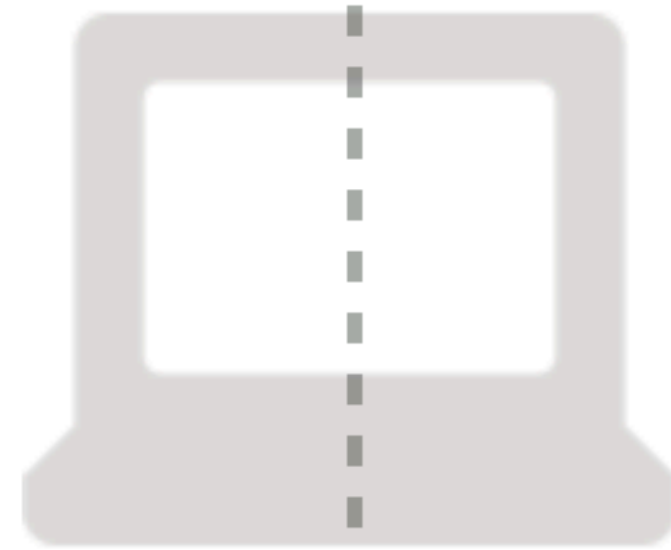
You



origin



not your  
problem



"fork and clone"

# 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.

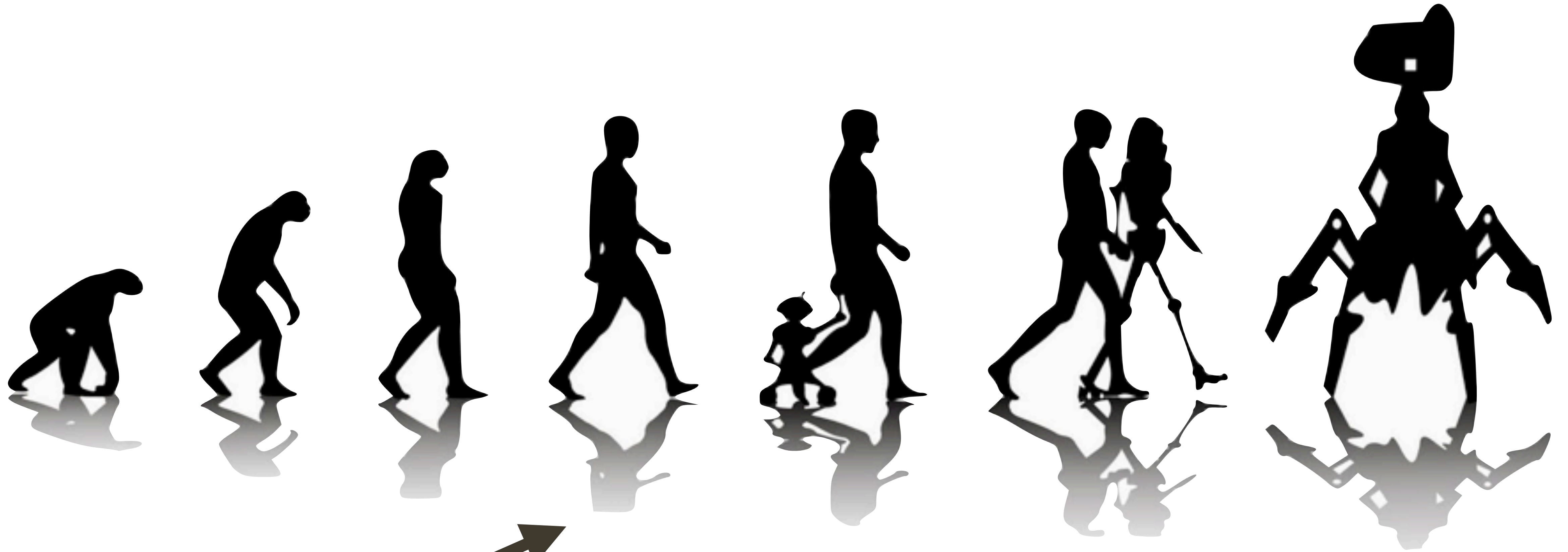
"I want to change the past."

