Open Source Contribution 101

Jayjeet Chakraborty

This is more of a chit chat. Feel free to jump in any time.

About me (real quick)

- PhD student in CSE (Fall 2021 -god knows)
  - Data management, databases, storage, systems
- Google Summer of Code Student 2019
- Google Summer of Code and OSRE Mentor 2020
- Been working on open-source projects in my research since then..

Google
Summer of Code

Agenda

- Why contribute to Open source
- How to start
- My experience with Open Source
- Tips and Best practices

Why Open Source ?

Don’t have the skills for this project

Code looks like Greek

I am scared of large codebases

I can work on any project. Will figure stuff out

Why Open Source ?

- Enhance your programming skills from code reviews
- Learn taking ownership of a project or a part of a project
- Learn different concepts like project structuring, testing, CI/CD, benchmarking
- Learn different tools and technologies fast and broaden your skill set
- Stay updated about the latest tech in your domain
- Go from a web/android/foo/bar developer to a **developer**

Why O

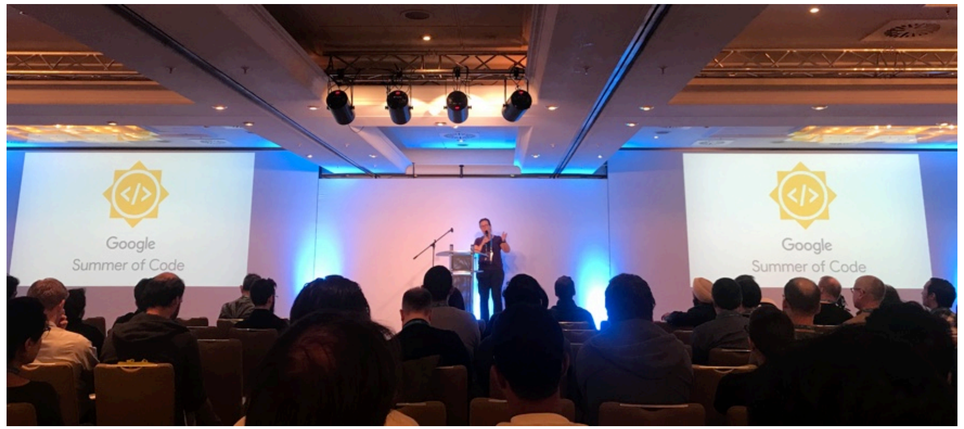
- Enhance y
- Learn taki
- Learn diffe
  benchmar
- Learn diffe set
- Stay upda
- Go from a


leetcode
open source
CS JUNIOR
JAKE-CLARK.TUMBLR
imgflip.com


leetcode
open source
CS JUNIOR
@Petirep
imgflip.com
JAKE-CLARK.TUMBLR

Why Open Source ?

- Get involved with large communities and well-known people/developers
  - Apache software foundation, CNCF, Mozilla, The Linux foundation
  - Attend conferences, make connections, travel the world for free (mostly)
  - Maybe get the next break in your career



Why Open Source ?

- Make a stronger case during job interviews
  - Mostly doesn’t work for big tech or MAANG
  - Targeted or core tech companies love seeing real work sample, show ‘em your contributions, drive the interview
  - Never do leetcode again !
  - Get several years of head start than your non-OS colleagues

Why Open Source ?

- Become a freelance consultant for a fast growing open source project
  - Devs and researchers are always looking for help with open source projects/codebases and the number is much more than you think

The estimated total pay for a Freelance Consultant is $89,814 per year in the United States area, with an average salary of $61,524 per year. These numbers represent the median, which is the midpoint of the ranges from our proprietary Total Pay Estimate model and based on salaries collected from our users.

Why Open Source ?

- Open source programs -> research projects -> researchers (PhD students/ Postdocs) -> professors—> universities -> grad school -> more amazing stuff !
  - This is what I did though..


HOW DO I
START THOUGH ?
imgflip.com

Do these now (if not already done)

- Move to Ubuntu/Mac/WSL
- Learn UNIX shell
- Learn Git, GithubSStop using GUIs
- Practice using these in your daily life

Here’s how

- Don’t drop this class
- Participate in Open Source student programs
- Clean up your class projects, push them to github, and maintain them
  - Build your open-source profile
- Make a habit of finding the source code of any project that you use or interests you
  - Fix a small bug or implement a feature
- Whenever you face an issue while using an open source project
  - Raise an issue
  - Don’t stop there, ask how to fix it
  - Raise a fix

Applying to Open Source programs

- Start looking for open-source student programs: GSoC, OSRE, Outreachy ,MLH Fellowship, Linux kernel mentorship program, GirlsScript Summer of Code, IRIS-HEP Fellowship.
- Start 3-4 months before, look for projects, reach out to maintainers, show your interest, ask for guidance on how to start
- Find a simple issue to start with, keep doing more and more till the proposal deadlines. Keep the talking going on chat, PRs, emails, etc
- Make a strong case in your proposal showing off your work and get accepted !


CAN OSS MAKE ME RICH ?
imgflip.com

Some OSS projects that made it…

Acquired by RedHat for $175M

$600M revenue in FY 2021.
Soon about to IPO

Raised $12M in Series A funding

My experience with Open Source

Some Tips and Best practices on..

- ..reading the codebase
- ..setting up the dev env
- ..writing aesthetic code
- ..contributing your code

Reading the codebase

First dive into a largish OSS project

1. Find the homepage/github repo readme and give the introduction section a quick read, maybe read a paper, or watch related videos on YouTube.

