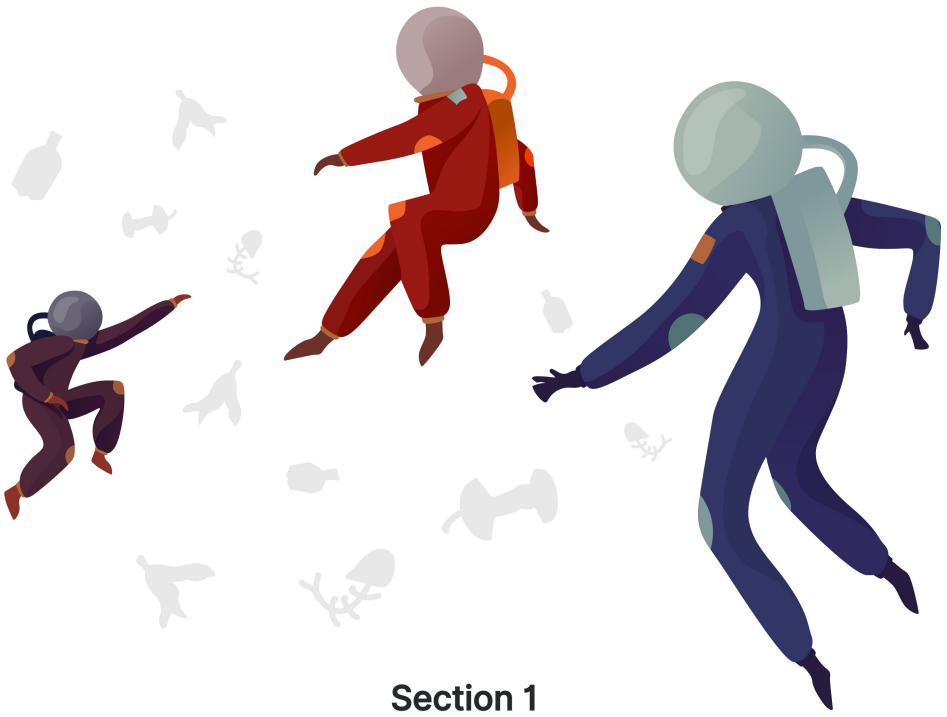
How to Contribute to Open Source

Want to contribute to open source? A guide to making open source contributions, for first-timers and for veterans.

Table of Contents



Section 1

Why contribute to open source?

Working on [freenode] helped me earn many of the skills I later used for my studies in university and my actual job. I think working on open source projects helps me as much as it

> helps the project!
>
> — @errietta, "Why I love contributing to open source software"

Contributing to open source can be a rewarding way to learn, teach, and build experience in just about any skill you can imagine.

Why do people contribute to open source? Plenty of reasons!

Improve software you rely on

Lots of open source contributors start by being users of software they contribute to. When you find a bug in an open source software you use, you may want to look at the source to see if you can patch it yourself. If that's the case, then contributing the patch back is the best way to ensure that your friends (and yourself when you update to the next release) will be able to benefit from it.

Improve existing skills

Whether it's coding, user interface design, graphic design, writing, or organizing, if you're looking for practice, there's a task for you on an open source project.

Meet people who are interested in similar things

Open source projects with warm, welcoming communities keep people coming back for years. Many people form lifelong friendships through their participation in open source, whether it's running into each other at conferences or late night online chats about burritos.

Find mentors and teach others

Working with others on a shared project means you'll have to explain how you do things, as well as ask other people for help. The acts of learning and teaching can be a fulfilling activity for everyone involved.

Build public artifacts that help you grow a reputation (and a career)

By definition, all of your open source work is public, which means you get free examples to take anywhere as a demonstration of what you can do.

Learn people skills

Open source offers opportunities to practice leadership and management skills, such as resolving conflicts, organizing teams of people, and prioritizing work.

It's empowering to be able to make changes, even small ones

You don't have to become a lifelong contributor to enjoy participating in open source. Have you ever seen a typo on a website, and wished someone would fix it? On an open source project, you can do just that. Open source helps people feel agency over their lives and how they experience the world, and that in itself is gratifying.

Section 2

What it means to contribute

If you're a new open source contributor, the process can be intimidating. How do you find the right project? What if you don't know how to code? What if something goes wrong?

Not to worry! There are all sorts of ways to get involved with an open source project, and a few tips will help you get the most out of your experience.

You don't have to contribute code

A common misconception about contributing to open source is that you need to contribute code. In fact, it's often the other parts of a project that are most neglected or overlooked. You'll do the project a *huge* favor by offering to pitch in with these types of contributions!

I've been renowned for my work on CocoaPods, but most people don't know that I actually don't do any real work on the CocoaPods tool itself. My time on the project is mostly spent doing things like documentation and working on branding.

— @orta, "Moving to OSS by default"

Even if you like to write code, other types of contributions are a great way to get involved with a project and meet other community members. Building those relationships will give you opportunities to work on other parts of the project.

Do you like planning events?

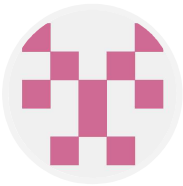
- Organize workshops or meetups about the project, like @fzamperin did for NodeSchool
- Organize the project's conference (if they have one)
- Help community members find the right conferences and submit proposals for speaking

Do you like to design?

- Restructure layouts to improve the project's usability
- Conduct user research to reorganize and refine the project's navigation or menus, like Drupal suggests
- Put together a style guide to help the project have a consistent visual design
- Create art for t-shirts or a new logo, like hapi.js's contributors did

Do you like to write?

- Write and improve the project's documentation
- Curate a folder of examples showing how the project is used
- Start a newsletter for the project, or curate highlights from the mailing list
- Write tutorials for the project, like PyPA's contributors did
- Write a translation for the project's documentation



Seriously, [documentation] is mega-important. The documentation so far has been great and has been a killer feature of Babel. There are sections that could certainly use some work and even the addition of a paragraph here or there is extremely appreciated.

— @kittens, "Call for contributors"

Do you like organizing?

- Link to duplicate issues, and suggest new issue labels, to keep things organized
- Go through open issues and suggest closing old ones, like @nzakas did for ESLint
- Ask clarifying questions on recently opened issues to move the discussion forward

Do you like to code?

- Find an open issue to tackle, like @dianjin did for Leaflet
- Ask if you can help write a new feature
- Automate project setup
- Improve tooling and testing

Do you like helping people?

- Answer questions about the project on e.g., Stack Overflow (like this Postgres example) or Reddit
- Answer questions for people on open issues
- Help moderate the discussion boards or conversation channels

Do you like helping others code?

- Review code on other people's submissions
- Write tutorials for how a project can be used
- Offer to mentor another contributor, like @ereichert did for @bronzdoc on Rust

You don't just have to work on software projects!

While "open source" often refers to software, you can collaborate on just about anything. There are books, recipes, lists, and classes that get developed as open source projects.

For example:

- @sindresorhus curates a list of "awesome" lists
- @h5bp maintains a list of potential interview questions for front-end developer candidates

- @stuartlynn and @nicole-a-tesla made a collection of fun facts about puffins

Even if you're a software developer, working on a documentation project can help you get started in open source. It's often less intimidating to work on projects that don't involve code, and the process of collaboration will build your confidence and experience.

Section 3

Orienting yourself to a new project

If you go to an issue tracker and things seem confusing, it's not just you. These tools require a lot of implicit knowledge, but people can help you navigate it and you can ask them questions.

— @shaunagm, "How to Contribute to Open Source"

For anything more than a typo fix, contributing to open source is like walking up to a group of strangers at a party. If you start talking about llamas, while they were deep in a discussion about goldfish, they'll probably look at you a little strangely.

Before jumping in blindly with your own suggestions, start by learning how to read the room. Doing so increases the chances that your ideas will be noticed and heard.

Anatomy of an open source project

Every open source community is different.

Spending years on one open source project means you've gotten to know one open source project. Move to a different project, and you might find the vocabulary, norms, and communication styles are completely different.

That said, many open source projects follow a similar organizational structure. Understanding the different community roles and overall process will help you get quickly oriented to any new project.

A typical open source project has the following types of people:

- **Author:** The person/s or organization that created the project

- **Owner:** The person/s who has administrative ownership over the organization or repository (not always the same as the original author)
- **Maintainers:** Contributors who are responsible for driving the vision and managing the organizational aspects of the project (They may also be authors or owners of the project.)
- **Contributors:** Everyone who has contributed something back to the project
- **Community Members:** People who use the project. They might be active in conversations or express their opinion on the project's direction

Bigger projects may also have subcommittees or working groups focused on different tasks, such as tooling, triage, community moderation, and event organizing. Look on a project's website for a "team" page, or in the repository for governance documentation, to find this information.

A project also has documentation. These files are usually listed in the top level of a repository.

- **LICENSE:** By definition, every open source project must have an open source license. If the project does not have a license, it is not open source.
- **README:** The README is the instruction manual that welcomes new community members to the project. It explains why the project is useful and how to get started.
- **CONTRIBUTING:** Whereas READMEs help people *use* the project, contributing docs help people *contribute* to the project. It explains what types of contributions are needed and how the process works. While not every project has a CONTRIBUTING file, its presence signals that this is a welcoming project to contribute to.
- **CODE_OF_CONDUCT:** The code of conduct sets ground rules for participants' behavior associated and helps to facilitate a friendly, welcoming environment. While not every project has a CODE_OF_CONDUCT file, its presence signals that this is a welcoming project to contribute to.
- **Other documentation:** There might be additional documentation, such as tutorials, walkthroughs, or governance policies, especially on bigger projects.

Finally, open source projects use the following tools to organize discussion. Reading through the archives will give you a good picture of how the community thinks and works.

- **Issue tracker:** Where people discuss issues related to the project.
- **Pull requests:** Where people discuss and review changes that are in progress.
- **Discussion forums or mailing lists:** Some projects may use these channels for conversational topics (for example, *"How do I..."* or *"What do you think about..."* instead of bug reports or feature requests). Others use the issue tracker for all conversations.

- **Synchronous chat channel:** Some projects use chat channels (such as Slack or IRC) for casual conversation, collaboration, and quick exchanges.

Section 4

Finding a project to contribute to

Now that you've figured out how open source projects work, it's time to find a project to contribute to!

If you've never contributed to open source before, take some advice from U.S. President John F. Kennedy, who once said, *"Ask not what your country can do for you - ask what you can do for your country."*

Contributing to open source happens at all levels, across projects. You don't need to overthink what exactly your first contribution will be, or how it will look.

Instead, start by thinking about the projects you already use, or want to use. The projects you'll actively contribute to are the ones you find yourself coming back to.

Within those projects, whenever you catch yourself thinking that something could be better or different, act on your instinct.

Open source isn't an exclusive club; it's made by people just like you. "Open source" is just a fancy term for treating the world's problems as fixable.

You might scan a README and find a broken link or a typo. Or you're a new user and you noticed something is broken, or an issue that you think should really be in the documentation. Instead of ignoring it and moving on, or asking someone else to fix it, see whether you can help out by pitching in. That's what open source is all about!

> 28% of casual contributions to open source are documentation, such as a typo fix, reformatting, or writing a translation.

If you're looking for existing issues you can fix, every open source project has a `/contribute` page that highlights beginner-friendly issues you can start out with. Navigate to the main page of the repository on GitHub, and add `/contribute` at the end of the URL (for example `https://github.com/facebook/react/contribute`).

You can also use one of the following resources to help you discover and contribute to new projects:

- GitHub Explore
- Open Source Friday
- First Timers Only
- CodeTriage
- 24 Pull Requests
- Up For Grabs
- Contributor-ninja
- First Contributions
- SourceSort

A checklist before you contribute

When you've found a project you'd like to contribute to, do a quick scan to make sure that the project is suitable for accepting contributions. Otherwise, your hard work may never get a response.

Here's a handy checklist to evaluate whether a project is good for new contributors.

Meets the definition of open source

- [ ] Does it have a license? Usually, there is a file called LICENSE in the root of the repository.

Project actively accepts contributions

Look at the commit activity on the main branch. On GitHub, you can see this information on a repository's homepage.

- [ ] When was the latest commit?
- [ ] How many contributors does the project have?
- [ ] How often do people commit? (On GitHub, you can find this by clicking "Commits" in the top bar.)

Next, look at the project's issues.

- [ ] How many open issues are there?
- [ ] Do maintainers respond quickly to issues when they are opened?
- [ ] Is there active discussion on the issues?
- [ ] Are the issues recent?
- [ ] Are issues getting closed? (On GitHub, click the "closed" tab on the Issues page to see closed issues.)

Now do the same for the project's pull requests.

- [ ] How many open pull requests are there?
- [ ] Do maintainers respond quickly to pull requests when they are opened?
- [ ] Is there active discussion on the pull requests?
- [ ] Are the pull requests recent?
- [ ] How recently were any pull requests merged? (On GitHub, click the "closed" tab on the Pull Requests page to see closed PRs.)

Project is welcoming

A project that is friendly and welcoming signals that they will be receptive to new contributors.

- [ ] Do the maintainers respond helpfully to questions in issues?
- [ ] Are people friendly in the issues, discussion forum, and chat (for example, IRC or Slack)?
- [ ] Do pull requests get reviewed?
- [ ] Do maintainers thank people for their contributions?

Whenever you see a long thread, spot check responses from core developers coming late in the thread. Are they summarizing constructively, and taking steps to bring the thread to a decision while remaining polite? If you see a lot of flame wars going on, that's often a sign that energy is going into argument instead of into development.

— @kfogel, *Producing OSS*

Section 5

How to submit a contribution

You've found a project you like, and you're ready to make a contribution. Finally! Here's how to get your contribution in the right way.

Communicating effectively

Whether you're a one-time contributor or trying to join a community, working with others is one of the most important skills you'll develop in open source.

> [As a new contributor,] I quickly realized I had to ask questions if I wanted to be able to close the issue. I skimmed through the code base. Once I had some sense of what was going on, I asked for more direction. And voilà! I was able to solve the issue after getting all the relevant details I needed.
>
> — @shubheksha, A Beginner's Very Bumpy Journey Through The World of Open Source

Before you open an issue or pull request, or ask a question in chat, keep these points in mind to help your ideas come across effectively.

Give context. Help others get quickly up to speed. If you're running into an error, explain what you're trying to do and how to reproduce it. If you're suggesting a new idea, explain why you think it'd be useful to the project (not just to you!).

> 😇 *"X doesn't happen when I do Y"*
>
> 😢 *"X is broken! Please fix it."*

Do your homework beforehand. It's OK not to know things, but show that you tried. Before asking for help, be sure to check a project's README, documentation, issues (open or closed), mailing list, and search the internet for an answer. People will appreciate it when you demonstrate that you're trying to learn.

> 😇 *"I'm not sure how to implement X. I checked the help docs and didn't find any mentions."*
>
> 😢 *"How do I X?"*

Keep requests short and direct. Much like sending an email, every contribution, no matter how simple or helpful, requires someone else's review. Many projects have more incoming requests than people available to help. Be concise. You will increase the chance that someone will be able to help you.

> 😇 *"I'd like to write an API tutorial."*

> 😢 *"I was driving down the highway the other day and stopped for gas, and then I had this amazing idea for something we should be doing, but before I explain that, let me show you..."*

Keep all communication public. Although it's tempting, don't reach out to maintainers privately unless you need to share sensitive information (such as a security issue or serious conduct violation). When you keep the conversation public, more people can learn and benefit from your exchange. Discussions can be, in themselves, contributions.

> 😇 *(as a comment) "@-maintainer Hi there! How should we proceed on this PR?"*
>
> 😢 *(as an email) "Hey there, sorry to bother you over email, but I was wondering if you've had a chance to review my PR"*

It's okay to ask questions (but be patient!). Everybody was new to the project at some point, and even experienced contributors need to get up to speed when they look at a new project. By the same token, even longtime maintainers are not always familiar with every part of the project. Show them the same patience that you'd want them to show to you.

> 😇 *"Thanks for looking into this error. I followed your suggestions. Here's the output."*
>
> 😢 *"Why can't you fix my problem? Isn't this your project?"*

Respect community decisions. Your ideas may differ from the community's priorities or vision. They may offer feedback or decide not to pursue your idea. While you should discuss and look for compromise, maintainers have to live with your decision longer than you will. If you disagree with their direction, you can always work on your own fork or start your own project.

> 😇 *"I'm disappointed you can't support my use case, but as you've explained it only affects a minor portion of users, I understand why. Thanks for listening."*
>
> 😢 *"Why won't you support my use case? This is unacceptable!"*

Above all, keep it classy. Open source is made up of collaborators from all over the world. Context gets lost across languages, cultures, geographies, and time zones. In addition, written communication makes it harder to convey a tone or mood. Assume good intentions in these conversations. It's fine to politely push back on an idea, ask for more context, or further clarify your position. Just try to leave the internet a better place than when you found it.

Gathering context

Before doing anything, do a quick check to make sure your idea hasn't been discussed elsewhere. Skim the project's README, issues (open and closed), mailing list, and Stack Overflow. You don't have to spend hours going through everything, but a quick search for a few key terms goes a long way.

If you can't find your idea elsewhere, you're ready to make a move. If the project is on GitHub, you'll likely communicate by opening an issue or pull request:

- **Issues** are like starting a conversation or discussion
- **Pull requests** are for starting work on a solution
- **For lightweight communication,** such as a clarifying or how-to question, try asking on Stack Overflow, IRC, Slack, or other chat channels, if the project has one

Before you open an issue or pull request, check the project's contributing docs (usually a file called CONTRIBUTING, or in the README), to see whether you need to include anything specific. For example, they may ask that you follow a template, or require that you use tests.

If you want to make a substantial contribution, open an issue to ask before working on it. It's helpful to watch the project for a while (on GitHub, you can click "Watch" to be notified of all conversations), and get to know community members, before doing work that might not get accepted.

> You'll learn *a lot* from taking a single project you actively use, "watching" it on GitHub and reading every issue and PR.
>
> — @gaearon on joining projects

Opening an issue

You should usually open an issue in the following situations:

- Report an error you can't solve yourself
- Discuss a high-level topic or idea (for example, community, vision or policies)
- Propose a new feature or other project idea

Tips for communicating on issues:

- **If you see an open issue that you want to tackle,** comment on the issue to let people know you're on it. That way, people are less likely to duplicate your work.
- **If an issue was opened a while ago,** it's possible that it's being addressed somewhere else, or has already been resolved, so comment to ask for confirmation before starting work.

