Dr. H N National College of Engineering

(Affiliated to VTU, Belagavi, Approved by AICTE, New Delhi and Govt. of Karnataka)
36B Cross, Jayanagar 7th block, Bengaluru – 560070 Tel: 080-2846 8196, email:

Project Management With Git

(BCS358C)

(As per Visvesvaraya Technological University Syllabus)
Compiled by:

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

2025-26

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Program Outcomes

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, 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 **Modern 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 Society**: 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 **Environment and Sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8 **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9 **Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10 **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.

11 **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.

12 **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 Specific Outcomes

PSO 1: Apply the skills of core computer science engineering, artificial intelligence, machine learning, deep learning to solve futuristic problems.

PSO 2: Demonstrate computer knowledge, practical competency and innovative ideas in computer science engineering, artificial intelligence and machine learning using machine tools and techniques.

Course Contents :

1. **Git Basics.**
2. **Git Installation**
3. **Git Basic Commands**
4. **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 stashedchanges.

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 theupdated 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.

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".

1. Git Basics

What is Git?

Git is a distributed version control system (VCS) that is widely used for tracking changes in source code during software development. It was created by Linus Torvalds in 2005 and has since become the de facto standard for version control in the software development industry.

Git allows multiple developers to collaborate on a project by providing a history of changes, facilitating the tracking of who made what changes and when. Here are some key concepts and features of Git:

1. Repository (Repo): A Git repository is a directory or storage location where your project's files and version history are stored. There can be a local repository on your computer and remote repositories on servers.
2. Commits: In Git, a commit is a snapshot of your project at a particular point in time. Each commit includes a unique identifier, a message describing the changes, and a reference to the previous commit.
3. Branches: Branches in Git allow you to work on different features or parts of your project simultaneously without affecting the main development line (usually called the "master" branch). Branches make it easy to experiment, develop new features, and merge changes back into the main branch when they are ready.
4. Pull Requests (PRs): In Git-based collaboration workflows, such as GitHub or GitLab, pull requests are a way for developers to propose changes and have them reviewed by their peers. This is a common practice for open-source and team-based projects.
5. Merging: Merging involves combining changes from one branch (or multiple branches) into another. When a branch's changes are ready to be incorporated into the main branch, you can merge them.
6. Remote Repositories: Remote repositories are copies of your project stored on a different server. Developers can collaborate by pushing their changes to a remote repository and pulling changes from it. Common remote repository hosting services include GitHub, GitLab, and Bitbucket.
7. Cloning: Cloning is the process of creating a copy of a remote repository on your local machine. This allows you to work on the project and make changes locally.
8. Forking: Forking is a way to create your copy of a repository, typically on a hosting platform like GitHub. You can make changes to your fork without affecting the original project and later create pull requests to contribute your changes back to the original repository.

Git is known for its efficiency, flexibility, and ability to handle both small and large-scale software projects. It is used not only for software development but also for managing and

tracking changes in various types of text-based files, including documentation and configuration files. Learning Git is essential for modern software development and collaboration.

Why we need git?

Git is an essential tool in software development and for many other collaborative and version-controlled tasks. Here are some key reasons why Git is crucial:

1. **Version Control:** Git allows you to track changes in your project's files over time. It provides a complete history of all changes, making it easy to understand what was done, when it was done, and who made the changes. This is invaluable for debugging, auditing, and collaboration.
2. **Collaboration:** Git enables multiple developers to work on the same project simultaneously without interfering with each other's work. It provides mechanisms for merging changes made by different contributors and resolving conflicts when they occur.
3. **Branching:** Git supports branching, which allows developers to create isolated environments for developing new features or fixing bugs. This is essential for managing complex software projects and experimenting with new ideas without affecting the main codebase.
4. **Distributed Development:** Git is a distributed version control system, meaning that every developer has a complete copy of the project's history on their local machine. This provides redundancy, facilitates offline work, and reduces the reliance on a central server.
5. **Backup and Recovery:** With Git, your project's history is distributed across multiple locations, including local and remote repositories. This provides redundancy and makes it easy to recover from accidental data loss or system failures.
6. **Code Review:** Git-based platforms like GitHub, GitLab, and Bitbucket provide tools for code review and collaboration. Developers can propose changes, comment on code, and discuss improvements, making it easier to maintain code quality.
7. **Open Source and Community Development:** Git has become the standard for open-source software development. It allows anyone to fork a project, make contributions, and create pull requests, which makes it easy for communities of developers to collaborate on a single codebase.
8. **Efficiency:** Git is designed to be fast and efficient. It only stores the changes made to files, rather than entire file copies, which results in small repository sizes and faster operations.
9. **History and Documentation:** Git's commit history and commit messages serve as a form of documentation. It's easier to understand the context and reasoning behind a change by looking at the commit history and associated messages.

10. **Customizability:** Git is highly configurable and extensible. You can set up hooks and scripts to automate workflows, enforce coding standards, and integrate with various tools.

In summary, Git is essential for tracking changes in your projects, facilitating collaboration among developers, and ensuring the integrity and version history of your code. Whether you're working on a personal project or as part of a large team, Git is a fundamental tool for modern software development and version control.

What is Version Control System (VCS)?

A Version Control System (VCS), also commonly referred to as a Source Code Management (SCM) system, is a software tool or system that helps manage and track changes to files and directories over time. The primary purpose of a VCS is to keep a historical record of all changes made to a set of files, allowing multiple people to collaborate on a project while maintaining the integrity of the codebase. There are two main types of VCS: centralized and distributed.

Centralized Version Control Systems (CVCS): In a CVCS, there is a single central repository that stores all the project files and their version history. Developers check out files from this central repository, make changes, and then commit those changes back to the central repository. Examples of CVCS include CVS (Concurrent Versions System) and Subversion (SVN).

Distributed Version Control Systems (DVCS): In a DVCS, every developer has a complete copy of the project's repository, including its full history, on their local machine. This allows developers to work independently, create branches for experimentation, and synchronize their changes with remote repositories. Git is the most well-known and widely used DVCS, but other DVCS options include Mercurial and Bazaar.

Key features and benefits of Version Control Systems include:

1. **History Tracking:** VCS systems maintain a complete history of changes, including who made the change, what was changed, and when it was changed. This makes it easy to review and understand the evolution of a project.
2. **Collaboration:** VCS allows multiple developers to work on the same project simultaneously. It provides mechanisms for merging changes made by different contributors and resolving conflicts when they occur.
3. **Branching and Isolation:** VCS systems support branching, allowing developers to create isolated environments for new features or bug fixes. This isolates changes and helps manage complex development tasks.
4. **Revert and Rollback:** If a mistake is made, it is possible to revert changes to a previous state or commit. This is essential for error correction and maintaining code quality.

5. **Backup and Recovery:** Project data is stored in multiple locations, providing redundancy and facilitating data recovery in case of accidental data loss or system failures.
6. **Documentation:** Commit messages and history serve as a form of documentation, explaining why a change was made, who made it, and when it was made.
7. **Efficiency:** VCS systems are designed to be fast and efficient. They typically store only the changes made to files, rather than entire file copies, which results in small repository sizes and faster operations.

VCS is a fundamental tool in software development and is used not only for source code but also for tracking changes in documentation, configuration files, and other types of text-based files. It is especially crucial for collaborative projects, allowing teams of developers to work together on the same codebase with confidence.

Git Life Cycle

The Git lifecycle refers to the typical sequence of actions and steps you take when using Git to manage your source code and collaborate with others. Here's an overview of the Git lifecycle:

1. **Initializing a Repository**:
    - To start using Git, you typically initialize a new repository (or repo) in your project directory. This is done with the command git init.
2. **Working Directory**:
    - Your project files exist in the working directory. These are the files you are actively working on.
3. **Staging**:
    - Before you commit changes, you need to stage them. Staging allows you to select which changes you want to include in the next commit. You use the git add command to stage changes selectively or all at once with git add ..
4. **Committing**:
    - After you've staged your changes, you commit them with a message explaining what you've done. Commits create snapshots of your project at that point in time. You use the git commit command to make commits, like git commit -m "Add new feature".
5. **Local Repository**:
    - Commits are stored in your local repository. Your project's version history is preserved there.
6. **Branching**:
    - Git encourages branching for development. You can create branches to work on new features, bug fixes, or experiments without affecting the main codebase. Use the git branch and git checkout commands for branching.
7. **Merging**:

- After you've completed work in a branch and want to integrate it into the main codebase, you perform a merge. Merging combines the changes from one branch into another. Use the git merge command.

8. **Remote Repository**:
   - For collaboration, you can work with remote repositories hosted on servers like GitHub, GitLab, or Bitbucket. These repositories serve as a central hub for sharing code.
9. **Pushing**:
   - To share your local commits with a remote repository, you push them using the git push command. This updates the remote repository with your changes.
10. **Pulling**:
    - To get changes made by others in the remote repository, you pull them to your local repository with the git pull command. This ensures that your local copy is up to date.
11. **Conflict Resolution**:
    - Conflicts can occur when multiple people make changes to the same part of a file. Git will inform you of conflicts, and you must resolve them by editing the affected files manually.
12. **Collaboration**:
    - Developers can collaborate by pushing, pulling, and making pull requests in a shared remote repository. Collaboration tools like pull requests are commonly used on platforms like GitHub and GitLab.
13. **Tagging and Releases**:
    - You can create tags to mark specific points in the project's history, such as version releases. Tags are useful for identifying significant milestones.
14. **Continuous Cycle**:
    - The Git lifecycle continues as you repeat these steps over time to manage the ongoing development and evolution of your project. This cycle supports collaborative and agile software development.

The Git lifecycle allows for effective version control, collaboration, and the management of complex software projects. It provides a structured approach to tracking and sharing changes, enabling multiple developers to work together on a project with minimal conflicts and a clear history of changes.

2. Git Installation

To install Git on your computer, you can follow the steps for your specific operating system:

1. Installing Git on Windows:

a. **Using Git for Windows (Git Bash):**

- Go to the official Git for Windows website: https://gitforwindows.org/
- Download the latest version of Git for Windows.
- Run the installer and follow the installation steps. You can choose the default settings for most options.

b. **Using GitHub Desktop (Optional):**

- If you prefer a graphical user interface (GUI) for Git, you can also install GitHub Desktop, which includes Git. Download it from https://desktop.github.com/ and follow the installation instructions.

2. Installing Git from Source (Advanced):

- If you prefer to compile Git from source, you can download the source code from the official Git website (https://git-scm.com/downloads) and follow the compilation instructions provided there. This is usually only necessary for advanced users.

After installation, you can open a terminal or command prompt and verify that Git is correctly installed by running the following command:

```
$ git --version
```

If Git is installed successfully, you will see the Git version displayed in the terminal. You can now start using Git for version control and collaborate on software development projects.

How to Configure the Git?

Configuring Git involves setting up your identity (your name and email), customizing Git options, and configuring your remote repositories. Git has three levels of configuration: system, global, and repository-specific. Here's how you can configure Git at each level:

1. System Configuration:

- System-level configuration affects all users on your computer. It is typically used for site-specific configurations and is stored in the /etc/gitconfig file.

To set system-level configuration, you can use the git config command with the --system flag (usually requires administrator privileges). For example:

```
$ git config --system user.name "Your Name"
$ git config --system user.email "your.email@example.com"
```

1. Global Configuration:

- Global configuration is specific to your user account and applies to all Git repositories on your computer. This is where you usually set your name and email.

To set global configuration, you can use the git config command with the --global flag. For example:

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

You can also view your global Git configuration by using:

```
$ git config --global --list
```

3. Git Commands List

Git is a popular version control system used for tracking changes in software development projects. Here's a list of common Git commands along with brief explanations:

1. **git init**: Initializes a new Git repository in the current directory.
2. **git clone <repository URL>**: Creates a copy of a remote repository on your local machine.
3. **git add <file>**: Stages a file to be committed, marking it for tracking in the next commit.
4. **git commit -m "message"**: Records the changes you've staged with a descriptive commit message.
5. **git status**: Shows the status of your working directory and the files that have been modified or staged.
6. **git log**: Displays a log of all previous commits, including commit hashes, authors, dates, and commit messages.
7. **git diff**: Shows the differences between the working directory and the last committed version.
8. **git branch**: Lists all branches in the repository and highlights the currently checked-out branch.
9. **git branch <branchname>**: Creates a new branch with the specified name.
10. **git checkout <branchname>**: Switches to a different branch.
11. **git merge <branchname>**: Merges changes from the specified branch into the currently checked-out branch.
12. **git pull**: Fetches changes from a remote repository and merges them into the current branch.
13. **git push**: Pushes your local commits to a remote repository.
14. **git remote**: Lists the remote repositories that your local repository is connected to.
15. **git fetch**: Retrieves changes from a remote repository without merging them.
16. **git reset <file>**: Unstages a file that was previously staged for commit.
17. **git reset --hard <commit>**: Resets the branch to a specific commit, discarding all changes after that commit.
18. **git stash**: Temporarily saves your changes to a "stash" so you can switch branches without committing or losing your work.
19. **git tag**: Lists and manages tags (usually used for marking specific points in history, like releases).
20. **git blame <file>**: Shows who made each change to a file and when.
21. **git rm <file>**: Removes a file from both your working directory and the Git repository.
22. **git mv <oldfile> <newfile>**: Renames a file and stages the change.

These are some of the most common Git commands, but Git offers a wide range of features and options for more advanced usage. You can use git --help followed by the command name to get more information about any specific command, e.g., git help commit.

Experiments On

Project Management with Git

(As Per VTU Syllabus)

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.

Solution:

To 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, follow these steps:

1. Open your terminal and navigate to the directory where you want to create the Git repository.
2. Initialize a new Git repository in that directory:

```
$ git init
```

3. Create a new file in the directory. For example, let's create a file named "my_file.txt." You can use any text editor or command-line tools to create the file.
4. Add the newly created file to the staging area. Replace "my_file.txt" with the actual name of your file:

```
$ git add my_file.txt
```

This command stages the file for the upcoming commit.

5. Commit the changes with an appropriate commit message. Replace "Your commit message here" with a meaningful description of your changes:

```
$ git commit -m "Your commit message here"
```

Your commit message should briefly describe the purpose or nature of the changes you made. For example:

```
$ git commit -m "Add a new file called my_file.txt"
```

After these steps, your changes will be committed to the Git repository with the provided commit message. You now have a version of the repository with the new file and its history stored in Git.

