Effective git use

Hall A/C Analysis Workshop

June 25-26, 2018

Stephen Wood

Why version control?

```
$ git log --oneline -reverse
…
a0de8a6 Add figure captions
decea72 Final proofing, ready for advisor
3ce5c28 Make advisors corrections
69c0f4c Address advisors comments
152d0f5 Deal with advisors corrections
e093339 Why did I come to grad school?

$ git diff decea72 e093339
```
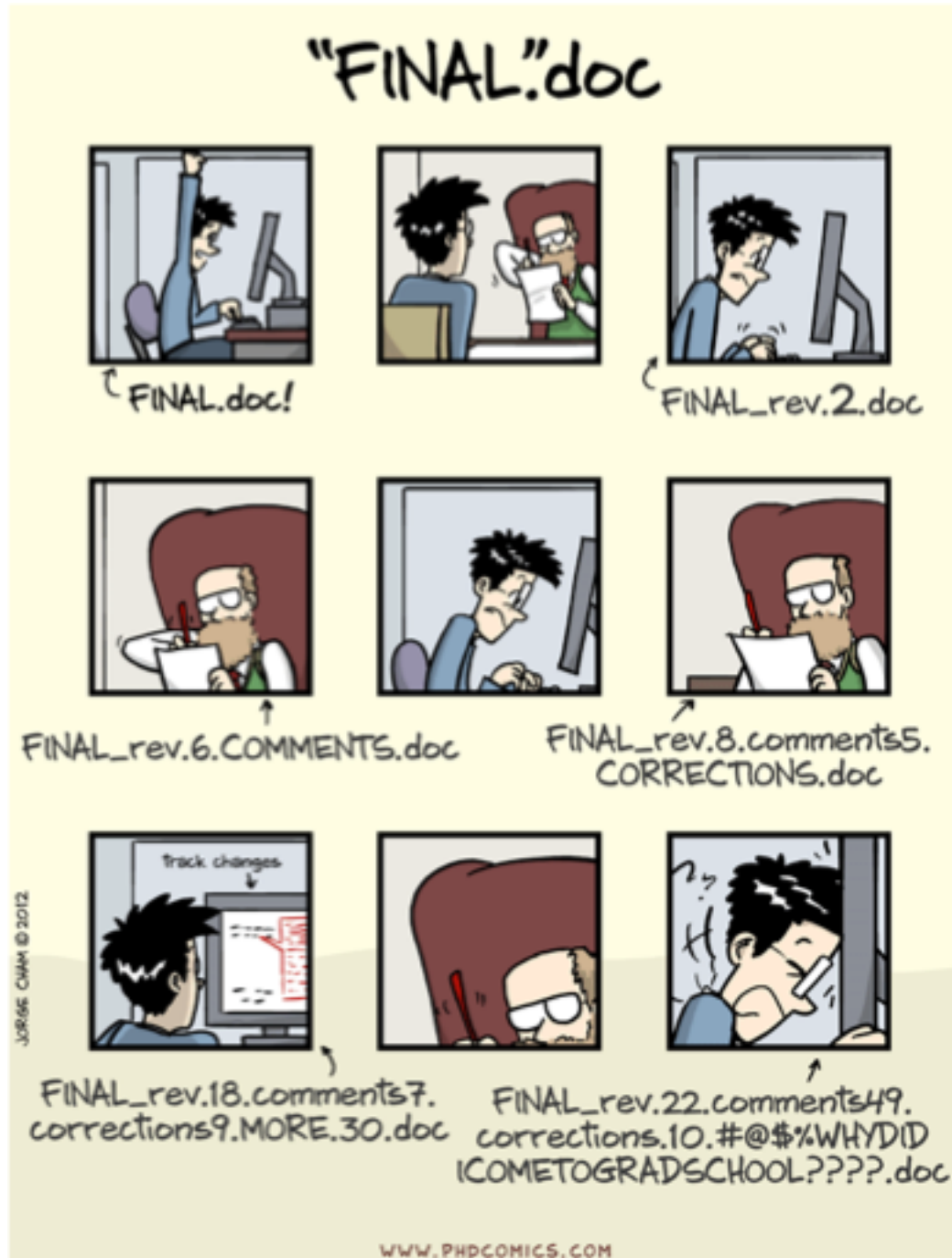
Collection of links to information about git:
https://hallcweb.ilab.org/wiki/index.php/Git_Howto

YouTube video series:
"Git and GitHub for Poets"

https://youtu.be/BC0HnlnPusY

"Piled Higher and Deeper" by Jorge Cham,

"FINAL".doc

FINAL.doc!

