THE LINUX FOUNDATION | Research

Recommended Practices for Hosting and Managing Open Source Projects on GitHub

March 2023

Ibrahim Haddad, Ph.D.
Vice President, Strategic Programs (AI & Data)
The Linux Foundation

Foreword by Jeff McAffer
Senior Director of Product
GitHub

Recommended Practices for Hosting and Managing Open Source Projects on GitHub

GitHub is a platform that **allows developers to collaborate and share code,** providing a wide range of tools to support open source development and project management.

Documentation is an essential component of open source projects on GitHub that explains the project's purpose, code, usage, and contributions, instructions, and guidelines.

Manage user support through communication channels, including issue trackers, feedback platforms, and community forums.

Choose a type of license that supports the level of use, modification, and sharing required for your project, such as those approved by the Open Source Initiative.

Protect project code by implementing security features such as two-factor authentication, access control, code reviews, and scanning tools.

Two common licensing concepts are the Developer Certificate of Origin (DCO) and the Contributor License Agreement (CLA), which **outline the terms and rights of a contribution.**

English is widely spoken and understood worldwide, and as a result, it is **the best language to use when writing Github content** or communications.

The core open source principles of **peer review, releasing early and often, and continuous testing and integration** will help establish collaborative and transparent projects.

Providing accurate licensing information is crucial for open source projects hosted on GitHub.

Git, the version control of GitHub, allows developers to keep track of code changes over time.

DCO is a way for developers to **certify that their contributions to the project are their own** and that they have the necessary rights to submit the code.

CLA is a legal agreement between a developer and the project owner or maintainer that outlines the terms and conditions for contributions and ensures that the project has the necessary rights to use and distribute the code.

Contents

Foreword

While open source principles are relatively straightforward, "getting it right" can be challenging. Open source sits at the intersection of technology, community, business, and personal identity, so it stands to reason that some amount of tooling and rigor—best practices—will help smooth the path to success.

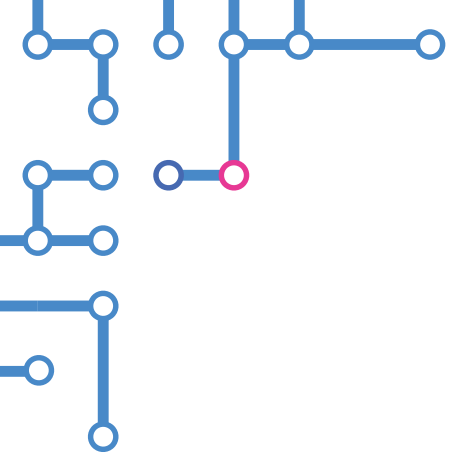
In my time as the Director of the Microsoft Open Source Programs Office (OSPO), the company evolved from having a few hundred developers and repos on GitHub to having tens of thousands of each. We moved from *ad hoc* operations to formal but still manual processes to near-full automation of our presence on GitHub and from project communities of dozens to thousands. We developed a ton of tools and best practices to smooth that path.

My move to work at GitHub nearly five years ago was meant to bring some of those tools and learnings to the GitHub product and, ultimately, you, the open source developers of the world. Today, GitHub has many facilities and possibilities, from discussions to actions / checks, packages, and releases, to security, that help you and your project. I can't even list them all.

Successful open source is as much about people and communities as it is about the code. With this article, Ibrahim applies his long history in open source to create a jumping-off point for you to make the most of your open source efforts on GitHub. Using the best practices outlined here will help you create a secure, robust, and vibrant community around your project. GitHub supplies much of the infrastructure, and you provide the innovation and community.

See you in a repo.

