How to Be a Git Wizard

Geoff Pleiss

Goals

- (Be less afraid of git)
- Use git to **empower** your research
- Better code/experiment collaborations

“Commit early and commit often”

The golden rule of Git

“Commit early and commit often”

… but tell a story

The golden rule of Git

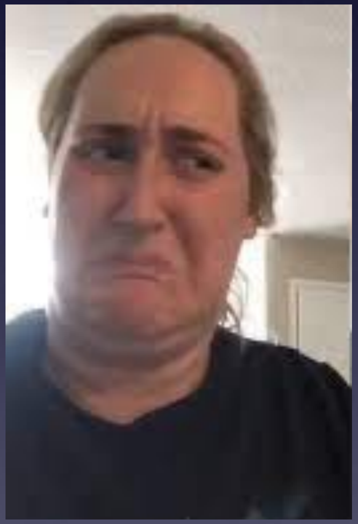
The **NEW** Golden Rule

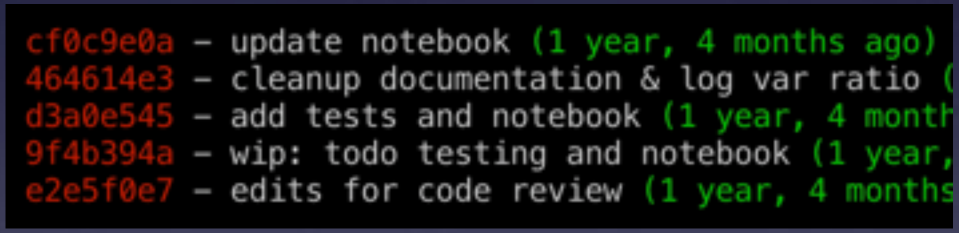
Treat your **Git history** as a first-class object

- Git history = automatic lab notebook
- Collaborate with confidence
- Easier to uncover bugs
- Higher quality code

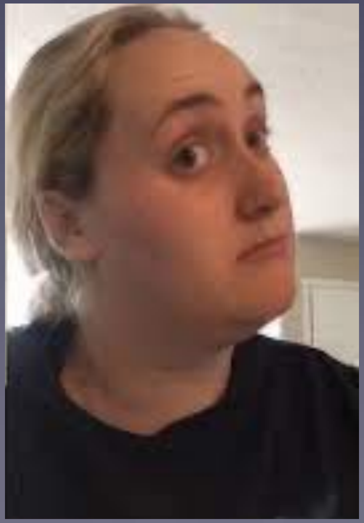
The Philosophy

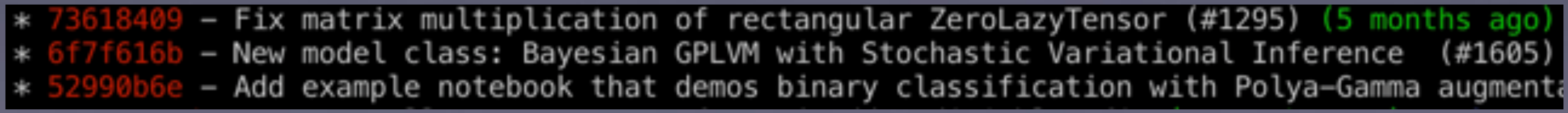
A commit should be a atomic + complete + workable idea





```
cf0c9e0a - update notebook (1 year, 4 months ago)
464614e3 - cleanup documentation & log var ratio (
d3a0e545 - add tests and notebook (1 year, 4 month
9f4b394a - wip: todo testing and notebook (1 year,
e2e5f0e7 - edits for code review (1 year, 4 months
```





```
* 73618409 - Fix matrix multiplication of rectangular ZeroLazyTensor (#1295) (5 months ago)
* 6f7f616b - New model class: Bayesian GPLVM with Stochastic Variational Inference  (#1605)
* 52990b6e - Add example notebook that demos binary classification with Polya-Gamma augment
```

Why is this Good?

- Each commit is runable code
- Living research notebook
- Easy to revert/undo changes

Secret Git Commands to Improve Your History

- .gitignore / git clean -nd
- git commit --amend
- git revert
- git add/checkout/reset -p
- git rebase -i

Don’t Dirty Your Repo with .DS_Store, *.pyc, etc.

- **.gitignore** (prevent useless files from being tracked)
    - Github has many language-specific .gitignore files that you can prepopulate your repository with
- **git clean -df** (remove untracked files)

One Fun Trick

```
echo “data/*” >> .gitignore

touch data/.gitkeep

git add -f data/.gitkeep

git commit -m “Prepopulate project with a data
folder (but don’t add any actual datasets)”
```

Make Each Commit Atomic (w/ Patch Mode)

- **git add -p <file_pattern>** (choose which lines to stage)
- **git reset -p <file_pattern>** (choose which lines to unstage)
- **git checkout -p <file_pattern>** (choose which lines to undo)

Aside: mastering git reset

- **git reset --soft HEAD^** (undo the last commit, but keep the changes in you working directly)
- **git reset --hard HEAD^** (undo the last commit, and completely remove the changes)
- **git revert HEAD^** (add a new commit that undoes the last previous commit - *good if you have already pushed your last commit!*)