- **If you opened an issue, but figured out the answer later on your own,** comment on the issue to let people know, then close the issue. Even documenting that outcome is a contribution to the project.

Opening a pull request

You should usually open a pull request in the following situations:

- Submit trivial fixes (for example, a typo, a broken link or an obvious error)
- Start work on a contribution that was already asked for, or that you've already discussed, in an issue

A pull request doesn't have to represent finished work. It's usually better to open a pull request early on, so others can watch or give feedback on your progress. Just open it as a "draft" or mark as a "WIP" (Work in Progress) in the subject line. You can always add more commits later.

If the project is on GitHub, here's how to submit a pull request:

- **Fork the repository** and clone it locally. Connect your local to the original "upstream" repository by adding it as a remote. Pull in changes from "upstream" often so that you stay up to date so that when you submit your pull request, merge conflicts will be less likely. (See more detailed instructions here.)
- **Create a branch** for your edits.
- **Reference any relevant issues** or supporting documentation in your PR (for example, "Closes #37.")
- **Include screenshots of the before and after** if your changes include differences in HTML/CSS. Drag and drop the images into the body of your pull request.
- **Test your changes!** Run your changes against any existing tests if they exist and create new ones when needed. Whether tests exist or not, make sure your changes don't break the existing project.
- **Contribute in the style of the project** to the best of your abilities. This may mean using indents, semi-colons or comments differently than you would in your own repository, but makes it easier for the maintainer to merge, others to understand and maintain in the future.

If this is your first pull request, check out Make a Pull Request, which @kentcdodds created as a walkthrough video tutorial. You can also practice making a pull request in the First Contributions repository, created by @Roshanjossey.

Section 6

What happens after you submit a contribution

You did it! Congratulations on becoming an open source contributor. We hope it's the first of many.

After you submit a contribution, one of the following will happen:

😭 You don't get a response.

Hopefully you checked the project for signs of activity before making a contribution. Even on an active project, however, it's possible that your contribution won't get a response.

If you haven't gotten a response in over a week, it's fair to politely respond in that same thread, asking someone for a review. If you know the name of the right person to review your contribution, you can @-mention them in that thread.

Don't reach out to that person privately; remember that public communication is vital to open source projects.

If you make a polite bump and still nobody responds, it's possible that nobody will respond, ever. It's not a great feeling, but don't let that discourage you. It's happened to everyone! There are many possible reasons why you didn't get a response, including personal circumstances that may be out of your control. Try to find another project or way to contribute. If anything, this is a good reason not to invest too much time in making a contribution before other community members are engaged and responsive.

🚧 Someone requests changes to your contribution.

It's common that you'll be asked to make changes to your contribution, whether that's feedback on the scope of your idea, or changes to your code.

When someone requests changes, be responsive. They've taken the time to review your contribution. Opening a PR and walking away is bad form. If you don't know how to make changes, research the problem, then ask for help if you need it.

If you don't have time to work on the issue anymore (for example, if the conversation has been going on for months, and your circumstances have changed), let the maintainer know so they're not expecting a response. Someone else may be happy to take over.

👎 Your contribution doesn't get accepted.

Your contribution may or may not be accepted in the end. Hopefully you didn't put too much work into it already. If you're not sure why it wasn't accepted, it's perfectly reasonable to ask the maintainer for feedback and clarification. Ultimately, however, you'll need to respect that this is their decision. Don't argue or get hostile. You're always welcome to fork and work on your own version if you disagree!

🎉 Your contribution gets accepted.

Hooray! You've successfully made an open source contribution!

Section 7

You did it!

Whether you just made your first open source contribution, or you're looking for new ways to contribute, we hope you're inspired to take action. Even if your contribution wasn't accepted, don't forget to say thanks when a maintainer put effort into helping you. Open source is made by people like you: one issue, pull request, comment, or high-five at a time.

Back to all guides

Related Guides

Building Welcoming Communities

Building a community that encourages people to use, contribute to, and evangelize your project.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Subscribe

fine print


with
by
and friends

Starting an Open Source Project

Learn more about the world of open source and get ready to launch your own project.

Table of Contents

Section 1

The “what” and “why” of open source

So you’re thinking about getting started with open source? Congratulations! The world appreciates your contribution. Let’s talk about what open source is and why people do it.

What does “open source” mean?

When a project is open source, that means **anybody is free to use, study, modify, and distribute your project for any purpose.** These permissions are enforced through an open source license.

Open source is powerful because it lowers the barriers to adoption and collaboration, allowing people to spread and improve projects quickly. Also because it gives users a potential to control their own computing, relative to closed source. For example, a business using open source software has the option to hire someone to make custom improvements to the software, rather than relying exclusively on a closed source vendor’s product decisions.

Free software refers to the same set of projects as *open source*. Sometimes you’ll also see these terms combined as “free and open source software” (FOSS) or “free, libre, and open source software” (FLOSS). *Free* and *libre* refer to freedom, not price.

Why do people open source their work?

> One of the most rewarding experiences I get out of using and collaborating on open source comes from the relationships that I build with other developers facing many of the same problems I am.
>
> — @kentcdodds, “How getting into Open Source has been awesome for me”

There are many reasons why a person or organization would want to open source a project. Some examples include:

- **Collaboration:** Open source projects can accept changes from anybody in the world. Exercism, for example, is a programming exercise platform with over 350 contributors.
- **Adoption and remixing:** Open source projects can be used by anyone for nearly any purpose. People can even use it to build other things. WordPress, for example, started as a fork of an existing project called b2.
- **Transparency:** Anyone can inspect an open source project for errors or inconsistencies. Transparency matters to governments like Bulgaria or the United States, regulated industries like banking or healthcare, and security software like Let’s Encrypt.

Open source isn't just for software, either. You can open source everything from data sets to books. Check out GitHub Explore for ideas on what else you can open source.

Does open source mean "free of charge"?

One of open source's biggest draws is that it does not cost money. "Free of charge", however, is a byproduct of open source's overall value.

Because an open source license requires that anyone can use, modify, and share your project for nearly any purpose, projects themselves tend to be free of charge. If the project cost money to use, anyone could legally make a copy and use the free version instead.

As a result, most open source projects are free, but "free of charge" is not part of the open source definition. There are ways to charge for open source projects indirectly through dual licensing or limited features, while still complying with the official definition of open source.

Section 2

Should I launch my own open source project?

The short answer is yes, because no matter the outcome, launching your own project is a great way to learn how open source works.

If you've never open sourced a project before, you might be nervous about what people will say, or whether anyone will notice at all. If this sounds like you, you're not alone!

Open source work is like any other creative activity, whether it's writing or painting. It can feel scary to share your work with the world, but the only way to get better is to practice - even if you don't have an audience.

If you're not yet convinced, take a moment to think about what your goals might be.

Setting your goals

Goals can help you figure out what to work on, what to say no to, and where you need help from others. Start by asking yourself, *why am I open sourcing this project?*

There is no one right answer to this question. You may have multiple goals for a single project, or different projects with different goals.

If your only goal is to show off your work, you may not even want contributions, and even say so in your README. On the other hand, if you do want contributors, you'll invest time into clear documentation and making newcomers feel welcome.

At some point I created a custom UIAlertView that I was using...and I decided to make it open source. So I modified it to be more dynamic and uploaded it to GitHub. I also wrote my first documentation explaining to other developers how to use it on their projects. Probably nobody ever used it because it was a simple project but I was feeling good about my contribution.

— @mavris, "Self-taught Software Developers: Why Open Source is important to us"

As your project grows, your community may need more than just code from you. Responding to issues, reviewing code, and evangelizing your project are all important tasks in an open source project.

While the amount of time you spend on non-coding tasks will depend on the size and scope of your project, you should be prepared as a maintainer to address them yourself or find someone to help you.

If you're part of a company open sourcing a project, make sure your project has the internal resources it needs to thrive. You'll want to identify who's responsible for maintaining the project after launch, and how you'll share those tasks with your community.

If you need a dedicated budget or staffing for promotion, operations and maintaining the project, start those conversations early.

As you begin to open source the project, it's important to make sure that your management processes take into consideration the contributions and abilities of the community around your project. Don't be afraid to involve contributors who are not employed in your business in key aspects of the project — especially if they are frequent contributors.

— @captainsafia, "So you wanna open source a project, eh?"

Contributing to other projects

If your goal is to learn how to collaborate with others or understand how open source works, consider contributing to an existing project. Start with a project that you already use and love. Contributing to a project can be as simple as fixing typos or updating documentation.

If you're not sure how to get started as a contributor, check out our How to Contribute to Open Source guide.

Section 3

Launching your own open source project

There is no perfect time to open source your work. You can open source an idea, a work in progress, or after years of being closed source.

Generally speaking, you should open source your project when you feel comfortable having others view, and give feedback on, your work.

No matter which stage you decide to open source your project, every project should include the following documentation:

- Open source license
- README
- Contributing guidelines
- Code of conduct

As a maintainer, these components will help you communicate expectations, manage contributions, and protect everyone's legal rights (including your own). They significantly increase your chances of having a positive experience.

If your project is on GitHub, putting these files in your root directory with the recommended filenames will help GitHub recognize and automatically surface them to your readers.

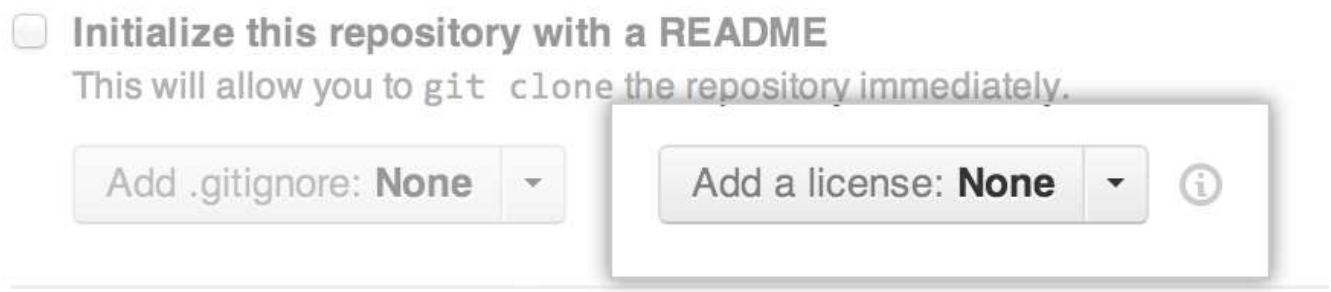
Choosing a license

An open source license guarantees that others can use, copy, modify, and contribute back to your project without repercussions. It also protects you from sticky legal situations. **You must include a license when you launch an open source project.**

Legal work is no fun. The good news is that you can copy and paste an existing license into your repository. It will only take a minute to protect your hard work.

MIT, Apache 2.0, and GPLv3 are the most popular open source licenses, but there are other options to choose from.

When you create a new project on GitHub, you are given the option to select a license. Including an open source license will make your GitHub project open source.



If you have other questions or concerns around the legal aspects of managing an open source project, we've got you covered.

Writing a README

READMEs do more than explain how to use your project. They also explain why your project matters, and what your users can do with it.

In your README, try to answer the following questions:

- What does this project do?
- Why is this project useful?
- How do I get started?
- Where can I get more help, if I need it?

You can use your README to answer other questions, like how you handle contributions, what the goals of the project are, and information about licenses and attribution. If you don't want to accept contributions, or your project is not yet ready for production, write this information down.

> Better documentation means more users, less support requests, and more contributors. (...) Remember that your readers aren't you. There are people who might come to a project who have completely different experiences.
>
> — @tracymakes, "Writing So Your Words Are Read (video)"

Sometimes, people avoid writing a README because they feel like the project is unfinished, or they don't want contributions. These are all very good reasons to write one.

For more inspiration, try using @dguo's "Make a README" guide or @PurpleBooth's README template to write a complete README.

When you include a README file in the root directory, GitHub will automatically display it on the repository homepage.

Writing your contributing guidelines

A CONTRIBUTING file tells your audience how to participate in your project. For example, you might include information on:

- How to file a bug report (try using issue and pull request templates)
- How to suggest a new feature
- How to set up your environment and run tests

In addition to technical details, a CONTRIBUTING file is an opportunity to communicate your expectations for contributions, such as:

- The types of contributions you're looking for
- Your roadmap or vision for the project
- How contributors should (or should not) get in touch with you

Using a warm, friendly tone and offering specific suggestions for contributions (such as writing documentation, or making a website) can go a long way in making newcomers feel welcomed and excited to participate.

For example, Active Admin starts its contributing guide with:

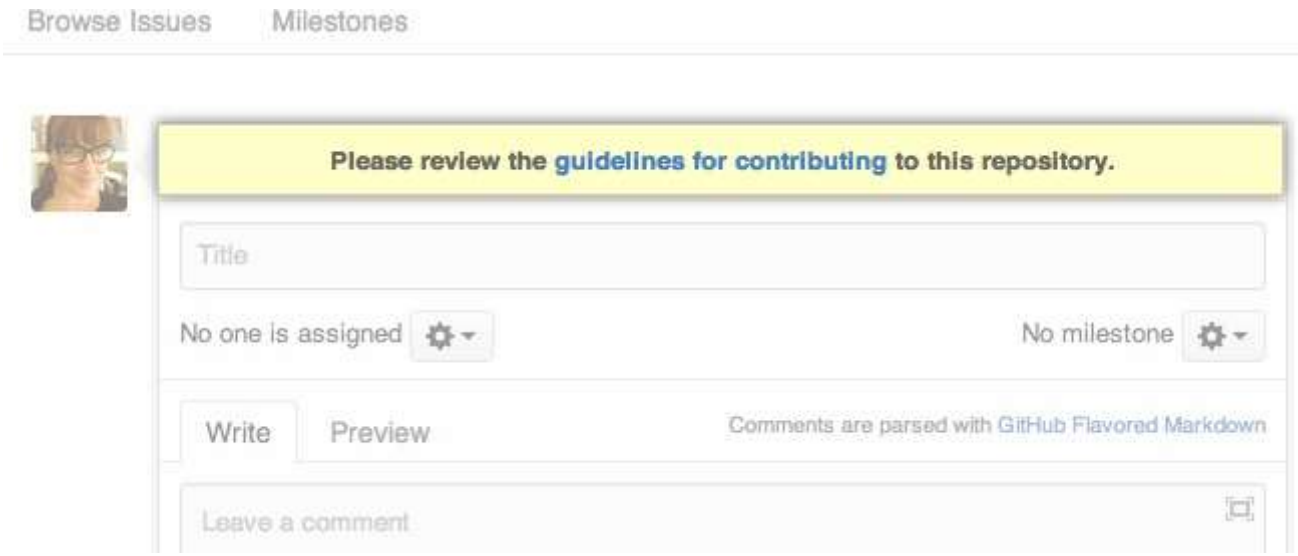
> First off, thank you for considering contributing to Active Admin. It's people like you that make Active Admin such a great tool.

In the earliest stages of your project, your CONTRIBUTING file can be simple. You should always explain how to report bugs or file issues, and any technical requirements (like tests) to make a contribution.

Over time, you might add other frequently asked questions to your CONTRIBUTING file. Writing down this information means fewer people will ask you the same questions over and over again.

For more help with writing your CONTRIBUTING file, check out @nayafia's contributing guide template or @mozilla's "How to Build a CONTRIBUTING.md".

Link to your CONTRIBUTING file from your README, so more people see it. If you place the CONTRIBUTING file in your project's repository, GitHub will automatically link to your file when a contributor creates an issue or opens a pull request.



Establishing a code of conduct

We've all had experiences where we faced what was probably abuse either as a maintainer trying to explain why something had to be a certain way, or as a user...asking a simple question. (...) A code of conduct becomes an easily referenced and linkable document that indicates that your team takes constructive discourse very seriously.

— @mlynch, "Making Open Source a Happier Place"

Finally, a code of conduct helps set ground rules for behavior for your project's participants. This is especially valuable if you're launching an open source project for a community or company. A code of conduct empowers you to facilitate healthy, constructive community behavior, which will reduce your stress as a maintainer.

For more information, check out our Code of Conduct guide.

In addition to communicating *how* you expect participants to behave, a code of conduct also tends to describe who these expectations apply to, when they apply, and what to do if a violation occurs.

Much like open source licenses, there are also emerging standards for codes of conduct, so you don't have to write your own. The Contributor Covenant is a drop-in code of conduct that is used by over 40,000 open source projects, including Kubernetes, Rails, and Swift. No matter which text you use, you should be prepared to enforce your code of conduct when necessary.

Paste the text directly into a CODE_OF_CONDUCT file in your repository. Keep the file in your project's root directory so it's easy to find, and link to it from your README.

Section 4

Naming and branding your project

Branding is more than a flashy logo or catchy project name. It's about how you talk about your project, and who you reach with your message.

Choosing the right name

Pick a name that is easy to remember and, ideally, gives some idea of what the project does. For example:

- Sentry monitors apps for crash reporting
- Thin is a fast and simple Ruby web server

If you're building upon an existing project, using their name as a prefix can help clarify what your project does (for example, node-fetch brings `window.fetch` to Node.js).

Consider clarity above all. Puns are fun, but remember that some jokes might not translate to other cultures or people with different experiences from you. Some of your potential users might be company employees: you don't want to make them uncomfortable when they have to explain your project at work!

Avoiding name conflicts

Check for open source projects with a similar name, especially if you share the same language or ecosystem. If your name overlaps with a popular existing project, you might confuse your audience.

If you want a website, Twitter handle, or other properties to represent your project, make sure you can get the names you want. Ideally, reserve those names now for peace of mind, even if you don't intend to use them just yet.

Make sure that your project's name doesn't infringe upon any trademarks. A company might ask you to take down your project later on, or even take legal action against you. It's just not worth the risk.

You can check the WIPO Global Brand Database for trademark conflicts. If you're at a company, this is one of the things your legal team can help you with.

Finally, do a quick Google search for your project name. Will people be able to easily find your project? Does something else appear in the search results that you wouldn't want them to see?

How you write (and code) affects your brand, too!

Throughout the life of your project, you'll do a lot of writing: READMEs, tutorials, community documents, responding to issues, maybe even newsletters and mailing lists.

Whether it's official documentation or a casual email, your writing style is part of your project's brand. Consider how you might come across to your audience and whether that is the tone you wish to convey.

I tried to be involved with every thread on the mailing list, and showing exemplary behaviour, being nice to people, taking their issues seriously and trying to be helpful overall. After a while,

> people stuck around not to only ask questions, but to help with the answering as well, and to my complete delight, they mimicked my style.
>
> — @janl on CouchDB, "Sustainable Open Source"

Using warm, inclusive language (such as "them", even when referring to the single person) can go a long way in making your project feel welcoming to new contributors. Stick to simple language, as many of your readers may not be native English speakers.

