QMP 7.1 D/F

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

Project Management (with Git)

&

1BCSL307A

B.E - III Semester Lab Manual

Academic Year: 2026-27

Name : _____

USN : _____

Batch :_____Section : _____

Mail -ID :_____Contact Number:

QMP 7.1 D/F

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

Project Management (with Git)

&

1BCSL307A

B.E – III Semester Lab Manual

Prepared By
Mr. Anil Kumar
Asst. Prof., Dept. of ISE

Reviewed by:
Mr. Arun Kumar M S
Asst. Prof., Dept. of ISE

Approved by:

Dr. Thara D K
Professor & HOD,
Dept. of ISE, CIT Gubbi

Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)
(NAAC Accredited & ISO 9001:2015 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

INSTITUTE VISION

To create centers of excellence in education and to serve the society by enhancing the quality of life through value based professional leadership

INSTITUTE MISSION

- To provide high quality technical and professionally relevant education in a diverse learning environment
- To provide the values that prepare students to lead their lives with personal integrity, professional ethics and civic responsibility in a global society
- To prepare the next generation of skilled professionals to successfully compete in the diverse global market
- To promote a campus environment that welcomes and honors women and men of all races, creeds and cultures, values and intellectual curiosity, pursuit of knowledge and academic integrity and freedom
- To offer a wide variety of off-campus education and training programmes to individuals and groups
- To stimulate collaborative efforts with industry, universities, government and professional societies
- To facilitate public understanding of technical issues and achieve excellence in the operations of the institute

QUALITY POLICY

Our organization delights customers (students, parents and society) by providing value added quality education to meet the national and international requirements. We also provide necessary steps to train the students for placement and continue to improve our methods of education to the students through effective quality management system, quality policy and quality objectives.

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)

(NAAC Accredited & ISO 9001:2015 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

Vision

To educate and train students by imparting strong foundational knowledge in Information Science and Engineering, while creating centres of excellence in computing education

Mission

- By providing qualified and dedicated human resources, we ensure students receive the best education in Information Science and Engineering
- By providing the state-of-the-art infrastructure, we facilitate a conducive learning environment for students pursuing excellence in the field
- By fostering collaboration with industry and research institutions, we enhance students' exposure to practical applications and cutting-edge developments
- By instilling a sense of social responsibility and promoting professional ethics, we prepare students to be ethical and socially conscious professionals in their careers.

Program Outcomes (POs)

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Engineering Tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and the world**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
8. **Individual and Collaborative team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
9. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
10. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
11. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Educational Objectives (PEOs)

After four Years of Graduation, our graduates can:

PEO1: Develop the next generation of highly skilled graduates equipped with a strong knowledge in Information Science and Engineering for creating innovative solutions to society's pressing challenges.

PEO2: To motivate students to pursue higher studies in various disciplines in Information Science and Engineering with the aim of cherishing careers in research and development, academia, industry and entrepreneurship.

PEO3: To Produce engineers who are professionals' entrepreneurs and capable of self-learning to excel in their career.

PEO4: To prepare graduates who excel in diverse learns upholding professional ethics and societal responsibilities.

Program Specific Outcomes (PSOs)

At the time of graduation, our graduates can:

PSO1: Students will possess the ability to analyze, design, develop, test and apply mathematical foundations, management principles, and adapt emerging technologies effectively in the creation of computational solutions.

PSO2: Students will possess advanced skills in artificial intelligence, cybersecurity, cloud computing, project management, mobile application development, embedded systems, and big data analytics, making them highly employable, well prepared for higher education, and capable of contributing to cutting-edge research and innovation in Information Science and Engineering.

Instructions to the Candidates

- Student should come with thorough preparation for the experiment to be conducted.
- Student should take prior permission from the concerned faculty before availing the leave.
- Student should come with proper dress code and to be present on time in the laboratory.
- Student will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.
- Student will not be permitted to attend the laboratory unless they bring the observation book fully completed in all respects pertaining to the experiment to be conducted in present class.
- Practical record and observation book should be neatly maintained.
- Student should obtain the signature of the staff-in-charge in the observation book after completing each experiment.
- Theory related to each experiment should be written in the practical record before procedure in your own words with appropriate references.

