

VTU 2025 Scheme
Project Management (with Git)
1BCSL307A – 3rd Semester

EXPERIMENT – 08
Tags, Releases, and Advanced Git Operations
Cherry-Picking a Range of Commits

Aim

To cherry-pick a range of commits from the **source-branch** and apply those commits to another branch using Git commands.

Objectives

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

- Understand the concept of cherry-picking in Git.
- Create and manage multiple Git branches.
- Create multiple commits in a source branch.
- Identify commit IDs using Git log.
- Cherry-pick selected commits.
- Cherry-pick a range of consecutive commits.
- Apply selected changes without merging the complete branch.
- Verify the final commit history.

Software Requirements

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

Theory

Cherry-pick is a Git operation used to apply one or more specific commits from one branch to another branch.

Normally, merging combines the development history of branches.

However, sometimes a developer may need only a few specific commits from another branch instead of merging the complete branch.

In such situations, the **git cherry-pick** command can be used.

What is Cherry-Picking?

Cherry-picking means selecting a particular commit from one branch and applying the changes introduced by that commit to the currently active branch.

For example:

```
git cherry-pick <commit-id>
```

This applies the selected commit to the current branch.

Cherry-Picking a Range of Commits

A range of consecutive commits can be cherry-picked using:

```
git cherry-pick <start-commit>^..<end-commit>
```

The caret symbol after the starting commit ensures that the starting commit itself is also included in the range.

For example:

```
git cherry-pick abc123^..xyz789
```

This applies all consecutive commits from `abc123` through `xyz789`, inclusive.

Why is Cherry-Pick Used?

- To copy selected commits between branches.
- To avoid merging an entire branch.
- To apply a specific bug fix to another branch.
- To transfer selected features.
- To reuse useful changes from another development branch.
- To manage selected changes in a controlled manner.

Cherry-Pick Concept

The following diagram shows selected commits being copied from `source-branch` to `master`.

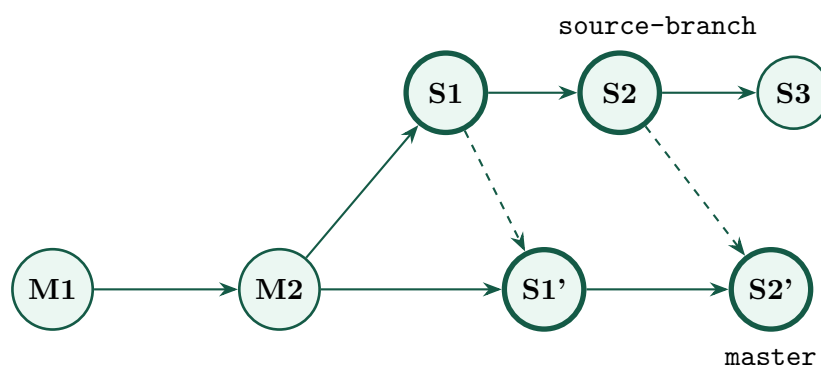


Diagram Explanation:

- M1 and M2 represent commits in the master branch.
- S1, S2, and S3 represent commits in **source-branch**.
- S1 and S2 are selected for cherry-picking.
- Git applies the changes of S1 and S2 to master.
- Git creates new commits S1' and S2' on master.
- S3 is not copied because it is outside the selected range.

Important Commands

Command	Purpose
<code>git branch source-branch</code>	Creates the source branch.
<code>git switch source-branch</code>	Switches to the source branch.
<code>git log --oneline</code>	Displays commit IDs and messages.
<code>git cherry-pick <commit-id></code>	Applies one selected commit.
<code>git cherry-pick A^..B</code>	Applies consecutive commits from A through B.
<code>git cherry-pick --continue</code>	Continues cherry-picking after resolving a conflict.
<code>git cherry-pick --abort</code>	Cancels an active cherry-pick operation.

Procedure

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

Step 2: Create a new directory for Experiment 08.

Step 3: Initialize the directory as a Git repository.

Step 4: Create and commit an initial project file.

Step 5: Ensure that the main branch is named **master**.

Step 6: Create a new branch named **source-branch**.

Step 7: Switch to **source-branch**.

Step 8: Create the first source-branch commit.

Step 9: Create the second source-branch commit.

Step 10: Create the third source-branch commit.

