Gizual

Repository Visualization for Git

Stefan Schintler, Andreas Steinkellner and Yaryna Korduba

706.057 Information Visualisation SS 2022
Graz University of Technology

07 Jul 2022

Abstract

Git is a tool widely used by software engineering teams to keep track of project development. Its native command, Git blame, provides additional metadata about the files at a particular point in time (commit). The information includes the author of each line of code, the time of its last update, and the commit reference. Unfortunately, at times when the analysis of the whole file is needed, git blame textual information might be hard to manually analyse. In this report we describe the development of our visualization tool for git blame data. The goal of this tool is to convert the data to a form which is easy to understand and analyse.

Contents

List of Figures

Chapter 1

Introduction

Git is powerful and popular software designed to help coordinate work in personal and team projects [Git 2022]. The core functionality of Git is version control. By the statistics from Open Hub, a platform with an objective to index the open-source software development community, 73% of Open Hub registered projects use a Git versioning system [Synopsys 2022].

Git blame is a native Git command-line utility, which allows examining specific points in file history [Atlassian 2022]. Git blame provides the context about each line inside a specific file. The information includes the author, the commit hash, and the last line update date. One can receive the information by running the command `git blame <name of file>` in the terminal.

Unfortunately, git blame operates only on individual files. Therefore, it becomes quite burdensome to compare the information from a few files simultaneously. Moreover, each line of text in a file is annotated by its own metadata. Obtaining detailed information about a line of interest is easy. However, using git blame makes it hard to answer questions about the whole file in general. For example, which author lately contributed the most to the file, or how significant was the latest change.

That is why we decided to investigate the opportunities for the visual representation of git blame results and make these results convenient to analyze. The project received the name “Gizual”, which is a mixture of “Git” and “visual”. The main objectives of the project were to:

- allow users to generalize git blame results about each file.
- allow users to compare git blame information for multiple files conveniently.
- create a solution for quick lookup of git blame metadata for different commits.

Chapter 2

Related Work

The initial idea for the Gizual project was largely inspired by a tool called Seesoft, developed by Eick et al. [1992]. While this current work on Gizual tries to incorporate state-of-the-art technology, the underlying core concepts of the visualizations are very similar to Seesoft, with the goal of creating a visual representation of software repositories.

The field of software visualisation has been growing ever since the work on Seesoft. The RepoVis project [Feiner and Andrews 2018; Hamidovic et al. 2018] collected commit data from a Git software repository on a dedicated server and provided a visualisation frontend in the spirit of Seesoft.

Similar work, but on a larger scale, was also done recently by Youngtaek et al. [2021], where researchers created an entire tool for metadata analysis, which features some statistics similar to our tool, but their work appears to be much more feature complete in comparison to our proof of concept implementation. However, their demonstration page only provides three example repositories, instead of letting users pick their own repository.

Instead of relying on a server to parse the Git repository structure, we wanted to achieve maximum performance by using the latest client-side technologies to handle as much of the work as possible inside the web browser. Our project is an attempt to create a tool similar to Seesoft using state-of-the-art technology and best practices.

Chapter 3

Data Acquisition

The first step of any visualization is to handle data acquisition. For our work, we needed to obtain metadata for each file in a Git repository at a particular point in time. This metadata would then be used to visualize various different statistics of a file. Since the goal of the project was to provide the visualization and the control panel in the web browser, specialised tools were needed to enable us to work with git repositories from within a web browser. This chapter gives an overview of the individual building blocks which enable us to move data from a low-level Git library into the web browser.

3.1 Git Command Line Tool

The initial idea to visualize git data within the browser was to just use a Git command line tool to generate some form of output we could later use for our browser visualization. While this approach would have worked well in theory, it was not explored further due to usability concerns. Users would have to use their local Git tools and work on the command line in order to use the tool, which would make it clunky and largely inaccessible. Thus, this idea was abandoned quite early on.