Jeff McAffer
GitHub

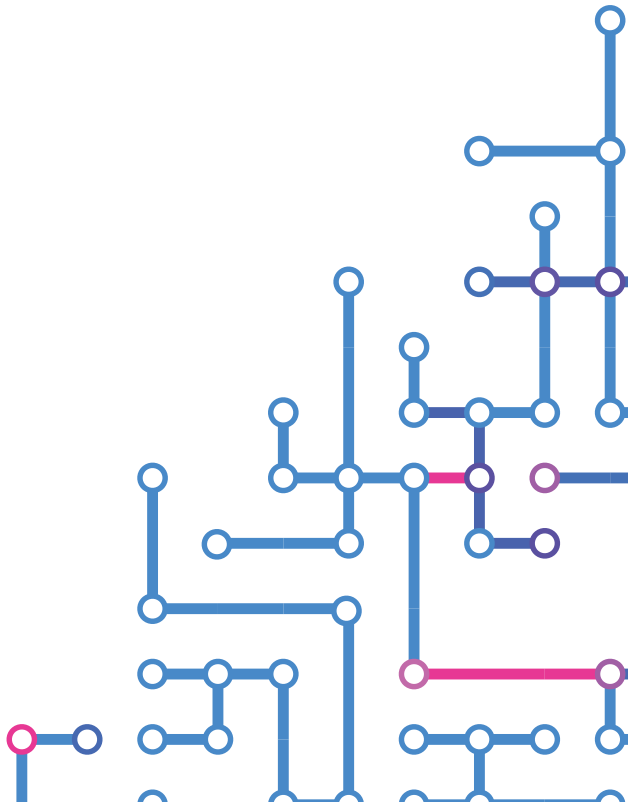


Abstract

Open source software (OSS) has transformed our world and become the backbone of our digital economy and the foundation of our digital world. Today, OSS powers the digital economy and enables scientific and technological breakthroughs that improve our lives. From the Internet and the mobile apps we use daily to the operating systems and programming languages we use to build the future, OSS has played a vital role. It is the lifeblood of the technology industry.

The use of GitHub for open source development has become increasingly popular in recent years, providing a platform for collaboration and sharing code.

Managing an open source project on GitHub can become less challenging with proper guidelines. This paper provides an overview of best practices for using GitHub for open source development. By following these practices, open source developers can improve the organization, understandability, and collaboration of their projects on GitHub, making it easier for other developers to adopt and contribute.



Introduction

The availability of OSS is changing how organizations develop and deliver products. A transparent development community and access to public source code enable organizations to think differently about procuring, implementing, testing, deploying, and maintaining software (FIGURE 1).

OSS has created an ecosystem with many benefits for all involved. Organizations across industries are building and growing their open source operations under an OSPO to help use and contribute to open source projects more efficiently and effectively and to benefit from its strategic impact (FIGURE 2).

OSS allows shared development and lowers research and development costs by enabling organizations to reap the benefit of billions of dollars of OSS, which they can harness to create better products and services. In addition, it helps to accelerate

FIGURE 1

OPEN SOURCE IS A TECHNOLOGY MARKET ACCELERANT

FIGURE 2

STRATEGIC IMPACT OF OSS

- Accelerate the development of open solutions
- Provide an implementation to an open standard

- Commoditize a market
- Reduce prices of nonstrategic software assets
- Share development costs

- Drive demand by building an ecosystem for products and services

- Partner with others
- Engage customers
- Strengthen relationships with common goals

product development and enables a faster time to market by aligning business needs with upstream open source projects. Organizations do not get involved in open source projects because it is fun; they do it because it is a part of their business or product strategy.

Where does most open source development happen? **GitHub**. GitHub is a crucial platform for open source development. It is a web-based hosting service that uses the Git version control system to allow developers to collaborate and share their code. It provides a centralized platform for open source developers to store, manage, and track changes to their codebase and collaborate on development efforts with other contributors.

One of the critical advantages of GitHub is that it allows developers to easily share their code with others, making it simple for others to contribute, review, and merge changes. It is a powerful tool for open source development, as it enables developers to work together on large-scale projects, regardless of their physical location. GitHub also provides a platform for community building and collaboration by allowing users to create open source projects, create and manage issues, and communicate with other developers via pull requests (PRs) and comments.

Additionally, GitHub provides a wide range of tools and features tailored to open source developers' needs. It includes features such as project management tools, code review functionality, packaging, release, deployment capabilities, and built-in integration with other developer tools such as continuous integration services, making it an all-in-one platform for open source development.

This paper will discuss recommended practices to help you improve your GitHub presence to attract more users and developers to your projects.

Documentation

Clear and detailed documentation is crucial for open source projects hosted on GitHub. It helps to ensure that the project is easy to understand and use for other developers, which can increase the number of contributors and users. Good documentation should include an overview of the project's purpose, usage instructions, and any dependencies or requirements. It should also include detailed explanations of the code, its workings, and any known issues or limitations. Having clear documentation can make it easier for other developers to understand the project and make contributions. It can also make it easier to maintain the project over time, as developers will better understand how the code works and can confidently make changes. Additionally, good documentation makes it easier for users to know how to use the software and troubleshoot any issues they may encounter.