Beyond how you write words, your coding style may also become part of your project's brand. Angular and jQuery are two examples of projects with rigorous coding styles and guidelines.

It isn't necessary to write a style guide for your project when you're just starting out, and you may find that you enjoy incorporating different coding styles into your project anyway. But you should anticipate how your writing and coding style might attract or discourage different types of people. The earliest stages of your project are your opportunity to set the precedent you wish to see.

Section 5

Your pre-launch checklist

Ready to open source your project? Here's a checklist to help. Check all the boxes? You're ready to go! Click "publish" and pat yourself on the back.

Documentation

- [ ] Project has a LICENSE file with an open source license
- [ ] Project has basic documentation (README, CONTRIBUTING, CODE_OF_CONDUCT)
- [ ] The name is easy to remember, gives some idea of what the project does, and does not conflict with an existing project or infringe on trademarks
- [ ] The issue queue is up-to-date, with issues clearly organized and labeled

Code

- [ ] Project uses consistent code conventions and clear function/method/variable names
- [ ] The code is clearly commented, documenting intentions and edge cases
- [ ]

There are no sensitive materials in the revision history, issues, or pull requests (for example, passwords or other non-public information)

People

If you're an individual:

- [ ] You've talked to the legal department and/or understand the IP and open source policies of your company (if you're an employee somewhere)

If you're a company or organization:

- [ ] You've talked to your legal department
- [ ] You have a marketing plan for announcing and promoting the project
- [ ] Someone is committed to managing community interactions (responding to issues, reviewing and merging pull requests)
- [ ] At least two people have administrative access to the project

Section 6

You did it!

Congratulations on open sourcing your first project. No matter the outcome, working in public is a gift to the community. With every commit, comment, and pull request, you're creating opportunities for yourself and for others to learn and grow.

Back to all guides

Related Guides

Finding Users for Your Project

Help your open source project grow by getting it in the hands of

Building Welcoming Communities

Building a community that encourages people to use, contribute to, and evangelize your project.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Subscribe

Finding Users for Your Project

Help your open source project grow by getting it in the hands of happy users.

Table of Contents

Section 1

Spreading the word

There's no rule that says you have to promote an open source project when you launch. There are many fulfilling reasons to work in open source that have nothing to do with popularity. Instead of hoping others will find and use your open source project, you have to spread the word about your hard work!

Section 2

Figure out your message

Before you start the actual work of promoting your project, you should be able to explain what it does, and why it matters.

What makes your project different or interesting? Why did you create it? Answering these questions for yourself will help you communicate your project's significance.

Remember that people get involved as users, and eventually become contributors, because your project solves a problem for them. As you think about your project's message and value, try to view them through the lens of what *users and contributors* might want.

For example, @robb uses code examples to clearly communicate why his project, Cartography, is useful:

Cartography 📱📐

Using Cartography, you can set up your Auto Layout constraints in declarative code and without any stringly typing!

Carthage compatible build passing

In short, it allows you to replace this:

```
addConstraint(NSLayoutConstraint(
    item: button1,
    attribute: .Right,
    relatedBy: .Equal,
    toItem: button2,
    attribute: .Left,
    multiplier: 1.0,
    constant: -12.0
))
```

with this

```
constrain(button1, button2) { button1, button2 in
    button1.right == button2.left - 12
}
```

For a deeper dive into messaging, check out Mozilla's "Personas and Pathways" exercise for developing user personas.

Section 3

Help people find and follow your project

> You ideally need a single "home" URL that you can promote and point people to in relation to your project. You don't need to splash out on a fancy template or even a domain name, but your project needs a focal point.
>
> — Peter Cooper & Robert Nyman, "How to Spread the Word About Your Code"

Help people find and remember your project by pointing them to a single namespace.

Have a clear handle to promote your work. A Twitter handle, GitHub URL, or IRC channel is an easy way to point people to your project. These outlets also give your project's growing community a place to convene.

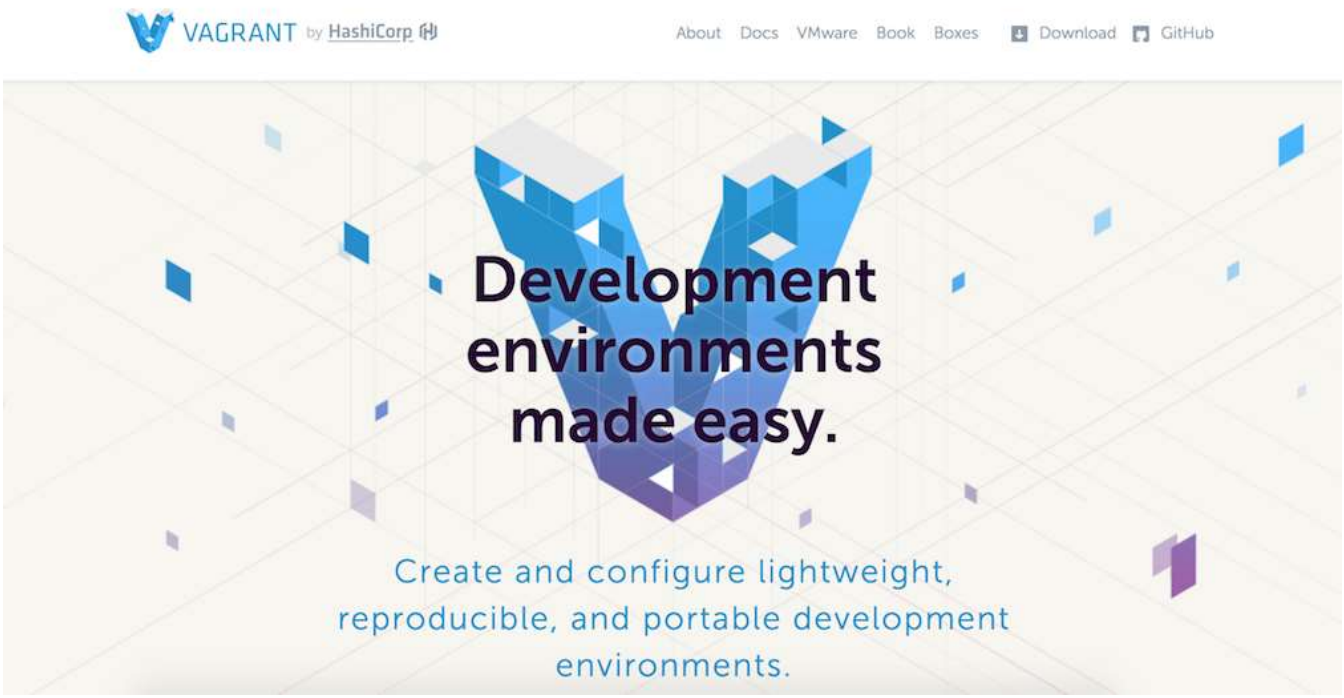
If you don't wish to set up outlets for your project yet, promote your own Twitter or GitHub handle in everything you do. Promoting your Twitter or GitHub handle will let people know how to contact you or follow your work. If you speak at a meetup or event, make sure that your contact information is included in your bio or slides.

> A mistake I made in those early days (...) was not starting a Twitter account for the project. Twitter's a great way to keep people up to date about a project as well as constantly expose people to the project.
>
> — @nathanmarz, "History of Apache Storm and Lessons Learned"

Consider creating a website for your project. A website makes your project friendlier and easier to navigate, especially when it's paired with clear documentation and tutorials. Having a website also suggests that your project is active which will make your audience feel more comfortable using it. Provide examples to give people ideas for how to use your project.

@adrianholovaty, co-creator of Django, said that a website was *"by far the best thing we did with Django in the early days"*.

If your project is hosted on GitHub, you can use GitHub Pages to easily make a website. Yeoman, Vagrant, and Middleman are a few examples of excellent, comprehensive websites.



Now that you have a message for your project, and an easy way for people to find your project, let's get out there and talk to your audience!

Section 4

Go where your project's audience is (online)

Online outreach is a great way to share and spread the word quickly. Using online channels, you have the potential to reach a very wide audience.

Take advantage of existing online communities and platforms to reach your audience. If your open source project is a software project, you can probably find your audience on Stack Overflow, Reddit, Hacker News, or Quora. Find the channels where you think people will most benefit from or be excited about your work.

> Each program has very specific functions that only a fraction of users will find useful. Don't spam as many people as possible. Instead, target your efforts to communities that will benefit from knowing about your project.
>
> — @pazdera, "Marketing for open source projects"

See if you can find ways to share your project in relevant ways:

- **Get to know relevant open source projects and communities.** Sometimes, you don't have to directly promote your project. If your project is perfect for data scientists who use Python, get to know the Python data science community. As people get to know you, natural opportunities will arise to talk about and share your work.
- **Find people experiencing the problem that your project solves.** Search through related forums for people who fall into your project's target audience. Answer their question and find a tactful way, when appropriate, to suggest your project as a solution.
- **Ask for feedback. I**ntroduce yourself and your work to an audience that would find it relevant and interesting. Be specific about who you think would benefit from your project. Try to finish the sentence: *"I think my project would really help X, who are trying to do Y*". Listen and respond to others' feedback, rather than simply promoting your work.

Generally speaking, focus on helping others before asking for things in return. Because anyone can easily promote a project online, there will be a lot of noise. To stand out from the crowd, give people context for who you are and not just what you want.

If nobody pays attention or responds to your initial outreach, don't get discouraged! Most project launches are an iterative process that can take months or years. If you don't get a response the first time, try a different tactic, or look for ways to add value to others' work first. Promoting and launching your project takes time and dedication.

Section 5

Go where your project's audience is (offline)

Offline events are a popular way to promote new projects to audiences. They're a great way to reach an engaged audience and build deeper human connections, especially if you are interested in reaching developers.

If you're new to public speaking, start by finding a local meetup that's related to the language or ecosystem of your project.

I was pretty nervous about going to PyCon. I was giving a talk, I was only going to know a couple of people there, I was going for an entire week. (...) I shouldn't have worried, though. PyCon was phenomenally awesome! (...) Everyone was incredibly friendly and outgoing, so much that I rarely found time not to talk to people!

— @jhamrick, "How I learned to Stop Worrying and Love PyCon"

If you've never spoken at an event before, it's perfectly normal to feel nervous! Remember that your audience is there because they genuinely want to hear about your work.

As you write your talk, focus on what your audience will find interesting and get value out of. Keep your language friendly and approachable. Smile, breathe, and have fun.

When you start writing your talk, no matter what your topic is, it can help if you see your talk as a story that you tell people.

— Lena Reinhard, "How to Prepare and Write a Tech Conference Talk"

When you feel ready, consider speaking at a conference to promote your project. Conferences can help you reach more people, sometimes from all over the world.

Look for conferences that are specific to your language or ecosystem. Before you submit your talk, research the conference to tailor your talk for attendees and increase your chances of being accepted to speak at the conference. You can often get a sense of your audience by looking at a conference's speakers.

I wrote very nicely to the JSConf people and begged them to give me a slot where I could present it at JSConf EU. (...) I was extremely scared, presenting this thing that I had been working on for six months. (...) The whole time I was just thinking, oh my God. What am I doing here?

— @ry, "History of Node.js" (video)

Section 6

Build a reputation

In addition to the strategies outlined above, the best way to invite people to share and contribute to your project is to share and contribute to their projects.

Helping newcomers, sharing resources, and making thoughtful contributions to others' projects will help you build a positive reputation. Being an active member in the open source community will help people have context for your work and be more likely to pay attention to and share your project. Developing relationships with other open source projects can even lead to official partnerships.

> The only reason urllib3 is the most popular third-party Python library today is because it's part of requests.
>
> — @shazow, "How to make your open source project thrive"

It's never too early, or too late, to start building your reputation. Even if you've launched your own project already, continue to look for ways to help others.

There is no overnight solution to building an audience. Gaining the trust and respect of others takes time, and building your reputation never ends.

Section 7

Keep at it!

It may take a long time before people notice your open source project. That's okay! Some of the most popular projects today took years to reach high levels of activity. Focus on building relationships instead of hoping that your project will spontaneously gain popularity. Be patient, and keep sharing your work with those who appreciate it.

Back to all guides

Related Guides

Starting an Open Source Project

Learn more about the world of open source and get ready to launch your own project.

Building Welcoming Communities

Building a community that encourages people to use, contribute to, and evangelize your project.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Subscribe

Building Welcoming Communities

Building a community that encourages people to use, contribute to, and evangelize your project.

Table of Contents

Section 1

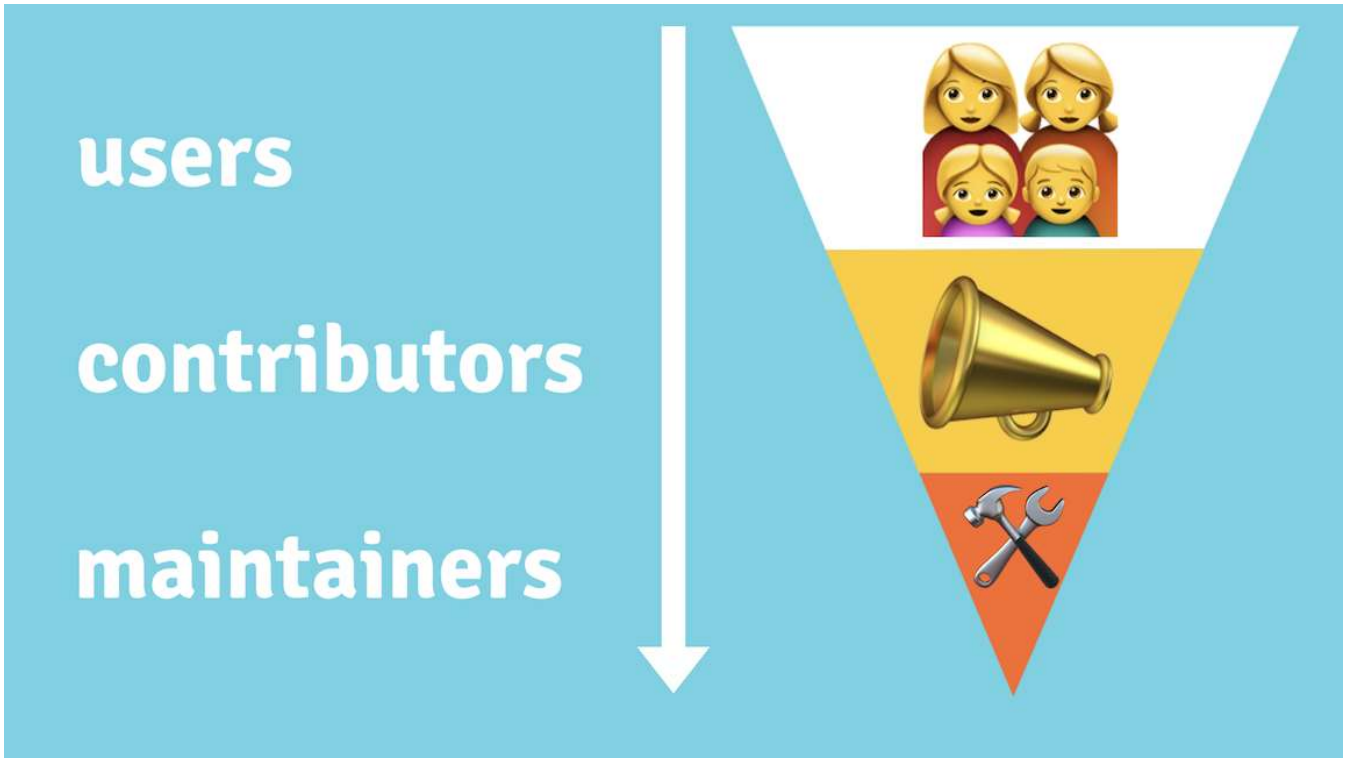
Setting your project up for success

You've launched your project, you're spreading the word, and people are checking it out. Awesome! Now, how do you get them to stick around?

A welcoming community is an investment into your project's future and reputation. If your project is just starting to see its first contributions, start by giving early contributors a positive experience, and make it easy for them to keep coming back.

Make people feel welcome

One way to think about your project's community is through what @MikeMcQuaid calls the contributor funnel:



As you build your community, consider how someone at the top of the funnel (a potential user) might theoretically make their way to the bottom (an active maintainer). Your goal is to reduce friction at each stage of the contributor experience. When people have easy wins, they will feel incentivized to do more.

Start with your documentation:

- **Make it easy for someone to use your project.** A friendly README and clear code examples will make it easier for anyone who lands on your project to get started.
- **Clearly explain how to contribute**, using your CONTRIBUTING file and keeping your issues up-to-date.
- **Good first issues**: To help new contributors get started, consider explicitly labeling issues that are simple enough for beginners to tackle. GitHub will then surface these issues in various places on the platform, increasing useful contributions, and reducing friction from users tackling issues that are too hard for their level.

GitHub's 2017 Open Source Survey showed incomplete or confusing documentation is the biggest problem for open source users. Good documentation invites people to interact with your project. Eventually, someone will open an issue or pull request. Use these interactions as opportunities to move them down the funnel.

- **When someone new lands on your project, thank them for their interest!** It only takes one negative experience to make someone not want to come back.
- **Be responsive.** If you don't respond to their issue for a month, chances are, they've already forgotten about your project.
- **Be open-minded about the types of contributions you'll accept.** Many contributors start with a bug report or a small fix. There are many ways to contribute to a project. Let people help how they want to help.
- **If there's a contribution you disagree with,** thank them for their idea and explain why it doesn't fit into the scope of the project, linking to relevant documentation if you have it.

> Contributing to open source is easier for some than others. There's a lot of fear of being yelled at for not doing something right or just not fitting in. (...) By giving contributors a place to contribute with very low technical proficiency (documentation, web content markdown, etc) you can greatly reduce those concerns.
>
> — @mikeal, "Growing a contributor base in modern open source"

The majority of open source contributors are "casual contributors": people who contribute to a project only occasionally. A casual contributor may not have time to get fully up to speed with your project, so your job is to make it easy for them to contribute.

Encouraging other contributors is an investment in yourself, too. When you empower your biggest fans to run with the work they're excited about, there's less pressure to do everything yourself.

Document everything

> Have you ever been to a (tech-) event where you didn't know anyone, but everyone else seemed to stand in groups and chat like old friends? (…) Now imagine you want to contribute to an open source project, but you don't see why or how this is happening.
>
> — @janl, "Sustainable Open Source"

When you start a new project, it may feel natural to keep your work private. But open source projects thrive when you document your process in public.

