SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 01

Date of Performance: **Date of Completion:**

Title: Demonstrate how to create a GitHub account.

---

1. Objective

To understand and demonstrate the process of creating a GitHub account for version control and collaboration.

2. Software/Tools Required

- Web browser (Google Chrome / Edge / Firefox)
- Internet connection
- GitHub website: https://github.com

3. Theory

What is Git?

Git is an open-source version control system that helps developers track changes in their code. It allows multiple people to work on the same project without overwriting each other's work. Git stores data in the form of *snapshots* and maintains a complete history of every change made to a file.

What is GitHub?

GitHub is a cloud-based platform that hosts Git repositories. It provides a user-friendly web interface to manage code repositories, collaborate with teams, and perform version control tasks such as commits, branching, merging, and pull requests.

Importance of GitHub in DevOps

GitHub plays a vital role in DevOps practices:

- It serves as the centralized repository for source code.
- It integrates with CI/CD tools like Jenkins, GitHub Actions, or Docker.
- It facilitates collaboration, code review, and version control.
- It allows teams to automate testing and deployment processes.

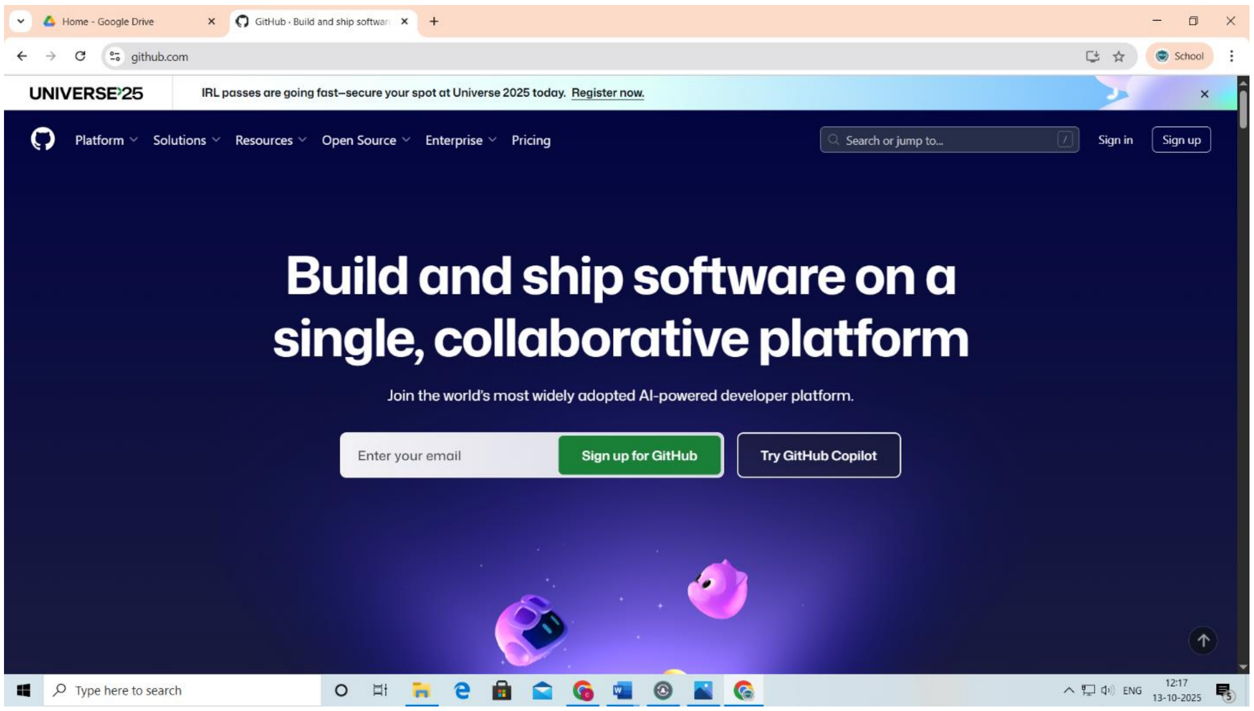
Key Features of GitHub

- **Repositories:** Containers where project files and history are stored.
- **Branches:** Separate versions of a project for developing features independently.
- **Commits:** Save points in the development process.
- **Pull Requests:** Proposals to merge changes from one branch to another.
- **Issues:** Used to track bugs or tasks.

4. Procedure (Step-by-Step Execution)

Step 1: Open GitHub Website

- Open a web browser and visit https://github.com.
- The homepage displays options to sign in or sign up.


Home - Google Drive
GitHub - Build and ship software
github.com
School
UNIVERSE'25
IRL passes are going fast—secure your spot at Universe 2025 today. Register now.
Platform
Solutions
Resources
Open Source
Enterprise
Pricing
Search or jump to...
Sign in
Sign up
Build and ship software on a single, collaborative platform
Join the world's most widely adopted AI-powered developer platform.
Enter your email
Sign up for GitHub
Try GitHub Copilot
Type here to search
ENG
12:17
13-10-2025

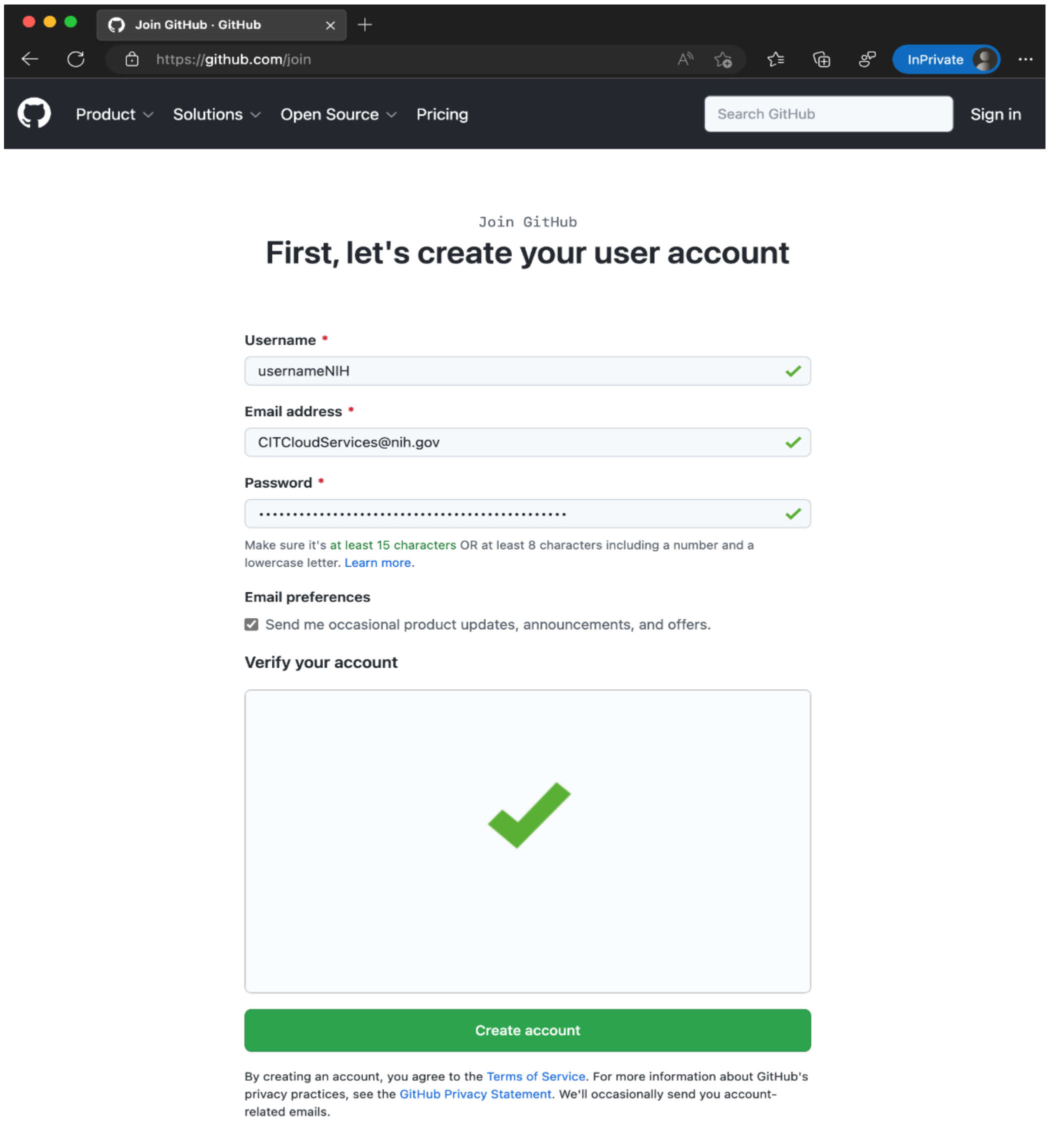
Step 2: Click on "Sign up"

