VTU 2025 Scheme

Project Management (with Git)

1BCSL307A – 3rd Semester

EXPERIMENT – 12

Git History Modification

Undoing Changes Introduced by a Specific Commit

Aim

To undo the changes introduced by a specific Git commit (`abc123`) using the `git revert` command and verify the resulting repository history.

Objectives

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

- Understand the purpose of undoing Git changes.
- Understand the `git revert` command.
- Identify a required commit using its commit ID.
- Undo the changes introduced by a specific commit.
- Understand that revert creates a new commit.
- Verify the repository after reverting a commit.
- Understand the difference between revert and reset.

Software Requirements

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

Theory

Sometimes a commit may introduce an unwanted change, incorrect code, or an error into a project.

Git provides the `git revert` command to undo the changes introduced by a specific commit.

The general syntax is:

```
git revert <commit-id>
```

For example:

```
git revert abc123
```

Here, `abc123` represents the ID of the commit whose changes must be reversed.

What Does git revert Do?

The `git revert` command:

- Identifies the selected commit.
- Determines the changes introduced by that commit.
- Applies the opposite changes.
- Creates a new commit containing the reversal.
- Keeps the original commit in the Git history.

Therefore, `git revert` does not normally delete the selected commit from history. Instead, it creates another commit that reverses its effect.

Example

Suppose the repository contains:

```
C1 -> C2 -> C3
```

If C2 introduced an unwanted change, reverting C2 produces:

```
C1 -> C2 -> C3 -> R
```

Here:

- C2 remains in the history.
- R is a new revert commit.
- R reverses the changes introduced by C2.

Git Revert Concept

The following diagram shows the basic working of `git revert`.

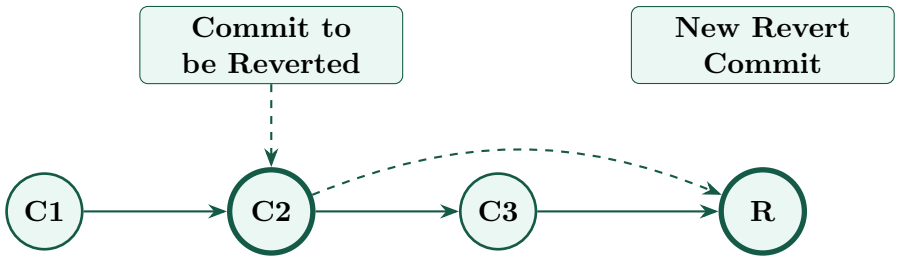


Diagram Explanation:

- C1 represents an earlier project commit.
- C2 represents the commit whose changes must be undone.
- C3 represents a later commit.
- Git does not remove C2 from the history.
- A new commit R is created.
- R reverses the changes introduced by C2.

Important Commands

Command	Purpose
`git log --oneline`	Displays compact commit history and commit IDs.
`git show <commit-id>`	Displays details of the selected commit.
`git revert <commit-id>`	Creates a new commit that reverses the selected commit.
`git revert --no-edit <commit-id>`	Reverts the commit using the automatically generated revert message.
`git status`	Displays the current repository status.
`git log --oneline --graph`	Displays the commit history in graphical compact form.

Procedure

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

Step 2: Create a directory for Experiment 12.

Step 3: Initialize a Git repository.

Step 4: Create an initial project file.

Step 5: Stage and commit the initial file.

Step 6: Introduce another change into the project.

Step 7: Commit the new change.

Step 8: Create another commit after it.

Step 9: Display the Git commit history.

Step 10: Identify the commit whose changes must be undone.

Step 11: Inspect the selected commit.

Step 12: Execute the `git revert` command.

Step 13: Verify the project files.

Step 14: Display the new Git history.

Step 15: Confirm that a new revert commit was created.

Execution

Step 1: Create Experiment Directory

Execute:

```
mkdir git-experiment-12
cd git-experiment-12
```

This creates and opens the directory for Experiment 12.