When you write things down, more people can participate at every step of the way. You might get help on something you didn't even know you needed.

Writing things down means more than just technical documentation. Any time you feel the urge to write something down or privately discuss your project, ask yourself whether you can make it public.

Be transparent about your project's roadmap, the types of contributions you're looking for, how contributions are reviewed, or why you made certain decisions.

If you notice multiple users running into the same problem, document the answers in the README.

For meetings, consider publishing your notes or takeaways in a relevant issue. The feedback you'll get from this level of transparency may surprise you.

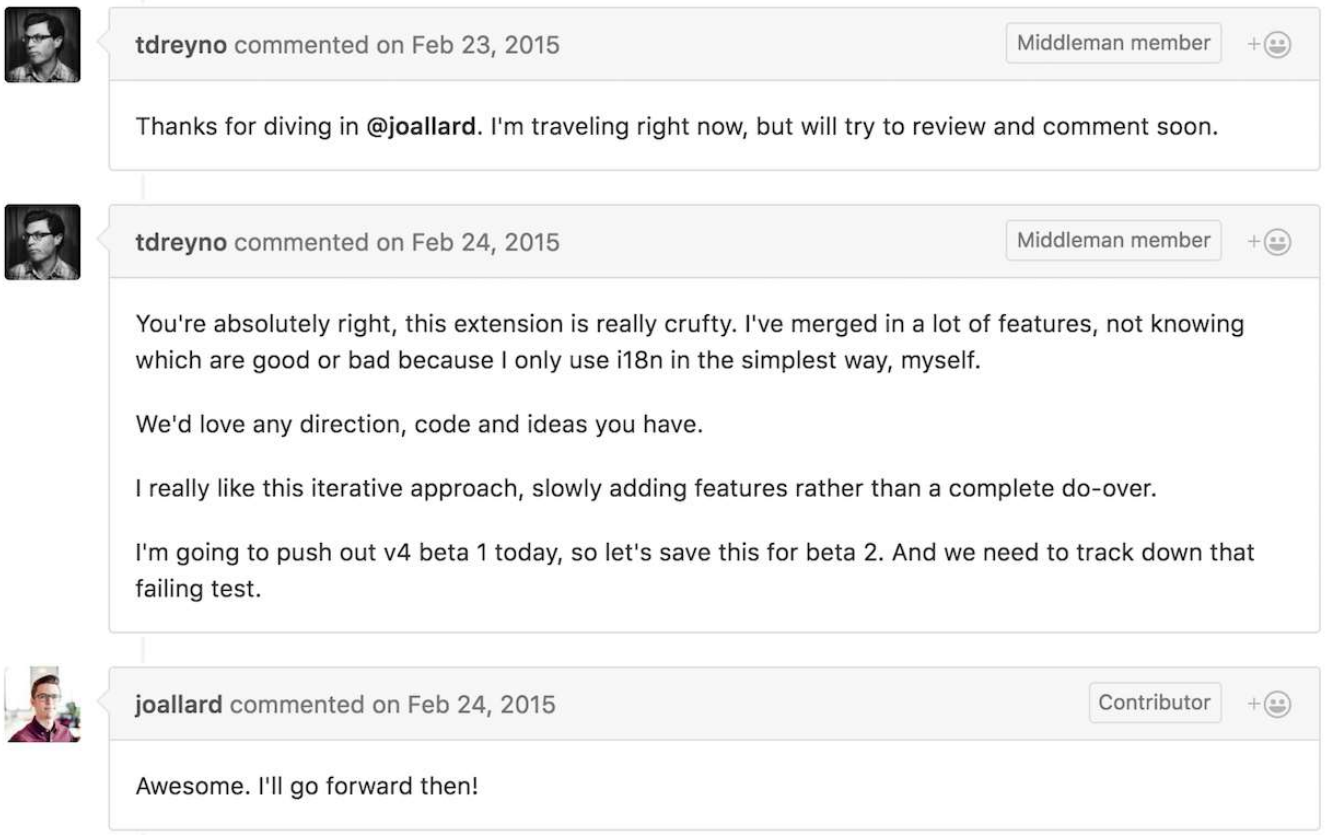
Documenting everything applies to the work you do, too. If you're working on a substantial update to your project, put it into a pull request and mark it as a work in progress (WIP). That way, other people can feel involved in the process early on.

Be responsive

As you promote your project, people will have feedback for you. They may have questions about how things work, or need help getting started.

Try to be responsive when someone files an issue, submits a pull request, or asks a question about your project. When you respond quickly, people will feel they are part of a dialogue, and they'll be more enthusiastic about participating.

Even if you can't review the request immediately, acknowledging it early helps increase engagement. Here's how @tdreyno responded to a pull request on Middleman:



A Mozilla study found that contributors who received code reviews within 48 hours had a much higher rate of return and repeat contribution.

Conversations about your project could also be happening in other places around the internet, such as Stack Overflow, Twitter, or Reddit. You can set up notifications in some of these places so you are alerted when someone mentions your project.

Give your community a place to congregate

There are two reasons to give your community a place to congregate.

The first reason is for them. Help people get to know each other. People with common interests will inevitably want a place to talk about it. And when communication is public and accessible, anybody can read past archives to get up to speed and participate.

The second reason is for you. If you don't give people a public place to talk about your project, they will likely contact you directly. In the beginning, it may seem easy enough to respond to private messages "just this once". But over time, especially if your project becomes popular, you will feel exhausted. Resist the temptation to communicate with people about your project in private. Instead, direct them to a designated public channel.

Public communication can be as simple as directing people to open an issue instead of emailing you directly or commenting on your blog. You could also set up a mailing list, or

create a Twitter account, Slack, or IRC channel for people to talk about your project. Or try all of the above!

Kubernetes kops sets aside office hours every other week to help community members:

> Kops also has time set aside every other week to offer help and guidance to the community. Kops maintainers have agreed to set aside time specifically dedicated to working with newcomers, helping with PRs, and discussing new features.

Notable exceptions to public communication are: 1) security issues and 2) sensitive code of conduct violations. You should always have a way for people to report these issues privately. If you don't want to use your personal email, set up a dedicated email address.

Section 2

Growing your community

Communities are extremely powerful. That power can be a blessing or a curse, depending on how you wield it. As your project's community grows, there are ways to help it become a force of construction, not destruction.

Don't tolerate bad actors

Any popular project will inevitably attract people who harm, rather than help, your community. They may start unnecessary debates, quibble over trivial features, or bully others.

Do your best to adopt a zero-tolerance policy towards these types of people. If left unchecked, negative people will make other people in your community uncomfortable. They may even leave.

The truth is that having a supportive community is key. I'd never be able to do this work without the help of my colleagues, friendly internet strangers, and chatty IRC channels. (...) Don't settle for less. Don't settle for assholes.

— @okdistribute, "How to Run a FOSS Project"

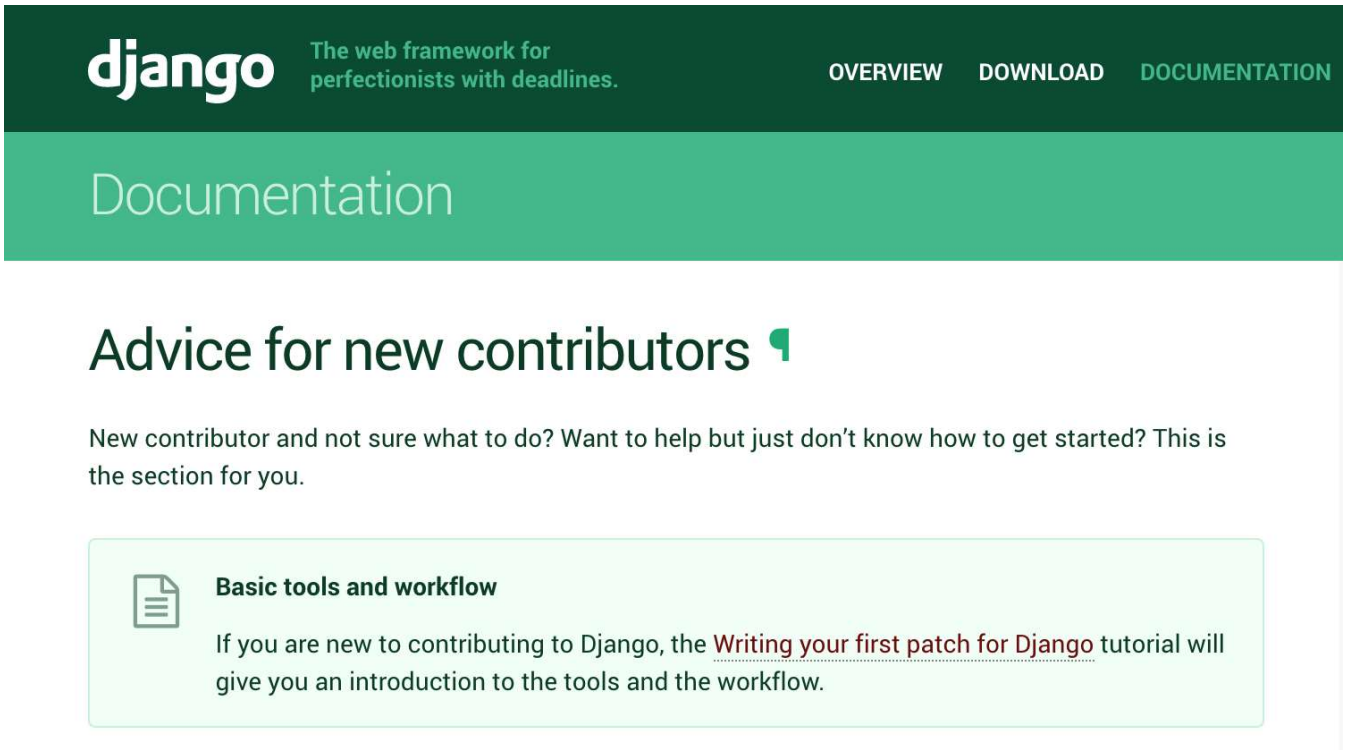
Regular debates over trivial aspects of your project distracts others, including you, from focusing on important tasks. New people who arrive to your project may see these conversations and not want to participate.

When you see negative behavior happening on your project, call it out publicly. Explain, in a kind but firm tone, why their behavior is not acceptable. If the problem persists, you may need to ask them to leave. Your code of conduct can be a constructive guide for these conversations.

Meet contributors where they're at

Good documentation only becomes more important as your community grows. Casual contributors, who may not otherwise be familiar with your project, read your documentation to quickly get the context they need.

In your CONTRIBUTING file, explicitly tell new contributors how to get started. You may even want to make a dedicated section for this purpose. Django, for example, has a special landing page to welcome new contributors.



In your issue queue, label bugs that are suitable for different types of contributors: for example, *"first timers only"*, *"good first issue"*, or *"documentation"*. These labels make it easy for someone new to your project to quickly scan your issues and get started.

Finally, use your documentation to make people feel welcome at every step of the way.

You will never interact with most people who land on your project. There may be contributions you didn't receive because somebody felt intimidated or didn't know where to get started. Even a few kind words can keep someone from leaving your project in frustration.

For example, here's how Rubinius starts its contributing guide:

> We want to start off by saying thank you for using Rubinius. This project is a labor of love, and we appreciate all of the users that catch bugs, make performance improvements, and help with documentation. Every contribution is meaningful, so thank you for participating. That being said, here are a few guidelines that we ask you to follow so we can successfully address your issue.

Share ownership of your project

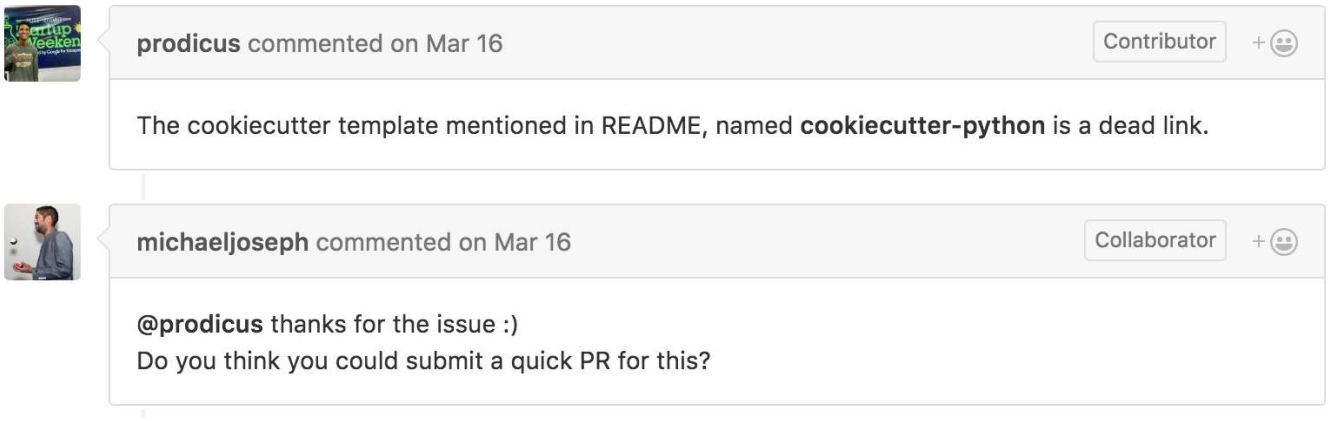
Your leaders will have different opinions, as all healthy communities should! However, you need to take steps to ensure the loudest voice doesn't always win by tiring people out, and that less prominent and minority voices are heard.

— @sagesharp, "What makes a good community?"

People are excited to contribute to projects when they feel a sense of ownership. That doesn't mean you need to turn over your project's vision or accept contributions you don't want. But the more you give credit to others, the more they'll stick around.

See if you can find ways to share ownership with your community as much as possible. Here are some ideas:

- **Resist fixing easy (non-critical) bugs.** Instead, use them as opportunities to recruit new contributors, or mentor someone who'd like to contribute. It may seem unnatural at first, but your investment will pay off over time. For example, @michaeljoseph asked a contributor to submit a pull request on a Cookiecutter issue below, rather than fix it himself.



- **Start a CONTRIBUTORS or AUTHORS file in your project** that lists everyone who's contributed to your project, like Sinatra does.
- If you've got a sizable community, **send out a newsletter or write a blog post** thanking contributors. Rust's This Week in Rust and Hoodie's Shoutouts are two good examples.
- **Give every contributor commit access.** @felixge found that this made people more excited to polish their patches, and he even found new maintainers for projects that he hadn't worked on in awhile.
- If your project is on GitHub, **move your project from your personal account to an Organization** and add at least one backup admin. Organizations make it easier to work on projects with external collaborators.

The reality is that most projects only have one or two maintainers who do most of the work. The bigger your project, and the bigger your community, the easier it is to find help.

While you may not always find someone to answer the call, putting a signal out there increases the chances that other people will pitch in. And the earlier you start, the sooner people can help.

> [It's in your] best interest to recruit contributors who enjoy and who are capable of doing the things that you are not. Do you enjoy coding, but not answering issues? Then identify those individuals in your community who do and let them have it.
>
> — @gr2m, "Welcoming Communities"

Section 3

Resolving conflicts

In the early stages of your project, making major decisions is easy. When you want to do something, you just do it.

As your project becomes more popular, more people will take interest in the decisions you make. Even if you don't have a big community of contributors, if your project has a lot of users, you'll find people weighing in on decisions or raising issues of their own.

For the most part, if you've cultivated a friendly, respectful community and documented your processes openly, your community should be able to find resolution. But sometimes you run into an issue that's a bit harder to address.

Set the bar for kindness

When your community is grappling with a difficult issue, tempers may rise. People may become angry or frustrated and take it out on one another, or on you.

Your job as a maintainer is to keep these situations from escalating. Even if you have a strong opinion on the topic, try to take the position of a moderator or facilitator, rather than jumping into the fight and pushing your views. If someone is being unkind or monopolizing the conversation, act immediately to keep discussions civil and productive.

> As a project maintainer, it's extremely important to be respectful to your contributors. They often take what you say very personally.
>
> — @kennethreitz, "Be Cordial or Be on Your Way"

Other people are looking to you for guidance. Set a good example. You can still express disappointment, unhappiness, or concern, but do so calmly.

Keeping your cool isn't easy, but demonstrating leadership improves the health of your community. The internet thanks you.

Treat your README as a constitution

Your README is more than just a set of instructions. It's also a place to talk about your goals, product vision, and roadmap. If people are overly focused on debating the merit of a particular feature, it may help to revisit your README and talk about the higher vision of your project. Focusing on your README also depersonalizes the conversation, so you can have a constructive discussion.

Focus on the journey, not the destination

Some projects use a voting process to make major decisions. While sensible at first glance, voting emphasizes getting to an "answer," rather than listening to and addressing each other's concerns.

Voting can become political, where community members feel pressured to do each other favors or vote a certain way. Not everybody votes, either, whether it's the silent majority in your community, or current users who didn't know a vote was taking place.

Sometimes, voting is a necessary tiebreaker. As much as you are able, however, emphasize "consensus seeking" rather than consensus.

Under a consensus seeking process, community members discuss major concerns until they feel they have been adequately heard. When only minor concerns remain, the community moves forward. "Consensus seeking" acknowledges that a community may not be able to reach a perfect answer. Instead, it prioritizes listening and discussion.

> Part of the reason why a voting system doesn't exist for Atom Issues is because the Atom team isn't going to follow a voting system in all cases. Sometimes we have to choose what we feel is right even if it is unpopular. (...) What I can offer and pledge to do...is that it is my job to listen to the community.
>
> — @lee-dohm on Atom's decision making process

Even if you don't actually adopt a consensus seeking process, as a project maintainer, it's important that people know you are listening. Making other people feel heard, and committing to resolving their concerns, goes a long way to diffuse sensitive situations. Then, follow up on your words with actions.

Don't rush into a decision for the sake of having a resolution. Make sure that everybody feels heard and that all information has been made public before moving toward a resolution.

Keep the conversation focused on action

Discussion is important, but there is a difference between productive and unproductive conversations.

Encourage discussion so long as it is actively moving towards resolution. If it's clear that conversation is languishing or going off-topic, jabs are getting personal, or people are quibbling about minor details, it's time to shut it down.

Allowing these conversations to continue is not only bad for the issue at hand, but bad for the health of your community. It sends a message that these types of conversations are permitted or even encouraged, and it can discourage people from raising or resolving future issues.

