Commit Man (Review III)

GitHub Repository: https://github.com/Atharva-Gundawar/Software-Enginering-Project

Version: 1.1

Date Created: 2021.05.30

List of Contributors

Name	Initials	Organization	E-Mail
Souris Ash (19BCE0392)	SA	VIT, Vellore	souris.ash2019@vitstudent.ac.in
Atharva Gundawar (19BCE0388)	AG	VIT, Vellore	atharva.gundawar2019@vitstudent.ac.in

Acknowledgment

I would like to express my special thanks and gratitude to my faculty Swathi JN who gave us the golden opportunity to do this wonderful project on the topic version control system, which also helped us in doing a lot of Research and we gained new insights into some of the most prestigious .

Executive Summary

Commit-Man is a version control system that helps the user keep track of and manage different versions of their files. It features a CLI and a GUI - the complete set of features are available in the CLI, while the GUI features a limited set of features currently (primarily repository browsing functions and viewing diffs across versions).

Table of Contents

List of Figures

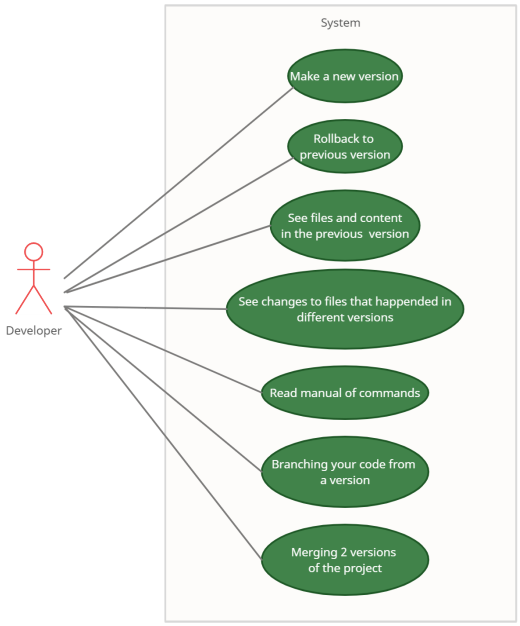


Fig 1.1 Use Case Diagram

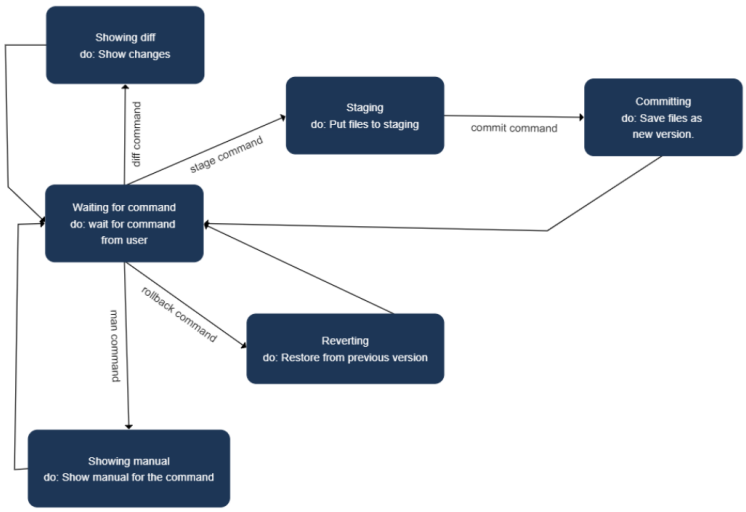


Fig 1.2 State Transition Diagram

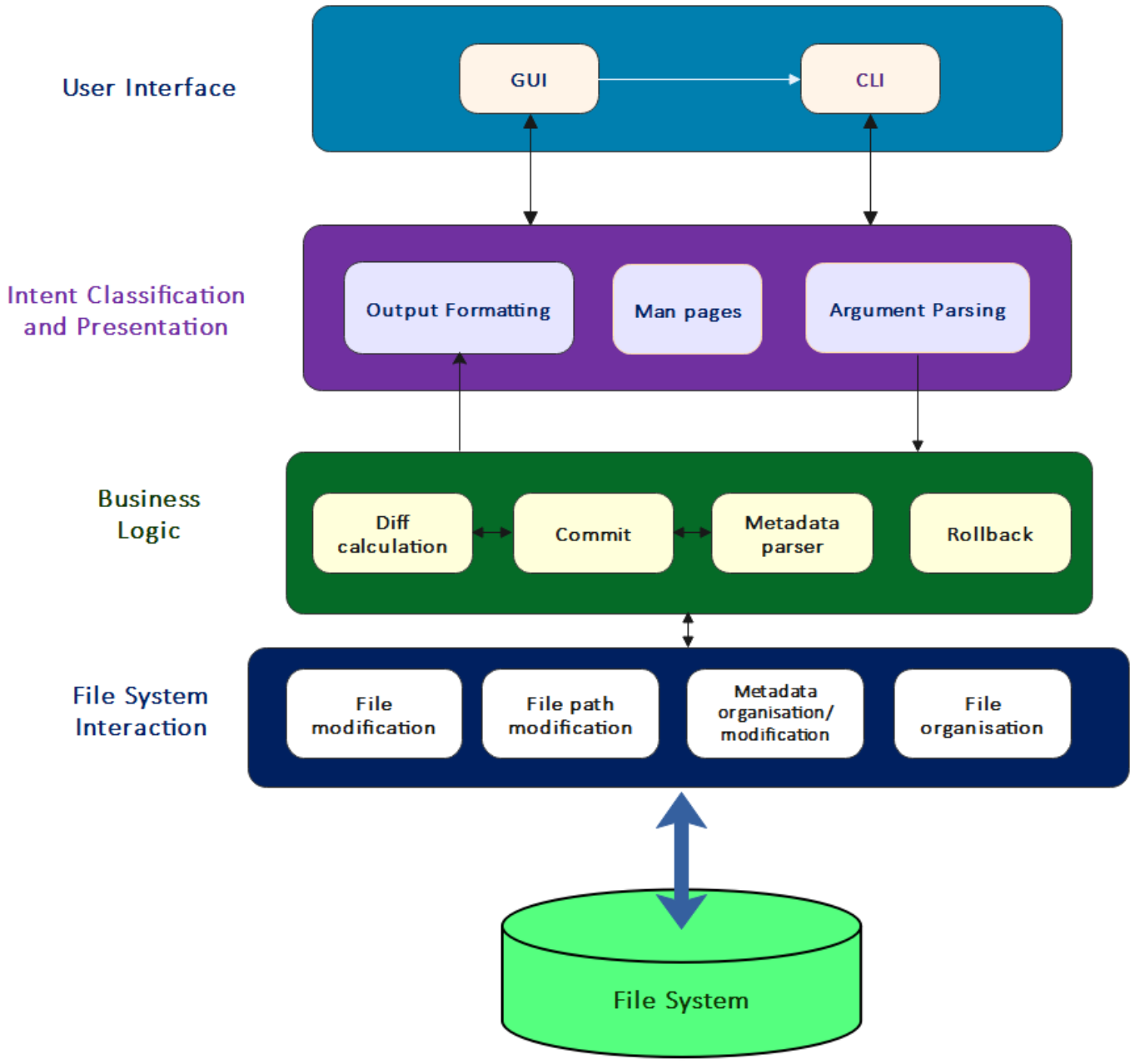


Fig 2.1 Architecture Diagram

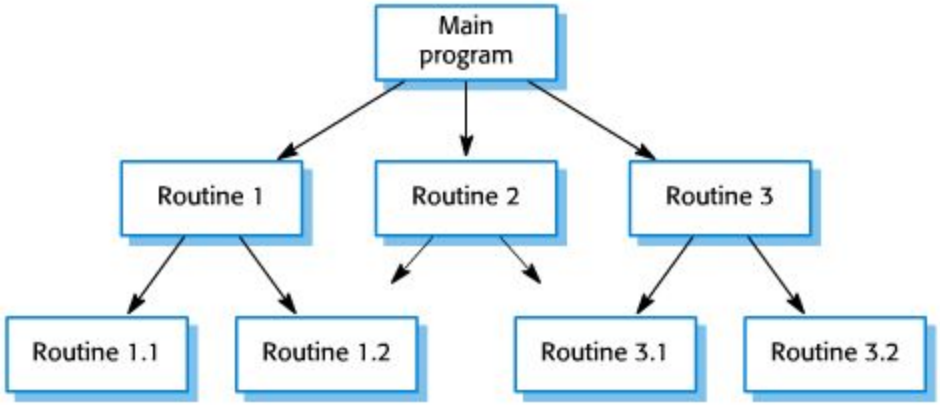


Fig 2.2 Control model- Centralized Model

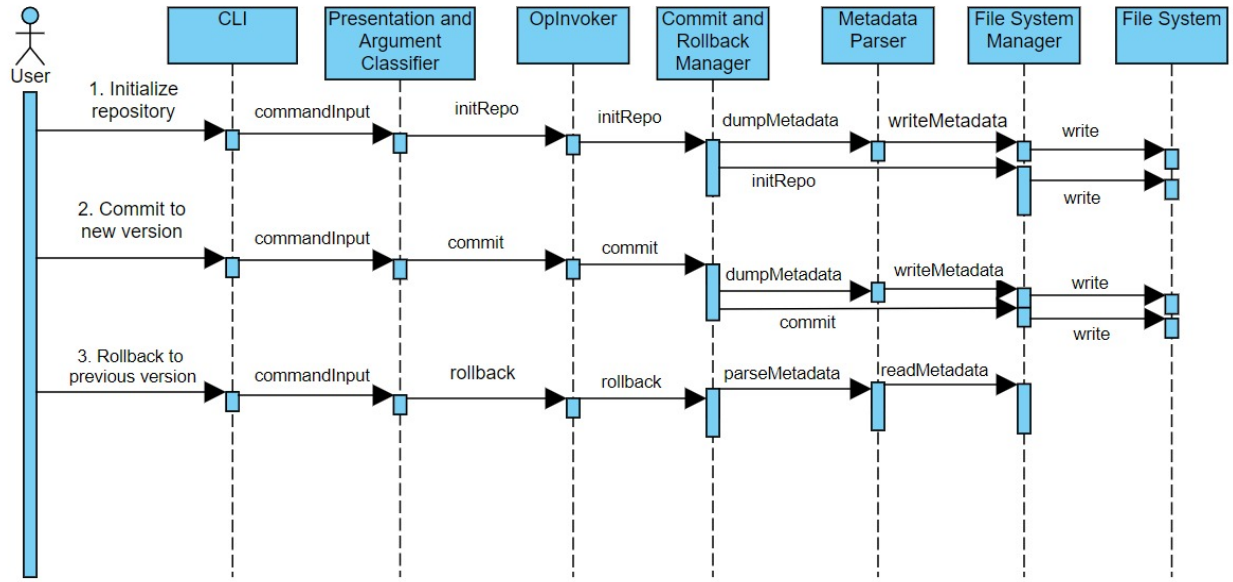


Fig 3.1 Sequence Diagrams - Commit and Rollback

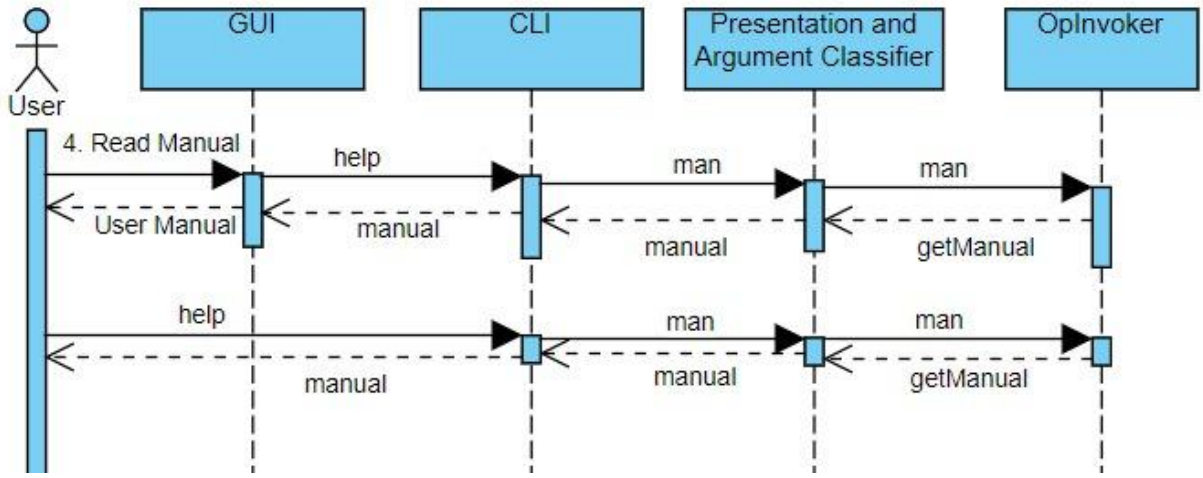


Fig 3.1 Sequence Diagrams - View Manual

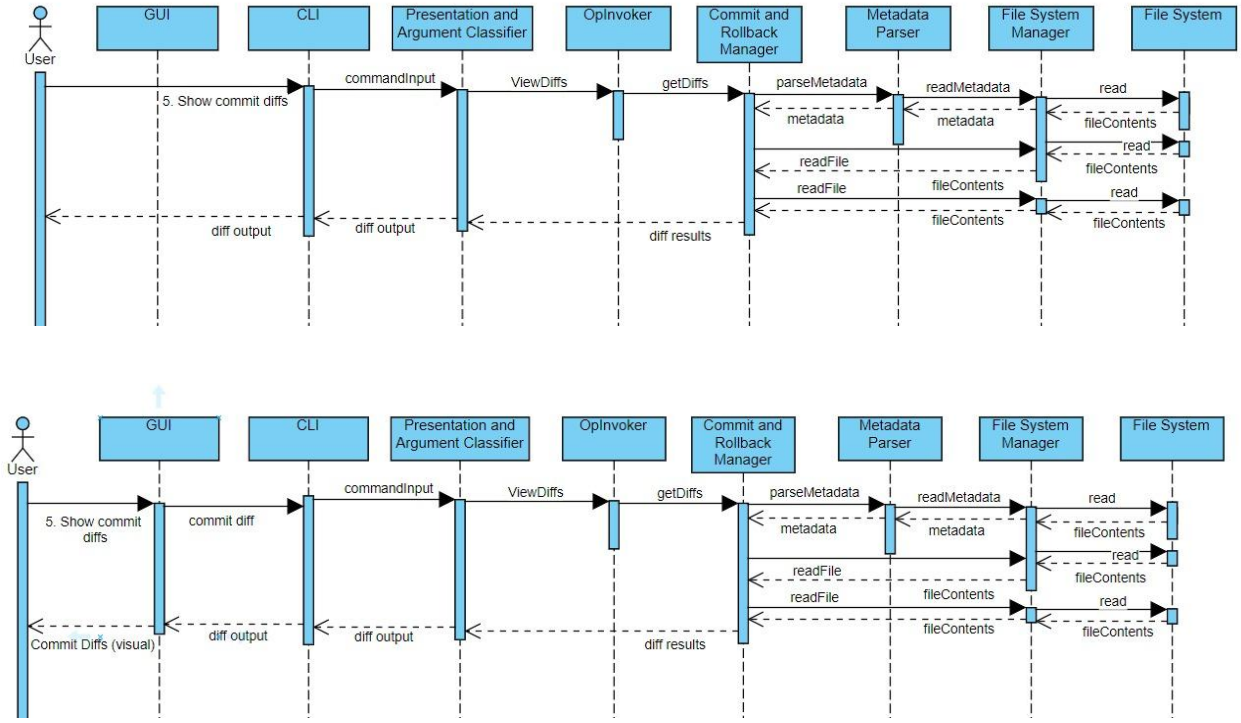
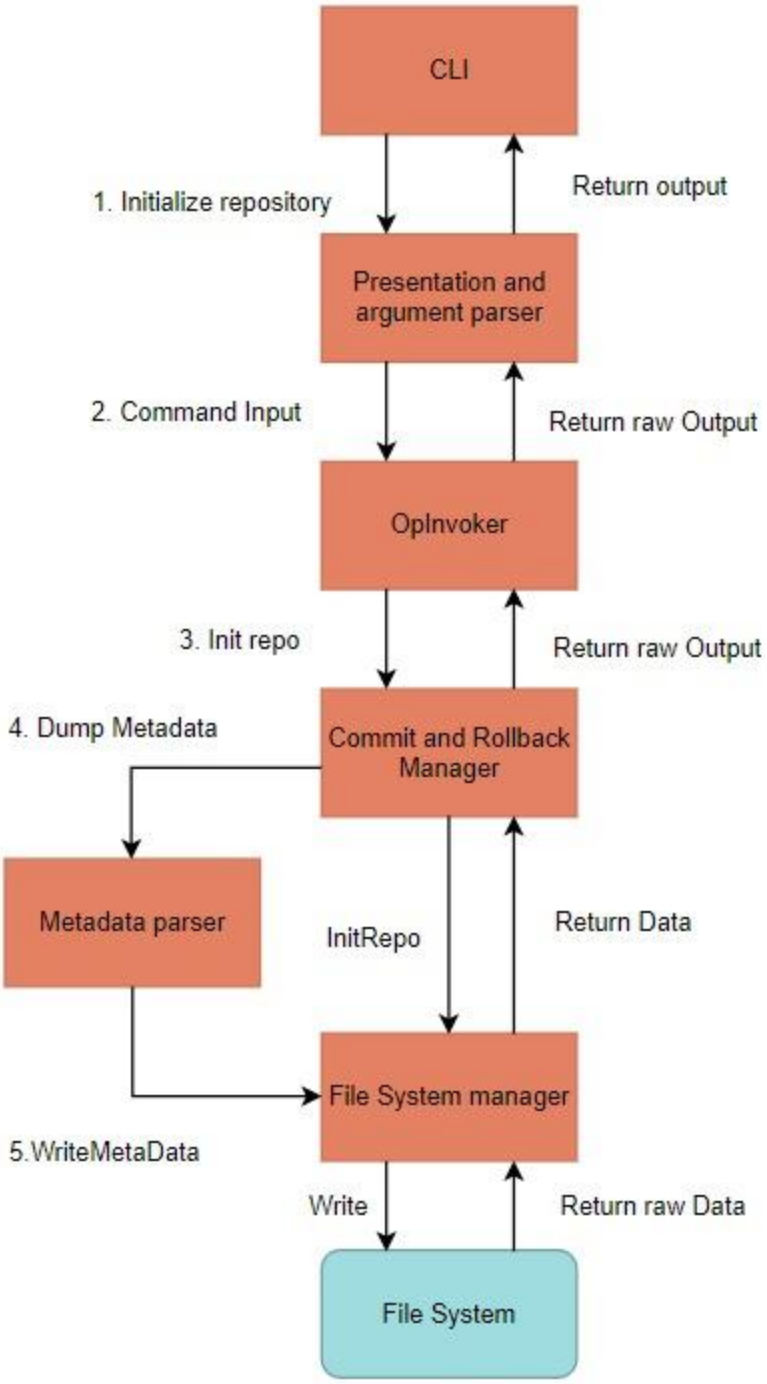


Fig 3.1 Sequence Diagrams - Commit Diffs



Commit and make a new version: Rollback to a previous version:

Fig 4.1 Collaboration Diagrams - Init

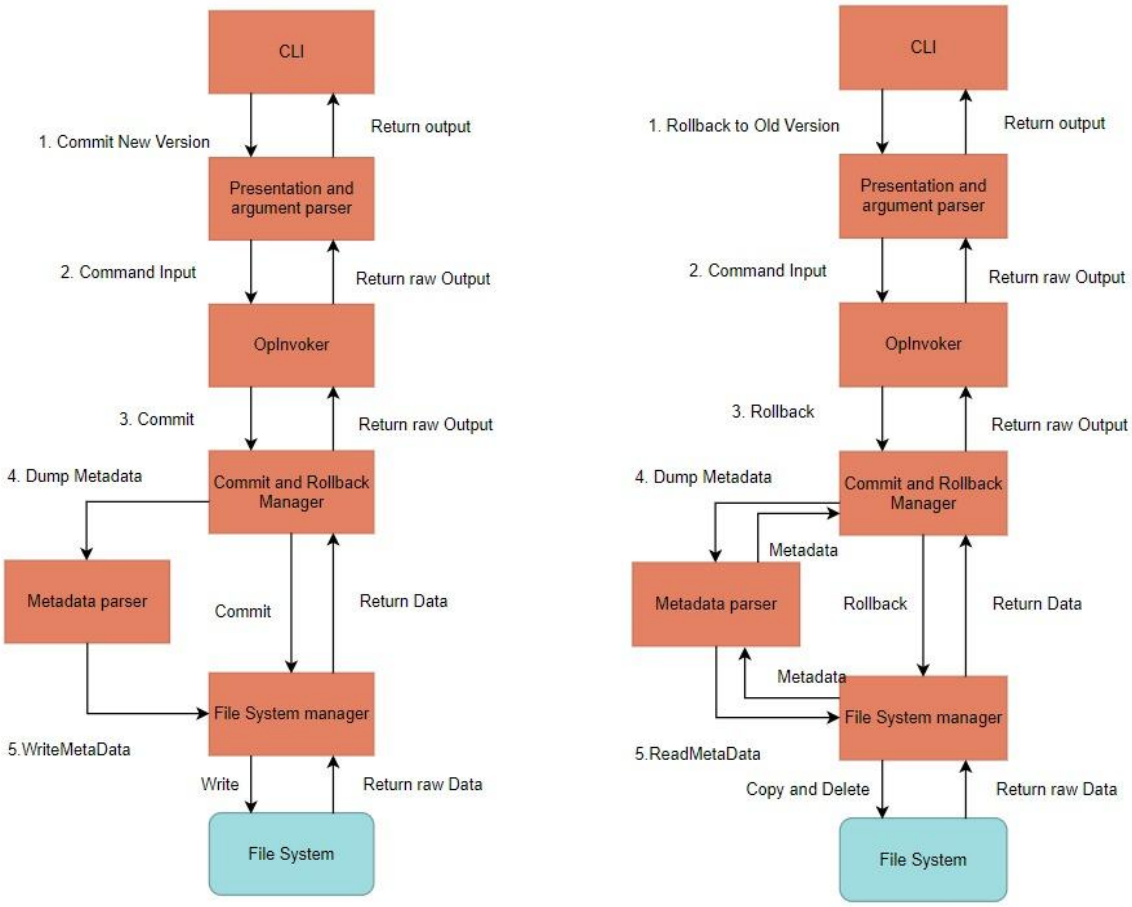


Fig 4.2 Collaboration Diagrams - Commit and Rollback

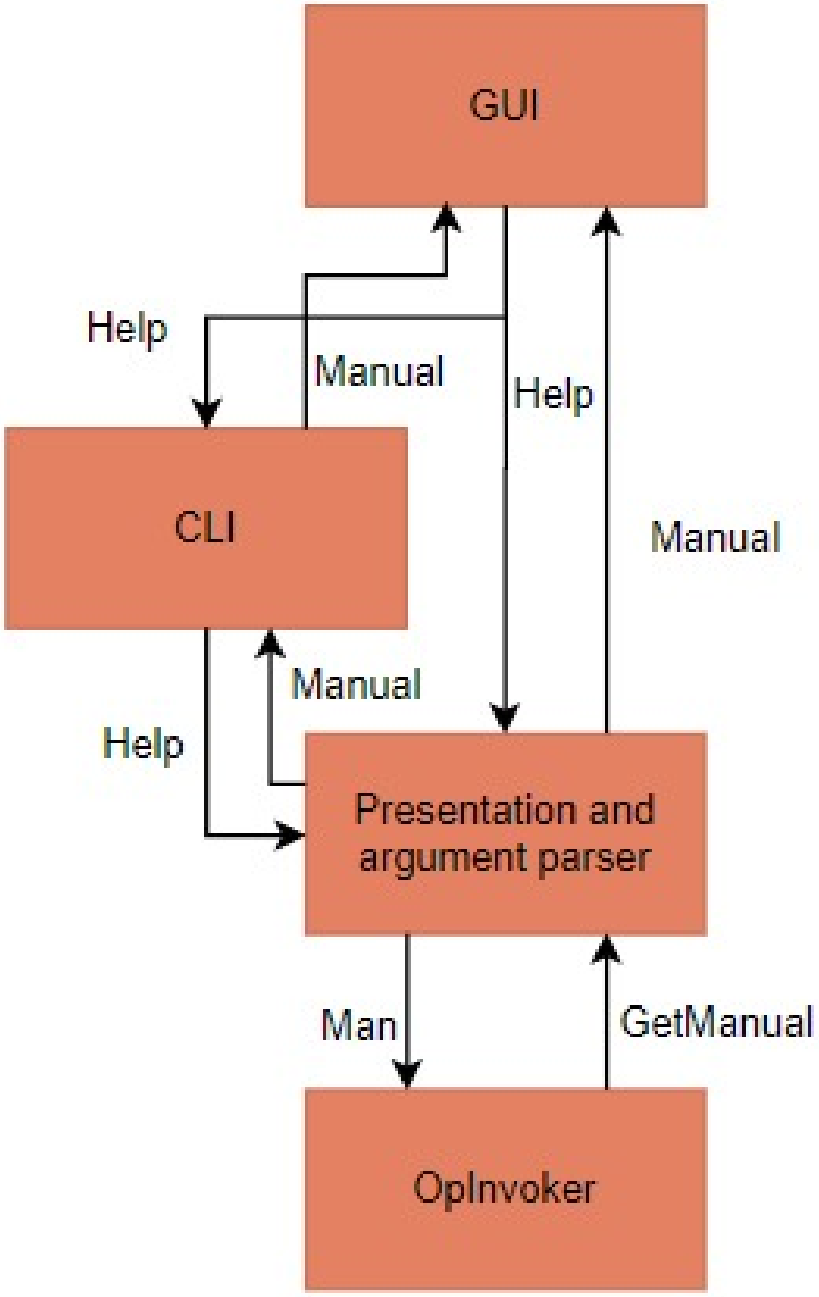


Fig 4.1 Collaboration Diagrams - View Manual

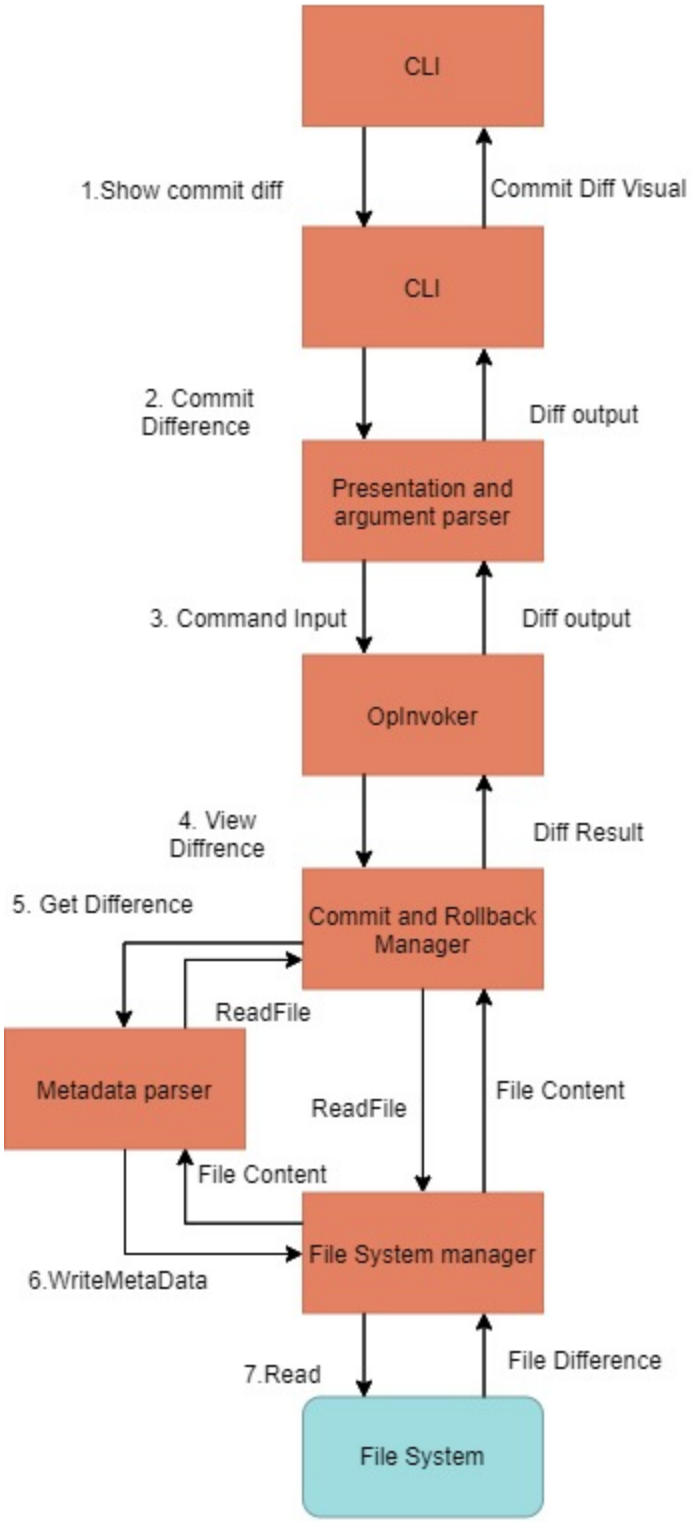


Fig 4.1 Collaboration Diagrams -Commit Diffs

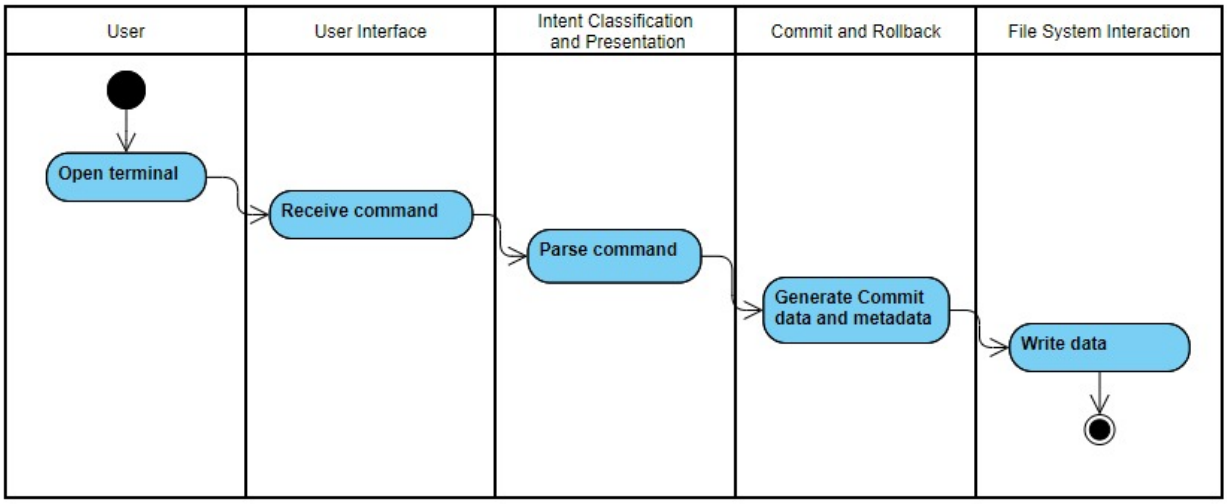


Fig 5.1 Swimlane Activity Network - Commit

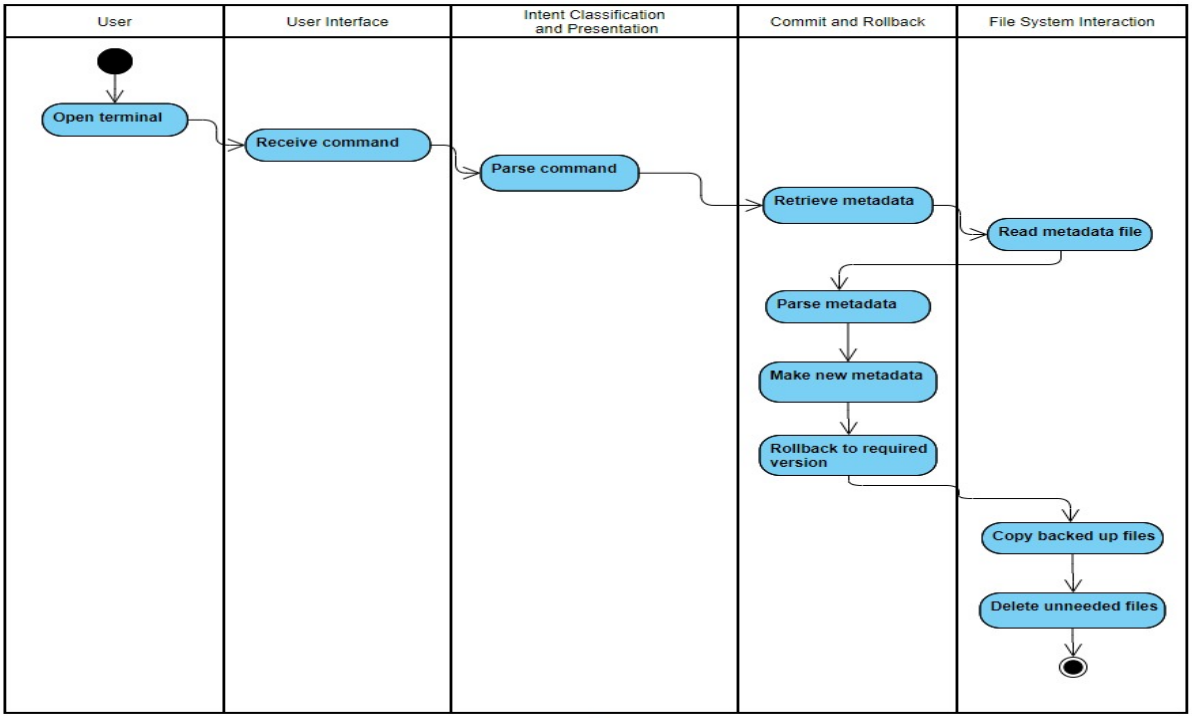


Fig 5.1 Swimlane Activity Network - Rollback

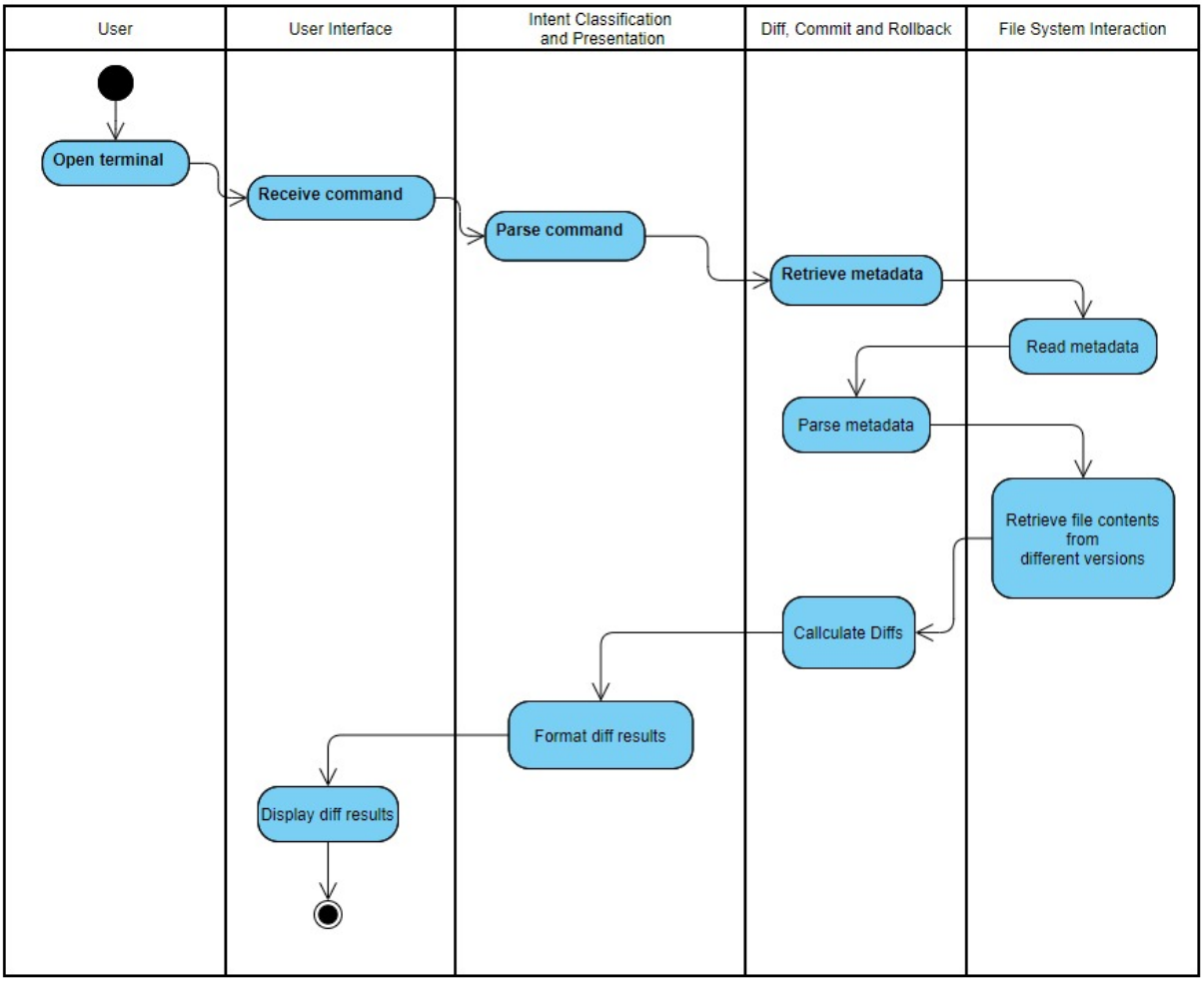


Fig 5.1 Swimlane Activity Network - Commit Diffs

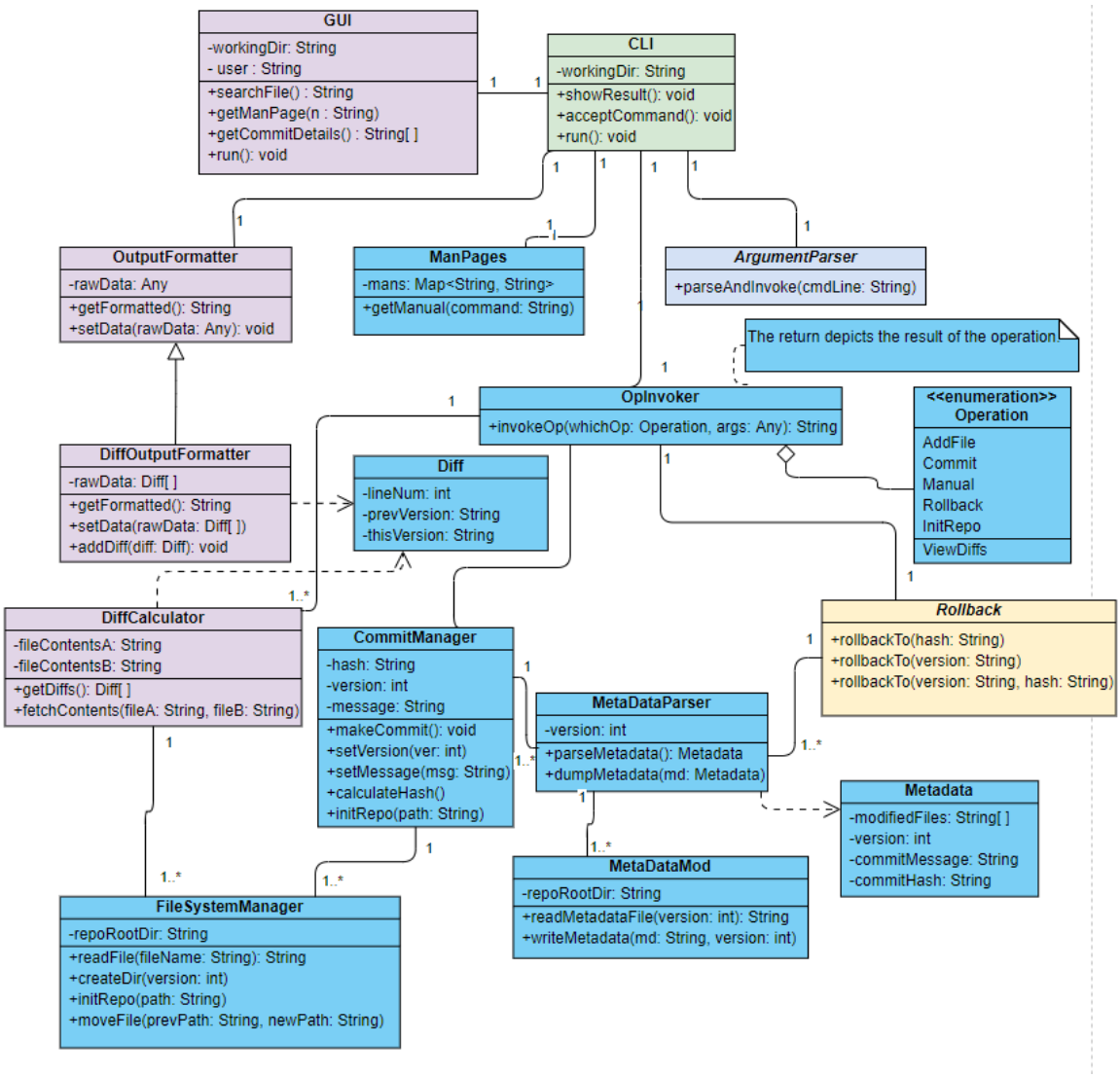


Fig 6 Class Diagram

List of Tables

Table of Definitions, Acronyms, and Abbreviations

Definition, Acronym, or Abbreviation	Description
SRS	Software Requirements Specification.
VCS	Version Control System

Table of References

References	Description
Software Development Plan	The Software Development Plan for our project was referenced.
Software Requirements Specification	The Software Requirements Specification from the Electronic Stamp project was referenced.

Table of Shall Requirements

ID	Stakeholder	Shall Requirement
1	*Developer*	The system shall allow the user to commit changes to files/folders using CLI.
2	*Developer*	The system shall allow the user to commit changes to the contents in files (deletion/appending etc.) using CLI.
3	*Developer*	The system shall allow the user to view commit diffs in an understandable GUI.
4	*Developer*	The system shall allow the user to revert back/rollback changes to contents in files using CLI to the previous commit.