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users
$ cd gitise

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise
$ git init
Initialized empty Git repository in C:/Users/gitise/.git/

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ nano 1.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git add 1.txt
warning: in the working copy of '1.txt', LF will be replaced by CRLF the next ti
me Git touches it

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git commit -m "file 1"
[master (root-commit) c4c556b] file 1
 1 file changed, 1 insertion(+)
 create mode 100644 1.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git log
commit c4c556bdbc39b85716ae6fd81244cac36ae8c7d0 (HEAD -> master)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:54:39 2025 +0530

    file 1
```

Experiment 2.

Creating and Managing Branches:

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

Solution:

To create a new branch named "feature-branch," switch to the "master" branch, and merge the "feature-branch" into "master" in Git, follow these steps:

1. Make sure you are in the "master" branch by switching to it:

$ git checkout master

2. Create a new branch named "feature-branch" and switch to it:

$ git checkout -b feature-branch

This command will create a new branch called "feature-branch" and switch to it.

3. Make your changes in the "feature-branch" by adding, modifying, or deleting files as needed.
4. Stage and commit your changes in the "feature-branch":

$ git add .
$ git commit -m "Your commit message for feature-branch"

Replace "Your commit message for feature-branch" with a descriptive commit message for the changes you made in the "feature-branch."

5. Switch back to the "master" branch:

$ git checkout master

6. Merge the "feature-branch" into the "master" branch:

$ git merge feature-branch

This command will incorporate the changes from the "feature-branch" into the "master" branch.

Now, your changes from the "feature-branch" have been merged into the "master" branch. Your project's history will reflect the changes made in both branches

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git checkout -b feature-branch
Switched to a new branch 'feature-branch'

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git branch
* feature-branch
  master

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ nano 2.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git add 2.txt
warning: in the working copy of '2.txt', LF will be replaced by CRLF the next ti
me Git touches it

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git commit -m "File 2"
[feature-branch c31e341] File 2
 1 file changed, 1 insertion(+)
 create mode 100644 2.txt
```

```
$ git log
commit c31e3419b6ad1a83639588f024839e13adcba4f4 (HEAD -> feature-branch)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:58:42 2025 +0530

    File 2

commit c4c556bdbc39b85716ae6fd81244cac36ae8c7d0 (master)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:54:39 2025 +0530

    file 1

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git checkout master
Switched to branch 'master'

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ ls
1.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git merge feature-branch
Updating c4c556b..c31e341
Fast-forward
 2.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 2.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ ls
1.txt  2.txt
```

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git log
commit c31e3419b6ad1a83639588f024839e13adcba4f4 (HEAD -> master, feature-branch)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:58:42 2025 +0530

    File 2

commit c4c556bdbc39b85716ae6fd81244cac36ae8c7d0
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:54:39 2025 +0530

    file 1
```

Experiment 3.

Creating and Managing Branches

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

Solution:

To stash your changes, switch branches, and then apply the stashed changes in Git, you can use the following commands:

1. Stash your changes:

$ git stash save "Your stash message"

This command will save your changes in a stash, which acts like a temporary storage for changes that are not ready to be committed.

2. Switch to the desired branch:

$ git checkout target-branch

Replace "target-branch" with the name of the branch you want to switch to.

3. Apply the stashed changes:

$ git stash apply

This command will apply the most recent stash to your current working branch. If you have multiple stashes, you can specify a stash by name or reference (e.g., git stash apply stash@{2}) if needed.

If you want to remove the stash after applying it, you can use git stash pop instead of git stash apply.

Remember to replace "Your stash message" and "target-branch" with the actual message you want for your stash and the name of the branch you want to switch to.

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ ls
1.txt  2.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ nano 3.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git add 3.txt
warning: in the working copy of '3.txt', LF will be replaced by CRLF the next ti
me Git touches it

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ ls
1.txt  2.txt  3.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git stash save "file 3 stash"
Saved working directory and index state On master: file 3 stash

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git branch
  feature-branch
* master

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git checkout feature-branch
Switched to branch 'feature-branch'
```

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ ls
1.txt  2.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git stash apply
On branch feature-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   3.txt


ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ ls
1.txt  2.txt  3.txt
```

Experiment 8
Collaboration and Remote Repositories:

Clone a remote Git repository to your local machine.

Solution:

To clone a remote Git repository to your local machine, follow these steps:

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to clone the remote Git repository. You can use the cd command to change your working directory.
3. Use the git clone command to clone the remote repository. Replace <repository_url> with the URL of the remote Git repository you want to clone. For example, if you were cloning a repository from GitHub, the URL might look like this:

$ git clone <repository_url>

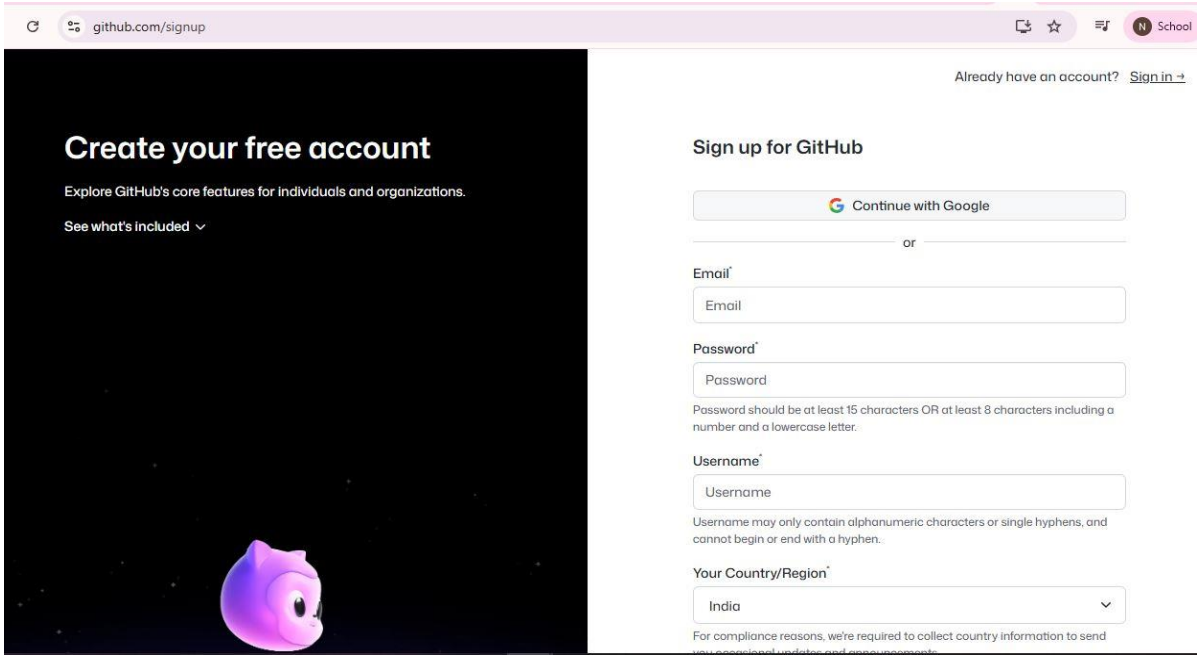
Here's a full example:

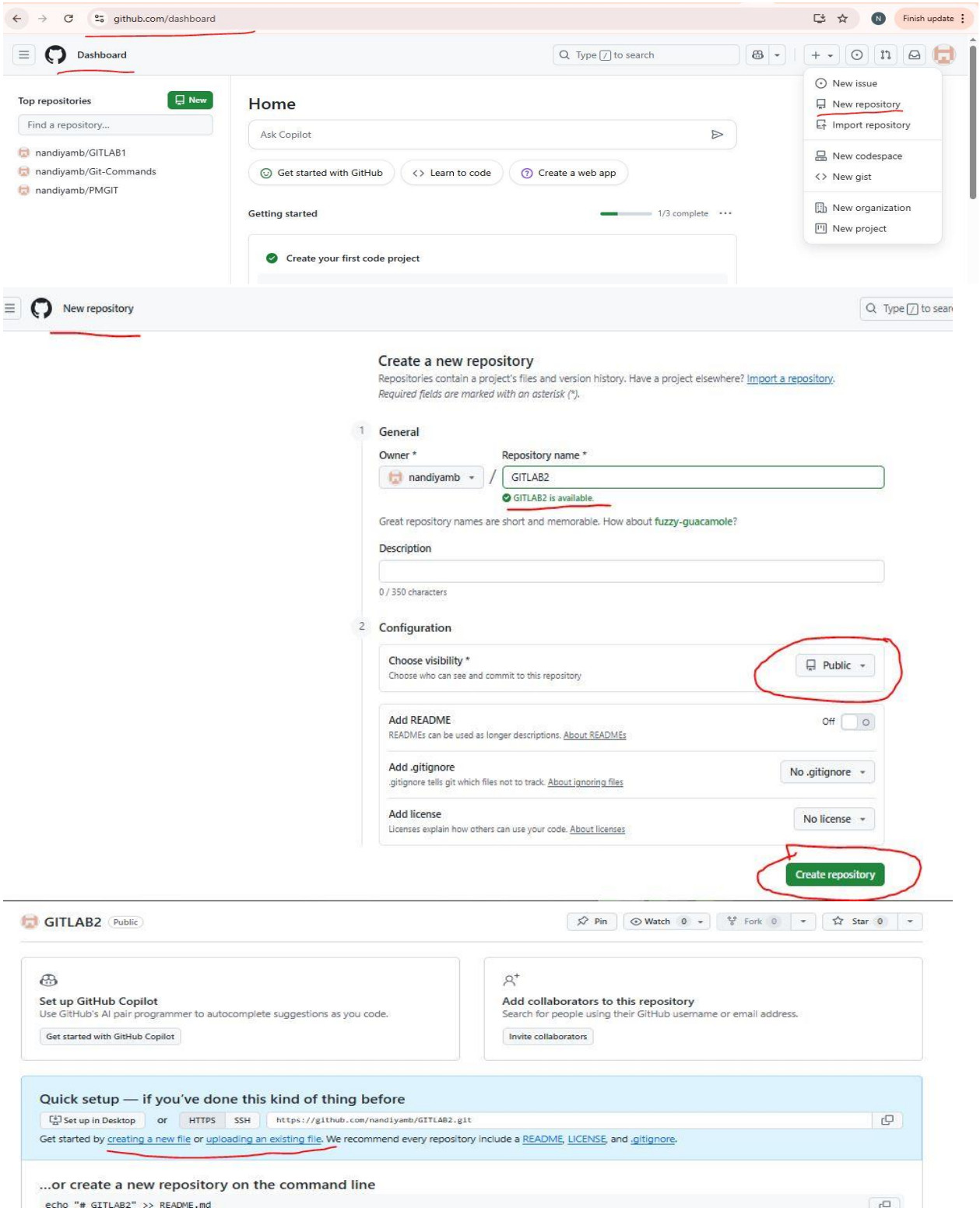
$ git clone https://github.com/username/repo-name.git

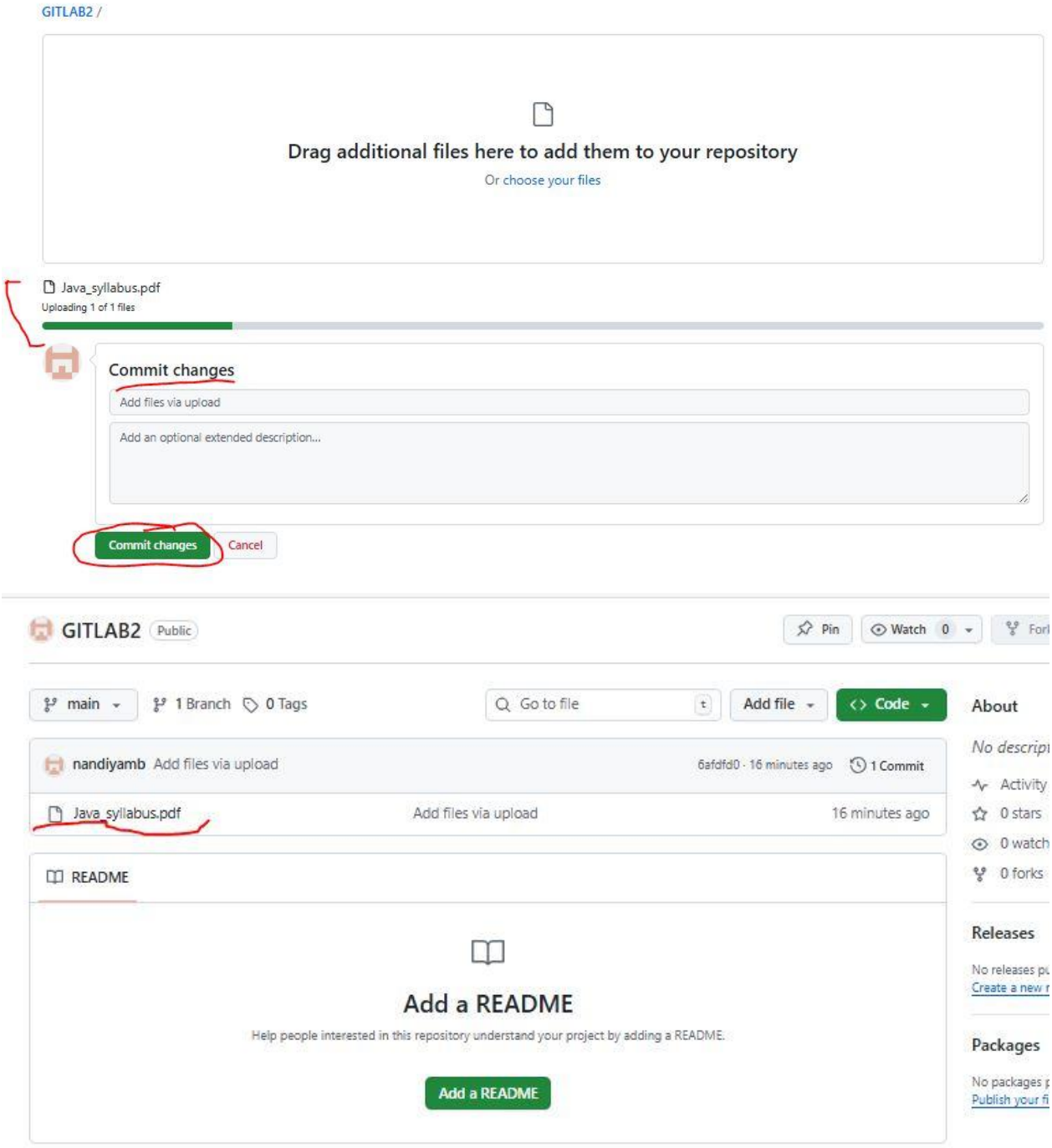
Replace https://github.com/username/repo-name.git with the actual URL of the repository you want to clone.

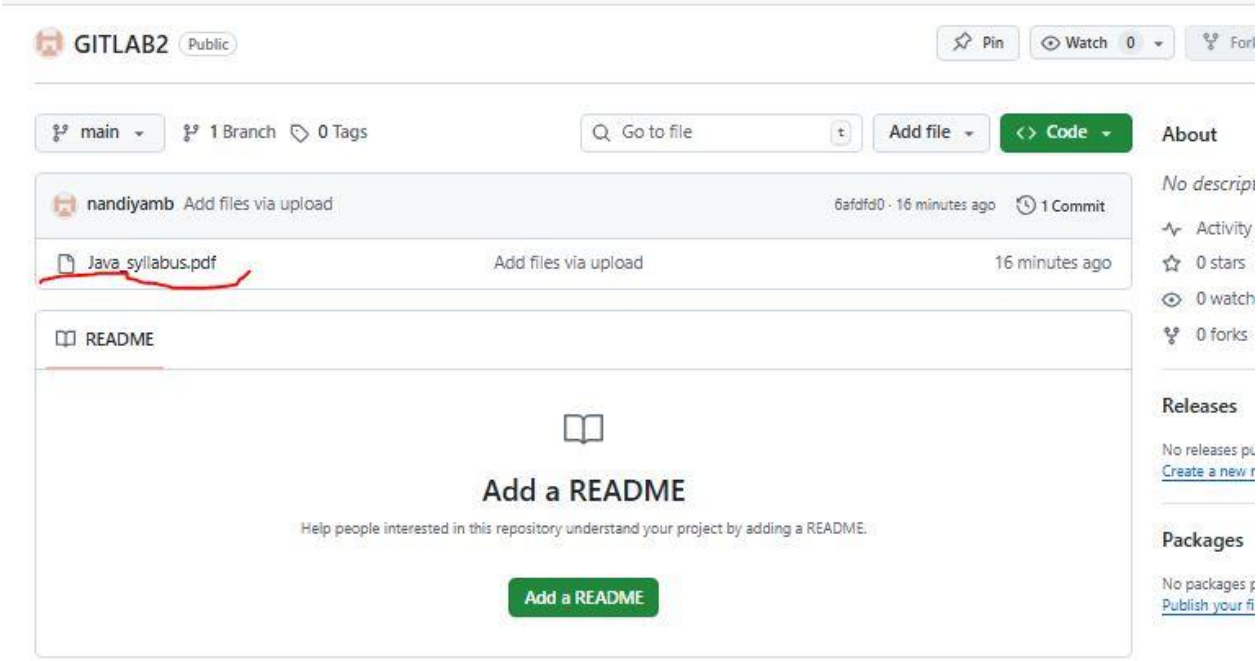
4. Git will clone the repository to your local machine. Once the process is complete, you will have a local copy of the remote repository in your chosen directory.

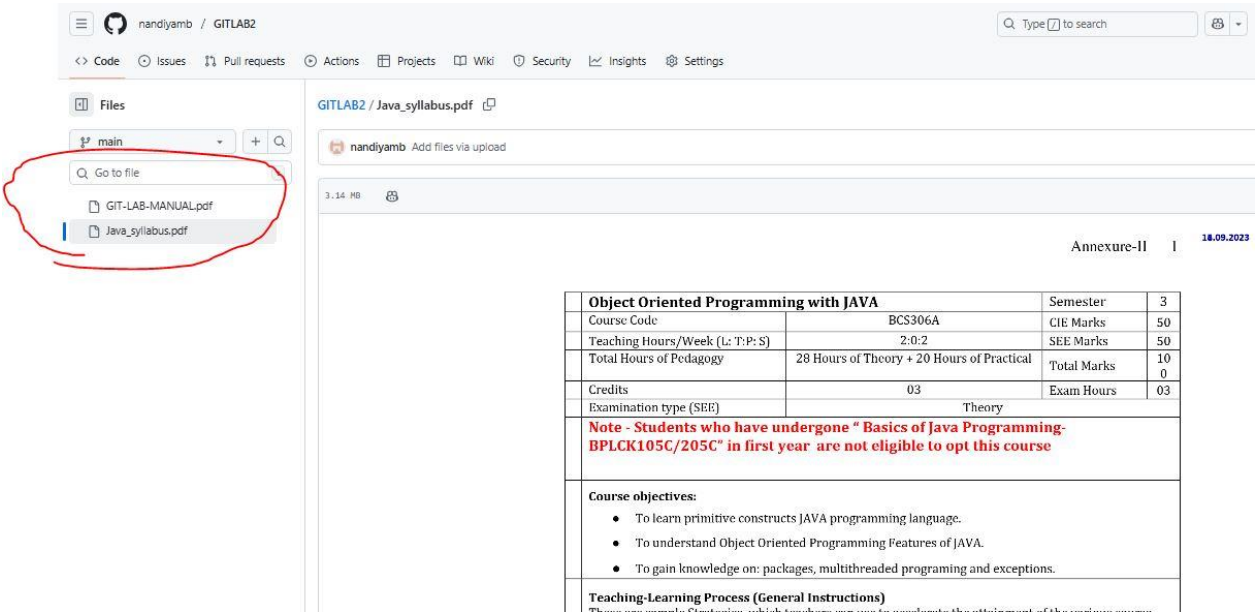
You can now work with the cloned repository on your local machine, make changes, and push those changes back to the remote repository as needed.

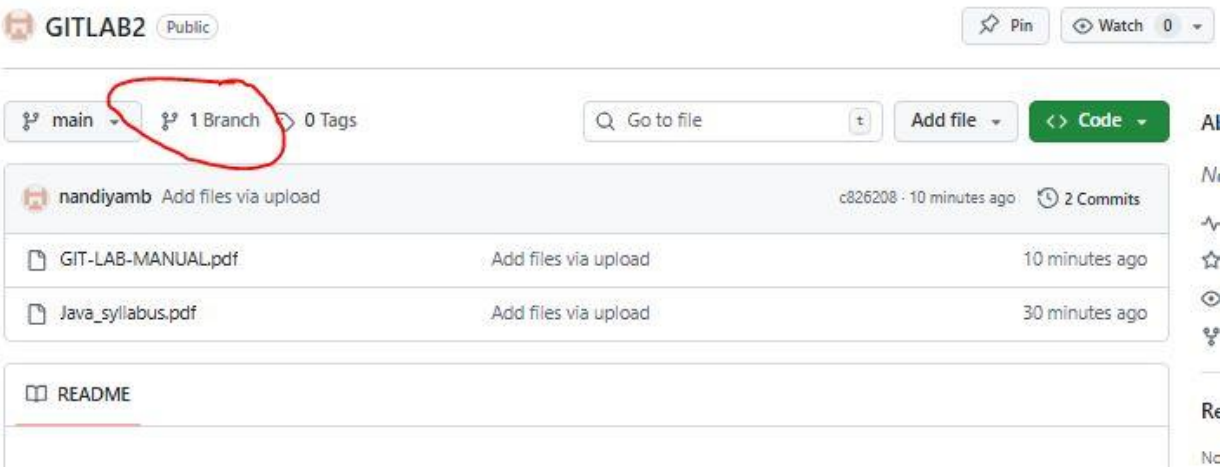


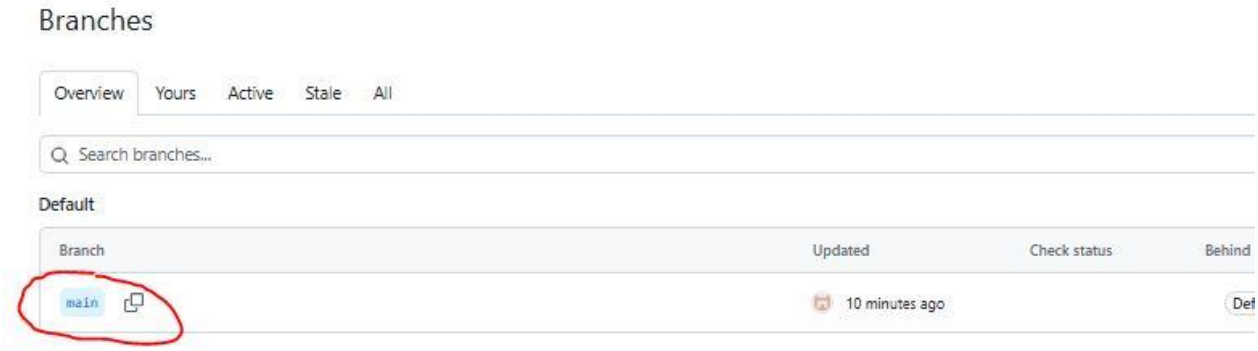

github.com/dashboard
Dashboard
Type / to search
Top repositories
New
Find a repository...
nandiyamb/GITLAB1
nandiyamb/Git-Commands
nandiyamb/PMGIT
Home
Ask Copilot
Get started with GitHub
Learn to code
Create a web app
Getting started
1/3 complete
Create your first code project
New issue
New repository
Import repository
New codespace
New gist
New organization
New project
New repository
Create a new repository
Repositories contain a project's files and version history. Have a project elsewhere? Import a repository.
Required fields are marked with an asterisk (*).
1 General
Owner *
Repository name *
nandiyamb
GITLAB2
GITLAB2 is available.
Great repository names are short and memorable. How about fuzzy-guacamole?
Description
0 / 350 characters
2 Configuration
Choose visibility *
Choose who can see and commit to this repository
Public
Add README
READMEs can be used as longer descriptions. About READMEs
Off
Add .gitignore
.gitignore tells git which files not to track. About ignoring files
No .gitignore
Add license
Licenses explain how others can use your code. About licenses
No license
Create repository
GITLAB2
Public
Pin
Watch 0
Fork 0
Star 0
Set up GitHub Copilot
Use GitHub's AI pair programmer to autocomplete suggestions as you code.
Get started with GitHub Copilot
Add collaborators to this repository
Search for people using their GitHub username or email address.
Invite collaborators
Quick setup — if you've done this kind of thing before
Set up in Desktop
or
HTTPS
SSH
https://github.com/nandiyamb/GITLAB2.git
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.
...or create a new repository on the command line
echo "# GITLAB2" >> README.md

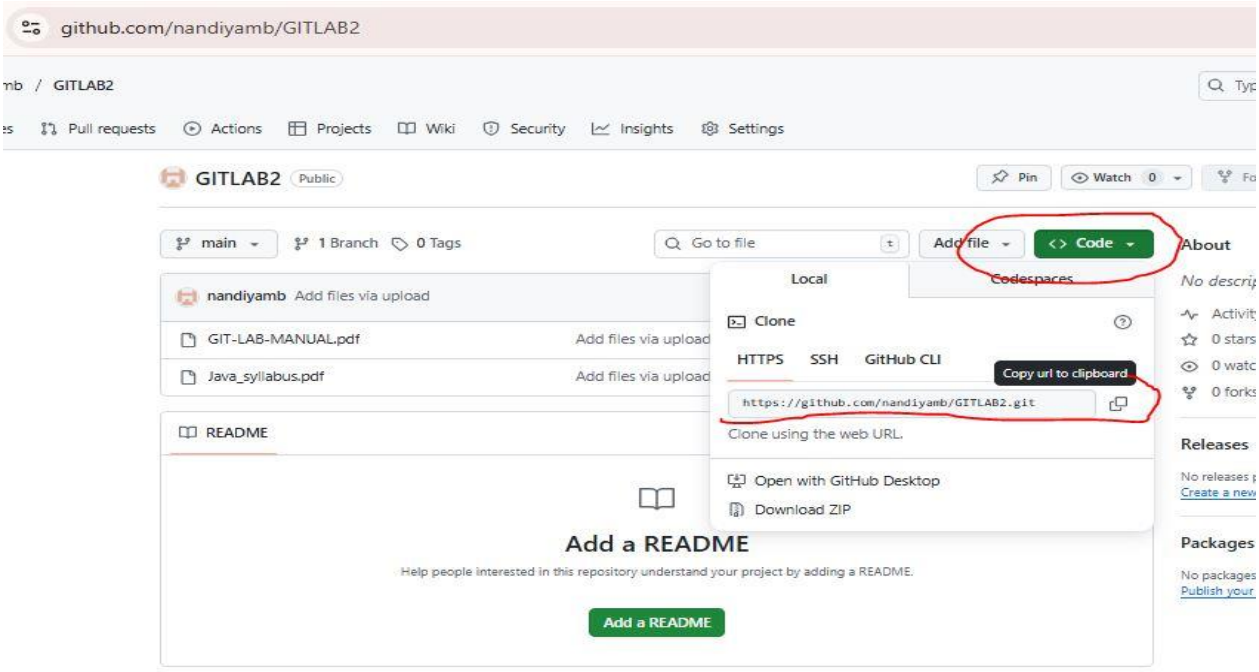

GITLAB2 /
Drag additional files here to add them to your repository
Or choose your files
Java_syllabus.pdf
Uploading 1 of 1 files
Commit changes
Add files via upload
Add an optional extended description...
Commit changes
Cancel
GITLAB2
Public
Pin
Watch 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
nandiyamb Add files via upload
6afdfd0 · 16 minutes ago
1 Commit
Java_syllabus.pdf
Add files via upload
16 minutes ago
README
Add a README
Help people interested in this repository understand your project by adding a README.
Add a README
Activity
0 stars
0 watch
0 forks
Releases
Packages

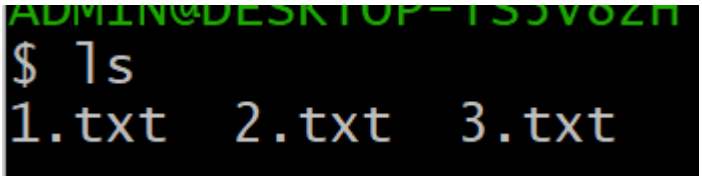

GITLAB2
Public
Pin
Watch 0
main
1 Branch
0 Tags
Go to file
Add file
Code
About
nandiyamb Add files via upload
6afdfd0 · 16 minutes ago
1 Commit
Java_syllabus.pdf
Add files via upload
16 minutes ago
README
Add a README
Help people interested in this repository understand your project by adding a README.
Add a README
Releases
Packages


nandiyamb / GITLAB2
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Files
main
Go to file
GIT-LAB-MANUAL.pdf
Java_syllabus.pdf
GITLAB2 / Java_syllabus.pdf
nandiyamb Add files via upload
Annexure-II
18.09.2023
Object Oriented Programming with JAVA
Semester 3
Course Code BCS306A
CIE Marks 50
Teaching Hours/Week (L: T:P: S) 2:0:2
SEE Marks 50
Total Hours of Pedagogy 28 Hours of Theory + 20 Hours of Practical
Total Marks 100
Credits 03
Exam Hours 03
Examination type (SEE) Theory
Note - Students who have undergone " Basics of Java Programming- BPLCK105C/205C" in first year are not eligible to opt this course
Course objectives:
To learn primitive constructs JAVA programming language.
To understand Object Oriented Programming Features of JAVA.
To gain knowledge on: packages, multithreaded programing and exceptions.
Teaching-Learning Process (General Instructions)

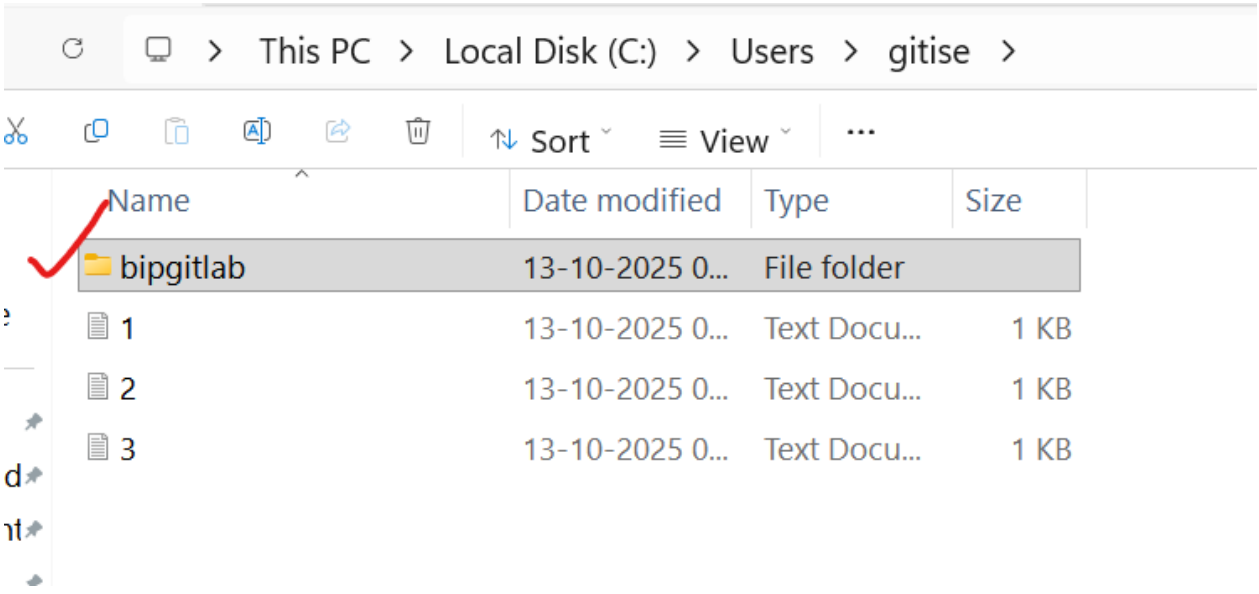

GITLAB2
Public
Pin
Watch 0
main
1 Branch
0 Tags
Go to file
Add file
Code
nandiyamb Add files via upload
c826208 · 10 minutes ago
2 Commits
GIT-LAB-MANUAL.pdf
Add files via upload
10 minutes ago
Java_syllabus.pdf
Add files via upload
30 minutes ago
README

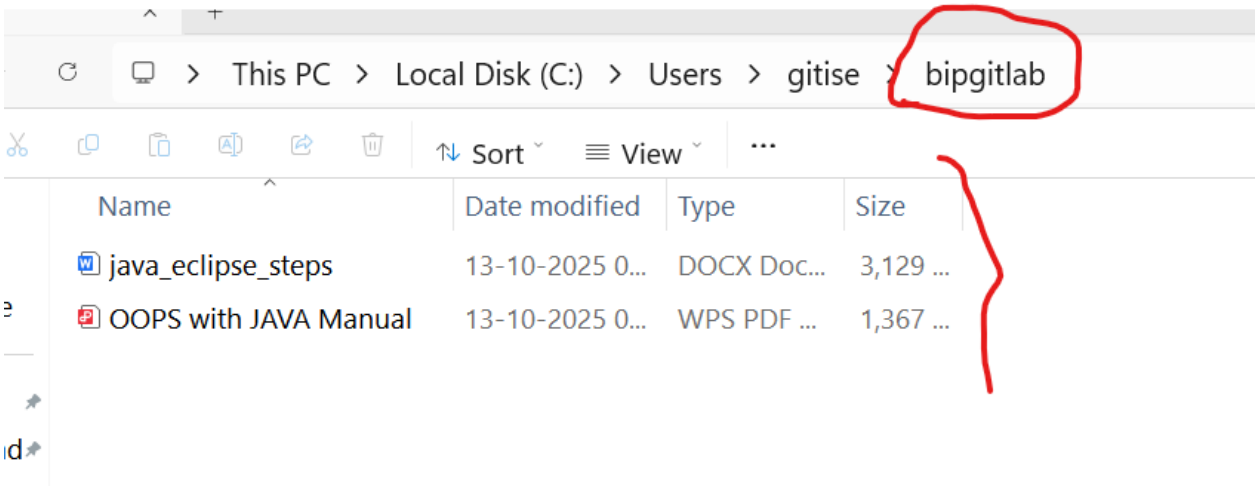


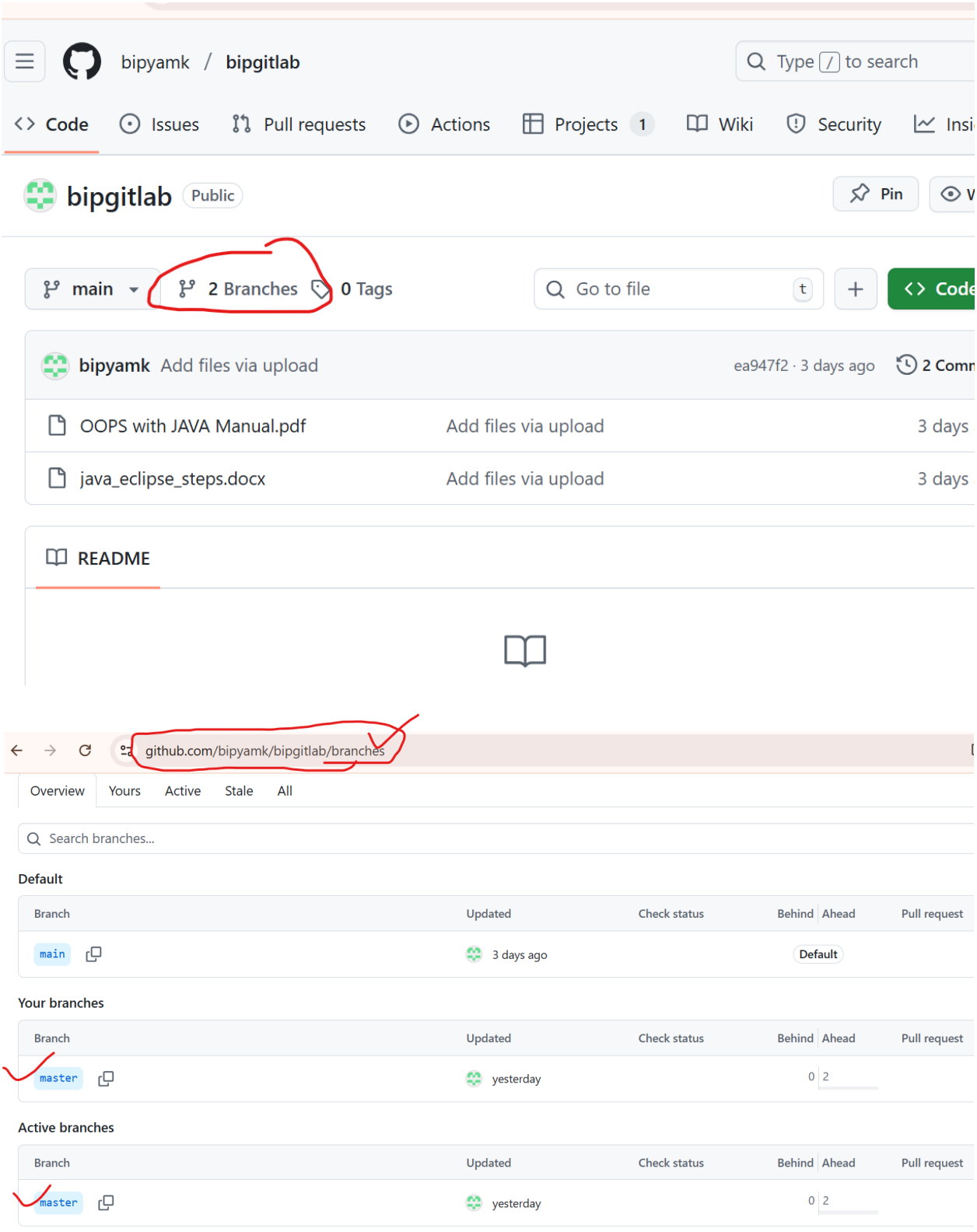