With every point made by you or by others, ask yourself, *"How does this bring us closer to a resolution?"*

If the conversation is starting to unravel, ask the group, *"Which steps should we take next?"* to refocus the conversation.

If a conversation clearly isn't going anywhere, there are no clear actions to be taken, or the appropriate action has already been taken, close the issue and explain why you closed it.

> Guiding a thread toward usefulness without being pushy is an art. It won't work to simply admonish people to stop wasting their time, or to ask them not to post unless they have something constructive to say. (...) Instead, you have to suggest conditions for further progress: give people a route, a path to follow that leads to the results you want, yet without sounding like you're dictating conduct.
>
> — @kfogel, *Producing OSS*

Pick your battles wisely

Context is important. Consider who is involved in the discussion and how they represent the rest of the community.

Is everybody in the community upset about, or even engaged with, this issue? Or is a lone troublemaker? Don't forget to consider your silent community members, not just the active voices.

If the issue does not represent the broader needs of your community, you may just need to acknowledge the concerns of a few people. If this is a recurring issue without a clear resolution, point them to previous discussions on the topic and close the thread.

Identify a community tiebreaker

With a good attitude and clear communication, most difficult situations are resolvable. However, even in a productive conversation, there can simply be a difference in opinion on how to proceed. In these cases, identify an individual or group of people that can serve as a tiebreaker.

A tiebreaker could be the primary maintainer of the project, or it could be a small group of people who make a decision based on voting. Ideally, you've identified a tiebreaker and the associated process in a GOVERNANCE file before you ever have to use it.

Your tiebreaker should be a last resort. Divisive issues are an opportunity for your community to grow and learn. Embrace these opportunities and use a collaborative process to move to a resolution wherever possible.

Section 4

Community is the ❤ of open source

Healthy, thriving communities fuel the thousands of hours poured into open source every week. Many contributors point to other people as the reason for working - or not working - on open source. By learning how to tap into that power constructively, you'll help someone out there have an unforgettable open source experience.

Back to all guides

Related Guides

Best Practices for Maintainers

Making your life easier as an open source maintainer, from documenting processes to leveraging your community.

Your Code of Conduct

Facilitate healthy and constructive community behavior by adopting and enforcing a code of conduct.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Subscribe

Best Practices for Maintainers

Making your life easier as an open source maintainer, from documenting processes to leveraging your community.

Table of Contents

Section 1

What does it mean to be a maintainer?

If you maintain an open source project that a lot of people use, you may have noticed you're coding less and responding to issues more.

In the early stages of a project, you're experimenting with new ideas and making decisions based on what you want. As your project increases in popularity, you'll find yourself working with your users and contributors more.

Maintaining a project requires more than code. These tasks are often unexpected, but they're just as important to a growing project. We've gathered a few ways to make your life easier, from documenting processes to leveraging your community.

Section 2

Documenting your processes

Writing things down is one of the most important things you can do as a maintainer.

Documentation not only clarifies your own thinking, but it helps other people understand what you need or expect, before they even ask.

Writing things down makes it easier to say no when something doesn't fit into your scope. It also makes it easier for people to pitch in and help. You never know who might be reading or using your project.

Even if you don't use full paragraphs, jotting down bullet points is better than not writing at all.

Remember to keep your documentation up-to-date. If you're not able to always do this, delete your outdated documentation or indicate it is outdated so contributors know updates are welcome.

Write down your project's vision

Start by writing down the goals of your project. Add them to your README, or create a separate file called VISION. If there are other artifacts that could help, like a project roadmap, make those public as well.

Having a clear, documented vision keeps you focused and helps you avoid "scope creep" from others' contributions.

For example, @lord discovered that having a project vision helped him figure out which requests to spend time on. As a new maintainer, he regretted not sticking to his project's scope when he got his first feature request for Slate.

> I fumbled it. I didn't put in the effort to come up with a complete solution. Instead of a half-assed solution, I wish I had said "I don't have time for this right now, but I'll add it to the long term nice-to-have list."
>
> — @lord, "Tips for new open source maintainers"

Communicate your expectations

Rules can be nerve-wracking to write down. Sometimes you might feel like you're policing other people's behavior or killing all the fun.

Written and enforced fairly, however, good rules empower maintainers. They prevent you from getting dragged into doing things you don't want to do.

Most people who come across your project don't know anything about you or your circumstances. They may assume you get paid to work on it, especially if it's something they regularly use and depend on. Maybe at one point you put a lot of time into your project, but now you're busy with a new job or family member.

All of this is perfectly okay! Just make sure other people know about it.

If maintaining your project is part-time or purely volunteered, be honest about how much time you have. This is not the same as how much time you think the project requires, or how much time others want you to spend.

Here are a few rules that are worth writing down:

- How a contribution is reviewed and accepted (*Do they need tests? An issue template?*)
- The types of contributions you'll accept (*Do you only want help with a certain part of your code?*)

- When it's appropriate to follow up (*for example, "You can expect a response from a maintainer within 7 days. If you haven't heard anything by then, feel free to ping the thread."*)
- How much time you spend on the project (*for example, "We only spend about 5 hours per week on this project"*)

Jekyll, CocoaPods, and Homebrew are several examples of projects with ground rules for maintainers and contributors.

Keep communication public

Don't forget to document your interactions, too. Wherever you can, keep communication about your project public. If somebody tries to contact you privately to discuss a feature request or support need, politely direct them to a public communication channel, such as a mailing list or issue tracker.

If you meet with other maintainers, or make a major decision in private, document these conversations in public, even if it's just posting your notes.

That way, anybody who joins your community will have access to the same information as someone who's been there for years.

Section 3

Learning to say no

You've written things down. Ideally, everybody would read your documentation, but in reality, you'll have to remind others that this knowledge exists.

Having everything written down, however, helps depersonalize situations when you do need to enforce your rules.

Saying no isn't fun, but *"Your contribution doesn't match this project's criteria"* feels less personal than *"I don't like your contribution"*.

Saying no applies to many situations you'll come across as a maintainer: feature requests that don't fit the scope, someone derailing a discussion, doing unnecessary work for others.

Keep the conversation friendly

One of the most important places you'll practice saying no is on your issue and pull request queue. As a project maintainer, you'll inevitably receive suggestions that you don't want to accept.

Maybe the contribution changes your project's scope or doesn't match your vision. Maybe the idea is good, but the implementation is poor.

Regardless of the reason, it is possible to tactfully handle contributions that don't meet your project's standards.

If you receive a contribution you don't want to accept, your first reaction might be to ignore it or pretend you didn't see it. Doing so could hurt the other person's feelings and even demotivate other potential contributors in your community.

> The key to handling support for large-scale open source projects is to keep issues moving. Try to avoid having issues stall. If you're an iOS developer you know how frustrating it can be to submit radars. You might hear back 2 years later, and are told to try again with the latest version of iOS.
>
> — @KrauseFx, "Scaling open source communities"

Don't leave an unwanted contribution open because you feel guilty or want to be nice. Over time, your unanswered issues and PRs will make working on your project feel that much more stressful and intimidating.

It's better to immediately close the contributions you know you don't want to accept. If your project already suffers from a large backlog, @steveklabnik has suggestions for how to triage issues efficiently.

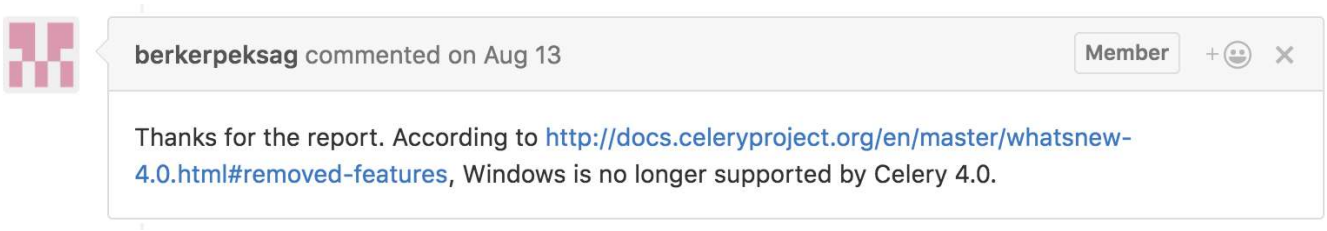
Secondly, ignoring contributions sends a negative signal to your community. Contributing to a project can be intimidating, especially if it's someone's first time. Even if you don't accept their contribution, acknowledge the person behind it and thank them for their interest. It's a big compliment!

If you don't want to accept a contribution:

- **Thank them** for their contribution
- **Explain why it doesn't fit** into the scope of the project, and offer clear suggestions for improvement, if you're able. Be kind, but firm.

- **Link to relevant documentation**, if you have it. If you notice repeated requests for things you don't want to accept, add them into your documentation to avoid repeating yourself.
- **Close the request**

You shouldn't need more than 1-2 sentences to respond. For example, when a user of celery reported a Windows-related error, @berkerpeksag responded with:



If the thought of saying no terrifies you, you're not alone. As @jessfraz put it:

> I've talked to maintainers from several different open source projects, Mesos, Kubernetes, Chromium, and they all agree one of the hardest parts of being a maintainer is saying "No" to patches you don't want.

Don't feel guilty about not wanting to accept someone's contribution. The first rule of open source, according to @shykes: *"No is temporary, yes is forever."* While empathizing with another person's enthusiasm is a good thing, rejecting a contribution is not the same as rejecting the person behind it.

Ultimately, if a contribution isn't good enough, you're under no obligation to accept it. Be kind and responsive when people contribute to your project, but only accept changes that you truly believe will make your project better. The more often you practice saying no, the easier it becomes. Promise.

Be proactive

To reduce the volume of unwanted contributions in the first place, explain your project's process for submitting and accepting contributions in your contributing guide.

If you're receiving too many low-quality contributions, require that contributors do a bit of work beforehand, for example:

- Fill out a issue or PR template/checklist
- Open an issue before submitting a PR

If they don't follow your rules, close the issue immediately and point to your documentation.

While this approach may feel unkind at first, being proactive is actually good for both parties. It reduces the chance that someone will put in many wasted hours of work into a pull request

that you aren't going to accept. And it makes your workload easier to manage.

Ideally, explain to them and in a CONTRIBUTING.md file how they can get a better indication in the future on what would or would not be accepted before they begin the work.

— @MikeMcQuaid, "Kindly Closing Pull Requests"

Sometimes, when you say no, your potential contributor may get upset or criticize your decision. If their behavior becomes hostile, take steps to defuse the situation or even remove them from your community, if they're not willing to collaborate constructively.

Embrace mentorship

Maybe someone in your community regularly submits contributions that don't meet your project's standards. It can be frustrating for both parties to repeatedly go through rejections.

If you see that someone is enthusiastic about your project, but needs a bit of polish, be patient. Explain clearly in each situation why their contributions don't meet the expectations of the project. Try pointing them to an easier or less ambiguous task, like an issue marked *"good first issue,"* to get their feet wet. If you have time, consider mentoring them through their first contribution, or find someone else in your community who might be willing to mentor them.

Section 4

Leverage your community

You don't have to do everything yourself. Your project's community exists for a reason! Even if you don't yet have an active contributor community, if you have a lot of users, put them to work.

Share the workload

If you're looking for others to pitch in, start by asking around.

One way to gain new contributors is to explicitly label issues that are simple enough for beginners to tackle. GitHub will then surface these issues in various places on the platform, increasing their visibility.

When you see new contributors making repeated contributions, recognize their work by offering more responsibility. Document how others can grow into leadership roles if they wish.

Encouraging others to share ownership of the project can greatly reduce your own workload, as @lmccart discovered on her project, p5.js.

> I'd been saying, "Yeah, anyone can be involved, you don't have to have a lot of coding expertise [...]." We had people sign up to come [to an event] and that's when I was really wondering: is this true, what I've been saying? There are gonna be 40 people who show up, and it's not like I can sit with each of them...But people came together, and it just sort of worked. As soon as one person got it, they could teach their neighbor.
>
> — @lmccart, "What Does "Open Source" Even Mean? p5.js Edition"

If you need to step away from your project, either on hiatus or permanently, there's no shame in asking someone else to take over for you.

If other people are enthusiastic about its direction, give them commit access or formally hand over control to someone else. If someone forked your project and is actively maintaining it elsewhere, consider linking to the fork from your original project. It's great that so many people want your project to live on!

@progrium found that documenting the vision for his project, Dokku, helped those goals live on even after he stepped away from the project:

> I wrote a wiki page describing what I wanted and why I wanted it. For some reason it came as a surprise to me that the maintainers started moving the project in that direction! Did it happen exactly how I'd do it? Not always. But it still brought the project closer to what I wrote down.

Let others build the solutions they need

If a potential contributor has a different opinion on what your project should do, you may want to gently encourage them to work on their own fork.

Forking a project doesn't have to be a bad thing. Being able to copy and modify projects is one of the best things about open source. Encouraging your community members to work on their own fork can provide the creative outlet they need, without conflicting with your project's vision.

I cater to the 80% use case. If you are one of the unicorns, please fork my work. I won't get offended! My public projects are almost always meant to solve the most common problems; I try to make it easy to go deeper by either forking my work or extending it.

— @geerlingguy, "Why I Close PRs"

The same applies to a user who really wants a solution that you simply don't have the bandwidth to build. Offering APIs and customization hooks can help others meet their own needs, without having to modify the source directly. @orta found that encouraging plugins for CocoaPods led to "some of the most interesting ideas":

> It's almost inevitable that once a project becomes big, maintainers have to become a lot more conservative about how they introduce new code. You become good at saying "no", but a lot of people have legitimate needs. So, instead you end up converting your tool into a platform.

Section 5

Bring in the robots

Just as there are tasks that other people can help you with, there are also tasks that no human should ever have to do. Robots are your friend. Use them to make your life as a maintainer easier.

Require tests and other checks to improve the quality of your code

One of the most important ways you can automate your project is by adding tests.

Tests help contributors feel confident that they won't break anything. They also make it easier for you to review and accept contributions quickly. The more responsive you are, the more engaged your community can be.

Set up automatic tests that will run on all incoming contributions, and ensure that your tests can easily be run locally by contributors. Require that all code contributions pass your tests before they can be submitted. You'll help set a minimum standard of quality for all submissions. Required status checks on GitHub can help ensure no change gets merged without your tests passing.

If you add tests, make sure to explain how they work in your CONTRIBUTING file.

> I believe that tests are necessary for all code that people work on. If the code was fully and perfectly correct, it wouldn't need changes – we only write code when something is wrong, whether that's "It crashes" or "It lacks such-and-such a feature". And regardless of the changes you're making, tests are essential for catching any regressions you might accidentally introduce.
>
> — @edunham, "Rust's Community Automation"

Use tools to automate basic maintenance tasks

The good news about maintaining a popular project is that other maintainers have probably faced similar issues and built a solution for them.

There are a variety of tools available to help automate some aspects of maintenance work. A few examples:

- semantic-release automates your releases
- mention-bot mentions potential reviewers for pull requests
- Danger helps automate code review

- no-response closes issues where the author hasn't responded to a request for more information
- dependabot checks your dependency files every day for outdated requirements and opens individual pull requests for any it finds

For bug reports and other common contributions, GitHub has Issue Templates and Pull Request Templates, which you can create to streamline the communication you receive. @TalAter made a Choose Your Own Adventure guide to help you write your issue and PR templates.

To manage your email notifications, you can set up email filters to organize by priority.

If you want to get a little more advanced, style guides and linters can standardize project contributions and make them easier to review and accept.

However, if your standards are too complicated, they can increase the barriers to contribution. Make sure you're only adding enough rules to make everyone's lives easier.

If you're not sure which tools to use, look at what other popular projects do, especially those in your ecosystem. For example, what does the contribution process look like for other Node modules? Using similar tools and approaches will also make your process more familiar to your target contributors.

Section 6

It's okay to hit pause

Open source work once brought you joy. Maybe now it's starting to make you feel avoidant or guilty.

Perhaps you're feeling overwhelmed or a growing sense of dread when you think about your projects. And meanwhile, the issues and pull requests pile up.

Burnout is a real and pervasive issue in open source work, especially among maintainers. As a maintainer, your happiness is a non-negotiable requirement for the survival of any open source project.

Although it should go without saying, take a break! You shouldn't have to wait until you feel burned out to take a vacation. @brettcannon, a Python core developer, decided to take a month-long vacation after 14 years of volunteer OSS work.

Just like any other type of work, taking regular breaks will keep you refreshed, happy, and excited about your work.

> In maintaining WP-CLI, I've discovered I need to make myself happy first, and set clear boundaries on my involvement. The best balance I've found is 2-5 hours per week, as a part of my normal work schedule. This keeps my involvement a passion, and from feeling too much like work. Because I prioritize the issues I'm working on, I can make regular progress on what I think is most important.
>
> — @danielbachhuber, "My condolences, you're now the maintainer of a popular open source project"

Sometimes, it can be hard to take a break from open source work when it feels like everybody needs you. People may even try to make you feel guilty for stepping away.

Do your best to find support for your users and community while you're away from a project. If you can't find the support you need, take a break anyway. Be sure to communicate when you're not available, so people aren't confused by your lack of responsiveness.

Taking breaks applies to more than just vacations, too. If you don't want to do open source work on weekends, or during work hours, communicate those expectations to others, so they know not to bother you.

Section 7

Take care of yourself first!

Maintaining a popular project requires different skills than the earlier stages of growth, but it's no less rewarding. As a maintainer, you'll practice leadership and personal skills on a level that few people get to experience. While it's not always easy to manage, setting clear boundaries and only taking on what you're comfortable with will help you stay happy, refreshed, and productive.

Back to all guides

Related Guides

Open Source Metrics

Make informed decisions to help your open source project thrive by measuring and tracking its success.

Leadership and Governance

Growing open source projects can benefit from formal rules for making decisions.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Subscribe

Leadership and Governance

Growing open source projects can benefit from formal rules for making decisions.

Table of Contents

Section 1

Understanding governance for your growing project

Your project is growing, people are engaged, and you're committed to keeping this thing going. At this stage, you may be wondering how to incorporate regular project contributors into your workflow, whether it's giving someone commit access or resolving community debates. If you have questions, we've got answers.

Section 2

What are examples of formal roles used in open source projects?

Many projects follow a similar structure for contributor roles and recognition.

What these roles actually mean, though, is entirely up to you. Here are a few types of roles you may recognize:

- **Maintainer**
- **Contributor**
- **Committer**

For some projects, "maintainers" are the only people in a project with commit access. In other projects, they're simply the people who are listed in the README as maintainers.