5	*Developer*	The system shall allow the user to revert back/rollback changes resulted by addition/deletion of files and folders, using CLI, to the previous commit.
6	*Developer*	The system should allow the user to initialize the code repository using the CLI.
7	*Developer*	The system shall allow the user to view the manual for the commands of the CLI.
8	*Developer*	The system shall allow the user to specify a commit message in the CLI.

Table of Design Constraints

ID	Origin	*Shall Requirement*
1	Software Project Management Overview	*The system shall be a centralized offline system and not a distributed system.*
2	Software Project Management Overview	*The system shall not support collaboration because of constraint ID 1.*
3	Software Project Management Overview	*The system shall require enough disk storage space.*
4	Software Project Management Overview	*The system shall require the operating system to be either POSIX compliant or have Windows version above Windows XP.*
5	Software Project Management Overview	*The system shall have atleast 512 MB.*

Table of User Characteristics

User	Description
Developer	The developer is any user who develops any kind of software.

Table of Performance Requirements

Performance Requirement	Description
Time taken for operations	The time taken by the VCS to run the operations shall be feasible and optimum.
Software Runtime Errors	The VCS will handle the runtime errors consistently and as gracefully as possible.

Table of Design Constraints

Design Constraint	Description
Storage	To keep various versions of the files, disk storage will be used.

