International Journal of Advance Research in Science and Engineering
Volume No. 14, Issue No. 05, May 2025
www.ijarse.com
IJARSE
ISSN 2319 - 8354

Snapshot Powered Version Control System For Content Production

Sudhesh Hegde[1], Asst. Prof Geena George[2], Sujata Jha[3], Owais Iftikhar Dhansay[4]

[1,2,3,4] *Department of CSE, AMC Engineering College, Bengaluru, India*

hegdesud2002@gmail.com, geena.george@amceducation.in

Sujatajha078@gmail.com, Owais.dhansay@gmail.com

ABSTRACT

FLVCS (FL Version Control System) is a version control solution designed for managing complex project files, particularly in creative fields like music production. It uses a "Snapshot Addressed Content Storage" method, where each commit captures the full project state and is stored in a uniquely hashed folder. This approach enables efficient tracking, recovery, and sharing of binary files that traditional version control systems struggle to handle. The paper outlines the architecture and benefits of this snapshot-based system for non-linear, media-rich workflows.

Keywords: *Version Control System (VCS), Snapshot Storage, Content-Addressed Storage, Digital Audio Workstation (DAW)*

I. INTRODUCTION

In the digital age, content creators—such as music producers, video editors, and photographers—work with complex project files that evolve rapidly over time. However, unlike software developers who benefit from robust version control systems like Git, creators dealing with large binary files lack a reliable way to track their project's history, manage changes, or recover from data loss. Traditional cloud backups and manual file duplication are inefficient, error-prone, and offer limited control over versioning. FLVCS (FL Version Control System) addresses this gap by offering an intuitive, snapshot-based version control system tailored specifically for non-textual, media-rich workflows [1]. It enables users to commit the full state of a project folder, store it with a unique hash, and retrieve or branch from any previous version when needed. With both a command-line tool and a client application, along with a supporting web interface, FLVCS empowers creators to safeguard their work, streamline their process, and collaborate more confidently. This paper delves deeper into the explanations and the solution, which is divided into --- sections: (1) Problem Statement, (2) Related Work, (3) Proposed System, (4) Architecture, (5) Implementation

II. PROBLEM STATEMENT

Large, complicated project files that change regularly throughout the production process are often worked with by creative professionals including designers, video editors, and music producers. Unlike software developers, they do not have access to a consistent, effectively handling binary file version control system. Most depend on

generic cloud storage options—which are not meant for structured version tracking—or manual using names like "final_v2" or "project_backup." This method provides little defense against data loss brought on by hardware failure, accidental overwrites, or file corruption, makes teamwork difficult, and is prone to mistakes. Recovering a specific version that worked can prove time-consuming and uncertain when something goes wrong. So, there is need for a system that not only saves versions but also lets users quickly navigate, restore, and control their project history free from technical overhead.

III. RELATED WORK

Version control systems (VCS) such as Git [2] have become integral in software development for managing code changes. However, Git and similar systems are optimized for text-based files and fail to handle large binary files efficiently. Git and other traditional VCS tools rely on file diffs to track changes, which makes them unsuitable for creative industries where project files frequently contain media-heavy assets like audio, video, and images. Basic file versioning is provided by cloud storage services like Dropbox and Google Drive, but they lack the structure and control of a dedicated VCS. Without the ability to commit and branch particular project versions, these services usually only support file restores or the tracking of simple changes. FL Studio, Adobe Premiere, and Final Cut Pro are examples of digital audio workstation (DAW) tools that emphasize automatic backups but lack the powerful, user-driven versioning system required for long-term project management. Furthermore, these systems are unable to offera systematic method for keeping track of various project states with distinct identifiers. By capturing and storing the complete project folder at each commit, FLVCS overcomes these constraints and gives creators a straightforward, dependable, and all-inclusive version control system. FLVCS is especially well-suited for the intricate, non-diffable files present in creative workflows because, in contrast to conventional systems, it stores full snapshots of the project rather than depending on diffs.

