Merge Conflicts: What to Do When Git Says CONFLICT

Git isn't broken; it's handing you a decision

You pull, or you merge a branch, and Git stops you cold: `CONFLICT (content): Merge conflict in main.js`. Your first thought is that you broke something and lost work. You didn't. A merge conflict is the one situation where Git refuses to guess and hands the decision back to you.

What a conflict actually is

Git merges changes automatically almost every time. It only stops when two edits touch the same lines of the same file and it cannot tell which one you meant. That is the whole problem: not corruption, not lost work, just two versions of the same lines with Git waiting for you to pick.

You caused it, usually, by editing the same file in two places. You changed `intake.py` on your desktop, changed the same function on the server, and now they disagree. Git will not silently overwrite one with the other. Good. That is it doing its job.

What it looks like

Open the conflicted file and you will see Git's markers:

```
<<<<<<< HEAD
port = 8087   # the version already on the branch you are merging into
=======
port = 9090   # the version coming in from the other branch
>>>>>>> feature-branch
```

Three markers. Everything between `<<<<<<< HEAD` and `=======` is your current version. Everything between `=======` and `>>>>>>>` is the incoming version. Git wants you to decide what the final lines should be.

How to fix it

Editing is the whole job:

1. Open the file and find every conflict block; search for `<<<<<<<`.

2. Decide the correct final content. Keep yours, keep theirs, keep a combination, or write something new.
3. Delete all three marker lines so only real code remains.
4. Save, then tell Git the conflict is resolved and finish the merge.

```
git add intake.py        # mark this file resolved
git commit               # complete the merge (opens a message; just save it)
```

Repeat for every conflicted file. `git status` lists them all and shows which still need attention.

The escape hatch

Not ready to deal with it? Back all the way out:

```
git merge --abort        # undo the merge, return to exactly before you started
```

Nothing is lost. You are back on solid ground, both versions intact, free to try again when you have time. This is the merge-conflict cousin of the rollback tip: the "get me out of here" button.

The playbook

Situation	Command
See which files are conflicted	`git status`
Mark a file resolved after editing	`git add <file>`
Finish the merge	`git commit`
Bail out entirely	`git merge --abort`
Keep only your version of a file	`git checkout --ours <file>`
Keep only the incoming version	`git checkout --theirs <file>`

Why this matters for a lawyer who codes

You run the same repos across more than one machine: the workstation, the server, the VPS. The day you edit the same file in two places before syncing, you get a conflict. It is not a sign you did something wrong; it is the normal cost of working in more than one place. Knowing the four moves above turns a thirty-minute panic into a two-minute fix.

No terminal? Same idea

GitHub Desktop lists the conflicted files and gives you buttons to keep one side or open the file and edit by hand. VS Code puts "Accept Current," "Accept Incoming," and "Accept Both" buttons right above each block. Same three markers underneath; you are clicking instead of deleting them yourself.

> **Using AI to help with this.** Paste the conflicted section, markers and all, and say: "Here is a Git merge conflict. The HEAD side is my server config; the incoming side is my local change. Keep the port from HEAD but the logging change from the incoming side, and give me the resolved block with the markers removed." You get clean lines to drop straight in.