VTU 2025 Scheme

Project Management (with Git)

1BCSL307A – 3rd Semester

EXPERIMENT – 02

Creating and Managing Branches

Aim

To create a new branch named `feature-branch`, switch to the `master` branch, and merge the `feature-branch` into the `master` branch using Git commands.

Objectives

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

- Understand the concept of branches in Git.
- Create a new branch in a Git repository.
- Display the available branches.
- Switch between different branches.
- Make changes independently in a feature branch.
- Merge changes from one branch into another branch.
- Verify the final branch and commit history.

Software Requirements

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

Theory

A **branch** in Git represents an independent line of development.

Branches allow developers to work on new features, bug fixes, or experiments without directly affecting the main version of the project.

For example, a developer can create a branch called `feature-branch`. New development can be performed inside this branch. After the work is completed and verified, the changes can be merged into the `master` branch.

Advantages of Git Branches

- Allows independent development of new features.
- Protects the stable version of the project.
- Allows multiple developers to work simultaneously.
- Makes testing of new features easier.
- Helps organize different development tasks.
- Supports collaborative software development.
- Makes project maintenance easier.

Important Git Branch Commands

Command	Purpose
`git branch`	Displays all 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 feature-branch`	Merges `feature-branch` into the current branch.
`git log --oneline --graph --all`	Displays the branch and commit history.

Branching Concept

A branch creates a separate development path. In this experiment, `feature-branch` is created from the main development branch. Changes can be made independently and later merged into `master`.

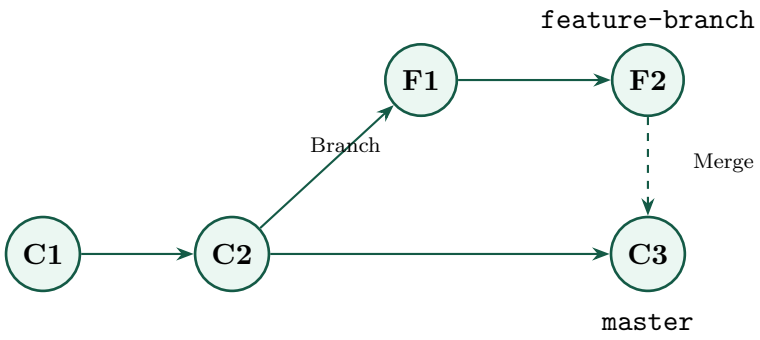


Diagram Explanation:

- **C1 and C2** represent commits in the `master` branch.
- The `feature-branch` is created after commit C2.
- **F1 and F2** represent commits made in the `feature-branch`.
- The feature changes are later merged back into the `master` branch.
- **C3** represents the master development line after integration.

Procedure

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

Step 2: Create a new directory for Experiment 02.

Step 3: Move into the newly created directory.

Step 4: Initialize the directory as a Git repository.

Step 5: Create an initial project file.

Step 6: Add the file to the staging area.

Step 7: Commit the initial file.

Step 8: Ensure that the main branch is named `master`.

Step 9: Create a new branch named `feature-branch`.

Step 10: Display all available branches.

Step 11: Switch to `feature-branch`.

Step 12: Modify the project file.

Step 13: Add and commit the changes.

Step 14: Switch back to the `master` branch.

Step 15: Merge `feature-branch` into `master`.

Step 16: Verify the final branch and commit history.

Execution

Step 1: Create a New Directory

Execute:

```
mkdir git-experiment-02
cd git-experiment-02
```

Explanation:

The first command creates a directory named `git-experiment-02`.

The second command changes the current working directory to the newly created directory.

Step 2: Initialize Git Repository

Execute:

```
git init
```

Sample Output:

```
Initialized empty Git repository in
.../git-experiment-02/.git/
```

Explanation:

The `git init` command initializes the current directory as a new local Git repository.

Step 3: Create an Initial File

For Windows Command Prompt, execute:

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

Explanation:

This creates a new file named `project.txt` containing the text:

```
Project Management with Git
```

Step 4: Add File to Staging Area

Execute:

```
git add project.txt
```

Explanation:

The `git add` command adds the file to the staging area.

Step 5: Create Initial Commit

Execute:

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

Explanation:
The command saves the staged file as the first commit in the local repository.
Note:
If Git asks for user identity, execute:

```
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
```

Explanation:
Some Git installations use `main` as the default branch name.
This command ensures that the main branch is named `master` for this experiment.

Step 7: Display Current Branch

Execute:

```
git branch
```

Sample Output:

```
* master
```

The asterisk (*) indicates the currently active branch.

Step 8: Create Feature Branch

Execute:

```
git branch feature-branch
```

Explanation:
This command creates a new branch named `feature-branch`.
The command only creates the branch. It does not automatically switch to the newly created branch.

Step 9: Display All Branches

Execute:

```
git branch
```

Sample Output:

```
  feature-branch
* master
```

Two branches are now available.
The asterisk indicates that `master` is currently active.

Step 10: Switch to Feature Branch

Execute:

```
git switch feature-branch
```

Sample Output:

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

The active branch is now `feature-branch`.
For older Git versions, the following command can also be used:

```
git checkout feature-branch
```

Step 11: Modify the Project File

Execute:

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

Explanation:
The `>>` operator appends new content to the existing file.
The file now contains:

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

Step 12: Check Repository Status

Execute:

```
git status
```

The output shows that `project.txt` has been modified.

Step 13: Stage the Modified File

Execute:

```
git add project.txt
```

The modified file is added to the staging area.

Step 14: Commit Feature Changes

Execute:

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

Explanation:
The changes are committed to the `feature-branch`.

Step 15: Switch Back to Master

Execute:

```
git switch master
```

Sample Output:

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

The active branch is now `master`.