 *there be dragons* 

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

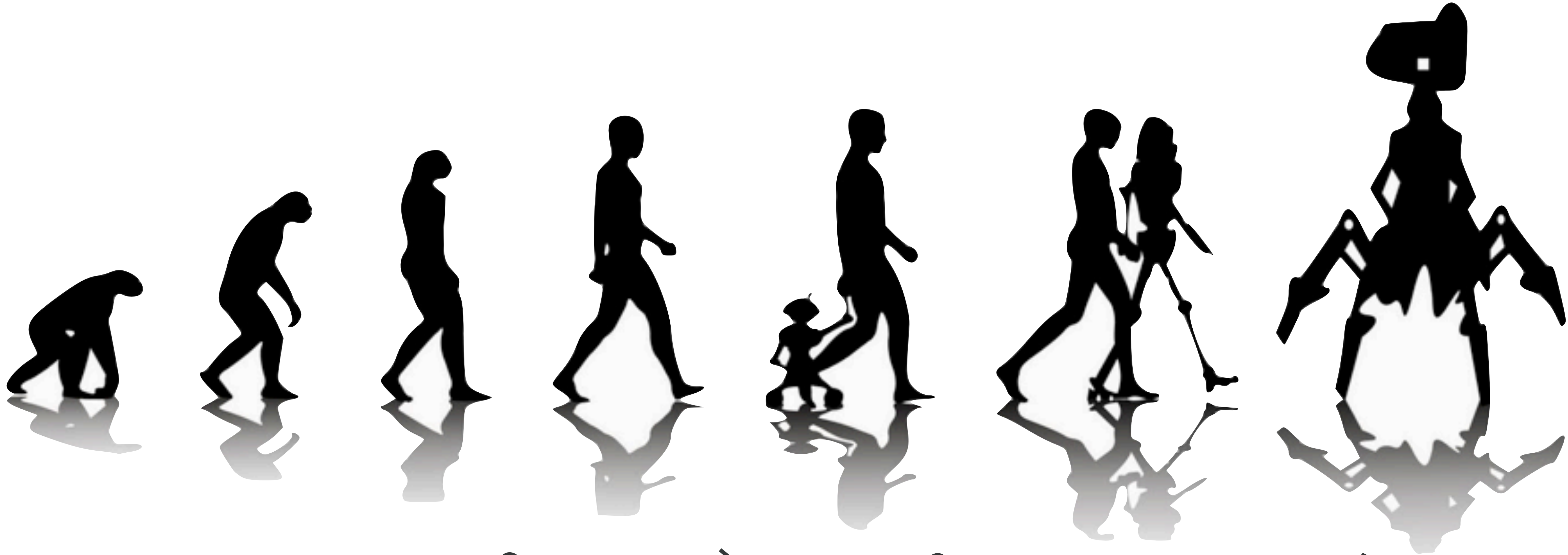
Because that is how we talk, precisely,  
about "time", with Git.

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



"commit"

a file or project state that is **meaningful to you**  
for inspection, comparison, restoration