Step 11: Display the source-branch commit history.

Step 12: Note the IDs of the first and second source commits.

Step 13: Switch back to `master`.

Step 14: Cherry-pick the required range of commits.

Step 15: Display the final commit history.

Step 16: Verify that the selected commits were applied successfully.

Execution

Step 1: Create Experiment Directory

Execute:

```
mkdir git-experiment-08  
cd git-experiment-08
```

This creates and opens the Experiment 08 directory.

Step 2: Initialize Repository

Execute:

```
git init
```

Sample Output:

```
Initialized empty Git repository in  
.../git-experiment-08/.git/
```

Step 3: Create Initial File

Execute:

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

Step 4: Create Initial Commit

Execute:

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

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 5: Set Main Branch as Master

Execute:

```
git branch -M master
```

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

Step 6: Create Source Branch

Execute:

```
git branch source-branch
```

Verify:

```
git branch
```

Sample Output:

```
* master  
source-branch
```

Step 7: Switch to Source Branch

Execute:

```
git switch source-branch
```

Sample Output:

```
Switched to branch 'source-branch'
```

Step 8: Create First Source Commit

Execute:

```
echo Feature One > feature1.txt  
git add feature1.txt  
git commit -m "Add feature one"
```

This creates the first commit to be cherry-picked.

Step 9: Create Second Source Commit

Execute:

```
echo Feature Two > feature2.txt
git add feature2.txt
git commit -m "Add feature two"
```

This creates the second commit to be cherry-picked.

Step 10: Create Third Source Commit

Execute:

```
echo Feature Three > feature3.txt
git add feature3.txt
git commit -m "Add feature three"
```

This creates another commit in the source branch.

For this experiment, only the first two source commits will be cherry-picked.

Step 11: Display Commit History

Execute:

```
git log --oneline
```

Sample Output:

```
c3d4e5f Add feature three
b2c3d4e Add feature two
a1b2c3d Add feature one
9z8y7x6 Initial commit
```

Important:

The commit IDs on your computer will be different.

Suppose:

```
a1b2c3d = Add feature one
b2c3d4e = Add feature two
```

These two commits form the range that we want to cherry-pick.

Step 12: Switch Back to Master

Execute:

```
git switch master
```

Sample Output:

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

Step 13: Cherry-Pick the Commit Range

Execute:

```
git cherry-pick a1b2c3d^..b2c3d4e
```

Important:

Replace the sample commit IDs with the actual IDs displayed by:

```
git log --oneline
```

The general syntax is:

```
git cherry-pick <first-commit>^..<last-commit>
```

Explanation:

The command applies all consecutive commits starting from the first specified commit and ending with the last specified commit.

The ^ before the two dots ensures that the first commit is included.

Step 14: Verify Files

On Windows, execute:

```
dir
```

The master branch should now contain:

```
project.txt  
feature1.txt  
feature2.txt
```

The file:

```
feature3.txt
```

should not be present because the third commit was not included in the cherry-pick range.

Step 15: Verify Commit History

Execute:

```
git log --oneline
```

Sample Output:

```
e5f6g7h Add feature two  
d4e5f6g Add feature one  
9z8y7x6 Initial commit
```

Explanation:

The commit messages are similar to the original source commits, but Git creates new commits on the current branch.

Therefore, the new commit IDs may differ from the original commit IDs.

Step 16: View Complete Branch History

Execute:

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

This displays the history of both master and source-branch.