An open source project can provide different types of documentation to help users and developers of its community. Historically, documentation has been an area that requires improvements. However, the situation is improving, with many projects having excellent documentation covering all areas of the projects.

In the following subsections, we highlight three core areas where documentation is essential.

1. Project
    - **b.** Mission
    - **c.** Governance
    - **d.** Community structure
    - **e.** Release cadence
    - **f.** Road map and priorities
    - **g.** Use cases
    - **h.** FAQs
2. Documentation targeted for users
    - **a.** User guide and tutorials
    - **b.** API guide
    - **c.** Architecture overview
    - **d.** Installation guide
    - **e.** Feature request and security vulnerability reporting process
    - **f.** Experience sharing section
3. Documentation targeted for developers
    - **a.** Detailed architecture and mapping to code sub-systems services when applicable
    - **b.** Development process
    - **c.** Getting involved
    - **d.** Guidelines for participation
    - **e.** Feature request process
    - **f.** Patch submission process
    - **g.** Signed-off-by process, when applicable
    - **h.** Developer guides and tutorials
    - **i.** API guide

Recommendations:

1. Use the REPOLINTER tool, created by the TODO Group[1], to identify common issues in GitHub repos.
2. Add a `README.md` file welcoming new community members to the project and explaining why the project is worthwhile and how to begin.

3. Add a CONTRIBUTING.md file explaining how to contribute to the project. The file explains the types of contributions required and how the process works.
4. Add a CODEOWNERS file to define individuals or teams responsible for code in a repository.
5. Add a CODE_OF_CONDUCT.md file that sets the ground rules for participants' behavior and helps facilitate a friendly, welcoming environment. The CODE _ OF _ CONDUCT.md file signals that this is a welcoming project to contribute to and defines standards for engaging with the project's community.
6. Add a SECURITY.md file that tells users how to report security issues in the project.
7. Provide documentation on the release methodology, cadence, criteria, etc.
8. Document your project governance, and make it available on the project's repo.

GitHub provides the ability, at the organization level, to automatically apply such community health files to any repo created within the org. To learn more, please visit GitHub Docs.

Support channels

Good user support is essential to maintaining an open source project on GitHub. Below, we provide recommended practices for open source projects to provide better support for their community:

1. Create a SUPPORT.md file that details how people can get help with your project(s).
2. Create detailed and well-organized documentation: This can include instructions on installing and using the project, troubleshooting tips, and examples of everyday use cases.
3. Use the issue tracker: This allows users to report bugs, request features, and ask for help. Responding promptly and providing precise and detailed responses can help to build trust and credibility with users.
4. Be open to feedback and suggestions: Encourage users to provide feedback, and actively seek feedback through surveys or interviews; this can help you understand what users want and how to improve your project.
5. Communicate actively and transparently: Using a blog, newsletter, or social media to share progress, road maps, and updates about your project can help users to know what's happening and how to plan for future changes.
6. Use GitHub Discussions to support and engage with your users and developers. Projects use GitHub Discussions. It can help build a community around the project and provide an easy way for users to get help and connect with other users.
7. Set up an easy-to-find and access help center: a centralized location where users can find answers to common questions, submit support tickets, and contact the core developers.
8. Archive inactive repos so that the users and developers know the repos are no longer receiving support.
9. Consider adding a FUNDING.md file to let users know how they can support your project.

By following these practices, open source projects hosted on GitHub can provide better support, resulting in a more successful and sustainable project.

Security

Implementing security measures for GitHub organizations is vital to protect the code and data of the organization's projects. Below, we provide a few recommendations focused on security measures for GitHub organizations:

1. Two-factor authentication (2FA): Enabling 2FA for all organization members helps add an extra security layer to the organization's accounts.
2. Access control: Use GitHub's built-in access control features, such as roles and teams, to limit who has access to the organization's repositories and what actions they can perform.
3. Use GitHub Branch Protection Rules and `CODEOWNERS` files to ensure that every change to your repos has been reviewed by the right people.
4. Secure code reviews: Implement a code review process that includes security checks, such as static code analysis, to identify and fix any potential vulnerabilities in the code.
5. Continuous integration & deployment (CI / CD): Use CI / CD tools like GitHub Actions or CircleCI to automate the building, testing, and deployment of code, allowing for faster identification and fixing of security issues.
6. Use GitHub Code Scanning to find vulnerabilities in your code.
7. Use dependency security scanning tools: Run tools like GitHub's Dependabot, Snyk, etc. to find vulnerabilities in packages, libraries, and other third-party code you depend on.
8. Employee security training: Enroll in security practices training provided by your organization.
9. Achieve and maintain the OpenSSF Best Practices Badge for the open source project. The OpenSSF Best Practices Badge recognizes and encourages open source projects that follow best security and vulnerability management practices. Projects that earn the badge will have demonstrated that they have a security policy, a vulnerability reporting process, and a method for handling reported vulnerabilities. By having the badge, projects indicate that they take security seriously and that users and contributors can trust that the project addresses vulnerabilities responsibly.
10. Identify who on the project will handle security issues (could be a team). Create a `SECURITY.md` file, and set up an email account to receive security vulnerability alerts.
11. Use GitHub Security Advisories to track, manage, and publish responses to vulnerabilities found in your project.
12. Use GPG to sign commits locally so you can mark them as verified on GitHub. Other people can be confident that the changes come from a trusted source.
13. The OpenSSF scorecard helps maintainers improve their security best practices, and open source consumers judge whether the software dependencies are safe. Scorecard assesses several heuristics associated with software security and assigns each check a score of 0 to 10. You can use these scores to understand specific areas to strengthen the security posture of the project.

By implementing these practices, organizations and individuals can help protect their code and data and keep their projects on GitHub safe and secure. Security is an ongoing process, and there is a need to continuously update and reinforce security measures.

Licensing

General licensing recommendations

The license of an open source project determines the rights to use, copy, modify, and distribute the code. The choice of license is essential in determining the project's openness. Open source projects are recommended only to use licenses approved by the Open Source Initiative and recognized as "free / libre" by the Free Software Foundation. Such licenses allow the software to be freely used, modified, and shared. To be approved by the Open Source Initiative, a license must go through its license review process to confirm that it satisfies its Open Source Definition (OSD). You may come across many other licenses that are incompatible with the OSD. Most of these licenses are "Source Available" licenses that commonly include restrictions or limitations on software use and distribution. These restrictions often render the licenses incompatible with the OSD.

Below is a list of recommended practices that open source projects can adopt to help provide accurate licensing information:

1. Include a `LICENSE.md` file in a conventional location (e.g., at the root of the repository): The file should specify the terms and conditions for using, distributing, and modifying the software.
2. Use OSI-approved open source licenses: These licenses have been widely reviewed and used, which makes them more accessible for developers to understand and comply with. In addition, GitHub has a convenient "choose a license" facility that makes it easy to use the most common OSI-approved licenses when creating your repos.
3. Include the license information in your `REAMDE.md` file: Include a note in the `REAMDE.md` file that specifies the project's license and provides a link to the `LICENSE.md` file.
4. Mark any third-party code: If the project contains third-party code, clearly mark it, and include information on the license of the third-party code.
5. Use a license badge in your `README.md` file: Users can quickly see what licenses your project uses by including a license badge in your `README.md` file.
6. Include a Software Package Data Exchange (SPDX) short-form identifier in a comment at the top headers of each source code file. SPDX is a standard format for communicating the components, licenses, and copyrights associated with software packages. Open source projects use SPDX to make the license and copyright information more accessible and machine-readable.
7. Ensure GitHub recognizes your license: Your chosen license should appear in the GitHub UI repo summary section. If it does not, it means GitHub could not find or understand your license. Unfortunately, that likely means others cannot as well.
8. Regularly review and update the licensing information to keep it current and accurate.
9. Provide clear guidelines for contributing to the project from a licensing perspective, especially if your project uses a DCO or a CLA (more on that in the next section).

By following these best practices, open source projects on GitHub can provide accurate licensing information and help developers comply with legal requirements, making it easier for developers to understand and use the software. These practices also help to avoid any legal or compliance issues.

DCOs and CLAs

DCO and CLA are two concepts used in the open source development on GitHub.

DCO stands for Developer Certificate of Origin, and it is a way for developers to certify that their contributions to a project are their original work or that they have the necessary rights to submit the code. It happens by adding a "Signed-off-by" line to the end of each commit message, which indicates that the developer agrees to the DCO and that they are the original author of the code or have the right to submit it. For instance, the Linux kernel development process requires all contributors to sign off their code, which indicates the contributor certifies the code as outlined in the DCO. The signature communicates that the contributor has created or received the contribution under an appropriate open source license that allows you to incorporate it into the project's code base under the license indicated in the file. The DCO establishes a chain of people responsible for the licensing and provenance of contributions to the project.

