

Git Source Control

Table of Contents

-  Project Overview
-  Branch Structure Summary
-  Git Configuration
-  Core Workflow Commands
-  Handling Merge Conflicts in VS Code
-  Reverting a Pushed Commit (Safe Method)
-  Merge vs Pull vs Rebase Summary
-  List All Branches in a Git Repository
-  Visualizing Git History
-  Best Practices & Lessons Learned
-  Practice Resources
-  Reference Materials

Project Overview

This project documents a full hands-on implementation of Git Flow, based on the article [A Successful Git Branching Model](#). It is designed as a comprehensive guide and training resource for engineers and developers with little to no Git experience.

The objective is to teach:

- How to use version control for collaborative development
- How to organize work using `main`, `develop`, `feature`, `release`, and `hotfix` branches
- How to manage merges, rebase, tagging, and upstream tracking
- How to resolve merge conflicts and preserve project history

Branch Structure Summary

Following Git Flow:

- `main` → Stable, production-ready code (only updated from `release/*` or `hotfix/*`)
- `develop` → Integration branch for all features
- `feature/*` → Short-lived branches for individual features (e.g., `feature/login-ui`)
- `release/*` → Pre-release stabilization branches (e.g., `release/<version>`)
- `hotfix/*` → Emergency fixes branched from `main` (e.g., `hotfix/<version>`)

Tags like `v1.0.0`, `v1.1.0`, and `v<version>` mark official release points.

 **Note:** Each feature should be developed in a **separate branch** from `develop`. Feature branches follow the naming convention `feature/*`, where * is a brief, dash-separated description (e.g., `feature/export-csv`).

Git Configuration (Recommended for Team)

SSH Tag Signing Setup

```
git config --global gpg.format ssh # Use SSH key for tag signing
```

```
git config --global user.signingkey ~/.ssh/id_ed25519.pub # Sets your public key for tag signing
```

Automatically Set Upstream for New Branches

```
git config --global push.autoSetupRemote true # Auto-sets upstream when pushing new branches
```

Git Line Ending Configuration (Cross-Platform)

Add this `.gitattributes` file:

```

# Use LF (Unix-style) endings for source code
*.c      text eol=lf
*.cpp    text eol=lf
*.h      text eol=lf
*.py     text eol=lf
*.sh      text eol=lf
*.md      text eol=lf
*.txt    text eol=lf

# Treat .git* files as text with LF
.gitignore text eol=lf
.gitattributes text eol=lf

# Use CRLF for Windows batch files (optional, for compatibility)
*.bat    text eol=crlf

# Prevent Git from touching line endings for binary files
*.jpg    binary
*.jpeg   binary
*.png    binary
*.gif    binary
*.pdf    binary
*.zip    binary
*.exe    binary

git add --renormalize . # Applies .gitattributes settings retroactively

git commit -m "Normalize line endings using .gitattributes" # Commits normalized line endings

```

Core Workflow Commands (Fully Explained)

Create Tag

```
git tag -sa v1.0.0 -m "Initial Version Release" # Creates a signed annotated tag
```

```
git tag -sa v<version> -m "Release Description" # Template for future tags
```

Create develop From main

```
git checkout main # Switches to main branch
```

```
git pull origin main # Syncs with remote main branch
```

```
git checkout -b develop # Creates and switches to new develop branch
```

```
git push -u origin develop # Pushes develop and sets upstream
```

Create a Feature Branch

```
git checkout develop # Switch to develop
```

```
git pull origin develop # Ensure it's up-to-date
```

```
git checkout -b feature/my-feature # Create and switch to feature branch
```

Maintain a Feature Branch

```
git add . # Stage all modified and new files
```

```
git commit -m "<your message>" # Commit your changes
```

```
git push # Push changes to remote feature branch
```

Merge a Feature into develop

```
git checkout develop # Switch to develop
```

```
git pull origin develop # Update with latest changes
```

```
git merge --no-ff feature/my-feature -m "Merge feature/my-feature" # Merge feature preserving his
```



```
git push # Pushes merged changes
```

Start a Release Branch

```
git checkout develop # Start from develop
```

```
git pull origin develop # Update local develop
```

```
git checkout -b release/<version> # Create and switch to release branch
```

Maintain a Release Branch

```
git add . # Stage all modified and new files
```

```
git commit -m "<your message>" # Commit your changes
```

```
git push # Push changes to remote release branch
```

Finish the Release

```
git checkout main # Switch to main for final release merge
```

```
git merge --no-ff release/<version> -m "Release v<version>" # Merge release into main with history
```

```
git tag -sa v<version> -m "Tagging release v<version>" # Create signed tag
```

```
git push # Push main branch updates
```

```
git push origin v<version> # Push tag to remote
```

```
git checkout develop # Switch to develop
```

```
git merge --no-ff release/<version> -m "Merge release v<version>" # Merge release back to develop
```

```
git push # Push develop with merged release
```

```
git branch -d release/<version> # Delete local release branch
```

```
git push origin --delete release/<version> # Delete remote release branch
```

