VTU 2025 Scheme

Project Management (with Git)

1BCSL307A – 3rd Semester

EXPERIMENT – 03

Creating and Managing Branches – Git Stash

Aim

To write and execute Git commands to stash uncommitted changes, switch to another branch, and then apply the previously stashed changes.

Objectives

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

- Understand the concept of stashing in Git.
- Temporarily save uncommitted changes.
- View the available Git stash entries.
- Switch between branches without losing current work.
- Apply previously stashed changes.
- Understand the difference between `git stash apply` and `git stash pop`.

Software Requirements

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

Theory

Git Stash is a Git feature used to temporarily save uncommitted changes in the working directory.

Sometimes a developer is working on a file but needs to switch to another branch before the current work is ready to be committed.

Instead of creating an incomplete commit or losing the changes, the developer can temporarily store the changes using `git stash`.

After completing the work on another branch, the developer can return and restore the saved changes.

Why is Git Stash Used?

- Temporarily saves unfinished work.
- Helps switch branches safely.
- Avoids unnecessary or incomplete commits.
- Keeps the working directory clean.
- Allows saved changes to be restored later.
- Helps developers manage multiple tasks efficiently.

Important Git Stash Commands

Command	Purpose
`git stash`	Temporarily saves the current tracked-file changes.
`git stash list`	Displays the available stash entries.
`git stash apply`	Applies the most recent stash without removing it from the stash list.
`git stash pop`	Applies the most recent stash and removes it from the stash list if the operation succeeds.
`git switch branch-name`	Switches to another branch.
`git status`	Displays the current repository status.

Git Stash Concept

The basic Git stash process is shown below.

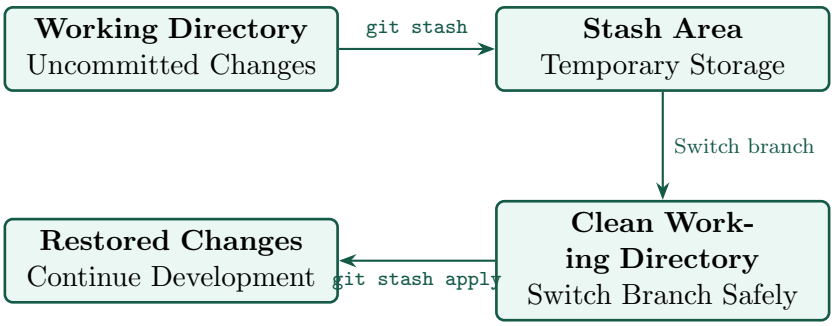


Diagram Explanation:

- The developer has uncommitted changes in the working directory.

- `git stash` temporarily stores those changes.
- The working directory becomes clean.
- The developer can switch to another branch.
- The saved changes can later be restored using `git stash apply`.

Procedure

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

Step 2: Create a new directory for Experiment 03.

Step 3: Initialize the directory as a Git repository.

Step 4: Create an initial file.

Step 5: Add and commit the initial file.

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

Step 7: Create another branch named `feature-branch`.

Step 8: Modify the file in the master branch without committing the changes.

Step 9: Check the repository status.

Step 10: Temporarily save the changes using `git stash`.

Step 11: Verify the stash using `git stash list`.

Step 12: Switch to `feature-branch`.

Step 13: Return to the `master` branch.

Step 14: Apply the previously stashed changes.

Step 15: Verify that the changes have been restored.

Execution

Step 1: Create Experiment Directory

Execute:

```
mkdir git-experiment-03
cd git-experiment-03
```

Explanation:
The first command creates a new directory named `git-experiment-03`.
The second command moves into the newly created directory.

Step 2: Initialize Git Repository

Execute:

```
git init
```

Sample Output:

```
Initialized empty Git repository in
.../git-experiment-03/.git/
```

The directory is now initialized as a Git repository.

Step 3: Create Initial File

Execute:

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

This creates a file named `project.txt`.

Step 4: Stage the Initial File

Execute:

```
git add project.txt
```

The file is added to the staging area.

Step 5: Create Initial Commit

Execute:

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

This creates the initial commit.
Note:
If Git requests user information, 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
```

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

Step 7: Create Feature Branch

Execute:

```
git branch feature-branch
```

The repository now contains the branches:

```
git branch
```

Sample Output:

```
  feature-branch
* master
```

Step 8: Modify the File

While remaining on the `master` branch, execute:

```
echo Unfinished work >> project.txt
```

The file now contains:

```
Project Management with Git
Unfinished work
```

The new change has not yet been committed.

Step 9: Check Repository Status

Execute:

```
git status
```

Sample Output:

```
Changes not staged for commit:
  modified: project.txt
```

This indicates that the file contains uncommitted changes.

Step 10: Stash the Changes

Execute:

```
git stash
```

Sample Output:

```
Saved working directory and index state
WIP on master: ...
```

Explanation:
The current uncommitted changes are temporarily stored in the Git stash.
The working directory returns to the state of the latest commit.

Step 11: Check Repository Status Again

Execute:

```
git status
```

Sample Output:

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

The working directory is now clean.

Step 12: Display Stash List

Execute:

```
git stash list
```

Sample Output:

```
stash@{0}: WIP on master: ...
```

The entry `stash@{0}` represents the most recent stash.

Step 13: Switch to Feature Branch

Execute:

```
git switch feature-branch
```

Sample Output:

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

Because the uncommitted changes were stashed, switching branches can now be performed with a clean working tree.

Step 14: Verify Current Branch

Execute:

```
git branch
```

Sample Output:

```
* feature-branch
  master
```

The asterisk indicates that `feature-branch` is currently active.

Step 15: Switch Back to Master

Execute:

```
git switch master
```

Sample Output:

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

We return to the branch where the changes were originally created.

Step 16: Apply the Stashed Changes

Execute:

```
git stash apply
```

Explanation:
The most recently stashed changes are restored to the working directory.
Unlike `git stash pop`, the `git stash apply` command does not automatically remove the stash entry.

Step 17: Verify Restored Changes

Execute:

```
git status
```

Sample Output:

```
Changes not staged for commit:
  modified: project.txt
```

This confirms that the previously stashed modification has been restored.
To display the file:

```
type project.txt
```

Sample Output:

```
Project Management with Git
Unfinished work
```

Step 18: Check Stash List

Execute:

```
git stash list
```

The stash entry will still be available because `git stash apply` restores the changes without deleting the stash.

Complete Command Sequence

The complete command sequence for Experiment 03 is:

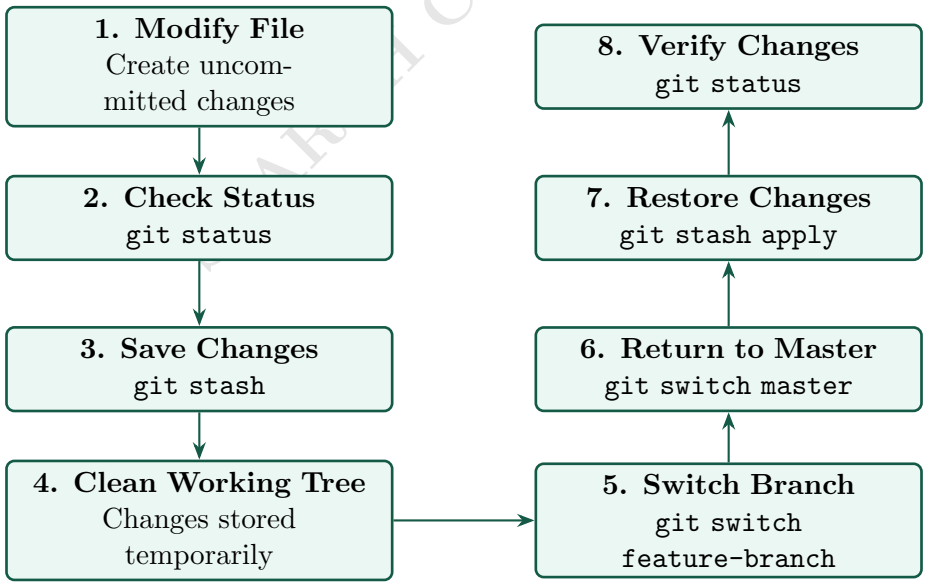
```
mkdir git-experiment-03
cd git-experiment-03

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

echo Unfinished work >> project.txt

git status

git stash

git status

git stash list

git switch feature-branch

git branch

git switch master

git stash apply

git status

type project.txt

git stash list
```

Command Summary

Command	Description
`git stash`	Temporarily stores current tracked-file changes.
`git stash list`	Displays all available stash entries.
`git switch feature-branch`	Switches to the feature branch.
`git switch master`	Returns to the master branch.
`git stash apply`	Restores the most recent stashed changes without removing the stash entry.
`git stash pop`	Restores the most recent stash and removes it from the stash list if successful.
`git status`	Displays the current repository status.

Experiment Workflow

The complete Git stash workflow is shown below.



Workflow Explanation:

1. Make changes to a tracked project file.
2. Check the repository status.
3. Temporarily store the changes using `git stash`.

4. The working directory becomes clean.
5. Switch to another branch.
6. Return to the original branch.
7. Restore the saved changes using `git stash apply`.
8. Verify that the changes have been restored.

Difference Between git stash apply and git stash pop

Feature	**git stash apply**	**git stash pop**
Restore changes	Yes	Yes
Remove stash entry	No	Yes, if successfully applied
Useful when	Stash may be needed again	Stash is no longer required

Result

Thus, the uncommitted changes were successfully stored temporarily using Git stash. The repository was switched to another branch, and the previously stashed changes were successfully applied after returning to the required branch.

Viva Questions and Answers

1. **What is Git stash?**

   Git stash temporarily stores uncommitted changes so that they can be restored later.

2. **Why is git stash used?**

   It is used when a developer wants to temporarily save unfinished work and switch to another branch.

3. **Which command is used to stash changes?**

```
git stash
```

4. **How do you display available stashes?**

```
git stash list
```

5. **How do you apply the latest stash?**

```
git stash apply
```

6. **What is stash@{0}?**

   It represents the most recent entry in the Git stash list.

7. **Does git stash apply delete the stash?**

   No. It applies the changes but keeps the stash entry.

8. **What does git stash pop do?**

   It applies the most recent stash and removes the stash entry if the operation succeeds.

9. **What is the main difference between stash apply and stash pop?**

   `git stash apply` keeps the stash entry, whereas `git stash pop` normally removes it after successful application.

10. **Which command is used to switch branches?**

```
git switch branch-name
```

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

```
git branch
```

    The branch marked with an asterisk (*) is the active branch.

12. **What happens to the working directory after git stash?**

    The tracked changes saved by the stash are removed from the working tree, returning those files to the state of the current commit.

13. **Can stashed changes be restored later?**

    Yes. They can be restored using commands such as `git stash apply` or `git stash pop`.

Conclusion

In this experiment, Git stash operations were demonstrated. Uncommitted changes were temporarily stored using `git stash`, allowing the user to switch branches with a clean working directory. After returning to the required branch, the saved changes were successfully restored using `git stash apply`.