VTU 2025 Scheme

Project Management (with Git)

1BCSL307A – 3rd Semester

EXPERIMENT – 06

Collaboration and Remote Repositories

Merge a Feature Branch with a Custom Merge Commit Message

Aim

To merge the `feature-branch` into the `master` branch and create a merge commit with a custom commit message.

Objectives

After completing this experiment, the student will be able to:

- Understand the concept of branch merging in Git.
- Create and manage a feature branch.
- Make independent changes in a feature branch.
- Switch between `feature-branch` and `master`.
- Merge a feature branch into the master branch.
- Create a merge commit with a custom message.
- Verify the merge using Git log commands.

Software Requirements

- Git
- Windows Command Prompt / PowerShell / Git Bash
- Text Editor such as Notepad or Visual Studio Code

Theory

Merging is the process of combining changes from one Git branch into another branch.

Developers normally use separate branches to develop new features. After a feature is completed and tested, the changes from the feature branch can be merged into the main development branch.

In this experiment, changes made in `feature-branch` are merged into `master`.

A custom merge commit message is also provided during the merge.

What is a Merge Commit?

A **merge commit** is a commit created when Git combines different development histories.

A merge commit records the point where the histories of two branches are combined.

A custom message can be assigned to the merge commit to clearly describe the purpose of the merge.

Why Use a Custom Merge Message?

- Clearly explains why the branches were merged.
- Makes the Git history easier to understand.
- Helps identify completed features.
- Improves project documentation.
- Makes collaborative development easier to track.

Important Commands

Command	Purpose
`git branch`	Displays local branches.
`git branch feature-branch`	Creates a new branch named `feature-branch`.
`git switch feature-branch`	Switches to the feature branch.
`git switch master`	Switches to the master branch.
`git merge --no-ff feature-branch -m "..."`	Merges the feature branch and creates a merge commit with a custom message.
`git log --oneline --graph --all`	Displays the commit and branch history graphically.

Merge Concept

The following diagram shows how a feature branch is merged into the master branch.

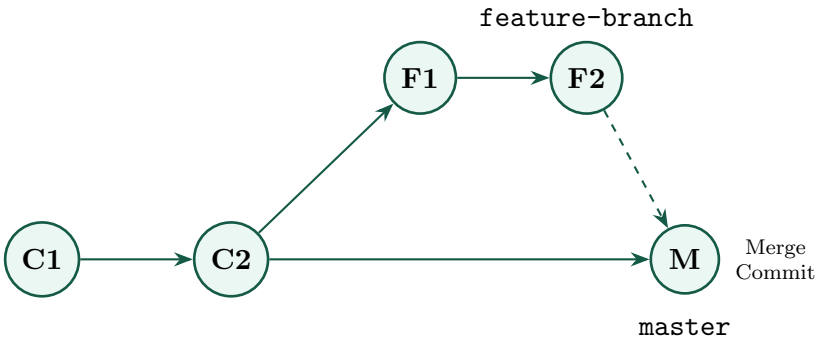


Diagram Explanation:

- C1 and C2 represent commits in the master branch.
- The `feature-branch` is created after C2.
- F1 and F2 represent commits made in the feature branch.
- The feature branch is merged back into master.
- M represents the merge commit.

Procedure

Step 1: Open Command Prompt, PowerShell, or Git Bash.

Step 2: Create a new directory for Experiment 06.

Step 3: Initialize the directory as a Git repository.

Step 4: Create an initial project file.

Step 5: Add and commit the initial file.

Step 6: Ensure that the main branch is named `master`.

Step 7: Create a new branch named `feature-branch`.

Step 8: Switch to the feature branch.

Step 9: Modify the project file.

Step 10: Add and commit the changes in the feature branch.

Step 11: Switch back to the master branch.

Step 12: Merge the feature branch into master using a custom merge commit message.

Step 13: Display the commit history.

Step 14: Verify the successful merge.

Execution

Step 1: Create Experiment Directory

Execute:

```
mkdir git-experiment-06
cd git-experiment-06
```

Explanation:
The first command creates a new directory named `git-experiment-06`.
The second command moves into the directory.