Creating a Hotfix Branch

```
git checkout main # Switch to main
```

```
git pull origin main # Sync latest main
```

```
git checkout -b hotfix/<version> # Create hotfix branch
```

Maintain a Hotfix Branch

```
git add . # Stage all modified and new files
```

```
git commit -m "<your message>" # Commit your changes
```

```
git push # Push changes to remote hotfix branch
```

Merge Hotfix into main , Tag It

```
git checkout main # Switch to main
```

```
git merge --no-ff hotfix/<version> -m "Apply hotfix v<version>" # Merge hotfix into main
```

```
git tag -sa v<version> -m "Hotfix v<version>" # Tag hotfix version
```

```
git push # Push hotfix to remote
```

```
git push origin v<version> # Push hotfix tag
```

Merge Hotfix into develop

```
git checkout develop # Switch to develop
```

```
git pull origin develop # Update local develop
```

```
git merge --no-ff hotfix/<version> -m "Backport hotfix v<version>" # Merge hotfix into develop
```

```
git push # Push changes
```

Clean Up Hotfix

```
git branch -d hotfix/<version> # Delete local hotfix branch
```

```
git push origin --delete hotfix/<version> # Delete remote hotfix branch
```

Syncing Hotfix with Latest Main Code

```
git checkout hotfix/<version> # Switch to hotfix
```

```
git fetch origin # Get latest remote changes
```

```
git merge origin/main --no-ff -m "Sync hotfix with main" # Merge main into hotfix
```

```
git push # Push updated hotfix
```

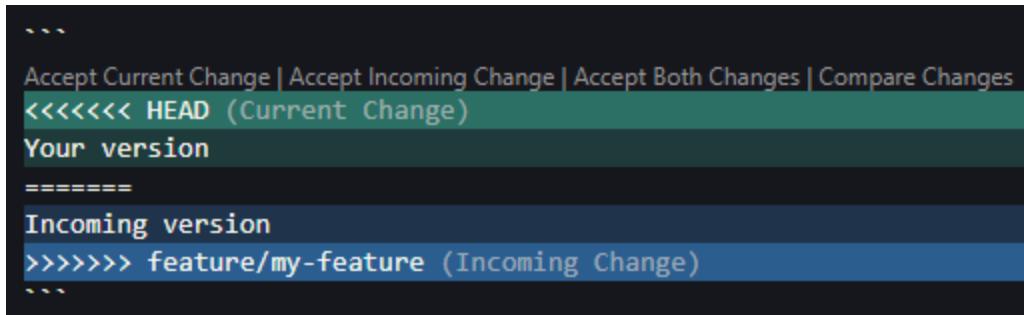
Handling Merge Conflicts in VS Code

1. Start the merge:

```
git merge feature/my-feature # Attempt to merge feature
```

2. VS Code highlights conflicted files

3. Open files to resolve blocks:



The screenshot shows a merge conflict in the GitSrcCtrl interface. At the top, there are four buttons: 'Accept Current Change', 'Accept Incoming Change', 'Accept Both Changes', and 'Compare Changes'. Below these buttons, the code is divided into two sections: 'Your version' (containing '<<<<< HEAD (Current Change)' and '=====') and 'Incoming version' (containing 'Incoming version' and '>>>>> feature/my-feature (Incoming Change)'). The 'Incoming version' section is highlighted with a blue background.

4. Use the resolution buttons provided in the editor:

- **Accept Current Change** – Keeps the version from your current branch (`HEAD`).
- **Accept Incoming Change** – Uses the version from the branch you're merging in.
- **Accept Both Changes** – Keeps both versions, stacked one after the other.
- **Compare Changes** – Opens a side-by-side diff view to help you decide.

5. Save and stage the resolved file:

```
git add . # Stage resolved files
```

6. Commit the merge:

```
git commit -m "Resolve merge conflict" # Finalize merge
```

7. Push the result to remote repository:

```
git push
```

Reverting a Pushed Commit (Safe Method)

If you've already pushed to the remote and others may have pulled it, the safest way to undo changes is by **reverting** the commit:

```
git revert <commit-hash>
git push
```

This creates a new commit that undoes the changes without modifying the existing history — ideal for shared branches.



Merge vs Pull vs Rebase Summary

Command	What It Does	When to Use
merge	Combines branches with a merge commit	Always use for features, releases, and hotfixes
pull	Shortcut for fetch + merge of current upstream branch	Keep local branch up to date
rebase	Rewrites commits onto a new base (linear history)	Use only on local/private feature branches



List All Branches in a Git Repository

To view all local branches:

```
git branch
```

To view all remote branches:

```
git branch -r
```

To view both local and remote branches:

```
git branch -a
```

To see branches with their last commit info:

```
git branch -vv
```



Visualizing Git History

```
git log --oneline --graph --all --decorate # Pretty git history visualization
```

```
git log --oneline --graph --all --decorate > git_graph.txt # Save history view to file
```

Git Graph (VS Code Extension)

The **Git Graph** extension for VS Code provides a powerful visual interface to explore your Git repository's history. It displays branches, merges, tags, and commit messages in a clean, interactive format.