IV. PROPOSED SYSTEM

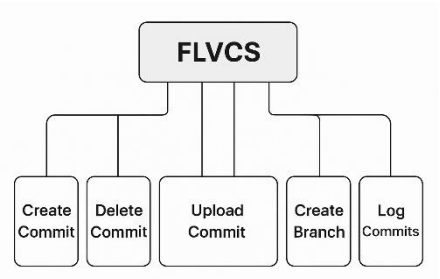

Fig 1.1 Proposed System of FLVCS

By using a snapshot-based system created especially for handling complicated, binary project files frequently found in creative workflows, FLVCS (FL Version Control System) offers a novel approach to version control. In contrast to conventional version control systems that depend on file diffs, FLVCS makes sure that all file changes are saved in a comprehensive, self-contained snapshot by capturing and storing the entire project folder at each commit. A hash serves as a version marker, uniquely identifying each snapshot and making it simple for users to track and retrieve particular project versions. FLVCS gives users flexibility based on their preferred

workflow by supporting both a client application and a command-line interface (CLI). Users can also manage their project files across devices, download particular versions, and view their commit history using a web interface. The system offers a dependable and easy-to-use version control solution for managing projects where conventional VCS systems are ineffective or impractical, catering to the particular requirements of content creators. FLVCS protects against data loss from file corruption or system failures by integrating local and cloud storage, ensuring that project data is securely stored and readily recoverable.

Both a graphical client application and a command-line interface (CLI) are available as the system's extensive feature set of version control. With a user-defined message for future reference, users can create a commit that copies and saves the entire project state. When necessary, users can clear out outdated or superfluous versions by deleting a commit. FLVCS allows users to upload commits to the cloud for redundancy and accessibility, guaranteeing device backup and safeguarding against hardware malfunctions. On the other hand, users can restore or switch to previous iterations of a project by downloading particular commits from local storage or the cloud.

FLVCS offers the capability of branch creation in addition to linear versioning, which enables users to experiment with different project development paths without erasing the primary workflow. With a flexible structure comparable to Git but tailored for large binary workflows, each branch keeps its own independent commit history. All commits can be logged by the system, providing users with a timeline of the development of their project along with metadata like timestamps, commit messages, and branch associations.

By enabling users to manage and view their project history online, a web-based interface improves usability. Tools for viewing commit logs, downloading versions, managing storage, and smoothly switching branches are available through this interface. When combined, these characteristics make FLVCS a strong and intuitive system that bridges a significant version control gap in content creation domains where traditional systems are inadequate.

V. ARCHITECTURE

The FLVCS (FL Version Control System) architecture is designed to offer a comprehensive version control experience that is tailored for workflows in creative projects. The client application, web interface, and backend service are the primary three components of the system, and they all work together harmoniously to provide versioning and cloud-based synchronization. The downloadable FLVCS client, which is built with Python and PyQt5, includes both a graphical user interface (GUI) and a command-line interface (CLI) for users to interact with the platform. In addition to supporting authenticated push and pull actions to the cloud, this client facilitates local operations like initializing a repository, committing project states and managing branches.

The main location for user authentication, repository browsing, commit history inspection, and data downloads is the FLVCS web application, which is built with React and Node.js and is hosted on Netlify. Users can register, log in, and access their repositories from a distance with its help. The backend, which is housed on Google Cloud Platform's App Engine, controls the long-term archiving of user metadata and commit snapshots. Quart, a Python ASGI web framework, was used in the design of the backend. It integrates with Google Cloud Storage to store binary commit data, and a structured database solution handles user and commit metadata. Commit data is uploaded to cloud storage and related metadata is sent to the backend when a user starts a push

from the client.

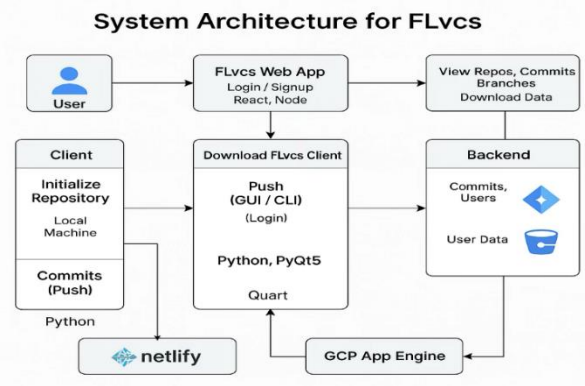

Fig 1.2 Architecture Diagram of FLVCS

The web application allows users to explore branches, download particular versions, and view their commit history, but all client-based operations are still closely linked to the local repository environment. Platform independence, strong data integrity, and system scalability are all guaranteed by this modular architecture. By providing flexibility, dependability, and an intuitive user interface through its unified ecosystem, FLVCS thereby bridges the gap for version control systems designed for creative projects that rely heavily on binary data.