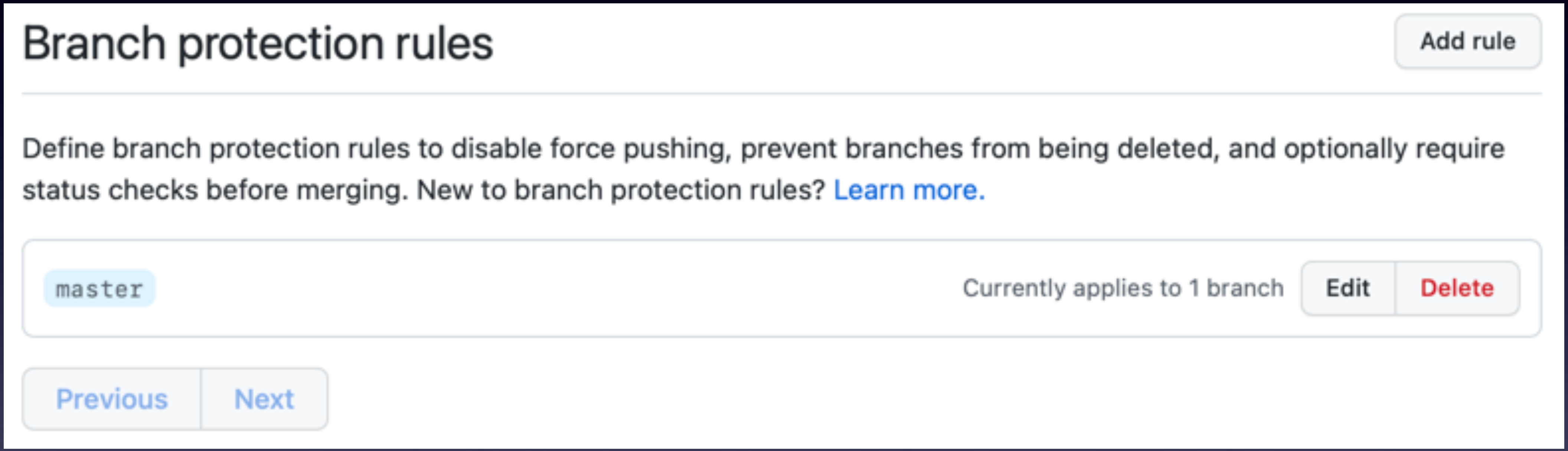
THE BEST COMMAND OF ALL TIME!!!

(AKA how to rewrite history)

- **git rebase -i HEAD^^^^^**
  - Merge WIP commits into a single commit
  - Reorder commits
  - <u>WARNING: You can't (shouldn't) rewrite the history after you've pushed!</u>

Writing Good Code With Others (and how git can help)

The Pull Request Strategy


Branch protection rules

Define branch protection rules to disable force pushing, prevent branches from being deleted, and optionally require status checks before merging. New to branch protection rules? Learn more.

`master` Currently applies to 1 branch

Everything must be a pull request!

Everything is developed on *branches*

The **Golden Rules** For Branches (AKA “Merge Early Merge Often”)

- If a branch is > 5 commits long, **merge it**
- If a branch is > 1 week old, **delete it**

Why is this Good?

- Enforces (quick) code review
- New “features” are easier to digest/understand
- Code doesn’t become stale

The 3 Branch Strategies

- **git merge --ff-only <branch>** (forces no merge commit)
- **git merge --no-ff <branch>** (forces merge commit)
- **git rebase <branch>** (your commits are merged on top of <branch>)

My Strategy For Clean History

```
git checkout my_branch

# … write good code

git checkout main

git pull origin main

# Now to update my_branch to build off of main

git checkout my_branch

git rebase master

git checkout master && git merge my_branch --no-ff  # or make a PR
```

Other Tips for Working Effectively with Others

- <u>**Have only 1-2 runnable files**</u>
  - This forces you to *write reusable code*, and forces everyone to *work on the same files*
  - More effort at first, but it pays off!
- **git blame** is your friend!

Aside: mastering git diff

- **git diff** (what's new and *unstaged* since my last commit)
- **git diff -w** (what's new and unstaged, *ignoring whitespace)*
- **git diff --cached** (what's new and *staged* since my last commit)
- **git diff <sha>..master** (what's changed since <sha>)
- **git diff <sha>..master --stat** (only list the changed filenames)

“Oh @&$#!” (Solving a Git Crisis)

“Ahhhhhh Conflicts!!!”

- **git merge --abort**
- **git rebase --abort**
- (You’ll still have to resolve conflicts some day, but maybe you can clean your commits first!)

“Ahhhhhhh I Deleted A Commit!”

- (This happens when you rebase too aggressively)
- **git reflog** (the *unchangeable* git history)
- Pair this with **git cherry-pick <sha>** (add a single commit to your history)

“Ahhhhhhh How Long Has This Bug Been In The Codebase?”

- **git bisect** (efficient binary search to find the first “bad” commit)