3.2 Libgit2

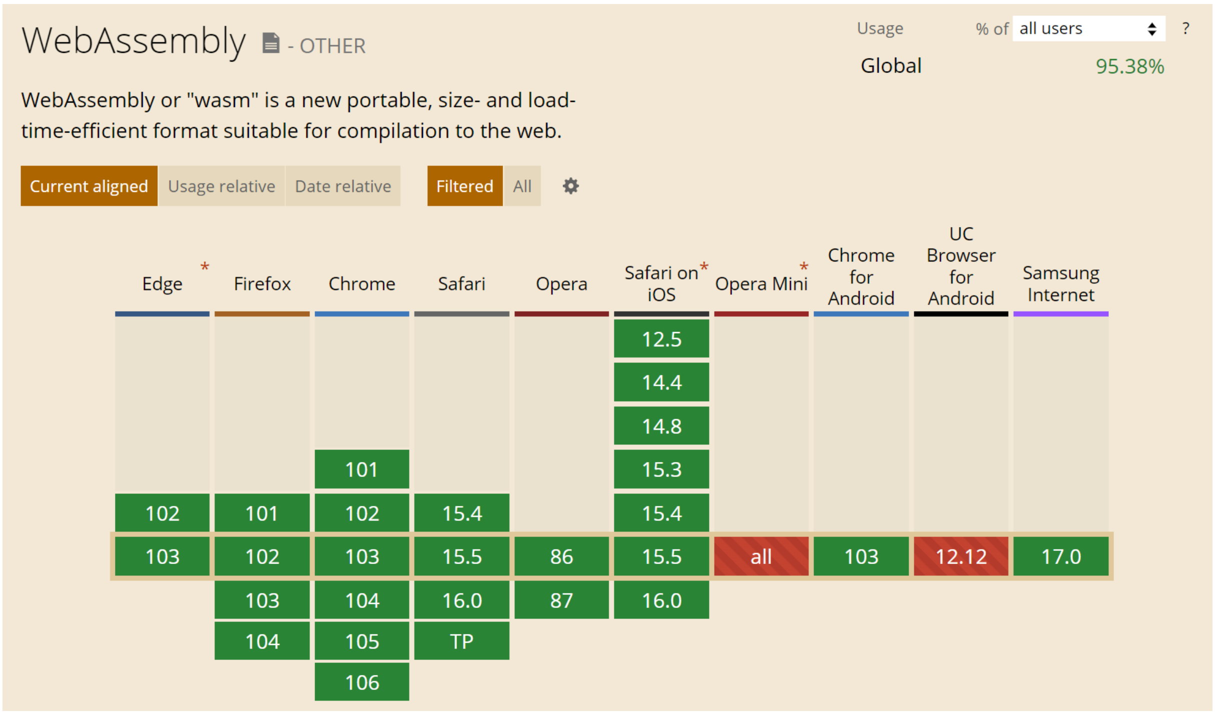
Since relying on users to provide their own git blame information to the browser visualization was not acceptable in terms of usability, we had to find a way to create git blame information automatically. Libgit2 [libgit2 2022] is a open-source implementation of the Git core, used by many large credible organizations. Its lightweight C implementation would allow us to create our own methods to access the Git core data we need for our visualization.

3.3 Gizual-cli

After settling on libgit2 as a core library for our project, the first minimum viable product was built on top of it, using C++ to create a command-line interface to generate git blame data for any desired repository. The initial design created output in JSON format, which we could then parse client-side in the browser to create a visualization. While this workflow worked well enough in theory, it was not acceptable to us in terms of overall usability, so we had to engineer a new solution of providing access to libgit2 from inside the browser.

3.4 WebAssembly

When thinking about getting any data from a C/C++ library into the browser, the easiest approach is usually to execute the library on the server side while providing a REST API for the browser to


WebAssembly - OTHER

WebAssembly or "wasm" is a new portable, size- and load-time-efficient format suitable for compilation to the web.

Usage % of all users

Global 95.38%

Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
					12.5				
					14.4				
					14.8				
		101			15.3				
102	101	102	15.4		15.4				
103	102	103	15.5	86	15.5	🟥 all	103	🟥 12.12	17.0
	103	104	16.0	87	16.0				
	104	105	TP						
		106							

Figure 3.1: Most modern browsers (about 95%) support executing binary code in WebAssembly format within their sandboxes. [Screenshot taken by the authors of this report.]

load its data. For our use case, this was however not sufficient, since it would entail uploading any repository to be explored onto the server. Instead, we decided to experiment with WebAssembly [W3C 2022]. WebAssembly is a modern binary instruction format and offers a load-time-efficient virtual stack machine within the context of the browser. Since WebAssembly is executed as part of the sandbox in a browser tab, security and memory safety is also taken care of. This allows us to compile C/C++ code (as well as other compilable languages) into the Wasm format which we can then load and execute within the browser. It is well-supported by most modern browsers used by 95% of the global internet users, as can be seen in Figure 3.1.