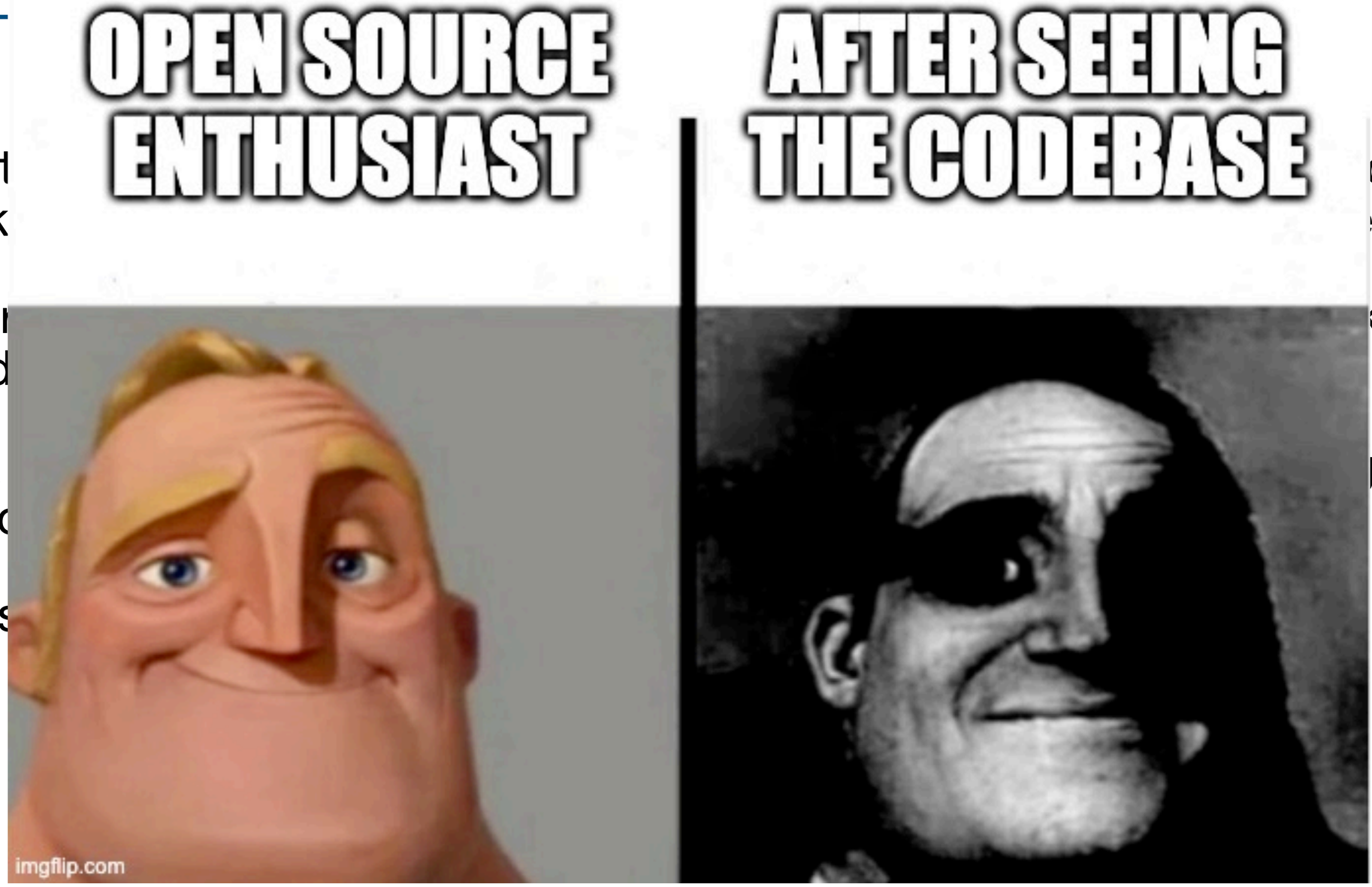
2. If there is a concept/features/design/architecture page, read it. It’s okay if you don’t understand all of it at once.

3. Look at getting started, usage guides, and example codes if any to get some idea of the API and how the project is used by the end users.

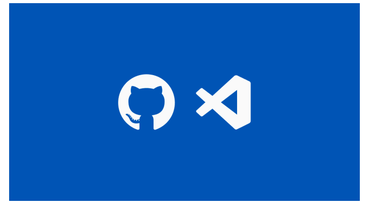
4. That’s enough. Open the source code now.

First

1. Find t ction a quick .
2. If ther kay if you d
3. Look et some idea c
4. That's



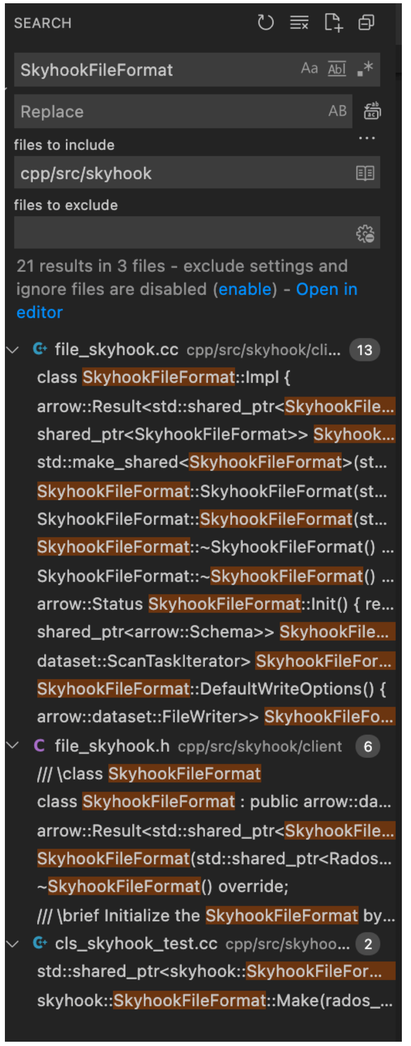
Get a feel of the source code



- Don’t try to understand/read the source code all at once. I tried that, didn’t work.
- Clone the source code or open github.dev (opens a vscode instance instantly in the browser and lets you browse the source code without cloning).
- Try to understand the high-level code structure, specifically find these things
  - Header files, Source files, Build scripts, Tests (Unit and Integration), Assets, 3rd Party Libs, Modules, Tools, Utilities, CI/CD tools, and config, etc.
- Now, either find the entry point of the code, like the main() function or use keywords in the issue to find the concerned source files.