The Client Module, which runs locally on the user's computer, serves as the framework for FLVCS. It allows users to set up a repository inside any project folder, after which it starts monitoring the directory's condition. When a user commits, the client takes a snapshot of the entire directory, compresses it, and gives it a unique SHA-256 hash [3]. This commit is stored locally and may also be pushed to the cloud, along with user-supplied metadata like commit messages. The client has a graphical user interface (GUI) for ease of use in more creative domains and a command-line interface (CLI) for automation and developer-centric workflows. These two interfaces provide a cross-platform experience and are powered by Python and PyQt5.

Users can manage their FLVCS repositories from any browser with the help of the Web Interface, which was created with React and Node.js. Users can view their repositories, browse through their commit history, examine branches, and download any version of their cloud-stored project by logging in through the web application. REST APIs are used by the web application to securely communicate with the backend. This frontend layer, which is hosted on Netlify, guarantees high availability and responsiveness while facilitating continuous deployment.

The central processing and data management component of FLVCS is the Backend Service. The backend, which was constructed with the Quart asynchronous Python framework, manages data orchestration between the client, cloud storage, and user interface, as well as authentication and API request routing. While big project snapshots are kept in Google Cloud Storage, user metadata, commit hashes, timestamps, and branch relationships are kept in a structured database. For large binary files, this hybrid structure keeps storage costs low while guaranteeing scalability and fast metadata retrieval.

The App Engine and Cloud Storage services offered by Google Cloud Platform (GCP) are utilised by the Cloud Infrastructure. To ensure safe and redundant storage, the client uploads the compressed snapshot to a bucket in GCP Storage that is specifically assigned when a user pushes a commit. The backend is hosted by GCP App

Engine, which offers autoscaling features and smooth Firebase integration for user authentication. In addition to safeguarding user data against device loss or corruption, this architecture makes cross-device access and teamwork easier.

Lastly, a uniform API schema across all components allows FLVCS to preserve data flow integrity. Secure REST API calls to the backend are triggered by user-initiated actions from the client or web application. Depending on the operation (e.g., commit creation, deletion, download, or branch switching), the backend replies with the relevant acknowledgements or data payloads. Future scalability is supported by this modular and consistent design, which allows for the addition of collaborative features, per-file versioning, and integration with outside creative tools.

VI. IMPLIMENTATION

The FLVCS (FL Version Control System) implementation prioritises dependability, extensibility, and simplicity. The system's fundamental idea is to take full snapshots of creative project folders rather than figuring out file differences. This choice makes it possible for FLVCS to support binary-heavy projects like music production files, timelines for video editing, design assets, and other non-diff-friendly formats like code. To ensure lossless version recovery, the tool records a project folder's complete state at the moment of each commit.

Both a Graphical User Interface (GUI) and a Command Line Interface (CLI) are supported by the Python-based client-side application. Advanced users who need automation, scripting, or integration with other terminal-based tools are the target audience for the CLI. The PyQt5 framework was used to create the GUI, which is intended for general users who like visual interfaces. Users can initialise repositories, create and delete commits, view commit logs, create and switch branches, and upload or download cloud snapshots using either version of the client. To keep track of commits and local configuration information, a hidden metadata directory is created inside the user's project folder when a repository is initialised.

All the project directory's contents are compressed into an archive and hashed with SHA-256 after a commit is made [4]. Both locally and on the cloud, the commit is stored and retrieved using this hash, which serves as a unique commit identifier. Additionally, metadata like the commit message, timestamp, branch association, and user identity are recorded by the commit process. The commit can be kept locally until it is manually uploaded, or users can push it to the FLVCS cloud backend right away. This adaptability facilitates collaborative remote storage as well as offline workflows.