ce25578

4792826

9037bdc

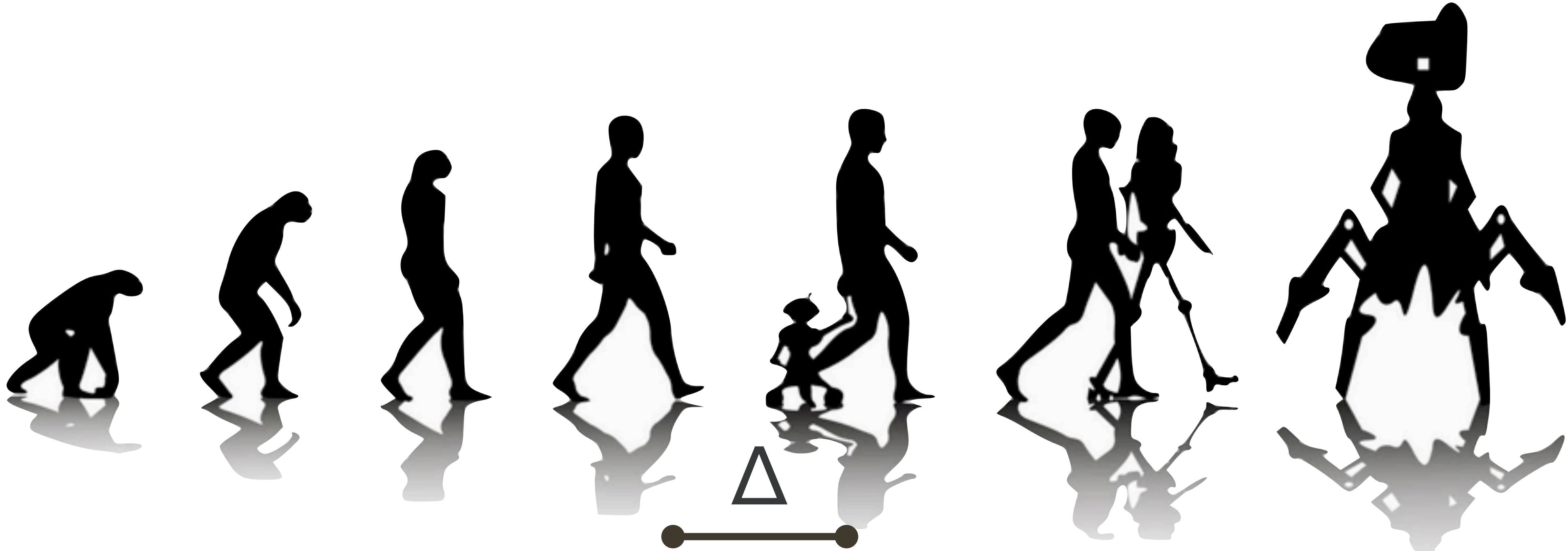
7c2a86d

01d20e1

ab06713

1d91495

commit  $\approx$  SHA-1 hash  $\approx$  SHA  $\approx$  40 chars  $\approx$  1st 7 chars



ce25578

4792826

9037bdc

7c2a86d

01d20e1

ab06713

1d91495

sometimes, a commit  $\approx$  a diff(erence)

