# Akshat Kumar (2310990271)

**SCM (Assignment 3)**

**Dated: 27 March 2024**

**Semester: 2**

**Group: 4**

**Date of Submission: 29 March 2024**

1. What is merge conflict? How to merge and Resolve conflicts created due to own activity. Add screenshots
2. How do we reset and revert a git commit. Add screenshots

# SOLUTIONS

**Ans1.** Git merge conflicts

Version control systems are all about managing contributions between multiple distributed authors (usually developers). Sometimes multiple developers may try to edit the same content. If Developer A tries to edit code that Developer B is editing a conflict may occur. To alleviate the occurrence of conflicts developers will work in separate [isolated branches.](https://www.atlassian.com/git/tutorials/using-branches) The git merge command's primary responsibility is to combine separate branches and resolve any conflicting edits.

Contribute to Managing Project files:

Git will notify you of the conflict when you attempt to merge branches. You'll see a message indicating which files have conflicts

**1.Identify the Conflict:**

**2.Open the Conflicting File(s):** Use a text editor or an integrated development environment (IDE) to open the conflicting file(s). Git will mark the conflicting sections in the file with special markers like

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | **<<<<<<<** | , | **=======** | , and **>>** | **>>>>>**. |  |
|  | **3.** | **Resolve the Conflict:** Manually edit the conflicting sections to resolve the differences | | | | | | |
| between the two versions. Remove the conflict markers and any unnecessary code.   1. **Complete the Merge:** Finally, commit the changes with **git commit**, and Git will finalize the merge 2. **Mark as Resolved:** After resolving the conflicts, stage the modified files using **git add**. | | | | | | | | |

**Ans 2.** Resetting and reverting a Git commit are two different approaches to undoing changes in a Git repository.

Resetting a Git Commit:

Resetting a Git commit involves moving the HEAD pointer and branch references to a

different commit, effectively undoing the changes introduced by the commit you're resetting to. There are different types of resets:

Soft Reset: This option keeps the changes made in the commit but moves them back to the staging area, allowing you to make further modifications before committing again.

Mixed Reset: This option moves the changes made in the commit back to the working

directory, but it doesn't delete the commit itself. This allows you to stage and commit the changes again.

Hard Reset: This option completely removes the changes made in the commit and resets your working directory and staging area to match the state of the commit you're resetting to. This can be useful if you want to completely discard the changes.

Reverting a Git Commit:

Reverting a Git commit involves creating a new commit that undoes the changes introduced by the commit you want to revert. This approach keeps a record of the reverted changes in your Git history, which can be useful for maintaining a clear audit trail.

To revert a commit, you use the git revert command followed by the hash of the commit you want to revert to. Git then creates a new commit that undoes the changes introduced by the

specified commit.

Each approach has its use cases, and the choice between resetting and reverting depends on your specific requirements and workflow. It's important to understand the implications of each approach, especially when working in a collaborative environment, to ensure the

integrity of your Git history and the stability of your codebase.