Start dissecting the code

- After you narrowed down to a set of files)(, search for keywords inside the file. If not, ask someone on to help you find a place to start.
- Bottom-Up Approach: Use the **find** feature to track down the function/class/method you are interested in. Try to backtrack from there, see what fn calls what, and mentally create a control flow of the program.
- Again, do this just enough to be able to start making code changes.



Setting up the “dev env”

Portable and reusable development environment

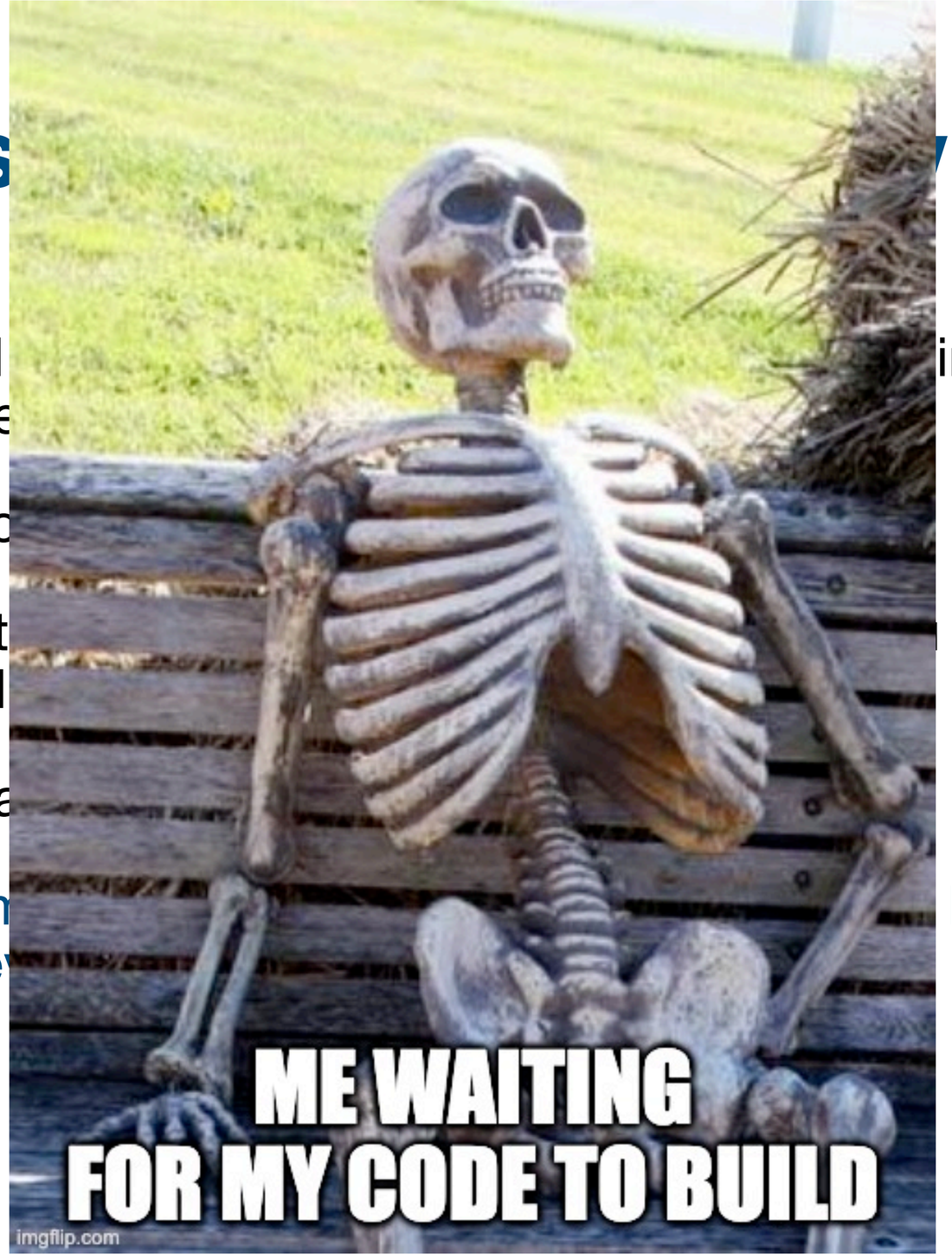
- If possible, work inside a Docker container or a Vagrant box (Virtual Machine).
- Create a Dockerfile with all the project dependencies and build tools installed and mount it to a local working directory that has the repo cloned.
  - `docker run -it dev_env -v $PWD:/workspace -w /workspace bash`
  - `sh# ./build.sh`
  - `sh# ./test.sh`
- Code on your local machine but build and test inside a Docker container.

Create a fast build/test workflow

- Use a DEBUG build for development with logging and tracing enabled but for benchmarking make sure to use a RELEASE build.
- First time compilation can be slow.
  - Try switching off the build of some components that you don’t need using CMake flags or alike.
  - Use multiple threads. make -j$(nproc).
- Automate every command run by dumping them in scripts like build.sh and test.sh as in the prev slide.