Abbreviations

Table of Definitions, Acronyms, and Abbreviations

Definition, Acronym, or Abbreviation	Description
SDS	Software Design Specification.
SRS	Software Requirements Specification

1. INTRODUCTION

1.1 Objective

1.2 Motivation

1.3 Background

2. PROJECT DESCRIPTION AND GOALS

Commit-Man is a version control system that helps the user keep track of and manage different versions of their files. It features a CLI and a GUI - the complete set of features are available in the CLI, while the GUI features a limited set of features currently (primarily repository browsing functions and viewing diffs across versions).

A repository can have a .gitignore file that specifies for which files changes are to be ignored across versions (when creating new versions). A special folder (.cm) in each repository stores data for version management.

CLI features

- Initializing a repository
- Committing a version and making a new version
- Reverting back to an old version
- Reinitializing in case of corrupted data
- Displaying Logs and Command description

GUI features

- Viewing commit history
- Browsing the repository
- Viewing diffs of files across versions
- Viewing repository info

3 . TECHNICAL SPECIFICATION

Installing

A step by step series of examples that tell you how to get a development env running

Install from PyPi:

```
pip install commit-man
```

Install from Source

```
git clone https://github.com/atharva-Gundawar/commit-man

cd commit-man

python setup.py install
```

Built With

- Python - An interpreted high-level general-purpose programming language.
- Gitignore Parser - A spec-compliant gitignore parser for Python 3.5+
- Docopt - Command-line interface description language

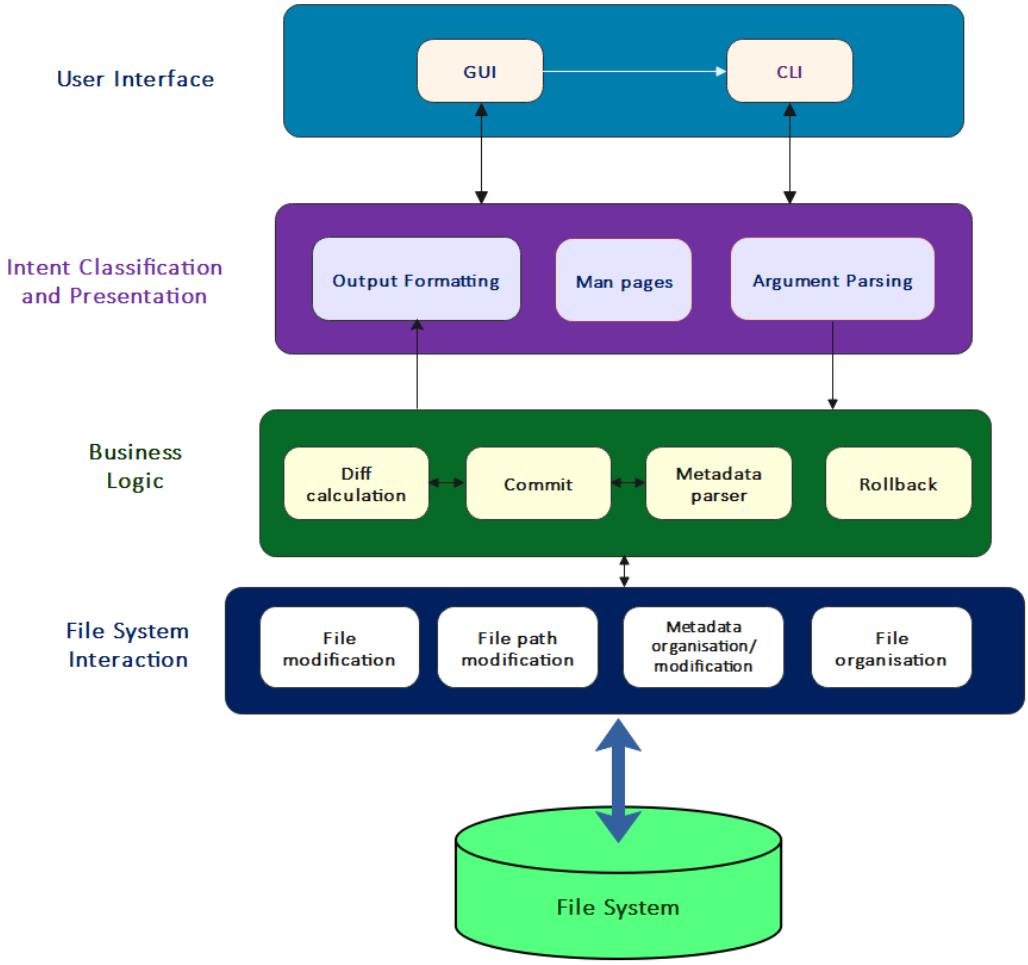
4. DESIGN APPROACH AND DETAILS

1. Design Approach / Materials & Methods

1.1. Architecture and Decomposition Description

1.1.1 Architecture Diagram

Our software shall follow a layered architecture:



The layers are explained below.

1.1.2 Layer Decomposition

The VCS has been decomposed into the following layers:

- User Interface layer: This layer communicates with the user and is in charge displaying the formatted output from the layer underneath it, and for taking raw input from the user. It consists of two modules: one for the CLI and the other for the GUI.
- Intent Classification and Presentation Layer: This layer handles the parsing of the input from the UI layer, and also handles the formatting of the output that comes from the layers below it.
- Business Logic Layer: This layer handles the calculation of diffs, committing files to make new versions and rolling back to previous versions.
- File System Interaction Layer: This layer handles the interaction with the file system on the computer under instructions from the business logic layer.

1.3 Data Decomposition

The following is the data that will be stored with each version to keep track of the version:

Metadata

The metadata shall store the following things for each version:

- Files modified: Depicts which files were modified.
- Version number: Depicts which version this is.
- Commit message: Commit message for this version (unique for each commit).
- Commit hash: A hash generated to identify this commit.

The metadata will be stored along with each version as a particular file in the file system itself.

1.2 Dependency Description

1.2.1 Inter-module Dependencies

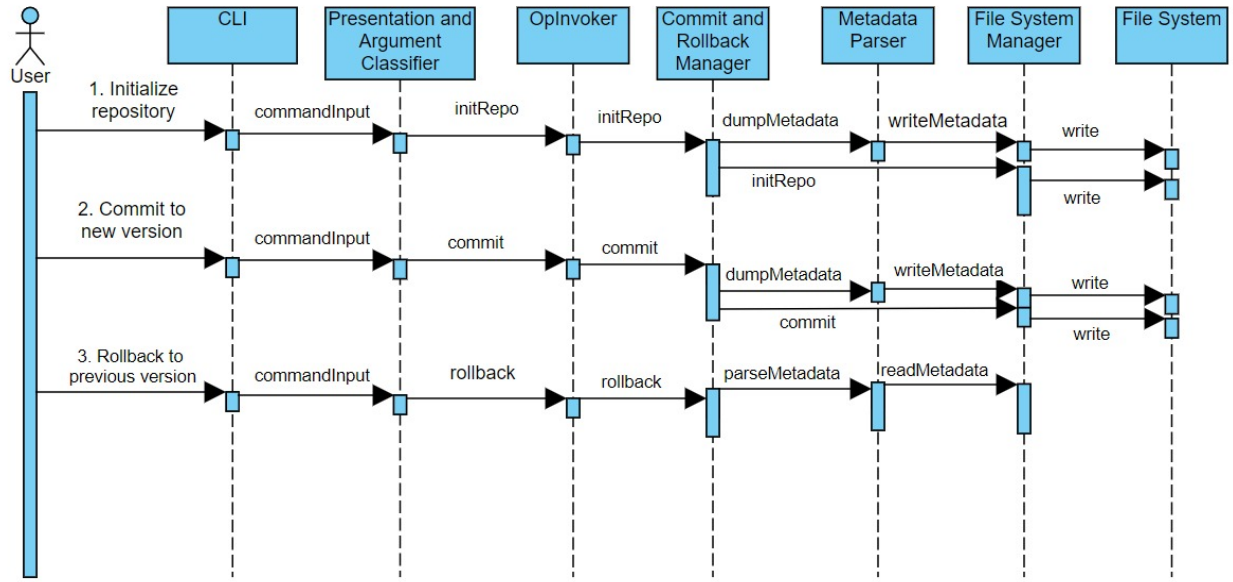
1.2.1.1 Dependent Modules

The layers are interdependent on each other for the successful working of the software. The modules within the layers are also interdependent.

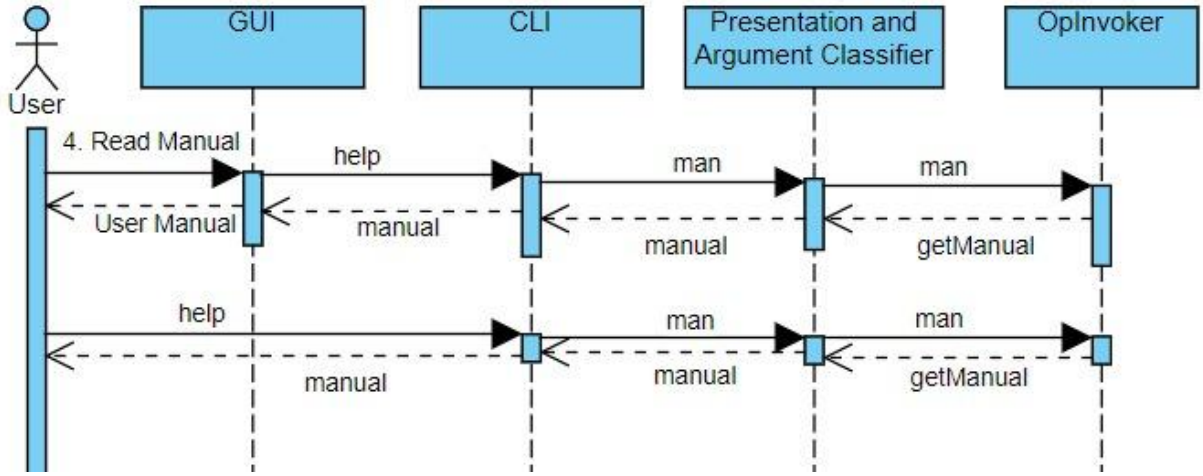
Please refer to the architecture diagram above under section 2.1 and the UML diagram in Appendix A for a clear representation.

