

# Git merge conflict basics quick reference

## Merge conflict indication

```
PS C:\Users\Student\Workspace\GitMergeConflictTutorial> git checkout branch-a
Switched to branch 'branch-a'
PS C:\Users\Student\Workspace\GitMergeConflictTutorial> git add .
PS C:\Users\Student\Workspace\GitMergeConflictTutorial> git commit -m "Change Hola to Meow"
[branch-a 2134ab2] Change Hola to Meow
1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\Student\Workspace\GitMergeConflictTutorial> git checkout main
Switched to branch 'main'
PS C:\Users\Student\Workspace\GitMergeConflictTutorial> git merge branch-a
Auto-merging src/main/java/org/example/Main.java
CONFLICT (content): Merge conflict in src/main/java/org/example/Main.java
Automatic merge failed; fix conflicts and then commit the result.
```

## Most common cause and prevention

- Cause - One developer attempts to merge their personal branch into the main branch when changes were already on remote on the remote main (origin).
- Prevention - Each developer should do a `git pull` before they (1) begin working and (2) before they attempt to merge into main.

## Merge conflict resolution steps:

Important: Do not add any new code while in the middle of a merge conflict! Wait to resolve it before changing anything else.

(1) Open the project in VSCode.

- Make sure you are on the main branch.
  - Use `git branch` to see which branch you're on.
  - Use `git checkout main` to get onto the main branch.

(2) Open the file that the console output indicated as in conflict.

(2) Look for the lines marked with angle brackets.

- The angles/arrows pointing left (<<<<<<) are the current changes at the HEAD of your branch. These are your changes.

- The angles/arrows pointing right (>>>>>>) indicate the incoming changes from the conflicting branch. These are the other developer's changes.
- The changes are separated by equals signs (=====).

```

J Main.java ! x
src > main > java > org > example > J Main.java
1  package org.example;
2
3  public class Main {
4      Run | Debug
5      | public static void main(String[] args) {
6          Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
7          <<<<<<< HEAD (Current Change)
8          | System.out.println("Howdy world!");
9          =====
10         | System.out.println("Meow world!");
11         >>>>>> branch-a (Incoming Change)
12     }
13 }

```

(3) Look at the options listed about the Current Change at HEAD. Select one of the following:

- *Accept Current Change* (your changes)
- *Accept Incoming Change* (the other developer's changes)
- *Accept Both Changes* - To do this, you'll have to manually merge them together.
- *Compare Changes* - See what is contained in each version of the file.

(4) Do an `add`, `commit`, and `push` to get the changes into the remote repository (`origin`).

(5) Both developers should now `pull` the changes from `main` into their respective branches.

## Undoing a merge

If you aren't sure how to handle a merge conflict and think you may need assistance, use `git merge --abort` to cancel it. This won't fix the conflict, but it will allow you to backtrack and wait for help.