Create a fas

- Use a DEBUG build ing enabled but for benchmarking make
- First time compilatic
  - Try switching off t don’t need using CMake flags or al
  - Use multiple threa
- Automate every com like build.sh and test.sh as in the prev



Leverage CI/CD

- Keep checking CI/CD results after every commit; catch issues early
- Make sure your exact changes/tests are getting executed through CI/CD
- Run the same CI/CD job on different OS versions, architectures, and compiler versions
  - Linux, mac
  - x86, amd64, arm64
  - g++11, g++14, g++17, python3.5, python3.7, python3.9

Leve

- Keep c
- Make s
- Run th versions
  - Linu
  - x86,
  - g++

Writing aesthetic code

The easier part

Making code changes

- Now that you have narrowed down to the function/class to add/modify, it’s time to write your code
- Look at the already written code and follow those practices for your added code. If some syntax/keyword seems alien, a quick google helps
- Follow the naming conventions, code organisation, error handling strategies, and other practices followed, so that you don’t lose track of your own code eventually
- Don’t focus way too much on code quality at the beginning though. Getting things to compile and start working is the most important

Making code changes

- Now that you have narrowed down to the function/class to add/modify, it's time to write your code
- Look at the already w[illegible]tices for your added code. If some syntax[illegible]ogle helps
- Follow the naming co[illegible]r handling strategies, and other practices f[illegible]ck of your own code eventually
- Don't focus way too much on code quality at the beginning though. Getting things to compile and start working is the most important

Basically, do a brain dump in a somewhat consistent and aesthetic manner

Making code changes

- Try to keep the number of added lines more than modified lines. Find out the correct classes to inherit or the correct functions to implement for this.
  - Helps with rebase and less scrutiny during code review.
- To achieve this, take some time to compare and contrast the different ways you can implement your changes/feature. Find out a strategy that requires minimum lines of code.
- The simpler your changes the more chances of it being bug-free and fast.

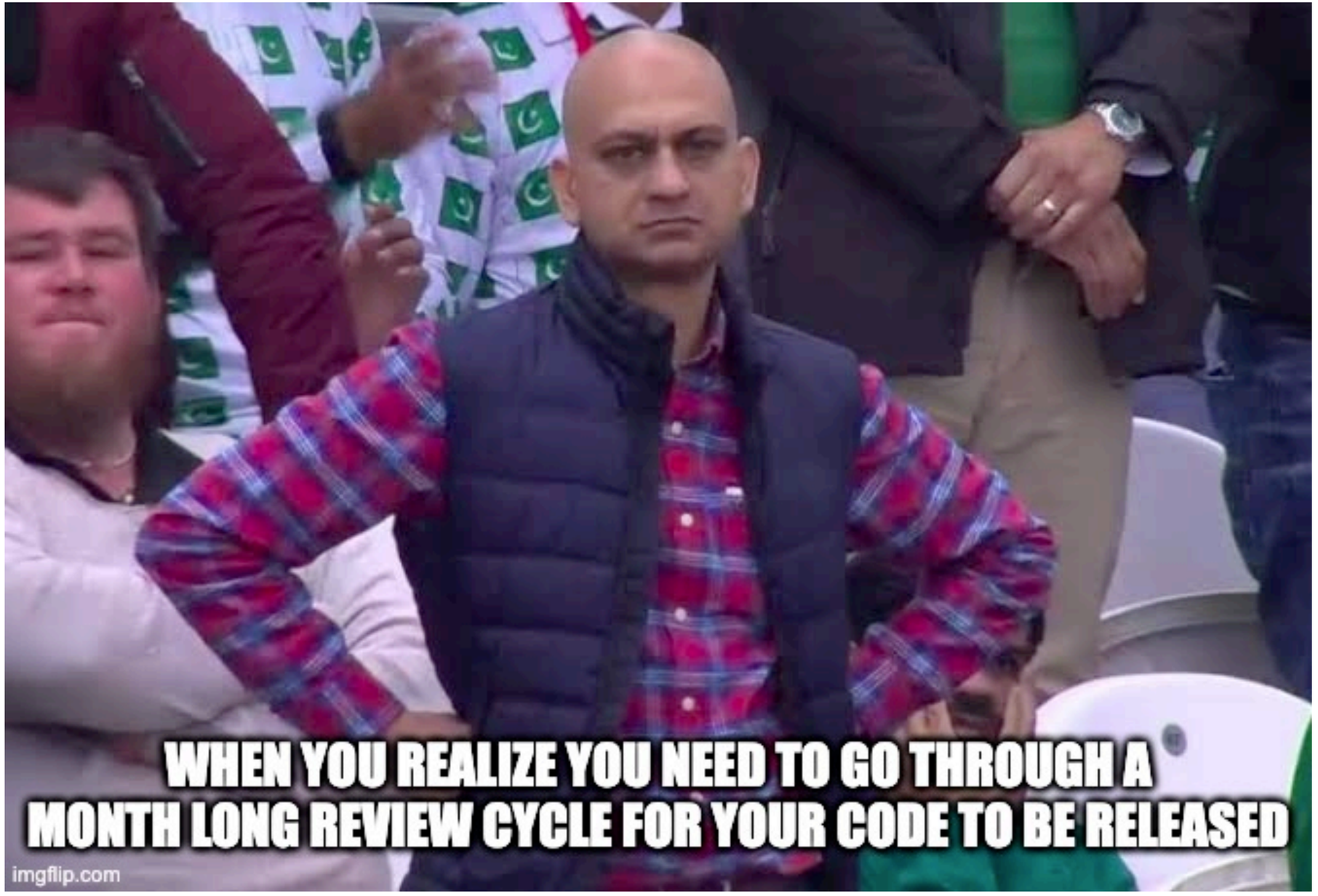
Some coding best practices…

- Variable names should make sense. Don’t use names like x, temp, mynum
- Avoid duplicated code. Create reusable functions/classes
- Handle exceptions. Write clear error messages
- Delete commented code (aka dead code) blocks before raising a PR
- Avoid hardcoding parameters
- Keep adding comments/docstrings to your code
- Commit code continuously with a somewhat descriptive message
- Always work on an experimental change in a new branch, test then branch, if works, raise PR to main branch
- The idea is to make your code indistinguishable from the previously written code

Som

- Variable
- Avoid du
- Handle e
- Delete c
- Avoid ha
- Keep ad
- Commit
- Always v branch,
- The idea written c

