Contributing to large open source projects

Who am I?

- Jérôme Petazzoni (@jpetazzo)
- French software engineer living in California
- I put things in containers
- I touched (with a 10-feet pole):
  - the Linux kernel; OpenStack
- I contributed (a tiny bit) to:
  - Docker

(I also maintain and contribute to other open source projects, but they don't count as being "large"!)

Foreword

Why this talk?

More contributors to open source

- Contributing to open source is not easy
- Contributing to large projects can be daunting
- This talk will:
  - encourage you to contribute
  - make your contributions more successful
- Applies to projects large and small!

More diverse contributors

- The most active contributors are those who are paid for it
- Getting such a job requires a track record of contributions*
- To establish this track record, you need:
  - to be working in open source
  - to do it in your spare time
- Who has a lot of spare time?
  - young white men (statistics!)
- Gender gap in open source: 2% women (vs. 20% in tech)
- Let's fix this!

* That's also a problem, by the way

Outline

- "I thought open source was free!"
- How to contribute (code and beyond)
- How to manage big projects
- The One Thing You Should Never Forget

"I thought open source was free!"

"Free" can have many different meanings

- Free beer
- Free speech
- Free puppy

Free beer

- You don't give money to get the software
- Someone still has to make it, though
- You still pay for the distribution medium
  (CD, DVD, internet connection, hosting...)
- Exercise: who gets paid for what when you...

```
apt-get install python-pip
pip install Django
```

- Exercise: what are the chances that someone contributes *your* favorite, unique feature, for free?

Free speech

- You can do whatever you want with the software
- Your use of the software cannot be restricted
- Interestingly, this conflicts with a bunch of US laws
  - e.g. if you write software providing crypto
  - or apparently sometimes if you *use* crypto (???)

Free puppy

- It's given to you for free!
- But you have to take care of it, otherwise it'll die
  (And you'll be a terrible human being)
- Free software is often like a free puppy:
  - you have to set it up yourself
  - you have to maintain it
  - you won't automatically be given the one you want

How to contribute

(code and beyond)

Use the software and report bugs

Sounds obvious, right?

Let's see ...

Be a champion*

Be a champion*

*As in "I demand code review by combat!"

Be a champion*

You are a "technical" person? Great!

If "non-technical" people in your organization complain about some software that you don't use, consider using it and becoming their "champion."

Identify common bugs, and report them upstream.

This is even more important for open source projects when you don't have a support contract!

*As in "I demand code review by combat!"

Be a tester

- Take one for the team!
- Use early (to-be-released) versions
  (release candidates, `master`, `trunk`, ...)
- Find bugs
  (conversely: confirm that it "works for me")
- Report bugs, *so that the released versions are bug-free**
- Case in point: a lot of "dot one" versions
- The best code coverage tool is YOU

*For liberal definition of "bug-free," of course

What makes a *bad* bug report?

- Emailing the user mailing list
- Emailing the maintainers directly
- Emailing the wrong people
  (e.g. of a different project)
- "It doesn't work"
- Rage tweet
- No follow-up

  (when you find the root cause/solution, or realize it was all your fault)

What makes a *good* bug report?

- Use the bug tracker when there is one
- Look for duplicates
- Look for instructions about filing bugs
  (should you use tags? run special commands?)
- Three parts:
  - when I do _____
  - I'm expecting _____ to happen
  - but instead, I'm seeing _____
- Include relevant versions and logs

Note: yes, "relevant" is subjective and therefore hard.

Reproduce bugs

- Look for open bugs in the issue tracker
- Try to reproduce on your setup
- Report status (worked or not?);
  tell which version you are using
- Bonus points if you can test on multiple versions
  (You Da Real MVP!)
- Reproducing older bugs helps to find stuff that has been fixed in newer versions
- Reproducing newer bugs helps to narrow down their possible causes

Triage issues

Note: this is a very special skillset

- Get familiar with the tagging system (when there is one)
- Look for new issues
- Tag issues, to make maintainer's lives easier
  - bugs vs feature requests vs proposals vs PRs ...
  - area, priority, difficulty
- If you see something, say something!
  (i.e. if you see a high priority issue, escalate it)

