Lesson Plan

Handling Conflicts in Git

PW SKILLS

Handling Conflicts in Git

Introduction

Handling conflicts is an essential part of using Git effectively. Conflicts occur when multiple developers make changes to the same lines of a file or when changes overlap in a way that Git cannot automatically reconcile. Understanding how to manage and resolve conflicts is crucial for maintaining a smooth workflow. This document will explore different types of conflicts, how to resolve them, and best practices for conflict management.

1. Understanding Conflicts

1.1 What is a Conflict?

A conflict arises when Git is unable to automatically merge changes from different branches. This usually happens when two changes affect the same part of a file, making it impossible for Git to decide which change to keep.

1.2 Types of Conflicts

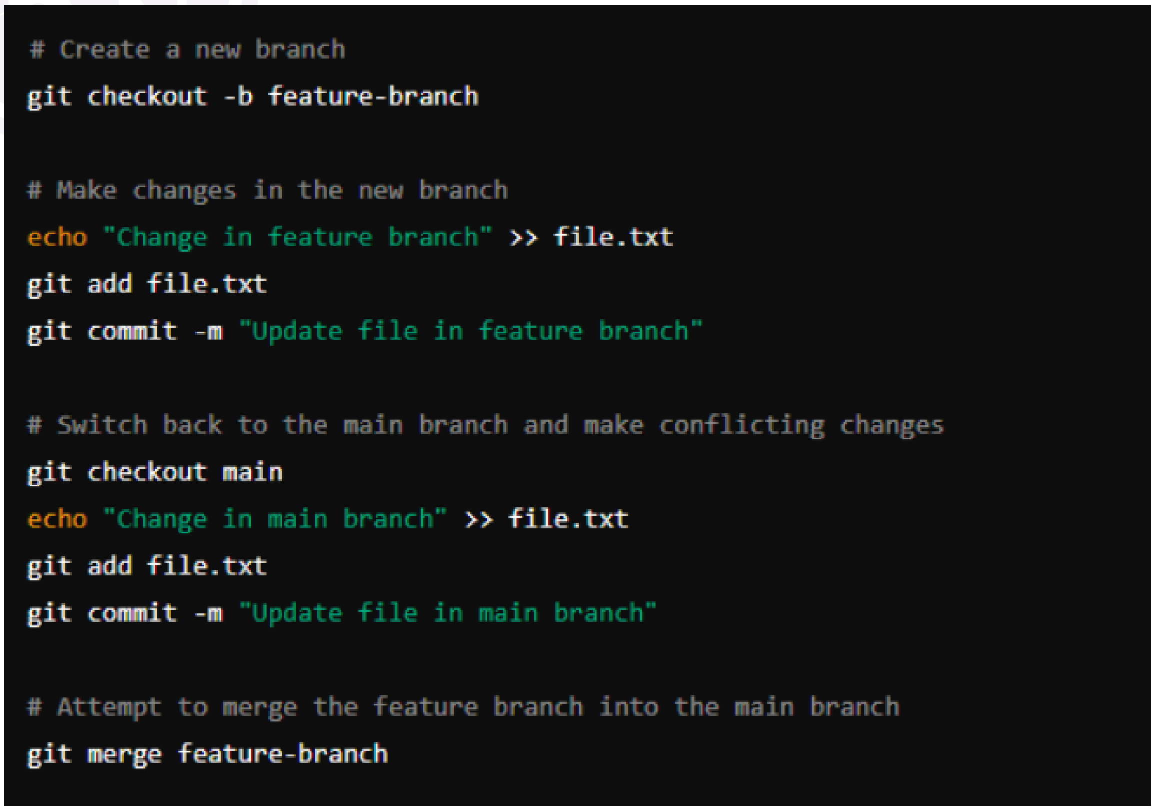
- Content Conflicts: When changes are made to the same lines of a file in different branches.
- Tree Conflicts: When changes affect the directory structure, such as renaming or deleting files.
- Semantic Conflicts: When changes are logically conflicting, even if they are not directly overlapping.

2. Common Scenarios for Conflicts

2.1 Merge Conflicts

Merge conflicts occur during the merging of branches. This is the most common type of conflict.

Example



```
# Create a new branch
git checkout -b feature-branch

# Make changes in the new branch
echo "Change in feature branch" >> file.txt
git add file.txt
git commit -m "Update file in feature branch"

# Switch back to the main branch and make conflicting changes
git checkout main
echo "Change in main branch" >> file.txt
git add file.txt
git commit -m "Update file in main branch"

# Attempt to merge the feature branch into the main branch
git merge feature-branch
```

2.2 Rebase Conflicts
Rebase conflicts occur when rebasing a branch onto another branch. Rebasing rewrites the commit history, which can lead to conflicts.