```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git clone https://github.com/bipyamk/bipgitlab.git
Cloning into 'bipgitlab'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), 4.21 MiB | 1.70 MiB/s, done.

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ ls
1.txt  2.txt  3.txt  bipgitlab/
```


This PC > Local Disk (C:) > Users > gitise >
Sort
View
Name
Date modified
Type
Size
bipgitlab
13-10-2025 0...
File folder
1
13-10-2025 0...
Text Docu...
1 KB
2
13-10-2025 0...
Text Docu...
1 KB
3
13-10-2025 0...
Text Docu...
1 KB


This PC > Local Disk (C:) > Users > gitise > bipgitlab
Sort
View
Name
Date modified
Type
Size
java_eclipse_steps
13-10-2025 0...
DOCX Doc...
3,129 ...
OOPS with JAVA Manual
13-10-2025 0...
WPS PDF ...
1,367 ...


bipyamk / bipgitlab
Type / to search
Code
Issues
Pull requests
Actions
Projects 1
Wiki
Security
bipgitlab
Public
Pin
main
2 Branches
0 Tags
Go to file
t
Code
bipyamk Add files via upload
ea947f2 · 3 days ago
2 Comm
OOPS with JAVA Manual.pdf
Add files via upload
3 days
java_eclipse_steps.docx
Add files via upload
3 days
README
github.com/bipyamk/bipgitlab/branches
Overview
Yours
Active
Stale
All
Search branches...
Default
Branch
Updated
Check status
Behind
Ahead
Pull request
main
3 days ago
Default
Your branches
Branch
Updated
Check status
Behind
Ahead
Pull request
master
yesterday
0
2
Active branches
Branch
Updated
Check status
Behind
Ahead
Pull request
master
yesterday
0
2

Experiment 5.

Collaboration and Remote Repositories:

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

Objective

The goal is to:

Bring your **local branch** up to date with the **latest changes** from the remote repository (e.g., `origin/main`).

Integrate those updates **cleanly and linearly** into your local work history by **rebasing** rather than merging.

1. Remote Repository

A **remote repository** is a version of your project stored on a server (e.g., GitHub, GitLab, Bitbucket).
It allows multiple collaborators to work on the same codebase.

Example remote name:

```
origin
```

2. Fetching

`git fetch` downloads all the **latest commits, branches, and tags** from the remote repository into your local repository — **without changing your working files or current branch.**

It updates references like:

```
origin/main
```

Example:

```
git fetch origin
```

This means: *"Get all the latest data from the remote called* `origin`*, but don't merge or rebase anything yet."*

3. Rebasing

Rebase means *"replay your local commits on top of another branch"*.

When you rebase your branch onto the latest remote version, Git:

Temporarily removes your local commits,

Moves your branch pointer to the updated remote branch,

Then **reapplies your commits one by one** on top of it.

So your work now appears as if it started **after** the remote's latest commits.

Solution:

To fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch in Git, follow these steps:

1. Open your terminal or command prompt.
2. Make sure you are in the local branch that you want to rebase. You can switch to the branch using the following command, replacing <branch-name> with your actual branch name:

$ git checkout <branch-name>

3. Fetch the latest changes from the remote repository. This will update your local repository with the changes from the remote without merging them into your local branch:

$ git fetch origin

Here, origin is the default name for the remote repository. If you have multiple remotes, replace origin with the name of the specific remote you want to fetch from.

4. Once you have fetched the latest changes, rebase your local branch onto the updated remote branch:

$ git rebase origin/<branch-name>

Replace <branch-name> with the name of the remote branch you want to rebase onto. This command will reapply your local commits on top of the latest changes from the remote branch, effectively incorporating the remote changes into your branch history.

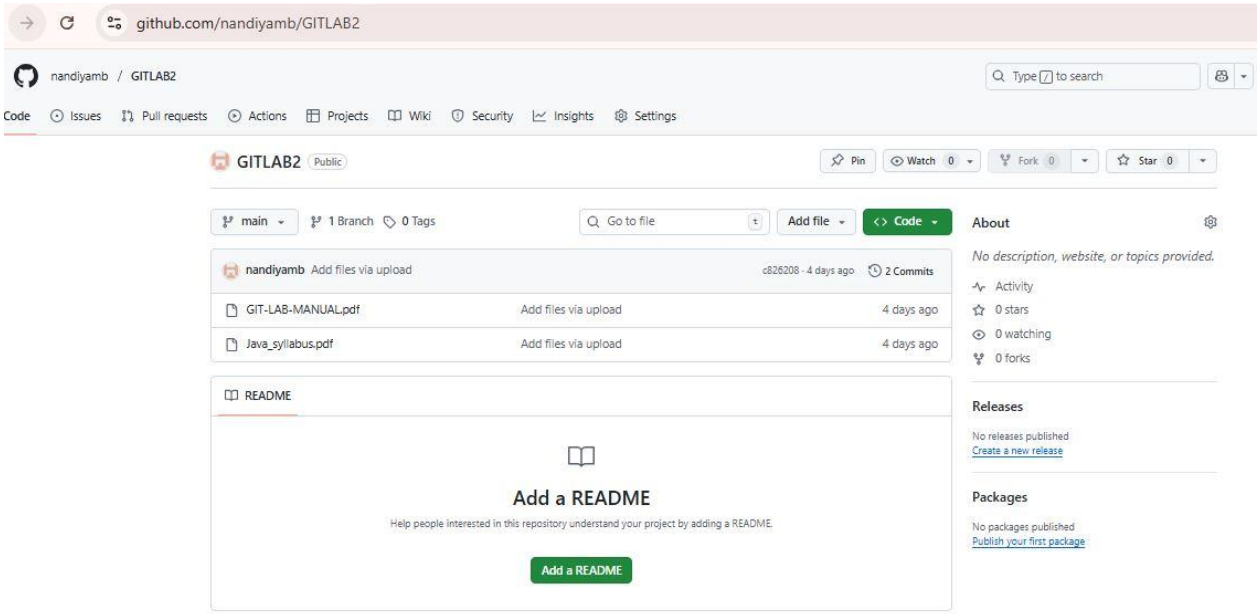
5. Resolve any conflicts that may arise during the rebase process. Git will stop and notify you if there are conflicts that need to be resolved. Use a text editor to edit the conflicting files, save the changes, and then continue the rebase with:

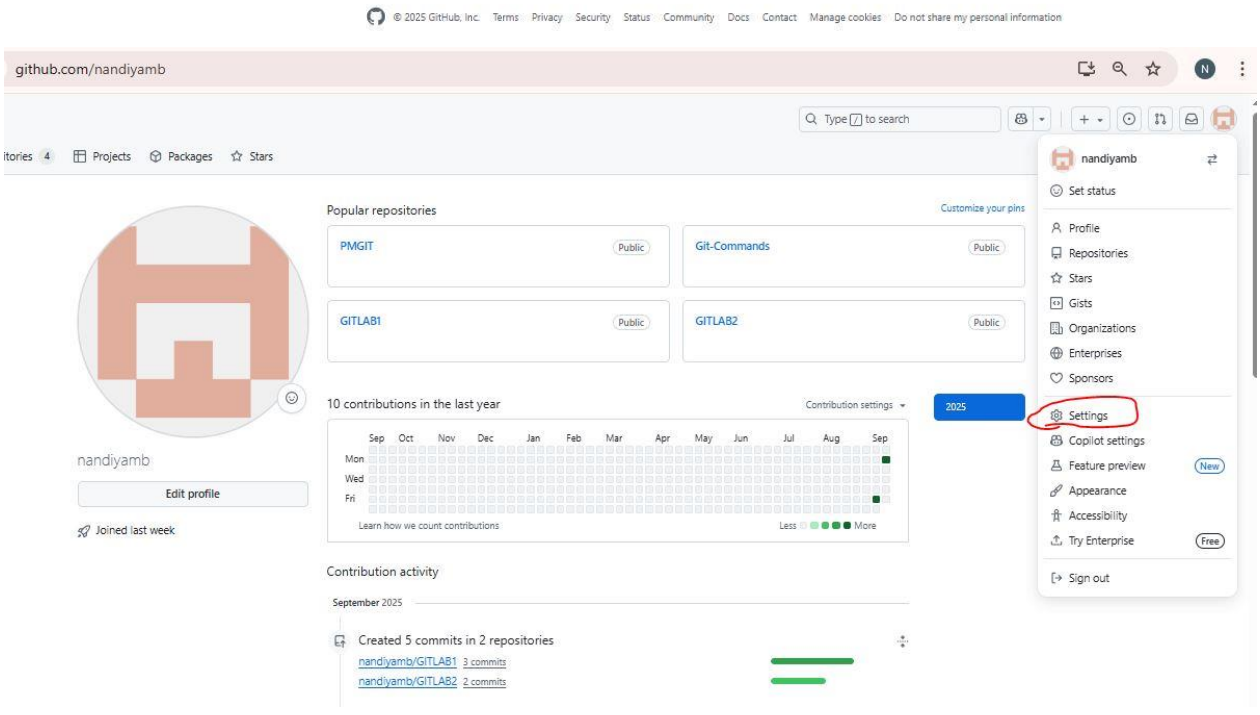
$ git rebase --continue

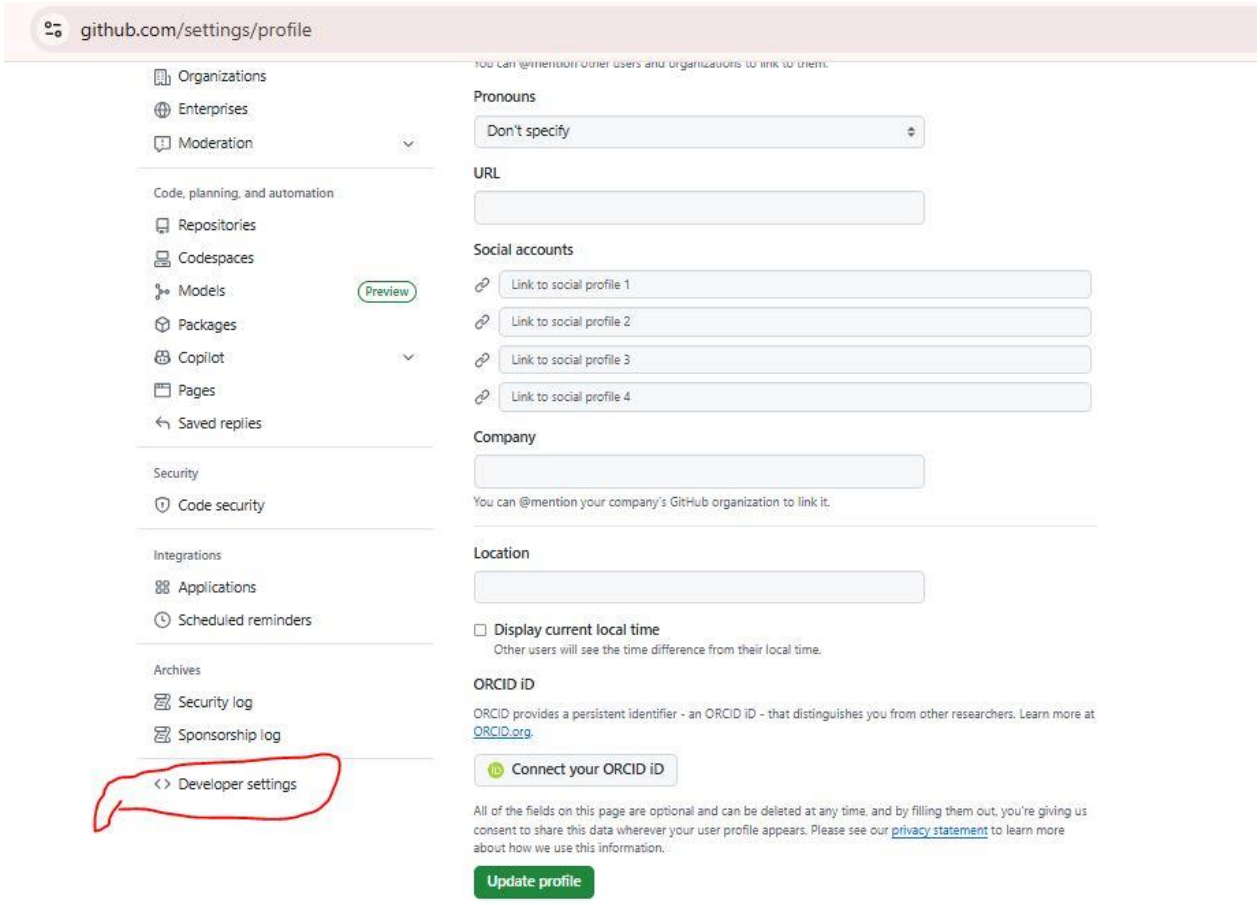
6. After resolving any conflicts and completing the rebase, you have successfully updated your local branch with the latest changes from the remote branch.
7. If you want to push your rebased changes to the remote repository, use the git push command. However, be cautious when pushing to a shared remote branch, as it can potentially overwrite other developers' changes:

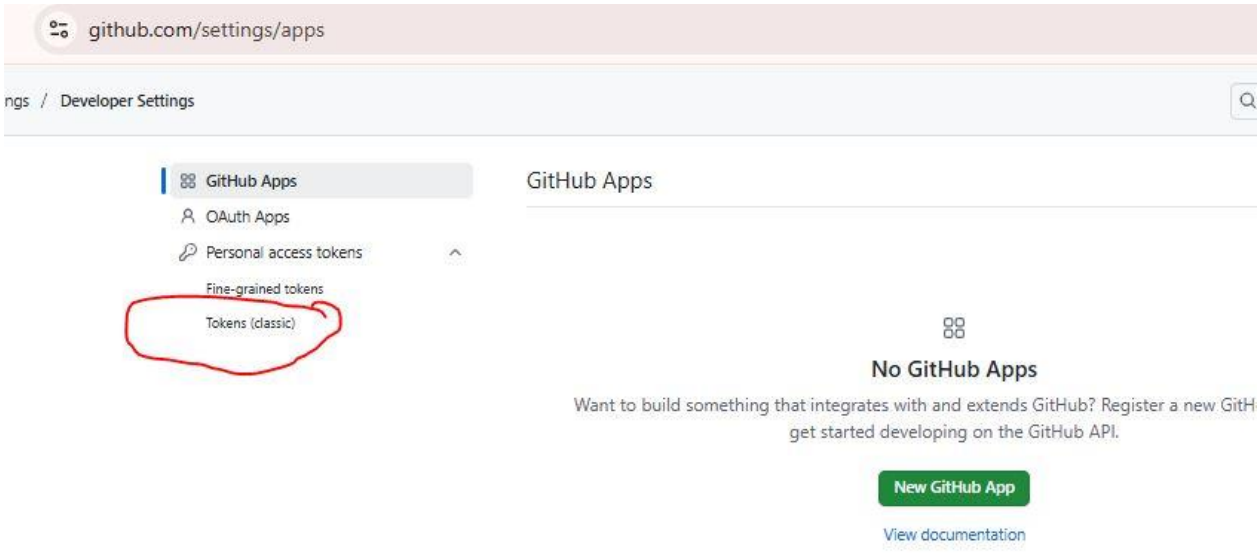
$ git push origin <branch-name>

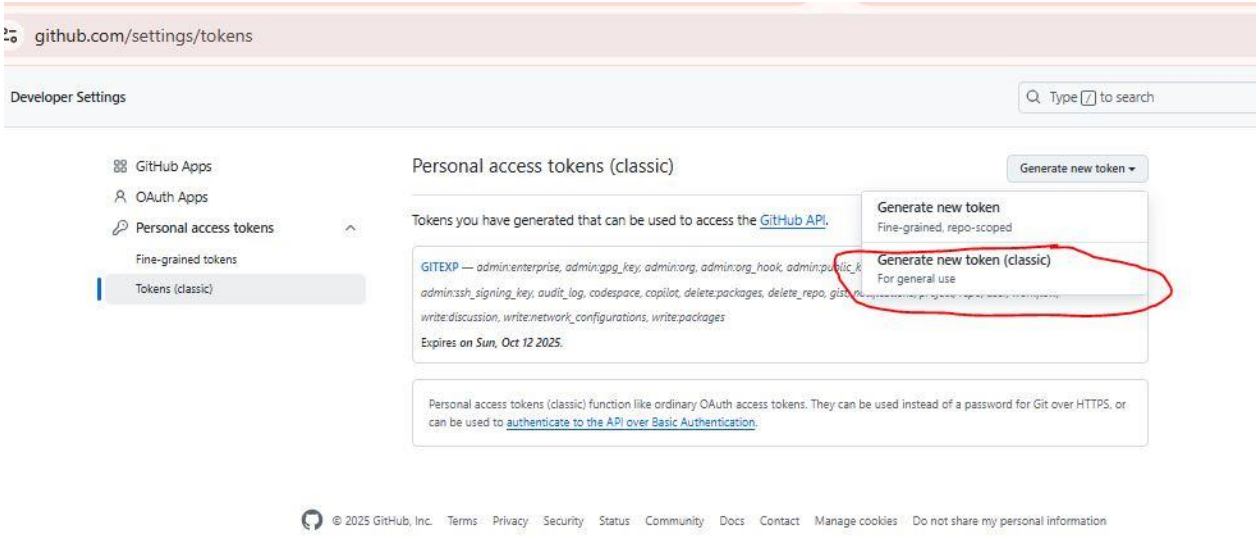
Replace <branch-name> with the name of your local branch. By following these steps, you can keep your local branch up to date with the latest changes from the remote repository and maintain a clean and linear history through rebasing.

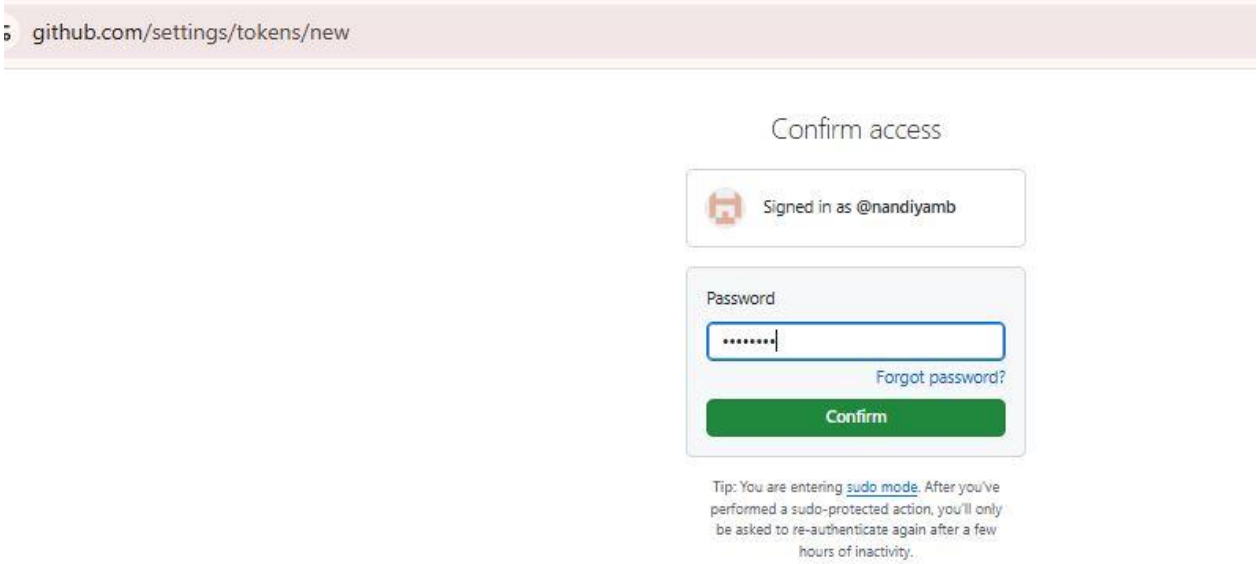


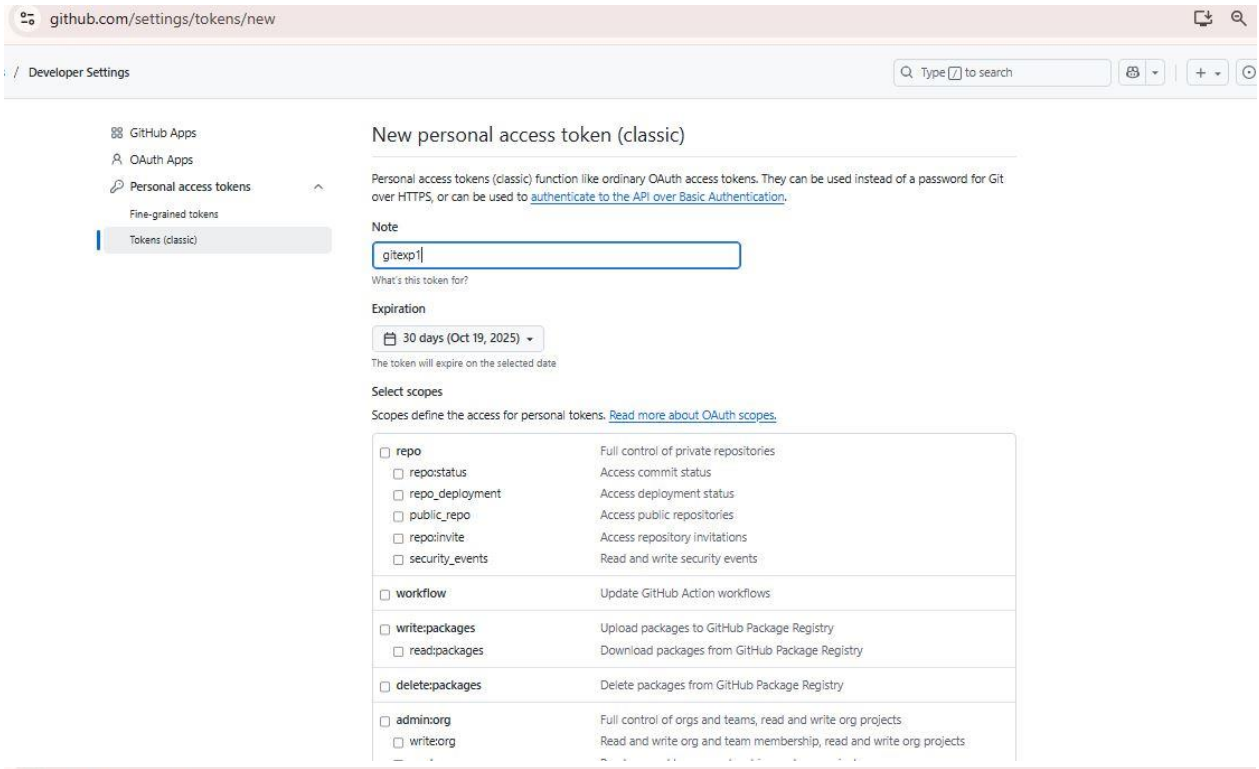


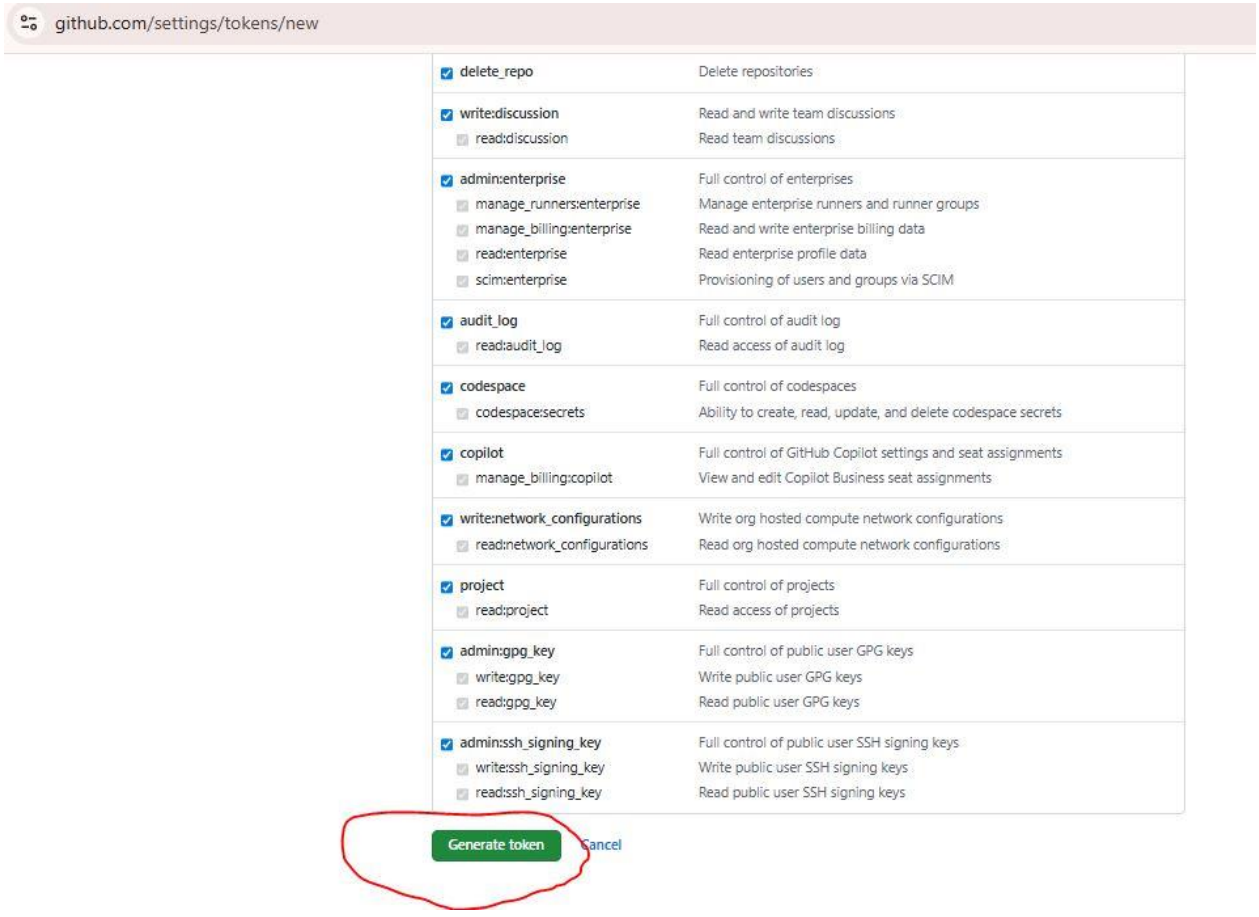

github.com/settings/profile
Organizations
Enterprises
Moderation
Code, planning, and automation
Repositories
Codespaces
Models
Preview
Packages
Copilot
Pages
Saved replies
Security
Code security
Integrations
Applications
Scheduled reminders
Archives
Security log
Sponsorship log
Developer settings
Pronouns
Don't specify
URL
Social accounts
Link to social profile 1
Link to social profile 2
Link to social profile 3
Link to social profile 4
Company
You can @mention your company's GitHub organization to link it.
Location
Display current local time
Other users will see the time difference from their local time.
ORCID iD
ORCID provides a persistent identifier - an ORCID iD - that distinguishes you from other researchers. Learn more at ORCID.org.
Connect your ORCID iD
All of the fields on this page are optional and can be deleted at any time, and by filling them out, you're giving us consent to share this data wherever your user profile appears. Please see our privacy statement to learn more about how we use this information.
Update profile

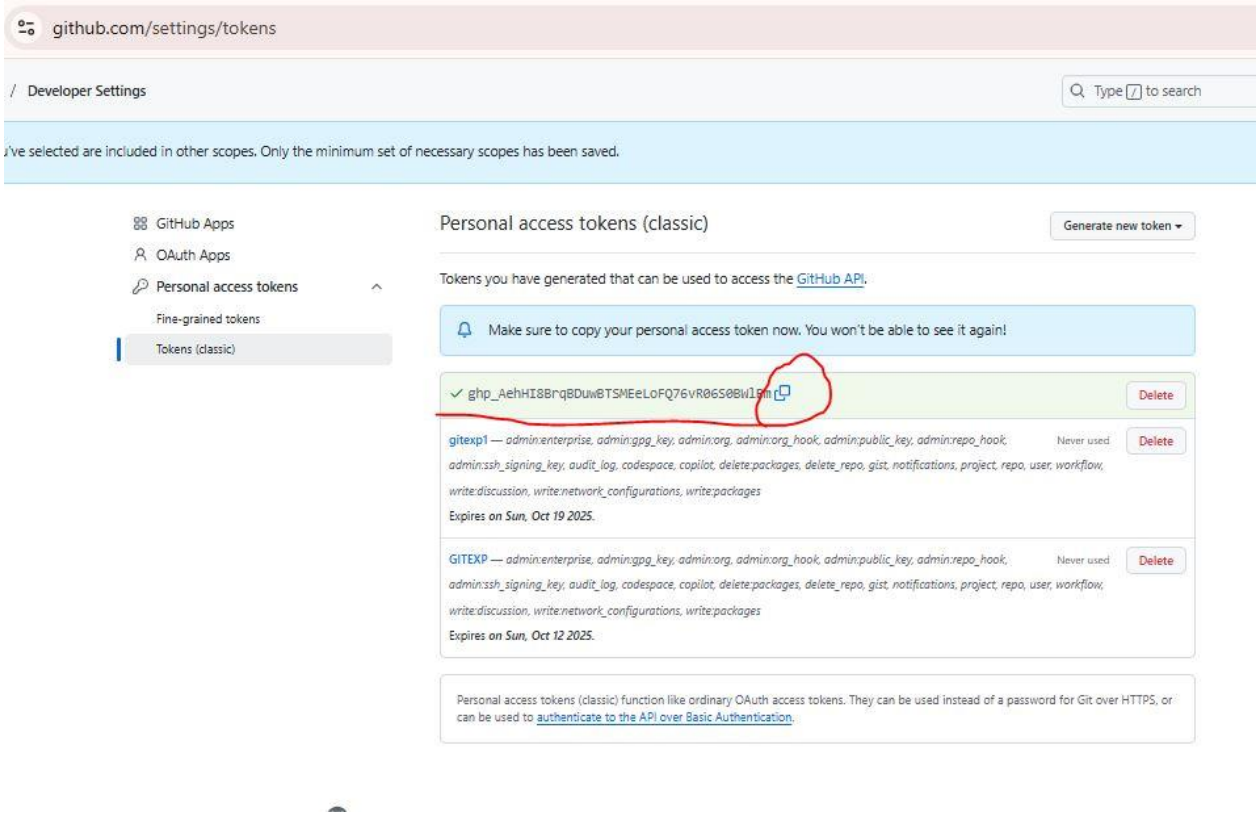

github.com/settings/apps
