Visualizing Software Evolution in Linux: A Hierarchical Graph-Based Heat Map

Adrian Volpe
College of Science and Math
Belmont University
Nashville, USA
adrian.volpe@bruins.belmont.edu

Esteban Parra
College of Science and Math
Belmont University
Nashville, USA
esteban.parrarodriguez@belmont.edu

***Abstract*—Linux is a large open-source operating system. It's size makes it difficult for developers to fully grasp the system as a whole. Visualizations of the Linux kernel can provide developers with better program comprehension and understanding of evolution processes. This paper presents a visualization of Linux commit activity using a hierarchical, heat-map based graph. Using a large data set of commit data from Zenodo, we model the Linux structure as a directed hierarchy. Each node in the graph represents a subsystem, and edges show a parent-child relationship within the Linux architecture. The graph is rendered radially in Unity with a color gradient applied to nodes to indicate the volume of commits made to each subsystem. The result is an interactive, intuitive view of development hot-spots in the Linux kernel, aimed at supporting further software evolution analysis.**

***Index Terms*—commit, graph, heat-map**

I. Introduction

Linux is a family of open source operating systems that are based on the Linux kernel. The Linux kernel is a core component of a device that manages the hardware and resources. It's responsible for the CPU, memory and peripherals. Linux is versatile and is used everywhere from powering smartphones to operating smart TV's. Linux is fast, secure and highly scaleable, so it is important to learn how Linux changes over time [1].

Linux is an open source system, meaning that the source code is available, for free, to be modified and redistributed. However, the main Linux operating system (OS) is only modified by specific authors and each modification is checked thoroughly [2]. These changes to the Linux OS source code are called commits. Each commit contains certain data including: the author of the commit, the time the commit was made, every line changed, and the number of added and deleted lines. Commits are used to keep track of what code was changed, when it was changed and by who.

Analyzing commit history is useful for developers because it can show areas in the Linux kernel that have not been modified recently or that have been heavily modified. Areas that have recently been modified often are more likely to contain bugs or errors [3]. Thus, commit history visualization can strengthen our ability to find bugs in the Linux OS. We created a graph visualization of Linux subsystems using commit frequency as a heat metric. This visualization is specifically useful because it provides an interactive and clear way of seeing commit history. This approach has not been done before, in this way, and can provide a new way of seeing the relationship between the Linux OS structure and how commits are distributed across it [4].

As we will see in section IV this tool illuminates patterns in how the Linux kernel is being changed. Furthermore, in section V we will talk about issues we faced with scalability in our tool and ideas for fixing them.

The paper will as such: in section II we explain works that inspired this tool as well as similarities and differences in adjacent works. Section III presents the architecture and design of our tool. In section IV, we will expound on patterns shown through our tool, and why these are useful to developers. Finally, in section V we will conclude with the current limitations of this tool and ideas for future work.

II. Related Work

There have been many papers on Linux software visualization techniques [5], [6], [7]. However, most methods of visualization involve looking at the code rather than commits. An interesting software visualization method is the code city visualization. The method used in [7] turns software artifacts into buildings inside of a city. This allows users to see both the package structure of the system and the kinds of software artifacts inside of the system. This approach is different from our own in a few ways. First, it is a 3D visualization while ours is 2D [7]. Another visualization technique used is a Linux package dependency visualization tool shown in [8]. The tool shown in [8] is quite similar to our own in the fact that it is a 2D graph representation that shows hierarchical relationships. Though, their tool shows these relationships for packages in the Linux kernel while ours shows relationships between subsystems.

III. Methodology

This section outlines the design and implementation of our visualization tool, from data collection to graph construction and user interaction.

A. Data Collection

The dataset used was obtained from Zenodo, provided by Bermejo et al. [9]. It consists of Linux kernel commit and bug data. The data was given to us in the form of a JavaScript Object Notation (JSON) file. We developed a Java script to parse through the data and output an intermediate text file. The text file held information on the relationships between the subsystems, the number of commits on each subsystem, and a color corresponding with each subsystem determined by its number of commits. This text file is what we use to construct and render the graph.

B. Color Encoding

Initially, node colors were determined by a ratio between each subsystems number of commits to the maximum observed number of commits. However, due to a large range in commit data - from 0 to over 500,000 - most nodes appeared as indistinguishable shades of green. The average was somewhere around 1000 commits per subsystem. Except for a select few nodes, the graphs color was monochromatic. To address this, we implemented a logarithmic transformation to the ratio. This compressed the dynamic range of the nodes. Special cases had to be handled explicitly: $\log(0)$ is undefined and $\log(1)$ returned an unintended blue color - signifying 0 commits made on the maximally committed node. This logarithmic scaling fixed the issue with most nodes looking identical. Nodes now take on a range of colors from blue (fewer commits) to red (more commits). This heat-map approach is useful because it makes it immediately obvious to the user which nodes (or subsystems) have been modified the most. This can be seen in Figure 1.

