ACS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Project Management with Git

(BCS358C)

LAB MANUAL

(Effective from the academic year 2025-2026)

SEMESTER – III

Vision

To equip students with a strong foundation in **project management principles and version control using Git**, enabling them to effectively plan, track, collaborate, and manage software projects in alignment with industry standards and professional practices.

Mission

- To introduce students to **project management concepts** such as planning, scheduling, risk management, and teamwork in software development.
- To provide hands-on experience with **Git and GitHub** for version control, collaboration, and source code management.
- To develop students' ability to **manage software projects systematically**, from initiation to deployment.
- To enhance problem-solving, documentation, and collaboration skills required for **real-world software engineering projects**.
- To prepare students for **industry-oriented development environments** by adopting best practices in project management and version control.

Project Management with Git		Semester	3
Course Code	**BCS358C**	CIE Marks	50
Teaching Hours/Week(L:T:P:S)	0:0:2:0	SEE Marks	50
Credits	01	Exam Hours	100
Examination type(SEE)	Practical		

Course objectives:

● To familiar with basic command of Git

● To create and manage branches.

● To understand how to collaborate and work with Remote Repositories

● To familiar with virion controlling commands.

SLNO	**Experiment**
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 "feature-branch" 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 command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."
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 (Course Skill Set):
At the end of the course the student will be able to:

● Use the basics commands related to git repository
● Create and manage the branches
● Apply commands related to Collaboration and Remote Repositories
● Use the commands related to Git Tags, Releases and advanced git operations
● Analyses and change the git history

PROJECT MANAGEMENT WITH GIT

PROGRAM 1

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.

PROGRAM:

1. Initialize a new Git repository:

mkdir your_project_directory

cd your_project_directory

git init

2. Create a new file:

touch your_new_file.txt

3. Add the file to the staging area:
git add your_new_file.txt

If you want to add all new and modified files to the staging area, you can use:
git add

4. Commit the changes with a message:
git commit -m "Initial commit - adding a new file"

PROGRAM 2

2. Creating and Managing Branches

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."

PROGRAM:

1. Create a new branch named "feature-branch":

git branch feature-branch

Alternatively, you can create and switch to the new branch in one step:

git checkout -b feature-branch

2. Switch to the "master" branch:

git checkout master

3. Merge "feature-branch" into "master":

git merge feature-branch

If there are no conflicts, Git will automatically perform a fast-forward merge. If there are conflicts, Git will prompt you to resolve them before completing the merge. If you used git checkout -b feature branch to create and switch to the new branch, you can switch back to master and merge in a single command:
git checkout master

git merge feature-branch

4. Resolve any conflicts (if needed) and commit the merge: If there are conflicts, Git will mark the conflicted files. Open each conflicted file, resolve the conflicts, and then:

git add

git commit -m "Merge feature-branch into master"

Now, the changes from "feature-branch" are merged into the "master" branch. If you no longer need the "feature-branch," you can delete it:

git branch -d feature-branch

This assumes that the changes in "feature-branch" do not conflict with changes in the "master" branch. If conflicts arise during the merge.

PROGRAM 3

3. Creating and Managing Branches
Write the commands to stash your changes, switch branches, and then apply the stashed changes.

PROGRAM:

1. Stash your changes:

git stash save "Your stash message"

This command will save your local changes in a temporary area, allowing you to switch branches without committing the changes.

2. Switch to another branch:

git checkout your-desired-branch

3. Apply the stashed changes:

git stash apply

If you have multiple stashes and want to apply a specific stash, you can use:

git stash apply stash@{1}

After applying the stash, your changes are reapplied to the working directory.

4. Remove the applied stash (optional):

If you no longer need the stash after applying it, you can remove it:

git stash drop

To remove a specific stash:

git stash drop stash@{1}

If you want to apply and drop in one step, you can use git stash pop:

git stash pop

Now, you've successfully stashed your changes, stashed changes.

PROGRAM 4

4. Collaboration and Remote Repositories

Clone a remote Git repository to your local machine.