ngs / Developer Settings
GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Tokens (classic)
GitHub Apps
No GitHub Apps
Want to build something that integrates with and extends GitHub? Register a new GitH
get started developing on the GitHub API.
New GitHub App
View documentation


github.com/settings/tokens
Developer Settings
Type / to search
GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Tokens (classic)
Personal access tokens (classic)
Generate new token
Tokens you have generated that can be used to access the GitHub API.
Generate new token
Fine-grained, repo-scoped
Generate new token (classic)
For general use
GITEXP — admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_k
admin:ssh_signing_key, audit_log, codespace, copilot, delete:packages, delete_repo,
write:discussion, write:network_configurations, write:packages
Expires on Sun, Oct 12 2025.
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.
© 2025 GitHub, Inc. Terms Privacy Security Status Community Docs Contact Manage cookies Do not share my personal information


github.com/settings/tokens/new
Confirm access
Signed in as @nandiyamb
Password
Forgot password?
Confirm
Tip: You are entering sudo mode. After you've performed a sudo-protected action, you'll only be asked to re-authenticate again after a few hours of inactivity.


github.com/settings/tokens/new
Developer Settings
Type / to search
GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Tokens (classic)
New personal access token (classic)
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.
Note
gitexp1
What's this token for?
Expiration
30 days (Oct 19, 2025)
The token will expire on the selected date
Select scopes
Scopes define the access for personal tokens. Read more about OAuth scopes.
repo
Full control of private repositories
repo:status
Access commit status
repo_deployment
Access deployment status
public_repo
Access public repositories
repo:invite
Access repository invitations
security_events
Read and write security events
workflow
Update GitHub Action workflows
write:packages
Upload packages to GitHub Package Registry
read:packages
Download packages from GitHub Package Registry
delete:packages
Delete packages from GitHub Package Registry
admin:org
Full control of orgs and teams, read and write org projects
write:org
Read and write org and team membership, read and write org projects

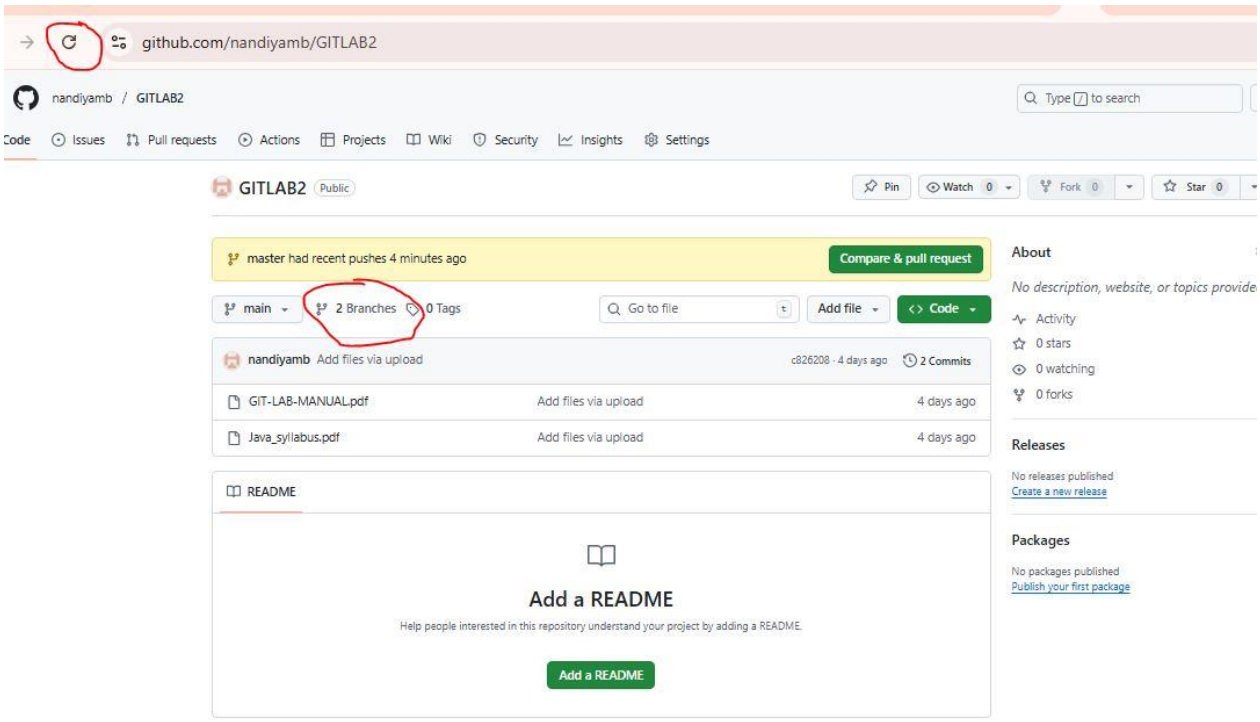

github.com/settings/tokens/new
delete_repo
Delete repositories
write:discussion
Read and write team discussions
read:discussion
Read team discussions
admin:enterprise
Full control of enterprises
manage_runners:enterprise
Manage enterprise runners and runner groups
manage_billing:enterprise
Read and write enterprise billing data
read:enterprise
Read enterprise profile data
scim:enterprise
Provisioning of users and groups via SCIM
audit_log
Full control of audit log
read:audit_log
Read access of audit log
codespace
Full control of codespaces
codespace:secrets
Ability to create, read, update, and delete codespace secrets
copilot
Full control of GitHub Copilot settings and seat assignments
manage_billing:copilot
View and edit Copilot Business seat assignments
write:network_configurations
Write org hosted compute network configurations
read:network_configurations
Read org hosted compute network configurations
project
Full control of projects
read:project
Read access of projects
admin:gpg_key
Full control of public user GPG keys
write:gpg_key
Write public user GPG keys
read:gpg_key
Read public user GPG keys
admin:ssh_signing_key
Full control of public user SSH signing keys
write:ssh_signing_key
Write public user SSH signing keys
read:ssh_signing_key
Read public user SSH signing keys
Generate token
Cancel

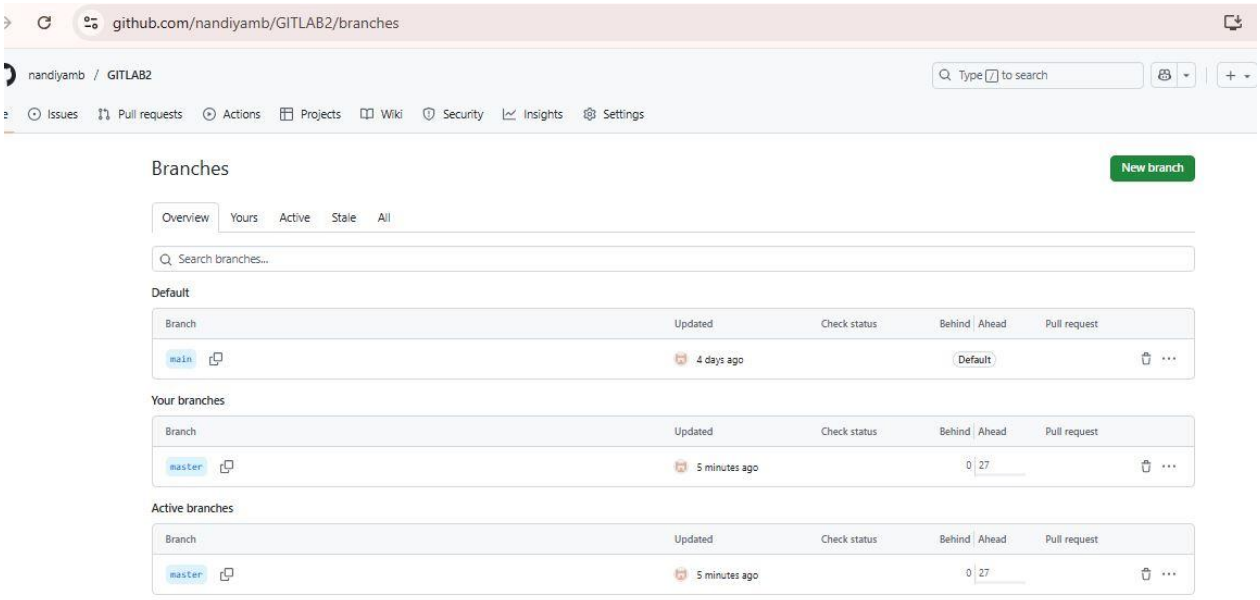


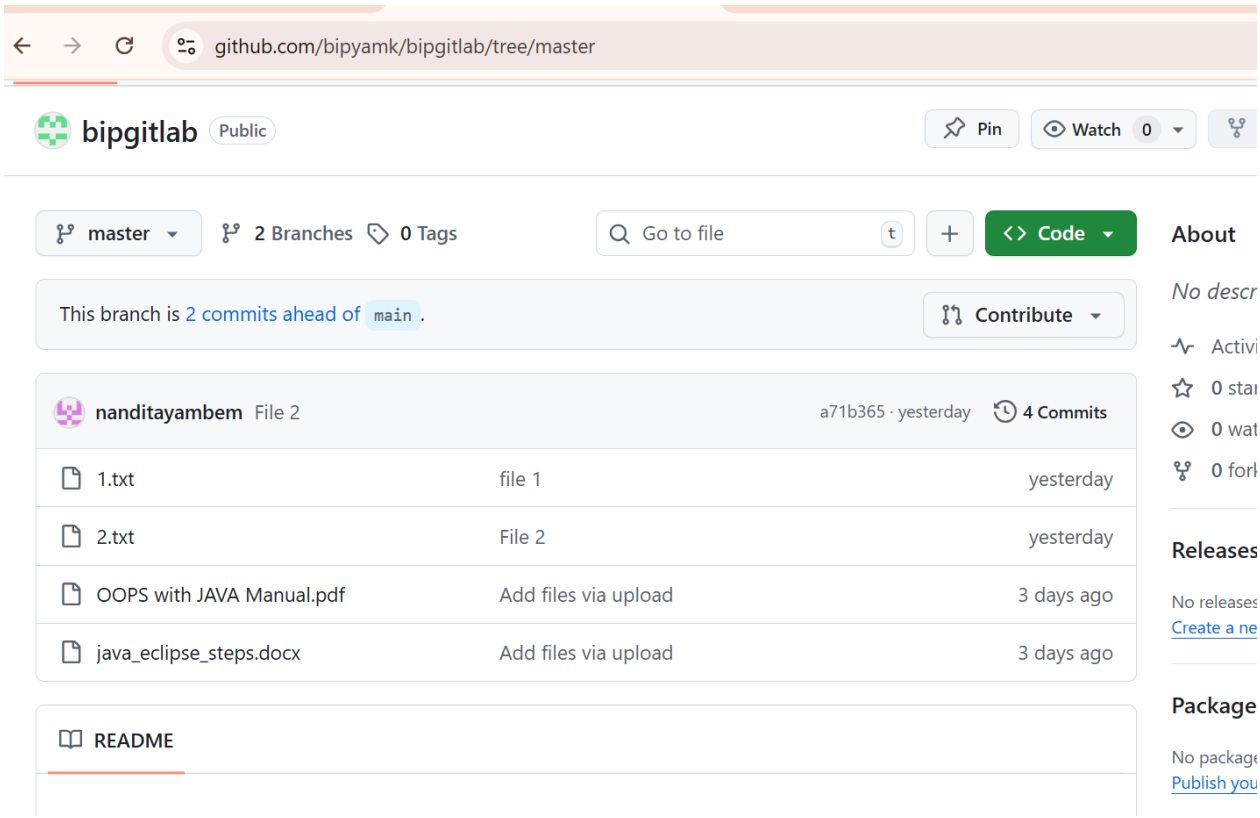

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise/bipgitlab (main)
$ cd ..

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git branch
  feature-branch
* master
```