1.3 Sequence Diagrams

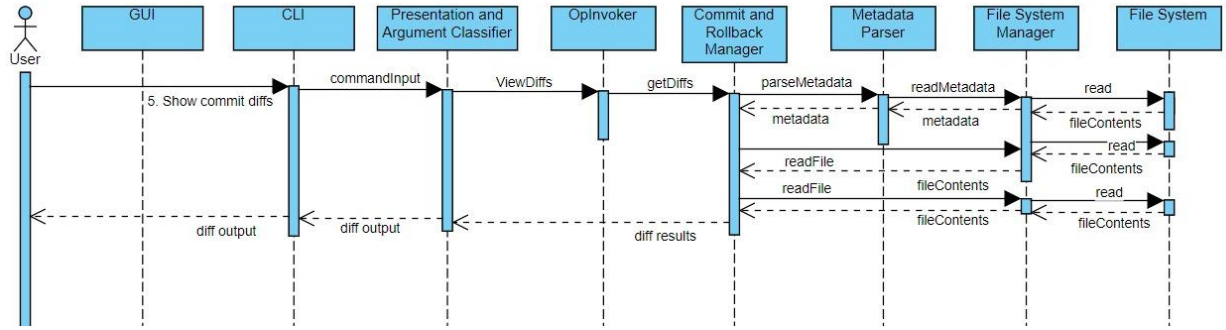
1.3.1 Commit and Rollback

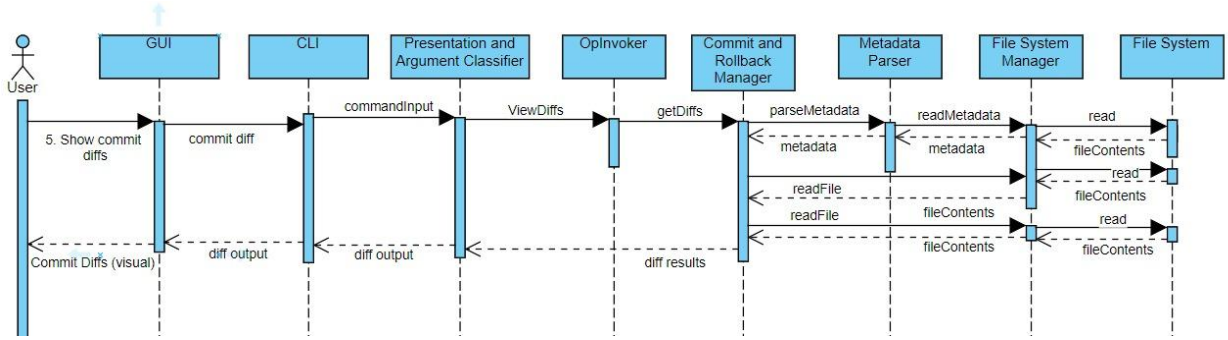


1.3.2 View Manual



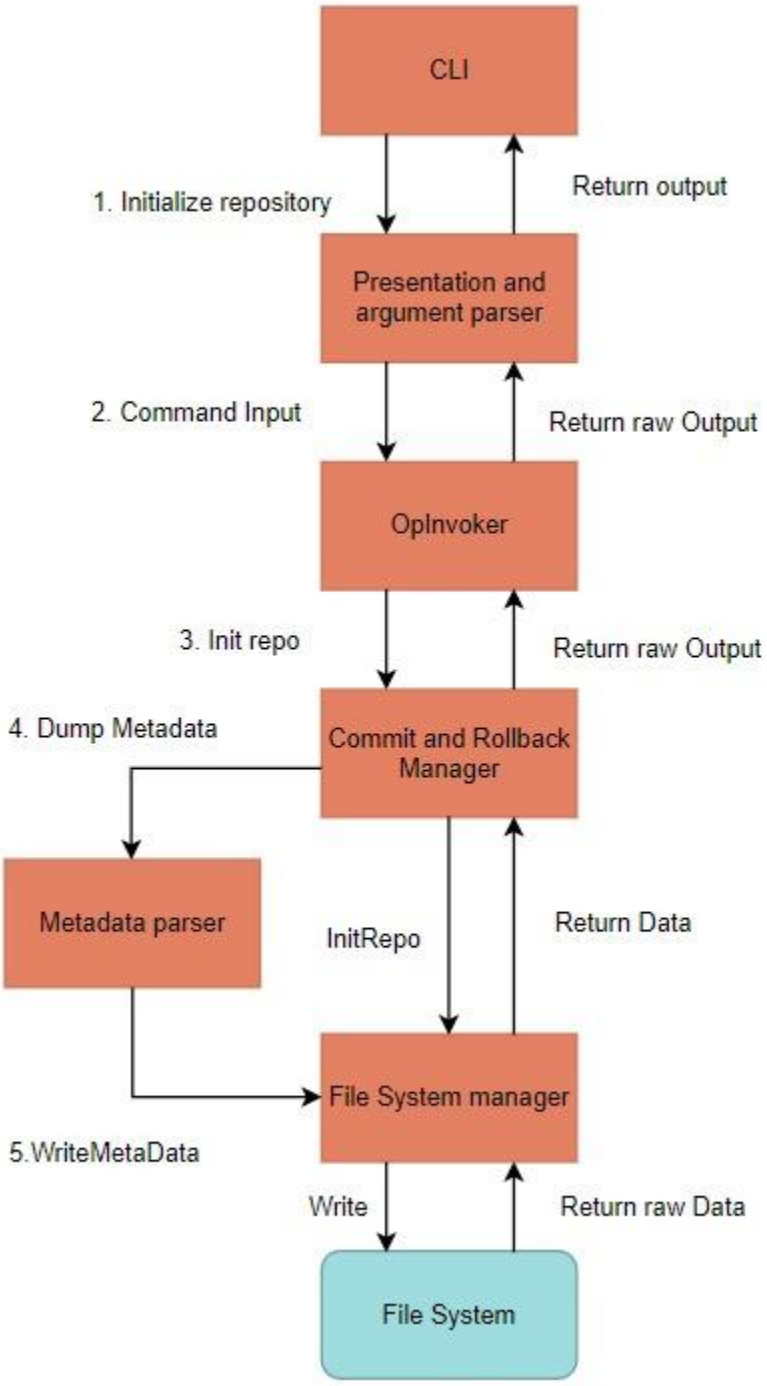
1.3.3 Commit Diffs





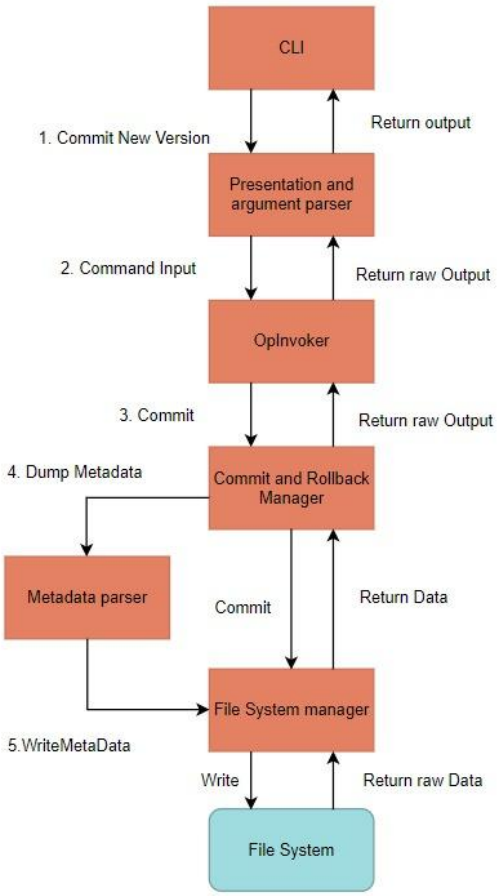
1.4 Collaboration Diagrams

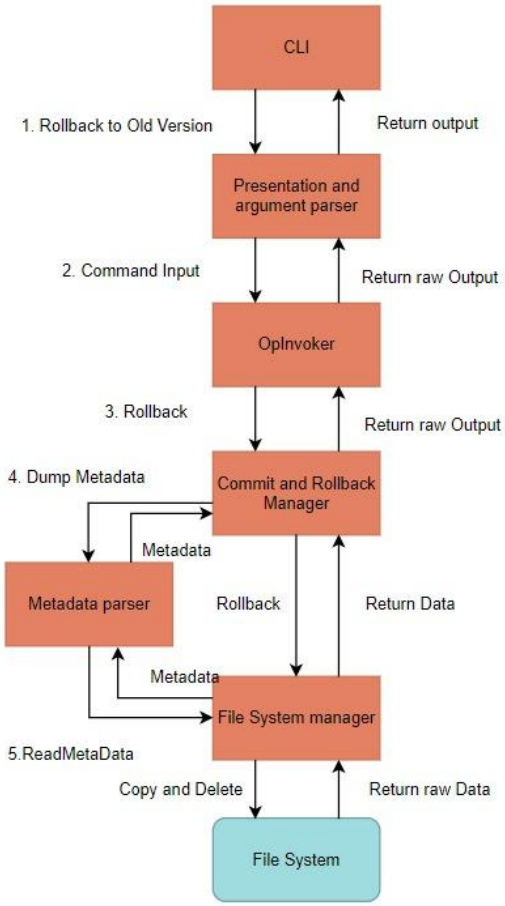
1.4.1 Initialize a new repository, Commit and Rollback:



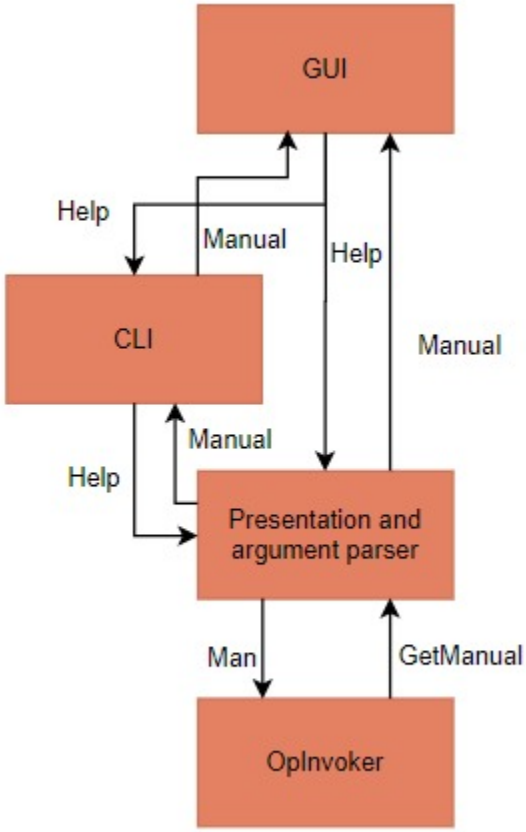
Commit and make a new version:

Rollback to a previous version:

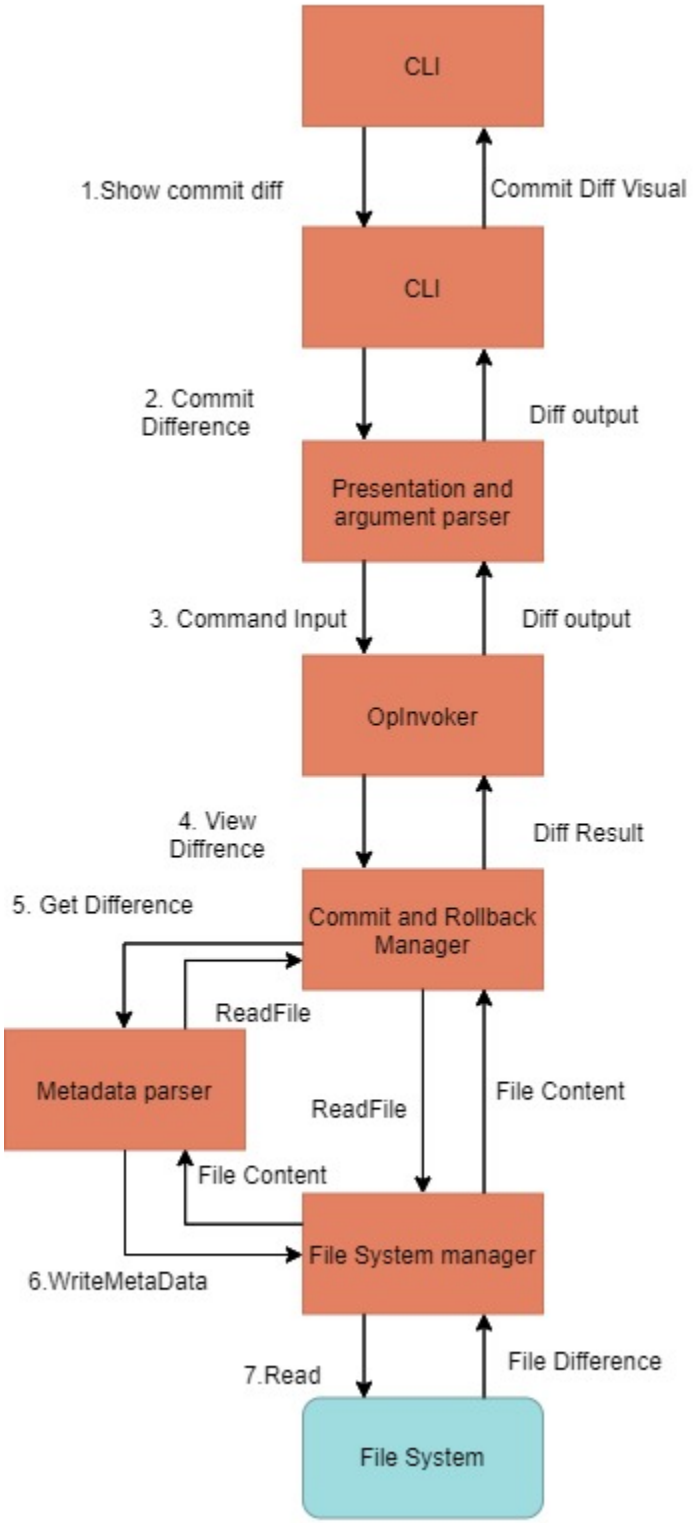

CLI
1. Commit New Version
Return output
Presentation and argument parser
2. Command Input
Return raw Output
OpInvoker
3. Commit
Return raw Output
4. Dump Metadata
Commit and Rollback Manager
Metadata parser
Commit
Return Data
File System manager
5.WriteMetaData
Write
Return raw Data
File System


CLI
1. Rollback to Old Version
Return output
Presentation and argument parser
2. Command Input
Return raw Output
OpInvoker
3. Rollback
Return raw Output
4. Dump Metadata
Commit and Rollback Manager
Metadata
Metadata parser
Rollback
Return Data
Metadata
File System manager
5.ReadMetaData
Copy and Delete
Return raw Data
File System

