Git Mastery Challenge — Solving Pre-Built Conflicts in 'DevOps Simulator'

Git : Git is a **version control system (VCS)** — a tool that helps you **track changes in your code or files over time**. It was created by **Linus Torvalds** in 2005 (the same person who created Linux).

It allows multiple developers to **work on the same project simultaneously**, without overwriting each other's work, and to **revert or review** any changes made in the past.

- **Repository (Repo):** A **repository** is a folder that Git tracks. It contains all your files and the complete change history. You can have a **local repo** (on your computer) and a **remote repo** (on GitHub or another server).
- **Commit:** A **commit** is like a "save point."
- **Branch:** Branches let you work on new features without affecting the main code.
- **Merge:** Combines changes from one branch into another (for example, merging your feature into main).
- **Clone:** Copy a remote repository to your computer.
- **Pull:** Update your local copy with the latest changes from the remote repository.
- **Push:** Uploads your local commits to a remote repository (e.g., GitHub).

GitHub: GitHub is an online platform that lets you **store, manage, and collaborate on Git repositories** in the cloud. It's built on top of **Git**, the version control system created by Linus Torvalds, and adds a user-friendly web interface and collaboration tools.

- **Remote Repositories:** GitHub hosts your Git repositories online so you can: Back up your code safely, Access it from anywhere, Share it with others.
- **Collaboration & Teamwork:** GitHub allows multiple developers to work together on the same project.
- **Branches and Merging:** Developers can work on branches (isolated versions of the project) and later merge their updates into the main branch.
- **GitHub Pages:** GitHub can host static websites for free directly from your repository.

CHANGELOG.md

- 0f809c7: docs: Add FAQ section (Fouzia77, 28 minutes ago)

- a8070be: Save my local changes before merging (Fouzia77, 66 minutes ago)

- 0312c6f: updating main (Hanu Gupta, 4 days ago)

- fadfd24: Revise README for DevOps Simulator project (Hanu Gupta, 4 days ago)

- ae630eb: Initial commit (Hanu Gupta, 4 days ago)

GIT_JOURNEY.md

My Git Mastery Challenge Journey

Student Information

- Name: [MOHAMMAD FOUZIA FIRDOUS]

- Student ID: [23A91A61H1]

- Repository: [https://github.com/Fouzia77/git-solved-23A91A61H1.git]

- Date Started: [27-10-2025]

- Date Completed: [31-10-2025]

Task Summary

Cloned instructor's repository with pre-built conflicts and resolved all

merge conflicts across multiple branches using proper Git workflows.

Commands Used

| Command | Times Used | Purpose |

|---------|------------|----------|

| git clone | 1 | Clone instructor's repository |

| git checkout | 20+ | Switch between branches |

| git branch | 10+ | View and manage branches |

| git merge | 2 | Merge dev and conflict-simulator into main |

| git add | 30+ | Stage resolved conflicts |

| git commit | 15+ | Commit resolved changes |

| git push | 10+ | Push to my repository |

| git fetch | 2 | Fetch updates from instructor |

| git pull | 1 | Pull updates |

| git stash | 2 | Save temporary work |

| git cherry-pick | 1 | Copy specific commit |

| git rebase | 1 | Rebase feature branch |

| git reset | 3 | Undo commits (soft/mixed/hard) |

| git revert | 1 | Safe undo |

| git tag | 2 | Create release tags |

| git status | 50+ | Check repository state |

| git log | 30+ | View history |

| git diff | 20+ | Compare changes |

Conflicts Resolved

Merge 1: main + dev (6 files)

Conflict 1: config/app-config.yaml

- **Issue**: Production used port 8080, development used 3000

- **Resolution**: Created unified config with environment-based settings

- **Strategy**: Keep production as default, add dev as optional

- **Difficulty**: Medium

- **Time**: 15 minutes

Conflict 2: config/database-config.json

- **Issue**: Different database hosts and SSL modes

- **Resolution**: Created separate profiles for production and development

- **Strategy**: Restructured JSON to support both environments

- **Difficulty**: Medium

- **Time**: 10 minutes

Conflict 3: scripts/deploy.sh

- **Issue**: Different deployment strategies (production vs docker-compose)
- **Resolution**: Added conditional logic based on DEPLOY_ENV variable
- **Strategy**: Made script handle both environments dynamically
- **Difficulty**: Hard
- **Time**: 20 minutes

Conflict 4: scripts/monitor.js

- **Issue**: Different monitoring intervals and log formats
- **Resolution**: Environment-based configuration object
- **Strategy**: Used process.env.NODE_ENV to determine behavior
- **Difficulty**: Medium
- **Time**: 15 minutes

Conflict 5: docs/architecture.md

- **Issue**: Different architectural descriptions
- **Resolution**: Merged both descriptions into comprehensive document
- **Strategy**: Created sections for each environment
- **Difficulty**: Easy
- **Time**: 10 minutes

Conflict 6: README.md

- **Issue**: Different feature lists and version numbers
- **Resolution**: Combined all features with clear environment labels
- **Strategy**: Organized features by category
- **Difficulty**: Easy
- **Time**: 10 minutes

Merge 2: main + conflict-simulator (6 files)

[Document the second set of conflicts similarly]

Most Challenging Parts

1. **Understanding Conflict Markers**: Initially confused by `<<<<<<<`, `=======`, `>>>>>>>` symbols. Learned that HEAD is current branch and the other side is incoming changes.

2. **Deciding What to Keep**: Hardest part was choosing between conflicting code. Learned to read both versions completely before deciding.

3. **Complex Logic Conflicts**: deploy.sh had completely different logic. Had to understand both approaches before combining.

4. **Testing After Resolution**: Making sure resolved code actually worked was crucial.

Key Learnings

Technical Skills

- Mastered conflict resolution process

- Understood merge conflict markers

- Learned to use git diff effectively

- Practiced all major Git commands

Best Practices

- Always read both sides of conflict before resolving

- Test resolved code before committing

- Write detailed merge commit messages

- Use git status frequently

- Commit atomically

Git Workflow Insights

- Conflicts are normal, not errors

- Take time to understand both changes

- When in doubt, ask for clarification

- Document your resolution strategy

- Keep calm and read carefully

Reflection

This challenge taught me that merge conflicts aren't scary - they're just Git asking "which version do you want?". The key is understanding what each side is trying to do before combining them. I now feel confident handling conflicts in real projects.

The hands-on practice with all Git commands (especially rebase and cherry-pick) was invaluable. I understand the difference between merge and rebase, and when to use each. Most importantly, I learned that git reflog is a lifesaver!