- On the top-right corner, click "Sign up" to create a new account.

Step 3: Enter Your Details

Fill in the following information:

1. **Email Address:** Provide a valid email ID.
2. **Password:** Choose a strong password containing letters, numbers, and symbols.
3. **Username:** Enter a unique username (this will be your public GitHub name).
4. **Email Preferences:** Choose whether you want to receive product updates.


Join GitHub · GitHub

https://github.com/join

Join GitHub

First, let's create your user account

Username *

usernameNIH

Email address *

CITCloudServices@nih.gov

Password *

••

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.

Email preferences

☑ Send me occasional product updates, announcements, and offers.

Verify your account

Create account

By creating an account, you agree to the Terms of Service. For more information about GitHub's privacy practices, see the GitHub Privacy Statement. We'll occasionally send you account-related emails.

Step 4: Verify Your Account

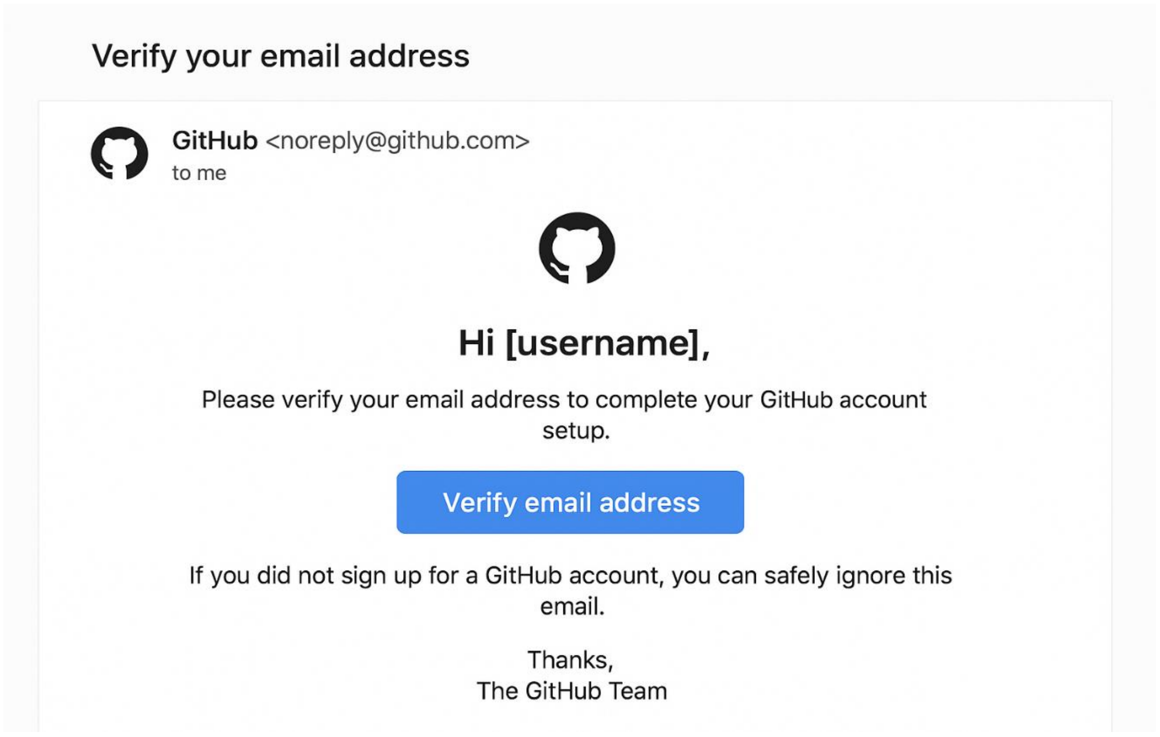
- Complete the CAPTCHA verification (“solve puzzle” to prove you’re human).
- Click “Create account”.

Step 5: Choose Your Plan

- GitHub provides both Free and Pro plans.
- Select Free Plan for students and individuals.
- Click “Continue”.

Step 6: Email Verification

- Open your registered email inbox.
- You’ll receive a confirmation email from GitHub.
- Click “Verify email address” to activate your account.



Verify your email address

GitHub <noreply@github.com>
to me

Hi [username],

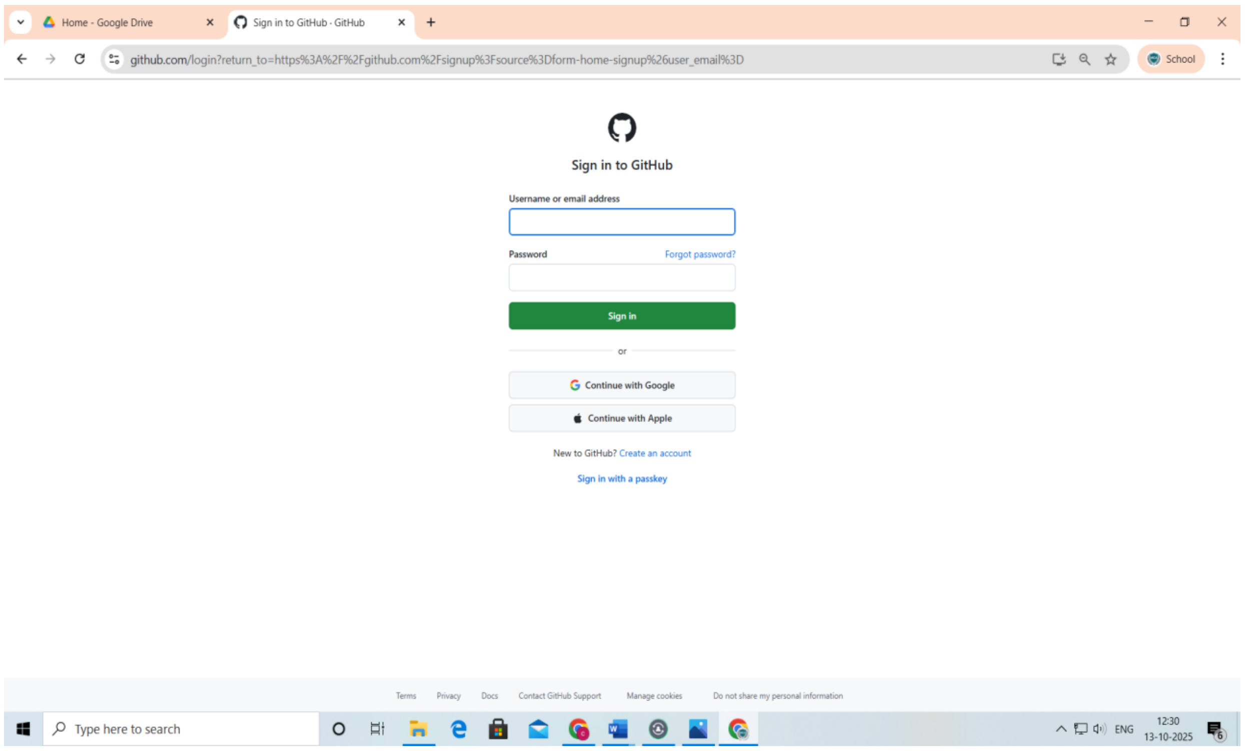
Please verify your email address to complete your GitHub account setup.

