**Case Study: Managing a Collaborative Project with Git Bash and GitHub**

**Scenario:**

A software development team is working on a web application project. The team consists of three developers: Alice, Bob, and Charlie. They need to collaborate efficiently using Git and GitHub. They have to:

1. Set up a shared repository.
2. Work on different features simultaneously.
3. Handle conflicts effectively.
4. Maintain a clean commit history.

**Solution:**

**Step 1: Setting Up the Repository**

1. Alice (team lead) creates a repository on GitHub named team-project.
2. She initializes it with a README.md file.
3. Alice adds Bob and Charlie as collaborators via GitHub settings.

**Step 2: Cloning the Repository**

Each developer runs the following command in Git Bash to clone the repository:

$ git clone https://github.com/username/team-project.git

**Step 3: Creating Feature Branches**

Developers create separate branches for their tasks to avoid conflicts in the main branch.

$ git checkout -b feature-1 # Alice

$ git checkout -b feature-2 # Bob

$ git checkout -b feature-3 # Charlie

**Step 4: Implementing Features and Committing Changes**

Each developer makes changes to their respective files, adds them, and commits locally.

$ git add .

$ git commit -m "Implemented feature X"

**Step 5: Pushing Changes to GitHub**

After committing, developers push their changes to their respective branches on GitHub.

$ git push origin feature-1 # Alice

$ git push origin feature-2 # Bob

$ git push origin feature-3 # Charlie

**Step 6: Creating a Pull Request (PR)**

1. Each developer navigates to GitHub and creates a **Pull Request (PR)** from their feature branch to the main branch.
2. The team reviews each PR and comments on changes if necessary.

**Step 7: Handling Merge Conflicts**

If there are conflicts, developers follow these steps:

1. Pull the latest changes from main:
2. $ git checkout main
3. $ git pull origin main
4. Merge the feature branch into main locally:
5. $ git merge feature-1
6. Resolve conflicts manually, add changes, and commit:
7. $ git add .
8. $ git commit -m "Resolved merge conflicts"
9. Push changes back to GitHub and merge the PR.

**Step 8: Keeping the Repository Clean**

1. After merging PRs, developers delete their feature branches:
2. $ git branch -d feature-1
3. $ git push origin --delete feature-1
4. They update their local main branch to stay up-to-date:
5. $ git checkout main
6. $ git pull origin main

**Best Practices:**

* Always pull the latest changes before starting work.
* Use meaningful commit messages.
* Work in feature branches instead of main.
* Review PRs before merging.
* Resolve conflicts carefully and communicate within the team.

By following these steps, the team can efficiently collaborate, manage their codebase, and avoid conflicts using Git Bash and GitHub.