Step 2: Initialize Git Repository

Execute:

```
git init
```

Sample Output:

```
Initialized empty Git repository in
.../git-experiment-12/.git/
```

Step 3: Create Initial Project File

Execute:

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

Step 4: Create Initial Commit

Execute:

```
git add project.txt
git commit -m "Initial project 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: Introduce a Change

Execute:

```
echo Temporary unwanted feature >> project.txt
```

Check the file:

```
type project.txt
```

Sample Output:

```
Project Management with Git
Temporary unwanted feature
```

Step 6: Commit the Change

Execute:

```
git add project.txt
git commit -m "Add temporary feature"
```

This commit represents the change that will later be reverted.

Step 7: Create Another Commit

Create another file:

```
echo Experiment 12 documentation > notes.txt
```

Stage and commit it:

```
git add notes.txt
git commit -m "Add experiment documentation"
```

Now the commit to be reverted is not necessarily the latest commit.

Step 8: Display Commit History

Execute:

```
git log --oneline
```

Sample Output:

```
c3d4e5f Add experiment documentation
b2c3d4e Add temporary feature
a1b2c3d Initial project commit
```

The actual commit IDs generated on your computer will be different.
Suppose:

```
b2c3d4e Add temporary feature
```

is the commit whose changes must be undone.

Step 9: Inspect the Selected Commit

Before reverting, inspect the commit:

```
git show b2c3d4e
```

Replace `b2c3d4e` with the actual commit ID displayed on your computer.
This allows us to confirm the changes introduced by the selected commit.

Step 10: Revert the Selected Commit

Execute:

```
git revert --no-edit b2c3d4e
```

Replace `b2c3d4e` with the actual commit ID.
The syllabus uses `abc123` as the commit identifier example.
Therefore, the general syllabus operation can be written as:

```
git revert abc123
```

In an actual repository, use the real commit ID shown by:

```
git log --oneline
```

Step 11: Understand –no-edit

The command:

```
git revert --no-edit <commit-id>
```

creates the revert commit using Git's automatically generated commit message without opening an editor.
Without `--no-edit`, the basic command is:

```
git revert <commit-id>
```

Git may open an editor so that the revert commit message can be reviewed or modified.

Step 12: Verify the Project File

On Windows, execute:

```
type project.txt
```

Expected Output:

```
Project Management with Git
```

The line:

```
Temporary unwanted feature
```

has been removed because its introducing commit was reverted.

Step 13: Verify Repository Status

Execute:

```
git status
```

Sample Output:

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

The branch name may be `main`, `master`, or another name depending on the repository.

Step 14: Display Updated History

Execute:

```
git log --oneline
```

Sample Output:

```
d4e5f6a Revert "Add temporary feature"
c3d4e5f Add experiment documentation
b2c3d4e Add temporary feature
a1b2c3d Initial project commit
```

Notice that:

- The original commit still exists.
- Git created a new revert commit.
- The new commit reverses the selected change.

Step 15: Display Graphical History

Execute:

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

Sample Output:

```
* d4e5f6a (HEAD -> master) Revert "Add temporary feature"
* c3d4e5f Add experiment documentation
* b2c3d4e Add temporary feature
* a1b2c3d Initial project commit
```

This confirms that the repository history has been preserved.

Main Command for Experiment 12

The main command specified by the experiment is:

```
git revert abc123
```

Here, `abc123` represents the commit whose changes must be undone.
For an actual experiment, first obtain the real commit ID:

```
git log --oneline
```

Then execute:

```
git revert <actual-commit-id>
```

For easier terminal execution without opening an editor:

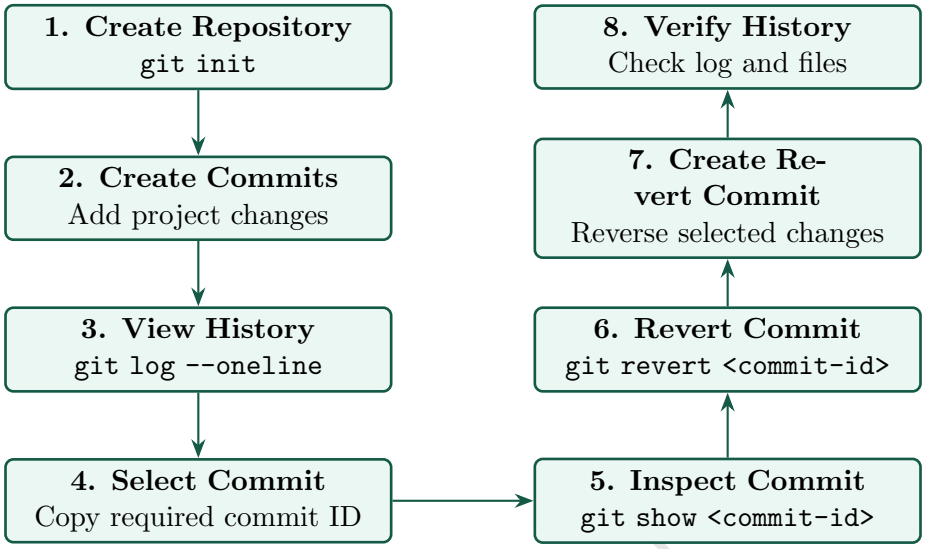
```
git revert --no-edit <actual-commit-id>
```

Complete Command Sequence