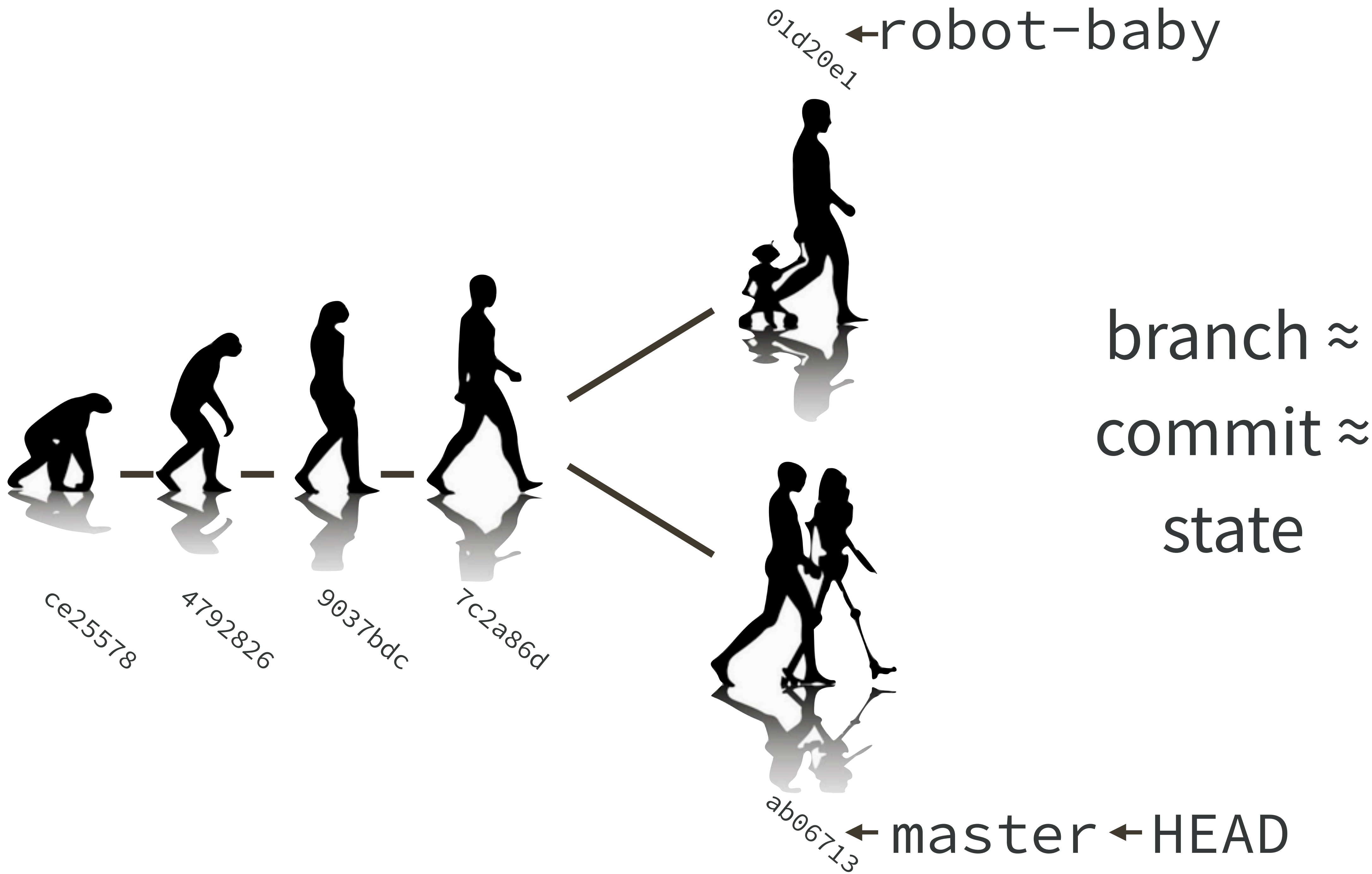


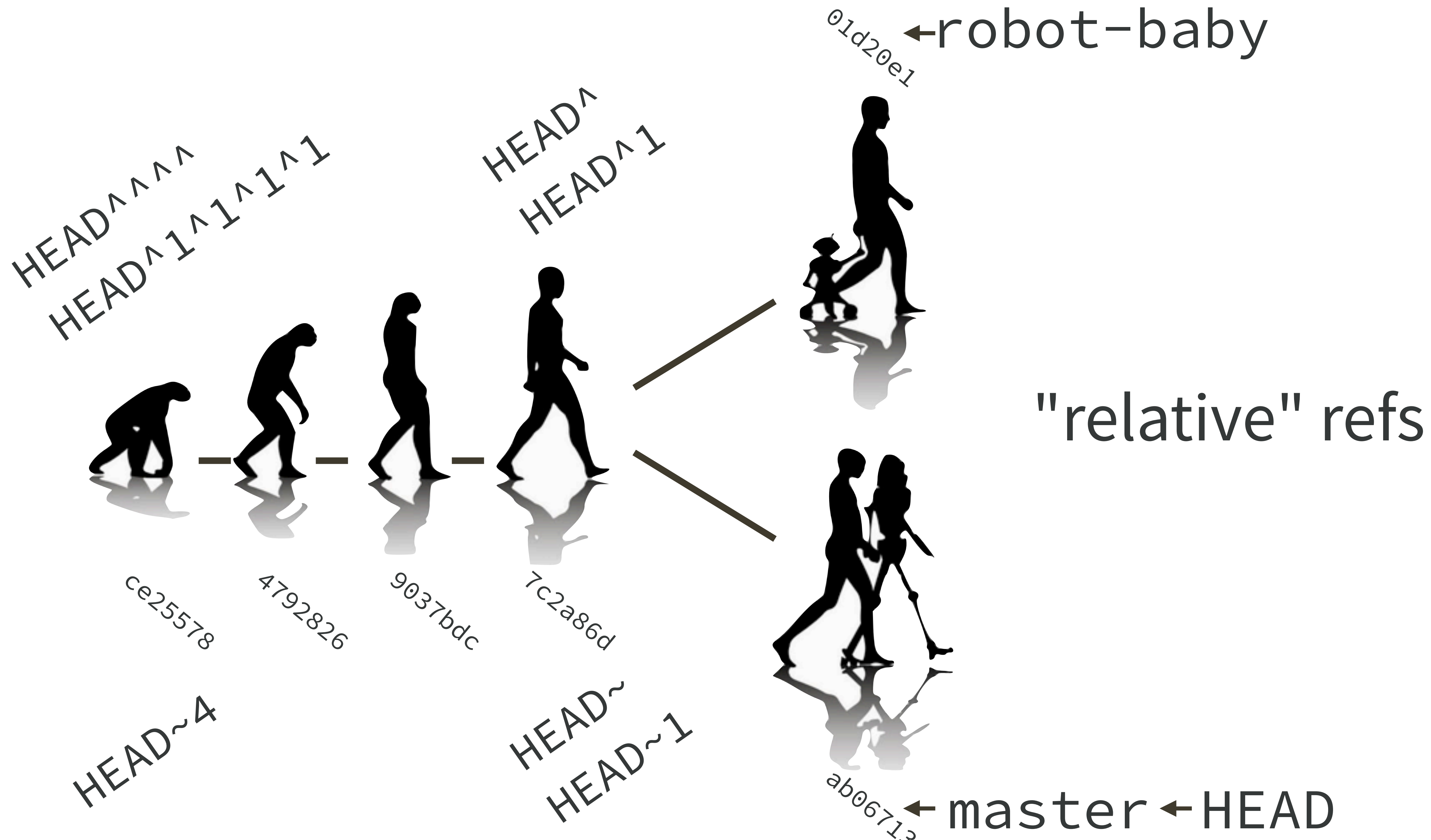
Δ + Δ + Δ + Δ