3.5 Emscripten

While WebAssembly offers a lot of abilities, it isnot nearly as fully featured as modern desktop environments. There is, for example, no standard library present within the browser, which means the entire logic necessary to execute a program needs to be statically compiled into the final binary executable. One approach to simplifying the compilation step to WebAssembly is offered by Emscripten [Zakai 2022]. It is essentially a compiler toolchain based on LLVM [LLVM 2022b] and Clang [LLVM 2022a] and offers a pre-compiled standard library which is optimized for the browser context. It also polyfills some missing features, like for example the lack of a file system by simulating a basic filesystem in the browser within memory.

3.6 Emscripten + libgit2

While the idea of compiling a C library for use in the browser using WebAssembly sounds straightforward, the actual implementation is not easy. We have come across multiple issues which had to be resolved in order to get the app working properly.

3.6.1 HTTP Requests

libgit2 uses HTTP requests to clone repositories. It natively ships with implementations for these on Windows and Unix systems. However, both are incompatible with the WebAssembly environment. We therefore had to fork libgit2 and apply some patches from Salomonsen [2022], in order to allow libgit2 to execute proper HTTP requests from within the confines of the browser.

3.6.2 CORS

Since we are running libgit2 within the sandbox of the browser, we also need to adhere to security rules enforced by the browser. Cloning from a server like GitHub from directly within the browser will result in a CORS error. CORS is short for Cross-Origin Resource Sharing and prohibits resources from origin A to be requested by origin B unless origin A has given explicit permission to the domain of origin B. To overcome this limitation, we set up a HTTP proxy on the server side, which is able to proxy Git requests via HTTP from the browser to the desired Git server. To ensure no malicious third party can abuse the proxy, we ensured to only allow proxying to the three major Git providers GitHub, GitLab, and BitBucket.

3.6.3 File System

A major limitation of WebAssembly and its execution within the browser sandbox is that there is currently no proper file system in place. However, libgit2 requires a working file system to clone and explore a repository. Luckily, Emscripten offers what is called MEMFS [Emscripten 2022], which essentially is a file system purely stored within memory. Using MEMFS, libgit2 has access to a working file system and can therefore be used.

3.6.4 Blocking the Main Thread

Code execution within the browser is usually inherently single threaded. Both the UI animations and any JavaScript code are executed on the same thread. Usually, this is not a problem since most browser APIs and libraries are designed to be executed asynchronously using Callbacks or Promises. libgit2's API, however, is executed synchronously, which means if we would call it to clone a repository, the whole UI would be frozen until the cloning process is finished. To overcome this limitation, we decided to move any interaction with the libgit2 library into a Web Worker, which enables its code to be run asynchronously alongside the main thread.

3.6.5 64-Bit Support

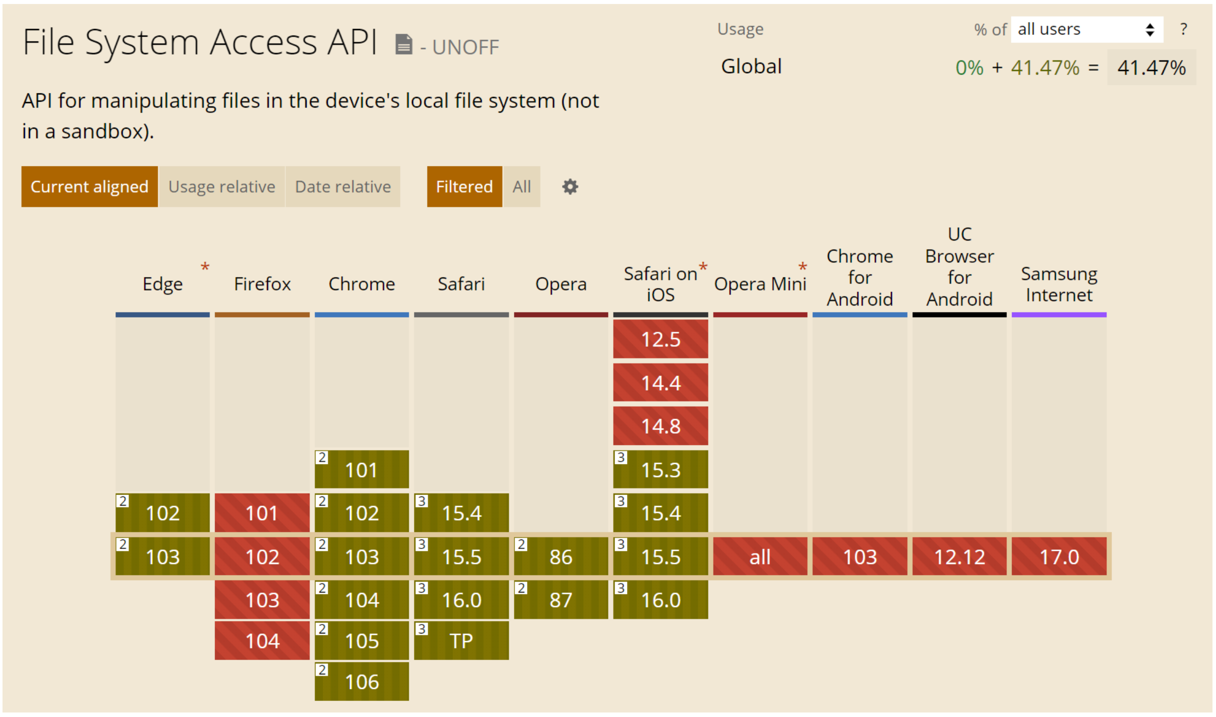
WebAssembly is inherently built as a 32-bit system, but libgit2 offers timestamps of commits as int64 values. To allow integers with 64 bits to be passed from libgit2 to the JavaScript context of a web app, we used Emscripten's support of BigInt in order to pass large numbers.