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git checkout feature-branch
A       3.txt
Switched to branch 'feature-branch'

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git remote add origin https://github.com/bipyamk/bipgitlab.git

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git remote set-url origin https://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@git
hub.com/bipyamk/bipgitlab.git

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git branch
* feature-branch
  master

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git checkout master
A       3.txt
Switched to branch 'master'

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git fetch origin
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
```

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git stash
Saved working directory and index state WIP on master: c31e341 File 2

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git rebase origin/main
Successfully rebased and updated refs/heads/master.

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git stash pop
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   3.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        bipgitlab/

Dropped refs/stash@{0} (45735fd20f0102552e0f47aeda6bfa8a09b810f3)

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 638 bytes | 159.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/bipyamk/bipgitlab/pull/new/master
```


github.com/nandiyamb/GITLAB2
nandiyamb / GITLAB2
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
GITLAB2
Public
Pin
Watch 0
Fork 0
Star 0
master had recent pushes 4 minutes ago
Compare & pull request
main
2 Branches
0 Tags
Go to file
t
Add file
Code
nandiyamb Add files via upload
c826208 · 4 days ago
2 Commits
GIT-LAB-MANUAL.pdf
Add files via upload
4 days ago
Java_syllabus.pdf
Add files via upload
4 days ago
README
Add a README
Help people interested in this repository understand your project by adding a README.
Add a README
About
No description, website, or topics provide
Activity
0 stars
0 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package


github.com/nandiyamb/GITLAB2/branches
nandiyamb / GITLAB2
Type / to search
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Branches
New branch
Overview
Yours
Active
Stale
All
Search branches...
Default
Branch
Updated
Check status
Behind | Ahead
Pull request
main
4 days ago
Default
Your branches
Branch
Updated
Check status
Behind | Ahead
Pull request
master
5 minutes ago
0 | 27
Active branches
Branch
Updated
Check status
Behind | Ahead
Pull request
master
5 minutes ago
0 | 27


github.com/bipyamk/bipgitlab/tree/master
bipgitlab
Public
Pin
Watch 0
master
2 Branches
0 Tags
Go to file
t
Code
About
No descr
This branch is 2 commits ahead of main.
Contribute
nanditayambem File 2
a71b365 · yesterday
4 Commits
1.txt
file 1
yesterday
2.txt
File 2
yesterday
OOPS with JAVA Manual.pdf
Add files via upload
3 days ago
java_eclipse_steps.docx
Add files via upload
3 days ago
README
Releases
No releases
Create a ne
Package
No package
Publish you

.

Experiment 6
Collaboration and Remote Repositories:

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

Solution:

To merge the "feature-branch" into "master" in Git while providing a custom commit message for the merge, you can use the following command:

$ git checkout master
$ git merge feature-branch -m "Your custom commit message here"

Replace "Your custom commit message here" with a meaningful and descriptive commit message for the merge. This message will be associated with the merge commit that is created when you merge "feature-branch" into "master."

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git branch
  feature-branch
* main
  master

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ ls
 1.txt   7.txt                        a.txt
 2.txt   7a.txt                       bipgitlab/
 3.txt  'OOPS with JAVA Manual.pdf'   java_eclipse_steps.docx

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git checkout feature-branch
Switched to branch 'feature-branch'

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ nano aa.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ ls
1.txt  2.txt  aa.txt  bipgitlab/
```

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (feature-branch)
$ git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 13 commits.
  (use "git push" to publish your local commits)

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git merge master -m " I am main/master"
Already up to date.

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ ls
 1.txt   3.txt   7a.txt                        a.txt    bipgitlab/
 2.txt   7.txt  'OOPS with JAVA Manual.pdf'    aa.txt   java_eclipse_steps.docx
```

Experiment 7.

Git Tags and Releases:

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

Solution:

A Git tag is a reference that points to a specific commit in your repository's history, typically used to mark important milestones like version releases (e.g., v1.0, v2.1) or significant updates

Unlike branches, which are designed to move and evolve with new commits, tags are static pointers that remain fixed on the commit they were created for.

There are two main types of Git tags:

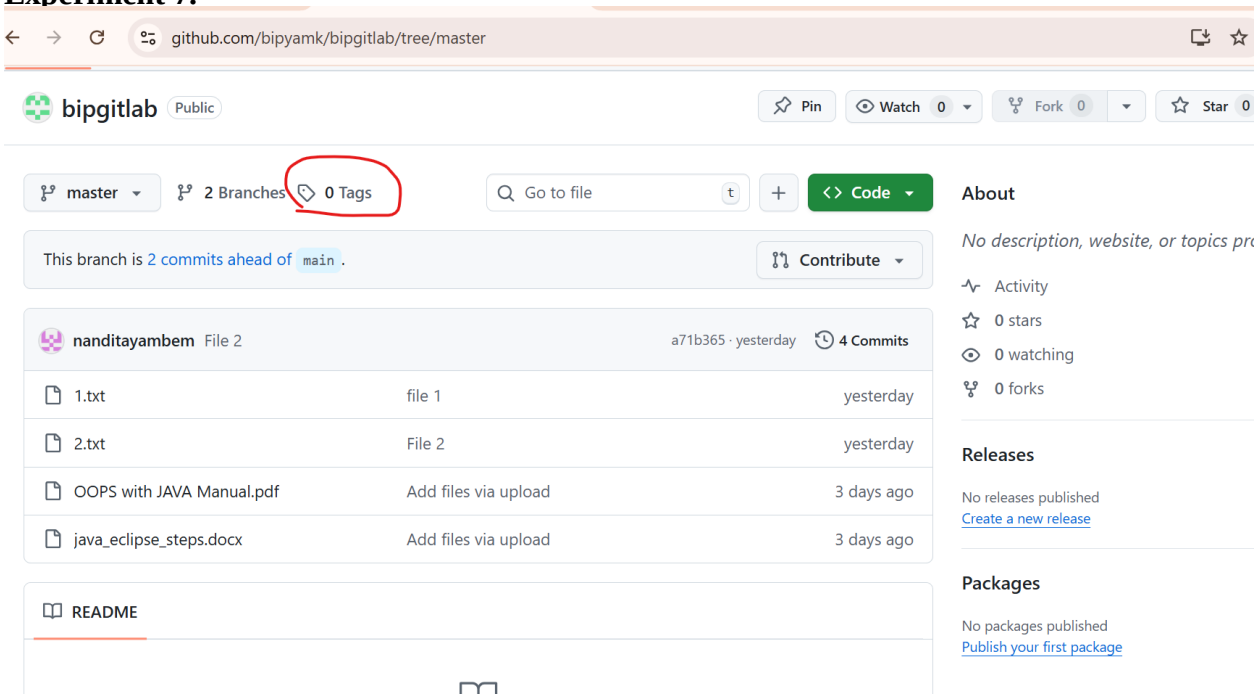
- **Lightweight Tags:** These are simple pointers to a specific commit, essentially just a name for a commit . They are quick to create and don't store any additional information beyond the commit reference.
- **Annotated Tags:** These are more comprehensive, storing not only the commit reference but also metadata such as the tagger's name, email, the date of tagging, and a tag message explaining the purpose of the tag.
- Annotated tags are generally preferred for marking releases as they provide more context and can be cryptographically signed for verification.

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

```
$ git tag v1.0
```

This command will create a lightweight tag called "v1.0" for the most recent commit in your current branch. If you want to tag a specific commit other than the most recent one, you can specify the commit's SHA-1 hash after the tag name. For example:

Experiment 7.




```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ ls
 1.txt   3.txt                          bipgitlab/
 2.txt  'OOPS with JAVA Manual.pdf'     java_eclipse_steps.docx

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ nano 7.txt
```

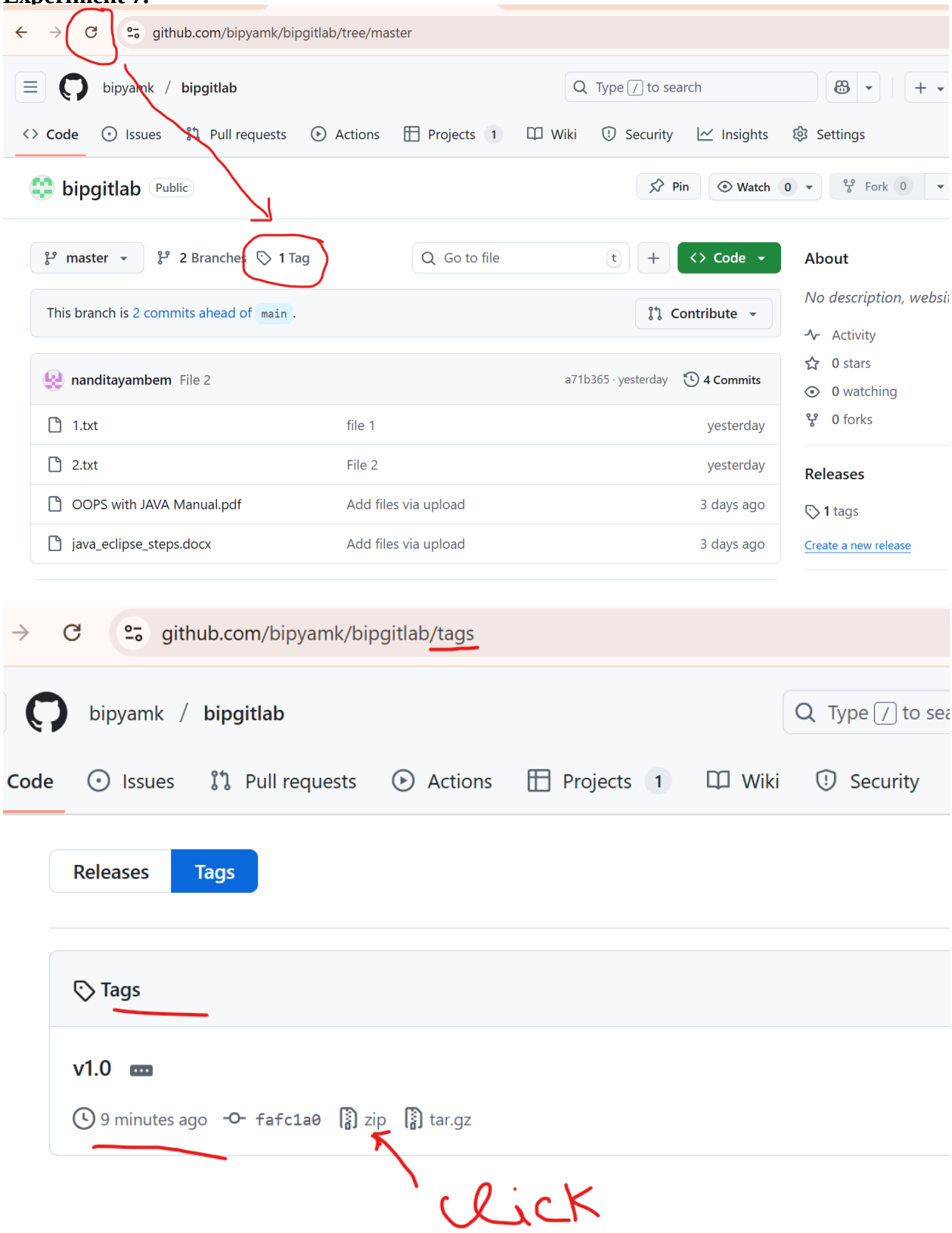
```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git add 7.txt
warning: in the working copy of '7.txt', LF will be replaced by CRLF the next ti
me Git touches it

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git commit -m "file7a"
[main fafc1a0] file7a
 2 files changed, 2 insertions(+)
 create mode 100644 3.txt
 create mode 100644 7.txt
```

Experiment 7.

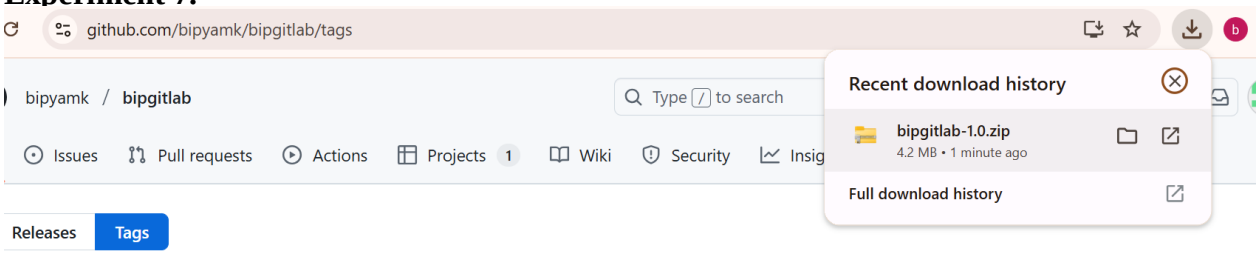
```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git log
commit fafc1a02eac3ff050b8cb6c79ab63cbf5322f221 (HEAD -> main)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Tue Oct 14 12:39:23 2025 +0530

    file7a

commit a71b36508cb38bdaa09154e4357dd5b6d32a1035 (origin/master, master)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:58:42 2025 +0530

    File 2

commit c562b61c321ef443e486f13dca1638066b866c0c
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:54:39 2025 +0530

    file 1

commit ea947f25e29ef54c5a6200ae718c3b1d49ea401e (origin/main, origin/HEAD)
Author: bipyamk <bipyamk@gmail.com>
Date:   Sat Oct 11 15:25:41 2025 +0530

    Add files via upload


ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git tag v1.0

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git tag
v1.0
```

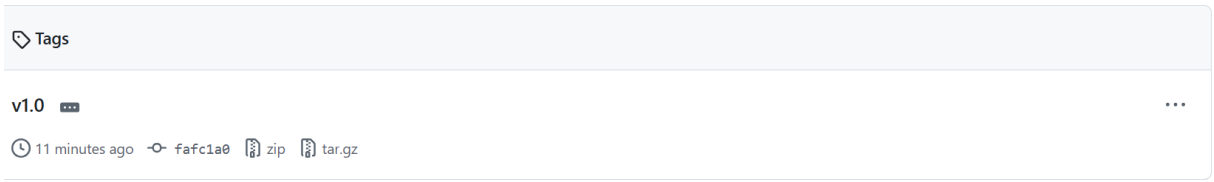
```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git push origin v1.0
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 340 bytes | 85.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/bipyamk/bipgitlab.git
 * [new tag]         v1.0 -> v1.0
```

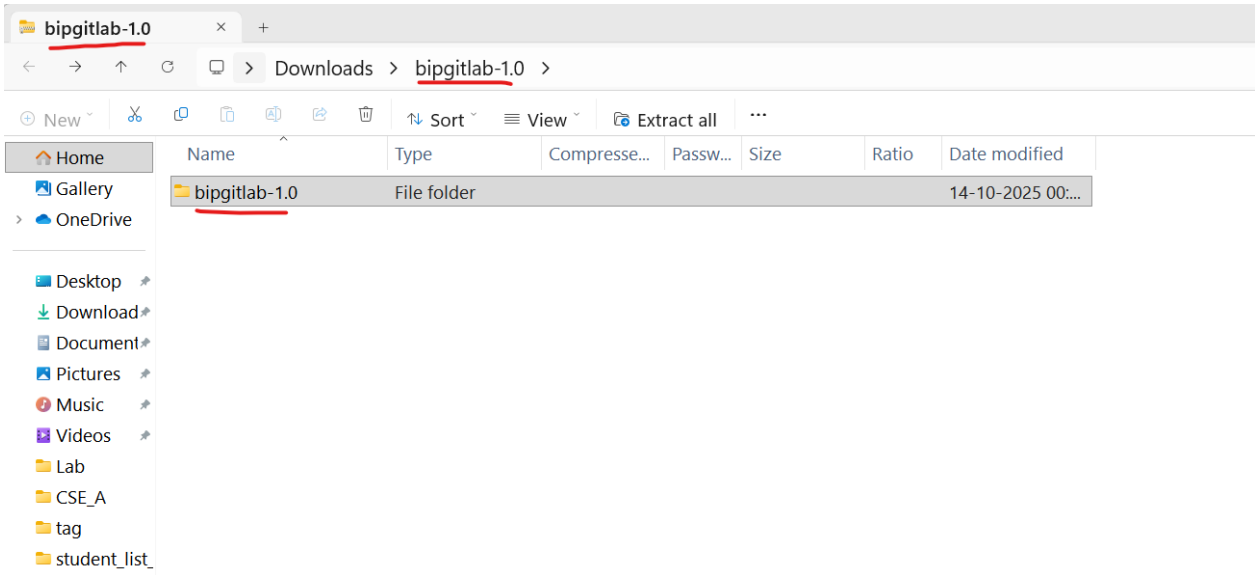
Experiment 7.