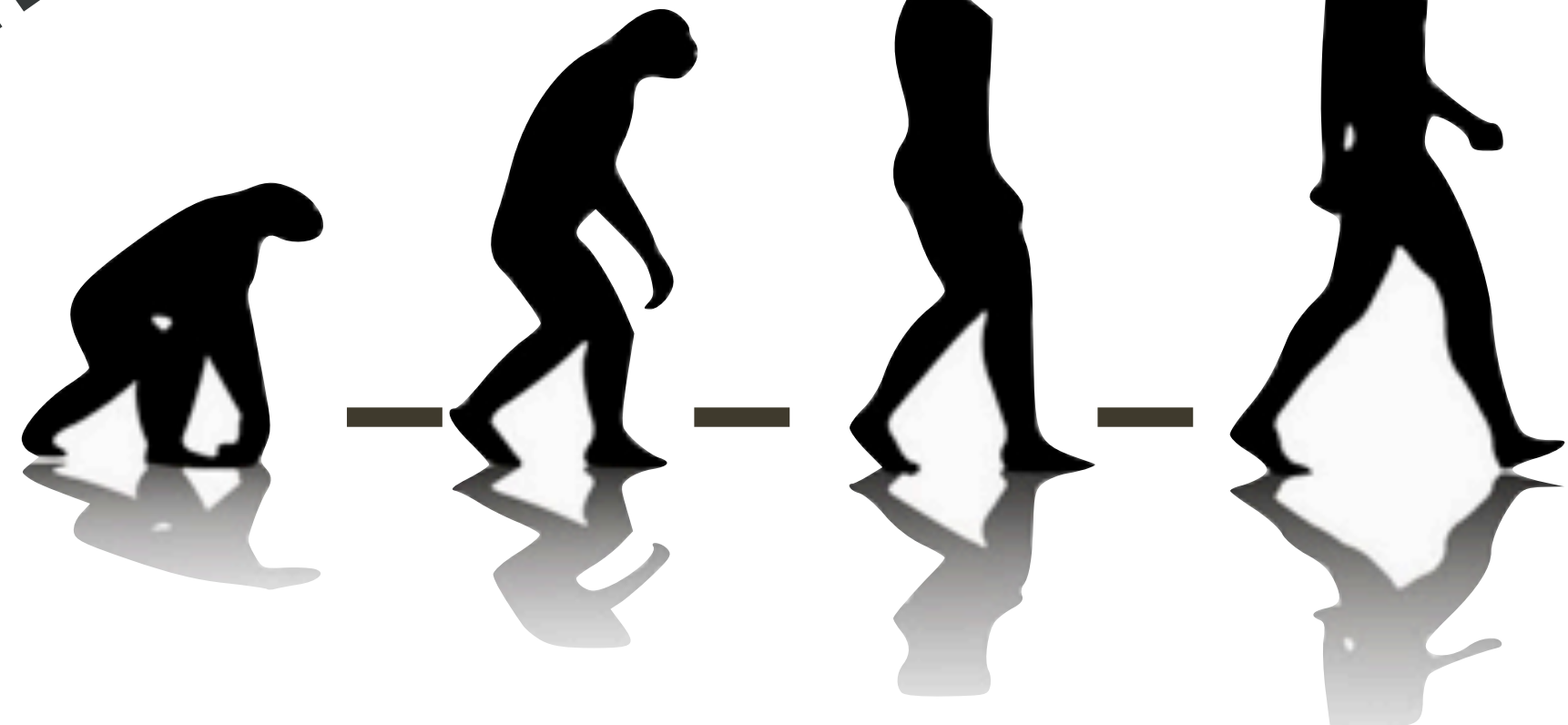
ce25578  
4792826  
9037bdc  
7c2a86d  
01d20e1

