# Conflict Resolution & Reference

*AiCE Software Engineering Foundations — The Developer Environment*

Branching gives you freedom to work in parallel, but that freedom occasionally collides — two branches editing the same lines of the same file in different ways. This lesson explains exactly what that collision looks like, how to resolve it calmly, how to read GitHub's ahead/behind indicators, and closes with a full command reference for everything covered across this unit.

## 1. Conflict Resolution

A merge conflict occurs when the same file has been modified in two branches in ways that Git cannot automatically reconcile. This is common in team environments, and it happens even when you're working entirely alone across your own branches.

### How Conflicts Arise

Suppose car.txt exists in both the main branch and a features branch. You edit the second line of car.txt in main and push. Then you switch to features and edit that same second line differently, and push again. Now, when you try to merge features into main, Git cannot decide which version of that line to keep — so it asks you to decide.

### What a Conflict Looks Like

When a conflict occurs, Git modifies the affected file and inserts conflict markers directly into it:

```
<<<<<<< HEAD
African cars will be electric and solar-powered.
=======
African transportation is being revolutionized by electric
vehicles.
>>>>>>> features
```

The section between <<<<<<< HEAD and ======= is the version from your current branch. The section between ======= and >>>>>>> features is the version from the branch you're merging in.

### Resolving the Conflict

Open the conflicted file in Vim or Nano. Delete the conflict markers entirely and keep the content you actually want — which might mean choosing one version, the other, or blending both into something new:

```
African cars will be electric, solar-powered, and locally
manufactured.
```

After resolving, save the file, then complete the merge exactly as you would any other change:

```
git add .
git commit -m "Resolve merge conflict in car.txt"
git push
```

### Preventing Conflicts — Best Practices

Always run git pull before starting work on any branch.

When switching to a different branch, merge the latest changes from main first.

Communicate with collaborators about which files are currently being edited.

Commit and push regularly — small, frequent commits reduce the risk of large conflicts.

## 2. Pull Conflicts — A Special Case

Conflicts can also arise during git pull. This happens when:

1. You edit a file locally, without pushing.

2. Someone else — or you, through the GitHub website — edits that same file on GitHub.

3. You then run git pull.

Git will report a conflict, and you resolve it the same way as before: open the file, remove the conflict markers, keep the correct content, then:

```
git add .
git commit -m "Resolve pull conflict in hello.txt"
git push
```

> **The Golden Rule of Pull Conflicts**
> *The easiest way to avoid pull conflicts is to run git pull before you start editing anything, every time you sit down to work. Make this a reflex, not an afterthought.*

## 3. Branches — Ahead and Behind

GitHub's branch comparison page shows you whether one branch is Ahead or Behind another:

**Ahead** — the branch contains commits that main does not yet have.

**Behind** — main contains commits that this branch has not yet received.

If a feature branch is 3 commits ahead of main, there are 3 committed changes in that branch waiting to be merged. If it's 2 commits behind, there are 2 changes in main that the feature branch hasn't received yet. After a merge in both directions, both branches should show 0 ahead and 0 behind — fully synchronized.

## 4. Full Git Command Reference

Every Git command introduced across this unit, in one place:

| Command | Description |
|---|---|
| git --version | Check the installed Git version |
| git config --global user.email "..." | Set your email for commits |
| git config --global user.name "..." | Set your name for commits |
| git clone <url> | Clone a repository from GitHub to your local machine |
| git status | Show the status of changed files |
| git add . | Stage all changed files for commit |

| Command | Description |
|---|---|
| git add filename | Stage a specific file |
| git commit -m "message" | Commit staged files with a message |
| git push | Push commits to the remote repository on GitHub |
| git pull | Pull the latest changes from GitHub into your local repository |
| git branch | List all local branches |
| git branch name | Create a new branch |
| git switch name | Switch to a branch |
| git merge name | Merge a branch into the current branch |
| git push --set-upstream origin name | Push a new branch to GitHub for the first time |

## Key Takeaways

A merge conflict is not a failure — it's Git being honest that it can't guess which version of a line you meant to keep, and handing that decision back to you. Resolve it by editing out the conflict markers, keeping the right content, and completing the commit as usual. Combined with the ahead/behind indicators and the full command reference above, you now have everything you need for the complete Git and GitHub workflow: from creating a repository and configuring Git, through cloning, the add/commit/push cycle, pulling, branching, merging, and resolving conflicts along the way. Every team, every company, and every serious open-source project runs on exactly this workflow — it's worth mastering.