3.7 File System Access API

The File System Access API offers the ability to interact with a user's local files and folders from within the context of the browser sandbox. It offers a great alternative for Gizual to access repository data without needing to always re-clone all data from third-party servers. It is, however, currently only supported by Chromium-based and WebKit-based browsers, as can be seen in Figure 3.2.

3.8 Data Acquisition Process

When putting all the different components together, the process of acquiring all the necessary data for the visualization becomes quite complex, as can be seen in Figure 3.3. First the user is offered a directory picker powered by the File System Access API, which they can use to select a local folder containing a


File System Access API - UNOFF

Usage: Global — % of all users — 0% + 41.47% = 41.47%

API for manipulating files in the device's local file system (not in a sandbox).

Edge*	Firefox	Chrome	Safari	Opera	Safari on iOS*	Opera Mini*	Chrome for Android	UC Browser for Android	Samsung Internet
					12.5				
					14.4				
					14.8				
		[2] 101			[3] 15.3				
[2] 102	101	[2] 102	[3] 15.4		[3] 15.4				
[2] 103	102	[2] 103	[3] 15.5	[2] 86	[3] 15.5	all	103	12.12	17.0
	103	[2] 104	[3] 16.0	[2] 87	[3] 16.0				
	104	[2] 105	[3] TP						
		[2] 106							

Figure 3.2: Only a minority of browsers (about 41%) support access to a user's file system through the File System Access API. [Screenshot taken by the authors of this paper.]

git repository. The DirectoryHandle is then passed along to the Web Worker where it can recursively copy the contents of the '.git' folder into Emscripten's MEMFS implementation. As soon as the data is available in the simulated file system, libgit2 is tasked with exploring the repository's details to extract the branches, commits, and files contained within. This information is passed over to the main thread in order for the user to be able to select which branch, commit, and files they are interested in. This selection is then passed to the Web Worker, where libgit2 creates blame information for each file, and returns it back to the main thread in order to start the rendering process of the visualization.

