Git Visualizations

Yash Kurkure

1 Introduction

Navigating large code bases is an essential skill for software developers to have. There are many tools that help developers navigate code bases such as code editors like CLion and PyCharm [13], and Google's Code Search [12]. There are different aspects of code bases that a developer may be interested in and there are various tools available for each of them. This project proposes a visualization tool to navigate code bases using a tree visualization of the directory structure. The interactive tree visualization can be further used by the programmer to query information on the files and directories. This project focuses on the version control aspect of code bases.

2 Related work

Here is short list of tools that fit this use case:

- Repo-Visualizer [19] is a project that comes closest to this idea. It is an open-source tool that offers an interactive interface for visualizing and understanding repositories' structure, history, and collaboration dynamics. Key features include file tree visualization, commit history graphs, branch representation, and contributor analysis.

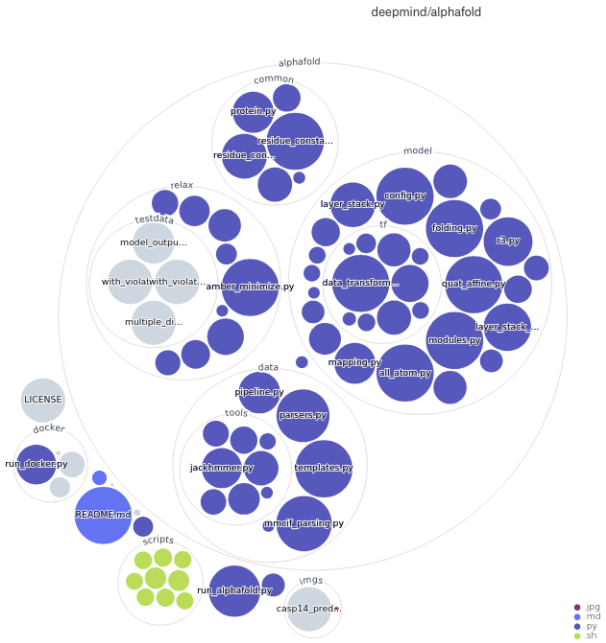

Fig. 1. Visualization produced by Repo-Visualizer

Repo Visualizer uses a tree like structure where files and directories are represented as circles. The circles of the members of a certain directory will lie within the circle of that directory as shown in the figure.

- Gource [8] is a visualization tool for source control repositories. The repository is displayed as a tree where the root of the repository is the centre, directories are branches and files are leaves. Contributors to the source code appear and disappear as they contribute to specific files and directories. Figure 2 shows a visualization produced by gource of the Yum project [17].

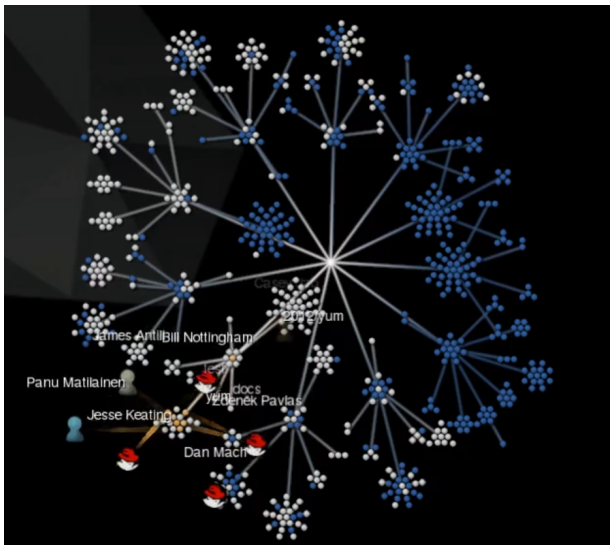

Fig. 2. Gource Visualization

The image showcases Gource's unique visualization of the Yum project's [17] commit history. The visualization appears as a tree-like structure with branches representing directories and nodes representing files. Commits are displayed as animated actions performed by avatars representing the contributors.

- GitKraken [2] is a cross-platform Git client with a visual interface for managing repositories, including commit history graphs, branch visualization, and more. It also supports GitFlow, a branching model that streamlines collaboration and release management.
- Sourcegraph [4] is a web-based tool that indexes code from various version control systems, such as Git, and provides a search engine that supports advanced search queries, including regular expressions, filters, and search scopes. While Sourcegraph's primary focus is on code search and navigation, it does provide some visualization features such as dependency graphs but is not the main focus.
- Additional work includes: Sourcetree [5], Github Desktop [1], GitLens [3], etc.

3 Data description

The data for this project lies within a cloned git repository. Git stores data as a series of snapshots instead of storing differences between file versions like many other version control systems. Each commit in Git represents a snapshot of the entire working directory at a given point in time. The data is stored as objects, which are identified by a unique SHA-1 hash. There are four types of objects:

- Blob: Represents a file and its contents.
- Tree: Represents a directory and stores references to blob objects (files) and other tree objects (subdirectories).