FINAL_rev.2.doc

FINAL_rev.6.COMMENTS.doc

FINAL_rev.8.comments5.
CORRECTIONS.doc

track changes

FINAL_rev.18.comments7.
corrections9.MORE.30.doc

FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc

JORGE CHAM © 2012

WWW.PHDCOMICS.COM

Advice

Use git for everything
> Don't need a server (GitHub) for personal projects
>
> Text/Tex, Web pages, poetry, configuration files, Reports, publications, theses

Don't work in the "master" ("develop" in case of hcana) branch
> Keep master branch in sync with main server

Keep crap out of repository with .gitignore
(intermediate files, binaries, emacs junk, latex junk, root files, …)

Use branches liberally

Commit early and often – commitment is not forever!
  First line of commit comment should summarize the changes

Rewrite your commit history before putting in public repository
   Make development process look logical, improve commit comments

Learn how to resolve conflicts when merging and rebasing

Learn and follow "rules" of projects you join

Read online git tutorials, practice

In case of fire 🔥

1. `git commit`

2. `git push`

3. `leave building`

GitHub.com

github.com is a git server with many added features

GitHub != git   --   GitHub not needed for personal projects without collaborators

By default, projects on GitHub are public

JeffersonLab has a corporate account. Allows private repositories, most are public.
         (private projects still visible to JLab users)

Anyone can setup unlimited projects repositories.
         Price = all your stuff is public

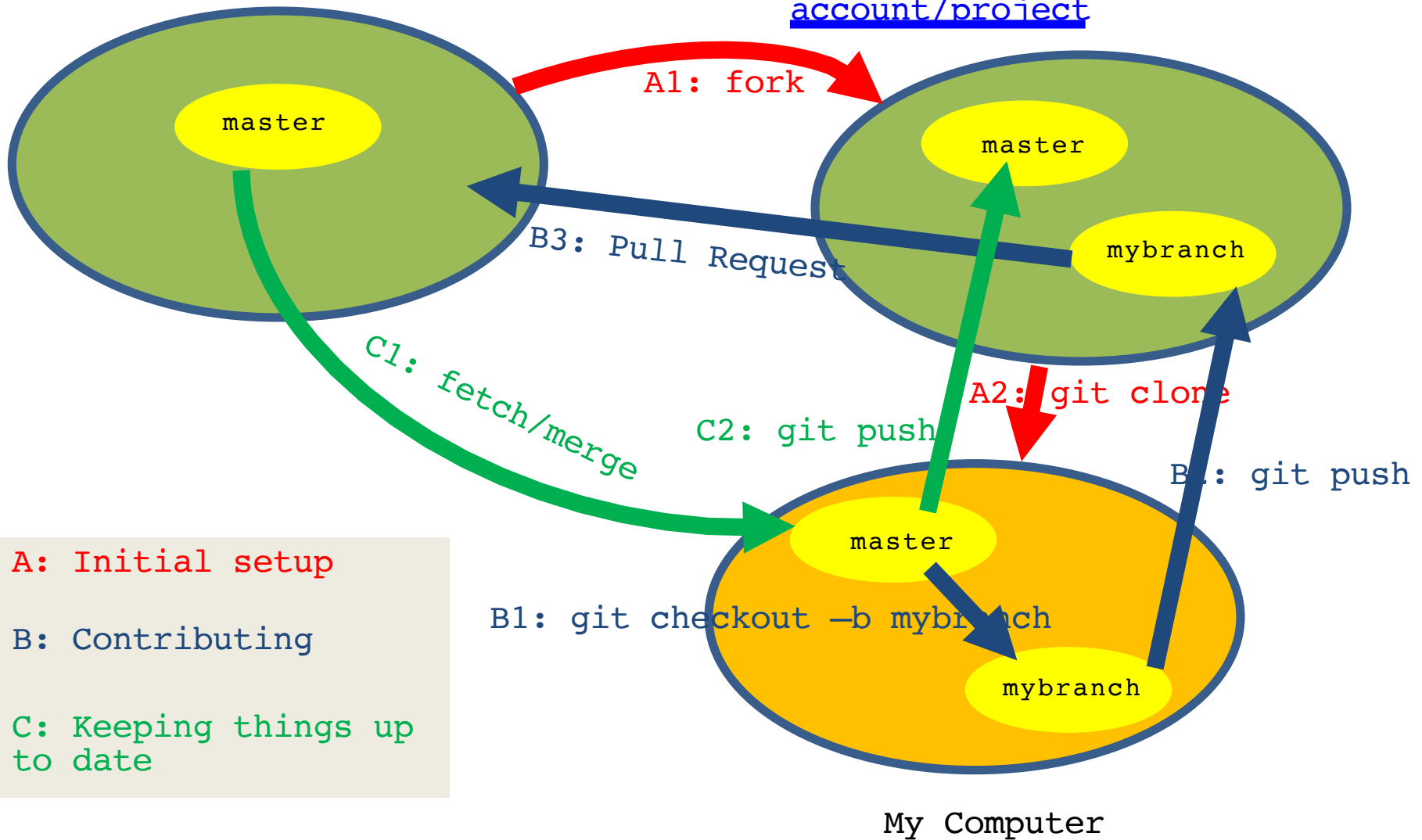There are many other free and $ git servers. We just happen to use GitHub.

