**Objective**

When two users (or two branches) change the same part of a file differently, Git can’t automatically decide whose changes to keep — this is a **merge conflict**. Conflict resolution means manually editing the file to choose the final content, then marking the conflict as resolved and committing the result.

**How to Resolve Merge Conflicts**

1. **Identify the conflict** – Git will mark the conflicting section in the file with:

<<<<<<< HEAD

// Your changes

=======

// Other branch's changes

>>>>>>> branch-name

1. **Manually edit** – Decide whether to keep your changes, the other branch’s changes, or a combination.
2. **Mark resolved** – After editing:

git add filename

git commit

1. **Push changes** – To update the remote repository.

**Lab Workflow Summary**

1. **Verify clean master**

git checkout master

git status

1. **Create branch & add file**

git checkout -b GitWork

echo "<msg1>" > hello.xml

git add hello.xml

git commit -m "Added hello.xml in GitWork"

1. **Switch to master & add different file content**

git checkout master

echo "<msg2>" > hello.xml

git add hello.xml

git commit -m "Added hello.xml in master"

1. **View history & differences**

git log --oneline --graph --decorate --all

git diff master GitWork

1. **Merge branch into master (will cause conflict)**

git merge GitWork

1. **Resolve conflict using 3-way merge or P4Merge**
   * Manually edit hello.xml to final content.
   * Save file, then:

git add hello.xml

git commit -m "Resolved merge conflict in hello.xml"

1. **Ignore backup files & commit**

echo "\*.bak" >> .gitignore

git add .gitignore

git commit -m "Ignore backup files"

1. **Delete merged branch**

git branch -d GitWork

1. **View final log**

git log --oneline --graph –decorate