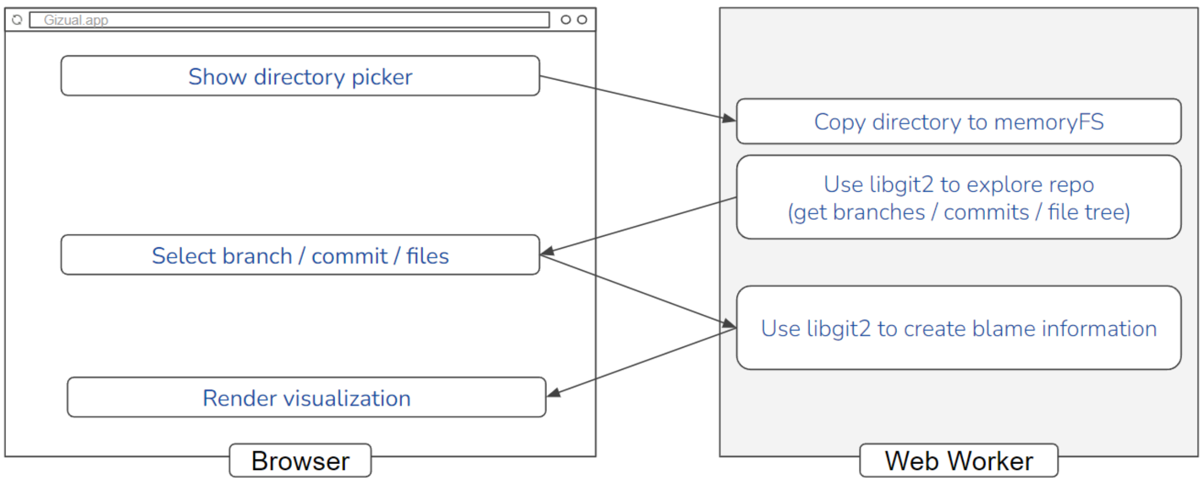

Gizual.app
Show directory picker
Select branch / commit / files
Render visualization
Browser
Copy directory to memoryFS
Use libgit2 to explore repo
(get branches / commits / file tree)
Use libgit2 to create blame information
Web Worker

Figure 3.3: The process flow of data acquisition. Multiple asynchronous callbacks between the main thread and the Web Worker are required to provide the necessary data for the visualization. [Diagram created by the authors of this report.]

Chapter 4

Visual Representation

The user interface of Gizual has two main components:

- *File Selection Panel*: the forms to select a commit and one or more files for visualization, on the lefthand side of the Gizual window.
- *Visual Overview Panel*: containing a visual representation (File Map) of each selected file.

A typical path through the Gizual user interface might be:

1. User selects a repository to analyse.
2. User selects a branch.
3. User selects a commit.
4. User selects a set of files to visualise.
5. User can see a File Map for each selected file.

4.1 Repository and File Selection

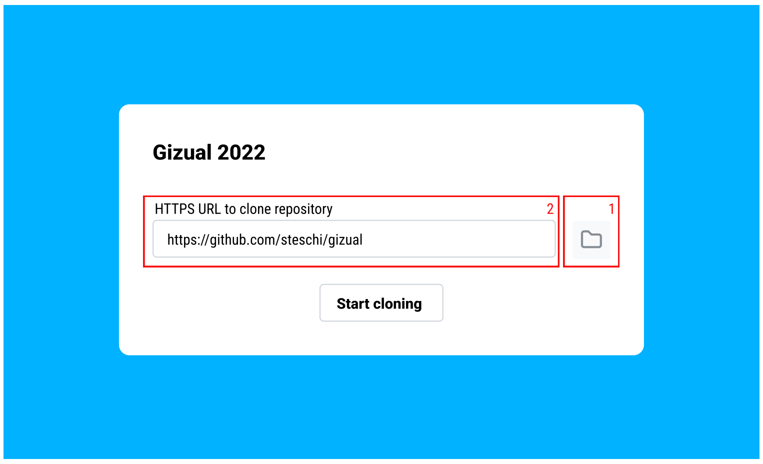
The first point of user interaction with Gizual is selecting a repository. For that, a separate screen with a small form is presented. This form acts as an “entrance” to the data visualisation. Two options for repository selection are possible, as shown in Figure 4.1. A user can upload a local Git repository directly using the Upload button on the right side of the form. Alternatively, a user can provide the external URL to the desired Git repository.

After clicking the "Start cloning" button, the user is redirected to the next view. The list of repository branches is fetched and displayed for selection in a dropdown. Once a branch has been selected, a list of its commits becomes available in a second dropdown, as shown in Figure 4.2. To proceed, the user selects the commit of interest. After that, a list of the project’s files becomes visible as they were at that point in time, as shown in Figure 4.3. Clicking on a folder expands it, clicking on a file selects it for visualization. To deselect a file, the user must click on it again.

4.2 File Map

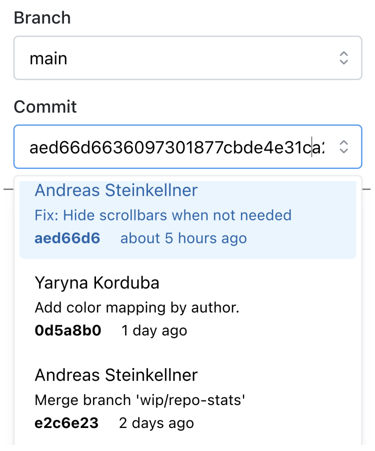
The core of Gizual is the File Map, a colour-coded representation generated for each selected file. File Maps are presented to a user in a CSS Grid layout. To conveniently represent multiple files side-by-side, the File Map containers are limited in height and width and are vertically scrollable. As soon as a file is selected, its File Map appears, and the legend becomes visible under the page header.