More on JLab GitHub use, see Tyler Hague's talk yesterday:

https://redmine.jlab.org/attachments/download/185/TritiumAnalysisOrganization.pdf

Contributing to typical Hall A/C project

GitHub.com - collaboration

To collaborate on a Jlab GitHub project (e.g. analyzer, hcana, hallc_replay, …) get your own GitHub account.

Install your public ssh key on GitHub (or give password on git push)

Visit project page – e.g. https://github.com/JeffersonLab/hallc_replay

Fork the project (top right)

Clone the "fork"

```
git clone git@github.com:YOURUSERNAME/hallc_replay.git
cd hallc_replay
git remote add --track master upstream https://github.com/JeffersonLab/hallc_replay


Git checkout –b myaddedfeatures
```
  Do stuff
```
git push origin myaddedfeatures
```

On hallc_replay project page on GitHub, select "myaddedfeatures" branch and click "New pull request". Wait for project manager to merge changes. Update your "master" branch.

See https://hallcweb.jlab.org/wiki/index.php/Analyzer/Git for detailed "hcana" information

git – Authorship

Sample commits from Hall C analyzer **hcana**

```
$ git log
...
commit d6e15d1667932495ec1c3e7e4723314cc496838d
Author: Carlos Yero <cyero002@fiu.edu>
...
commit 4254664d54055f61046ed90d1a0e901d3f980342
Author: hallc-online <hallconline@gmail.com>
...
commit d3c4f8c1938f408968443d977f141981dd9f1d1
Author: hallc-online <hallconline@gmail.com>
...
commit f26d1d0dde44da558e67940950b9c1d00548269c
Author: hallc-online <hallconline@gmail.com>
```

Commits on Feb 12, 2018

Revert "fix the crash for the new lib"
    sawjlab committed on Feb 12

fix the crash for the new lib
    sawjlab committed on Feb 12

fix the crash for the new lib
    sawjlab committed on Feb 12

I didn't make these commits to HallA-Online-Tritium.

Who made these commits?

Commits were made on counting house analysis accounts.

How to show proper authorship with out making other authors show up as yourself?

git – Overriding authorship

1. Override default author with:

    $ git commit –author="Susan B. Anthony <dollar@treasury.com>" {files}

    Need to remember to use "—author" every commit. If you forget, can do:

    $ git commit –author="Susan B. Anthony <dollar@treasury.com>" --amend

    (if done before commit is pushed to GitHub)

2. Or set author for personal clone on analysis account:

    ```
    cd directorywiththepersonalclone
    git config user.name "your name"
    git config user.email xxx@ilab.org

    cat .git/config
    ```

3. Or set environment variables:

    ```
    GIT_AUTHOR_NAME
    GIT_AUTHOR_EMAIL
    ```

The ".gitignore" file

Avoid including "derived" files and editor junk in git repository.
.o files, executables, log files, intermediate files, etc.

Create a file ".gitignore" with one line for each thing to ignore.

```
*~
\#*\#
*.o
*.root
ROOTfiles/
…
```

```
git add .gitignore
git commit .gitignore
```

Or edit ".git/info/exclude"

If for whatever reason you don't want to edit .gitignore:

Add lines to

```
.git/info/exclude
```
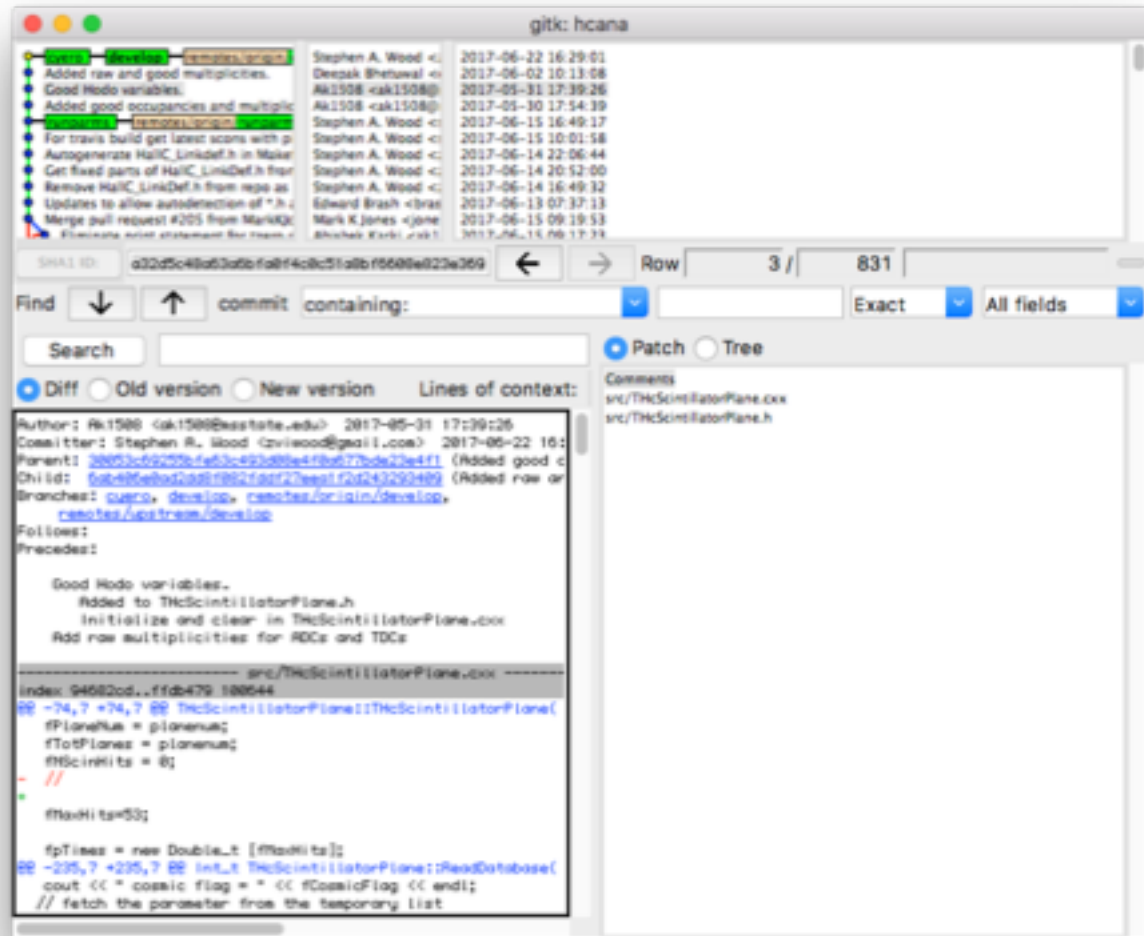
with the same syntax as .gitignore

gitk – a useful utility

Graphical display of commit history and changes.

gitk

Or "tig", similar but without Xwindows.

On your virtual machine yum install tig

git – setting defaults – Exercise 0

```
git config --global user.name "your name"
git config --global user.email xxx@jlab.org
git config --global core.editor "emacs"  (or nano or vim)
git config --global push.default simple

cat ~/.gitconfig
```

Rebasing / Editing history

Starting from "master branch"

git checkout —b workbranch

Do some work, lots of commits

git rebase —i master~/hallc

   Edit the list and save

git log --oneline
078c9b0 Create filec
b64f990 Create fileb
b3a7efc Create filea
5376520 Initial commit

git checkout master
git merge workbranch

```
pick d8f7fa2 Start working on file a.
pick ae17be0 Start working a fileb
pick 082c27e More work on filea, didn't finish, had to go home
pick 2ee43df More work on file b
pick 393b333 Fix typo in file b
pick d5a2c39 Finish filea
pick d8df357 Something else
```

```
pick d8f7fa2 Start working on file a.
squash 082c27e More work on filea, didn't finish, had to go home
squash d5a2c39 Finish filea
pick ae17be0 Start working a fileb
squash 2ee43df More work on file b
squash 393b333 Fix typo in file b
edit d8df357 Something else
```

Rebasing – Exercise 1 – page 1

1. Start a new git project
   ```
   mkdir hello ; cd hello ; git init ; git status
   ```

2. Create "README.md" file, "add" and "commit" it
   ```
   emacs README.md
   git add README.md
   git commit README.md
   git status
   ```

3. Create a working branch
   ```
   git checkout –b workingbranch
   ```

   ```
   /* hello.c */
   #include "stdio.h"
   Main(){
     printf("Hello World\n");
   }
   ```

4. Create program "hello.c"
   ```
   emacs hello.c; cc –o hello hello.c ; ./hello
   git add hello.c ; git commit hello.c
   git log
   ```

   ```
   # Makefile
   hello: hello.o


   hello.o: hello.c
   ```

5. Add a Makefile
   ```
   emacs Makefile
   ```
   Add and commit
   ```
   git status
   ```

Should see commits for README, hello.c and Makefile.