If you did not sign up for a GitHub account, you can safely ignore this email.

Thanks,
The GitHub Team

Step 7: Log in to GitHub

- Return to https://github.com/login.
- Enter your username/email and password.



- After login, you will see your GitHub Dashboard, where you can:
    - Create new repositories
    - Explore projects
    - Manage your profile

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Chetana M. Kawale**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 02

Date of Performance: **Date of Completion:**

Title: Exploring Git Commands through Collaborative Coding

- Setting Up Git Repository
- Creating and Committing Changes
- Branching and Merging.

---

1. Objective

To understand and demonstrate the basic Git commands for repository creation, committing changes, and managing multiple branches collaboratively.

2. Software / Tools Required

- **Hardware:** Computer with internet access
- **Software:**
  - Git (installed locally)
  - GitHub account (created in Practical No. 1)
  - Command-line interface (Git Bash / Terminal)

3. Theory

About Git

Git is a distributed version control system used to track changes in source code during software development. It allows multiple developers to collaborate efficiently by maintaining a complete history of code changes.

Key Concepts

1. **Repository (Repo):**
   A storage location for your project files and revision history.
   - *Local repository* → stored on your computer.
   - *Remote repository* → stored on GitHub or any online platform.
2. **Commit:**
   A snapshot of your changes. Each commit represents a save point.
3. **Branch:**
   A separate line of development that allows you to work on new features without affecting the main code.
4. **Merge:**
   Combines changes from one branch into another, typically from a feature branch into the main branch.

4. Procedure / Steps

1. Setting Up Git Repository

What is Git repository?

A **Git repository** is like a folder where Git keeps track of all changes in your project. It helps you go back to earlier versions if needed.

Steps:

Step 1: Install Git

If not already installed, download from:
https://git-scm.com/

Step 2: Create a Project Folder

```
mkdir my-project
```

- mkdir stands for **make directory**.
- This command **creates a new folder** named my-project in your current location.

```
cd my-project
```

- This creates a new folder and opens it.
- cd stands for **change directory**.

- This command **moves into** the newly created folder my-project.

Step 3: Start Git in that Folder

```
git init
```

- This tells Git to start tracking your project.
- Sets up Git in your folder.
- Required before you can commit or track any files.
- Only needs to be run once per project.

Step 4: Check Git Status

```
git status
```

- Shows which files are being tracked or not.
- The git status command shows you the **current state** of your working directory and staging area.
- It tells you:
    - Which files have been **changed**
    - Which files are **staged** (ready to commit)
    - Which files are **untracked** (new files Git doesn't know yet)

2. Creating and Committing Changes

What is a Commit?

- A **commit** is like saving a version of your work in Git.

Steps:

Step 1: Create or Change a File

```
echo "Hello Git" > hello.txt
```

- Example: Create a file named hello.txt
- This command **creates a new file** named hello.txt and writes the text:

Step 2: Check Git Status Again

```
git status
```

- Git will show that hello.txt is **untracked**.

Step 3: Stage the File

```
git add hello.txt
```

- This adds the file to the **staging area**, ready to commit.

Step 4: Commit the File

```
git commit -m "Added hello.txt file"
```

- This saves the version with a message.

- This command **creates a new commit** in your Git repository. It saves the current state of the staged files (in this case, hello.txt) with a message.
- The -m option allows you to write a **commit message** directly in the command.

3. Branching and Merging

What is a Branch?

- A **branch** is a separate version of your project — useful for testing new features without breaking the main project.

Steps:

Step 1: Create a New Branch

```
git branch new-feature
```

- This creates a branch named new-feature.
- This command creates a **new branch** in your Git repository named new-feature.
- Think of a branch as a **parallel version** of your project. You can experiment and make changes without affecting the main version (main or master).

Step 2: Switch to the New Branch

```
git checkout new-feature
```

- You're now working in the new branch.
- This switches your working directory from the current branch (e.g., master) to the branch named new-feature.
- You're now working **inside** the new-feature branch — any changes you make and commit will stay in this branch until you merge them.

Step 3: Make and Commit Changes

```
echo "New Feature Code" > feature.txt
git add feature.txt
git commit -m "Added feature.txt"
```

- Your changes are now saved in the new-feature branch.

Step 4: Switch Back to Main Branch

```
git checkout master
```

- You're back to the main project.
- This command **switches you back** from your current branch (like new-feature) to the **main branch**, typically named master (or sometimes main in newer setups).

Step 5: Merge the Feature into Main

```
git merge new-feature
```

- Combines the changes from new-feature into master.
- This command **merges changes** from the new-feature branch **into your current branch**, which should now be master.
- It combines the commits from new-feature into master.

Step 6: Delete the Branch (Optional)

```
git branch -d new-feature
```

- Clean up if the branch is no longer needed.
- This command **deletes the new-feature branch** from your local repository
-

Submitted By	**Checked By :**
Sign :	
Name :	**Asst. Prof. Chetana M. Kawale**
Roll No :	

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 03

Date of Performance: **Date of Completion:**

Title: Implement GitHub Operations using Git

• Cloning a Repository

• Making Changes and Creating a Branch

• Push/Pull Changes to GitHub

---

1. Objective

To demonstrate the process of performing GitHub operations using Git commands — including cloning a remote repository, creating a branch locally, making code changes, and synchronizing updates with the GitHub remote repository using push and pull operations.

2. Software / Tools Required

- **Hardware:** Computer with internet connectivity
- **Software:**
  - Git (installed on local system)
  - GitHub account (already created)
  - Web browser (Chrome / Edge / Firefox)
  - Git Bash or Command Prompt

3. Theory

What is GitHub Operation?

GitHub operations refer to the set of tasks performed to synchronize your local repository (on your computer) with the remote repository (on GitHub). These include:

- **Cloning** → Downloading a remote repository locally
- **Pushing** → Uploading your local commits to the GitHub repository
- **Pulling** → Downloading updates made by others from GitHub
- **Branching** → Creating isolated workspaces for new features or fixes

4. Procedure / Steps

1. Cloning a Repository

Concept:
"Cloning" means downloading a **full copy** of a Git repository from a remote server (like GitHub) to your local computer.

Key Terms:

- **Repository (Repo)**: A folder where your project's files and the full history of changes are stored.
- **Remote Repository**: The version stored on GitHub (online).
- **Local Repository**: Your copy stored on your PC.
- **HTTPS URL**: A link that uses HTTPS protocol to communicate with GitHub (can require username/password or token).
- **SSH URL**: A link that uses the SSH protocol for secure, password-less communication (after key setup).

```
git clone https://github.com/USER/REPO.git
```

- git → Calls Git itself.
- clone → Git command to create a local copy from a remote.
- https://github.com/USER/REPO.git → URL of the remote repository.
- Git makes a .git hidden folder in your project — this stores all commit history.
- Git sets a “remote” named **origin** pointing to the original GitHub repo URL.
- You end up on the default branch (often main).