A maintainer doesn't necessarily have to be someone who writes code for your project. It could be someone who's done a lot of work evangelizing your project, or written documentation that made the project more accessible to others. Regardless of what they do day-to-day, a maintainer is probably someone who feels responsibility over the direction of the project and is committed to improving it.

A "contributor" could be anyone who comments on an issue or pull request, people who add value to the project (whether it's triaging issues, writing code, or organizing events), or anybody with a merged pull request (perhaps the narrowest definition of a contributor).

[For Node.js,] every person who shows up to comment on an issue or submit code is a member of a project's community. Just being able to see them means that they have crossed the line from being a user to being a contributor.

— @mikeal, "Healthy Open Source"

The term "committer" might be used to distinguish commit access, which is a specific type of responsibility, from other forms of contribution.

While you can define your project roles any way you'd like, consider using broader definitions to encourage more forms of contribution. You can use leadership roles to formally recognize people who have made outstanding contributions to your project, regardless of their technical skill.

> You might know me as the "inventor" of Django...but really I'm the guy who got hired to work on a thing a year after it was already made. (...) People suspect that I'm successful because of my programming skill...but I'm at best an average programmer.
>
> — @jacobian, "PyCon 2015 Keynote" (video)

Section 3

How do I formalize these leadership roles?

Formalizing your leadership roles helps people feel ownership and tells other community members who to look to for help.

For a smaller project, designating leaders can be as simple as adding their names to your README or a CONTRIBUTORS text file.

For a bigger project, if you have a website, create a team page or list your project leaders there. For example, Postgres has a comprehensive team page with short profiles for each contributor.

If your project has a very active contributor community, you might form a "core team" of maintainers, or even subcommittees of people who take ownership of different issue areas (for example, security, issue triaging, or community conduct). Let people self-organize and volunteer for the roles they're most excited about, rather than assigning them.

> [We] supplement the core team with several "subteams". Each subteam is focused on a specific area, e.g., language design or libraries. (...) To ensure global coordination and a strong, coherent vision for the project as a whole, each subteam is led by a member of the core team.
>
> — "Rust Governance RFC"

Leadership teams may want to create a designated channel (like on IRC) or meet regularly to discuss the project (like on Gitter or Google Hangout). You can even make those meetings public so other people can listen. Cucumber-ruby, for example, hosts office hours every week.

Once you've established leadership roles, don't forget to document how people can attain them! Establish a clear process for how someone can become a maintainer or join a subcommittee in your project, and write it into your GOVERNANCE.md.

Tools like Vossibility can help you publicly track who is (or isn't) making contributions to the project. Documenting this information avoids the community perception that maintainers are a clique that makes its decisions privately.

Finally, if your project is on GitHub, consider moving your project from your personal account to an Organization and adding at least one backup admin. GitHub Organizations make it easier to manage permissions and multiple repositories and protect your project's legacy through shared ownership.

Section 4

When should I give someone commit access?

Some people think you should give commit access to everybody who makes a contribution. Doing so could encourage more people to feel ownership of your project.

On the other hand, especially for bigger, more complex projects, you may want to only give commit access to people who have demonstrated their commitment. There's no one right way of doing it - do what makes you most comfortable!

If your project is on GitHub, you can use protected branches to manage who can push to a particular branch, and under which circumstances.

Whenever somebody sends you a pull request, give them commit access to your project. While it may sound incredibly stupid at first, using this strategy will allow you to unleash the true power of GitHub. (...) Once people have commit access, they are no longer worried that their patch might go unmerged...causing them to put much more work into it.

— @felixge, "The Pull Request Hack"

Section 5

What are some of the common governance structures for open source projects?

There are three common governance structures associated with open source projects.

- **BDFL:** BDFL stands for "Benevolent Dictator for Life". Under this structure, one person (usually the initial author of the project) has final say on all major project decisions. Python is a classic example. Smaller projects are probably BDFL by default, because there are only one or two maintainers. A project that originated at a company might also fall into the BDFL category.
- **Meritocracy: (Note: the term "meritocracy" carries negative connotations for some communities and has a complex social and political history.)** Under a meritocracy, active project contributors (those who demonstrate "merit") are given a formal decision making role. Decisions are usually made based on pure voting consensus. The meritocracy concept was pioneered by the Apache Foundation; all Apache projects are meritocracies. Contributions can only be made by individuals representing themselves, not by a company.
- **Liberal contribution:** Under a liberal contribution model, the people who do the most work are recognized as most influential, but this is based on current work and not historic contributions. Major project decisions are made based on a consensus seeking process (discuss major grievances) rather than pure vote, and strive to include as many community

perspectives as possible. Popular examples of projects that use a liberal contribution model include Node.js and Rust.

Which one should you use? It's up to you! Every model has advantages and trade-offs. And although they may seem quite different at first, all three models have more in common than they seem. If you're interested in adopting one of these models, check out these templates:

- BDFL model template
- Meritocracy model template
- Node.js's liberal contribution policy

Section 6

Do I need governance docs when I launch my project?

There is no right time to write down your project's governance, but it's much easier to define once you've seen your community dynamics play out. The best (and hardest) part about open source governance is that it is shaped by the community!

Some early documentation will inevitably contribute to your project's governance, however, so start writing down what you can. For example, you can define clear expectations for behavior, or how your contributor process works, even at your project's launch.

If you're part of a company launching an open source project, it's worth having an internal discussion before launch about how your company expects to maintain and make decisions about the project moving forward. You may also want to publicly explain anything particular to how your company will (or won't!) be involved with the project.

We assign small teams to manage projects on GitHub who are actually working on these at Facebook. For example, React is run by a React engineer.

— @caabernathy, "An inside look at open source at Facebook"

Section 7

What happens if corporate employees start submitting contributions?

Successful open source projects get used by many people and companies, and some companies may eventually have revenue streams eventually tied to the project. For example, a company may use the project's code as one component in a commercial service offering.

As the project gets more widely used, people who have expertise in it become more in-demand - you may be one of them! - and will sometimes get paid for work they do in the project.

It's important to treat commercial activity as normal and as just another source of development energy. Paid developers shouldn't get special treatment over unpaid ones, of course; each contribution must be evaluated on its technical merits. However, people should feel comfortable engaging in commercial activity, and feel comfortable stating their use cases when arguing in favor of a particular enhancement or feature.

"Commercial" is completely compatible with "open source". "Commercial" just means there is money involved somewhere - that the software is used in commerce, which is increasingly likely as a project gains adoption. (When open source software is used as part of a non-open-source product, the overall product is still "proprietary" software, though, like open source, it might be used for commercial or non-commercial purposes.)

Like anyone else, commercially-motivated developers gain influence in the project through the quality and quantity of their contributions. Obviously, a developer who is paid for her time may be able to do more than someone who is not paid, but that's okay: payment is just one of many possible factors that could affect how much someone does. Keep your project discussions focused on the contributions, not on the external factors that enable people to make those contributions.

Section 8

Do I need a legal entity to support my project?

You don't need a legal entity to support your open source project unless you're handling money.

For example, if you want to create a commercial business, you'll want to set up a C Corp or LLC (if you're based in the US). If you're just doing contract work related to your open source project, you can accept money as a sole proprietor, or set up an LLC (if you're based in the US).

If you want to accept donations for your open source project, you can set up a donation button (using PayPal or Stripe, for example), but the money won't be tax-deductible unless you are a qualifying nonprofit (a 501c3, if you're in the US).

Many projects don't wish to go through the trouble of setting up a nonprofit, so they find a nonprofit fiscal sponsor instead. A fiscal sponsor accepts donations on your behalf, usually in exchange for a percentage of the donation. Software Freedom Conservancy, Apache Foundation, Eclipse Foundation, Linux Foundation and Open Collective are examples of organizations that serve as fiscal sponsors for open source projects.

Our goal is to provide an infrastructure that communities can use to be self sustainable, thus creating an environment where everyone—contributors, backers, sponsors—get concrete benefits out of it.

— @piamancini, "Moving beyond the charity framework"

If your project is closely associated with a certain language or ecosystem, there may also be a related software foundation you can work with. For example, the Python Software Foundation helps support PyPI, the Python package manager, and the Node.js Foundation helps support Express.js, a Node-based framework.

Back to all guides

Related Guides

Best Practices for Maintainers

Making your life easier as an open source maintainer, from documenting processes to leveraging your community.

Open Source Metrics

Make informed decisions to help your open source project thrive by measuring and tracking its success.

Contribute

Stay in touch

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Subscribe

fine print

Getting Paid for Open Source Work

Sustain your work in open source by getting financial support for your time or your project.

Table of Contents

Section 1

Why some people seek financial support

Much of the work of open source is voluntary. For example, someone might come across a bug in a project they use and submit a quick fix, or they might enjoy tinkering with an open source project in their spare time.

I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. (...) I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately. (...) I chose Python as a working title.

— @gvanrossum, "Programming Python"

There are many reasons why a person would not want to be paid for their open source work.

- **They may already have a full-time job that they love,** which enables them to contribute to open source in their spare time.
- **They enjoy thinking of open source as a hobby** or creative escape and don't want to feel financially obligated to work on their projects.
- **They get other benefits from contributing to open source,** such as building their reputation or portfolio, learning a new skill, or feeling closer to a community.

Financial donations do add a feeling of responsibility, for some. (...) It's important for us, in the globally connected, fast-paced world we live in, to be able to say "not now, I feel like doing something completely different".

— @alloy, "Why We Don't Accept Donations"

For others, especially when contributions are ongoing or require significant time, getting paid to contribute to open source is the only way they can participate, either because the project requires it, or for personal reasons.

Maintaining popular projects can be a significant responsibility, taking up 10 or 20 hours per week instead of a few hours per month.

> Ask any open source project maintainer, and they will tell you about the reality of the amount of work that goes into managing a project. You have clients. You are fixing issues for them. You are creating new features. This becomes a real demand on your time.
>
> — @ashedryden, "The Ethics of Unpaid Labor and the OSS Community"

Paid work also enables people from different walks of life to make meaningful contributions. Some people cannot afford to spend unpaid time on open source projects, based on their current financial position, debt, or family or other caretaking obligations. That means the world never sees contributions from talented people who can't afford to volunteer their time. This has ethical implications, as @ashedryden has described, since work that is done is biased in favor of those who already have advantages in life, who then gain additional advantages based on their volunteer contributions, while others who are not able to volunteer then don't get later opportunities, which reinforces the current lack of diversity in the open source community.

> OSS yields massive benefits to the technology industry, which, in turn, means benefits to all industries. (...) However, if the only people who can focus on it are the lucky and the obsessed, then there's a huge untapped potential.
>
> — @isaacs, "Money and Open Source"

If you're looking for financial support, there are two paths to consider. You can fund your own time as a contributor, or you can find organizational funding for the project.

Section 2

Funding your own time

Today, many people get paid to work part- or full-time on open source. The most common way to get paid for your time is to talk to your employer.

It's easier to make a case for open source work if your employer actually uses the project, but get creative with your pitch. Maybe your employer doesn't use the project, but they use Python, and maintaining a popular Python project help attract new Python developers. Maybe it makes your employer look more developer-friendly in general.

If you don't have an existing open source project you'd like to work on, but would rather that your current work output is open sourced, make a case for your employer to open source some of their internal software.

Many companies are developing open source programs to build their brand and recruit quality talent.

@hueniverse, for example, found that there were financial reasons to justify Walmart's investment in open source. And @jamesgpearce found that Facebook's open source program made a difference in recruiting:

> It is closely aligned with our hacker culture, and how our organization was perceived. We asked our employees, "Were you aware of the open source software program at Facebook?". Two-thirds said "Yes". One-half said that the program positively contributed to their decision to work for us. These are not marginal numbers, and I hope, a trend that continues.

If your company goes down this route, it's important to keep the boundaries between community and corporate activity clear. Ultimately, open source sustains itself through contributions from people all over the world, and that's bigger than any one company or location.

> Getting paid to work on open source is a rare and wonderful opportunity, but you should not have to give up your passion in the process. Your passion should be why companies want to pay you.
>
> — @jessfraz, "Blurred Lines"

If you can't convince your current employer to prioritize open source work, consider finding a new employer that encourages employee contributions to open source. Look for companies that make their dedication to open source work explicit. For example:

- Some companies, like Netflix or PayPal, have websites that highlight their involvement in open source
- Zalando published its open source contribution policy for employees

Projects that originated at a large company, such as Go or React, will also likely employ people to work on open source.

Depending on your personal circumstances, you can try raising money independently to fund your open source work. For example:

- @Homebrew (and many other maintainers and organizations) fund their work through GitHub Sponsors
- @gaearon funded his work on Redux through a Patreon crowdfunding campaign
- @andrewgodwin funded work on Django schema migrations through a Kickstarter campaign

Finally, sometimes open source projects put bounties on issues that you might consider helping with.

- @ConnorChristie was able to get paid for helping @MARKETProtocol work on their JavaScript library through a bounty on gitcoin.
- @mamiM did Japanese translations for @MetaMask after the issue was funded on Bounties Network.

Section 3

Finding funding for your project

Beyond arrangements for individual contributors, sometimes projects raise money from companies, individuals, or others to fund ongoing work.

Organizational funding might go towards paying current contributors, covering the costs of running the project (such as hosting fees), or investing in new features or ideas.

As open source's popularity increases, finding funding for projects is still experimental, but there are a few common options available.

Raise money for your work through crowdfunding campaigns or sponsorships

Finding sponsorships works well if you have a strong audience or reputation already, or your project is very popular. A few examples of sponsored projects include:

- **webpack** raises money from companies and individuals through OpenCollective
- **Ruby Together,** a nonprofit organization that pays for work on bundler, RubyGems, and other Ruby infrastructure projects

Create a revenue stream

Depending on your project, you may be able to charge for commercial support, hosted options, or additional features. A few examples include:

- **Sidekiq** offers paid versions for additional support
- **Travis CI** offers paid versions of its product
- **Ghost** is a nonprofit with a paid managed service

Some popular projects, like npm and Docker, even raise venture capital to support their business growth.

Apply for grant funding

Some software foundations and companies offer grants for open source work. Sometimes, grants can be paid out to individuals without setting up a legal entity for the project.

- **Read the Docs** received a grant from Mozilla Open Source Support
- **OpenMRS** work was funded by Stripe's Open-Source Retreat
- **Libraries.io** received a grant from the Sloan Foundation
- The **Python Software Foundation** offers grants for Python-related work

For more detailed options and case studies, @nayafia wrote a guide to getting paid for open source work. Different types of funding require different skills, so consider your strengths to

figure out which option works best for you.

Section 4

Building a case for financial support

Whether your project is a new idea, or has been around for years, you should expect to put significant thought into identifying your target funder and making a compelling case.

Whether you're looking to pay for your own time, or fundraise for a project, you should be able to answer the following questions.

Impact

Why is this project useful? Why do your users, or potential users, like it so much? Where will it be in five years?

Traction

Try to collect evidence that your project matters, whether it's metrics, anecdotes, or testimonials. Are there any companies or noteworthy people using your project right now? If not, has a prominent person endorsed it?

Value to funder

Funders, whether your employer or a grantmaking foundation, are frequently approached with opportunities. Why should they support your project over any other opportunity? How do they personally benefit?

Use of funds

What, exactly, will you accomplish with the proposed funding? Focus on project milestones or outcomes rather than paying a salary.

How you'll receive the funds

Does the funder have any requirements around disbursal? For example, you may need to be a nonprofit or have a nonprofit fiscal sponsor. Or perhaps the funds must be given to an individual contractor rather than an organization. These requirements vary between funders, so be sure to do your research beforehand.

> For years, we've been the leading resource of website friendly icons, with a community of over 20 million people and been featured on over 70 million websites, including Whitehouse.gov. (...) Version 4 was three years ago. Web tech's changed a lot since then, and frankly, Font Awesome's gotten a bit stale. (...) That's why we're introducing Font Awesome 5. We're modernizing and rewriting the CSS and redesigning every icon from top to bottom. We're talking better design, better consistency, and better readability.
>
> — @davegandy, Font Awesome Kickstarter video

Section 5

Experiment and don't give up

Raising money isn't easy, whether you're an open source project, a nonprofit, or a software startup, and in most cases require you to get creative. Identifying how you want to get paid, doing your research, and putting yourself in your funder's shoes will help you build a convincing case for funding.

Back to all guides

Related Guides

Best Practices for Maintainers

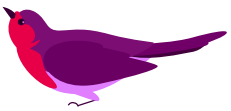
Making your life easier as an open source maintainer, from documenting processes to leveraging your community.

Leadership and Governance

Growing open source projects can benefit from formal rules for making decisions.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.



Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Contribute

Subscribe

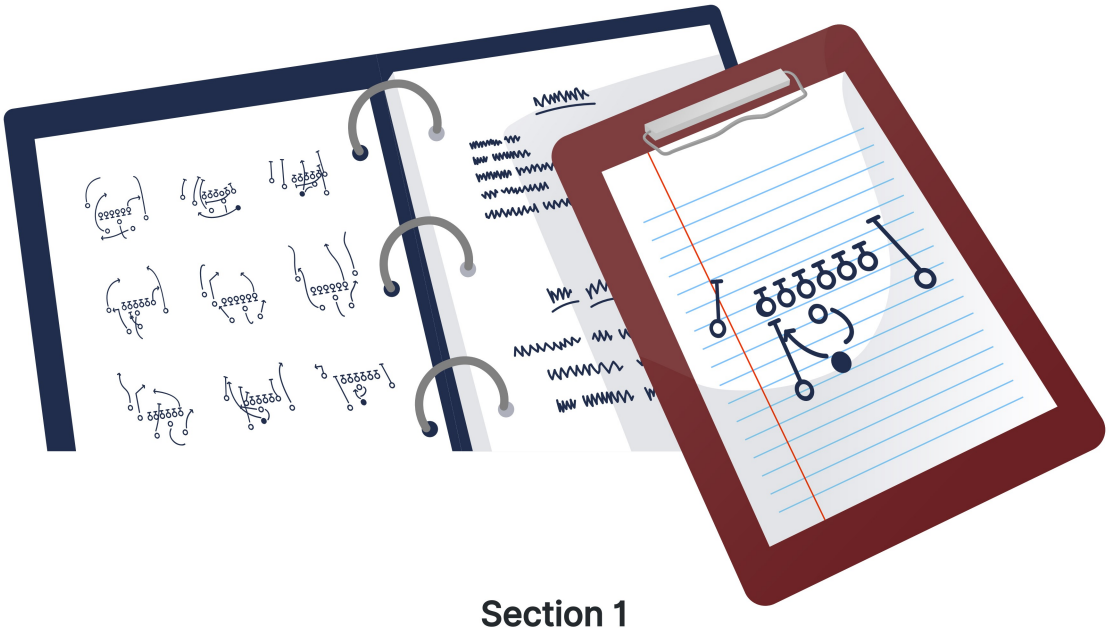
fine print

with ♥ by and friends

Your Code of Conduct

Facilitate healthy and constructive community behavior by adopting and enforcing a code of conduct.