Contributing your code


WHEN YOU REALIZE YOU NEED TO GO THROUGH A
MONTH LONG REVIEW CYCLE FOR YOUR CODE TO BE RELEASED
imgflip.com

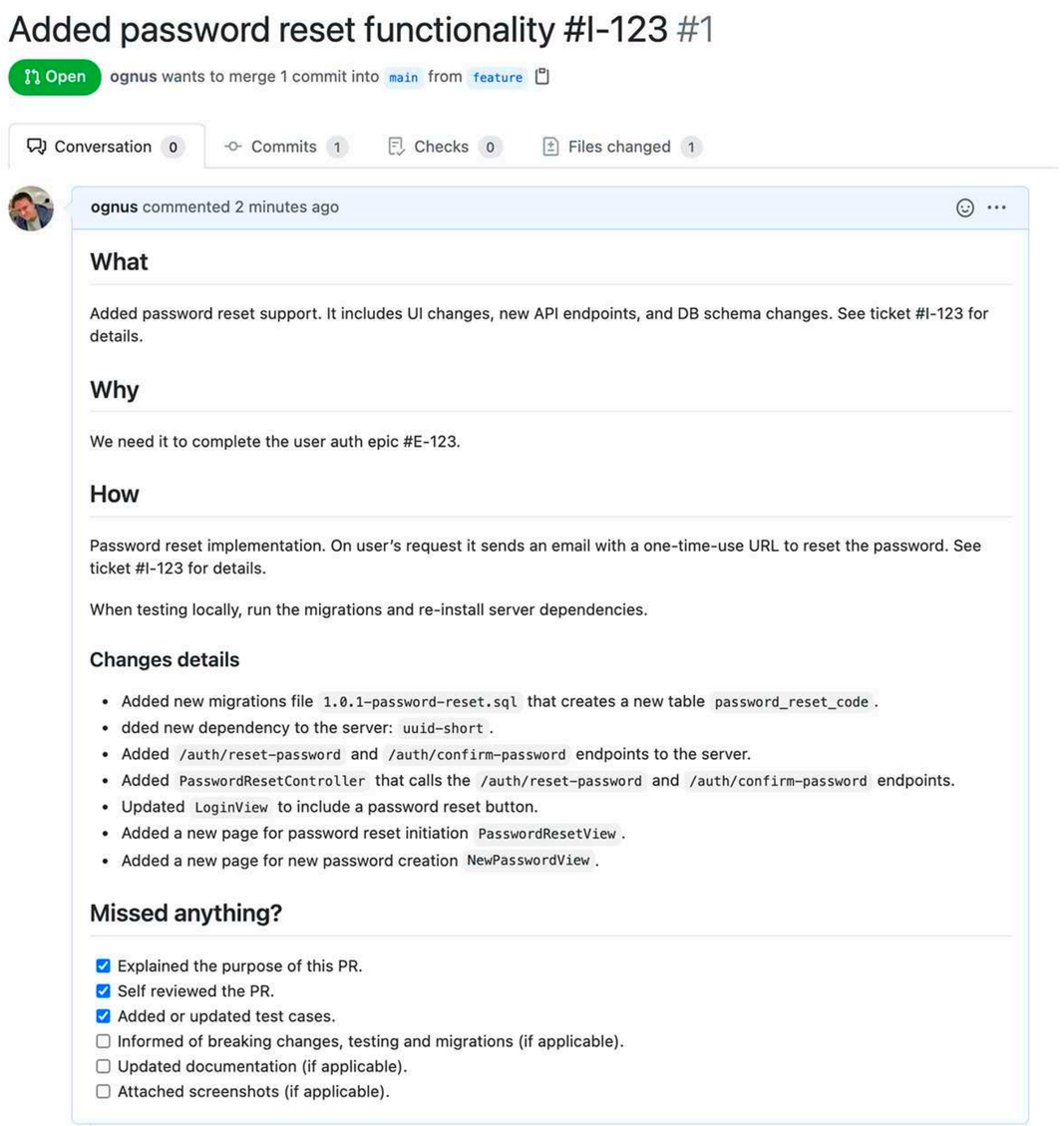
Raising a PR

- Name your branch either meaningfully or point to an issue.
  - support-csv-input
  - fix-4567
- Give a self-explanatory PR title.
  - PROJECT-4567: [C++][FileReader] Support reading CSV files
  - Or just follow the guidelines for your specific project
- Write a brief description of code, tests, and CI/CD changes.
- Ensure CI build/test is passing.

https://www.pullrequest.com/blog/writing-a-great-pull-request-description/

Raising a PR

- Name your branch eith
  - support-csv-input
  - fix-4567
- Give a self-explanator
  - PROJECT-4567: [C-
  - Or just follow the gu
- Write a brief descriptic
- Ensure CI build/test is



Added password reset functionality #I-123 #1

Open ognus wants to merge 1 commit into `main` from `feature`

Conversation 0 · Commits 1 · Checks 0 · Files changed 1

ognus commented 2 minutes ago

What

Added password reset support. It includes UI changes, new API endpoints, and DB schema changes. See ticket #I-123 for details.

Why

We need it to complete the user auth epic #E-123.

How

Password reset implementation. On user's request it sends an email with a one-time-use URL to reset the password. See ticket #I-123 for details.

When testing locally, run the migrations and re-install server dependencies.

Changes details

- Added new migrations file `1.0.1-password-reset.sql` that creates a new table `password_reset_code`.
- dded new dependency to the server: `uuid-short`.
- Added `/auth/reset-password` and `/auth/confirm-password` endpoints to the server.
- Added `PasswordResetController` that calls the `/auth/reset-password` and `/auth/confirm-password` endpoints.
- Updated `LoginView` to include a password reset button.
- Added a new page for password reset initiation `PasswordResetView`.
- Added a new page for new password creation `NewPasswordView`.

Missed anything?