2. Making Changes & Creating a Branch

- Git encourages making changes in **branches** rather than directly in main.
- Branch is a copy of your project where you can make changes **without affecting the main project** until you decide to merge.

Check current branch and status

```
git status
```

```
git branch
```

- Lists branches in your local repo.
- The * marks the current branch.

Create and switch to a new branch

```
git checkout -b feature/my-change
```

- checkout → Used to switch branches or restore files.
- -b → Creates a **new** branch and switches to it immediately.
- feature/my-change → Branch name (follow good naming: feature/, bugfix/, etc.)

Alternative modern command:

```
git switch -c feature/my-change
```

- switch → Newer, clearer way to change branches in Git.
- -c → Create and switch.

Make and stage changes

Example:

```
echo "Hello" > hello.txt
```

- echo writes text into a file.
- creates or overwrites a file.

```
git add hello.txt
```

- Moves changes into the **staging area** (preparing for commit).
- You can add multiple files or all files:

```
git add .
```

- (. means "all changes in the current folder and subfolders.")

Commit changes

```
git commit -m "Add hello.txt to demonstrate changes"
```

- commit → Saves staged changes into your local repo.
- -m → Message flag.
- "..." → Short, clear description of the change.

3. Push / Pull Changes

Concept:
Git keeps your local repo and remote repo in sync with push (send changes up) and pull (bring changes down).

Push for the first time

```
git push -u origin feature/my-change
```

- push → Sends commits from your local branch to the remote.
- -u → Sets **upstream tracking** (so later git push works without specifying remote/branch).
- origin → Name of the default remote.
- feature/my-change → Name of the branch.

Next pushes are just:

```
git push
```

Pull changes from remote

```
git pull
```

- pull = fetch + merge.
- Updates your local branch with remote changes.

If push is rejected

Reason: someone else pushed changes to GitHub before you.
Solution:

```
git pull --rebase origin main
```

- --rebase → Reapplies your commits on top of the updated branch to keep history clean.

Or:

```
git pull origin main
```

- (Merges the changes instead.)

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Chetana M. Kawale**

Roll No :

SSBT’s College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 04

Date of Performance: **Date of Completion:**

Title: Create account on docker step by steps.

---

1. Objective

To understand the concept of containerization and demonstrate how to create an account on **Docker Hub** to store and share Docker images.

2. Software / Tools Required

- **Hardware Requirements:**
    - Computer/Laptop with internet access
- **Software Requirements:**
    - Web browser (Google Chrome / Microsoft Edge / Mozilla Firefox)
    - Internet connectivity
    - Docker Hub website: https://hub.docker.com

3. Theory

What is Docker?

Docker is an open-source platform that automates the deployment of applications inside lightweight, portable containers.
Containers package applications along with their dependencies, making them easily portable across environments (development, testing, and production).

Key Components of Docker

1. **Docker Engine:** Core software that runs containers.
2. **Docker Images:** Read-only templates used to create containers.
3. **Docker Containers:** Running instances of Docker images.
4. **Docker Hub:** A cloud-based registry where Docker users can store and share container images.

Why Create a Docker Hub Account?

- To **store** and **share** custom-built Docker images.
- To **access** public images for applications and environments.
- To **integrate** with CI/CD tools for automated deployments.
- To **collaborate** with team members on containerized projects.

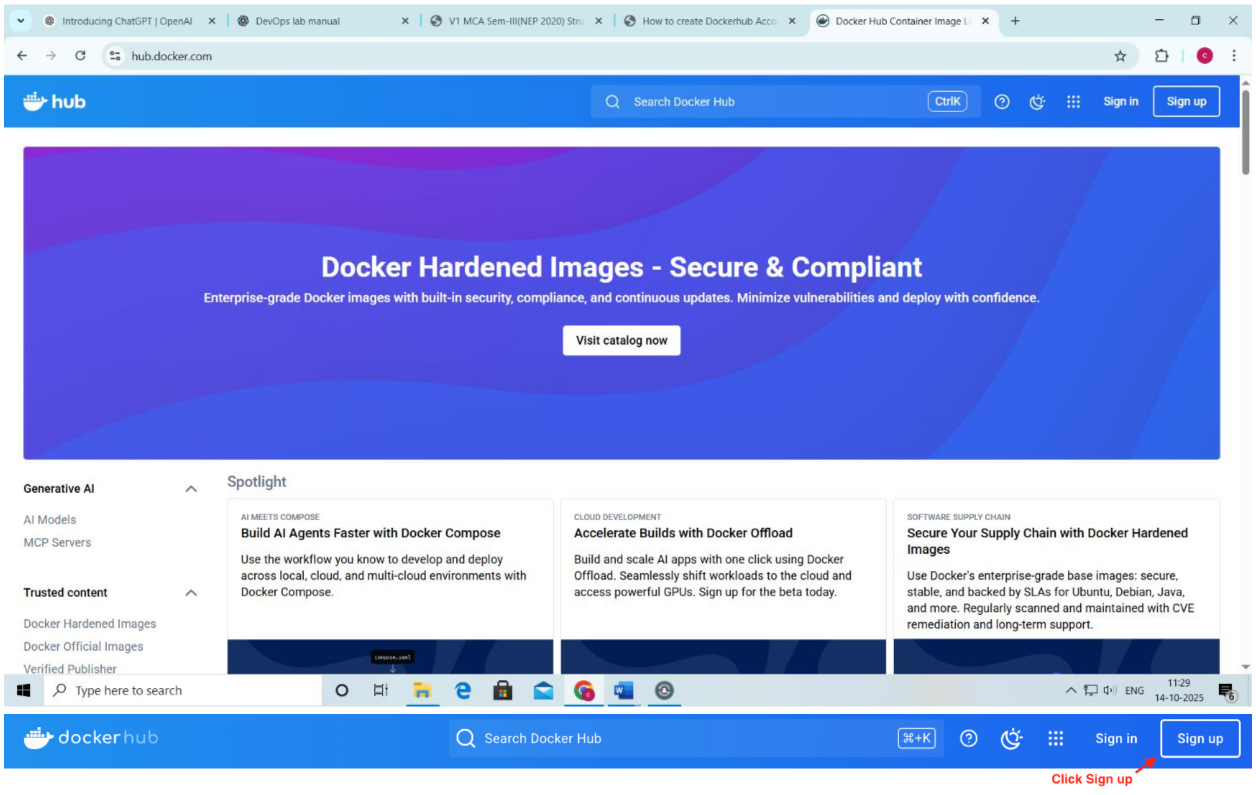
4. Procedure / Step-by-Step Execution

Step 1: Open the Docker Hub Website

- Open your preferred web browser.
- Visit https://hub.docker.com.
- The homepage will display a “Sign In” and “Sign Up” option.

Step 2: Click on “Sign Up”

- Click the **“Sign Up”** button on the top-right corner of the homepage.
- You’ll be redirected to the registration page.



Docker Hardened Images - Secure & Compliant

Enterprise-grade Docker images with built-in security, compliance, and continuous updates. Minimize vulnerabilities and deploy with confidence.

AI MEETS COMPOSE
Build AI Agents Faster with Docker Compose
Use the workflow you know to develop and deploy across local, cloud, and multi-cloud environments with Docker Compose.

CLOUD DEVELOPMENT
Accelerate Builds with Docker Offload
Build and scale AI apps with one click using Docker Offload. Seamlessly shift workloads to the cloud and access powerful GPUs. Sign up for the beta today.