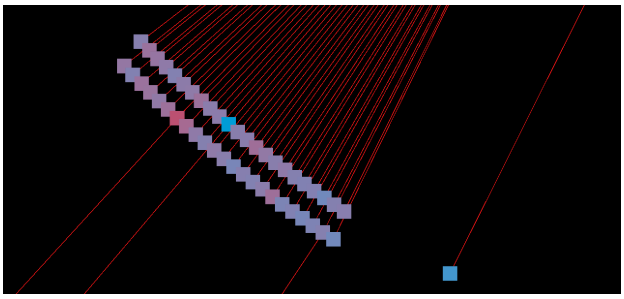

Fig. 1. This image shows a variety of colors attributed to nodes based on commit frequency of that subsystem.

C. Graph Construction

We utilized C# in Unity to visualize the graph. The text file contained the parent-child relationships between all of the subsystems, the exact number of commits made to each subsystem, and a color corresponding to each node. Our code created a set of nodes and a set of vertices based on this text file. Each subsystem is represented using a node and, edges are created to reflect parent-child relationships. This creates a hierarchical graph layout that not only shows us the relationships between the subsystems, but also the relative commit frequency between them as well. Nodes are positioned using a radial layout to help with spacing and user traversal.

D. UI/UX

Our visualization implementation is interactive. In particular, the user is able to zoom in/out and pan the camera. Clicking on a node reveals the subsystem's name and its commit count, providing detailed context on demand. An example of the node information panel is shown in Figure 2.

Fig. 2. View of the tool showing the info panel that appears on click.

IV. Results and Discussion

- talk about scalability issues, groups of high commit frequency, and think of other things to talk about as well

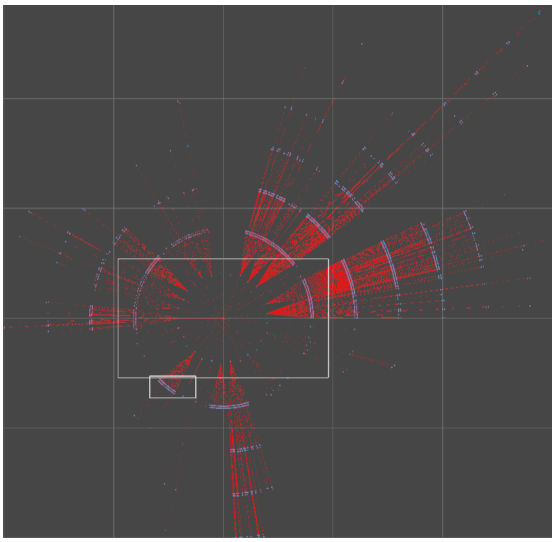

Fig. 3. Unity Scene view that shows how large the graph became with only a subsection of Linux commit data. The small rectangle is what is shown in Figure 1.

V. Future Directions and Limitations

The current state of the tool fulfills it's original purpose of visualizing Linux commit data. However, there are still many ways the tool could be improved to both help with the user experience and usefulness of the tool.

The tool does not keep track of some important data regarding the commits such as the time the commit was made or the author of the commit. These would a great first step in improving the tool. Furthermore, the tool is currently working on a dataset from 2023. A future improvement would be cloning the Linux GitHub repository for commit data rather than relying on the Zenodo dataset [9] we have been using.

Another implementation that would help developers using this tool would be filters to remove redundant or unnecessary information. The ability to sort by what dates the desired commits were made on, by author or by which subsystems the user is interested in would help this tool be more useful for developers. These filters would also help with the largest limitation of this tool, which is scalability. We used a small subsection of Linux commit data and yet the graph constructed is enormous. In Figure 3, it is clear to see that the graph is expansive and for a user to search that for a desired subsystem would be unrealistic.

Acknowledgments

Thank you to Belmont University SURFS program, to Dr. Esteban for his help with this project, and God, who without I could do nothing and to whom belongs all the glory.

References

[1] R. Love, *Linux kernel development*. Pearson Education, 2010.

[2] A. Israeli and D. G. Feitelson, "The linux kernel as a case study in software evolution," *Journal of Systems and Software*, vol. 83, no. 3, pp. 485–501, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121209002519

[3] M. Pradel and K. Sen, "Deep learning to find bugs," *TU Darmstadt, Department of Computer Science*, vol. 4, no. 1, 2017.

[4] K. Perumalla, A. Soni, R. Dey, and S. Rich, "Zeroin: Characterizing the data distributions of commits in software repositories," 2022. [Online]. Available: https://arxiv.org/abs/2204.07863

[5] P. Jayaraj, R. Gopalakrishnan, V. Jain, and P. I. Campus, "Effective performance analysis and visualization on embedded linux," *Software Engineering Applications*, pp. 27–29, 2006.

[6] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, "Cityvr: Gameful software visualization," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 633–637.

[7] H. Liu, Y. Jiang, and C. Xu, "Understanding the linux kernel, visually," in *Proceedings of the Twentieth European Conference on Computer Systems*, ser. EuroSys '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1044–1060. [Online]. Available: https://doi.org/10.1145/3689031.3696095

[8] X. L. E. Mithun and H. van de Wetering, "Linux package dependency visualization," *Master's Thesis at Department of Mathematics and Computer Science, Aug*, pp. 1–64, 2009.

[9] M. Maes Bermejo, J. M. Gonzalez-Barahona, M. Gallego, and G. Robles, "A dataset of linux kernel commits," 2024, doi: 10.5281/zenodo.10654193.