- [x] Explained the purpose of this PR.
- [x] Self reviewed the PR.
- [x] Added or updated test cases.
- [ ] Informed of breaking changes, testing and migrations (if applicable).
- [ ] Updated documentation (if applicable).
- [ ] Attached screenshots (if applicable).

https://www.pullrequest.com/blog/writing-a-great-pull-request-description/

Raising a PR: Tip #1

- Make sure the code is linted before opening a PR. Try to keep the code linted after every code review change.

Raising a PR: Tip #2

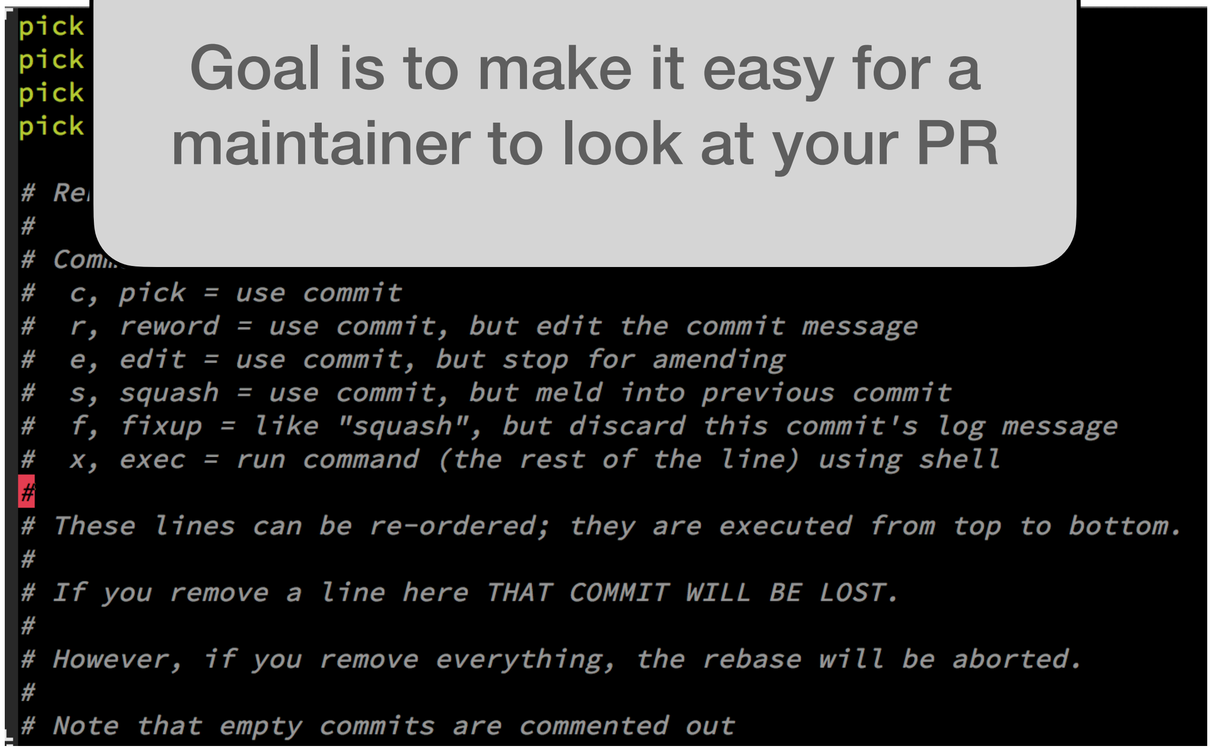
- If not already maintaining descriptive commit messages, rewrite your commit history to have small meaningful commits if you have more than 100 lines of code changes.

```
pick d05e1cb Added debugging code. To be removed.
pick 5a9126e Step 2
pick e7131a5 Found bug. Really? A tab vs. space issue?
pick 43e0adc Oops. Forgot to remove debugging code.

# Rebase e003543..43e0adc onto e003543
#
# Commands:
#  c, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Raising a PR: Tip #2

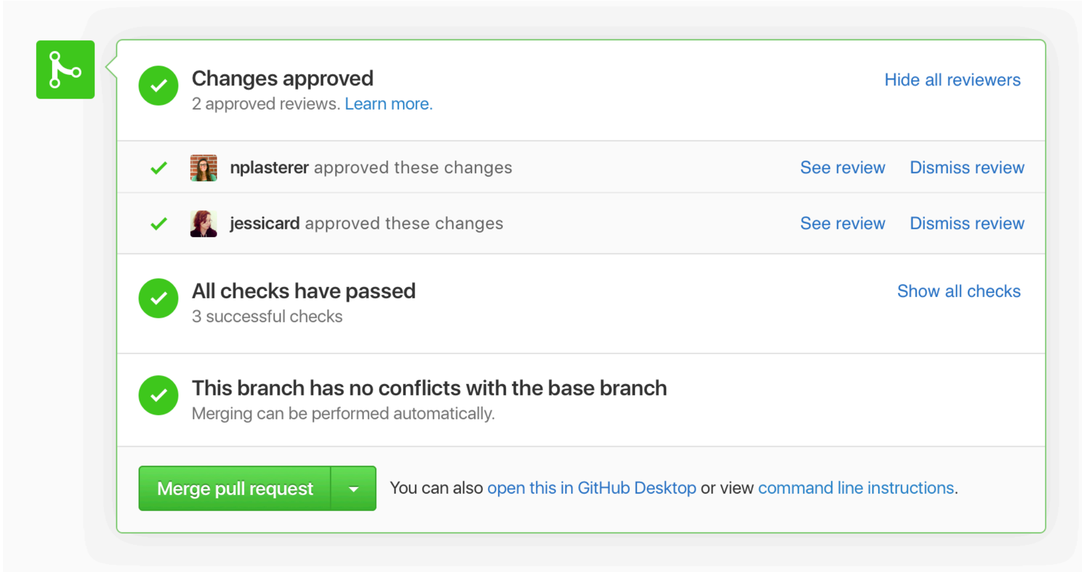
- If not already maintaining descriptive commit messages, rewrite your commit history to have small meaningful commits if you have more than 100 lines of code changes.



Goal is to make it easy for a maintainer to look at your PR

Getting your PR approved

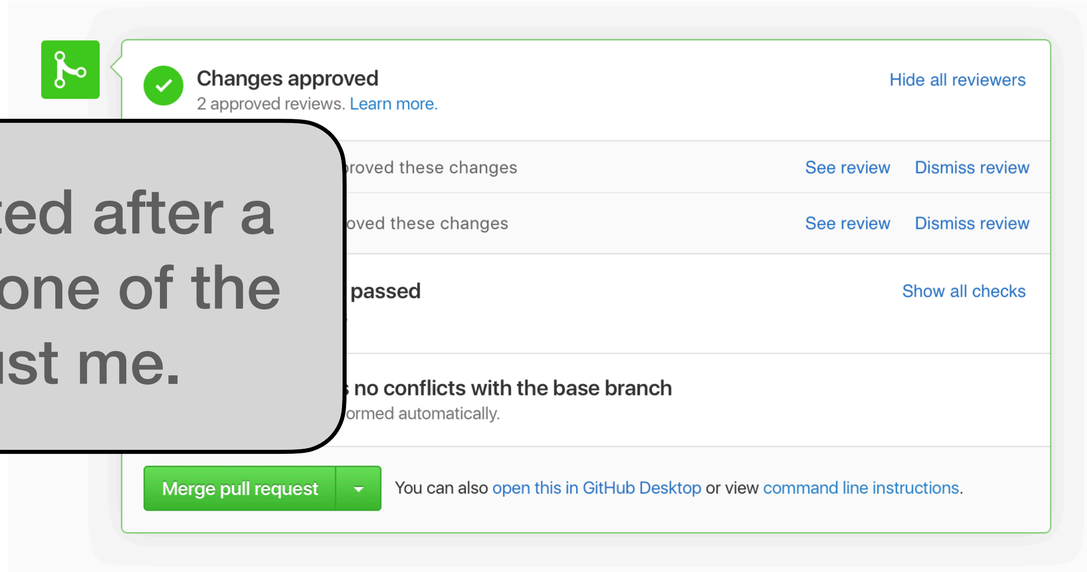
- Try to address review changes as soon as possible.
- Reply to any questions/clarifications or ask questions in a hedging language always.
