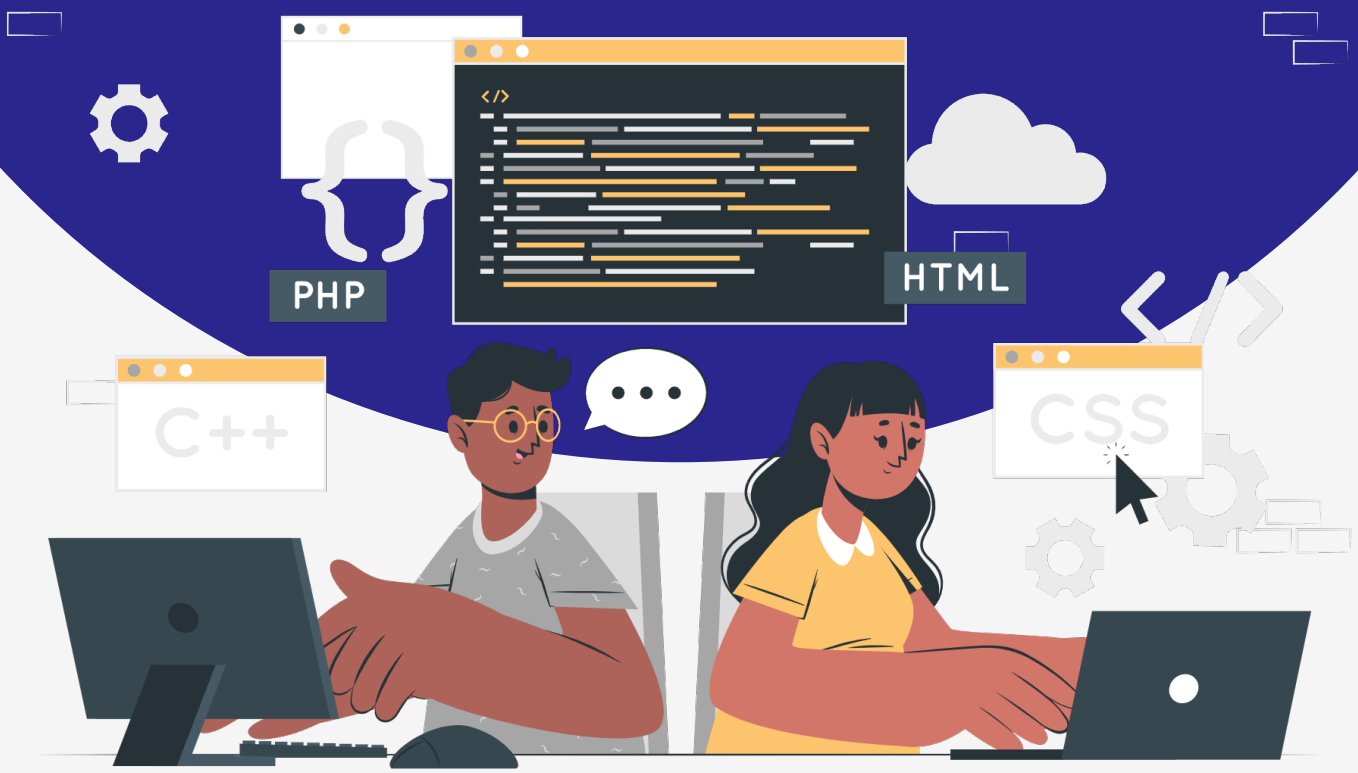
Lesson Plan

Branching and Merging in Git



PW SKILLS

Git Workflows and Lifecycle

Introduction
Git is a powerful version control system that allows multiple developers to work on a project simultaneously. Two of the most important features in Git are branching and merging. Branching allows developers to create isolated environments for their work, and merging integrates changes from different branches into a single branch. This document will cover the concepts, workflows, and examples of branching and merging in Git.

1. Branching

1.1 What is Branching?

Branching in Git allows you to create a separate line of development. By default, Git creates a main branch (formerly master). When you start working on a new feature or bug fix, you create a new branch. This branch is a copy of the main branch at the point you create it.

1.2 Creating Branches

You can create a branch using the git branch command

```
git branch feature-branch
```

You can also create and switch to a new branch in one command using git checkout -b.

```
git checkout -b feature-branch
```

1.3 Switching Between Branches

To switch between branches, use the git checkout command.

```
git checkout main
git checkout feature-branch
```

1.4 Viewing Branches

To see a list of all branches in your repository, use the git branch command.

```
git branch
```

1.5 Deleting Branches

You can delete a branch using the -d flag.

```
git branch -d feature-branch
```

If the branch has unmerged changes and you still want to delete it, use the -D flag.

```
git branch -D feature-branch
```

2. Merging

2.1 What is Merging?
Merging is the process of combining the changes from one branch into another. This is essential for integrating features and fixes developed in separate branches into the main branch.

2.2 Fast-forward Merging

```
git checkout main
git merge feature-branch
```

If there have been no changes in the target branch since the branch you are merging was created, Git performs a fast-forward merge. The branch pointer simply moves forward to the most recent commit.

2.3 Three-way Merge
If there have been changes in both branches, Git performs a three-way merge. It considers the changes from both branches and the common ancestor to create a new commit that combines all changes.

```
git checkout main
git merge feature-branch
```

2.4 Merge Conflicts
When the same part of the same file is modified in both branches, Git cannot automatically merge the changes and a conflict occurs. You must resolve conflicts manually by editing the conflicted files.

Example of a Conflict:

```
<<<<<<< HEAD
Current change in the main branch.
=======
Change from the feature branch.
>>>>>>> feature-branch
```

Resolve the conflict, stage the changes, and commit.

```
git add conflicted-file
git commit -m "Resolve merge conflict"
```

3. Branching Strategies

3.1 Gitflow Workflow

Gitflow is a popular branching strategy that uses multiple branches to manage different stages of development and release.

- **main:** Production-ready code.
- **develop:** Latest development changes.
- **feature/*:** Individual features.
- **release/*:** Release preparation.
- **hotfix/*:** Critical fixes.

Example:

```
git checkout -b develop main
git checkout -b feature/awesome-feature develop
git checkout develop
git merge feature/awesome-feature
git checkout -b release/1.0.0 develop
git checkout main
git merge release/1.0.0
git tag -a 1.0.0 -m "Release 1.0.0"
git checkout develop
git merge release/1.0.0
```

3.2 GitHub Flow

GitHub Flow is a simpler branching strategy used in many open-source projects.

- **main:** Production-ready code.
- **feature/*:** Individual features or fixes.

Example:

```
git checkout -b feature/awesome-feature main
git push -u origin feature/awesome-feature
# Create a pull request on GitHub
# Merge the pull request on GitHub
git pull origin main
```

4. Advanced Branching and Merging

4.1 Rebasing

Rebasing is an alternative to merging that can result in a cleaner project history. It moves or combines a sequence of commits to a new base commit.

Example:

```
git checkout feature-branch
git rebase main
git checkout main
git merge feature-branch
```

4.2 Cherry-picking

Cherry-picking allows you to apply the changes from a specific commit to another branch.

Example:

```
git checkout main
git cherry-pick commit-hash
```

Conclusion

Branching and merging are fundamental concepts in Git that enable effective collaboration and project management. By understanding and utilizing these features, development teams can work more efficiently and maintain a clear project history.