sometimes, a commit  $\approx$  a state









ce25578

4792826

9037bdc

7c2a86d

HEAD^1  
HEAD^1



HEAD~1  
HEAD~1

←robot-baby  
01d20e1



←master ←HEAD  
ab06713

I avoid this  
madness

"I need to visit the past."

Create and checkout a branch at a specific  
commit  $\approx$  state.

Then return to present  $\approx$  checkout master.

create & checkout  
a branch

at a specific state

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

```
git checkout master
```

return to present

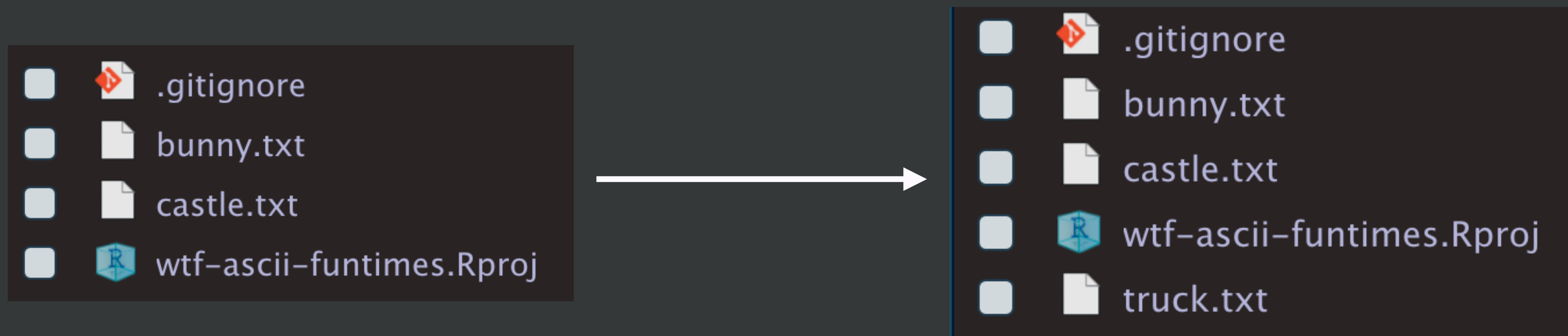
Reset your local files to this state:

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



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

Reset your local files to the present  
`git checkout master`



\* or use RStudio or GitKraken to switch back to master

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.