CONTENTS

SET OF EXPERIMENTS

1. **Setting Up and Basic Commands**: Initialize a new Git repository in a directory. Create a new file and add it to the staging area, and commit the changes with an appropriate commit message.
2. **Creating and Managing Branches:** Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "featurebranch" into "master."
3. **Creating and Managing Branches:** Write the commands to stash your changes, switch branches, and then apply the stashed changes
4. **Collaboration and Remote Repositories:** Clone a remote Git repository to your local machine.
5. **Collaboration and Remote Repositories:** Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.
6. **Collaboration and Remote Repositories:** Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.
7. **Git Tags and Releases:** Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.
8. **Advanced Git Operations:** Write the command to cherry-pick a range of commits from "source-branch" to the current branch.
9. **Analysing and Changing Git History:** Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?
10. **Analysing and Changing Git History**: Write the Git command to display all commits made by the author "JohnDoe" between “January 1, 2025” and “December 31, 2025.”
11. **Analysing and Changing Git History**: Write the command to display the last five commits in the repository's history.
12. **Analysing and Changing Git History:** Write the command to undo the changes introduced by the commit with the ID "abc123".

COURSE OUTCOMES	
CO1:	Demonstrate the use of basic Git repository commands for version control operations.
CO2:	Implement branch management strategies for collaborative software development
CO3:	Apply collaboration techniques using remote repositories and distributed version control systems.
CO4:	Configure Git tags, releases, and advanced Git operations for effective project management.
CO5:	Modify Git history to maintain repository integrity and support project requirements.

RBT LEVELS						
COS	**Remember (L1)**	**Understand (L2)**	**Apply (L3)**	**Analyze (L4)**	**Evaluate (L5)**	**Create (L6)**
CO1			√			
CO2			√			
CO3			√			
CO4			√			
CO5				√		

CO – PO MAPPING														
CO-PO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO1	3	1	1	0	3				1	1	1		2	2
CO2	3	2	2	1	3				1	1	2		2	2
CO3	2	2	2	1	3				1	1	2		2	2
CO4	3	2	2	1	3				1	1	2		2	2
CO5	3	3	2	2	3				1	1	2		2	2
Average	**2.8**	**2**	**1.8**	**1**	**3**				**1**	**1**	**1.8**		**2**	**2**

PROJECT MANAGEMENT WITH GIT

LABORATORY MANUAL

Git Repository Management, Branching, Collaboration,
Releases and History Management

Course: Project Management with Git

Laboratory Theme: Version Control and Collaborative Software Development

Tool: Git / Git Bash / Terminal

Repository Platform: GitHub or any Git-compatible remote repository

General Instructions

- Install Git and verify the installation using git --version.
- Configure the Git username and email before performing commits.
- Commands shown using master assume that the local primary branch is named master. If the repository uses main, replace master with main.
- Replace placeholder values such as <commit-id>, <remote-url>, and <file-name> with actual values from the student's repository.
- The outputs shown in this manual are representative examples. Actual commit IDs, dates, and file names will vary.
- For destructive history operations, work on a practice repository or create a backup branch before modifying history.

Basic Git Configuration

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

The following experiments provide practical exercises covering repository initialization, staging and committing, branching, stashing, remote collaboration, rebasing, merging, tagging, cherry-picking, viewing history, filtering commits, and reverting changes.

Experiment 1: Setting Up and Basic Commands

Experiment Statement: Initialize a new Git repository in a directory. Create a new file, add it to the staging area, and commit the changes with an appropriate commit message.

Objective: To initialize a Git repository and demonstrate the basic working-tree, staging-area, and commit workflow.

Solution / Commands

```
mkdir GitLab01
cd GitLab01
git init
echo "# Git Lab Experiment 1" > README.md
git status
git add README.md
git status
git commit -m "Add initial README file"
git log --oneline -1
```

Expected Result / Output

```
Initialized empty Git repository in .../GitLab01/.git/
On branch master
Changes to be committed:
  new file: README.md

[master <commit-id>] Add initial README file
 1 file changed, 1 insertion(+)
```

```
 create mode 100644 README.md

<commit-id> Add initial README file
```

Learning Note: git init creates the .git directory. git add moves the file to the staging area, and git commit permanently records the staged snapshot in repository history.

Experiment 2: Creating and Managing Branches

Experiment Statement: Create a new branch named feature-branch. Switch to the master branch and merge feature-branch into master.

Objective: To create, switch between, and merge Git branches for parallel development.