Each File Map is a visual representation of a file. Each line of the file is represented by a color-coded rectangle showing its indentation and length. These measures are related to the starting column and



https://github.com/steschi/gizual

Figure 4.1: Repository selection. Two options for selection are possible: direct upload of a Git repository using the Upload button (1) or entering the external URL (2). [Screenshot taken by the authors of this report.]


Branch
main

Commit
aed66d6636097301877cbde4e31ca:

Andreas Steinkellner
Fix: Hide scrollbars when not needed
aed66d6 about 5 hours ago

Yaryna Korduba
Add color mapping by author.
0d5a8b0 1 day ago

Andreas Steinkellner
Merge branch 'wip/repo-stats'
e2c6e23 2 days ago

Figure 4.2: Branch and commit selection. As soon as the user selects the branch, the list of its commits is fetched for selection. [Screenshot taken by the authors of this report.]

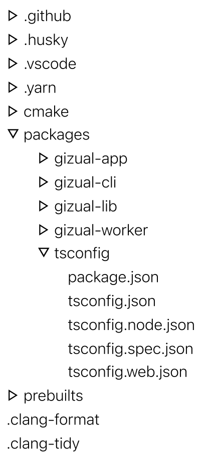

```
▷ .github
▷ .husky
▷ .vscode
▷ .yarn
▷ cmake
▽ packages
    ▷ gizual-app
    ▷ gizual-cli
    ▷ gizual-lib
    ▷ gizual-worker
    ▽ tsconfig
        package.json
        tsconfig.json
        tsconfig.node.json
        tsconfig.spec.json
        tsconfig.web.json
▷ prebuilts
.clang-format
.clang-tidy
```

Figure 4.3: File tree showing the folders and files of the particular commit. [Screenshot taken by the authors of this report.]

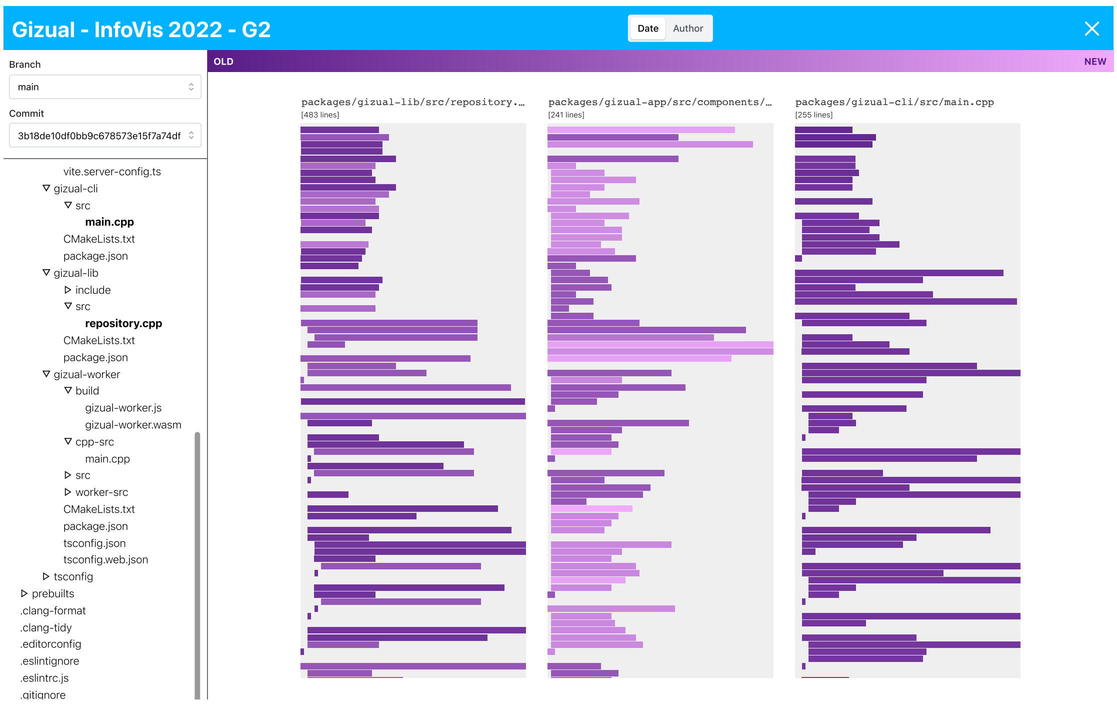


Figure 4.4: File Map color-coded by date of most recent change to each line of code (a linear color scale). [Screenshot taken by the authors of this report.]

number of characters in the line. Two options are currently available for color-coding each line: by date of last update, shown in Figure 4.4, and by author of last update, shown in Figure 4.5. The user can switch between the two options using the toggle button at the top of the Gizual window.