PROGRAM:

To clone a remote Git repository to your local machine, you can use the git clone command. Here's the general syntax:

git clone <repository_url>

Replace <repository_url> with the actual URL of the Git repository you want to clone. For example:

git clone https://github.com/example/repo.git

This command will create a new directory with the name of the repository and download all the files from the remote repository into that directory. If the repository is private and requires authentication, you might need to use the SSH URL or provide your credentials during the cloning process.

For SSH:

git clone git@github.com:example/repo.git

After running the git clone command, you'll have a local copy of the remote repository on your machine, and you can start working with the code.

PROGRAM 5

5. Collaboration and Remote Repositories

Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

PROGRAM:

1. Fetch the latest changes from the remote repository:

<mark>git fetch</mark>

This command fetches the latest changes from the remote repository without automatically merging them into your local branches.

2. Rebase your local branch onto the updated remote branch:

Assuming you are currently on the branch you want to update (replace `your-branch` with the actual name of your branch):

<mark>git rebase origin/your-branch</mark>

This command applies your local commits on top of the changes fetched from the remote branch. If conflicts arise, Git will pause the rebase process and ask you to resolve them. Alternatively, you can use the interactive rebase to review and modify commits during the rebase:

<mark>git rebase -i origin/your-branch</mark>

This opens an editor where you can pick, squash, or edit individual commits.

3. Continue the rebase or resolve conflicts:

If conflicts occurred during the rebase, Git will prompt you to resolve them. After resolving conflicts, you can continue the rebase with:

<mark>git rebase –continue</mark>

If you decide to abort the rebase at any point, you can use:

<mark>git rebase –abort</mark>

4. Push the rebased branch to the remote repository:

After successfully rebasing your local branch, you may need to force-push the changes to the remote repository:

<mark>git push origin your-branch –force</mark>

Be cautious with force-pushing, especially if others are working with the same branch, as it rewrites the commit history. Now, your local branch is rebased onto the updated remote branch. Keep in mind that force-pushing should be done with caution, especially on shared branches, to avoid disrupting collaborative work.

PROGRAM 6

6. Collaboration and Remote Repositories
Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

PROGRAM:

To merge "feature-branch" into "master" and provide a custom commit message, you can use the following command:

git merge feature-branch -m "Your custom commit message"

Replace "Your custom commit message" with the actual message you want to use for the merge commit. This command performs the merge and creates a new commit on the "master" branch with the specified message. If there are no conflicts, Git will complete the merge automatically.

If conflicts occur during the merge, Git will pause and prompt you to resolve the conflicts manually. After resolving conflicts, you can continue the merge process with:

git merge –continue

Alternatively, you can use an interactive merge to modify the commit message before finalizing the merge:

git merge feature-branch --no-ff –e

This opens an editor where you can edit the commit message before completing the merge. Again, replace "feature-branch" with the name of your actual feature branch.

PROGRAM 7

7. Git Tags and Releases

Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

PROGRAM:

To create a lightweight Git tag named "v1.0" for a specific commit in your local repository, you can use the following command:

git tag v1.0 <commit_hash>

Replace **<commit_hash>** with the actual hash of the commit for which you want to create the tag.

For example, if you want to tag the latest commit, you can use the following:

git tag v1.0 HEAD

This creates a lightweight tag pointing to the specified commit. Lightweight tags are simply pointers to specific commits and contain only the commit checksum.

If you want to push the tag to a remote repository, you can use:

git push origin v1.0

This command pushes the tag named "v1.0" to to the remote repository. Keep in mind that Git tags, by default, are not automatically pushed to remotes, so you need to explicitly push them if needed.

PROGRAM 8

8. Advanced Git Operations
Write the command to cherry-pick a range of commits from "source-branch" to the current branch.

PROGRAM:

To cherry-pick a range of commits from "source-branch" to the current branch, you can use the following command:

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

Replace <start-commit> and <end-commit> with the commit hashes or references that define the range of commits you want to cherry-pick. The ^ (caret) symbol is used to exclude the starting commit itself from the range.