• *Yash Kurkure, E-mail: yash.kurkure1903@gmail.com*

- Commit: Represents a snapshot of the repository, including metadata such as author, committer, date, and a message. A commit object also references the tree object representing the root directory of the snapshot.
- Tag: A lightweight object that references a commit object to create a more human-readable name for a particular commit, typically used for marking releases.

Git objects are stored in the ".git/objects" directory as individual files. However, to save space and improve performance, Git periodically repacks objects into packfiles. A packfile contains multiple objects and their delta-compressed versions, which helps reduce storage space. Although we could write our own scripts to go through all this data stored in the .git folder, it is not necessary to do so. Git already provides a efficient command line interface to query this database. Additional details about the inner workings of git can be found at [14] and the git docs [9].

Some data does not reside locally in the cloned repository and can be accessed using Github's REST API [10].

A general list of attributes that one can query using git/Github's REST API:

- branches
- commits
- contributors
- pull requests
- forks
- files
- directories
- code and line count - additions/deletions

This list is not perfect as each attribute on the list can have further details such as each commit has a date associated with it and each pull request only affects the changed files and directories. The goal of this project is to use a combination of these attributes and make it easy for a developer to gain insights about the code base. For example, a developer might be interested in the pull requests that have modified a specific file.

4 Research

I would like to divide the work done in this project according to the front end and back end parts of the web application.

4.1 Front end

For the front end of the application I explored various software to display the file structure of the repository as a graph and ways top make the graph interactive. Two main libraries I experimented with were three-force-graph [7] and 3d-force-directed-graph [6].

Both libraries use a similar format for the data of the graph, which is a simple JSON file with the list of nodes and the links between the nodes.

```
{
    "nodes": [
        {
          "id": "1",
          "name": "A",
          "val": 1
        },
        {
          "id": "2",
          "name": "B",
          "val": 10
        },
        ...
    ],
    "links": [
        {
            "source": "1",
            "target": "2"
        },
        ...
    ]
}
```

My initial attempt was using the library three-force-graph [7] , which arranged the directory tree on a plane.

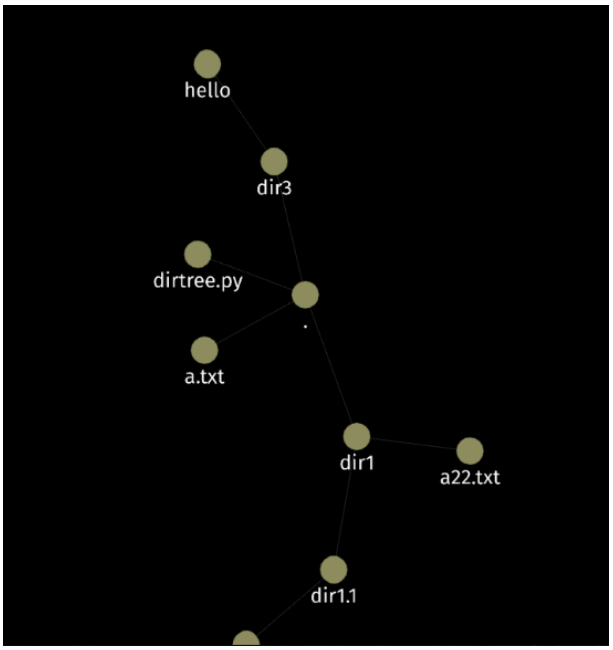

Fig. 3. Directory Tree using Three-Force-Graph

Figure 3 shows an example of a small directory structure being displayed as a graph. While this works perfectly for small directorial structures, this is not the case for large directories as there would be an overcrowding of nodes. The 2D plane also fails to capture the idea of "levels" in a directory tree. Compared to this, the library [6] created a 3D structure and also allowed the graph to be displayed as a DAG. This allowed the visualization of the levels in the directory tree as shown in figure 4

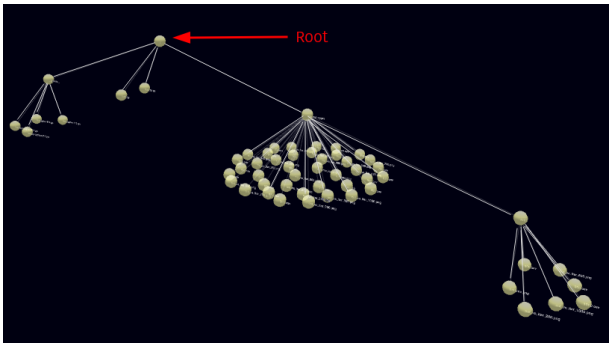

Fig. 4. Directory Tree using 3d-force-directed-graph

The next task was to try and load the graph for a large repository such as the Linux kernel [18]. As expected, this failed because of the size of the repository and rendering could not keep up with the amount of nodes and links. To overcome this we must dynamically load the data

for parts of the tree that should be visible to the user. I experimented with two ways to provide the user a way to select parts of the tree they care about.

