# Assignment

**What is Git merging and why is it important in collaborative software development?**

Git merging is the process of combining changes from different branches into a single branch. It is important in collaborative software development because it allows multiple developers to work on separate branches concurrently, and then merge their changes into a common branch, ensuring a unified and coherent codebase. Merging facilitates collaboration, parallel development, and the integration of new features or bug fixes from different contributors.

**Explain the difference between Git merge and Git rebase.**

Git merge combines changes from different branches into a new commit, preserving the commit history of both branches. Git rebase, on the other hand, moves or combines commits from one branch onto another, resulting in a linear history. While merge retains original commits, rebase provides a cleaner, more linear history but should be used cautiously in shared branches to avoid rewriting commit history.

**How do you resolve a merge conflict in Git? Provide an example.**

To resolve a merge conflict in Git:

1) Identify conflicted files using git status.

2) Open the conflicted file(s) in a text editor.

3) Manually resolve conflicts, removing markers (<<<<<<<, =======, >>>>>>>).

4) Add the resolved files using git add.

5) Complete the merge with git merge --continue or git commit.

Example:

```
<<<<<<< HEAD
Current branch changes
=======
Incoming branch changes
>>>>>>> branch_name
```

Manually edit to keep the desired changes, remove markers, then git add and commit.

**What is the 'ours' merge strategy in Git and in what scenario it can be useful?**

The 'ours' merge strategy in Git automatically resolves conflicts in favor of the current branch, discarding changes from the incoming branch. It can be useful in scenarios where you want to merge branches but prioritize the changes from the current branch over the incoming branch, for example, when incorporating stable or critical changes from one branch into another.

**In Git, describe the purpose and usage of the 'squash merge'.**

The 'squash merge' in Git combines all changes from a feature branch into a single, new commit on the target branch, providing a cleaner commit history. It is used to condense multiple commits into a more readable and cohesive form before merging, typically in pull requests, to maintain a more streamlined and meaningful commit history.

**Explain the concept of 'fast-forward merge' in Git with an example.**

A fast-forward merge in Git occurs when the target branch has not diverged from the source branch, allowing Git to move the branch pointer forward without creating a new commit. Example:

```
# Starting with a feature branch
git checkout feature_branch

# Making changes and committing
git commit -m "Feature commit 1"
git commit -m "Feature commit 2"

# Moving to the main branch
git checkout main

# Performing a fast-forward merge
git merge feature_branch
```

If main hasn't changed since creating feature_branch, Git performs a fast-forward merge, updating main to point to the latest commit on feature_branch.

**What is 'recursive merge' in Git and how does it differ from 'resolve merge'?**

In Git, a 'recursive merge' is a default merge strategy that handles complex merge scenarios. It automatically resolves conflicts when possible but may require manual intervention.

On the other hand, 'resolve merge' isn't a standard term in Git. If you meant 'resolve conflicts during a merge,' it implies manually addressing conflicts using a text editor or a merge tool after Git attempts an automatic merge. 'Recursive merge' may encompass both automatic resolution and manual conflict resolution.

**Can you explain the advantages and disadvantages of using 'rebase' instead of 'merge' in Git?**

Advantages of 'rebase':

1. Creates a linear commit history, making it cleaner and easier to understand.
2. Reduces unnecessary merge commits, providing a more streamlined branch structure.
3. Allows for a more organized history when integrating changes from a feature branch into a main branch.

Disadvantages of 'rebase':

1. Rewrites commit history, which can be problematic in shared branches, potentially causing conflicts for collaborators.
2. Should be used cautiously in public or shared branches to avoid disrupting collaboration and causing confusion.
3. May lead to loss of context if not used appropriately, as original commit IDs change during the rebase process.

**How does Git handle conflicts during the merging process?**

When conflicts arise during the merging process in Git:

1. Git marks the conflicted files.
2. Developers must manually resolve conflicts by editing the files.
3. After resolving conflicts, developers add the files with git add .
4. Finally, they complete the merge with git merge --continue or git commit.

**Describe the steps involved in performing a three-way merge in Git.**

Steps in performing a three-way merge in Git:

1. Identify the common ancestor, source branch, and target branch.
2. Git automatically combines changes from both branches and marks conflicts if any.
3. Manually resolve conflicts in conflicted files.
4. Add the resolved files with git add .
5. Complete the merge with git merge --continue or git commit.

**When should you avoid using 'rebase' in Git merging?**

Avoid using 'rebase' in Git merging when the branch is shared with others. Rewriting commit history with 'rebase' can cause conflicts for collaborators who have pulled the changes, leading to confusion and making it challenging to synchronize work. Use 'merge' instead for shared branches to maintain a consistent and collaborative history.