This graph helps validate whether Git Flow conventions—like release merges, tag placement, and hotfix branching—have been followed correctly.

Graph	Description	Date	Author	Commit
Uncommitted Changes (3)				
○	○ develop origin new git graph	12 May ...	*	*
●	Release v4.0.0	12 May ...	Connor	9b2a251d
●	● main origin ● origin/HEAD ● v4.0.0 Release v4.0.0	12 May ...	Connor	0a2e8f99
●	Final Push for Release v4.0.0	12 May ...	Connor	2565f6d2
●	more small changes	12 May ...	Connor	cdbd176d
●	Normalize line endings using .gitattributes	12 May ...	Connor	250a8be6
●	Made some small tweaks before release	12 May ...	Connor	931cd396
●	minor updates	12 May ...	Connor	8b9174e6
●	● v3.0.0 Weekend Development is Over	12 May ...	Connor	5740510a
●	GitSrcCtrl.md is looking pretty good. Need to run through the commands to look for ...	11 May ...	Connor	bf340ad7
●	Added resources to dev channel	11 May ...	Connor	da0f1309
●	updated git graph txt file to use everything referenced from main branch	11 May ...	Connor	c2b04e55
●	Added command history list of relevant commands	10 May ...	Connor	92fb7de3
●	Merge hotfix 2 into dev, trying not to remove secret message	10 May ...	Connor	af0ac877
●	Merge hotfix 2 into main	9 May 2...	Connor	d7f60196
●	● hotfixes origin added 2nd hotfix	9 May 2...	Connor	c3a655d8
●	Merge main into hotfixes to sync with latest production changes	9 May 2...	Connor	fde973ac
●	added secret message to dev branch	9 May 2...	Connor	cce8b265
●	Although Identical, I want to pull development from main again, since we did have an ...	9 May 2...	Connor	dd8974d8
●	● v2.0 Merging develop with main	9 May 2...	Connor	4da3026c
●	resolved merge conflicts, all features merged with develop	9 May 2...	Connor	cb0f37b3
●	Merge branch 'features-merge_vs_pull_vs_rebase' into develop	9 May 2...	Connor	4206e0a8
●	● feature-hidden_cost_of_merge origin Added feature about merging while preservi...	9 May 2...	Connor	97284eb6
●	Merge branch 'hotfixes' into develop	9 May 2...	Connor	9f34ebb4
●	● features-merge_vs_pull_vs_rebase origin Updated table with pull details	9 May 2...	Connor	f05ca6b8
●	Added initial table explaining merge vs pull vs rebase	9 May 2...	Connor	dc7d5b75
●	added hotfix flow	9 May 2...	Connor	51187b78
●	● v1.1 Severe bug fixed, hotfix	9 May 2...	Connor	7db3989f
●	Added Src PDF	9 May 2...	Connor	c5168bd6
●	Modified md file	9 May 2...	Connor	149208fb
●	Committing md file to dev branch	8 May 2...	Connor	32609fc8
●	● v1.0 Added git ignore, pushing to main for first time	8 May 2...	Connor	103b2445
●		8 May 2...	Connor	970cc8ab

✓ Key Git Flow Confirmations:

- `release/4.0.0` was created and used for staging commits (e.g., "Final Push for Release v4.0.0").
- The release was merged into `main` and **tagged** `v4.0.0`, as required.
- It was then merged back into `develop`, completing the release cycle.

- The branch reference for `release/4.0.0` is deleted, but its history is preserved via merges.
- Multiple hotfixes exist and are visible with tags (`v1.1`, `v2.0`) and a dedicated `hotfixes` branch.

⚠️ Issues to Note:

- **Hotfix Divergence:**

The second hotfix (`added 2nd hotfix`) exists on the `hotfixes` branch and was merged into `main`, but **was not merged back into `develop`**.

- This breaks the Git Flow model, which requires hotfixes to be merged into both `main` and `develop` to prevent regressions.
- Consider creating a merge commit from `hotfixes` into `develop` to synchronize branches.

Recommendation:

Regularly use Git Graph or `git log --oneline --graph --all --decorate` to audit your branching and merging practices. This ensures alignment with Git Flow and avoids lost fixes or duplicate work.

🧠 Best Practices & Lessons Learned

- Use `--no-ff` to preserve branch history
- One feature per `feature/*` branch from `develop`
- Create `release/*` for each version
- Hotfix from `main`, merge into `main` and `develop`
- Tag releases **after** merging into `main`
- Clean up merged branches
- Avoid rebase on shared branches

🧪 Practice Resources

- Simulate merge conflicts
- Try reverting a merge commit:

```
git revert -m 1 <merge-commit> # Reverts a specific merge
```

- Use `git stash` for managing WIP
- Use reflog to recover deleted branches

Reference Materials

- [A successful Git branching model](#)
- `cmd_hist` : full command history
- `git_graph.txt` : visual history output
- This document: internal Git Flow tutorial