<!-- File: GitTimeTravelGUI_UserGuide.md Path: Documentation/Tools/GitTimeTravelGUI_UserGuide.md Standard: AIDEV-PascalCase-2.0 Created: 2025-07-10 Last Modified: 2025-07-10 07:46PM Description: Comprehensive user guide for the Git Time Travel GUI application Framework: Markdown documentation standard -->

Git Time Travel GUI - User Guide

Overview

The Git Time Travel GUI is a powerful tool for safely exploring Git repository history through an intuitive graphical interface. This application allows you to travel back to any previous commit, explore the codebase at that point in time, and safely return to your current working state.

Key Features

The application provides comprehensive time travel capabilities including safe commit exploration with automatic stashing of uncommitted changes, temporary branch creation for risk-free exploration, visual diff display between commits and current state, one-click return to original branch with automatic change restoration, GitHub integration for viewing commits online, and comprehensive error handling with detailed user feedback.

System Requirements

Prerequisites

Your system must have Python 3.8 or higher installed, Git 2.20 or higher available in system PATH, and PySide6 library for the graphical interface. The application also requires a valid Git repository as the working directory and sufficient system memory for diff operations on large repositories.

Installation Steps

First, ensure you have Python and Git installed on your system. Install the required Python dependencies by running `pip install PySide6` in your terminal. Download the `GitTimeTravelGUI.py` file to your preferred location. Optionally, for enhanced timestamp accuracy, install the git-restore-mtime utility with `pip install git-restore-mtime`.

Getting Started

Launching the Application

Navigate to your Git repository directory in the terminal and run the application with `python GitTimeTravelGUI.py`. The application will automatically detect your repository and validate the Git

environment. If successful, you will see the main interface with your recent commit history loaded.

Interface Overview

The main window consists of several key components. The status bar at the top displays current branch information and operation status. The GitHub repository link button opens your repository in a web browser when clicked. The commit list shows the last 30 commits with timestamps and descriptions, where the topmost commit is marked as "current" with a crown emoji. The difference viewer displays changes between the selected commit and your current branch. The action buttons include Travel to Selected Commit for initiating time travel, Return to Original Branch for ending time travel safely, and Exit for closing the application.

Using Time Travel Features

Exploring Commit History

Browse through the commit list to see your repository's recent history. Click on any commit to view the differences between that commit and your current branch in the difference viewer. The differences show what changes would be "undone" if you traveled to that commit. Commits are displayed with short hashes, dates, and commit messages for easy identification.

Traveling to a Commit

Select any commit from the list except the current one and click the "Travel to Selected Commit" button. The application will present you with travel mode options: creating a temporary branch (recommended for most users) or using detached HEAD mode (for advanced users who understand Git internals).

Before traveling, the application automatically stashes any uncommitted changes in your working directory to preserve your work. After successful time travel, you can explore the codebase as it existed at that point in time, run tests, examine files, or perform any read-only operations you need.

Returning to Present

When you're finished exploring, click the "Return to Original Branch" button to safely return to your original state. The application will switch back to your original branch and automatically restore any previously stashed changes. Your working directory will be restored to exactly the state it was in before time travel began.

Safety Features

Automatic Change Preservation

The application automatically detects and stashes any uncommitted changes before time travel operations. This ensures that your work in progress is never lost during exploration. Upon return, these

changes are automatically restored to your working directory.

State Validation

Before any Git operation, the application validates repository state, checks for Git availability and proper repository structure, and verifies commit integrity before attempting travel. All operations include comprehensive error checking with detailed user feedback.

Emergency Recovery

If any operation fails, the application attempts automatic recovery to restore your original state. In extreme cases, manual recovery instructions are provided with specific Git commands to run. The application never leaves you in an unknown or dangerous state.

Advanced Features

Temporary Branch Creation

When traveling with branch creation mode, the application generates unique branch names using timestamps in the format `travel_YYYYMMDD_HHMMSS`. These branches can be kept for future reference or deleted after exploration. The application automatically manages branch switching and cleanup.