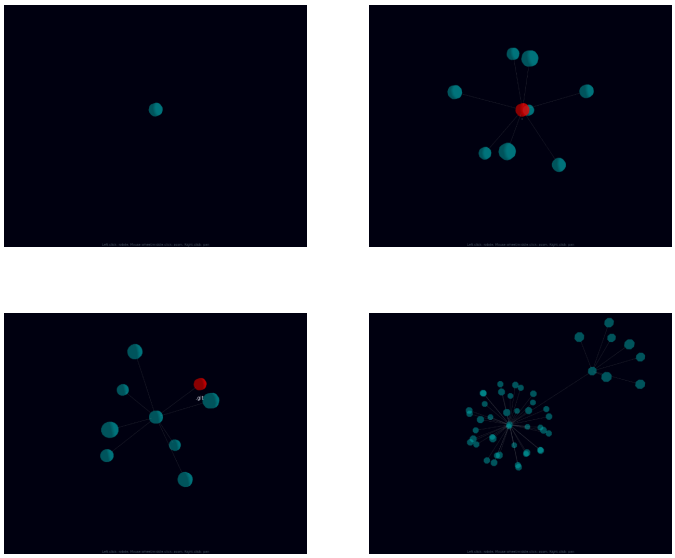

Fig. 5. Node click and expansion mechanism

Figure 5 shows a mechanism that allows the user to click on the nodes to expand them and view the paths to other files and directories. By doing this the application could avoid loading the entire graph and rendering slow downs. Although this method seems intuitive, it is time consuming as the user will have to manually expand all the nodes of the directories that they want to run a query for.

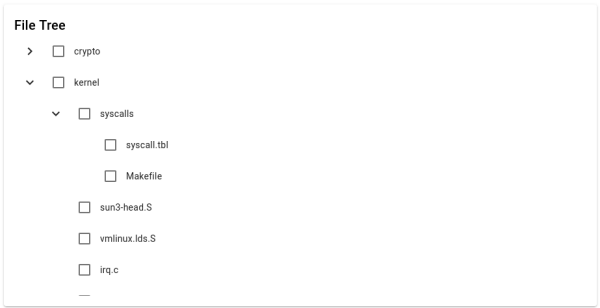

Fig. 6. File Tree

The second way to allow exploration of the graph was to use a file tree. Figure 6 shows an example of the Linux kernel file tree loaded. The file tree contains a check box next to each entry, which the user can check if the path to that directory or file should be visible in the graph.

Figure 7 shows an example where the ”run” directory is selected in the file tree on the left. On the right in the figure, the user can see the path from the root to the ”run” directory and also see the children of the selected directory. Using the file tree the user can select multiple paths they want to visualize and run queries on. For the UI such as the file tree, the application uses Angular Material [11].

4.2 Back end

The back end uses a python Flask server that is also responsible for cloning the requested repository and generating the data required by the front end to visualize it. The server clones the file structure and runs a simple algorithm on the repository to generate the graph along the selected paths in the front end. The algorithm uses Depth First Search on the directory structure to traverse all the selected paths. It also works with wild card (*) entries in the path.

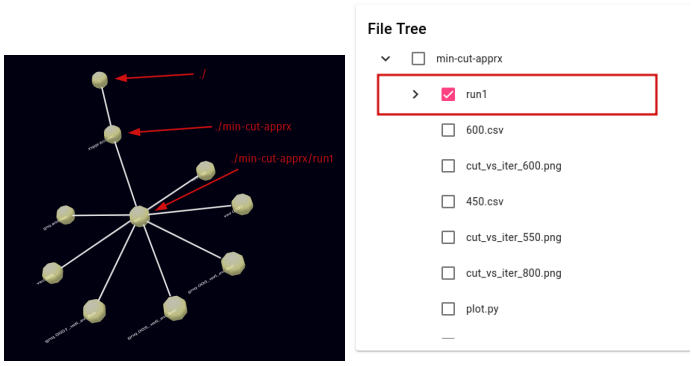

Fig. 7. File Tree with Graph

The server is also responsible for sending the complete file tree to the front end. The file tree is sent in the form of a JSON object after crawling the clones repository. Below is an example of the format of the data required to load the file tree into the front end:

```
{
    name: "dir1",
    isFile: false,
    path: "/dir1",
    children: [
                {
                    name: "file1",
                    isFile: true,
                    path: "/dir1/file1",
                    children: []
                },
                {
                    name: "dir2",
                    isFile: false,
                    path: "/dir1/dir2",
                    children: [
                                {
                                    name: "file3",
                                    isFile: true,
                                    path: "/dir1/dir2/file3",
                                    children: []
                                }
                    ]
                }
            ]
}
```