Example

```
# Make conflicting changes in the main branch
echo "Change in main branch" >> file.txt
git add file.txt
git commit -m "Update file in main branch"

# Switch to the feature branch and rebase onto the main branch
git checkout feature-branch
git rebase main
```

2.3 Cherry-Pick Conflicts
Cherry-pick conflicts occur when selectively applying commits from one branch to another.

Example

```
# Make conflicting changes in the main branch
echo "Change in main branch" >> file.txt
git add file.txt
git commit -m "Update file in main branch"

# Switch to the feature branch and cherry-pick the commit from main
git checkout feature-branch
git cherry-pick <commit-hash>
```

3. Resolving Conflicts

3.1 Identifying Conflicts
When a conflict occurs, Git marks the conflicted areas in the affected files. Conflict markers indicate the changes from each branch.

Example of Conflict Markers

```
<<<<<<< HEAD
Change in main branch
=======
Change in feature branch
>>>>>>> feature-branch
```

3.2 Manual Conflict Resolution

To resolve conflicts manually, you need to edit the conflicted file and decide which changes to keep. After resolving, stage and commit the changes.

Example

```
# Edit the file to resolve conflicts
# Keep the desired changes and remove conflict markers

# Stage the resolved file
git add file.txt

# Commit the resolution
git commit -m "Resolved merge conflict"
```

3.3 Using Conflict Resolution Tools

Git integrates with various conflict resolution tools that provide a graphical interface to simplify the process.

Example

```
# Launch a conflict resolution tool (e.g., vimdiff)
git mergetool
```

3.4 Aborting a Merge or Rebase

If you are unable to resolve conflicts, you can abort the merge or rebase process.

Example

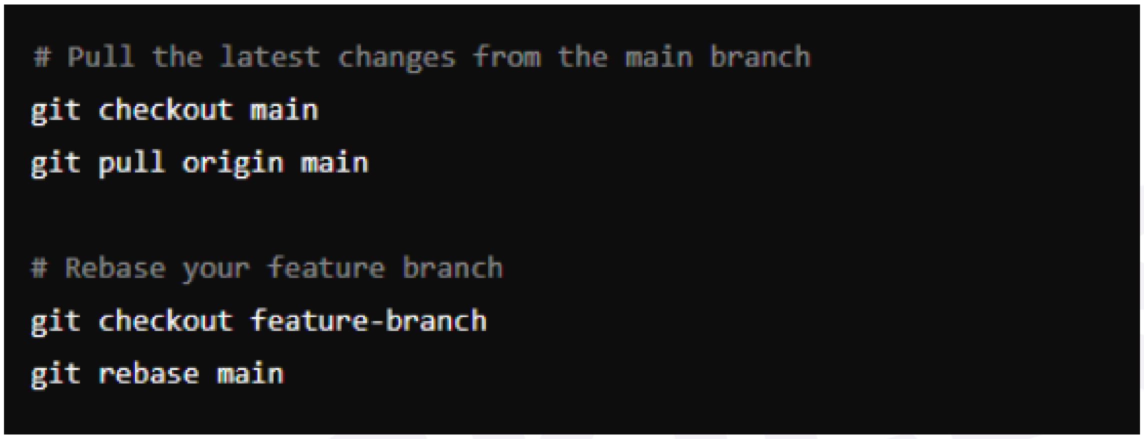
```
# Abort a merge
git merge --abort

# Abort a rebase
git rebase --abort
```

4. Best Practices for Conflict Management

4.1 Communicate with Your Team
Effective communication with your team can prevent many conflicts. Discuss changes and coordinate merges to avoid overlapping work.

4.2 Pull and Rebase Regularly
Regularly pull the latest changes from the main branch and rebase your feature branch. This keeps your branch up-to-date and reduces the risk of conflicts.

Example



```
# Pull the latest changes from the main branch
git checkout main
git pull origin main

# Rebase your feature branch
git checkout feature-branch
git rebase main
```

4.3 Use Smaller Commits
Smaller, frequent commits make it easier to identify and resolve conflicts. They also provide a clearer commit history.