4.3 Interactivity

Hovering over a line of code displays a tooltip with the most recent author's name and email, and the timestamp of the last change, as shown in Figure 4.6. The positioning of the tooltip adjusts to the boundaries of its File Map. The tooltip is displayed above or below, or to the left or right of the cursor, depending on how close to the tooltip boundaries are to the File Map's boundaries.

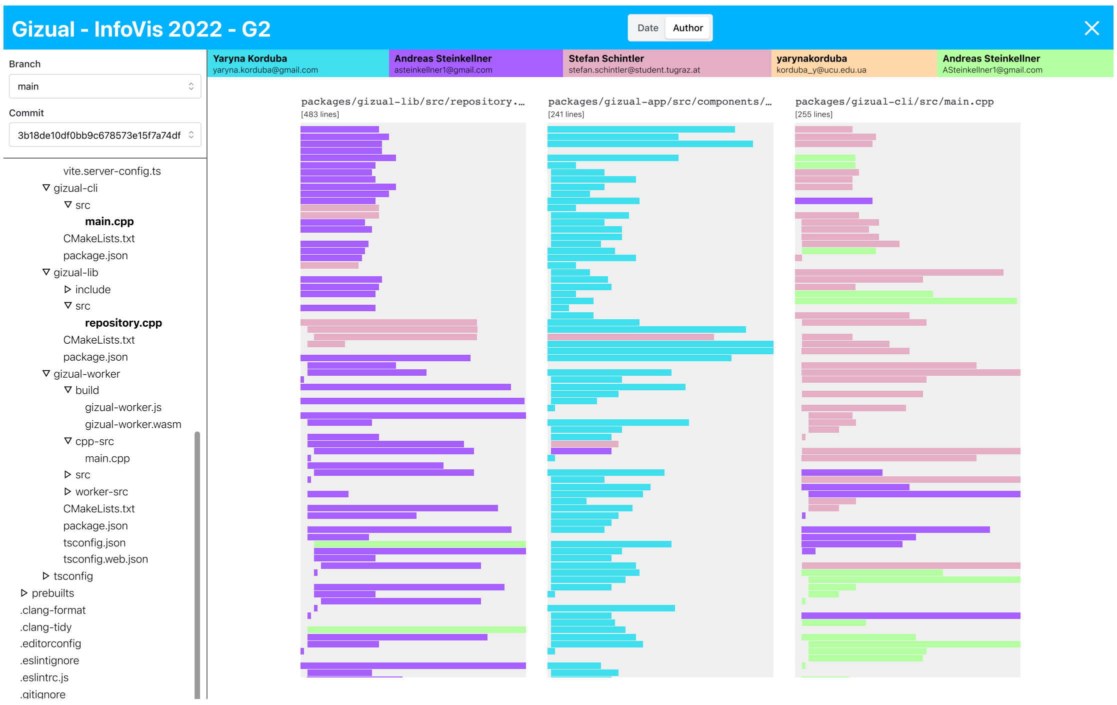


Figure 4.5: File Map color-coded by author of most recent change to each line of code (a categorical color pallete). [Screenshot taken by the authors of this report.]

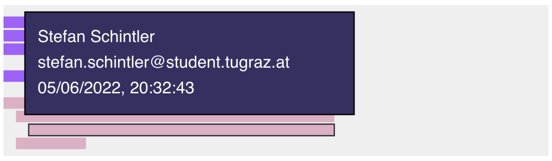


Figure 4.6: Hovering over a line of code in a File Map opens a tooltip with morer detailed information about that line of code. [Screenshot taken by the authors of this report.]

Chapter 5

Conclusions and Future Work

Since our work focused on creating an minimum viable product (MVP) for analyzing Git statistics, there is still room for improvement in terms of available features and overall usability. The following presents some of the things that could be improved in a future iteration.

The first obvious room for improvement in our design is the commit selection, where users select the desired commit to visualize. Problems occur when the number of commits in the desired branch is in the hundreds or thousands, where the UI will freeze for a second before displaying all of the desired data in a single box. This could be improved by "lazy loading" only a subset of commits at first, loading more as the user scrolls down the list. Additionally, the box is also not well optimized for very long commit messages, where an additional abbreviation step could be useful.

