**4.GIT-HOL**

**GIT Merge Conflicts**

In collaborative software development, multiple developers often work on the same codebase simultaneously. This can lead to merge conflicts when changes from different branches overlap. Git provides tools and workflows to resolve such conflicts effectively, ensuring code integrity and smooth collaboration.

**Objectives:**

1. Understand how merge conflicts occur in Git.
2. Learn how to handle and resolve merge conflicts during integration.
3. Practice using tools like Git Bash and P4Merge for conflict visualization and resolution.
4. Gain hands-on experience with:

Branching and merging

Conflict detection

3-way conflict resolution

Cleaning up post-merge state (e.g., updating .gitignore, deleting merged branches)

**Requirements:**

A GitHub account (please create a free account; do **not** use Cognizant credentials)

Git installed on your system

P4Merge tool installed for visual diff and merge

**Step 1**: Verify if master is in clean state

git checkout master  
 git status

Ensure there are no uncommitted changes.

**Step 2**: Create a branch “GitWork” and add a file “hello.xml”

git checkout -b GitWork  
 echo"<greeting>Hello from GitWork branch</greeting>"> hello.xml  
 git add hello.xml  
 git commit -m"Add hello.xml in GitWork branch"

**Step 3**: Update the content of “hello.xml” and observe the status

Echo"<note>This is an update in GitWork branch</note>">> hello.xml  
 git status

**Step 4**: Commit the changes to reflect in the branch

git add hello.xml  
 git commit -m"Update hello.xml in GitWork branch"

**Step 5**: Switch to master

git checkout master

**Step 6**: Add a file “hello.xml” to the master with different content

echo"<greeting>Hello from master branch</greeting>"> hello.xml  
 git add hello.xml  
 git commit -m"Add hello.xml in master with different content"

**Step 7**: Observe the log

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

**Step 8**: Check the differences with Git diff tool

git diff master GitWork

**Step 9**: Use P4Merge tool for better visualization

git mergetool --tool=p4merge

**Step 10**: Merge the branch to master

git merge GitWork

**Step 11**: Observe the git markup

Git will show a conflict in hello.xml with markers like:

<<<<<<< HEAD  
 <greeting>Hello from master branch</greeting>  
 =======  
 <greeting>Hello from GitWork branch</greeting>  
 <note>This is an update in GitWork branch</note>  
 >>>>>>> GitWork

**Step 12**: Use 3-way merge tool to resolve the conflict

git mergetool --tool=p4merge

Manually edit to keep the desired content. For example:

<greeting>Hello from both branches</greeting>  
 <note>This is an update in GitWork branch</note>

**Step 13**: Commit the changes after conflict resolution

git add hello.xml  
 git commit -m"Resolve merge conflict in hello.xml"

**Step 14**: Observe git status and add backup file to .gitignore

git status  
 echo"\*.orig">> .gitignore  
 git add .gitignore  
 git commit -m"Add backup files to .gitignore"

**Step 15**: List out all available branches

git branch

**Step 16**: Delete the merged branch

git branch -d GitWork

**Step 17**: Observe the final log

git log --oneline--graph--decorate

**Output:**  
  