Improve / fix documentation

- Let's face it: most devs suck at writing docs
  (sometimes they don't even try)
- If there are no docs: take notes as you progress
  (they will be super helpful to the person after you)
- If the docs are incomplete/outdated: update them
- When you spend time writing/updating docs,
  you help a developer to work on code instead
- As a project gets larger and more complex,
  the gap between devs and users gets wider
- Candid eyes (yours!) are needed to help bridge that gap

Working on code

- Contributors ≠ maintainers
- Contributors write code
- Maintainers *review* that code,
  and vet it for inclusion (merge)
- Why don't we merge everything blindly?
  - quality would get horrible very quickly
  - bugs are easier to catch when looking at a small change
  - you can't always test everything automatically,
    so Alice's change might break Barbara's code

So, what does the process look like?

Contributing 101

- When possible, look for an easy target
  (ask for guidance if you're unsure)
- Make changes
- Submit them
  (patches, pull requests...)
- Wait for feedback
  (be patient!)
- Address concerns voiced by maintainers
- Repeat until merged
- ... or abandoned (not all contributions get merged)

How it works in big open source projects

Benevolent Dictator For Life (BDFL)

- Reviews and approves everything
- Fine for smaller projects
  - ensures consistency
  - great if the BDFL has an outstanding vision
- Doesn't scale
- Eventually, you need "governance" and "rules" (stating who does what, how things get in...)

Three examples

- Linux kernel
- OpenStack
- Docker

The Linux kernel

The Linux kernel in numbers

- ~1,200 *companies* contributing (across all releases)
- ~12,000 *individual contributors* (across all releases)
- ~200(?) *companies* contributing (recent releases)
- ~1,500 *individual contributors* (recent releases)

Note: 1 release = ~2 months, or ~10,000 patches
(Also: ~185 commits/day!)

Source: "Who Writes Linux" by Linux Foundation, Feb.2015

The Linux kernel contribution process

- Decentralized process with subsystem maintainers
- *Relatively* little stuff goes directly to Linus or Greg KH
- Historical workflow: patches sent over LKML
  (and, often, on per-topic mailing lists)
- Still the case today, but git makes things easier

OpenStack

OpenStack in numbers

- 250+ *companies* contributing (across all releases)
- 4,300+ *individual contributors* (across all releases)
- 150+ *companies* contributing (last release)
- 1,700+ *individual contributors* (last release)

Note: 1 release = 6 months

Source: stackalytics.com

OpenStack contribution process

- Git + Gerrit
- Gerrit enforces the workflow
- Write a blueprint* (for new features)
- Push your branch and submit it for review
  (it should mention the blueprint or bug number)

*Spec sheet; doesn't always have to be complex

OpenStack review process

- Jenkins will run the "check" tests on your code (and assign a "ok/fail" score)
- Reviewers will vote +1/0/-1 your changes
- Core reviewers can vote -2/+2
- To be merged, your code needs +2 +2
- Code cannot be merged if it has a -2
- Before merging, Jenkins will run the "gate" tests (more complex tests) (and they have to pass)

Scaling OpenStack contributions

- Independant projects (Nova, Neutron, Cinder, ...)
- Avoids slowdowns due to lockstep
- But integration and coherence suffers
- Big community, very fragmented
- "It's very hard to follow everything"

Docker

Docker in numbers

- ~1,100 contributors (across all releases)
- 150~200 contributors (last few releases)
- 100-150 pull requests per week

Note: 1 release = ~2 months

Source: manually running gitdm

Docker contribution process

- Git + GitHub
- Extensive (ab)use of GitHub labels

  (also: Gordon, an open source bot to help with the workflow)
- Each proposal/change/fix is materialized by a pull request
- Imagine a kanban board; 1 PR = 1 post-it, with:
  - pending design review (skipped for bug fixes)
  - pending code review (skipped for docs changes)
  - pending doc review
  - merged (victory!)
- "Fail fast" philosophy

  (don't write or review docs for a feature whose design hasn't even been accepted)

Docker review process

- Maintainers and contributors have very different roles and responsibilities
- Your PR needs a "LGTM*" from 2 maintainers to proceed
- Maintainers' changes follow the rules too (no "superpowers")
- No official veto, but if one maintainer says "hold," others will generally follow suit