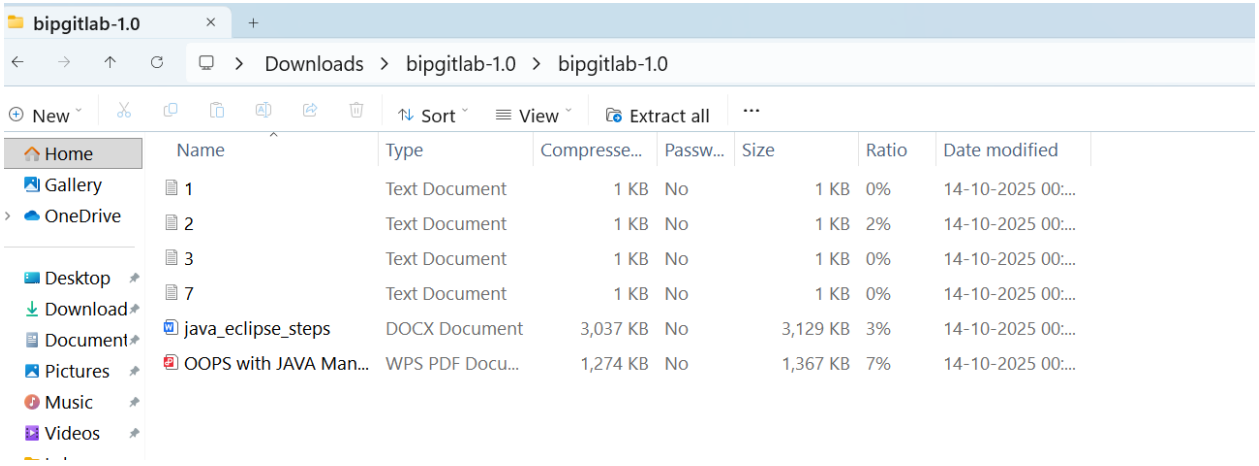


Experiment 7.









Experiment 8.

Advanced Git Operations:

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

Solution:

In Git, "cherry-picking" refers to the act of selecting a specific commit from one branch and applying its changes to another branch.
Instead of merging or rebasing an entire branch, which brings in all commits, cherry-picking allows you to isolate and apply only the changes contained within a particular commit.

Cherry picking is the act of picking a commit from a branch and applying it to another.

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> with the commit at the beginning of the range, and <end-commit> with the commit at the end of the range. The ^ symbol is used to exclude the <start-commit> itself and include all commits after it up to and including <end-commit>. This will apply the changes from the specified range of commits to your current branch.

For example, if you want to cherry-pick a range of commits from "source-branch" starting from commit ABC123 and ending at commit DEF456, you would use:

$ git cherry-pick ABCDEF456

Make sure you are on the branch where you want to apply these changes before running the cherry-pick command.

Experiment 8.

```
TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (main)
$ git branch
  feature-branch
  feature-branchA
* main
  master

TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (main)
$ git log --oneline
b19cf62 (HEAD -> main, tag: v5.0) file7b
c17d30c (tag: v4.0) file7a
2c7b32a (tag: v3.0) file7
915eb59 (master) Revert " added a new line for cherrypicking"
cf70ed3  added a new line for cherrypicking
c0a09bd (tag: v1.0, origin/master) a2.txt
b6dfa1b  Hi file a1
f8c14c5  file 3
2183c1b (tag: v2.0) file 2
c400b03 Hi file1
be81c04  Hi file1
faf9ede 2.txt
f1e6d25  Hi file 1
bb17cef  commiteed message
40e8a1d (origin/main, origin/HEAD) Create hello.txt
df019af Add files via upload
e2c615d Initial commit

TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (main)
$ git checkout master
Switched to branch 'master'
```

Experiment 8.

```
TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (master)
$ git log main --oneline
b19cf62 (tag: v5.0, main) file7b
c17d30c (tag: v4.0) file7a
2c7b32a (tag: v3.0) file7
915eb59 Revert " added a new line for cherrypicking"
cf70ed3  added a new line for cherrypicking
c0a09bd (tag: v1.0, origin/master) a2.txt
b6dfa1b  Hi file a1
f8c14c5  file 3
2183c1b (tag: v2.0) file 2
c400b03 Hi file1
be81c04  Hi file1
faf9ede 2.txt
f1e6d25  Hi file 1
bb17cef  commiteed message
40e8a1d (origin/main, origin/HEAD) Create hello.txt
df019af Add files via upload
e2c615d Initial commit

TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (master)
$ git cherry-pick c17d30c
[master da3654e] file7a
 Date: Fri Sep 12 13:23:05 2025 +0530
 1 file changed, 1 insertion(+)
 create mode 100644 gitlab1/7a.txt

TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (master)
$ git log
commit da3654e362a8f3660dd757ec3b50e0517bccf03f (HEAD -> master)
Author: gitlab1 <nandiyamb@gmail.com>
Date:   Fri Sep 12 13:23:05 2025 +0530

    file7a

commit f3b955a5d16d32e2dd6bec56744f4def805f7aa3
Author: gitlab1 <nandiyamb@gmail.com>
Date:   Fri Sep 12 13:32:25 2025 +0530

    file7b

commit 915eb5965e466e098b30eccf94bdb895d00091b9
Author: gitlab1 <nandiyamb@gmail.com>
```

Experiment 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?

Solution:

To view the details of a specific commit, including the author, date, and commit message, you can use the git show or git log command with the commit ID. Here are both options:

1. Using git show:

bash
git show <commit-ID>

Replace <commit-ID> with the actual commit ID you want to view. This command will display detailed information about the specified commit, including the commit message, author, date, and the changes introduced by that commit.

For example:

$ git show abc123

2. Using git log:

$ git log -n 1 <commit-ID>

The -n 1 option tells Git to show only one commit. Replace <commit-ID> with the actual commit ID. This command will display a condensed view of the specified commit, including its commit message, author, date, and commit ID.

For example:

$ git log -n 1 abc123

Both of these commands will provide you with the necessary information about the specific commit you're interested in.

Experiment 9.

```
TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (main)
$ git log --oneline
b19cf62 (HEAD -> main, tag: v5.0) file7b
c17d30c (tag: v4.0) file7a
2c7b32a (tag: v3.0) file7
915eb59 (master) Revert " added a new line for cherrypicking"
cf70ed3  added a new line for cherrypicking
c0a09bd (tag: v1.0, origin/master) a2.txt
b6dfa1b  Hi file a1
f8c14c5  file 3
2183c1b (tag: v2.0) file 2
c400b03 Hi file1
be81c04  Hi file1
faf9ede 2.txt
f1e6d25  Hi file 1
bb17cef  commiteed message
40e8a1d (origin/main, origin/HEAD) Create hello.txt
df019af Add files via upload
e2c615d Initial commit

TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (main)
$ git show b19cf62
commit b19cf629fe5b0739b336ecccfbb38f0651f7ca71 (HEAD -> main, tag: v5.0)
Author: gitlab1 <nandiyamb@gmail.com>
Date:   Fri Sep 12 13:32:25 2025 +0530

    file7b

diff --git a/gitlab1/7b.txt b/gitlab1/7b.txt
new file mode 100644
index 0000000..eb73a28
--- /dev/null
+++ b/gitlab1/7b.txt
@@ -0,0 +1 @@
+Program 7b

TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (main)
$ git log -n 1 b19cf62
commit b19cf629fe5b0739b336ecccfbb38f0651f7ca71 (HEAD -> main, tag: v5.0)
Author: gitlab1 <nandiyamb@gmail.com>
Date:   Fri Sep 12 13:32:25 2025 +0530

    file7b
```

Experiment 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."

Solution:

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31" in Git, you can use the git log command with the --author and --since and --until options. Here's the command:

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

This command will display a list of commits made by the author "JohnDoe" that fall within the specified date range, from January 1, 2023, to December 31, 2023. Make sure to adjust the author name and date range as needed for your specific use case.

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git log
commit e2ce6ad43132fcb3890002a232d04dfef46a9d81 (HEAD -> main, tag: v2.0)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 27 09:30:57 2025 +0530

    file 7a

commit fafc1a02eac3ff050b8cb6c79ab63cbf5322f221 (tag: v1.0)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Tue Oct 14 12:39:23 2025 +0530

    file7a

commit a71b36508cb38bdaa09154e4357dd5b6d32a1035 (origin/master)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:58:42 2025 +0530

    File 2

commit c562b61c321ef443e486f13dca1638066b866c0c
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:54:39 2025 +0530

    file 1
```

Experiment 10.

Analysing and Changing Git History

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (main)
$ git log --author="pmgitlab" --since="2025-10-13" --until="2025-10-27"
commit e2ce6ad43132fcb3890002a232d04dfef46a9d81 (HEAD -> main, tag: v2.0)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 27 09:30:57 2025 +0530

    file 7a

commit fafc1a02eac3ff050b8cb6c79ab63cbf5322f221 (tag: v1.0)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Tue Oct 14 12:39:23 2025 +0530

    file7a

commit a71b36508cb38bdaa09154e4357dd5b6d32a1035 (origin/master)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:58:42 2025 +0530

    File 2

commit c562b61c321ef443e486f13dca1638066b866c0c
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Mon Oct 13 08:54:39 2025 +0530

    file 1
```

Experiment 11.

Analysing and Changing Git History

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

Solution:

To display the last five commits in a Git repository's history, you can use the git log command with the -n option, which limits the number of displayed commits. Here's the command:

$ git log -n 5

This command will show the last five commits in the repository's history. You can adjust the number after -n to display a different number of commits if needed.

```
TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (master)
$ git log -5
commit da3654e362a8f3660dd757ec3b50e0517bccf03f (HEAD -> master)
Author: gitlab1 <nandiyamb@gmail.com>
Date:   Fri Sep 12 13:23:05 2025 +0530

    file7a

commit f3b955a5d16d32e2dd6bec56744f4def805f7aa3
Author: gitlab1 <nandiyamb@gmail.com>
Date:   Fri Sep 12 13:32:25 2025 +0530

    file7b

commit 915eb5965e466e098b30eccf94bdb895d00091b9
Author: gitlab1 <nandiyamb@gmail.com>
Date:   Fri Sep 12 12:30:11 2025 +0530

    Revert " added a new line for cherrypicking"

    Tddedi
    rr
    s reverts commit cf70ed3546f149d0536d4b2dbba301dde77c780d.

commit cf70ed3546f149d0536d4b2dbba301dde77c780d

TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (master)
$ git log -5 --oneline
da3654e (HEAD -> master) file7a
f3b955a file7b
915eb59 Revert " added a new line for cherrypicking"
cf70ed3  added a new line for cherrypicking
c0a09bd (tag: v1.0, origin/master) a2.txt

TOSHIBA@DESKTOP-GE2MQ7Q MINGW64 /c/Users/Toshiba/gitlab1 (master)
$ git log -5 --pretty=format:"%h - %an, %ar : %s"
da3654e - gitlab1, 21 hours ago : file7a
f3b955a - gitlab1, 21 hours ago : file7b
915eb59 - gitlab1, 22 hours ago : Revert " added a new line for cherrypicking"
cf70ed3 - gitlab1, 22 hours ago :  added a new line for cherrypicking
c0a09bd - gitlab1, 3 days ago : a2.txt
```

Experiment 12.

Analysing and Changing Git History

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

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

$ git revert abc123

Replace "abc123" with the actual commit ID that you want to revert. After running this command, Git will create a new commit that negates the changes introduced by the specified commit. This is a safe way to undo changes in Git because it preserves the commit history and creates a new commit to record the reversal of the changes.

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ nano 12.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git add 12.txt
warning: in the working copy of '12.txt', LF will be replaced by CRLF the next time Git touches it

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git commit -m " 12.txt added"
[master d0863a7]  12.txt added
 1 file changed, 1 insertion(+)
 create mode 100644 12.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git show
commit d0863a7288662c2d40cf0751b4780ef6f7008571 (HEAD -> master)
Author: pmgitlab <nanditayambem@gmail.com>
Date:   Wed Oct 29 14:37:59 2025 +0530

     12.txt added

diff --git a/12.txt b/12.txt
new file mode 100644
index 0000000..fb6d37a
--- /dev/null
+++ b/12.txt
@@ -0,0 +1 @@
+hello 12th programS
```

Experiment 12.

Analysing and Changing Git History

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

```
ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ ls
 1.txt    2.txt    'OOPS with JAVA Manual.pdf'    java_eclipse_steps.docx
 12.txt   7a.txt    bipgitlab/

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git log --oneline
8c7f384 (HEAD -> master)  12.txt added
4b46a55 Revert " 12.txt added"
d0863a7  12.txt added
fe6a04a Revert " a1.txt added"
f14ef55  a1.txt added
a2bd20b file 7a
a71b365 (origin/master) File 2
c562b61 file 1
ea947f2 (origin/main, origin/HEAD) Add files via upload
afe7c98 Add files via upload

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ git revert 8c7f384
```

```
MINGW64:/c/Users/gitise
  GNU nano 8.5          C:/Users/gitise/.git/COMMIT_EDITMSG
Revert " 12.txt added"

This reverts commit 8c7f384c6090f9df43fa9822fd79c131ea9f9443.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       deleted:    12.txt
#
# Untracked files:
#       bipgitlab/
#

                        [ Read 14 lines ]
^G Help      ^O Write Out ^F Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

Experiment 12.

Analysing and Changing Git History

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

```
[master 4317434] Revert " 12.txt added"
 1 file changed, 1 deletion(-)
 delete mode 100644 12.txt

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$ ls
 1.txt   7a.txt                          bipgitlab/
 2.txt  'OOPS with JAVA Manual.pdf'      java_eclipse_steps.docx

ADMIN@DESKTOP-TS5V82H MINGW64 /c/Users/gitise (master)
$
```