# Code Review

Alex Gaynor

Please ask questions
as we go!

# About me

- Software Engineer at Rackspace
- Frequent Python open source contributor
  - Django, PyPy, OpenStack, CPython, cryptography.io
- Director of the Python Software Foundation

# What is code review?

```
82  +       sigbuf = self._backend._ffi.new("char[]", max_size)
83  +       siglen_ptr = self._backend._ffi.new("unsigned int[]", 1)
84  +       res = self._backend._lib.ECDSA_sign(
85  +           0,
86  +           digest,
87  +           len(digest),
88  +           sigbuf,
89  +           siglen_ptr,
90  +           ec_key
91  +       )
92  +       assert res == 1
```

0

Write | Preview — Parsed as Markdown — Edit in fullscreen

Hmmm, this is a comment?

Attach images by dragging & dropping, selecting them, or pasting from the clipboard.

Close form — Comment on this line

```
93  +       return self._backend._ffi.buffer(sigbuf)[:siglen_ptr[0]]
```

# People reading other people's code

Not an architecture review

# Why code review?

# Raise the bus factor

Force another person to be familiar enough with the code to pass judgement on it. The more people who understand the code, the more people who'll be able to maintain it in the future.

# Ensure readability

The properties of readable code and writable code are different. Getting someone who has never had to write the code to read the code makes it more likely the next person who has to read it will be able to.

# Catch bugs

“Given enough eyeballs, all bugs are shallow”, “Many hands make light work”. Other people will catch the things you missed.

# Encourage a healthy engineering culture

People need feedback to get better at their jobs. Code review gives you a structure in which to give people feedback. When feedback is irregular, rather than a core part of the job, feedback can be used for criticism, instead of learning and growth.

Everything here works on two conditions. People are acting in good faith, people are committed to not being jerks, and people understand that they are reviewing code and not people. This applies to reviewers and reviews. There are a lot of people who believe that code review requires being a jerk. This is wrong, and this is mean, and I'd much prefer this people just walked away.

# Ground rules for effective code review

# Don’t make people do a machine’s work

People are good at people things, machines are good at machine things. Code review is not running the tests, or checking for style guide violations. Get a CI server to do those. Humans will screw it up because it’s busy work, and machines are better at some types of feedback.

# Everybody gets code reviewed

Code review isn’t something senior engineers do to junior engineers. It’s something everybody does. The person on their first day gets to review the first engineer. No one is above review, no change is above review.

# Do code review before code gets merged

Some people do post-commit code review, meaning code gets landed on master and then it gets reviewed. This promotes a feeling of inevitability, where reviewers don’t want to give what’s perceived as “nit picky” comments because the change is landed already.

Every change.
Every time.

No patch is too small or too simple for code review. I have a 100% rate of patches which are “too small for feedback” breaking the build. This also forces you to have a system where code review is easy, and means you never have an argument about what really is “too small”.

# How to get started

# Get a tool

- Phabricator
- Github
- Gerrit
- Whatever.

Get yourself a tool that tracks the history of a patch and lets you leave inline comments. It doesn't matter what, as long as giving feedback and acting on it is **easy**.

# How to be a good patch author

# Describe what the patch does

Make it easy for a reviewer to know what a patch is supposed to do. They have to be able to verify the code you wrote does what you meant it to do.

# Keep it small

A study has show that beyond 200-400 lines of diff, the efficacy of code review decreases, so keep your patches small. A long series of small patches is much better than a few giant patches.

# Code review is collaborative

You need to work with your reviewer to get the best patch. If you disagree with some feedback, talk to them.

# How to be a good reviewer

# What are you looking for?

# What are you looking for?

- **Intent**

What is the patch supposed to do? Is it really a bug? Do you really want this feature?

# What are you looking for?

- Intent
- **Design**

Is the change in the right place? Did they change some CSS when the HTML was wrong? Did they add javascript to patch the CSS?

# What are you looking for?

- Intent
- Architecture
- **Implementation**

Does the patch do what it says? Does it do it completely? Does it add new bugs? Does it have documentation and tests? This is the bulk of code review.

# What are you looking for?

- Intent
- Architecture
- Implementation
- **Grammar**

This is the small stuff. Does the variable have a good name? Should something be a keyword argument instead of a positional argument?

# What are you looking for?

- Intent
- Architecture
- Implementation
- Grammar

You want to work from intent down to grammar, in order. If you start at grammar, giving feedback on variable name, it’s easy to lose sight of what you’re doing and miss the fact that the patch is totally in the wrong place.

# Types of review elements

# Types of review elements

- **TODO**

These are things that MUST be fixed before a patch can be land, such as a bug or a regression.

# Types of review elements

- TODO
- **Questions**

These are points of clarification. For example “Would this be faster if it was rewritten like so?” or “Can we reuse the logic in the standard library for this?”. Nothing necessarily needs to be done here, it’s just a question for follow up.

# Types of review elements

- TODO
- Questions
- **Suggestions for follow ups**

These are things that the patch author might want to do, but doesn’t have to, and might be done later. For example when adding a new feature you might suggest that a bug be filed to make it be used somewhere else in the code base.

# Anti-patterns

# Very manual processes

Makes doing code review a pain, 5 minutes of thoughtful work become 20 minutes of fighting with details of the CI system or issue tracker. Makes people less likely to want to do code review, which is terrible for an OSS project.

# Irregular

When code review is irregular, what that really means is “the new people have to deal with code review, but the rest of us don’t”. This separates the core developers from the new contributors, and that’s crummy, for one it makes it hard for core devs to care about the experience of new people, they dont’ have that experience! It’s also a bad social dynamic.

Code review is an important part of engineering culture

(It also makes it easier
to ship good code)

# Thanks!

Questions?
@alex_gaynor