Solution / Commands

```
# Create and switch to the feature branch
git switch -c feature-branch

# Make a sample change
echo "Feature work" >> feature.txt
git add feature.txt
git commit -m "Add feature work"

# Switch to master
git switch master

# Merge the feature branch into master
git merge feature-branch

# Verify branches
git branch
git log --oneline --graph --all
```

Expected Result / Output

```
Switched to a new branch 'feature-branch'
[feature-branch <commit-id>] Add feature work

Switched to branch 'master'
Updating <old-id>..<new-id>
Fast-forward
 feature.txt | 1 +
 1 file changed, 1 insertion(+)

The feature-branch changes are now part of master.
```

Learning Note: If master and feature-branch have diverged, Git may create a merge commit instead of a fast-forward merge.

Experiment 3: Stashing Changes and Switching Branches

Experiment Statement: Write the commands to stash your changes, switch branches, and then apply the stashed changes.

Objective: To temporarily save uncommitted work and apply it after switching branches.

Solution / Commands

```
# Check the current changes
git status

# Stash tracked changes
git stash push -m "Temporary work"

# Switch to another branch
git switch master

# Return to the original branch
git switch feature-branch

# Apply the most recent stash
git stash apply

# Check the result
git status
git stash list
```

Expected Result / Output

```
Saved working directory and index state On feature-branch: Temporary work
Switched to branch 'master'
Switched to branch 'feature-branch'

On branch feature-branch
Changes not staged for commit:
  modified: <file-name>

The stashed changes are restored in the working directory.
```

Learning Note: git stash apply restores the changes but keeps the stash entry. Use git stash pop if you want to restore the changes and remove the stash entry.

Experiment 4: Collaboration and Remote Repositories – Clone

Experiment Statement: Clone a remote Git repository to the local machine.

Objective: To obtain a complete local working copy of a remote Git repository.

Solution / Commands

```
git clone <remote-url>
cd <repository-name>
git remote -v
git status
```

Expected Result / Output

```
Cloning into '<repository-name>'...
remote: Enumerating objects...
Receiving objects: 100% ...
Resolving deltas: 100% ...

origin  <remote-url> (fetch)
origin  <remote-url> (push)

On branch master/main
Your branch is up to date with 'origin/master' or 'origin/main'.
```

Learning Note: git clone copies the repository, its history, and remote-tracking information to the local machine.

Experiment 5: Fetch and Rebase onto an Updated Remote Branch

Experiment Statement: Fetch the latest changes from a remote repository and rebase the local branch onto the updated remote branch.

Objective: To synchronize local work with the latest remote history using rebase.

Solution / Commands

```
# Fetch remote changes
git fetch origin

# Rebase the current local branch onto updated remote master
git rebase origin/master

# Verify the history
git log --oneline --graph --decorate --all

# If conflicts occur:
git status
# Resolve the conflicting files, then:
git add <resolved-file>
git rebase --continue

# To cancel the rebase if required:
git rebase --abort
```

Expected Result / Output

```
From <remote-repository>
   <old-id>..<new-id>  master -> origin/master

Successfully rebased and updated refs/heads/<local-branch>.

The local commits now appear on top of the latest origin/master history.
```

Learning Note: If the repository uses main instead of master, use git rebase origin/main. Rebase rewrites local commit IDs, so avoid rebasing commits that other collaborators are already depending on.

Experiment 6: Merge a Feature Branch with a Custom Commit Message

Experiment Statement: Write the command to merge feature-branch into master while providing a custom commit message for the merge.

Objective: To merge feature development into the main development branch with an explicit merge commit message.

Solution / Commands

```
git switch master
git merge --no-ff feature-branch -m "Merge feature-branch into master"
git log --oneline --graph -5
```

Expected Result / Output

```
Switched to branch 'master'
Merge made by the 'ort' strategy.
<files changed>

The history contains a merge commit with message:
Merge feature-branch into master
```

Learning Note: The --no-ff option ensures that a merge commit is created even when a fast-forward merge would otherwise be possible. This makes the custom merge message visible in the history.

Experiment 7: Git Tags and Releases – Lightweight Tag

Experiment Statement: Write the command to create a lightweight Git tag named v1.0 for a commit in the local repository.

Objective: To mark a specific commit with a version identifier using a lightweight tag.

