Linux Best Practices

Sanjana Cheerla
Vincent Xin
Maya Patel
Sourabh Pardeshi
Saail Gurunath Ganesh

ABSTRACT

The purpose of this document is to link our team's practices towards Linux Kernel Practices, specifically

(1) Short Release Cycles
(2) Zero Internal Boundaries
(3) The No-Regressions Rule
(4) Consensus-Oriented Model
(5) Distributed Development Model

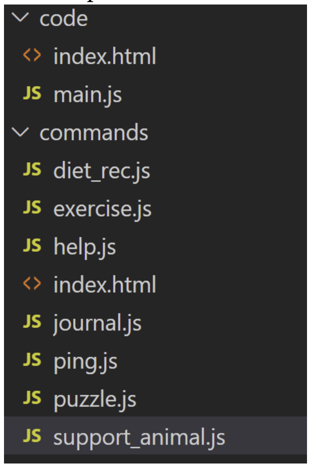
Through detailed explanations and also examples provided, we will connect some of the practices towards how our team has made progress for this project. For our main communication channel, we use discord, and our codebase and documentation is located on Github. Documentation and actions are also generated using Github actions. Issues are generated based on requirements that were discussed beforehand and we are actively communicating each other to talk about project progress. To track our contribution to the project, you can also look at our individual commit history, and also how many lines of code contributed in the insights section of our github page. Number of commits does not necessarily mean a member contributed less or more to the project.

1 SHORT RELEASE CYCLES

For our project, we implemented short release cycles to measure the progress of our project setup and feature development. After developing our ideas and requirements, we set out to create a release about every 2 days or whenever a large feature was implemented. Of course, stability is an issue when we are looking at short releases. We had to make sure that the main branch was stable through pull requests and code review. To look at all of our releases, you can see them at https://github.com/CSC510-G35-Fall2022/Mental_Health/releases. The first 2 releases were related to starting up the bot and getting it up and running for the first addition to the command list, the journal command. The next 2 releases were related to the implementation of new features, and were published after major progress was made on updating the bot.

2 ZERO INTERNAL BOUNDARIES

Our code base revolves around having files for every feature in the commands directory as well as a main execution file in the code directory. By having it in this style, it is easy for our team members to understand the structure of our code and know where to contribute. This requires all of us to understand how the system works in a more meaningful way, and allows us to develop code with the same capabilities. If there is an issue arising in any of the other files, any team member is able to pick up the slack and fix the issue or contribute new code to the existing structure. To give an example of what our code structure looks like here's a picture.

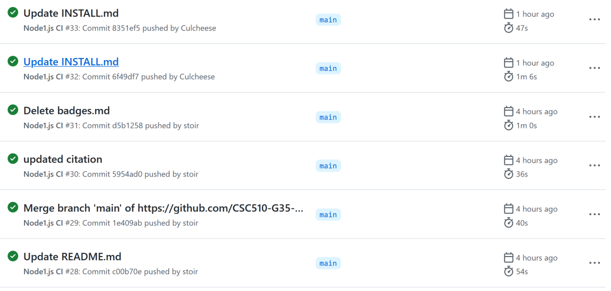


```
code
  index.html
  main.js
commands
  diet_rec.js
  exercise.js
  help.js
  index.html
  journal.js
  ping.js
  puzzle.js
  support_animal.js
```

It is also important to use the same tools so that everyone knows how to contribute to the code in a standardized way. This helps promote uniform understanding for setting up our bot and being ready to contribute new features.

3 THE NO-REGRESSIONS RULE

The No-Regression rules emphasizes that if a program worked given a certain configuration, that in any future updates to that program, the program will continue to work as intended. The way we have integrated this practice into our project is through the use of continuous integration. We are using Github actions to run our tests automatically. To show an example see the picture below.



```
✓ Update INSTALL.md                                          main    1 hour ago
  Node1.js CI #33: Commit 8351ef5 pushed by Culcheese                47s
✓ Update INSTALL.md                                          main    1 hour ago
  Node1.js CI #32: Commit 6f49df7 pushed by Culcheese                1m 6s
✓ Delete badges.md                                           main    4 hours ago
  Node1.js CI #31: Commit d5b1258 pushed by stoir                    1m 0s
✓ updated citation                                           main    4 hours ago
  Node1.js CI #30: Commit 5954ad0 pushed by stoir                    36s
✓ Merge branch 'main' of https://github.com/CSC510-G35-...   main    4 hours ago
  Node1.js CI #29: Commit 1e409ab pushed by stoir                    40s
✓ Update README.md                                           main    4 hours ago
  Node1.js CI #28: Commit c00b70e pushed by stoir                    54s
```

You can see that after updating certain files, the tests will rerun and display results on the github actions page. This helps ensure us that the program will continue to work, even as we add more and more new features to the project. Any future developers can also easily extend our testing to help make sure that old functionality doesn't completely break down and new features are able to be developed with ease.

4 CONSENSUS-ORIENTED MODEL

In a project, it is hard to keep track of who is exactly committing what. If we are not careful, big changes can be made to the project and it can lead to the deprecation of features that other members have worked on or break down the efficiency of the code. Mentioned in another section, we are also concerned with the stability of our project. To do so, we implement the Consensus-Oriented Model by having important additions to the code such as the addition of features use the standard github branching process. We would create branches for issues/features that were important coding markers for our releases. After implementation of the said feature, a member of the team would create a pull request. If another team member did not like the changes, they are free to make suggestions to change the code. Of course, this is not the only platform we are trying to reach consensus. We use Discord to constantly discuss the implementation of our project and argue any changes. To continue with a specific implementation of the project, a consensus would need to be met. Another avenue of communication is through in person meetings.

5 DISTRIBUTED DEVELOPMENT MODEL

In order to execute a distributed development model, we needed to split up parts of the project among different team members. This was done through Github issues assignment feature to assign tasks to certain team members. However, this was not the main way we discussed our split of the project. We used communication channels like Discord and group meetings to manage our work and talk about who is doing what part of the project. When an issue is fixed and needs code review, one member would submit a pull request, and whoever was available and knew the code well would review the code and merge the pull request. As mentioned before in the Zero Internal Boundaries section, the structure of our directories makes it easy to compare code among different command files, but it also allows us to separate features into different files that developers can work on separately. Multiple team members are able to work on different parts of the code at the same time, and progress is made in parallel rather than sequentially.