Table of Contents



Section 1

Why do I need a code of conduct?

A code of conduct is a document that establishes expectations for behavior for your project's participants. Adopting, and enforcing, a code of conduct can help create a positive social atmosphere for your community.

Codes of conduct help protect not just your participants, but yourself. If you maintain a project, you may find that unproductive attitudes from other participants can make you feel drained or unhappy about your work over time.

A code of conduct empowers you to facilitate healthy, constructive community behavior. Being proactive reduces the likelihood that you, or others, will become fatigued with your project, and helps you take action when someone does something you don't agree with.

Section 2

Establishing a code of conduct

Try to establish a code of conduct as early as possible: ideally, when you first create your project.

In addition to communicating your expectations, a code of conduct describes the following:

- Where the code of conduct takes effect *(only on issues and pull requests, or community activities like events?)*
- Whom the code of conduct applies to *(community members and maintainers, but what about sponsors?)*
- What happens if someone violates the code of conduct
- How someone can report violations

Wherever you can, use prior art. The Contributor Covenant is a drop-in code of conduct that is used by over 40,000 open source projects, including Kubernetes, Rails, and Swift.

The Django Code of Conduct and the Citizen Code of Conduct are also two good code of conduct examples.

Place a CODE_OF_CONDUCT file in your project's root directory, and make it visible to your community by linking it from your CONTRIBUTING or README file.

Section 3

Deciding how you'll enforce your code of conduct

A code of conduct that isn't (or can't be) enforced is worse than no code of conduct at all: it sends the message that the values in the code of conduct aren't actually important or respected in your community.

— Ada Initiative

You should explain how your code of conduct will be enforced ***before*** a violation occurs. There are several reasons to do so:

- It demonstrates that you are serious about taking action when it's needed.
- Your community will feel more reassured that complaints actually get reviewed.
- You'll reassure your community that the review process is fair and transparent, should they ever find themselves investigated for a violation.

You should give people a private way (such as an email address) to report a code of conduct violation and explain who receives that report. It could be a maintainer, a group of maintainers, or a code of conduct working group.

Don't forget that someone might want to report a violation about a person who receives those reports. In this case, give them an option to report violations to someone else. For example, @ctb and @mr-c explain on their project, khmer:

> Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by emailing **khmer-project@idyll.org** which only goes to C. Titus Brown and Michael R. Crusoe. To report an issue involving either of them please email **Judi Brown Clarke, Ph.D.** the Diversity Director at the BEACON Center for the Study of Evolution in Action, an NSF Center for Science and Technology.*

For inspiration, check out Django's enforcement manual (though you may not need something this comprehensive, depending on the size of your project).

Section 4

Enforcing your code of conduct

Sometimes, despite your best efforts, somebody will do something that violates this code. There are several ways to address negative or harmful behavior when it comes up.

Gather information about the situation

Treat each community member's voice as important as your own. If you receive a report that someone violated the code of conduct, take it seriously and investigate the matter, even if it does not match your own experience with that person. Doing so signals to your community that you value their perspective and trust their judgment.

The community member in question may be a repeat offender who consistently makes others feel uncomfortable, or they may have only said or done something once. Both can be grounds for taking action, depending on context.

Before you respond, give yourself time to understand what happened. Read through the person's past comments and conversations to better understand who they are and why they might have acted in such a way. Try to gather perspectives other than your own about this person and their behavior.

> Don't get pulled into an argument. Don't get sidetracked into dealing with someone else's behavior before you've finished dealing with the matter at hand. Focus on what you need.
>
> — Stephanie Zvan, "So You've Got Yourself a Policy. Now What?"

Take appropriate action

After gathering and processing sufficient information, you'll need to decide what to do. As you consider your next steps, remember that your goal as a moderator is to foster a safe, respectful, and collaborative environment. Consider not only how to deal with the situation in question, but how your response will affect the rest of your community's behavior and expectations moving forward.

When somebody reports a code of conduct violation, it is your, not their, job to handle it. Sometimes, the reporter is disclosing information at great risk to their career, reputation, or physical safety. Forcing them to confront their harasser could put the reporter in a compromising position. You should handle direct communication with the person in question, unless the reporter explicitly requests otherwise.

There are a few ways you might respond to a code of conduct violation:

- **Give the person in question a public warning** and explain how their behavior negatively impacted others, preferably in the channel where it occurred. Where possible, public communication conveys to the rest of the community that you take the code of conduct seriously. Be kind, but firm in your communication.
- **Privately reach out to the person** in question to explain how their behavior negatively impacted others. You may want to use a private communication channel if the situation

involves sensitive personal information. If you communicate with someone privately, it's a good idea to CC those who first reported the situation, so they know you took action. Ask the reporting person for consent before CCing them.

Sometimes, a resolution cannot be reached. The person in question may become aggressive or hostile when confronted or does not change their behavior. In this situation, you may want to consider taking stronger action. For example:

- **Suspend the person** in question from the project, enforced through a temporary ban on participating in any aspect of the project
- **Permanently ban** the person from the project

Banning members should not be taken lightly and represents a permanent and irreconcilable difference of perspectives. You should only take these measures when it is clear that a resolution cannot be reached.

Section 5

Your responsibilities as a maintainer

A code of conduct is not a law that is enforced arbitrarily. You are the enforcer of the code of conduct and it's your responsibility to follow the rules that the code of conduct establishes.

As a maintainer you establish the guidelines for your community and enforce those guidelines according to the rules set forth in your code of conduct. This means taking any report of a code of conduct violation seriously. The reporter is owed a thorough and fair review of their complaint. If you determine that the behavior that they reported is not a violation, communicate that clearly to them and explain why you're not going to take action on it. What they do with that is up to them: tolerate the behavior that they had an issue with, or stop participating in the community.

A report of behavior that doesn't *technically* violate the code of conduct may still indicate that there is a problem in your community, and you should investigate this potential problem and act accordingly. This may include revising your code of conduct to clarify acceptable behavior and/or talking to the person whose behavior was reported and telling them that while they did not violate the code of conduct, they are skirting the edge of what is expected and are making certain participants feel uncomfortable.

In the end, as a maintainer, you set and enforce the standards for acceptable behavior. You have the ability to shape the community values of the project, and participants expect you to enforce those values in a fair and even-handed way.

Section 6

Encourage the behavior you want to see in the world 🌎

When a project seems hostile or unwelcoming, even if it's just one person whose behavior is tolerated by others, you risk losing many more contributors, some of whom you may never even meet. It's not always easy to adopt or enforce a code of conduct, but fostering a welcoming environment will help your community grow.

Back to all guides

Related Guides

Building Welcoming Communities

Building a community that encourages people to use, contribute to, and evangelize your project.

Leadership and Governance

Growing open source projects can benefit from formal rules for making decisions.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Subscribe

Open Source Metrics

Make informed decisions to help your open source project thrive by measuring and tracking its success.

Table of Contents

Section 1

Why measure anything?

Data, when used wisely, can help you make better decisions as an open source maintainer.

With more information, you can:

- Understand how users respond to a new feature
- Figure out where new users come from
- Identify, and decide whether to support, an outlier use case or functionality
- Quantify your project's popularity
- Understand how your project is used
- Raise money through sponsorships and grants

For example, Homebrew finds that Google Analytics helps them prioritize work:

> Homebrew is provided free of charge and run entirely by volunteers in their spare time. As a result, we do not have the resources to do detailed user studies of Homebrew users to decide on how best to design future features and prioritise current work. Anonymous aggregate user analytics allow us to prioritise fixes and features based on how, where and when people use Homebrew.

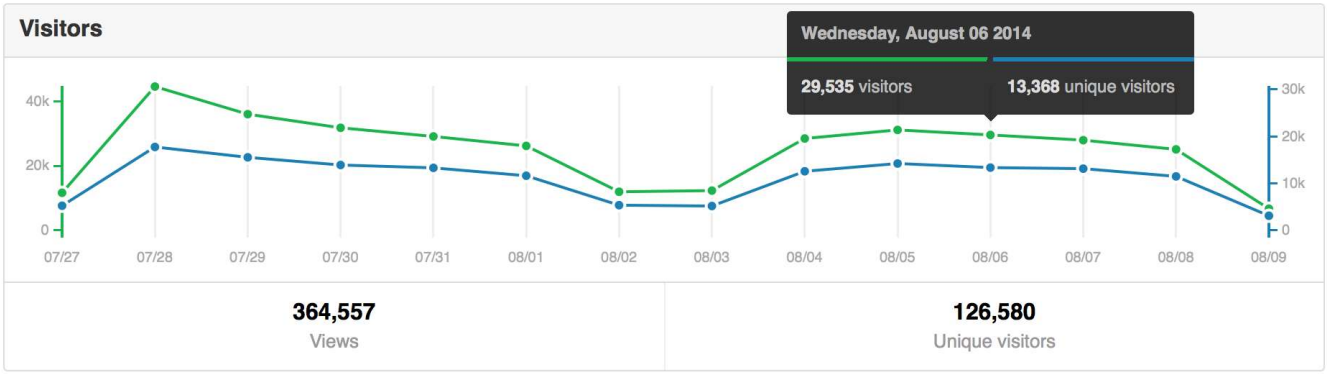
Popularity isn't everything. Everybody gets into open source for different reasons. If your goal as an open source maintainer is to show off your work, be transparent about your code, or just have fun, metrics may not be important to you.

If you *are* interested in understanding your project on a deeper level, read on for ways to analyze your project's activity.

Section 2

Discovery

Before anybody can use or contribute back to your project, they need to know it exists. Ask yourself: *are people finding this project?*



If your project is hosted on GitHub, you can view how many people land on your project and where they come from. From your project's page, click "Insights", then "Traffic". On this page, you can see:

- **Total page views:** Tells you how many times your project was viewed
- **Total unique visitors:** Tells you how many people viewed your project
- **Referring sites:** Tells you where visitors came from. This metric can help you figure out where to reach your audience and whether your promotion efforts are working.
- **Popular content:** Tells you where visitors go on your project, broken down by page views and unique visitors.

GitHub stars can also help provide a baseline measure of popularity. While GitHub stars don't necessarily correlate to downloads and usage, they can tell you how many people are taking notice of your work.

You may also want to track discoverability in specific places: for example, Google PageRank, referral traffic from your project's website, or referrals from other open source projects or websites.

Section 3

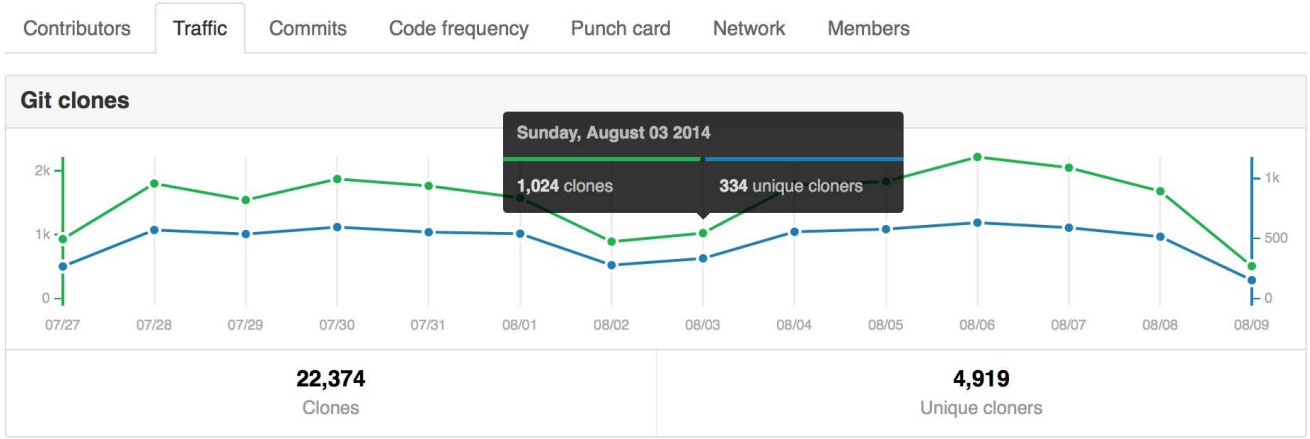
Usage

People are finding your project on this wild and crazy thing we call the internet. Ideally, when they see your project, they'll feel compelled to do something. The second question you'll want to ask is: *are people using this project?*

If you use a package manager, such as npm or RubyGems.org, to distribute your project, you may be able to track your project's downloads.

Each package manager may use a slightly different definition of "download", and downloads do not necessarily correlate to installs or use, but it provides some baseline for comparison. Try using Libraries.io to track usage statistics across many popular package managers.

If your project is on GitHub, navigate again to the "Traffic" page. You can use the clone graph to see how many times your project has been cloned on a given day, broken down by total clones and unique cloners.



If usage is low compared to the number of people discovering your project, there are two issues to consider. Either:

- Your project isn't successfully converting your audience, or
- You're attracting the wrong audience

For example, if your project lands on the front page of Hacker News, you'll probably see a spike in discovery (traffic), but a lower conversion rate, because you're reaching everyone on Hacker News. If your Ruby project is featured at a Ruby conference, however, you're more likely to see a high conversion rate from a targeted audience.

Try to figure out where your audience is coming from and ask others for feedback on your project page to figure out which of these two issues you're facing.

Once you know that people are using your project, you might want to try to figure out what they are doing with it. Are they building on it by forking your code and adding features? Are they using it for science or business?

Section 4

Retention

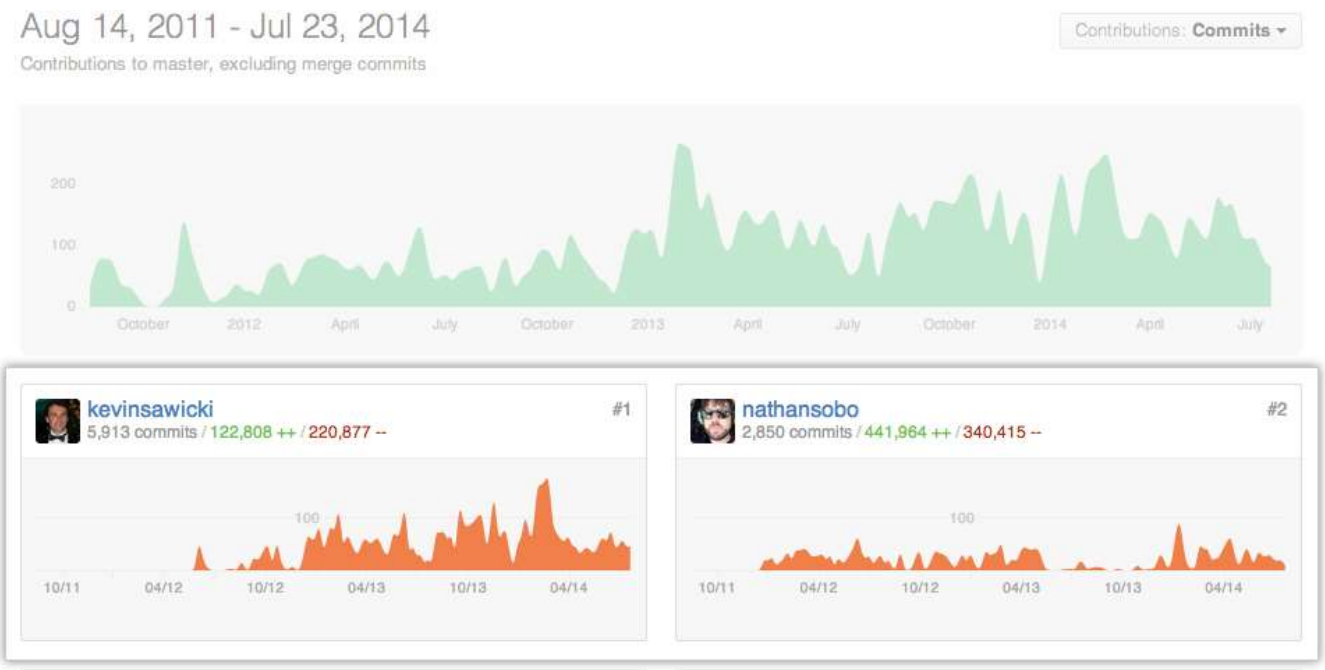
People are finding your project and they're using it. The next question you'll want to ask yourself is: *are people contributing back to this project?*

It's never too early to start thinking about contributors. Without other people pitching in, you risk putting yourself into an unhealthy situation where your project is *popular* (many people use it) but not *supported* (not enough maintainer time to meet demand).

Retention also requires an inflow of new contributors, as previously active contributors will eventually move on to other things.

Examples of community metrics that you may want to regularly track include:

- **Total contributor count and number of commits per contributor:** Tells you how many contributors you have, and who's more or less active. On GitHub, you can view this under "Insights" -> "Contributors." Right now, this graph only counts contributors who have committed to the default branch of the repository.



- **First time, casual, and repeat contributors:** Helps you track whether you're getting new contributors, and whether they come back. (Casual contributors are contributors with a low number of commits. Whether that's one commit, less than five commits, or something else is up to you.) Without new contributors, your project's community can become stagnant.

- **Number of open issues and open pull requests:** If these numbers get too high, you might need help with issue triaging and code reviews.
- **Number of *opened* issues and *opened* pull requests:** Opened issues means somebody cares enough about your project to open an issue. If that number increases over time, it suggests people are interested in your project.
- **Types of contributions:** For example, commits, fixing typos or bugs, or commenting on an issue.

Open source is more than just code. Successful open source projects include code and documentation contributions together with conversations about these changes.

— @arfon, "The Shape of Open Source"

Section 5

Maintainer activity

Finally, you'll want to close the loop by making sure your project's maintainers are able to handle the volume of contributions received. The last question you'll want to ask yourself is: *am I (or are we) responding to our community?*

Unresponsive maintainers become a bottleneck for open source projects. If someone submits a contribution but never hears back from a maintainer, they may feel discouraged and leave.

Research from Mozilla suggests that maintainer responsiveness is a critical factor in encouraging repeat contributions.