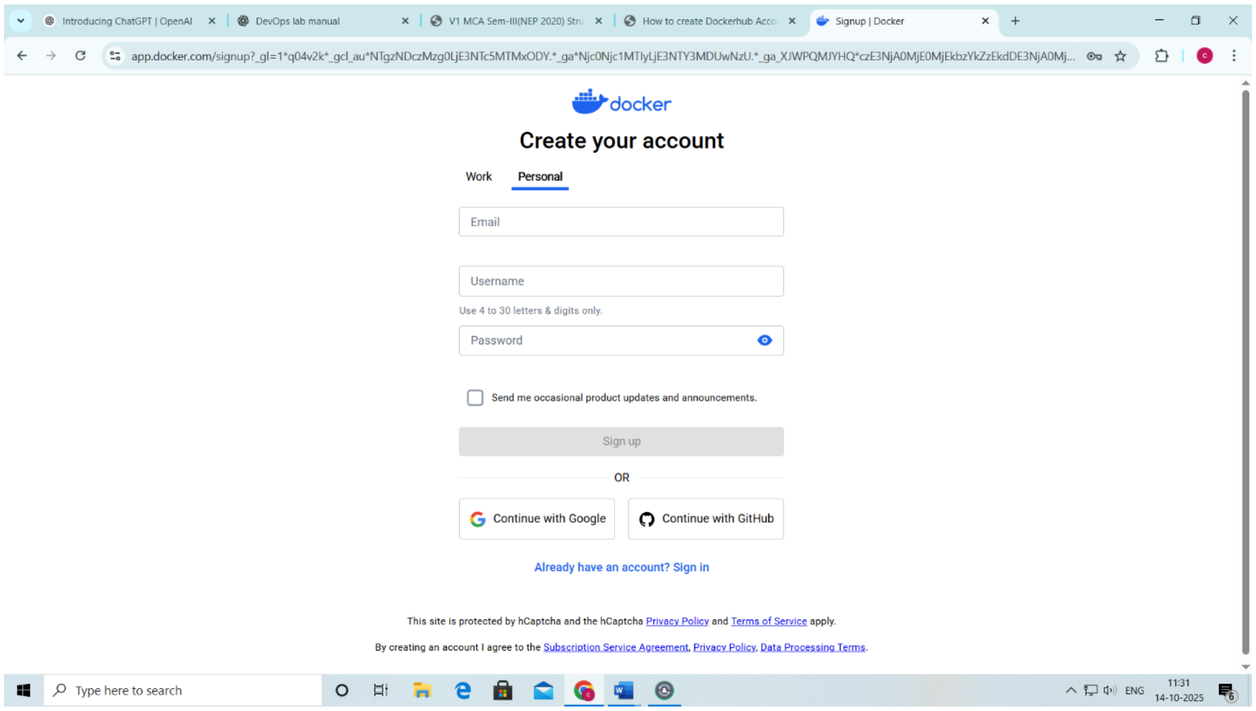
SOFTWARE SUPPLY CHAIN
Secure Your Supply Chain with Docker Hardened Images
Use Docker's enterprise-grade base images: secure, stable, and backed by SLAs for Ubuntu, Debian, Java, and more. Regularly scanned and maintained with CVE remediation and long-term support.

Step 3: Fill Out the Registration Form

Enter the required details:

- **Username:** Choose a unique username for your Docker ID.
- **Email Address:** Provide a valid email address.
- **Password:** Enter a strong password (at least 8 characters, including a symbol and number).

Then, check the box to accept the **Docker Terms of Service**.



Create your account

Use 4 to 30 letters & digits only.

Send me occasional product updates and announcements.

Already have an account? Sign in

This site is protected by hCaptcha and the hCaptcha Privacy Policy and Terms of Service apply.

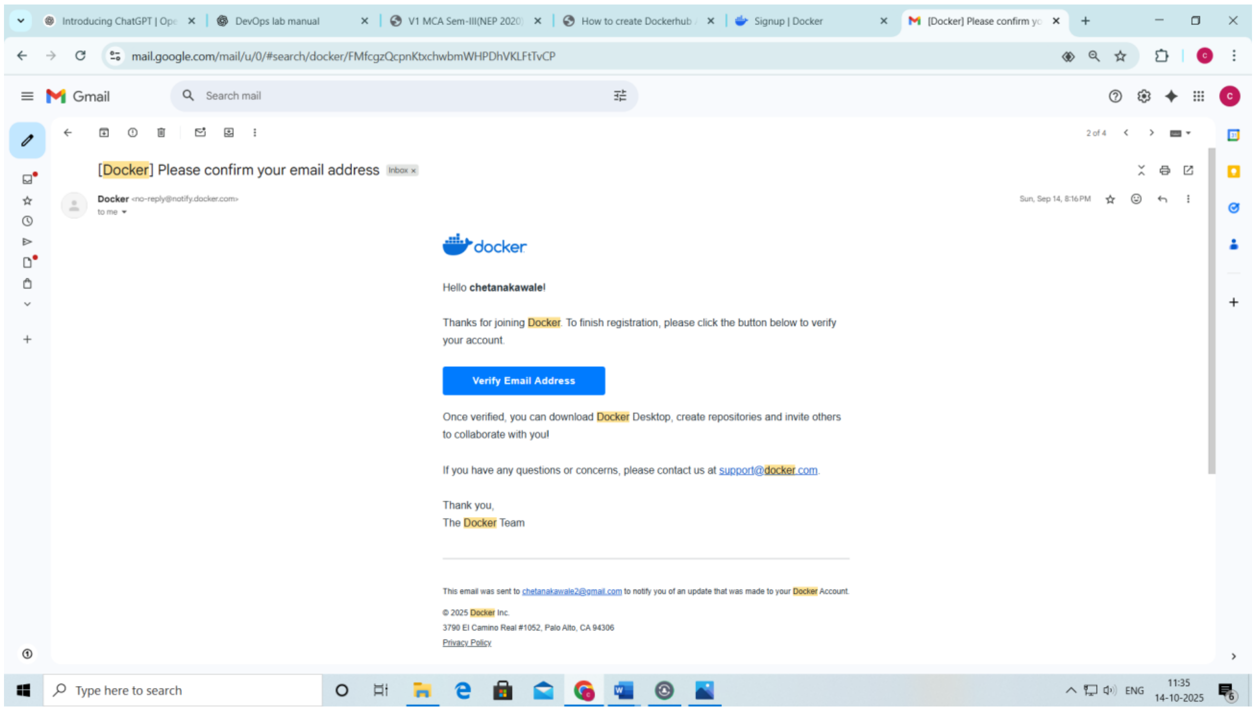
By creating an account I agree to the Subscription Service Agreement, Privacy Policy, Data Processing Terms.

Step 4: Click "Sign Up"

- After entering your details, click on the **"Sign Up"** button.
- Docker will display a confirmation message asking you to verify your email.

Step 5: Verify Your Email Address

- Open your registered email inbox.
- Find the email from **Docker Hub** with the subject "Verify your email."
- Click the **"Verify Email"** button in the email.



mail.google.com/mail/u/0/#search/docker/FMfcgzQcpnKtxchwbmWHPDhVKLFtTvCP

[Docker] Please confirm your email address

Docker <no-reply@notify.docker.com>
to me

Sun, Sep 14, 8:16 PM

Hello **chetanakawale**!

Thanks for joining Docker. To finish registration, please click the button below to verify your account.

Verify Email Address

Once verified, you can download Docker Desktop, create repositories and invite others to collaborate with you!

If you have any questions or concerns, please contact us at support@docker.com.