Rebasing – Exercise 1 – page 2

6. Add a line after the printf, but with a compile bug

   ```
   git commit hello.c
   make  – Observe that it doesn't compile
   ```

   `printf("Hello Sunshine\n"):`

   Use colon instead of semi-colon

7. Fix the bug

   ```
   git commit hello.c
   ```

8. Add a .gitignore file

   <span style="background:yellow">
   ```
   # .gitignore
   *.o
   hello
   ```


   ```
   git status
   emacs .gitignore ; commit .gitignore
   git status
   ```

9. Observe the history

   ```
   git log
   ```

```
$ git log --oneline
8e00f82 (HEAD -> work) Add a git ignore file
5b591b8 Fix the sunshine bug
e9aed89 Add some sunshine
e5075c2 Add a Makefile
c8bfc82 First program
4718615 (master) First commit
```

Rebasing – Exercise 1 – page 3

A more complicated Change the history to look like:

Create .gitignore

Create Makefile

Create Working hello.c (with sunshine)

```
git rebase –i master
```

pick c8bfc82 First program
pick e5075c2 Add a Makefile

pick e9aed89 Add some sunshine
pick 5b591b8 Fix the sunshine bug
pick 8e00f82 Add a git ignore file

Edit list to look like this

Save rebase list

Rewrite hello.c commit when prompted

pick 8e00f82 Add a git ignore file
pick e5075c2 Add a Makefile

pick c8bfc82 First program
squash e9aed89 Add some sunshine
squash 5b591b8 Fix the sunshine bug

If you mess up, delete all the pick, etc. lines in the buffer and save. This will abort the rebasing.

11. Merge "workingbranch" into master (or make pull request for collabortive project.)

```
git checkout master
git merge workingbranch
git log
git branch –d workingbranch
```

Rebasing – Exercise 2 – page 1

In this exercise, we have developed two independent features in parallel in a series of commits. Before merging with master (or pull requesting), we wish to rebase into a single commit for each feature, hiding our messy development process.

There are two problems:

1. After branching to "features_branch", a change that conflicts with our changes has been made.

2. On one commit we made include changes related to both features. We need to split this commit into separate commits for each feature before squshing related commits together.

1. Start a new git project
    ```
    git clone https://github.com/sawilab/rebase_exercise rebase
    cd rebase
    git checkout features_branch
    ```

2. Resolve the conflict
    ```
    git rebase master
    emacs justcode.c
    ```
    (Or use "git mergetool" if configured)

```
<<<<<<< HEAD
  printf("This is a program to compute prime …
  printf("Sieve of Eratosthenes\n);

=======

  /* Sieve of Eratosthenes */
>>>>>>> justcode: add a comment
```

Rebasing – Exercise 2 – page 2

2. (continued)
   Usually choose code before or after =====
   Here, keep it all
   Remove <<<<<, =====, and >>>>> lines.

   ```
   git rebase —continue
   ```

3. Split the combined commit
   ```
   git rebase —i master
   ```
   Replace "pick" with "edit"
   Save buffer
   ```
   git reset HEAD~
   git commit justcode.c
   git commit printfs.c
   git rebase --continue
   ```

```
pick 239792e justcode, declare i
pick 2a9192a justcode: add a comment
edit ca1ddcb justcode: bug fixes, printfs: print more numbers
pick a39322d justcode: Print out the primes
pick 53e26c1 justcode: Finally got it working
pick c174bac printfs: Add computation of Lucas Numbers
```

4. Reorder and squash
   ```
   git rebase —i master
   ```
   Reorder the commits.
   For each feature, squash extra commits.

```
pick 239792e justcode, declare i
squash 2a9192a justcode: add a comment
squash 7a6adb1 justcode: bug fixes
squash c053fdf justcode: Print out the primes
squash 1c7674c justcode: Finally got it working
pick 0e5858d printfs: print more numbers
squash a38eb3d printfs: Add computation of Lucas Numbers
```

5. Merge into master (or make pull request)
   ```
   git checkout master
   git merge features_branch
   ```