Other work required by the back end which remains incomplete at the time of this report includes running the git queries. As mentioned previously, this can be achived by running the command line program git or using Github’s REST API [10]. The idea is to allow the user to run queries from the front end and use colors to highlight the nodes affected by the results of that query. Here is a list of queries and corresponding commands for git that can be run by the user:

- Query: List all branches in the repository, both local and remote. Command: `git branch --all`
- Query: Find the most recent 10 commits that affected a specific file. Command: `git log -n 10 -- <file_path>`
- Query: Show the commit history of a specific author. Command: `git log --author="<author_name>"`
- Query: Display the commit history between two dates. Command: `git log --since="<start_date>" --until="<end_date>"`
- Query: Show the changes introduced by a specific commit, including file modifications, additions, and deletions. Command: `git show <commit_hash>`

- Query: Find commits containing a specific search term in the commit message. Command: `git log --grep="<search_term>"`
- Query: Show the files that have changed between two branches. Command: `git diff --name-status <branch1>..<branch2>`
- Query: Find the most changed files in the repository (by the number of lines changed). Command: `git log --pretty=format: --name-only | sort | uniq -c | sort -rg`
- Query: Show the history of changes to a specific file, including the changes made in each commit, the author, and the commit message. Command: `git log -p --follow -- <file_path>`

5 Results

Over the course of this semester I was able to implement the font and back end for rendering the graph and loading different git repositories. This includes the methods discussed in the research section 4. The project overall gave me an idea on how to handle large graph data and experiment with different UI setups to make it easy for the client to explore the graph.

The next step would be implement the front end for running the queries.

6 Conclusion

The goal of this project is to allow the developer to explore a code base and the git version control system associated with it. With the current implementation the developer will need to load a separate window with a web application, which can be inconvenient as the application is not part of the general workflow. A better way to implement this tool would be as a vs code extension. This is possible as vs code extensions are able to run web applications inside them with the help of a webview [15]. For future work, I would like to begin using Microsoft's webview tool kit [16] to convert this into an extension. The changes would include getting rid of the material file tree and using vs code editor's file tree instead. This would make it easy for developers to run this tool in a familiar environment. The key takeaways from this project were the libraries and softwares used to bring it together.

Code for this project can be found at: `https://github.com/yashkurkure/gitvisualizations`

References

[1] Github desktop. `https://desktop.github.com`. Accessed: 2023-05-01.

[2] Gitkraken. `https://www.gitkraken.com`. Accessed: 2023-05-01.

[3] Gitlens. `https://gitlens.amod.io`. Accessed: 2023-05-01.

[4] Source graph. `https://sourcegraph.com`. Accessed: 2023-05-01.

[5] Sourcetree. `https://www.sourcetreeapp.com`. Accessed: 2023-05-01.

[6] V. Asturiano. 3d force-directed graph. `https://github.com/vasturiano/3d-force-graph`. Accessed: 2023-05-01.

[7] V. Asturiano. Threejs force directed graph. `https://github.com/vasturiano/three-forcegraph`. Accessed: 2023-05-01.

[8] A. Caudwell. Gource. `https://gource.io/`. Accessed: 2023-05-01.

[9] S. F. Conservancy. Git docs. `https://git-scm.com/docs/`. Accessed: 2023-05-01.

[10] Github. Git rest api. `https://docs.github.com/en/rest?apiVersion=2022-11-28`. Accessed: 2023-05-01.

[11] Google. Angular material ui components. `https://material.angular.io/`. Accessed: 2023-05-01.

[12] Google. Code search. `https://cs.android.com/`. Accessed: 2023-05-01.

[13] JetBrains. Tools for software developers. `https://www.jetbrains.com/`. Accessed: 2023-05-01.

[14] H. Kapadia. Git internals. `https://git.harshkapadia.me/`. Accessed: 2023-05-01.

[15] Microsoft. Vscode webview api. `https://code.visualstudio.com/api/extension-guides/webview`. Accessed: 2023-05-01.

[16] Microsoft. Vscode webview ui tookit samples. `https://github.com/microsoft/vscode-webview-ui-toolkit-samples`. Accessed: 2023-05-01.

[17] Redhat. Yum. `https://fedoraproject.org/wiki/Yum`. Accessed: 2023-05-01.

[18] L. Torvalds. Linux kernel github. `https://github.com/torvalds/linux`. Accessed: 2023-05-01.

[19] A. Wattenberger. Repo visualizer. `https://githubnext.com/projects/repo-visualization`. Accessed: 2023-05-01.