Thank you,
The Docker Team

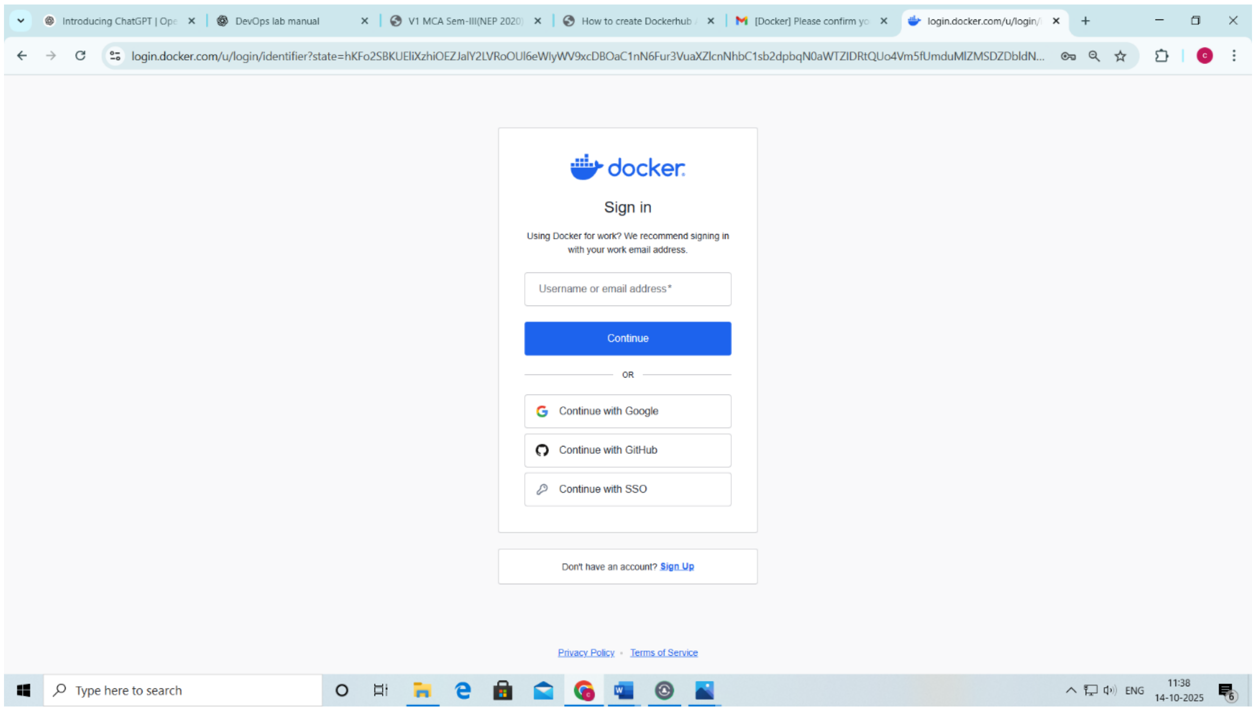
This email was sent to chetanakawale2@gmail.com to notify you of an update that was made to your Docker Account.

© 2025 Docker Inc.
3790 El Camino Real #1052, Palo Alto, CA 94306
Privacy Policy

✅ *Your Docker Hub account is now activated.*

Step 6: Log In to Docker Hub

- Return to https://hub.docker.com.
- Click on **"Sign In."**
- Enter your registered email/username and password.
- Click **"Sign In."**


Sign in

Using Docker for work? We recommend signing in with your work email address.

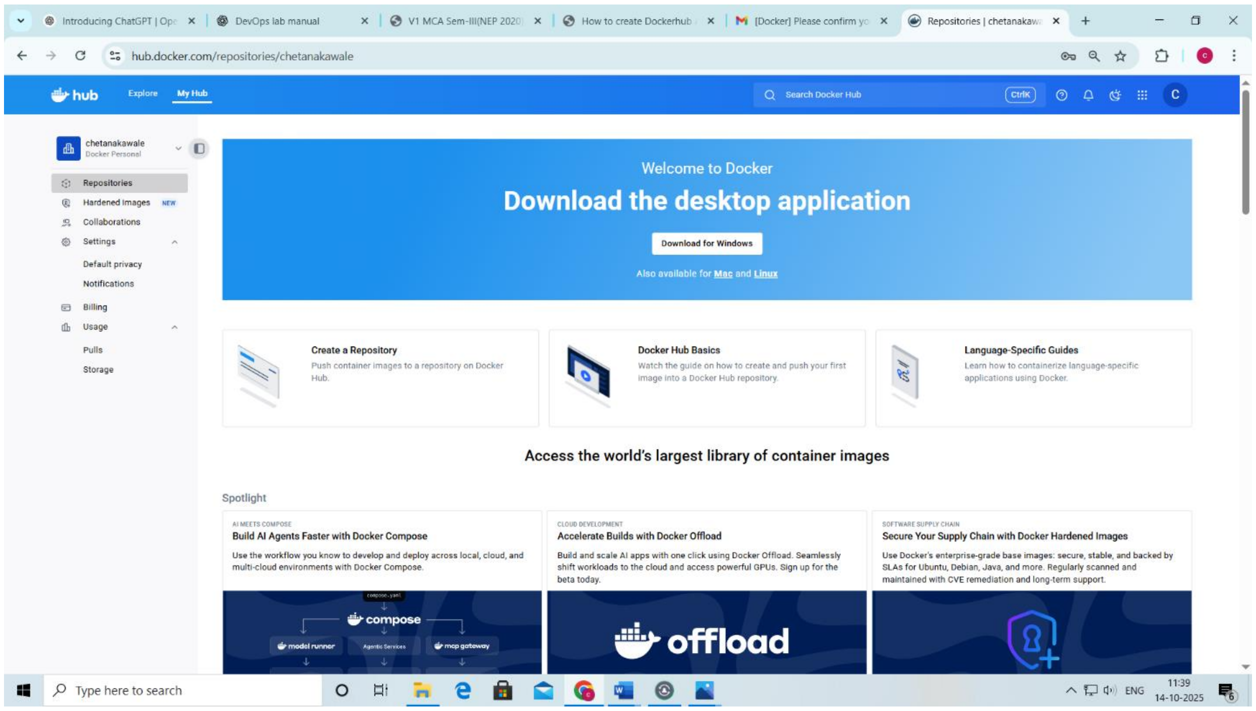
Don't have an account? Sign Up

✅ *You will now see your Docker Hub dashboard.*

Step 7: Explore Your Docker Hub Dashboard

Once logged in, you can:

- View your **profile** and **repositories**
- Create new **repositories** to host images
- Browse **official Docker images**
- Manage **organization and access control**



Welcome to Docker

Download the desktop application

Download for Windows

Also available for Mac and Linux

Create a Repository
Push container images to a repository on Docker Hub.

Docker Hub Basics
Watch the guide on how to create and push your first image into a Docker Hub repository.

Language-Specific Guides
Learn how to containerize language-specific applications using Docker.

Access the world's largest library of container images

Spotlight

AI MEETS COMPOSE
Build AI Agents Faster with Docker Compose
Use the workflow you know to develop and deploy across local, cloud, and multi-cloud environments with Docker Compose.

CLOUD DEVELOPMENT
Accelerate Builds with Docker Offload
Build and scale AI apps with one click using Docker Offload. Seamlessly shift workloads to the cloud and access powerful GPUs. Sign up for the beta today.

SOFTWARE SUPPLY CHAIN
Secure Your Supply Chain with Docker Hardened Images
Use Docker's enterprise-grade base images: secure, stable, and backed by SLAs for Ubuntu, Debian, Java, and more. Regularly scanned and maintained with CVE remediation and long-term support.

