Best Practices- Linux Kernel

Apoorva Chauhan*
achauha5@ncsu.edu
The North Carolina State University
Raleigh, North Carolina , USA

Hiral Bhanushali
hbhanus@ncsu.edu
The North Carolina State University
Raleigh, USA

Angela Ho
aho@ncsu.edu
North Carolina State University
North Carolina , USA

Hrushabhsingh Chouhan
hychouha@ncsu.edu
The North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

The Linux kernel is a shining illustration of open source's power and the benefits of collaborative effort. The kernel has been around for three decades, due to its contributing developers from over large number of firms all over the world. The project's durability and success thus far have been unprecedented, especially when considering the development mayhem that occurs on a daily basis. The Linux kernel is undoubtedly the most dynamic kernel on the market, with stable versions issued every 3 months. Of doubt, such a massive undertaking must have had its share of setbacks and difficulties.

CCS CONCEPTS

• **Software Engineering** ; • **Best practices for Linux kernel** ; • **ReadMe**;

KEYWORDS

Linux Kernel Best Practices, Software Engineering

ACM Reference Format:
Apoorva Chauhan, Hiral Bhanushali, Angela Ho, and Hrushabhsingh Chouhan. 2018. Best Practices- Linux Kernel . In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The Linux kernel project lays emphasis on certain rules and practices that must be followed. The importance of which is specified later in this document. The 5 most important ad essential practices are as follows –

• Short Release Cycles • Zero Internal Boundaries • The No-Regressions Rule • Consensus Oriented Model • Distributed Development The Linux open source community follows these five guidelines, which have been critical to their success over the previous 30 years and are still used and followed.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

2 LINUX KERNEL BEST PRACTICE

The Linux kernel is a prime example of the power of open source and the merits of constructive collaboration. Initially developed in 1991 by Linus Torvalds, the kernel has been around for 27 years now, all thanks to its 14,000+ contributing developers from more than 13,000 companies all over the globe. The project's longevity and success has been unparalleled so far, especially when you imagine the development chaos that ensues almost on a daily basis.

2.1 Short Release Cycle

Major kernel releases used to happen every few years in the early days of kernel development. However, this resulted in a number of bugs and issues during the development process. In Long release cycles, developers were forced to include features in the next release even if they weren't totally stable. As a result, transition to shorter release cycles eliminated all of the problems associated with long release cycles.

The items in the rubric that must be taken care of while developing a Linux best practice projects are

•version control tools - We used the version control system/ tool to commit the changes. • The number of commits made by different people • The number of commits each developer made can be calculated as number of pull requests made to merge changes to main branch after each developer has worked

• Issues reports, Issues are being closed - Stable and robust code was created by continuous integration and collaboration

We have tried our best to follow the practices of Short Cycle Releases

2.2 Zero Internal Node

Developers typically work on certain sections of the kernel, but this does not preclude them from making changes to other parts of the kernel provided the changes are justified. This technique ensures that problems are resolved at the source rather than allowing various workarounds to emerge, which is always bad for kernel stability. Furthermore, it provides developers with a broader view of the kernel as a whole. Below are the items in the rubric that checks Zero Internal Boundaries.

- Evidence that the whole team is using the same tools (files in the repository , updated by lots of different people)
- Evidence that the whole team is using the same tools: everyone can get to all tools and files

- Evidence that the members of the team are working across multiple places in the code base
- Evidence that the whole team is using the same tools (e.g. Everyone ha the system/ software working on their machines)

All the team members have used Python 3, Flask and HTML/CSS to create the application. We documented all the commands to be executed to set-up the project. Whenever a developer wants to work on the projects. Whatever dependencies the developers encounter while coding the functionalities, they are included immediately in the requirements file, so that other developers will not face any issue. The work is divided in such a way that each member gets a chance to push code to different parts of the code base. Everyone has equally contributed and committed properly to show the contribution and hence establishing the zero internal boundaries criterion.

2.3 No Regressions Rule

The kernel developer community has constantly updated the kernel codebase over the decades, but not at the expense of quality. They consent that if a given kernel works in a given environment, all subsequent kernels will work in the same place. When the community discovers that a change caused a regression, they fix the problem as soon as possible. The no regression rule gives users confidence that updates will not affect their systems; Therefore, they are ready to follow the kernel in developing new capabilities.

Below are the items in the rubric to checks if the No Regression Rule is followed.

- **Test cases exist** - Test cases are written for all functions in the project.
- **Test cases are routinely executed** - Github action is used to test the project. All test cases will be executed every time a new code is pushed to the repository. The project is also tested before merging any pull request. The GitHub Repository will show the build status of the recent build.
- **The files CONTRIBUTING.md lists coding standards and lots of tips on how to extend the system without screwing things up** - Developers who contribute to the projects should follow the instructions provided in CONTRIBUTING.md file to avoid any harm to the project.

2.4 Consensus-Oriented Model

If a respected developer rejected the proposed change, the change will not be merged. This rule ensures that the kernel remains suitable for a wide variety of users and problems. No particular user community is allowed to make changes at the loss of other groups. As a result, we have a kernel that can be suitable for a wide variety of uses, regardless of the complexity.

Below are the items in the rubric to checks if the Consensus-Oriented Model is followed.

- **Is your source code stored in a repository under revision control?** - Yes, the entire code base is pushed to the GitHub repository, and all the members in the team are added as collaborators that allow them to contribute to the project.
- **Issues are discussed before they are closed** - All the members get a chance to open discussion with other members on the issues that are being closed. Whenever a member requests a pull request, the request is reviewed by at least one other member of the team. If necessary, the issue will be discussed on the discord or google meet.

2.5 Distributed Development Model

No single company must dominate kernel development. While any company can improve the kernel for its specific needs, no company can drive the kernel to hurt other companies or restrain what the kernel can do.

Below are the items in the rubric to checks if the Distributed Development Model is followed.

- **Chat channel: exists** - We have created a discord group chat for group members to discuss the distribution and progress of the project.
- **Issues reports**
- **Issues are discussed before they are closed** - Team members create issues in GitHub and the status of issues are tracked by other members. Every pull request is reviewed by at least one team member before merging.
- **Workload is distributed across the entire team** - We discuss the progress of projects through the discord group to keep track of the timeline. To ensure the tasks are well distributed, we used GitHub issues to keep track of each team member's workload.

3 CONCLUSION

These Linux Best Practices ensured that our software meets with all open source development standards. Every team members could work on all aspects of project as there exists no internal boundaries. Our program was constantly being changes a dn updated as per the requirement. Following a consensus-based approach guaranteed that everyone on the team had a say in every decision. W e discussed about all the changes that were made and everybody contributed to the code. Short Release Cycles enabled continual improvement and delivery of software throughout the software development life cycle.

4 REFERENCES

Tim Menzies. 2021. proj1rubric. Retrieved September 29, 2021 from https://github.com/txt/se21/blob/master/docs/proj1rubric.md