# 🕰️ Git Time Machine GUI – User Guide (Himalaya Edition)

Welcome to the **Git Time Machine** – your safe, powerful, visual gateway into the history of any Git project. Whether you’re reviewing changes, resurrecting old code, or just exploring, this tool helps you **time travel without blowing things up**.

## 🚀 What It Does

* **Lists recent commits** in your current Git repository
* Shows a **description and timestamp** for each
* Lets you **view diffs** between a selected commit and your current state
* Allows safe checkout:
  + In **Detached HEAD mode** (read-only snapshot)
  + Or create a **temporary branch** to hack in that state
* Automatically **stashes uncommitted changes**
* Provides an **Undo** button to return to your original branch
* Includes **GitHub integration**: click to view repo or commit online

## 🧭 How to Use It

### 1. Launch the GUI

python3 git\_time\_travel\_gui.py

Run this from any directory that is a valid Git repo.

### 2. Understand the Interface

* **Top Label**: Shows your current branch (e.g., main)
* **Commit List**: Recent 30 commits with short description and date
* **Diff Viewer**: Shows what’s changed between selected commit and current state
* **Buttons**:
  + 📎 View GitHub: Opens the repo in browser
  + 🧭 Travel to Selected Commit: Checkout that state (safely!)
  + ↩ Return to Original Branch: Revert to your pre-travel state

## ✨ Time Traveling (Safely)

1. Click a commit from the list
2. See the changes in the diff viewer (optional, but recommended!)
3. Click **Travel**
4. Choose:
   * **Yes** → Create a new temporary branch
   * **No** → Detach HEAD (read-only mode)
5. Use **Return to Original Branch** to come back

✅ If you had uncommitted changes, they’re automatically stashed and will be restored when you return.

## 🧠 Best Practices

| ✅ Do This | 🚫 Avoid This |
| --- | --- |
| Travel in repos with clean state | Forcing resets or deletions manually |
| Use temp branches to try fixes | Working too long in detached mode |
| Use Return button – it’s safe! | Forgetting to return and merging accidentally |
| Review diffs before traveling | Assuming no changes were made |
| Back up your repo for big jumps | Overwriting remote branches |

## ⚙️ Advanced Tips

* After each time-travel event, the tool now automatically runs git restore-mtime to restore filesystem timestamps based on commit history — critical for audit compliance.
* You can also manually run git restore-mtime if needed, or integrate it into post-checkout workflows.
* Files and logs generated by restore scripts should be ignored using .gitignore to avoid triggering audit systems or CI tools unnecessarily.
* You can edit or create .himalaya.json to define repo metadata (e.g. override GitHub username)
* Works great as part of Himalaya tool suite – consider creating .desktop launcher
* Fully portable – drop it into any project under /tools and go
* Keyboard-friendly: select, diff, travel, return – no terminal fiddling

## 🧹 What Happens Under the Hood?

Here’s how the Time Machine works behind the curtain:

* **Detached HEAD** is like time-traveling to the past without changing history. You’re not on a branch, so you can’t commit (unless you create a new branch).
* **Temporary branches** are safe work zones. They preserve history and allow experimentation.
* **GitHub stays clean**: You’re not pushing anything unless you intentionally do so. Local-only by default.
* **No permanent damage**: The Time Machine doesn’t alter your repo unless you explicitly merge, commit, or push changes.
* **Undo is always an option**: You’re just a git switch away from safety.

## 📂 Best Practice: Ignore Timestamp Artifacts

To avoid audit flags or rebuilds, add any output from git restore-mtime or Himalaya audit tools to your .gitignore:

# Timestamp restoration and audit metadata  
.timestamp\_restore\_log  
.restore-audit.md

You’ve already excluded .venv, but this helps reinforce safe practice.

## 🧨 What *Could* Go Wrong

* ❗ **If you push a temp branch accidentally** → Not harmful, but clutters GitHub. Just delete the branch there.
* ❗ **Working in detached mode and committing** → Easy to lose work unless you branch from that state.
* ❗ **Returning without popping stash** → You might forget about saved changes. Use git stash list to find them.

## 🆘 Backup Before You Time Travel

Before you start experimenting, it’s smart to clone a backup of your current repo. Run this in the VS Code terminal or any shell inside the repo:

REPO\_NAME=$(basename "$PWD")  
rsync -av --exclude=".venv" ./ ../${REPO\_NAME}\_BACKUP  
```bash  
REPO\_NAME=$(basename "$PWD")  
git clone . ../${REPO\_NAME}\_BACKUP

This creates a sibling folder called <YourRepo>\_BACKUP one directory up — but skips the .venv folder, which should never be committed to GitHub or included in backups. It’s a full, clean clone of your project. A perfect safety net.

## 🆘 Troubleshooting

**Issue:** “Not a Git repo”  
**Fix:** Make sure you’re running inside a folder with .git/

**Issue:** Diff is empty  
**Fix:** There may be no changes from that commit to current HEAD

**Issue:** Can’t return  
**Fix:** May be due to deleted branches or stashed errors – switch manually:

git switch <original-branch>  
git stash pop

## 🧪 You’re Now Dangerously Curious

This tool won’t make you a Git master overnight, but it’ll keep you safe, confident, and **curious enough to explore history without breaking things**.

Next steps:

* Learn about git log, git reflog, and git revert
* Start committing in small steps
* Try branching and merging using git switch and git merge

This is how the learning begins. 👨‍🔬

Happy time traveling, Professor Herb.

*— Written with love and caution by your assistant, Navi.*