Step 2: Initialize Git Repository

Execute:

```
git init
```

Sample Output:

```
Initialized empty Git repository in
.../git-experiment-06/.git/
```

The current directory is now a Git repository.

Step 3: Create Initial Project File

Execute:

```
echo Project Management with Git > project.txt
```

The command creates a file named `project.txt`.

Step 4: Add File to Staging Area

Execute:

```
git add project.txt
```

The file is added to the staging area.

Step 5: Create Initial Commit

Execute:

```
git commit -m "Initial commit"
```

The initial version of the project is stored in the local repository.
Note:
If Git asks for user identity, configure it using:

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

Then execute the commit command again.

Step 6: Set Main Branch as Master

Execute:

```
git branch -M master
```

This ensures that the main branch is named `master`.

Step 7: Create Feature Branch

Execute:

```
git branch feature-branch
```

This creates a new branch named `feature-branch`.
Verify the branches:

```
git branch
```

Sample Output:

```
  feature-branch
* master
```

Step 8: Switch to Feature Branch

Execute:

```
git switch feature-branch
```

Sample Output:

```
Switched to branch 'feature-branch'
```

The feature branch is now active.

Step 9: Modify the Project

Execute:

```
echo New feature added to project >> project.txt
```

The file now contains:

```
Project Management with Git
New feature added to project
```

Step 10: Stage Feature Changes

Execute:

```
git add project.txt
```

The modified file is added to the staging area.

Step 11: Commit Feature Changes

Execute:

```
git commit -m "Add new feature"
```

Explanation:
The feature changes are now committed inside `feature-branch`.

Step 12: Switch Back to Master

Execute:

```
git switch master
```

Sample Output:

```
Switched to branch 'master'
```

The master branch is now active.

Step 13: Merge with Custom Commit Message

Execute:

```
git merge --no-ff feature-branch -m "Merge feature-branch into master"
```

Explanation:
The command merges `feature-branch` into `master`.
The option:

```
--no-ff
```

forces Git to create a merge commit even when a fast-forward merge would otherwise be possible.
The option:

```
-m "Merge feature-branch into master"
```

provides a custom message for the merge commit.

Step 14: Check Repository Status

Execute:

```
git status
```

Sample Output:

```
On branch master
nothing to commit, working tree clean
```

This confirms that the merge operation has completed and the working tree is clean.

Step 15: View Commit History

Execute:

```
git log --oneline --graph --all
```

Sample Output:

```
*   a1b2c3d Merge feature-branch into master
|\
| * d4e5f6g Add new feature
|/
* h7i8j9k Initial commit
```

Explanation:
The output displays the branch history and the merge commit.
The exact commit IDs will be different on each computer.

Step 16: View Latest Commit Details

Execute:

```
git log -1
```

The latest commit should display the custom merge commit message:

```
Merge feature-branch into master
```

This verifies that the custom merge commit was successfully created.

Complete Command Sequence

The complete command sequence for Experiment 06 is:

```
mkdir git-experiment-06
cd git-experiment-06

git init

echo Project Management with Git > project.txt

git add project.txt

git commit -m "Initial commit"

git branch -M master
```

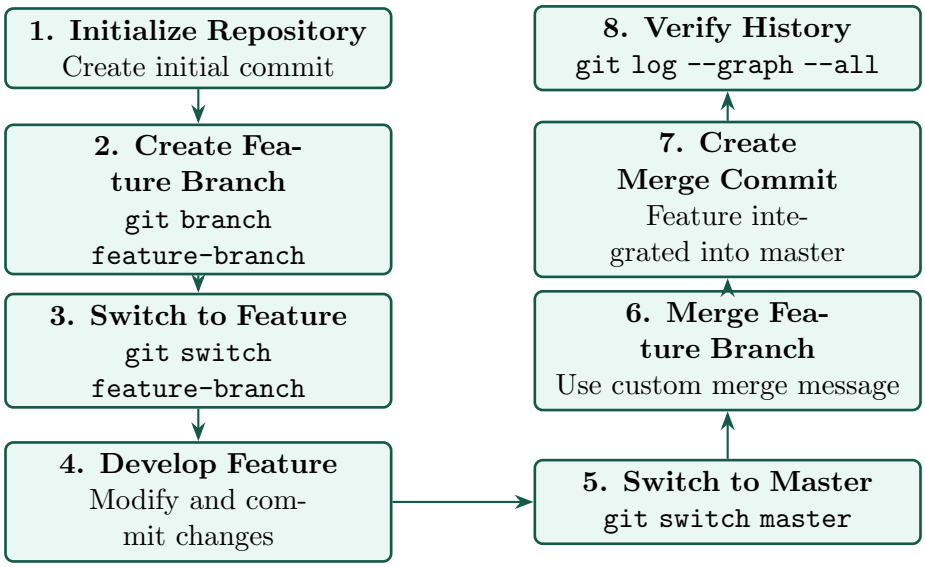
```
git branch feature-branch

git branch

git switch feature-branch

echo New feature added to project >> project.txt

git add project.txt

git commit -m "Add new feature"

git switch master

git merge --no-ff feature-branch -m "Merge feature-branch into master"

git status

git log --oneline --graph --all

git log -1
```