React for the front end and Node.js for the middleware API that connects to the Python-based back end are used in the development of the web application. Users can sign up and log in, browse their commits, view repositories, switch branches, and download any version straight from the platform. Firebase is used for all authentication and session management, and it is closely linked with the backend for user role management and token validation. Secure REST APIs are used by the web application to connect to the backend. Quart, an asynchronous Python web framework that facilitates WebSocket communication and high concurrency, is used to develop the backend. The backend manages file retrieval, metadata management, branch switching, and API routing for commit uploads. It offloads binary data (the commit archives) to Google Cloud Storage and stores metadata in a structured format. The speed and effectiveness of tasks like listing commits, logging metadata, and

confirming the existence of particular commits are enhanced by this division of metadata and bulk data.

Cloud services are made available via Google Cloud Platform (GCP) to preserve security and scalability. Because the backend is powered by GCP App Engine and has automatic scaling enabled, it can manage growing traffic volumes without the need for human intervention. User-pushed commits, which are uniquely identified by their hashed name, are uploaded to a secure bucket in Google Cloud Storage. This makes retrieval easy and verifiable in addition to guaranteeing data consistency.

Additionally, FLVCS has branching support, which is similar to Git but was created with simplicity in mind. To test out different project directions or workflows, users can create branches. Branches can be switched at any time from the web application or client and are stored as lightweight pointers to the most recent commit hash. This enables artists to continue working on a project in several creative directions without erasing earlier work.

Overall, FLVCS's deployment shows a strong focus on easy-to-use version control for non-code content, supported by effective local tooling and scalable cloud infrastructure. For creative professionals, the system provides dependable and user-friendly project versioning through support for both CLI and GUI interfaces, full snapshot commits, secure cloud backups, and rich metadata tracking.

VII. USE CSES AND APPLICATIONS

The main use of FLVCS is to give content producers who work with non-textual project files strong and dependable version control. Conventional version control systems, such as Git, have trouble handling binary formats because they are made for source code. FLVCS, on the other hand, takes snapshots of entire project directories, which makes it perfect for fields like graphic design, photography, video editing, and music production where files are frequently binary and incompatible with conventional diff-based tracking.

Using digital audio workstations (DAWs) like FL Studio, Ableton Live, or Logic Pro in music production settings is one of the most important applications of FLVCS. Producers may want to go back to a previous version of the project after experimenting with various instrument configurations, plugins, or arrangements. FLVCS eliminates the need to manually duplicate entire project folders by allowing every session to be committed and instantly recovering older versions. Additionally, users can experiment with different versions of their project by branching off without affecting the main timeline.

FLVCS can be used by video editors who use programs like Adobe Premiere Pro, DaVinci Resolve, or Final Cut Pro to commit different stages of editing. A distinct cut, effect experiment, or storyline may be represented by each version. Editors can quickly revert to a stable version of the timeline if an edit goes awry or corrupts it. This feature eliminates the hassle of creating multiple backup folders and significantly lowers the risk of project loss due to file corruption or errors.

It can be challenging to manually monitor the iterative changes made by photographers and digital artists using programs like Photoshop, Lightroom, Illustrator, or Blender. Artists can use FLVCS to commit each step of their workflow, including layer changes, color grading, and retouching, and then switch between them for comparison. They can also experiment with different paths without worrying about losing their progress thanks to the branching capability.

3D modelers and game developers can also benefit from FLVCS. These experts frequently deal with project files, external plugins, and large assets that are incompatible with line-based versioning. Teams or individuals

can manage version histories of entire models, animation rigs, or game scenes using FLVCS. In addition to maintaining multiple iterations and retrieving backups when necessary, developers can create branches for new features.

Collaborative and educational settings represent yet another significant use case. Teachers can model effective project management techniques in the classroom by encouraging students to use FLVCS to version their creative assignments. Members of collaborative teams are able to share progress and commit changes independently without erasing one another's work. FLVCS can be used by lone learners working on individual creative projects to monitor progress over time and gain knowledge from past iterations.

