Edu-Setu : Linux Kernel Best Practices

Jayraj Mulani*
jmulani2@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Dhrumil Shah*
dshah6@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Yashasya Shah*
yshah3@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Anisha Chazhoor*
aschazho@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Harshil Shah*
hshah6@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

The Linux Kernel's success can be owed to seamless collaboration between multiple individuals. Inspired by the policies that helped achieve their success, we have tried to implement the proven practices in this project as much as possible.

ACM Reference Format:
Jayraj Mulani, Dhrumil Shah, Yashasya Shah, Anisha Chazhoor, and Harshil Shah. 2018. Edu-Setu : Linux Kernel Best Practices. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 SHORT RELEASE CYCLE

It is important for a product to have short release cycles. When kernel development was at a nascent stage, releases would be performed very infrequently. This would lead to a huge backlog in feature development and difficulty in debugging, causing inefficiency. Thus, kernel development has shifted to shorter release cycle model. Aping this in the Edu-Setu project at a smaller scale, the project has been continuously worked on by various contributors. As we can see in the Insights any sentence page, a total of 225 commits have been done in a span of 4 weeks (30 days). This is an average of 7.5 commits per day, which shows that features are being continuously added in the project on a regular basis. Issues are being raised frequently and are addressed in the next commit. Thus, the "release" cycles are short and updated.

2 DISTRIBUTED DEVELOPMENT MODEL

During the development of Linux Kernel, it was concluded that distributed system is the most feasible one to implement. Responsibility is distributed amongst different people and feature release moves faster. We have implemented the same in our project. The entire project has been divided into 3 main parts – backend, frontend and documentation. Each part has been further divided into 2 fragments, which form the branches of the repository. For each feature that is discussed, there is a corresponding branch. For example, studentTable addresses the student dashboard, profDb corresponds to professor table. Another branch dockerize handles the containerization of the entire project. After each feature is completed, pull requests are generated when the branch has to be merged with the main branch. The permissions for pull requests are not monopolized and contributors who have permission to access the project, can approve pull request. This ensures that the approval process is not centralized, improves the integration speed and code correctness

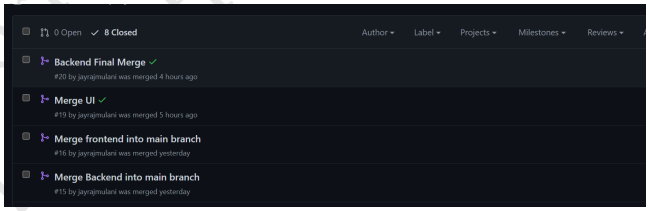

Figure 1: Pull requests for multiple branches ensure that the responsibility for integration is divided amongst team members

3 CONSENSUS-ORIENTED MODEL

For the model to be consensus oriented, it should adhere to the baselines provided by the senior developers. This is performed to ensure that the codebase remains true to the original purpose and none of the changes enacted can tamper with the existing features. This also takes authority from a single group of developers and distributes it amongst well known and respected developers. While there is no hierarchy in the group for this project, every feature is implemented in consensus with the developers responsible for the section. The initial planning phase of the project was conducted together as a team , where everyone pitched their ideas. Issues were created corresponding to every functionality and ideas were finalized only after thorough discussion and common consensus between team members.

4 THE NO-REGRESSIONS RULE

Regression testing refers to the testing of all main existing functionalities, before a new functionality is released. Linux kernel development follows a no-regression rule, as it assumes that each

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

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

functionality is running when a new feature is released in the market. There are automated tests in github workflow, which ensure that after each commit, the build is passing, test cases are running and automated documentation are getting generated. The main readme file also contains badges displaying the current status of build, the latest Zenodo DOI release ID and whether unit testing cases are passing or not.

5 ZERO INTERNAL BOUNDARIES

There are no limitations to the ideas which can incorporated in this project. Even if any of the developers have an suggestion in a portion of the project for which they are not responsible, they can do so. For example, frontend team has, on several occasions requested for special API endpoints for certain usecases. Issues corresponding to the requirement have been created and appropriately assigned. The backend team has took the requirements into consideration and developed accordingly. Vice verse is also true. Backend team has suggested new functionalities to be included on the front end of the page like withdrawal of application from a student's side. This shows that there are no boundaries in the team for idea discussions.

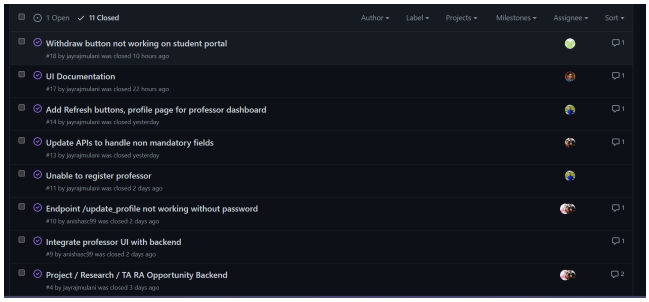

Figure 2: Any team members can suggest new ideas to be integrated in the main project, irrespective of their background and current work