A Contributor License Agreement (CLA) is a legal agreement between a developer or their employer and the project owner or maintainer. A CLA outlines the terms and conditions for contributing code to the project, such as the rights and responsibilities of both parties. Open source projects use CLAs to protect themselves from legal issues that may arise from contributions made by third parties or when a company or organization is developing a project, and there is a solicitation of contributions from outside contributors.

Recommendations:

1. Include a copy of or reference to the DCO in your `CONTRIBUTING.md` file.
2. Set up a bot to enforce a "Signed-off-by:" tag in each commit. For example, you can install the GitHub DCO app to manage this aspect.
3. For projects hosted in the Linux Foundation or any umbrella foundations, use The Linux Foundation EasyCLA tool (https://lfcla.com/) to enforce signed CLAs before accepting contributions.
4. For projects hosted and managed by their founders, consider using CLA Assistant. The recommendation is that employees consult with their corporate counsel on the most appropriate ways to handle CLAs.

Language

At the Linux Foundation, we work with open source projects from around the globe. Some projects come to the Linux Foundation for hosting from countries where English is not the official language. Much of the project's documentation on GitHub is sometimes not provided in English. Our recommendation has always been to use the English language for any content published on GitHub targeted at a global audience. It is the most common language in software development and open source communities. When a project makes available documentation in English, it can be easily understood and utilized by a much larger audience.

Adopt core open source principles

Open source development is a collaborative approach to software development in which the source code is freely available for anyone to use, modify, and distribute. A defining characteristic of the development process is openness throughout the entire life cycle, from design to release. All aspects of the development process, including design, planning, implementation, testing, and release, are transparent and open to contributions from the wider community. This allows for a collaborative approach to software development, where individuals and organizations can share their expertise, knowledge, and resources to create high-quality software that meets the needs of a diverse set of users.

Core concepts of open source development that enable the collaborative aspect at scale include peer review, release early and release often, and continuous testing and integration.

- Peer review is having other developers review and critique the code, which helps improve the quality of the software.
- Release early and release often is a strategy for making software available to users as soon as possible, with frequent updates and improvements.
- Continuous testing and integration is constantly testing and integrating new code changes, which helps to identify and fix bugs early in the development process.

These concepts work together to create a collaborative, transparent, and efficient approach to software development. In the next subsections, we will explore how GitHub supports these core open source development concepts via various features that enable large-scale implementation of these concepts.

Peer review

The peer review practice reduces variations in style, prompts valuable conversations, and preserves the project's quality standards. GitHub manages it through the PR feature. A PR is a way for developers to submit changes they have made to a project's code for review by other collaborators. With the creation of a PR, other collaborators can review the code changes, leave comments, and approve or request changes before there is a merger of the code into the project's main branch. To ensure peer review, it is recommended that the owner of the GitHub org adjusts the settings and enables the branch protection rules (either for the org or specific report) to require a PR.

Release early and often

"Release early and often" is an OSS development philosophy that advocates for frequent and incremental releases of a project. GitHub supports this development approach in open source projects through branches and releases.

By using a development branch for ongoing work and a separate release branch for stable, production-ready code, developers work on new features and bug fixes in the development branch and periodically merge the changes into the release branch.

Another way is using feature branches, where developers create a new branch for each feature or bug fix they are working on. Once the work on a feature branch is complete, its merger into the development branch and eventually into the release branch can take place.

Additionally, GitHub has a feature called releases, which allows packaging, tagging, and the distribution of software versions. This feature can mark specific versions of the code as a release and provide a way for users to download and use the release version. Users can also view the release notes and see the changes made in each release.

Continuous testing and integration

Continuous testing and integration in GitHub are managed through a combination of tools, such as GitHub Actions and other CI / CD tools.

GitHub Actions is a feature that allows developers to automate their build and test workflow by creating custom actions triggered by specific events, such as a push to a branch or a PR. With GitHub Actions, developers can set up automated tests and integration steps that run whenever specific events occur. For example, with the creation of a PR, developers can set up GitHub Actions and Status Checks to run a suite of tests on the code changes to ensure that they work as expected before merging into the main branch.