- After addressing requested changes, use the “Request Review” feature of Github.
- If your PR is lying unreviewed for a some time, don’t hesitate to give a gentle reminder by tagging some maintainers on the PR.
- Keep your eye open for merge conflicts and keep resolving them regularly to keep the review cycle going.



Getting your PR approved

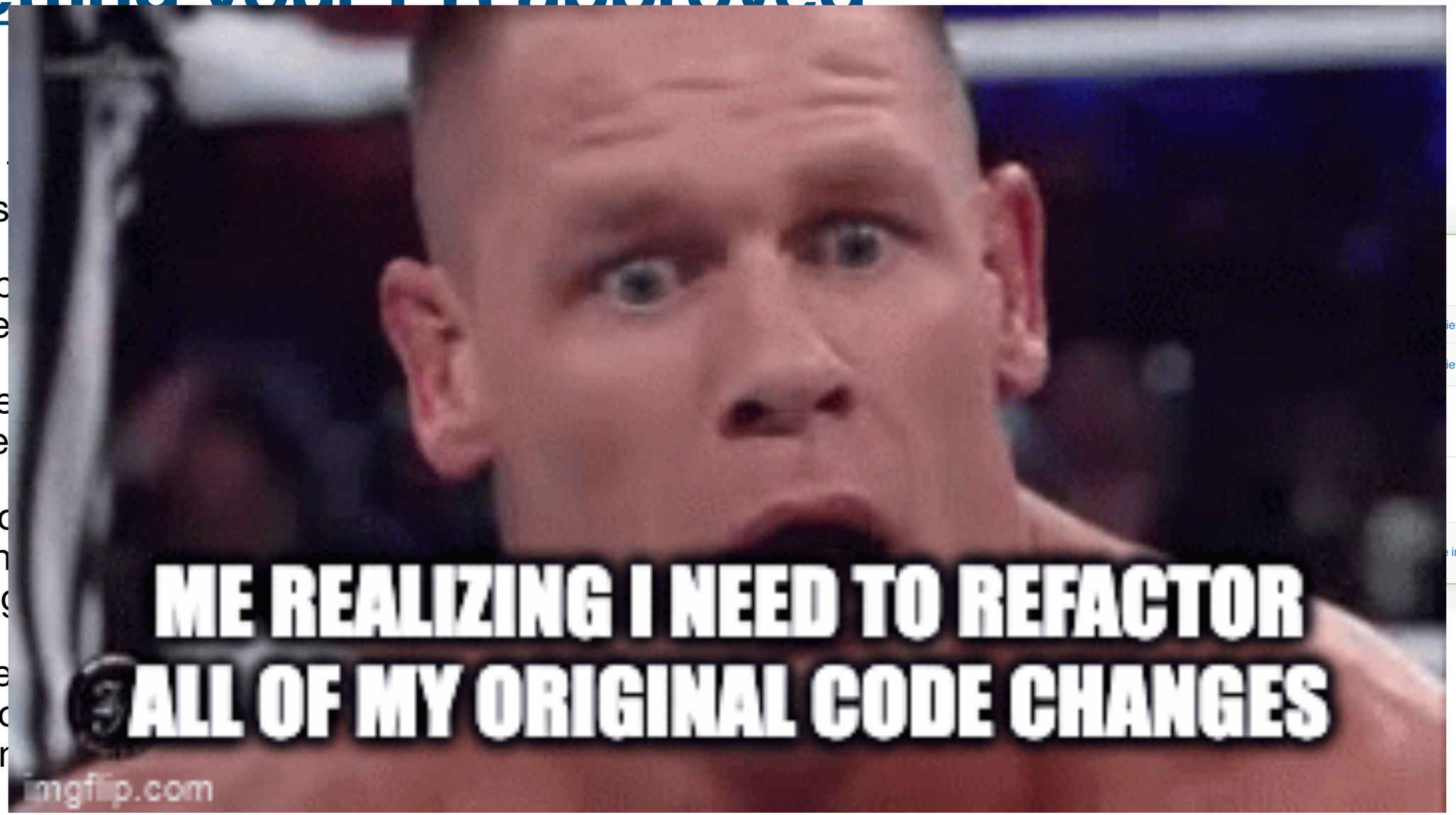
- Try to address review changes as soon as possible.
- Reply to any questions/clarifications or ask questions in a hedging lan
- After addressing requeste “Request Review” feature
- If your PR is lying unreviewed for a long time, don’t hesitate to give a gentle reminder by tagging some maintainers on the PR.
- Keep your eye open for merge conflicts and keep resolving them regularly to keep the review cycle going.

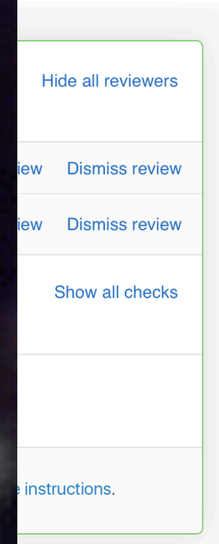
Getting a PR accepted after a long review cycle is one of the best feelings. Trust me.



Getting your PR approved

- Try
pos
- Rep
que
- Afte
“Re
- If yo
don
tagg
- Kee
resc
goir






WHEN YOUR PR
GETS ACCEPTED
imgflip.com

Try contributing to OSS for once !
You will not regret.

Thank You !

Questions or Comments ?

jayjeetc@ucsc.edu

Adding tests

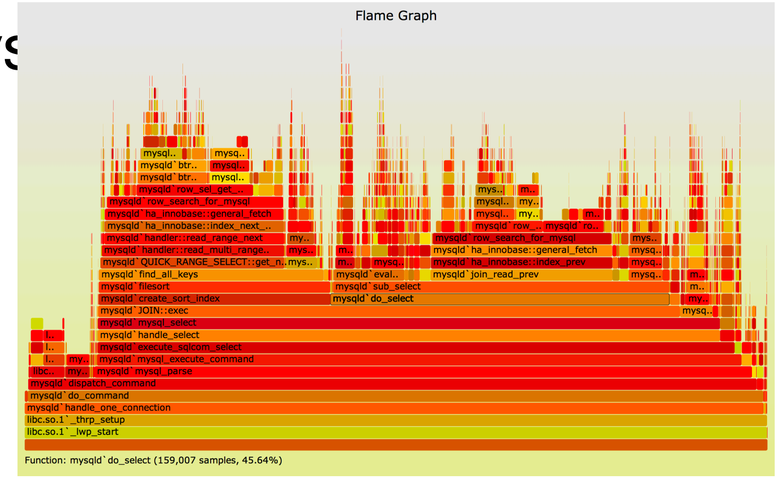
- Integration tests:
  - Test your functionality end-to-end with dummy input data.
  - Start single-node minimal clusters/daemons in CI when needed. Make sure starting services in CI does not fail abruptly.