Complete Command Sequence

```
mkdir git-experiment-08
cd git-experiment-08

git init

echo Project Management with Git > project.txt

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

git branch -M master

git branch source-branch

git switch source-branch

echo Feature One > feature1.txt
git add feature1.txt
git commit -m "Add feature one"

echo Feature Two > feature2.txt
git add feature2.txt
git commit -m "Add feature two"

echo Feature Three > feature3.txt
git add feature3.txt
git commit -m "Add feature three"

git log --oneline

git switch master

git cherry-pick <first-commit>^..<second-commit>

git status

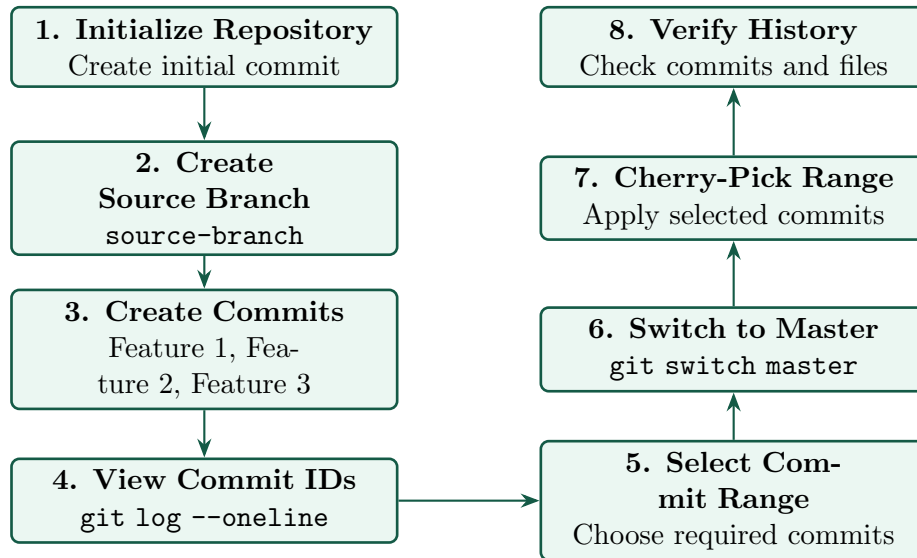
dir

git log --oneline

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

Experiment Workflow

The complete workflow is shown below.



Cherry-Pick vs Merge

Feature	Cherry-Pick	Merge
Purpose	Apply selected commits	Combine branch histories
Selection	Specific commits can be selected	Normally integrates branch changes
New Commit IDs	Cherry-picked commits normally receive new IDs	Existing commits remain part of merged history
Basic Command	<code>git cherry-pick <id></code>	<code>git merge branch-name</code>
Useful For	Selected fixes or features	Integrating complete development work

Handling Cherry-Pick Conflicts

If Git finds conflicting changes during cherry-picking, it pauses the operation.

First check the conflict:

```
git status
```

Open the conflicting file and resolve the conflict manually.

Then stage the corrected file:

```
git add <file-name>
```

Continue the cherry-pick operation:

```
git cherry-pick --continue
```

To cancel the complete cherry-pick operation:

```
git cherry-pick --abort
```

Result

Thus, a range of commits from **source-branch** was successfully cherry-picked and applied to the **master** branch. The selected commits and project files were verified using Git log and repository commands.

Viva Questions and Answers

1. What is cherry-pick in Git?

Cherry-pick is a Git operation used to apply a specific commit from one branch to another branch.

2. Which command is used to cherry-pick one commit?

```
git cherry-pick <commit-id>
```

3. How do you cherry-pick a range of consecutive commits?

```
git cherry-pick <start-commit>^..<end-commit>
```

4. Why is the caret used before the two dots?

It refers to the parent of the starting commit so that the starting commit itself is included in the selected range.

5. How can we obtain commit IDs?

```
git log --oneline
```

6. What is source-branch?

It is the branch containing the commits that we want to cherry-pick.

7. Which branch receives the cherry-picked commits?

The branch that is currently checked out receives the cherry-picked commits.

8. Does cherry-pick merge the complete source branch?

No. It applies only the selected commit or commits.

9. Does cherry-pick create new commits?

Yes. Git applies the selected changes to the current branch and creates corresponding commits there.

10. **Do cherry-picked commits always have the same commit IDs?**

No. The resulting commits generally have different IDs because they are new commits in a different history.

11. **What happens if a cherry-pick causes a conflict?**

Git pauses the operation so that the conflict can be resolved manually.

12. **Which command continues cherry-picking after resolving a conflict?**

```
git cherry-pick --continue
```

13. **Which command cancels a cherry-pick operation?**

```
git cherry-pick --abort
```

14. **What is the main difference between cherry-pick and merge?**

Cherry-pick applies selected commits, while merge is used to combine development histories between branches.

Conclusion

In this experiment, advanced Git operations using `git cherry-pick` were demonstrated. Multiple commits were created in `source-branch`, a required range of commits was identified, and those commits were applied to the `master` branch. The final repository history was then verified successfully.

SEARCH CREATORS

VTU 2025 Scheme – 3rd Semester

searchcreators.org