Consider tracking how long it takes for you (or another maintainer) to respond to contributions, whether an issue or a pull request. Responding doesn't require taking action. It can be as simple as saying: "*Thanks for your submission! I'll review this within the next week.*"

You could also measure the time it takes to move between stages in the contribution process, such as:

- Average time an issue remains open
- Whether issues get closed by PRs
- Whether stale issues get closed
- Average time to merge a pull request

Section 6

Use 📊 to learn about people

Understanding metrics will help you build an active, growing open source project. Even if you don't track every metric on a dashboard, use the framework above to focus your attention on the type of behavior that will help your project thrive.

CHAOSS is a welcoming, open source community focused on analytics, metrics and software for community health.

Back to all guides

Related Guides

Finding Users for Your Project

Help your open source project grow by getting it in the hands of happy users.

Best Practices for Maintainers

Making your life easier as an open source maintainer, from documenting processes to leveraging your community.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

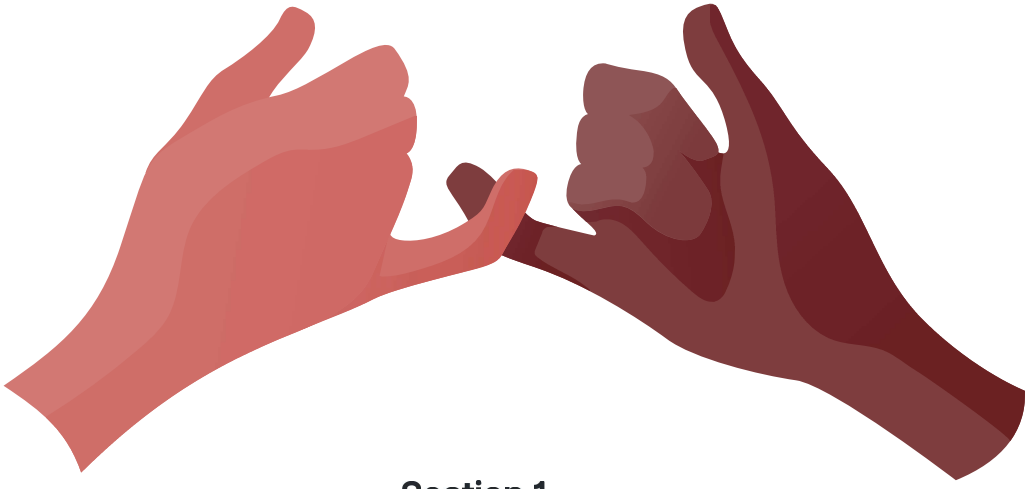
Email Address

Subscribe

The Legal Side of Open Source

Everything you've ever wondered about the legal side of open source, and a few things you didn't.

Table of Contents



Section 1

Understanding the legal implications of open source

Sharing your creative work with the world can be an exciting and rewarding experience. It can also mean a bunch of legal things you didn't know you had to worry about. Thankfully, you don't have to start from scratch. We've got your legal needs covered. (Before you dig in, be sure to read our disclaimer.)

Section 2

Why do people care so much about the legal side of open source?

Glad you asked! When you make a creative work (such as writing, graphics, or code), that work is under exclusive copyright by default. That is, the law assumes that as the author of your work, you have a say in what others can do with it.

In general, that means nobody else can use, copy, distribute, or modify your work without being at risk of take-downs, shake-downs, or litigation.

Open source is an unusual circumstance, however, because the author expects that others will use, modify, and share the work. But because the legal default is still exclusive copyright, you need a license that explicitly states these permissions.

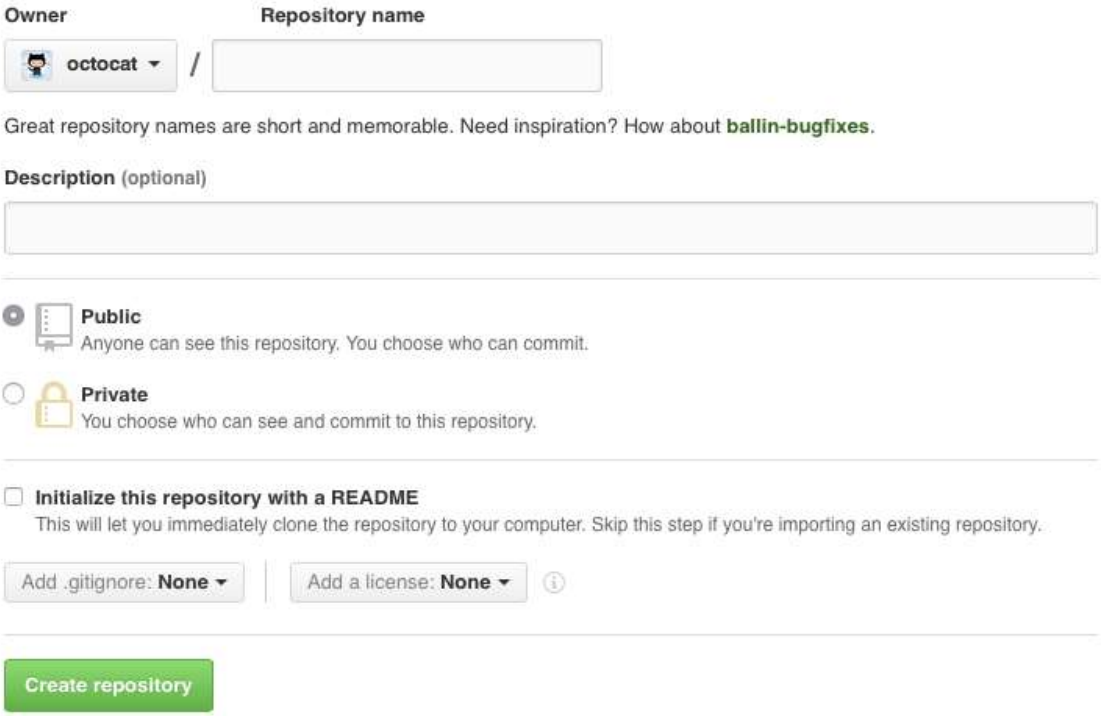
If you don't apply an open source license, everybody who contributes to your project also becomes an exclusive copyright holder of their work. That means nobody can use, copy, distribute, or modify their contributions – and that "nobody" includes you.

Finally, your project may have dependencies with license requirements that you weren't aware of. Your project's community, or your employer's policies, may also require your project to use specific open source licenses. We'll cover these situations below.

Section 3

Are public GitHub projects open source?

When you create a new project on GitHub, you have the option to make the repository **private** or **public**.



Making your GitHub project public is not the same as licensing your project. Public projects are covered by GitHub's Terms of Service, which allows others to view and fork your project, but your work otherwise comes with no permissions.

If you want others to use, distribute, modify, or contribute back to your project, you need to include an open source license. For example, someone cannot legally use any part of your GitHub project in their code, even if it's public, unless you explicitly give them the right to do so.

Section 4

Just give me the TL;DR on what I need to protect my project.

You're in luck, because today, open source licenses are standardized and easy to use. You can copy-paste an existing license directly into your project.

MIT, Apache 2.0, and GPLv3 are the most popular open source licenses, but there are other options to choose from. You can find the full text of these licenses, and instructions on how to use them, on choosealicense.com.

When you create a new project on GitHub, you'll be asked to add a license.

A standardized license serves as a proxy for those without legal training to know precisely what they can and can't do with the software. Unless absolutely required, avoid custom, modified, or non-standard terms, which will serve as a barrier to downstream use of the agency code.

— @benbalter, "Everything a government attorney needs to know about open source software licensing"

Section 5

Which open source license is appropriate for my project?

If you're starting from a blank slate, it's hard to go wrong with the MIT License. It's short, very easy to understand, and allows anyone to do anything so long as they keep a copy of the license, including your copyright notice. You'll be able to release the project under a different license if you ever need to.

Otherwise, picking the right open source license for your project depends on your objectives.

Your project very likely has (or will have) **dependencies**. For example, if you're open sourcing a Node.js project, you'll probably use libraries from the Node Package Manager (npm). Each of those libraries you depend on will have its own open source license. If each of their licenses is "permissive" (gives the public permission to use, modify, and share, without any condition for downstream licensing), you can use any license you want. Common permissive licenses include MIT, Apache 2.0, ISC, and BSD.

On the other hand, if any of your dependencies' licenses are "strong copyleft" (also gives public same permissions, subject to condition of using the same license downstream), then your project will have to use the same license. Common strong copyleft licenses include GPLv2, GPLv3, and AGPLv3.

You may also want to consider the **communities** you hope will use and contribute to your project:

- **Do you want your project to be used as a dependency by other projects?** Probably best to use the most popular license in your relevant community. For example, MIT is the most popular license for npm libraries.
- **Do you want your project to appeal to large businesses?** A large business will likely want an express patent license from all contributors. In this case, Apache 2.0 has you (and them) covered.
- **Do you want your project to appeal to contributors who do not want their contributions to be used in closed source software?** GPLv3 or (if they also do not wish to contribute to closed source services) AGPLv3 will go over well.

Your **company** may have specific licensing requirements for its open source projects. For example, it may require a permissive license so that the company can use your project in the company's closed source product. Or your company may require a strong copyleft license and an additional contributor agreement (see below) so that only your company, and nobody else, can use your project in closed source software. Or your company may have certain needs related to standards, social responsibility, or transparency, any of which could require a particular licensing strategy. Talk to your company's legal department.

When you create a new project on GitHub, you are given the option to select a license. Including one of the licenses mentioned above will make your GitHub project open source. If you'd like to see other options, check out choosealicense.com to find the right license for your project, even if it isn't software.

Section 6

What if I want to change the license of my project?

Most projects never need to change licenses. But occasionally circumstances change.

For example, as your project grows it adds dependencies or users, or your company changes strategies, any of which could require or want a different license. Also, if you neglected to license your project from the start, adding a license is effectively the same as changing licenses. There are three fundamental things to consider when adding or changing your project's license:

It's complicated. Determining license compatibility and compliance and who holds copyright can get complicated and confusing very quickly. Switching to a new but compatible license for new releases and contributions is different from relicensing all existing contributions. Involve your legal team at the first hint of any desire to change licenses. Even if you have or can obtain permission from your project's copyright holders for a license change, consider the impact of the change on your project's other users and contributors. Think of a license change as a "governance event" for your project that will more likely go smoothly with clear communications and consultation with your project's stakeholders. All the more reason to choose and use an appropriate license for your project from its inception!

Your project's existing license. If your project's existing license is compatible with the license you want to change to, you could just start using the new license. That's because if license A is compatible with license B, you'll comply with the terms of A while complying with the terms of B (but not necessarily vice versa). So if you're currently using a permissive license (e.g., MIT), you could change to a license with more conditions, so long as you retain a copy of the MIT license and any associated copyright notices (i.e., continue to comply with the MIT license's minimal conditions). But if your current license is not permissive (e.g., copyleft, or you don't have a license) and you aren't the sole copyright holder, you couldn't just change your project's license to MIT. Essentially, with a permissive license the project's copyright holders have given permission in advance to change licenses.

Your project's existing copyright holders. If you're the sole contributor to your project then either you or your company is the project's sole copyright holder. You can add or change to whatever license you or your company wants to. Otherwise there may be other copyright holders that you need agreement from in order to change licenses. Who are they? People who have commits in your project is a good place to start. But in some cases copyright will be held by those people's employers. In some cases people will have only made minimal contributions, but there's no hard and fast rule that contributions under some number of lines of code are not subject to copyright. What to do? It depends. For a relatively small and young project, it may be feasible to get all existing contributors to agree to a license change in an issue or pull request. For large and long-lived projects, you may have to seek out many contributors and even their heirs. Mozilla took years (2001-2006) to relicense Firefox, Thunderbird, and related software.

Alternatively, you can have contributors agree in advance (via an additional contributor agreement – see below) to certain license changes under certain conditions, beyond those allowed by your existing open source license. This shifts the complexity of changing licenses a

bit. You'll need more help from your lawyers up front, and you'll still want to clearly communicate with your project's stakeholders when executing a license change.

Section 7

Does my project need an additional contributor agreement?

Probably not. For the vast majority of open source projects, an open source license implicitly serves as both the inbound (from contributors) and outbound (to other contributors and users) license. If your project is on GitHub, the GitHub Terms of Service make "inbound=outbound" the explicit default.

An additional contributor agreement – often called a Contributor License Agreement (CLA) – can create administrative work for project maintainers. How much work an agreement adds depends on the project and implementation. A simple agreement might require that contributors confirm, with a click, that they have the rights necessary to contribute under the project open source license. A more complicated agreement might require legal review and sign-off from contributors' employers.

Also, by adding "paperwork" that some believe is unnecessary, hard to understand, or unfair (when the agreement recipient gets more rights than contributors or the public do via the project's open source license), an additional contributor agreement may be perceived as unfriendly to the project's community.

We have eliminated the CLA for Node.js. Doing this lowers the barrier to entry for Node.js contributors thereby broadening the contributor base.

— @bcantrill, "Broadening Node.js Contributions"

Some situations where you may want to consider an additional contributor agreement for your project include:

- Your lawyers want all contributors to expressly accept (*sign*, online or offline) contribution terms, perhaps because they feel the open source license itself is not enough (even though it is!). If this is the only concern, a contributor agreement that affirms the project's open source license should be enough. The jQuery Individual Contributor License Agreement is a good example of a lightweight additional contributor agreement.
- You or your lawyers want developers to represent that each commit they make is authorized. A Developer Certificate of Origin requirement is how many projects achieve this. For example, the Node.js community uses the DCO instead of their prior CLA. A simple option to automate enforcement of the DCO on your repository is the DCO Probot.
- Your project uses an open source license that does not include an express patent grant (such as MIT), and you need a patent grant from all contributors, some of whom may work for companies with large patent portfolios that could be used to target you or the project's other contributors and users. The Apache Individual Contributor License Agreement is a commonly used additional contributor agreement that has a patent grant mirroring the one found in the Apache License 2.0.
- Your project is under a copyleft license, but you also need to distribute a proprietary version of the project. You'll need every contributor to assign copyright to you or grant you (but not the public) a permissive license. The MongoDB Contributor Agreement is an example this type of agreement.
- You think your project might need to change licenses over its lifetime and want contributors to agree in advance to such changes.

If you do need to use an additional contributor agreement with your project, consider using an integration such as CLA assistant to minimize contributor distraction.

Section 8

What does my company's legal team need to know?

If you're releasing an open source project as a company employee, first, your legal team should know that you're open sourcing a project.

For better or worse, consider letting them know even if it's a personal project. You probably have an "employee IP agreement" with your company that gives them some control of your

projects, especially if they are at all related to the company's business or you use any company resources to develop the project. Your company *should* easily give you permission, and maybe already has through an employee-friendly IP agreement or a company policy. If not, you can negotiate (for example, explain that your project serves the company's professional learning and development objectives for you), or avoid working on your project until you find a better company.

If you're open sourcing a project for your company, then definitely let them know. Your legal team probably already has policies for what open source license (and maybe additional contributor agreement) to use based on the company's business requirements and expertise around ensuring your project complies with the licenses of its dependencies. If not, you and they are in luck! Your legal team should be eager to work with you to figure this stuff out. Some things to think about:

- **Third party material:** Does your project have dependencies created by others or otherwise include or use others' code? If these are open source, you'll need to comply with the materials' open source licenses. That starts with choosing a license that works with the third party open source licenses (see above). If your project modifies or distributes third party open source material, then your legal team will also want to know that you're meeting other conditions of the third party open source licenses such as retaining copyright notices. If your project uses others' code that doesn't have an open source license, you'll probably have to ask the third party maintainers to add an open source license, and if you can't get one, stop using their code in your project.
- **Trade secrets:** Consider whether there is anything in the project that the company does not want to make available to the general public. If so, you could open source the rest of your project, after extracting the material you want to keep private.
- **Patents:** Is your company applying for a patent of which open sourcing your project would constitute public disclosure? Sadly, you might be asked to wait (or maybe the company will reconsider the wisdom of the application). If you're expecting contributions to your project from employees of companies with large patent portfolios, your legal team may want you to use a license with an express patent grant from contributors (such as Apache 2.0 or GPLv3), or an additional contributor agreement (see above).
- **Trademarks:** Double check that your project's name does not conflict with any existing trademarks. If you use your own company trademarks in the project, check that it does not cause any conflicts. FOSSmarks is a practical guide to understanding trademarks in the context of free and open source projects.
- **Privacy:** Does your project collect data on users? "Phone home" to company servers? Your legal team can help you comply with company policies and external regulations.

If you're releasing your company's first open source project, the above is more than enough to get through (but don't worry, most projects shouldn't raise any major concerns).

Longer term, your legal team can do more to help the company get more from its involvement in open source, and stay safe:

- **Employee contribution policies:** Consider developing a corporate policy that specifies how your employees contribute to open source projects. A clear policy will reduce confusion among your employees and help them contribute to open source projects in the company's best interest, whether as part of their jobs or in their free time. A good example is Rackspace's Model IP and Open Source Contribution Policy.

> Letting out the IP associated with a patch builds the employee's knowledge base and reputation. It shows that the company is invested in the development of that employee and creates a sense of empowerment and autonomy. All of these benefits also lead to higher morale and better employee retention.
>
> — @vanl, "A Model IP and Open Source Contribution Policy"

- **What to release:** (Almost) everything? If your legal team understands and is invested in your company's open source strategy, they'll be best able to help rather than hinder your efforts.
- **Compliance:** Even if your company doesn't release any open source projects, it uses others' open source software. Awareness and process can prevent headaches, product delays, and lawsuits.

> Organizations must have a license and compliance strategy in place that fits both ["permissive" and "copyleft"] categories. This begins with keeping a record of the licensing terms that apply to the open source software you're using — including subcomponents and dependencies.
>
> — Heather Meeker, "Open Source Software: Compliance Basics And Best Practices"

- **Patents:** Your company may wish to join the Open Invention Network, a shared defensive patent pool to protect members' use of major open source projects, or explore other alternative patent licensing.

- **Governance:** Especially if and when it makes sense to move a project to a legal entity outside of the company.

Back to all guides

Related Guides

How to Contribute to Open Source

Want to contribute to open source? A guide to making open source contributions, for first-timers and for veterans.

Leadership and Governance

Growing open source projects can benefit from formal rules for making decisions.

Contribute

Want to make a suggestion? This content is open source. Help us improve it.

Contribute

Stay in touch

Be the first to hear about GitHub's latest open source tips and resources.

Email Address

Subscribe