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Chetana M. Kawale**

Roll No :

SSBT’s College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 05

Date of Performance: **Date of Completion:**

Title: Demonstrate a practical on Version Control Tools.

---

1. Objective

To understand and demonstrate the working of version control tools used in software development for tracking and managing changes to source code.

2. Software / Tools Required

- **Hardware:**
  - Computer system with minimum 4 GB RAM
  - Internet connection
- **Software:**
  - Git (installed locally)
  - GitHub account (created in previous practicals)
  - Optional: Git GUI tools such as **GitHub Desktop** or **SourceTree**

3. Theory

What is Version Control?

Version Control is a system that records changes to files over time so that specific versions can be recalled later. It helps developers collaborate efficiently without overwriting each other’s work. In software development, version control tools are essential for maintaining the integrity of codebases and enabling teamwork, rollback, and change history.

Types of Version Control Systems

Type	**Description**	**Example Tools**
Local Version Control	Tracks changes on a single computer.	RCS (Revision Control System)
Centralized Version Control (CVCS)	A single server stores all files and versions; developers check out files.	CVS, Subversion (SVN), Perforce
Distributed Version Control (DVCS)	Each developer has a full local copy of the repository, enabling offline work.	**Git**, Mercurial, Bazaar

Common Version Control Tools

1. **Git** – Most popular distributed VCS, used with GitHub, GitLab, Bitbucket.
2. **SVN (Subversion)** – Centralized system for large enterprise projects.
3. **Mercurial** – Lightweight distributed VCS similar to Git.
4. **Perforce** – Enterprise-grade VCS used for large-scale projects.

Advantages of Version Control

- Tracks history of changes
- Enables teamwork and collaboration
- Allows rollback to previous versions
- Supports multiple branches and merging
- Integrates with DevOps CI/CD pipelines

4. Procedure / Steps

Step 1: Install Git

- Download Git from ☞ https://git-scm.com/downloads
- Install it (default options are fine).
- Verify installation:
- git --version

You should see a version number.

Step 2: Configure Git (One-Time Setup)

Set your username and email (used in commits):

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

Check configuration:

```
git config --list
```

Step 3: Create a Project Folder

```
mkdir vcs-demo
cd vcs-demo
```

Now create a simple file:

```
echo "Hello, Version Control!" > file1.txt
```

Step 4: Initialize Git Repository

Turn this folder into a Git repository:

```
git init
```

This creates a hidden .git folder.

Step 5: Add File to Staging Area

Check status:

```
git status
```

Add file:

```
git add file1.txt
```

Step 6: Commit the File

```
git commit -m "First commit - added file1.txt"
```

This saves a version (snapshot).

Step 7: Modify File & Commit Again

Edit file:

```
echo "Adding second line to file1" >> file1.txt
```

Check changes:

```
git status
git diff
```

Commit changes:

```
git add file1.txt

git commit -m "Updated file1 with second line"
```

Step 8: View Commit History

```
git log
```

or in one line:

```
git log --oneline
```

Step 9: Create & Switch Branch

```
git branch new-feature
git checkout new-feature
```

Now you are in new-feature branch.

Make a change:

```
echo "Feature branch update" >> feature.txt
git add feature.txt
git commit -m "Added feature.txt in new-feature branch"
```

Step 10: Merge Branch into Main

Switch back to main:

```
git checkout master
```

Merge:

```
git merge new-feature
```

Now feature.txt will appear in master branch.

Step 11: Push to Remote Repository (GitHub)

- Create a repository on GitHub.
- Link local repo:
- git remote add origin https://github.com/yourusername/vcs-demo.git
- Push changes:
- git push -u origin master

Practical Demonstration on Version Control Tool (SVN)

Step 1: Install SVN

- Download from https://subversion.apache.org/packages.html
- Verify installation:
- `svn --version`

Step 2: Create a Central Repository

1. Choose a folder for repository:
2. `svnadmin create /path/to/svn-repo`

   This initializes a central repository.

Step 3: Checkout Working Copy

To start working on files, checkout:

```
svn checkout file:///path/to/svn-repo myproject
cd myproject
```

This creates a working copy of the repository.

Step 4: Add a File

Create a file:

```
echo "Hello, SVN!" > file1.txt
```

Add and commit:

```
svn add file1.txt
svn commit -m "First commit - added file1.txt"
```

Step 5: Update & Modify

Edit the file:

```
echo "Second line added" >> file1.txt
```

Check status and commit:

```
svn status
svn commit -m "Updated file1 with second line"
```

◆ **Step 6: View History**

```
svn log
```

Step 7: Branching & Merging in SVN

In SVN, branches are just copies inside the repository.

1. Create a branch:
   ```
   svn copy file:///path/to/svn-repo/trunk \
   file:///path/to/svn-repo/branches/new-feature \
   -m "Created new-feature branch"
   ```

2. Switch to branch:
   ```
   svn switch file:///path/to/svn-repo/branches/new-feature
   ```

3. Make changes:
   ```
   echo "Feature branch update" > feature.txt
   svn add feature.txt
   svn commit -m "Added feature.txt in new-feature branch"
   ```

4. Merge back to trunk:
   ```
   svn switch file:///path/to/svn-repo/trunk
   svn merge file:///path/to/svn-repo/branches/new-feature
   svn commit -m "Merged new-feature into trunk"
   ```

Practical Demonstration on Version Control Tool (Mercurial)

Step 1: Install Mercurial

- Download from ☞ https://www.mercurial-scm.org/downloads
- Verify installation:
- hg --version

Step 2: Create a Repository

Make a project folder:

```
mkdir hg-demo
cd hg-demo
hg init
```

This initializes a Mercurial repository (creates .hg folder).

Step 3: Add a File

Create and add a file:

```
echo "Hello, Mercurial!" > file1.txt
hg add file1.txt
hg commit -m "First commit - added file1.txt"
```

Step 4: Modify and Commit Again

```
echo "Second line added" >> file1.txt
hg commit -m "Updated file1 with second line"
```

Step 5: View History

```
hg log
```

Step 6: Branching & Merging

1. Create and switch branch:
   ```
   hg branch new-feature
   hg commit -m "Created new-feature branch"
   ```

2. Make changes:
```
echo "Feature branch update" > feature.txt
hg add feature.txt
hg commit -m "Added feature.txt in new-feature branch"
```

3. Switch back to default branch:
```
hg update default
```

4. Merge changes:
```
hg merge new-feature
hg commit -m "Merged new-feature into default branch"
```

Step 7: Push to Remote Repository

Mercurial can also use hosting services like **Bitbucket** (earlier supported Mercurial).
To push:

```
hg push https://example.com/your-repo
```

Now we’ve seen **3 different VCS tools in practice**:

- **Git** (most popular, distributed)
- **SVN** (centralized)
- **Mercurial** (distributed, simpler than Git)

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Chetana M. Kawale**

Roll No :

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 06

Date of Performance: **Date of Completion:**

Title: Create a merge request on gitlab and Review the merge request.

---

1. Objective

To demonstrate how to collaborate in a GitLab project by creating a **Merge Request (MR)** and performing a **code review** before merging changes into the main branch.

2. Software / Tools Required

- **Hardware:**
  - Computer system with internet access
- **Software / Accounts:**
  - Git installed on system
  - GitLab account (https://gitlab.com)
  - A project repository on GitLab

3. Theory

What is a Merge Request (MR)?

A **Merge Request (MR)** in GitLab is a request to merge one branch (usually a feature or bug-fix branch) into another branch (typically the main or develop branch).

It allows **team collaboration** by enabling code review, discussion, and approval before integration.