ce25578 4792826 9037bdc 7c2a86d 01d20e1

```
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?
<b>Commit Level</b>				
<code>reset --soft [commit]</code>	REF	NO	NO	YES
<code>reset [commit]</code>	REF	YES	NO	YES
<code>reset --hard [commit]</code>	REF	YES	YES	<b>NO</b>
<code>checkout [commit]</code>	HEAD	YES	YES	YES
<b>File Level</b>				
<code>reset (commit) [file]</code>	NO	YES	NO	YES
<code>checkout (commit) [file]</code>	NO	YES	YES	<b>NO</b>

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 committed 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

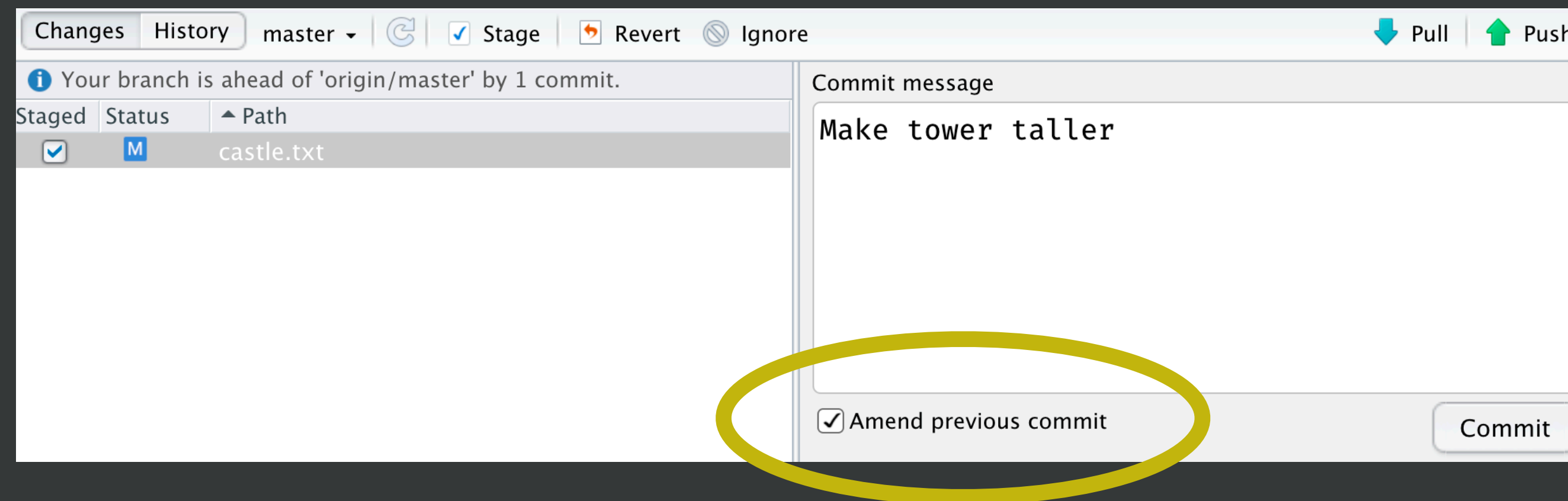
work, commit, work, amend, work, amend, work, amend, PUSH

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
```



\* 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.
```

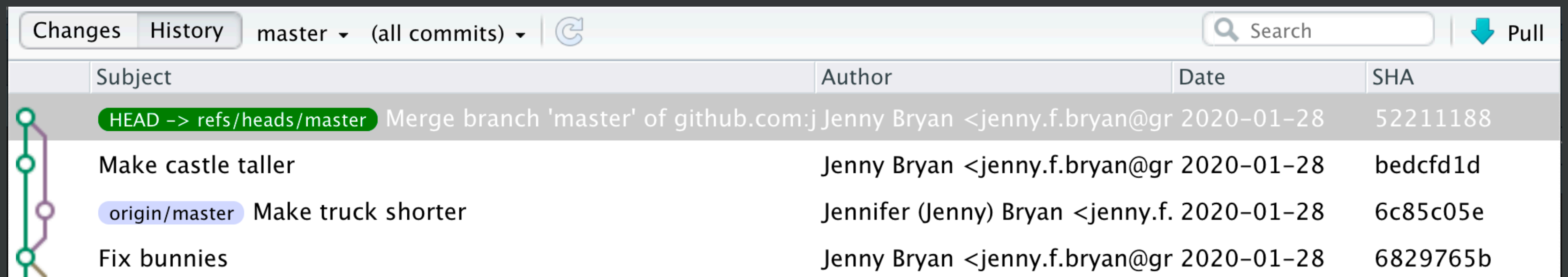
```
truck.txt | 4 ++--
```

```
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:jenny.f.bryan@gr	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