Solution / Commands

```
# Tag the current commit
git tag v1.0

# Or tag a specific commit
git tag v1.0 <commit-id>

# Verify the tag
git tag
git show v1.0
```

Expected Result / Output

```
v1.0

commit <commit-id>
Author: <name>
Date:   <date>

    <commit message>
```

Learning Note: A lightweight tag is simply a named reference to a commit. Annotated tags can be created using git tag -a v1.0 -m "Release v1.0".

Experiment 8: Advanced Git Operations – Cherry-Pick a Range

Experiment Statement: Write the command to cherry-pick a range of commits from source-branch to the current branch.

Objective: To selectively apply a sequence of existing commits from another branch onto the current branch.

Solution / Commands

```
# First inspect the source branch history
git log --oneline source-branch

# Example: cherry-pick commits from <first-commit> through <last-commit>
git cherry-pick <first-commit>^..<last-commit>

# Alternatively, for the last three commits on source-branch:
git cherry-pick source-branch~3..source-branch

# Verify
git log --oneline -5
```

Expected Result / Output

```
[current-branch <new-id>] Commit message 1
[current-branch <new-id>] Commit message 2
[current-branch <new-id>] Commit message 3

The selected commits are applied to the current branch with new commit IDs.
```

Learning Note: The ^ before the first commit makes the range inclusive. If a cherry-pick conflict occurs, resolve the files, run git add <file>, then git cherry-pick --continue. Use git cherry-pick --abort to cancel.

Experiment 9: Analysing Git History – View a Specific Commit

Experiment Statement: Given a commit ID, use Git to view the details of that specific commit, including the author, date, and commit message.

Objective: To inspect detailed metadata and changes associated with a particular commit.

Solution / Commands

```
git show <commit-id>

# To display only the commit metadata and message
git show --no-patch --format=fuller <commit-id>

# To inspect the changed files/statistics
git show --stat <commit-id>
```

Expected Result / Output

```
commit <commit-id>
Author: John Doe <john@example.com>
AuthorDate: <date>
Commit: John Doe <john@example.com>
CommitDate: <date>

    Add login validation

    <diff / file changes>
```

Learning Note: git show is useful for examining who made a change, when it was made, the commit message, and the actual content changes.

Experiment 10: Analysing Git History – Commits by Author and Date

Experiment Statement: Write the Git command to display all commits made by the author JohnDoe between January 1, 2025 and December 31, 2025.

Objective: To filter repository history by author and a specified date range.

Solution / Commands

```
git log --author="JohnDoe" --since="2025-01-01" --until="2025-12-31 23:59:59"

# Compact version
git log --author="JohnDoe" --since="2025-01-01" --until="2025-12-31 23:59:59" --oneline
```

Expected Result / Output

```
commit <commit-id-1>
Author: JohnDoe <...>
Date:   <date in 2025>

    Implement feature A

commit <commit-id-2>
Author: JohnDoe <...>
Date:   <date in 2025>

    Fix validation issue

Only commits matching the specified author and date range are displayed.
```

Learning Note: The --author option searches the author identity. The date filters restrict the history to the specified period.

Experiment 11: Analysing Git History – Last Five Commits

Experiment Statement: Write the command to display the last five commits in the repository's history.

Objective: To inspect the most recent commits in a concise format.

Solution / Commands

```
git log -5

# Compact one-line format
git log -5 --oneline
```

Expected Result / Output

```
<commit-id-1> (HEAD -> master) Latest commit
<commit-id-2> Previous commit
<commit-id-3> Earlier commit
<commit-id-4> Earlier commit
<commit-id-5> Earlier commit

Exactly the latest five commits are displayed.
```

Learning Note: The -5 option limits the number of commits shown to five.

Experiment 12: Changing Git History – Undo a Commit Using Revert

Experiment Statement: Write the command to undo the changes introduced by the commit with the ID abc123.

Objective: To safely undo the effect of an existing commit without deleting it from the repository history.

Solution / Commands

```
git revert abc123

# If Git opens an editor, save the generated revert commit message.
# Then verify the history:
git log --oneline -5
git status
```

Expected Result / Output

```
[master <new-commit-id>] Revert "Original commit message"
 1 file changed, <n> insertions(+), <n> deletions(-)

The changes introduced by abc123 are undone by a new revert commit, while abc123 remains in
the repository history.
```

Learning Note: git revert is generally preferred for shared branches because it preserves history. It is different from git reset, which moves the branch reference and can discard or rewrite local history.