Command Summary

Command	Description
`git branch feature-branch`	Creates the feature branch.
`git switch feature-branch`	Switches to the feature branch.
`git add project.txt`	Adds the modified file to the staging area.
`git commit -m "..."`	Commits the staged changes.
`git switch master`	Switches to the master branch.
`git merge --no-ff ...`	Merges the feature branch and forces creation of a merge commit.
`git log --oneline --graph --all`	Displays graphical branch and merge history.

Experiment Workflow

The complete workflow of Experiment 06 is shown below.



Workflow Explanation:

1. Initialize the repository and create the initial commit.
2. Create `feature-branch`.
3. Switch to the feature branch.
4. Make and commit the feature changes.
5. Switch back to `master`.
6. Merge `feature-branch` using a custom merge message.
7. Git creates a merge commit.
8. Verify the history using the Git log.

Fast-Forward Merge vs No-Fast-Forward Merge

Feature	Fast-Forward	No-Fast-Forward
Merge Commit	May not be created	Merge commit is created
Command	`git merge branch`	`git merge --no-ff branch`
History	More linear	Clearly shows branch integration
Custom Merge Message	Not useful if no merge commit is created	Can be attached to the merge commit

Result

Thus, the `feature-branch` was successfully merged into the `master` branch. A merge commit was created with the custom message `Merge feature-branch into master`, and the merge was verified using the Git commit history.

Viva Questions and Answers

1. **What is merging in Git?**

   Merging is the process of combining changes from one branch into another branch.

2. **What is a merge commit?**

   A merge commit records the integration of different branch histories.

3. **Which command is normally used to merge a branch?**

   ```
   git merge branch-name
   ```

4. **How do you merge feature-branch into master?**

   First switch to master:

   ```
   git switch master
   ```

   Then execute:

   ```
   git merge feature-branch
   ```

5. **How can we provide a custom merge commit message?**

   A custom merge commit message can be supplied using the `-m` option.

   For example:

   ```
   git merge --no-ff feature-branch -m "Merge feature-branch into master"
   ```

6. **What is the purpose of –no-ff?**

   The `--no-ff` option forces Git to create a merge commit even when a fast-forward merge is possible.

7. **What is a fast-forward merge?**

   A fast-forward merge occurs when the target branch has no independent commits after the feature branch was created. Git can move the branch pointer forward without creating a merge commit.

8. **Why is –no-ff useful in this experiment?**

   It ensures that an actual merge commit is created so that a custom merge commit message can be recorded.

9. **Which branch should be active before merging feature-branch into master?**

   The `master` branch should be active.

10. **How can we check the current branch?**

```
git branch
```

11. **How can we view the graphical commit history?**

```
git log --oneline --graph --all
```

12. **What is a merge conflict?**

    A merge conflict occurs when Git cannot automatically combine different changes made to the same part of a file.

13. **Can a custom message make Git history easier to understand?**

    Yes. A clear merge message helps developers understand why a branch was integrated into another branch.

Conclusion

In this experiment, branch merging in Git was successfully demonstrated. A feature branch was created and used for independent development. After completing the feature, the repository was switched to the `master` branch. The feature branch was then merged using a custom merge commit message, and the final repository history was verified.