Timestamp Restoration

If you have `git-restore-mtime` installed, the application will restore file modification timestamps to match their last commit dates. This feature is useful for forensic analysis or when timestamp accuracy is important for your workflow.

GitHub Integration

The application automatically detects your repository name and provides direct links to view commits on GitHub. This feature helps correlate local exploration with online repository browsing and facilitates sharing specific commits with team members.

Troubleshooting

Common Issues and Solutions

If the application reports "Not a Git repository," ensure you're running the application from within a Git repository directory and verify that the `.git` folder exists in your current directory.

For "Git not found" errors, confirm Git is installed on your system and available in your system PATH. Test by running `git --version` in your terminal.

When encountering "No commits found," verify your repository has at least one commit by running `git log --oneline` manually.

For timeout errors during operations, this may indicate a very large repository or system performance issues. Consider closing other applications or trying again when system load is lower.

If stash operations fail, you may have conflicts or unusual file states. Try manually running `git status` to check for issues and `git stash` to test stashing capability.

Permission Issues

On some systems, you may encounter permission errors when accessing Git operations. Ensure you have proper read/write permissions to the repository directory and Git configuration is properly set up for your user account.

Performance Considerations

For very large repositories, some operations may take longer than usual. The application includes timeout protections, but you may need to be patient with large diff operations. Consider using the tool on smaller repositories or specific branches for better performance.

Best Practices

Safe Exploration Guidelines

Always ensure important work is committed before using time travel features, even though the application automatically stashes changes. Use branch creation mode unless you specifically need detached HEAD for advanced Git operations. Explore freely but avoid making changes during time travel sessions. Return to your original branch promptly after exploration to avoid confusion.

Workflow Integration

The time travel tool works excellently for code archaeology when investigating when specific changes were introduced, understanding the evolution of complex features, debugging issues by examining working versions from the past, and preparing for code reviews by understanding change context.

Team Usage

When sharing findings with team members, use the GitHub integration to reference specific commits and copy commit hashes from the application for easy sharing. Document your findings while exploring to maximize the value of your investigation time.

Keyboard Shortcuts and Tips

Navigation Tips

Use the scroll wheel or arrow keys to navigate through the commit list quickly. Double-click commits to automatically start the time travel process. The difference viewer supports standard text selection and copying for sharing findings.

Efficiency Recommendations

Keep the application open while working on features to quickly reference historical implementations. Use the tool during code reviews to understand the context of changes being reviewed. Bookmark significant commits by noting their hashes for future reference.

Security Considerations

Safe Operations

The application only performs read operations during time travel exploration and never modifies historical commits or repository structure. All changes are isolated to branch switching and stashing operations. Your original work is always preserved and recoverable.

Data Protection

Uncommitted changes are safely stashed with descriptive messages including timestamps. The application validates all operations before execution to prevent data loss. Emergency recovery procedures ensure you can always return to a safe state.

Support and Maintenance

Getting Help

If you encounter issues not covered in this guide, check Git status manually with `git status` to understand repository state and verify all prerequisites are met according to the system requirements section.

For persistent issues, consult Git documentation for underlying Git operations or seek assistance from experienced Git users in your organization.

Updates and Improvements

The application follows Project Himalaya design standards and is designed for maintainability and extension. Future versions may include additional features like commit search capabilities, extended history viewing, and integration with other Project Himalaya tools.

Conclusion

The Git Time Travel GUI provides a safe, intuitive way to explore repository history without the complexity and risks of manual Git operations. By following this guide and the safety recommendations, you can confidently investigate your codebase's evolution and gain valuable insights into your project's development history.

Remember that time travel is a powerful tool for understanding and debugging, but should complement, not replace, proper Git workflow practices like meaningful commit messages, regular commits, and proper branch management.