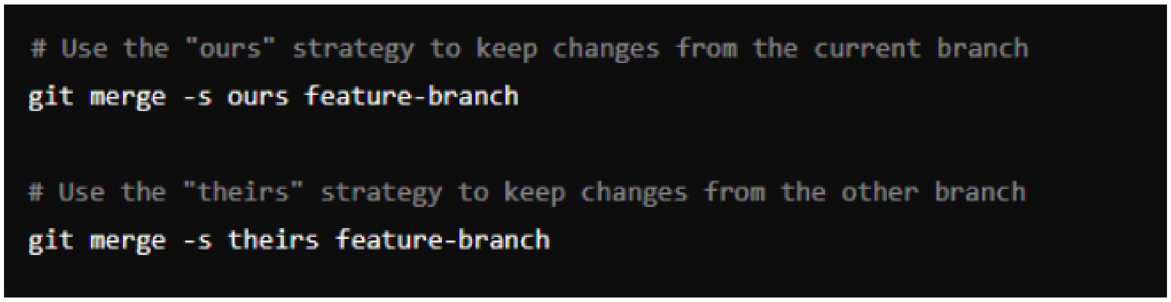
4.4 Write Clear Commit Messages
Clear commit messages help others understand the changes you made, making it easier to resolve conflicts.

4.5 Use Feature Branches
Use feature branches for new work. This isolates changes and makes it easier to manage conflicts.

5. Advanced Conflict Resolution Techniques

5.1 Conflict Resolution Strategies
Git offers various strategies to resolve conflicts during merges.

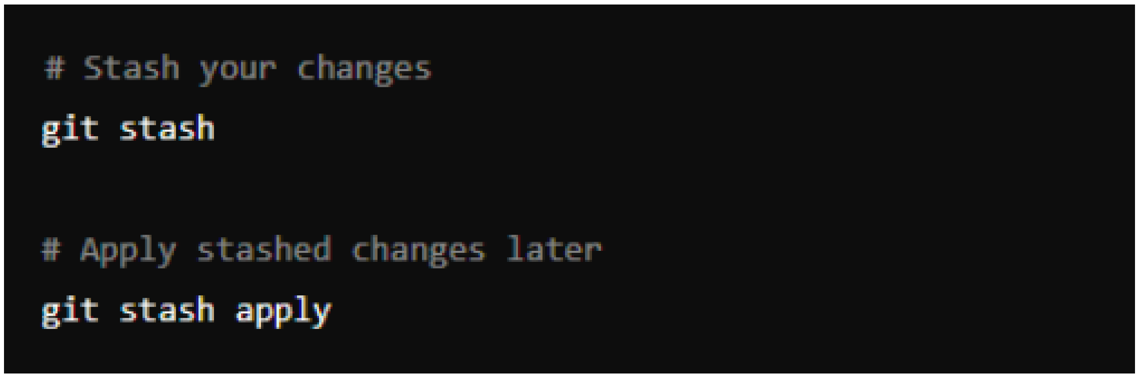
Example



```
# Use the "ours" strategy to keep changes from the current branch
git merge -s ours feature-branch

# Use the "theirs" strategy to keep changes from the other branch
git merge -s theirs feature-branch
```

5.2 Stash Changes
If you need to switch branches but have uncommitted changes, you can stash them to avoid conflicts.

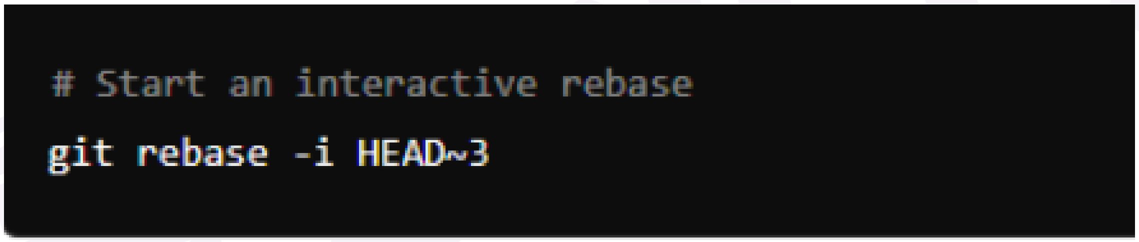
Example



```
# Stash your changes
git stash

# Apply stashed changes later
git stash apply
```

5.3 Interactive Rebase
Interactive rebase allows you to edit, reorder, and squash commits, helping to resolve conflicts before they occur.

Example



```
# Start an interactive rebase
git rebase -i HEAD~3
```

Conclusion

Handling conflicts in Git is an essential skill for any developer working in a collaborative environment. By understanding the types of conflicts, how to resolve them, and following best practices, you can maintain a smooth workflow and minimize disruptions. Git provides powerful tools and strategies to help manage conflicts effectively, ensuring that your project stays on track.