1.4.2 View Manual

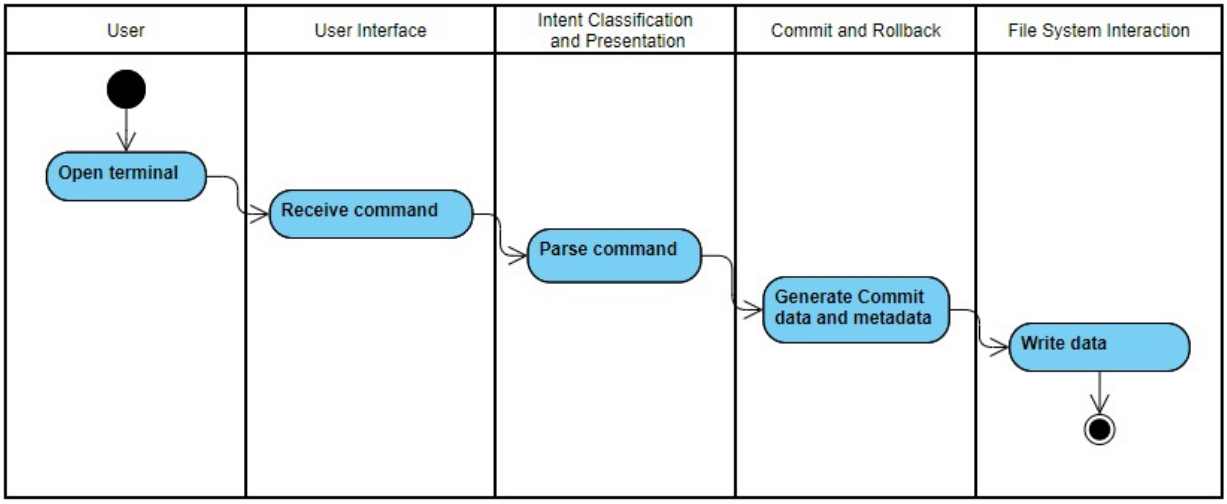


1.4.3 Commit Diffs

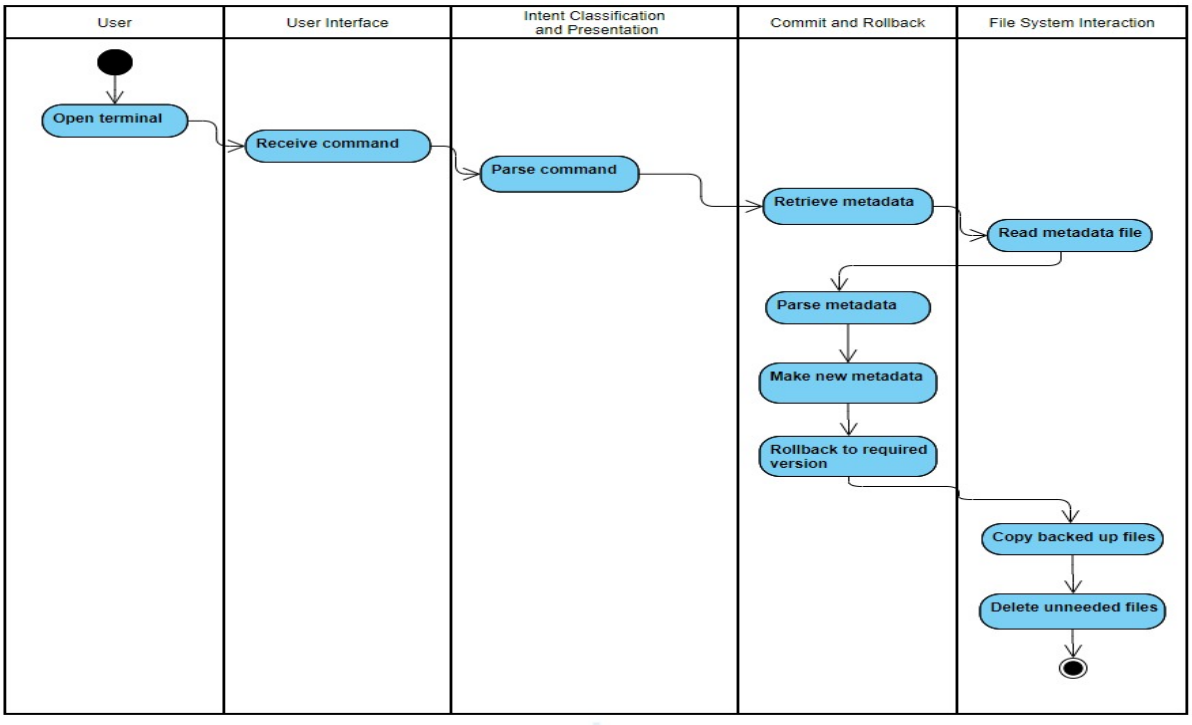


1.5 Swimlane Activity Network

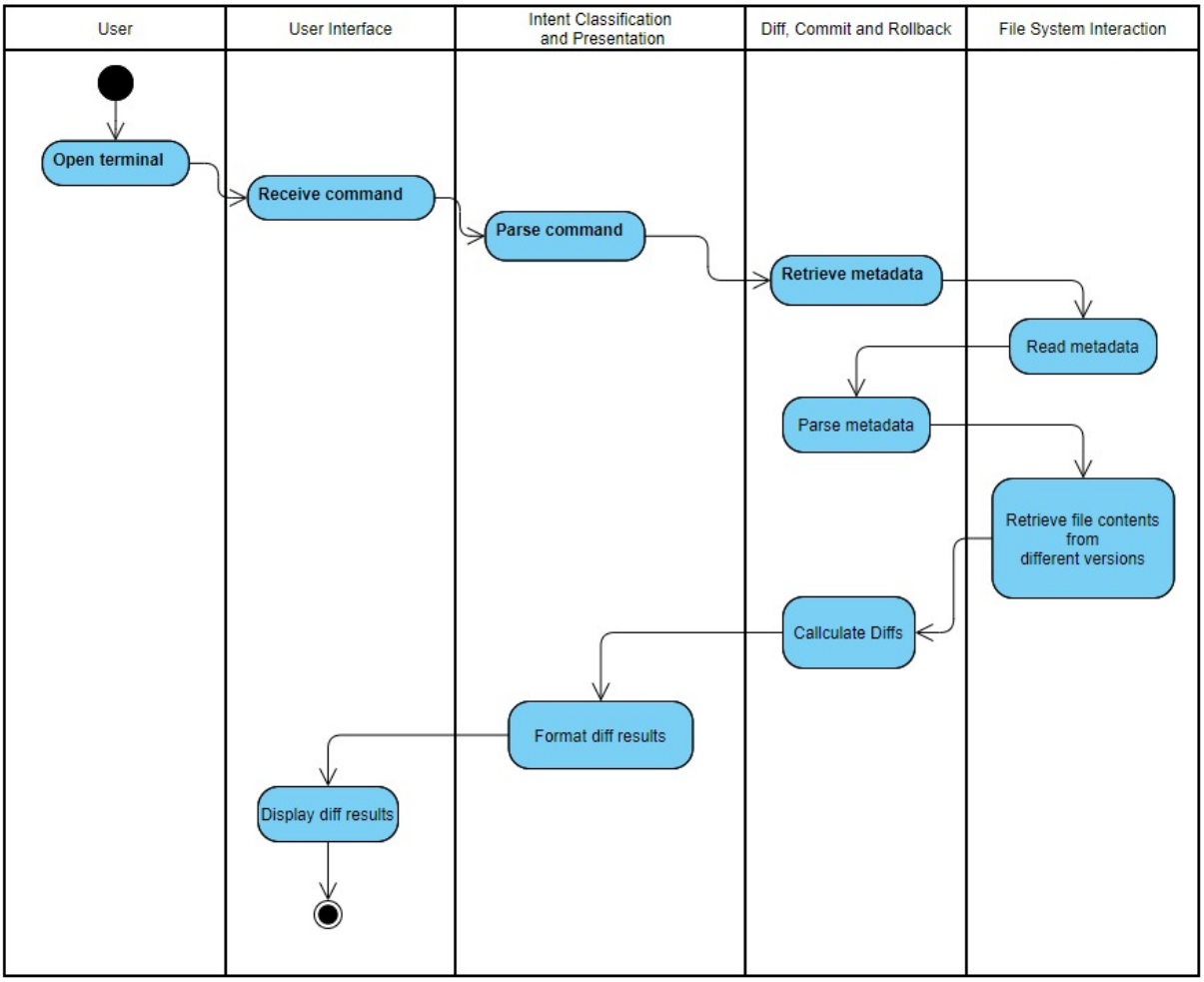
1.5.1 Commit



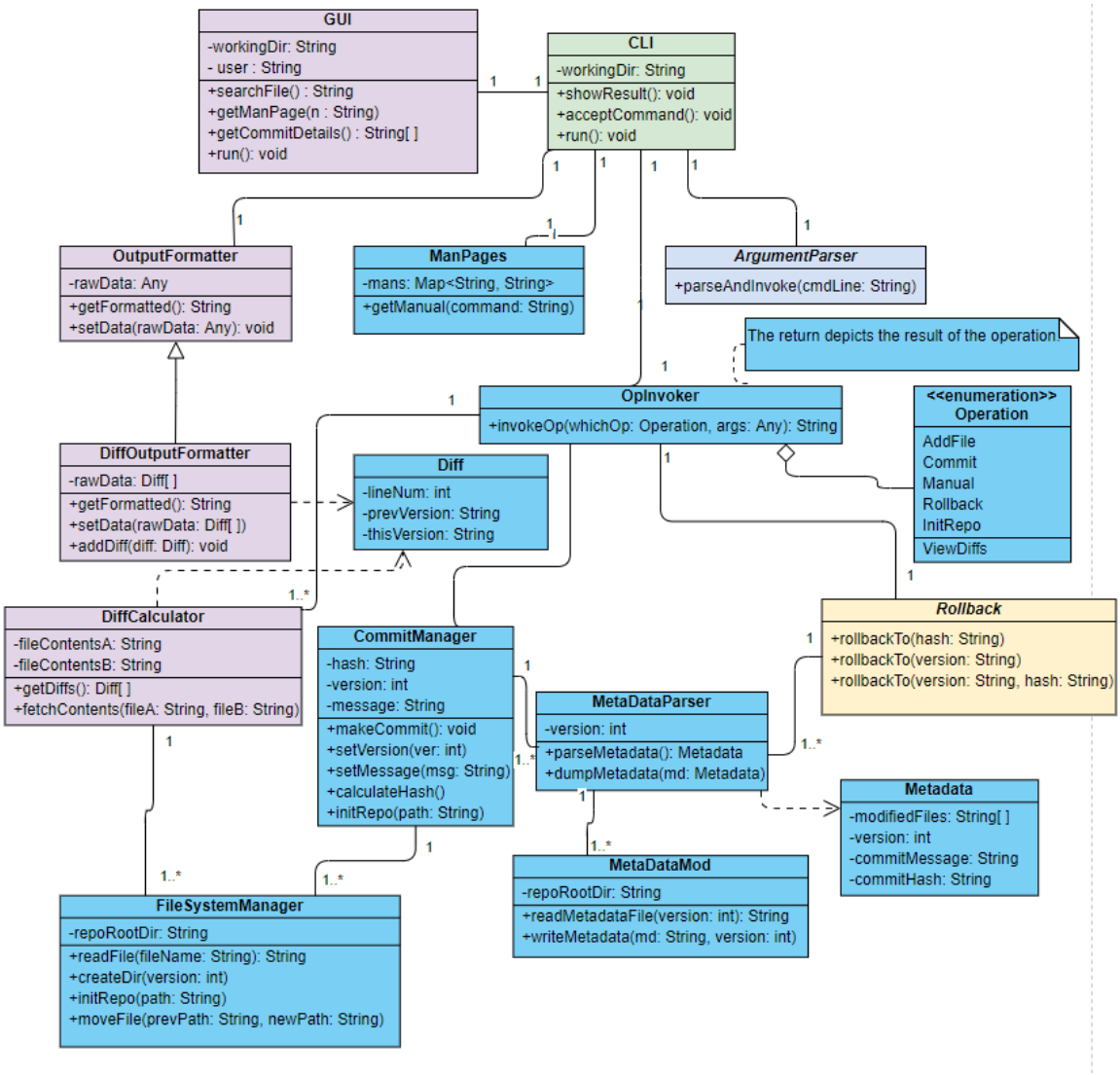
1.5.2 Rollback



1.5.3 Commit Diffs



1.5.4 CommitMan Class Diagram



2. Codes and Standards

A. PEP8

- Style Guide for Python Code
- https://www.python.org/dev/peps/pep-0008/

B. ESLint

- ESLint is a static code analysis tool for identifying problematic patterns found in JavaScript code.
- https://eslint.org/

3. Constraints, Alternatives and Tradeoffs

3.1 WBS - Work Breakdown Structure

Tool used: create.ly

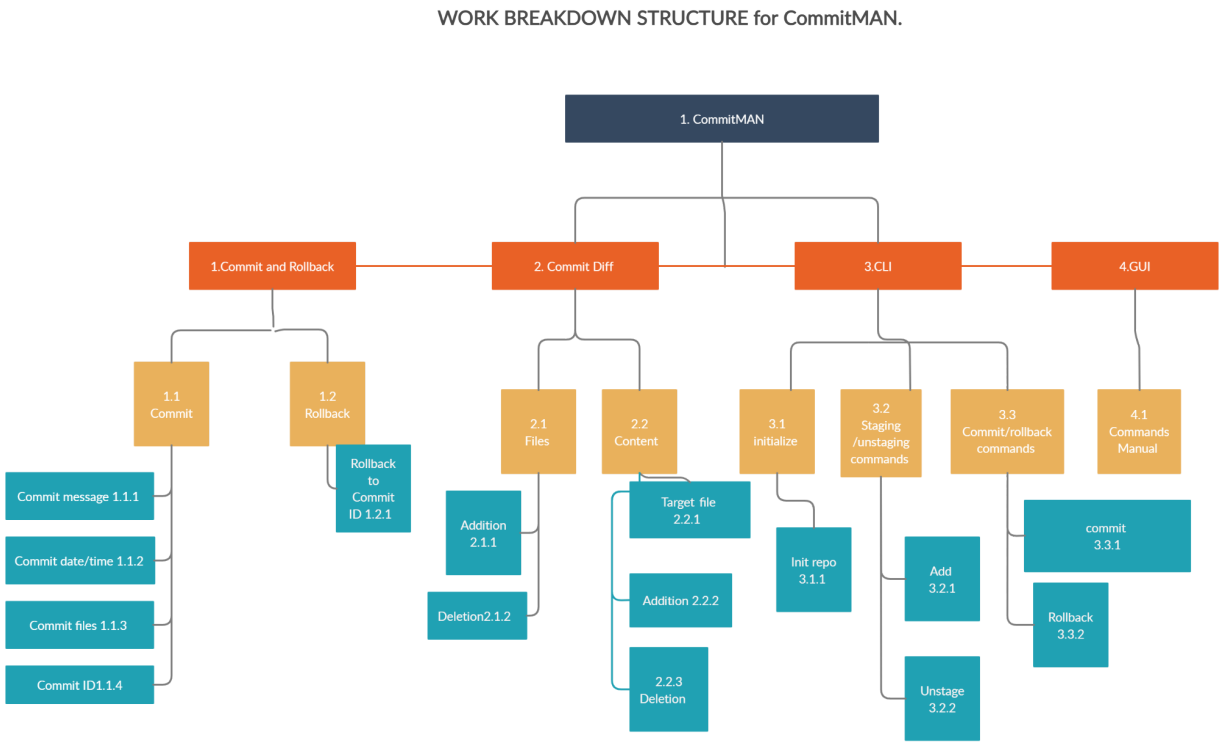


Fig 7 Work Breakdown Structure

3.2 Gantt Chart

Tool used: ClickUp

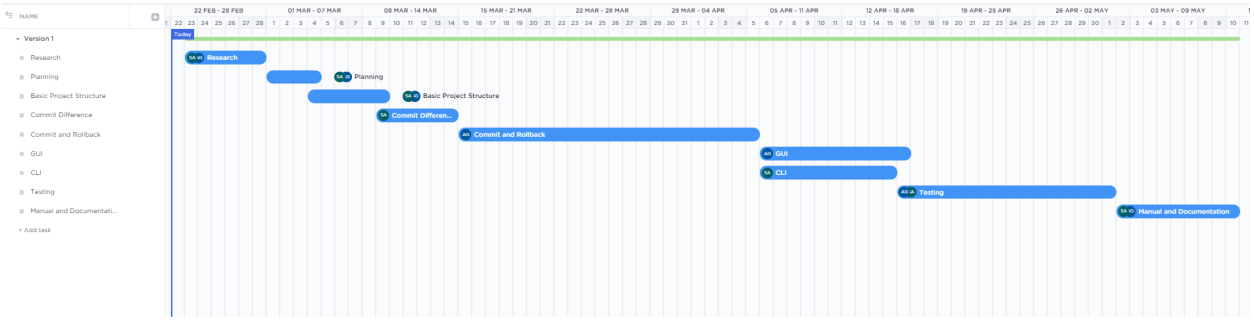


Fig 8 Gantt Chart

3.3 Activity Network Diagram

Made using: Smart Draw

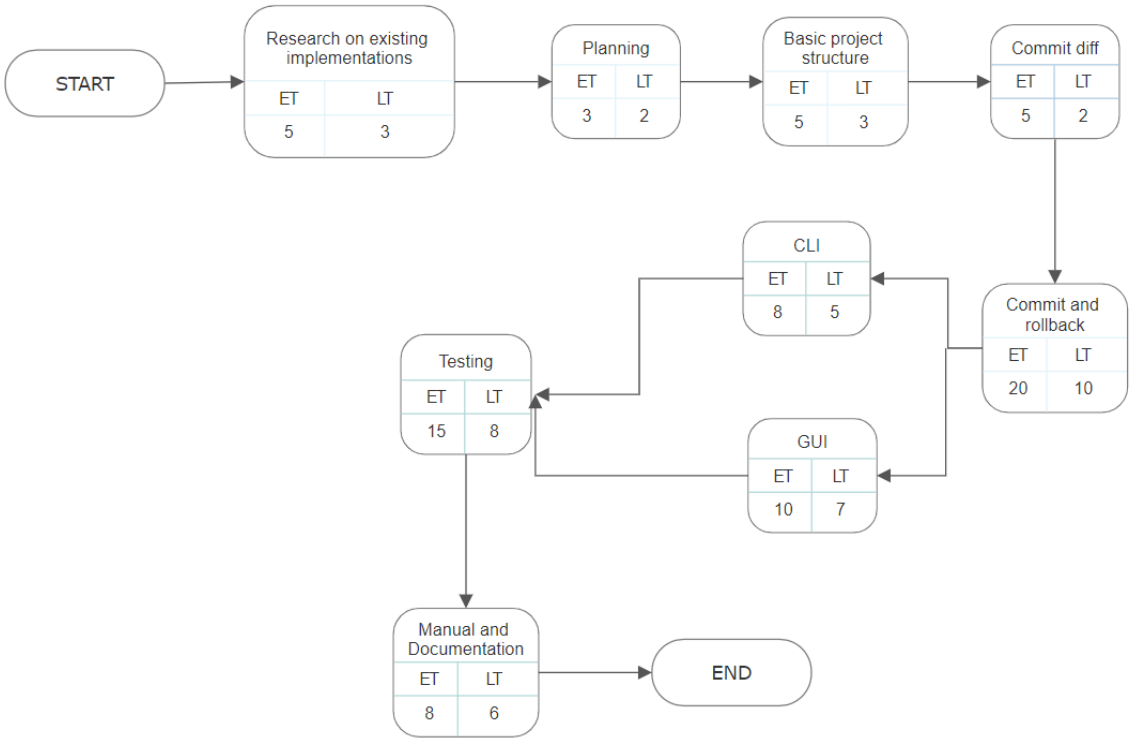


Fig 9 Activity Network Diagram

3.4 Timeline Chart

Tool used: ClickUp

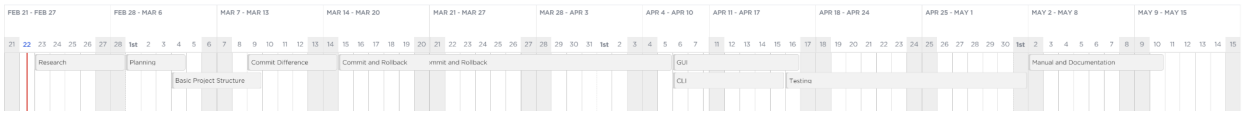


Fig 10 Timeline Chart

6. PROJECT DEMONSTATION

1) CLI:

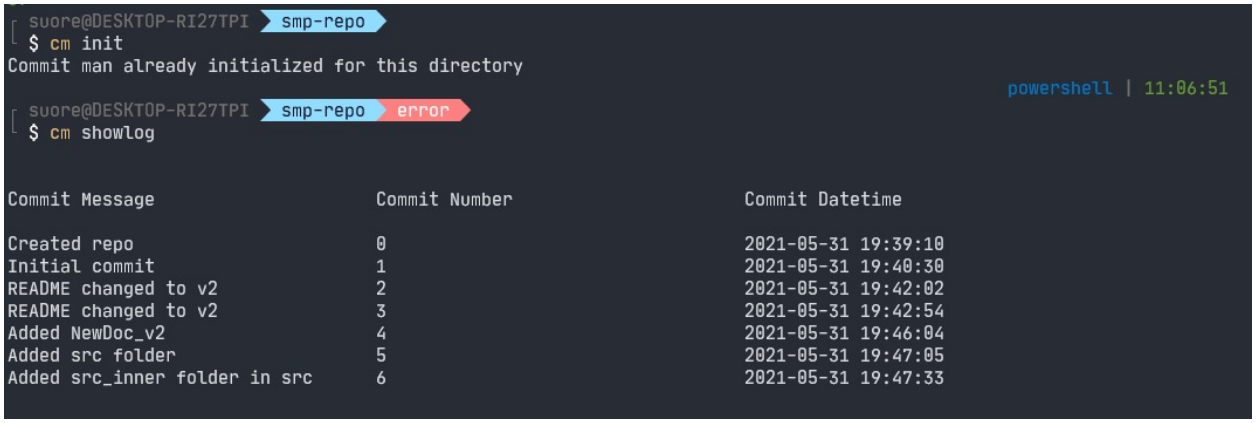


Fig 11.1 init and showlog

```
suore@DESKTOP-RI27TPI  smp-repo
$ cm man


############################################################
------------------------------------------------------------

init:

Initialize Commit man in the current directory

Usage : cm init

This will create a .cm folder in the current directory,
and a log file inside that folder.

------------------------------------------------------------

reinit:

Reinitialize Commit man in the current directory

Usage : cm reinit

This will reinitialize .cm folder in case of
logfile corruption or unavailability.

------------------------------------------------------------

commit:

Commits curent version of working directory

Usage : cm commit <message>

This will create a new commit folder insider the .cm folder.

------------------------------------------------------------

revert:

Reverts to an old version of working directory

Usage : cm revert <Commit_Number> [-f | --force]

This revert to an older version of the project and with the
force option revert will take place even if the latest code
```

Fig 11.2 man pages

```
commit:

Commits curent version of working directory

Usage : cm commit <message>

This will create a new commit folder insider the .cm folder.

-------------------------------------------------------------

revert:

Reverts to an old version of working directory

Usage : cm revert <Commit_Number> [-f | --force]

This revert to an older version of the project and with the
force option revert will take place even if the latest code
has not been commited.

-------------------------------------------------------------

showlog:

Displays Log file in a tabular format on the Terminal

Usage : cm showlog

Queries data from the log file and adds headers and spacing,
then displays them to the terminal.


-------------------------------------------------------------
#############################################################
```

Fig 11.3 man continued

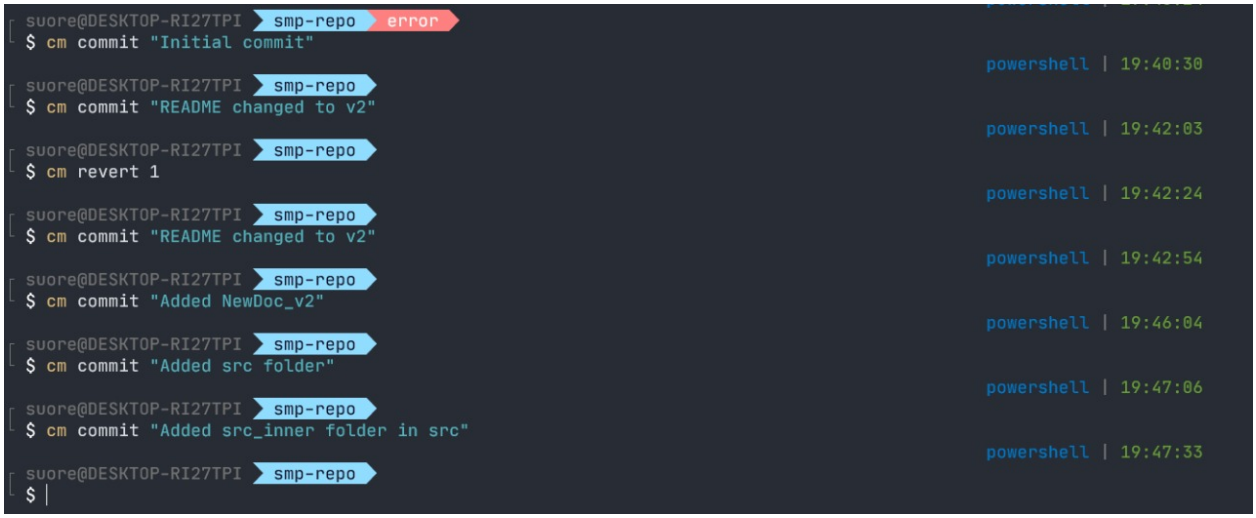


Fig 11.4 commit and revert

2) GUI :

