**🌳 Stage 3 – Git Tree Mastery (My Learning Notes)**

**📂 Folder:** GetInspiredQuoteSite\_GitPro  
**🧠 Purpose:** Deep dive into pro-level Git commands like rebase, stash, squash, tagging, semantic commits, etc.

**✅ 1. Git Rebase – Keeping My Feature Branches Updated**

**Why I used it:**  
To make sure feature/header was based on the latest main without unnecessary merge commits.

**Commands I used:**

git checkout feature/header

git rebase main

**What it did:**  
Re-applied my header work *on top of* the latest version of main in a clean history.

**✅ 2. Git Stash – Temporarily Hiding My Changes**

**Why I used it:**  
I had uncommitted work on feature/header and needed to switch to main for updates.

**Commands I used:**

git stash

git checkout main

# Later...

git stash pop

**What it did:**  
Saved my local changes temporarily and reapplied them after I finished work on the main branch.

**✅ 3. Git Tags – Marking My First Version**

**Why I used it:**  
To label a clean, production-ready commit after merging header and footer changes.

**Commands I used:**

git tag v1.0.0

git push origin v1.0.0

**What it did:**  
Created a version checkpoint (v1.0.0) on my GitHub repo — useful for future releases.

**✅ 4. Semantic Commits – Writing Clean, Meaningful Messages**

**Why I used it:**  
To ensure each commit clearly explains what was changed, making history easy to read and automate (changelogs, etc.).

**Format:**

<type>: <summary>

**Common types I used:**

| **Type** | **When I Use It** |
| --- | --- |
| feat: | When adding a new feature (e.g. header) |
| fix: | When fixing a bug |
| docs: | For documentation or comments |
| style: | Visual/CSS changes |
| refactor: | Code cleanup without changing behavior |
| test: | For writing or fixing tests |
| chore: | For configs or tooling changes |

**Example:**

git commit -m "docs: add attribution comment in footer"

**✅ 5. Amend Commit – Fixing My Last Commit**

**Why I used it:**  
To fix something I forgot in the last commit without creating a new one.

**Commands I used:**

git add forgotten-file.html

git commit --amend

**What it did:**  
Combined the latest change with the previous commit and let me update the message too.

**✅ 6. Interactive Rebase – Cleaning Up Messy Commit History**

**Why I used it:**  
To squash multiple small commits (like debug logs) into one clean commit.

**Commands I used:**

git rebase -i HEAD~3

**What I did:**

* Changed pick to squash for the second and third commits
* Wrote a new summary message like:
* debug: clean up console logging

**Result:**  
3 debug commits merged into one clean commit in the history.

**🔁 Other Git Commands I Practiced**

| **Command** | **What It Did** |
| --- | --- |
| git init | Started a new Git project |
| git status | Checked current file status |
| git log --oneline | Quick view of commit history |
| git merge feature/header | Merged header work into main |
| git push origin branch | Pushed changes to GitHub |
| git tag -a v1.0.0 -m "msg" | Created a tag with a custom message |
| git commit --amend | Edited the most recent commit |
| git rebase -i HEAD~n | Squashed multiple commits into one |
| git config --global core.editor nano | Made Nano default editor for Git |

**🧠 Final Project State**

* main has both header and footer merged
* feature/header was rebased and merged cleanly
* Version v1.0.0 is tagged on the clean production commit
* Commit history is neat and readable
* All commits use semantic conventions

**🚀 Next Topics I Plan to Learn**

* Pull Request workflows
* Protected branches
* Undoing mistakes with git reflog
* Git submodules (for multi-projects)
* Git hooks (automated scripts on commit/push)