For example, if you want to cherry-pick the commits from commit A to commit B (excluding A) from "source-branch" to the current branch, you would run:

git cherry-pick A^..B

After running this command, Git will apply the specified range of commits onto your current branch. If there are any conflicts, Git will pause the cherry-pick process and ask you to resolve them. After resolving conflicts, you can continue the cherry-pick with:

git cherry-pick –continue

If you encounter issues and need to abort the cherry-pick operation, you can use:

git cherry-pick –abort

PROGRAM 9

9. Analyzing 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?

PROGRAM:

To view the details of a specific commit, including the author, date, and commit message, you can use the following Git command:

git show <commit-id>

Replace <commit-id> with the actual commit hash or commit reference of the commit you want to inspect.

For example:

git show abc123

This command will display detailed information about the specified commit, including the author, date, commit message, and the changes introduced by that commit.

If you only want a more concise summary of the commit information (without the changes), you can use:

git log -n 1 --pretty=format:"%h - %an, %ar : %s" <commit-id>

This command displays a one-line summary of the commit, showing the abbreviated commit hash (%h), author name (%an), relative author date (%ar), and commit message (%s).

Remember to replace <commit-id> with the actual commit hash or reference you want to inspect.

PROGRAM 10

10. Analyzing and Changing Git History

Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023 12-31."

PROGRAM:

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31," you can use the following git log command with the --author and --since / --until options:

git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"

This command will display the commit history that meets the specified criteria. Adjust the author name and date range according to your requirements. The --since and --until options accept a variety of date and time format, providing flexibility in specifying the date range.

PROGRAM 11

11. Analyzing and Changing Git History

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

PROGRAM:

To display the last five commits in the repository's history, you can use the following git log command with the -n option:

git log -n 5

This command shows the latest five commits in the repository, with the most recent commit displayed at the top. Adjust the number after the -n option if you want to see a different number of commits.

If you want a more concise output, you can use the --oneline option:

git log -n 5 –oneline

This provides a one-line summary for each commit, including the abbreviated commit hash and the commit message.

PROGRAM 12

12. Analyzing and Changing Git History

Write the command to undo the changes introduced by the commit with the ID "abc123".

PROGRAM:

To undo the changes introduced by a specific commit with the ID "abc123," you can use the git revert command. The git revert command creates a new commit that undoes the changes made in a previous commit. Here's the command:

git revert abc123

Replace "abc123" with the actual commit hash or commit reference of the commit you want to undo. After running this command, Git will open a text editor for you to provide a commit message for the new revert commit.

Alternatively, if you want to completely remove a commit and all of its changes from the commit history, you can use the git reset command. However, keep in mind that using git reset can rewrite history and should be used with caution, especially if the commit has been pushed to a remote repository.

git reset --hard abc123

Again, replace "abc123" with the actual commit hash or commit reference. After using git reset --hard, your working directory will be modified to match the specified commit, discarding all commits made after it. Be cautious when using --hard as it is a forceful operation and can lead to data loss.

Assessment Details (both CIE and SEE)

The weight-age of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.
The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.
● Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
● Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
● Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
● Weightage to be given for neatness and submission of record/write-up on time.
● Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
● In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weight-age of 60% and the rest 40% for viva-voce.
● The suitable rubrics can be designed to evaluate each student's performance and learning ability.
● The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).
The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

SEE marks for the practical course are 50 Marks.

● SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
● The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.
● All laboratory experiments are to be included for practical examination.
● (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.

● Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
● Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
● General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)
● Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.
The minimum duration of SEE is 02 hours

Suggested Learning Resources:

Version Control with Git, 3rd Edition, by Prem Kumar Ponuthorai, Jon Loeliger Released October 2022, Publisher(s): O'Reilly Media, Inc.
● Pro Git book, written by Scott Chacon and Ben Straub and published by Address, https://gitscm. com/book/en/v2
● https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_0130944433473699842782_shared /overview
● https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01330134712177459211926_share d/overview