Note: changing the process = PR against `CONTRIBUTING.md`

*Looks Good To Me

Docker guidelines

- Check the public roadmap to make sure that your change doesn't oppose future changes
- Talk to people on IRC (`#docker` and `#docker-dev`)
- 90% of PRs are merged (or rejected) within a month
- Pets vs cattle analogy applies too ☺

Docker guidelines

- Check the public roadmap to make sure that your change doesn't oppose future changes
- Talk to people on IRC (`#docker` and `#docker-dev`)
- 90% of PRs are merged (or rejected) within a month
- Pets vs cattle analogy applies too ☺
  - as a new contributor, your 1st PR is Your Precious
  - for the maintainers reviewing it, it's the 10th today
- Sometimes, maintainers will carry (=champion) your PR

What we learned ...

- GitHub is much friendlier than a plain mailing list
- But the workflow (PR/merge by maintainer) is too basic
- It doesn't scale (from an organizational POV)
- You need to implement your own workflow manually
- See:
  - Gordon (github.com/docker/gordon)
  - Leeroy (github.com/docker/leeroy)
- GitHub activity dashboard is great,
  but it doesn't show maintainer activity
  (which stinks, because it's already unrewarding)

About DCOs, CLAs ...

- Developer's Certificate of Origin (kernel, Docker)
- Contributors License Agreement (Open Stack)
- Legal safety blanket (?)
- TL,DR: "Lawyers got involved"

> *No one really knows why the CLA requirement was included in the launch of OpenStack in July 2010. The only reason anyone can think of is that maybe it provides some additional protection somehow. It's kind of like the requirement to put your phones in airplane mode during takeoff: no one can really explain why it's necessary, but at least it can't hurt, right?*

(At least, that's how @s0ulshake put it after 10 mins of googling & 0 years of law school)

The One Thing You Should Never Forget

We are all human beings*

We are all human beings*

*Except our Reptilian Overlords

We are all human beings

- Users
- Bug reporters
- Bug reproducers
- Triagers
- Doc writers
- Proofreaders
- Code contributors
- Reviewers
- Maintainers
- etc.

Human beings have basic needs

- Remember Maslow's pyramid?
- We need food, shelter
- Those things cost money
- Software doesn't happen out of thin air

How can I support open source development?

- Support independent developers:
  hire them for contract work
- Support open source companies:
  buy licenses/support
- Contribute rather than reinvent the wheel
- Open source by default / closed source when sensitive
  (vs. closed source by default / open source when it suits your agenda)
- Tell the authors about your use-cases
- ... And do all the things we mentioned earlier
  (use, test, report/reproduce bugs, improve docs, contribute code...)

Assume best intents

- Computers are evil, right
- All software sucks, and we're doomed
- Yet, we all work very hard to make it better
- Screaming "IT DOESN'T WORK, DAMMIT!" won't help
- We did not *deliberately* break your favorite feature
- Case in point: Docker on CentOS 6

Human beings have feelings

- When you vocally attack a project,
  this is what its creators hear:

> *Your offspring is the worst; it's ugly and smells like rotten cabbage, please never reproduce again.*
> *That being said, can you teach it to use a lawnmower so it can mow my lawn?*

(Loosely based on a much better explanation by @robynbergeron)

- This doesn't mean that you can't criticize

Human beings are human beings

- Treat them like you would treat your next of kin
- Threats, ad hominem attacks and harrassment are **bad**

Human beings are human beings

- Treat them like you would treat your next of kin
- Threats, ad hominem attacks and harrassment are **bad**
  - wait, that's obvious, right?

Human beings are human beings

- Treat them like you would treat your next of kin
- Threats, ad hominem attacks and harrassment are **bad**
  - wait, that's obvious, right?
  - yup, which is why it needs to stop
  - especially stop targeting women, ffs

Human beings are human beings

- Treat them like you would treat your next of kin
- Threats, ad hominem attacks and harrassment are **bad**
  - wait, that's obvious, right?
  - yup, which is why it needs to stop
  - especially stop targeting women, ffs
- That's why we have codes of conduct, by the way

Be kind

- When somebody hands out free puppies,
  who gets the cutest one?
- Be cooperative
  - follow contributing rules
  - listen to maintainer feedback
- Maintainers will often carry/champion your work

Thanks!
Questions?

@icecrime
@jpetazzo
@docker