    - Maybe, introduce some sleep between different daemons start.
  - Try to capture some frequently used scenarios in the tests.
  - No such tools required, mostly bash scripting and UNIX utilities.

Adding tests

- Unit tests:
  - Create a matrix with different kinds of input to a function. Write a test case for each of them.
    - Capture both if and then statements. Same for all the cases of a switch.
  - Use testing frameworks like GTest for C++ or PyTest for Python.
  - While writing unit tests, extract common functionality into utility functions.
  - If possible, use coverage checking tools like CodeCov to check test coverage.

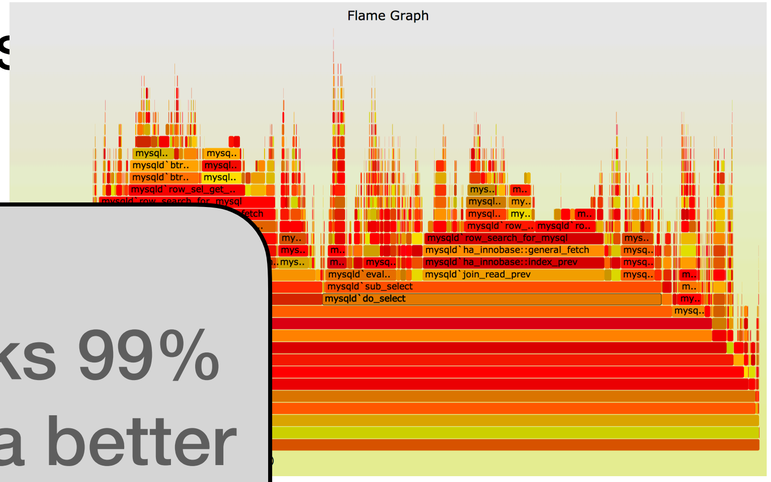
Debugging code

- Print statements generally work great but not always
- Narrow down the suspected code using comments
- Use Logging and error handling in your code.
- Debugging tools:
  - Valgrind, gdb. Helps with memory leaks and SIGSEVs.
  - Profilers and flame graphs. Collects and plots stack traces, helps figure out code paths and relative time taken by each function.



Debugging code

- Print statements generally work great but not always
- Narrow down the suspected code using comments
- Use Logging a
- Debugging too
  - Valgrind, gdb
  - Profilers and flame graphs. Collects and plots stack traces, helps figure out code paths and relative time taken by each function.



Log the heck out. Not only works 99% of the time but leaves you with a better understanding of your code