Git Command Quick Reference

Operation	Command
Initialize repository	git init
Check status	git status
Stage a file	git add <file>
Commit changes	git commit -m "message"
Create and switch branch	git switch -c <branch>
Switch branch	git switch <branch>
Merge branch	git merge <branch>
Stash changes	git stash push -m "message"
Apply stash	git stash apply
Clone repository	git clone <remote-url>
Fetch remote changes	git fetch origin
Rebase	git rebase origin/master
Create tag	git tag v1.0
Cherry-pick commits	git cherry-pick <first>^..<last>
View commit	git show <commit-id>
Filter author/date	git log --author="..." --since="..." --until="..."
Last five commits	git log -5 --oneline
Undo a commit safely	git revert <commit-id>

Important Note on master and main

Modern Git hosting platforms commonly use main as the default branch, while some lab exercises and older repositories use master. The experiments in this manual retain master to match the experiment statements. If your repository uses main, replace master with main in the relevant commands.

Suggested Lab Record Format

For each experiment, students should record: Experiment Statement, Objective, Commands/Solution, Screenshots of important command outputs, Result, and Viva Questions (if prescribed by the instructor).

VIVA QUESTIONS

Experiment 1: Setting Up and Basic Git Commands

1. **What is Git?**
   Answer: Git is a distributed version control system used to track changes in source code and manage different versions of a project.

2. **What is the purpose of git init?**
   Answer: It initializes a new Git repository in the current directory by creating the .git directory.

3. **What is the purpose of git add?**
   Answer: It moves changes from the working directory to the staging area, preparing them for commit.

4. **What is a Git commit?**
   Answer: A commit is a snapshot of staged changes stored in the repository history.

5. **What is the difference between git add and git commit?**
   Answer: git add stages changes, while git commit permanently records the staged changes in the repository history.

Experiment 2: Creating and Managing Branches

1. **What is a Git branch?**
   Answer: A branch is an independent line of development that allows developers to work on features without directly affecting another branch.

2. **How do you create a new branch?**
   Answer: git branch feature-branch or git switch -c feature-branch.

3. **What is the purpose of git switch?**
   Answer: It is used to switch from one branch to another.

4. **What is the purpose of git merge?**
   Answer: It combines the changes from one branch into another branch.

5. **What happens when feature-branch is merged into master?**
   Answer: The changes committed in feature-branch become part of the master branch.

Experiment 3: Stashing Changes and Switching Branches

1. **What is Git stash?**
   Answer: Git stash temporarily stores uncommitted changes so that the working directory can be used for another task.

2. **Which command is used to stash changes?**
   Answer: git stash or git stash push -m "message".

3. **Why is stashing useful?**
   Answer: It allows a developer to temporarily save incomplete work without committing it.

4. **What is the difference between git stash apply and git stash pop?**
   Answer: git stash apply restores the changes but keeps the stash; git stash pop restores the changes and removes the stash entry.

5. **How can you view available stashes?**
   Answer: Using git stash list.

Experiment 4: Cloning a Remote Repository

1. **What is a remote repository?**
   Answer: A remote repository is a Git repository hosted on another system or platform, such as GitHub, that can be accessed over a network.

2. **What is the purpose of git clone?**
   Answer: It creates a local copy of a remote repository along with its history.

3. **What is origin in Git?**
   Answer: origin is the default name given to the remote repository from which a repository was cloned.

4. **Which command displays remote repository information?**
   Answer: git remote -v.

5. **What is the difference between cloning and downloading a ZIP file?**
   Answer: Cloning provides the complete Git history and remote connection, whereas a ZIP download normally provides only the current files.

Experiment 5: Fetch and Rebase

1. **What is the purpose of git fetch?**
   Answer: It downloads the latest changes and commits from a remote repository without modifying the current working branch.

2. **What is rebasing in Git?**
   Answer: Rebasing moves or reapplies local commits on top of another branch's latest commit history.

3. **What does git rebase origin/master do?**
   Answer: It reapplies the current branch's local commits on top of the latest origin/master.

4. **What is one advantage of rebase?**
   Answer: It creates a cleaner, more linear project history.