Step 16: Merge Feature Branch into Master

Execute:

```
git merge feature-branch
```

Possible Output:

```
Updating ...
Fast-forward
 project.txt | 1 +
 1 file changed, 1 insertion(+)
```

Explanation:
The changes from `feature-branch` are now included in the `master` branch.
In this simple experiment, Git may perform a **fast-forward merge** because the master branch has not received another commit after the feature branch was created.

Step 17: Verify the Commit History

Execute:

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

Sample Output:

```
* a2b3c4d Add new feature
* a1b2c3d Initial commit
```

Note:
The commit IDs displayed on your computer will be different.

Step 18: Verify Current Branch

Execute:

```
git branch
```

Sample Output:

```
  feature-branch
* master
```

This confirms that the active branch is `master`.

Complete Command Sequence

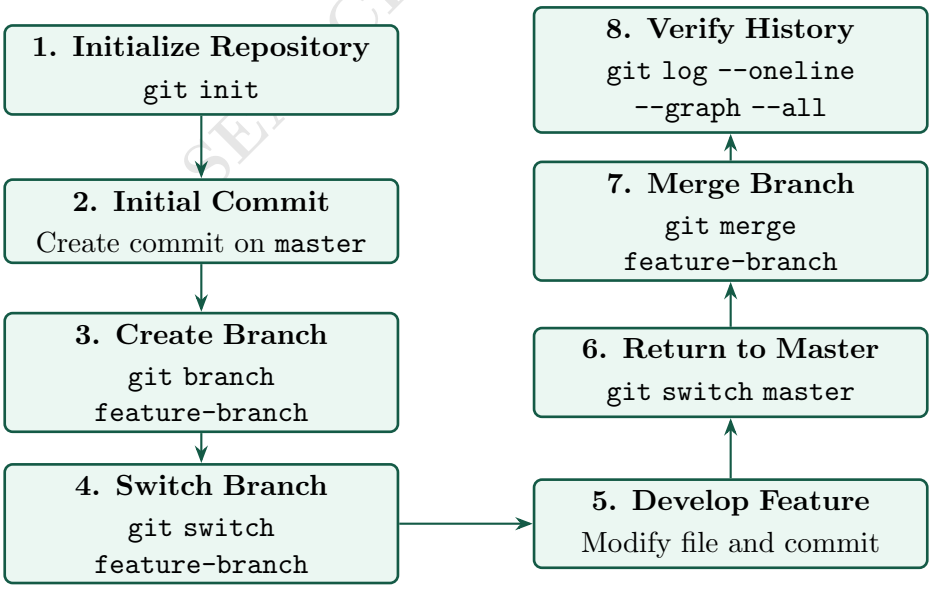
```
mkdir git-experiment-02
cd git-experiment-02

git init

echo Project Management with Git > project.txt

git add project.txt
git commit -m "Initial commit"

git branch -M master

git branch

git branch feature-branch

git branch

git switch feature-branch

echo New feature added >> project.txt

git status

git add project.txt

git commit -m "Add new feature"

git switch master

git merge feature-branch

git log --oneline --graph --all

git branch
```

Command Summary

Command	Description
`git init`	Initializes a Git repository.
`git branch`	Displays the available local branches.
`git branch feature-branch`	Creates `feature-branch`.
`git switch feature-branch`	Switches to the feature branch.
`git add project.txt`	Stages the modified project file.
`git commit -m "..."`	Creates a new commit.
`git switch master`	Switches back to the master branch.
`git merge feature-branch`	Merges the feature branch into master.
`git log --oneline --graph --all`	Displays repository history in graphical form.

Experiment Workflow

The complete workflow of Experiment 02 is shown below.



Workflow Explanation:

1. Initialize the local Git repository.
2. Create the initial commit on the `master` branch.

3. Create `feature-branch`.
4. Switch to the newly created feature branch.
5. Modify the project and commit the changes.
6. Switch back to the `master` branch.
7. Merge `feature-branch` into `master`.
8. Verify the final commit history.

Result

Thus, a new branch named `feature-branch` was successfully created and managed. Changes were committed in the feature branch. The repository was then switched to the `master` branch, and `feature-branch` was successfully merged into `master`.

Viva Questions and Answers

1. **What is a branch in Git?**

   A branch is an independent line of development in a Git repository.

2. **Why are branches used in Git?**

   Branches allow developers to work on new features without directly affecting the main version of the project.

3. **Which command displays local branches?**

```
git branch
```

4. **How do you create a new Git branch?**

   The syntax is:

```
git branch branch-name
```

5. **How do you create feature-branch?**

```
git branch feature-branch
```

6. **How do you switch to feature-branch?**

```
git switch feature-branch
```

7. **What does the asterisk (*) in git branch indicate?**

   The asterisk indicates the currently active branch.

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

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

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

   First switch to the master branch:

   ```
   git switch master
   ```

   Then merge:

   ```
   git merge feature-branch
   ```

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

    A fast-forward merge occurs when the target branch has no additional commits after the feature branch was created. Git moves the branch pointer forward to include the feature commits.

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

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

12. **Does git branch feature-branch switch to the branch?**

    No. It only creates the branch.

    To switch to it, use:

    ```
    git switch feature-branch
    ```

13. **What is the difference between git branch and git switch?**

    `git branch` is mainly used to create, list, or manage branches, while `git switch` is used to change the currently active branch.

14. **Which command shows the graphical commit history?**

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

Conclusion

In this experiment, Git branch management was successfully demonstrated. A new `feature-branch` was created from the main development branch. Changes were made and committed independently in the feature branch. Finally, the repository was switched back to `master`, and the feature branch was successfully merged into it.