

ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 560 028



A T M E
College of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

(ACADEMIC YEAR 2023-24)

LABORATORY MANUAL

SUBJECT: PROJECT MANAGEMENT WITH GIT

SUB CODE: BCSL307A

SEMESTER: III-2026 CBCS Scheme

Composed By
Ms. Vidyashree H M
PROGRAMMER

Verified By
Mrs. Darshini M S
Ms. Yashaswini
Ms. Rakshitha N M
Ms. Kavitha S
FACULTY INCHARGE

Approved By
Dr. Puttegowda D
HOD, CSE

INSTITUTIONAL MISSION AND VISION

Objectives

- ☐ To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- ☐ To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- ☐ To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels
- ☐ To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- ☐ To cultivate strong community relationships and involve the students and the staff in local community service.
- ☐ To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- ☐ Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

Sl.NO	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 "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 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".

INTRODUCTION TO GIT

Git is a distributed version control system[9] that tracks changes in any set of computer files, usually used for coordinating work among programmers who are collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different computers)

Characteristics

Git's design is a synthesis of Torvalds's experience with Linux in maintaining a large distributed development project, along with his intimate knowledge of file-system performance gained from the same project and the urgent need to produce a working system in short order. These influences led to the following implementation choices:

- **Strong support for non-linear development**

Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers. In Git, branches are very lightweight: a branch is only a reference to one commit.

- **Distributed development**

Like Darcs, BitKeeper, Mercurial, Bazaar, and Monotone, Git gives each developer a local copy of the full development history, and changes are copied from one such repository to another. These changes are imported as added development branches and can be merged in the same way as a locally developed branch.

- **Compatibility with existing systems and protocols**

Repositories can be published via Hypertext Transfer Protocol Secure (HTTPS), Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), or a Git protocol over either a plain socket or Secure Shell (ssh). Git also has a CVS server emulation, which enables the use of existing CVS clients and IDE plugins to access Git repositories. Subversion repositories can be used directly with git-svn.

- **Efficient handling of large projects**

Torvalds has described Git as being very fast and scalable,^[37] and performance tests done by Mozilla^[38] showed that it was an order of magnitude faster diffing large repositories than Mercurial and GNU Bazaar; fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server.

- **Cryptographic authentication of history**

The Git history is stored in such a way that the ID of a particular version (a *commit* in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed. The structure is similar to a Merkle tree, but with added data at the nodes and leaves.^[40] (Mercurial and Monotone also have this property.)

- **Toolkit-based design**

Git was designed as a set of programs written in C and several shell scripts that provide wrappers around those programs.^[41] Although most of those scripts have since been rewritten in C for speed and portability, the design remains, and it is easy to chain the components together.

- **Pluggable merge strategies**

As part of its toolkit design, Git has a well-defined model of an incomplete merge, and it has multiple algorithms for completing it, culminating in telling the user that it is unable to complete the merge automatically and that manual editing is needed.

- **Garbage accumulates until collected**

Aborting operations or backing out changes will leave useless dangling objects in the database. These are generally a small fraction of the continuously growing history of wanted objects. Git will automatically perform garbage collection when enough loose objects have been created in the repository. Garbage collection can be called explicitly using `git gc`.

Commonly used Linux commands

- **ls**-Display information about files in the current directory
- **pwd**-Displays the current working directory
- **mkdir**- Creates a directory
- **cd**- To navigate between different folders
- **cat**-displays the contents on terminal
- **clear**-Clear the terminal
- **rmdir**-Removes the empty directories from the directory list
- **cp**-Moves files from one directory to other another
- **mv**-Rename and Replace the files
- **rm**-Delete files
- **uname**-Command to get basic information about OS
- **locate**-Find a file in Database
- **touch**-create empty files
- **ln**-Create shortcuts to other files
- **ps**-Display the Processes in terminal
- **man**-Access manual for all Linux commands
- **grep**-Search for a specific string in output
- **echo**-Display active process on the terminal
- **wget**-Download files from net
- **whoami**-Create or update passwords for existing users
- **sort**-sort the file contents
- **cal**-View the calendar in terminal
- **df**-check the details of the file system
- **wc**-check the lines, word count, and characters in a file using different option

Program 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

- **git --version**
- **git init**
- **git status**
- **git add file names**
- **git commit -m "commit message"**
- **git log**

Program 2:

Creating and Managing Branches

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

- **git branch**
- **git checkout branch name**
- **git merge branch name**

Program 3:

Creating and Managing Branches

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

- **git branch**
- **git checkout branch name**
- **git stash save**
- **git stash pop**

Program 4

Collaboration and Remote Repositories

Clone a remote Git repository to your local machine.

- **git clone**
- **git pull**
- **git push**

Program 5

Collaboration and Remote Repositories

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

- **git clone**
- **git pull**
- **git push**
- **git rebase**

Program 6

Collaboration and Remote Repositories

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

- **git clone**
- **git pull**
- **git push**
- **git merge**

Program 7

Git Tags and Releases

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

- **git tag v1.0**

Program 8

Advanced Git Operations

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

- **git cherry-pick commit-id**

Program 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

- **git log**
- **git show**

Program 10

Analyzing and Changing Git History

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

- **git log --author=<Name>--since=<Date>/--after=<Date>**

Program 11

Analyzing and Changing Git History

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

- **git log -n (any latest number of commits need to display)**

Program 12

Analyzing and Changing Git History

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

- **git revert commit-id**