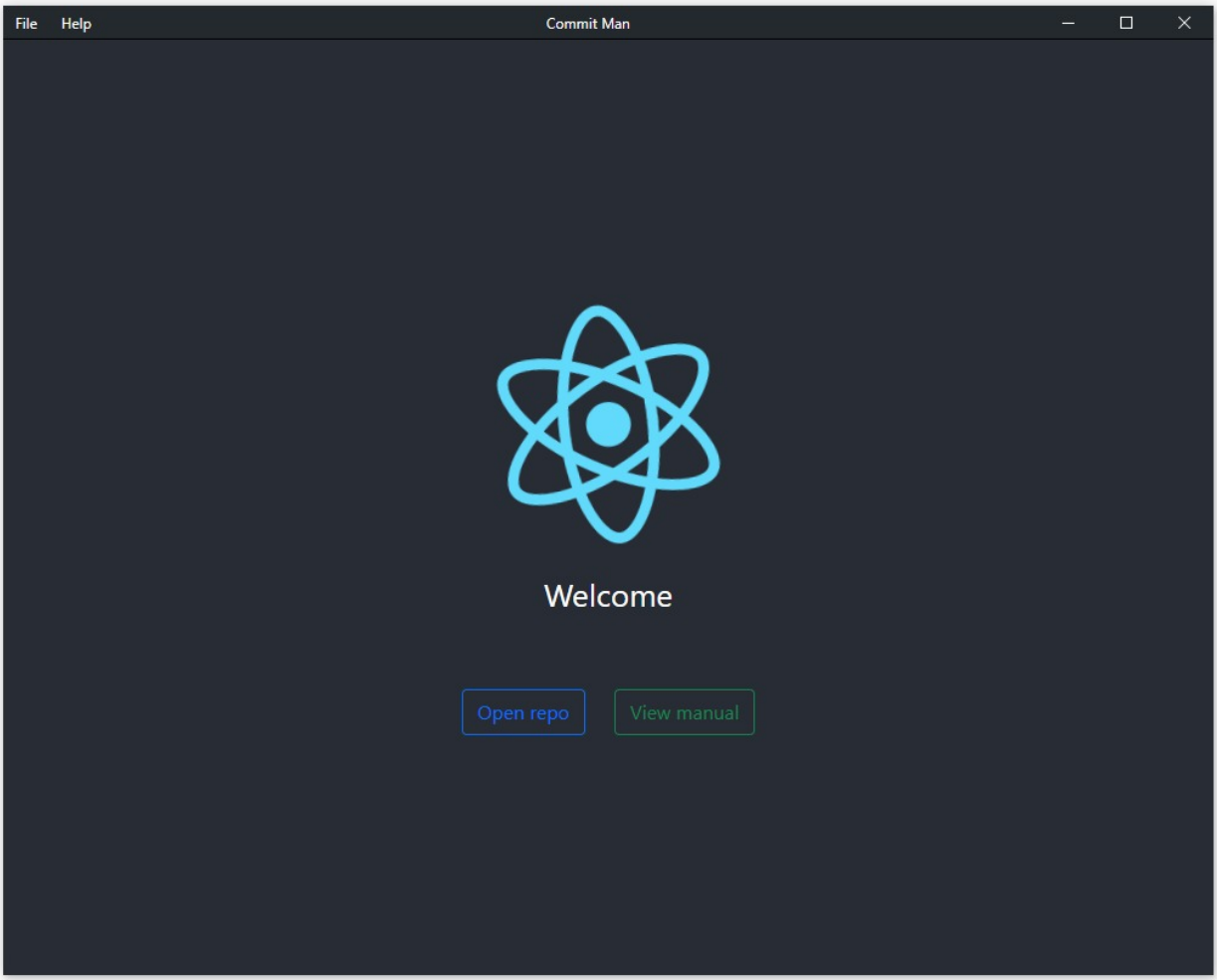


Fig 12.1 welcome screen

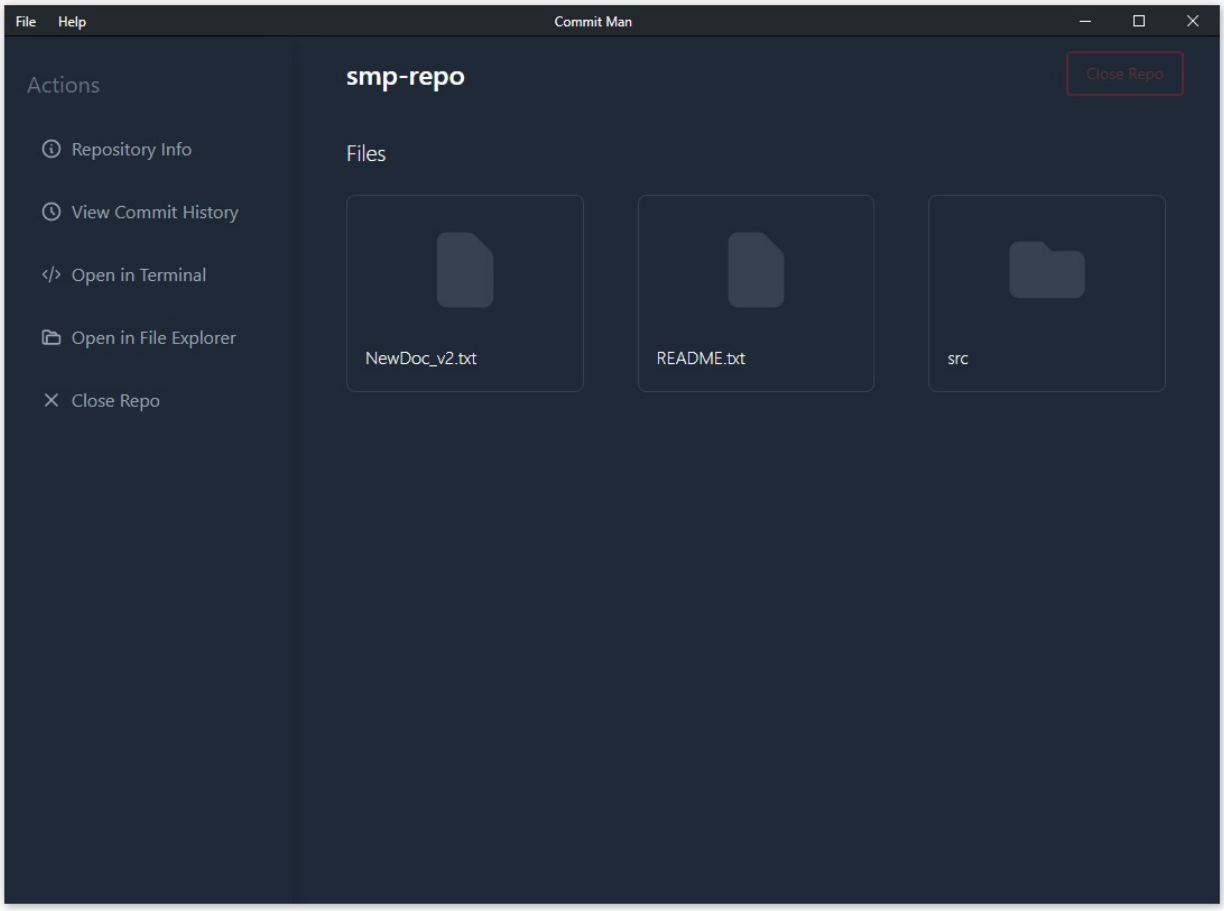


Fig 12.2 Displaying Repository

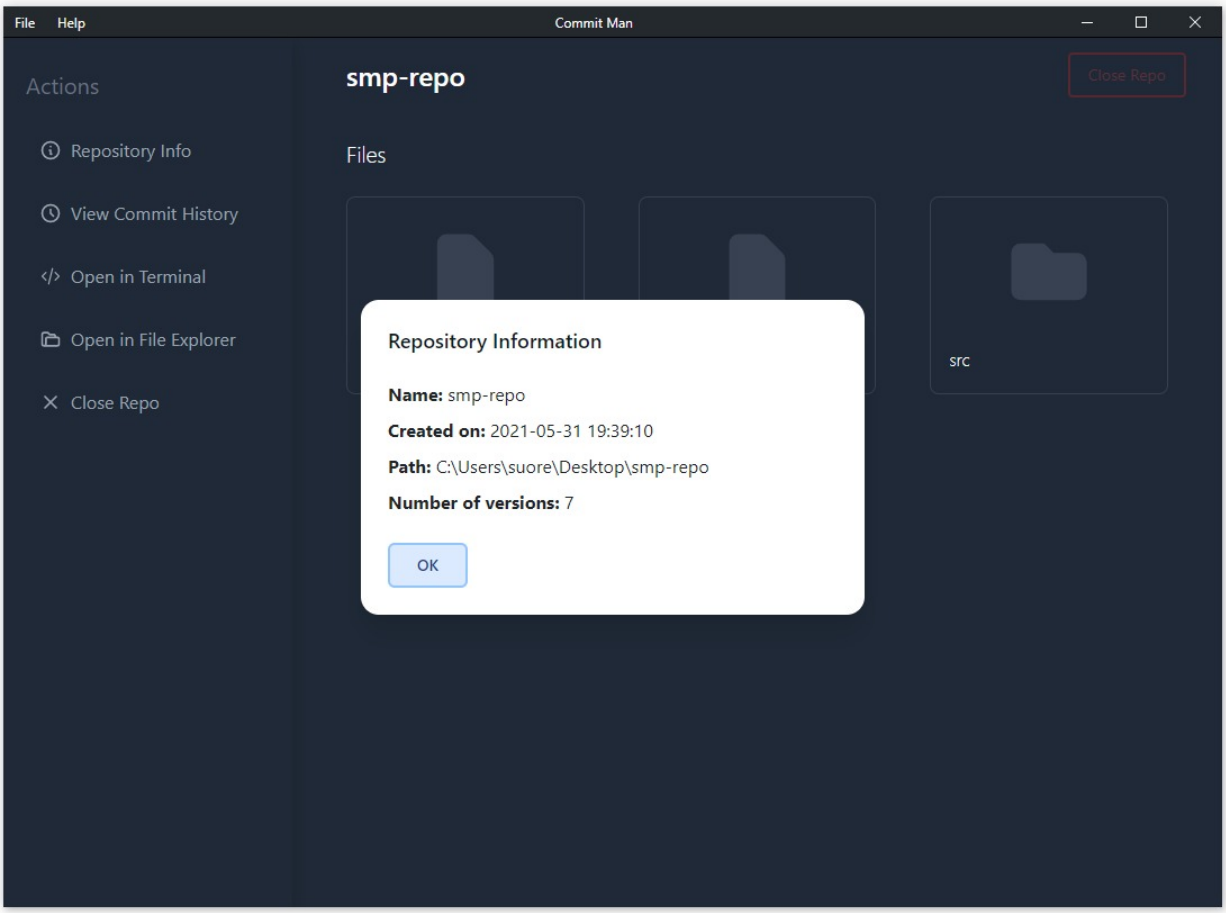


Fig 12.3 Repository information

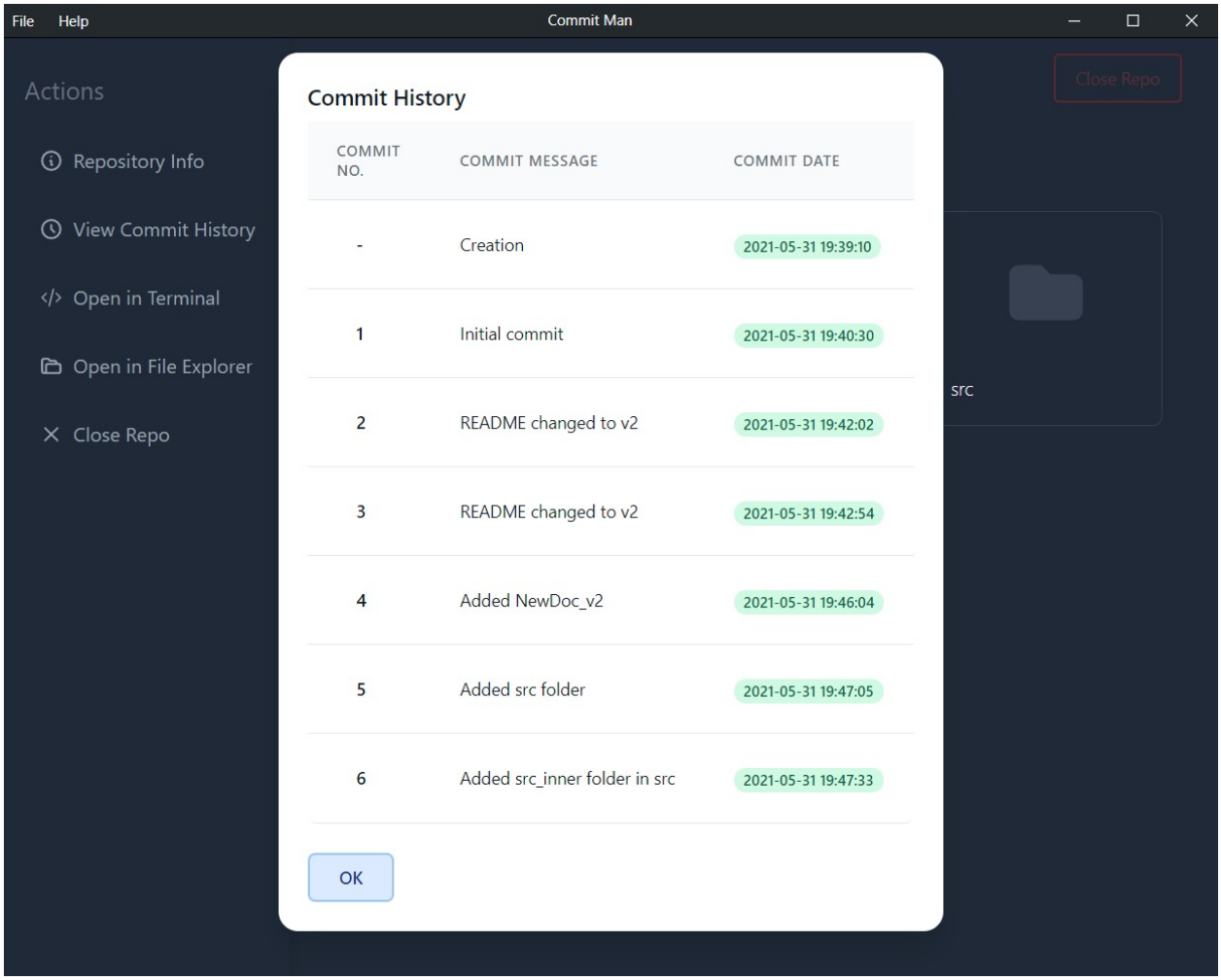


Fig 12.4 Commit history

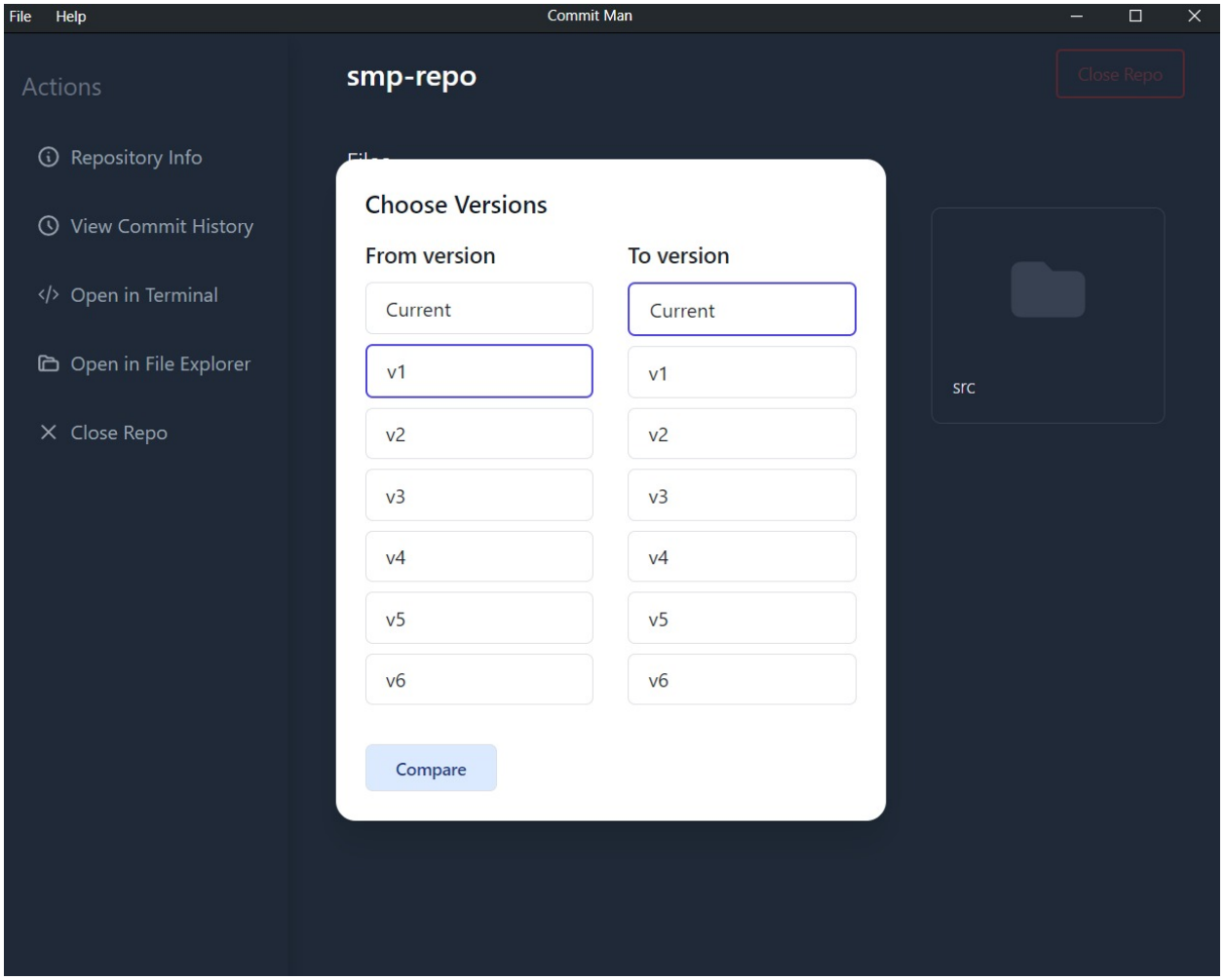


Fig 12.5 Choosing file versions for viewing their differences

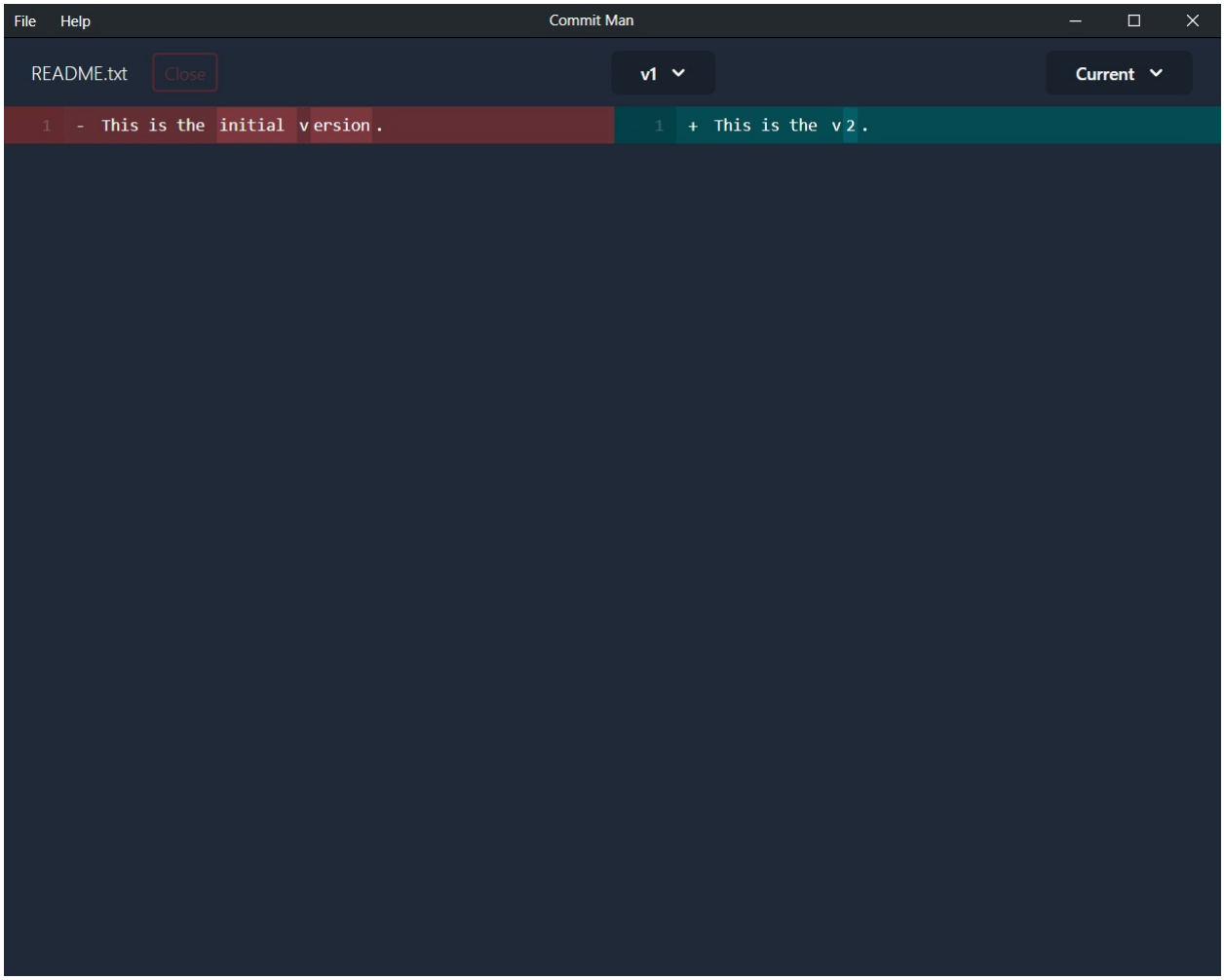


Fig 12.6 File differences

7. COST ANALYSIS/ RESULT & DISCUSSION

Following is our Tech stack :

CLI :

1) Python
2) Docopt
3) Gitignore Parser
4) PyPi

GUI :

1) React js
2) Electron js
3) Tailwind css
4) Node js

Case tools :

1) Visual studio code
2) Pencil tool
3) Click up
4) Notion
5) Star UML

All the above are open source or free softwares/modules, so our total financial cost came to $0.