Another way to manage continuous testing and integration is by using third-party CI / CD tools such as CircleCI, TravisCI, Jenkins, etc. These tools integrate with GitHub and allow developers to set up a pipeline of tests and integration steps that run automatically whenever there are code changes pushed to a repository. For example, when there are code changes pushed to the development branch, the CI / CD tool can automatically run tests, build and deploy the code to a staging environment, and then run additional tests to ensure that everything is working as expected before deploying to production.

Additionally, many third-party tools offer various testing options, such as unit testing, integration testing, and end-to-end testing, which allows for testing different aspects of the application.

The design of both GitHub Actions and third-party CI / CD tools makes the continuous testing and integration process as seamless and efficient as possible.

Conclusion

GitHub plays an essential role in open source development by providing a centralized platform for code collaboration, community building, and project management, making it easy for developers to collaborate on open source projects and bring them to fruition. It is a vital tool that allows developers to share, collaborate, and improve their contribution to the open source ecosystem.

As for getting started, there are several steps individuals can follow to bootstrap their open source development activities on GitHub:

1. Start by finding a project that interests you. Look for projects with a transparent development process and an OSI-approved license.
2. Read the project's documentation and guidelines. Make sure you understand the project's goals and its organization.
3. Create a project fork on GitHub to have a copy of the project to which you can work and submit changes.
4. Make small changes, and submit them as PRs to the upstream project. This process will allow the project maintainers to review your work and provide feedback.
5. Be open to feedback and willing to make changes. Open source development is collaborative, and good communication is critical.
6. Contribute to the project's community by answering questions, providing support, and helping new contributors.
7. Look for mentorship opportunities; if you find a maintainer whose work you admire, reach out to them, and ask if they would be willing to mentor you.
8. Keep your fork up to date with the main project by regularly syncing your fork with the original repository.

We hope you find this paper helpful in providing an overview of best practices for using GitHub for open source development. If you're looking to start working with open source development, head to GitHub, set up an account, and begin!

Endnotes

1 The TODO Group is an open community of practitioners hosted at the Linux Foundation whose goal is to create and share knowledge, collaborate on practices, tools, and other ways to run successful and effective open source initiatives and program offices.

Acknowledgments

The author would like to express his gratitude to Hilary Carter, Jason Perlow, Melissa Schmidt, and Barry Hall for their invaluable contributions to the development of this paper. Special thanks to Jeff McAffer for contributing the Foreword and for his insights and detailed feedback that has played a crucial role in improving this paper. Thank you all for your time and effort. The author also would like to acknowledge the contributions he received via LinkedIn from Sumanta Mukhopadhyay, Phil Coval, and Andreas Fehlner.

Linux Foundation resources

- E-book: A Road Map to Improve the Effectiveness and Impact of Enterprise Open Source Development
- E-book: A Deep Dive into Open Source Program Offices: Structure, Roles, Responsibilities, and Challenges
- E-book: A Guide to Enterprise Open Source
- E-book: Open Source Audits in Merger and Acquisition Transactions
- Linux Foundation Enterprise Guides
- Linux Foundation Open Source Compliance Program
- TODO Group
- The Software Package Data Exchange®
- Linux Foundation Training & Certification
- Linux Foundation Events

Feedback

The author apologizes in advance for any spelling errors or possible errors and is grateful to receive corrections and suggestions for improvements.

About the author

Dr. Ibrahim Haddad is the Executive Director of the Linux Foundation AI & Data and the PyTorch Foundation. His primary focus is facilitating a vendor-neutral environment for advancing the open source AI platform. Throughout his career, Haddad held technology and portfolio management roles at Ericsson Research, the Open Source Development Labs, Motorola, Palm, Hewlett-Packard, Samsung Research, and the Linux Foundation. Haddad graduated with honors from Concordia University (Montréal, Canada) with a Ph.D. in computer science.

LinkedIn: @ibrahimhaddad Twitter: @IbrahimAtLinux Website: IbrahimAtLinux.com

THE LINUX FOUNDATION | Research

Founded in 2021, Linux Foundation Research explores the growing scale of open source collaboration, providing insight into emerging technology trends, best practices, and the global impact of open source projects. Through leveraging project databases and networks, and a commitment to best practices in quantitative and qualitative methodologies, Linux Foundation Research is creating the go-to library for open source insights for the benefit of organizations the world over.

To reference the work, please cite as follows: Ibrahim Haddad, Ph.D., "Recommended Practices for Hosting and Managing Open Source Projects on GitHub," foreword by Jeff McAffer, The Linux Foundation, March 2023.