5. **What should you do if a rebase produces conflicts?**
   Answer: Resolve the conflicts, stage the resolved files, and use git rebase --continue. To cancel, use git rebase --abort.

Experiment 6: Merge with a Custom Commit Message

1. **What is the purpose of git merge?**
   Answer: It combines the history and changes of another branch into the current branch.

2. **What does the --no-ff option do?**
   Answer: It forces Git to create a merge commit even when a fast-forward merge is possible.

3. **How can you provide a custom merge commit message?**
   Answer: Using git merge --no-ff feature-branch -m "Merge feature-branch into master".

4. **What is a merge conflict?**
   Answer: A merge conflict occurs when Git cannot automatically reconcile changes made in different branches.

5. **How do you verify the merge history?**
   Answer: Using git log --oneline --graph --all.

Experiment 7: Git Tags and Releases

1. **What is a Git tag?**
   Answer: A tag is a reference used to mark a specific commit, commonly for version releases.

2. **What is a lightweight tag?**
   Answer: A lightweight tag is a simple pointer to a specific commit without additional tag metadata.

3. **How do you create the tag v1.0?**
   Answer: git tag v1.0.

4. **How do you list all tags?**
   Answer: git tag.

5. **What is the difference between lightweight and annotated tags?**
   Answer: A lightweight tag points directly to a commit, while an annotated tag contains additional information such as the tagger, date, and message.

Experiment 8: Cherry-Picking a Range of Commits

1. **What is cherry-picking in Git?**
   Answer: Cherry-picking applies the changes introduced by selected commits to the current branch.

2. **Which command is used for cherry-picking?**
   Answer: git cherry-pick <commit-id>.

3. **How can you cherry-pick a range of commits?**
   Answer: Using a command such as git cherry-pick <first-commit>^..<last-commit>.

4. **Does cherry-pick preserve the original commit ID?**
   Answer: No. A new commit is created with a new commit ID.

5. **What should you do if a cherry-pick conflict occurs?**
   Answer: Resolve the conflict, stage the resolved files, and run git cherry-pick --continue.

Experiment 9: Viewing Details of a Specific Commit

1. **Which command displays the details of a commit?**
   Answer: git show <commit-id>.

2. **What information does git show provide?**
   Answer: It can display the commit ID, author, date, commit message, and changes introduced by the commit.

3. **What is a commit ID?**
   Answer: It is a unique SHA-based identifier assigned to a Git commit.

4. **How can you display only commit metadata without the changes?**
   Answer: git show --no-patch --format=fuller <commit-id>.

5. **Why is commit history important in project management?**
   Answer: It provides traceability of changes, identifies contributors, and helps developers understand how the project evolved.

Experiment 10: Filtering Commits by Author and Date

1. **Which Git command is used to view commit history?**
   Answer: git log.

2. **Which option filters commits by author?**
   Answer: --author.

3. **Which options filter commits by date?**
   Answer: --since and --until.

4. **Write the command to find JohnDoe's commits during 2025.**
   Answer:

git log --author="JohnDoe" --since="2025-01-01" --until="2025-12-31 23:59:59"

5. **Why is author-based filtering useful?**
   Answer: It helps identify and review the contributions made by a particular developer.

Experiment 11: Displaying Last Five Commits

1. **Which command displays the commit history?**
   Answer: git log.

2. **How do you display only the last five commits?**
   Answer: git log -5.

3. **How can you display the last five commits in one-line format?**
   Answer: git log -5 --oneline.

4. **What information is normally displayed by git log?**
   Answer: Commit ID, author, date, and commit message.

5. **Why is --oneline useful?**
   Answer: It provides a compact view of commit history, making it easier to review recent changes.

Experiment 12: Undoing Changes Using Git Revert

1. **Which command is used to undo a specific commit safely?**
   Answer: git revert <commit-id>.

2. **What does git revert abc123 do?**
   Answer: It creates a new commit that reverses the changes introduced by commit abc123.

3. **Does git revert delete the original commit?**
   Answer: No. The original commit remains in the repository history.

4. **What is the difference between git revert and git reset?**
   Answer: git revert creates a new commit to undo changes, while git reset moves the branch pointer and can rewrite or discard local history.

5. **Why is git revert preferred for shared branches?**
   Answer: It preserves the existing history and does not rewrite commits that other collaborators may already have pulled.