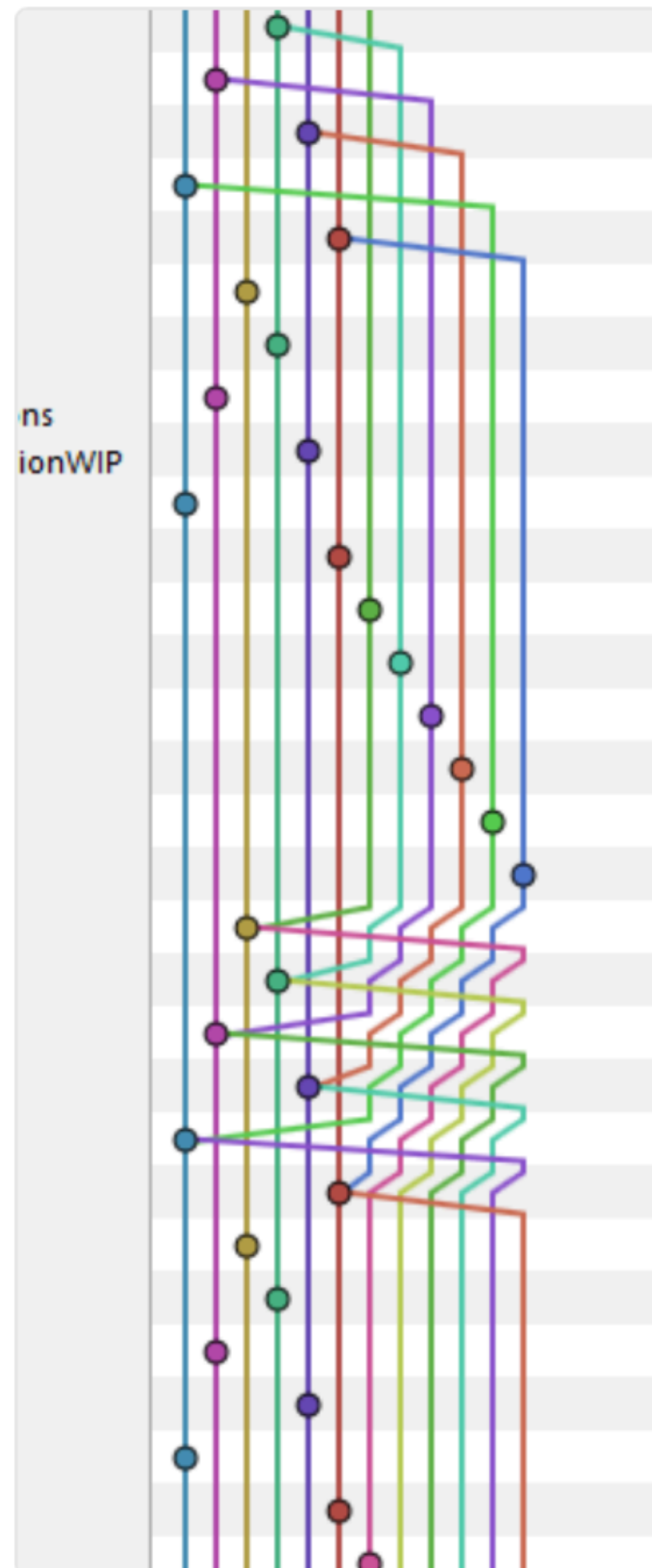
Huenry Hueffman

@HenryHoffman

Follow



I fucked up Git so bad it turned into Guitar Hero



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(_-\_  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 skills  
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.



[rstudio.io/ws-survey](https://rstudio.io/ws-survey)



**rstudio::conf**

SAN FRANCISCO // JANUARY 27 - 30, 2020

from  R Studio



## git operations via ssh

example	<code>git clone git@github.com:OWNER/REPO.git</code>
creds	local private ssh key + public key on GitHub
git operations via https	
example	<code>git clone https://github.com/OWNER/REPO.git</code>
creds	username + password (password can be GITHUB_PAT)



## GitHub API requests via REST

example	<code>curl -H "Authorization: token \$GITHUB_PAT" https://api.github.com/user/repos</code>
creds	GITHUB_PAT



```
git branch --set-upstream-to=upstream/master
```

```
git branch --set-upstream-to=origin/master
```

```
git log --pretty=oneline
```

```
git branch -d localBranchName
```

```
git push origin --delete remoteBranchName
```

```
usethis :: create_from_github(  
  "rstats-wtf/wtf-ascii-funtimes",  
  fork = TRUE,  
  destdir = "you/pick/this/path"  
)
```

- \* You MUST have a GitHub PAT setup for `fork = TRUE` to work.
- \* I didn't use today because this does too much setup! It prepares master to track upstream/master, not origin/master.