When developing software for creative tools, developers can use FLVCS to capture non-code artefacts like configuration files, assets, and compiled libraries while working on simulation projects, graphic engines, or audio plugins. Because of their size or format, these are frequently left out of Git repositories. FLVCS guarantees that these parts are still recoverable and versioned when required.

Moreover, snapshot-based versioning is advantageous for scientists and researchers working on simulations, machine learning models, or data-driven visualizations. Because binary models, datasets, and environment setups are frequently included in these projects, FLVCS enables them to organize and store each experimental setup and output state, which can be essential for auditing and reproducibility.

To sum up, FLVCS is a flexible tool made to bridge the gap in non-code creative project management that traditional VCS systems left. It is appropriate for a broad range of applications, from research and education to the creative industries, due to its capacity to capture, store, restore, and branch out entire project states. FLVCS's straightforward user interface and robust backend storage make it simple for developers, artists, producers, and editors to incorporate into their daily workflows.

VII. CONCLUSION

For creative professionals who deal with binary and non-textual data on a regular basis, FLVCS offers a feature-rich version control system. FLVCS guarantees total project fidelity across all saved states by abandoning conventional line-based diffs and adopting a snapshot-based architecture. For music producers, video editors, graphic designers, and other digital artists, this method is ideal because it ensures that each commit appropriately captures the entire structure and content of a project at any given time.

The system can be used by users with different levels of technical expertise because it is accessible in two modes: a graphical client and a command-line interface. A web application's integration also improves usability by facilitating cloud backups, remote access, and the safe download or restoration of commits. With capabilities like branching, cloud syncing, commit creation, deletion, and version history logging, FLVCS emulates the stability of conventional VCS tools while meeting use cases that they frequently overlook.

Technically speaking, FLVCS is scalable and effective due to design choices like snapshot hashing, cloud-based storage on Google Cloud, and a lightweight backend driven by Quart. By separating metadata from blob storage and using hashed commit integrity, security and performance have been considered. The project fills a glaring market gap where creative users frequently rely on duplicate folders and manual backups because they lack a centralized, consistent method to manage versions of their work.

FLVCS currently functions as a single-user model, but it would become enterprise-useable if it were expanded

to accommodate team workflows with merge capabilities, real-time synchronization, and permission-based branching. For distributed content creation teams, like co-editors, design studios, or collaborative music bands, this could have a special effect.

Scripting interfaces and automation hooks could also be added. These would enable users to configure post-processing scripts, triggers for automatic commits, and integration with other programs, such as video editing software and digital audio workstations (DAWs). Versioning within current creative pipelines would be made even simpler by a smooth integration.

Lastly, investigating machine learning integrations for version comparisons, intelligent commit recommendations, or anomaly detection (such as project corruption alerts) may improve user experience. These features would actively help users choose the most pertinent restore points and maintain the health of the project.

In conclusion, FLVCS offers a much-needed version control advancement that is suited to contemporary creative workflows. It gives users complete control over their creative process and project history by emphasizing simplicity, completeness, and dependability. FLVCS is positioned to become an essential tool in the ecosystem of digital content creation with planned advancements in automation, collaboration, and storage optimization.

REFERENCES

[1] A. Panda, S. R. Sarangi, and Department of Computer Science and Engineering, Indian Institute of Technology Delhi, “SnapStore: a snapshot storage system for serverless systems,” journal-article, 2023. [Online]. Available: https://www.cse.iitd.ac.in/~srsarangi/files/papers/snapstore.pdf

[2] Ghodke, Gayatri & Chavan, Trupti. (2024). An Overview of Git. International Journal of Scientific Research in Modern Science and Technology. 3. 17-23. 10.59828/ijsrmst.v3i6.216. https://www.researchgate.net/publication/384049322_An_Overview_of_Git

[3] “SHA-512/256,” *IEEE Conference Publication | IEEE Xplore*, [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5945260

[4] “An optimized pipelined architecture of SHA-256 hash function,” *IEEE Conference Publication | IEEE Xplore*, Dec. 01, 2017. https://ieeexplore.ieee.org/abstract/document/8303943