The file tree currently does not support any form of directory selection. In a future version, some form of selecting multiple files at once, or even entire directories, should be supported.

When color-coding by author, the legend is currently limited to displaying only the most recent six authors. This would need to be reworked for large repositories, where this is not indicative of the development history.

The current tooltip implementation is constrained to fit within the bounding box of its File Map, which can lead to clipping issues on narrow browser windows. Instead of rendering the tooltip for each file separately, it would need to be reworked into a global tooltip element which sits in its own layer on top of the visualization.

The visualization could benefit from actually displaying the code contents of each line, so users could easily explore the code snippets with the most recent changes or the author of their choice.

Many potential additional features spring to mind, including:

- Show the most productive contributor for a given repository, folder, or file.
- Show the average length of the files in a given folder.
- Group commits by hunks and provide a way to visualize entire hunks.
- List the total number of contributors for a given selection.
- Provide export functionality for the generated visualization.
- Allow users to pick up where they left off (using local storage or import/export functionality).

Gizual provides just a glimpse of what might be possible with a fully featured Git visualization tool based on modern frontend technologies. Much of the basic functionality has been implemented in the MVP design presented in this work, leaving room for many future development possibilities, both in terms of data acquisition using low-level Git libraries and sophisticated high-level visual interfaces.

Bibliography

Atlassian [2022]. *Git blame*. 06 Jul 2022. https://atlassian.com/git/tutorials/inspecting-a-repository/git-blame (cited on page 1).

Eick, Stephen, Joseph Steffen, and Eric Sumner Jr. [1992]. *Seesoft: A Tool for Visualizing Line Oriented Software Statistics*. IEEE Transactions on Software Engineering 18.11 (Nov 1992), pages 957–968. doi:10.1109/32.177365. http://www.sdml.cs.kent.edu/library/Eick92.pdf (cited on page 3).

Emscripten [2022]. *File System Overview*. 07 Jul 2022. https://emscripten.org/docs/porting/files/file_systems_overview.html (cited on page 7).

Feiner, Johannes and Keith Andrews [2018]. *RepoVis: Visual Overviews and Full-Text Search in Software Repositories*. Proc. 6th IEEE Working Conference on Software Visualization (VISSOFT 2018) (Madrid, Spain). 24 Sep 2018, pages 1–11. doi:10.1109/VISSOFT.2018.00009. https://ftp.isds.tugraz.at/pub/papers/feiner-vissoft2018-repovis.pdf (cited on page 3).

Git [2022]. *Git*. Software Freedom Conservancy, 07 Jul 2022. https://git-scm.com/ (cited on page 1).

Hamidovic, Amel, Jakov Matic, Günther Kniewasser, and Andreas Wöls [2018]. *RepoVis Timeline Extension*. 706.057 Information Visualisation SS 2018 Project Report. Graz University of Technology, 16 Apr 2018. https://courses.isds.tugraz.at/ivis/projects/ss2018/ivis-ss2018-g4-project-reposvis-timeline.pdf (cited on page 3).

libgit2 [2022]. *libgit2*. libgit2 Contributors, 07 Jul 2022. https://libgit2.org/ (cited on page 5).

LLVM [2022a]. *Clang*. LLVM Developer Group, 07 Jul 2022. https://clang.llvm.org/ (cited on page 6).

LLVM [2022b]. *LLVM Compiler Infrastructure Project*. LLVM Developer Group, 24 Jun 2022. https://llvm.org/ (cited on page 6).

Salomonsen, Peter [2022]. *Wasm-git*. 07 Jul 2022. https://github.com/petersalomonsen/wasm-git (cited on page 7).

Synopsys [2022]. *Compare Repositories*. Open Hub, 06 Jul 2022. https://openhub.net/repositories/compare (cited on page 1).

W3C [2022]. *WebAssembly*. 07 Jul 2022. https://webassembly.org/ (cited on page 6).

Youngtaek, Kim, Kim Jaeyoung, Jeon Hyeon, Kim Young-Ho, Song Hyunjoo, Kim Bohyoung, and Jinwook Seo [2021]. *Githru: Visual Analytics for Understanding Software Development History Through Git Metadata Analysis*. IEEE Transactions on Visualization and Computer Graphics 27.2 (Feb 2021), pages 656–666. doi:10.1109/TVCG.2020.3030414. https://arxiv.org/pdf/2009.03115.pdf (cited on page 3).

Zakai, Alon [2022]. *Emscripten*. 07 Jul 2022. https://emscripten.org/ (cited on page 6).