```
mkdir git-experiment-12
cd git-experiment-12

git init

echo Project Management with Git > project.txt

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

echo Temporary unwanted feature >> project.txt

git add project.txt
git commit -m "Add temporary feature"

echo Experiment 12 documentation > notes.txt

git add notes.txt
git commit -m "Add experiment documentation"

git log --oneline

git show <commit-id>

git revert --no-edit <commit-id>

type project.txt

git status

git log --oneline

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

Experiment Workflow

The complete workflow of Experiment 12 is shown below.



Workflow Explanation:

1. Create or open the required Git repository.
2. Create the project commits.
3. Display the commit history.
4. Identify the commit whose changes must be undone.
5. Inspect the selected commit.
6. Execute the `git revert` command.
7. Git creates a new commit that reverses the selected changes.
8. Verify the final files and Git history.

Git Revert vs Git Reset

Feature	git revert	git reset
Main Purpose	Reverse changes using a new commit	Move the current branch reference to another commit
Original Commit	Remains in history	Commits may no longer be reachable from the moved branch tip, depending on the reset target
Creates New Commit	Yes, normally	No
Shared History	Usually preferred for reversing an already shared commit	Requires caution because it can rewrite branch history
Basic Command	`git revert <id>`	`git reset <id>`

Handling Revert Conflicts

Sometimes a revert may conflict with changes made after the selected commit.

If a conflict occurs, first execute:

```
git status
```

Open the conflicting files and resolve the conflicts.

Then stage the corrected files:

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

Continue the revert:

```
git revert --continue
```

To cancel an in-progress revert operation:

```
git revert --abort
```

Result

Thus, the changes introduced by the specified Git commit were successfully undone using the `git revert` command. Git preserved the original commit history and created a new revert commit containing the reverse changes.

Viva Questions and Answers

1. **What is git revert?**

   `git revert` is a command used to reverse the changes introduced by a selected commit by creating a new commit.

2. **What is the basic syntax of git revert?**

   ```
   git revert <commit-id>
   ```

3. **Which commit ID is mentioned in this experiment?**

   The experiment uses:

   ```
   abc123
   ```

   as the commit identifier.

4. **How do we find the actual commit ID?**

   Execute:

   ```
   git log --oneline
   ```

5. **Does git revert delete the original commit?**

   No. The original commit remains in the repository history.

6. **Does git revert create a new commit?**

   Yes. It normally creates a new commit containing the reverse changes.

7. **Why is git revert useful?**

   It allows unwanted committed changes to be reversed while preserving the existing commit history.

8. **Which command can be used to inspect a commit before reverting it?**

   ```
   git show <commit-id>
   ```

9. **What does –no-edit do?**

   It accepts Git's automatically generated revert commit message without opening an editor.

10. **What is the difference between revert and reset?**

    Revert normally creates a new commit to reverse changes. Reset moves the branch reference and can change the visible branch history.

11. **Which command displays compact commit history?**

    ```
    git log --oneline
    ```

12. **What should we do if a revert conflict occurs?**

    Resolve the conflicting files, stage them, and execute:

```
git revert --continue
```

13. **How can an in-progress revert be cancelled?**

    Execute:

```
git revert --abort
```

14. **Can an older commit be reverted?**

    Yes. A specific older commit can be selected using its commit ID. However, conflicts may occur if later changes depend on it.

15. **Why is the commit ID important?**

    The commit ID uniquely identifies the commit whose changes must be inspected or reverted.

Conclusion

In this experiment, the process of undoing committed changes was demonstrated using `git revert`. The required commit was identified from the Git history, inspected using its commit ID, and reverted. The final Git log confirmed that the original commit was preserved and a new revert commit was created.