Key Features of Merge Requests

- Code review and feedback mechanism
- Conflict resolution before merging
- Approval and comments from team members
- CI/CD pipeline integration for automated testing

Merge Request Workflow

1. Developer clones the repository.
2. Creates a **new branch** for the feature.
3. Makes changes and commits them.
4. Pushes the branch to GitLab.
5. Creates a **Merge Request (MR)**.
6. Reviewer checks and approves the MR.
7. The MR is merged into the main branch.

4. Procedure / Steps

1. Steps to Create a GitLab Account

1. **Go to GitLab's website**
   Open your browser and visit: https://gitlab.com
2. **Click on "Register"**
   - You'll see **Sign in / Register** in the top-right corner.
   - Click **Register**.
3. **Fill in your details**
   You can register in two ways:
   - **Using Email** (common option):
     - Enter your **Full name**

        - Choose a **Username** (this becomes part of your profile URL)
        - Enter your **Email address**
        - Set a **Password**
        - Agree to GitLab's terms
    - **Using OAuth providers**: You can also sign up directly with **Google, GitHub, Bitbucket, or GitLab.com SSO**.

4. **Verify your email**
    - GitLab will send a confirmation link to the email you used.
    - Open your inbox, click the verification link.
5. **Complete setup**
    - After verification, log in with your email/username & password.
    - You may be asked a few setup questions (like choosing a plan: Free, Premium, Ultimate).
    - If you're just starting, select the **Free Plan**.
6. **Start using GitLab**
    - Once logged in, you can create a new project or import one.
    - You'll also get access to GitLab's **remote repositories, issues, pipelines, etc.**

2. Steps to Create a Repository (Project) in GitLab

Step 1: Log in to GitLab

1. Open https://gitlab.com.
2. Click **Sign in**.
3. Enter your **username/email** and **password**.

Step 2: Go to New Project

1. On the top-right, click the **"+" button**.

2. Click **New project**.

Step 3: Choose Project Type

You will see 4 options:

- **Create blank project** → Start fresh (use this).
- **Create from template** → Predefined project setup.
- **Import project** → Bring code from GitHub or Bitbucket.
- **Fork project** → Copy someone else's project.

For beginners, choose **Create blank project**.

Step 4: Fill Project Details

1. **Project name** → e.g., my-first-project
2. **Project slug** → auto-generated from project name (used in URL).
3. **Description** → Optional, e.g., "My first GitLab project."
4. **Visibility** → Choose one:
    - **Private** → Only you can see it.
    - **Internal** → Only GitLab users can see it.
    - **Public** → Anyone can see it.
5. **Initialize repository with a README** → Tick this box (so repo isn't empty).

Step 5: Create Project

- Click **Create project**.
- You will now see your project dashboard with your repository ready.

3. Steps to Create token

Step 1: Log in to GitLab

1. Open https://gitlab.com
2. Sign in with your username/email and password.

Step 2: Go to Personal Access Tokens

1. Click on your **profile picture** in the top-right corner.
2. Select **Settings**.
3. On the left sidebar, click **Access Tokens**.

Step 3: Fill Token Details

1. **Name** → Give a name for your token, e.g., my-token.
2. **Expires at** → Optional, you can set a date when the token will expire.
3. **Scopes** → Choose permissions for the token:
    - read_repository → Read-only access
    - write_repository → Push & pull code (most common)
    - api → Full API access

For beginners who want to push code, select **write_repository** and **read_repository**.

Step 4: Create Token

1. Click **Create personal access token**.
2. GitLab will generate a token **once**.
3. **Copy the token immediately** and save it somewhere secure (like a password manager).
    - You **cannot see this token again** after leaving the page.

Step 5: Use Token

- When Git asks for your password during Git push/pull, use the **token instead of your GitLab password**.

Part 1: Create a Merge Request on GitLab

Step 1: Clone your repository (if not already)

```
git clone <repository-url>
cd <repository-name>
```

Step 2: Create a new branch

```
git checkout -b feature/my-feature
```

Replace feature/my-feature with a descriptive branch name.

Step 3: Make changes and commit

```
# Make your code changes
git add .
git commit -m "Add feature XYZ"
```

Step 4: Push the branch to GitLab

```
git push origin feature/my-feature
```

Step 5: Create a Merge Request

1. Go to your GitLab project in the browser.
2. Click **Merge Requests → New merge request**.
3. Select:
   - **Source branch:** feature/my-feature
   - **Target branch:** main (or master/develop depending on your workflow)
4. Add:
   - Title: Clear and descriptive, e.g., Add feature XYZ.
   - Description: Explain what the MR does and why.
5. Click **Create merge request**.

Part 2: Review a Merge Request

Step 1: Open the MR

- Go to **Merge Requests** → Click on the MR you want to review.

Step 2: Review the changes

- Navigate to the **Changes** tab.
- Check:
  - Code correctness.
  - Consistency with coding standards.
  - No unnecessary files or commits.
  - Proper test coverage (if applicable).

Step 3: Comment on code

- Click **Add a comment** next to a line in **Changes**.
- Provide constructive feedback, e.g., suggest improvements or ask for clarification.

Step 4: Approve or Request Changes

- If the MR looks good: click **Approve**.
- If changes are needed: leave comments and click **Request changes**.

Step 5: Merge

- Once approved and tests pass:
  - Click **Merge**.
  - Delete the source branch if no longer needed

Submitted By	**Checked By :**
Sign :	
Name :	**Asst. Prof. Chetana M. Kawale**
Roll No :	

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Department of Computer Applications

Practical: 07

Date of Performance: **Date of Completion:**

Title: To study docker file instructions, build an image for a sample web application using docker file

---

1. Objective

To study **Dockerfile instructions** and demonstrate how to **build and run a Docker image** for a sample web application.

2. Software / Tools Required

- **Hardware:**
  - Computer system with at least 8 GB RAM and internet access
- **Software:**
  - Docker Desktop (Windows / Mac) or Docker Engine (Linux)
  - Text editor (VS Code / Notepad++)
  - Sample web application files (HTML or Python/Node.js app)

3. Theory

What is Docker?

Docker is a platform that automates the deployment of applications inside lightweight, portable **containers**. Containers include everything needed to run the application — code, runtime, libraries, and dependencies — ensuring consistency across environments.

What is a Dockerfile?

A **Dockerfile** is a plain text file containing a set of instructions used by Docker to build an image. Each instruction in a Dockerfile creates a **layer** in the image.

Common Dockerfile Instructions

Instruction	**Description**	**Example**
FROM	Specifies the base image	FROM ubuntu:20.04
MAINTAINER / LABEL	Specifies author info	LABEL maintainer="user@example.com"
RUN	Executes commands during image build	RUN apt-get install -y python3
COPY	Copies files from host to image	COPY . /app
ADD	Copies files or URLs to the image	ADD index.html /var/www/html/
WORKDIR	Sets the working directory	WORKDIR /app
CMD	Defines default command to run a container	CMD ["python3", "app.py"]
EXPOSE	Opens a port for communication	EXPOSE 80
ENTRYPOINT	Sets the main executable for the container	ENTRYPOINT ["python3", "app.py"]

4. Procedure / Steps

Step 1: Create a Project Folder

```
mkdir my-flask-app
cd my-flask-app
```

This folder will contain your application and Dockerfile.

Step 2: Create a Sample Web Application

1. Create app.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Docker!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- host='0.0.0.0' allows the app to be accessed outside the container.
- port=5000 is the port Flask will run on.

2. Create requirements.txt

```
flask
```

- This file lists Python dependencies.

Step 3: Create a Dockerfile

Create a file named Dockerfile in the same folder:

```
# 1. Use an official Python image as base
FROM python:3.11

# 2. Set working directory inside container
WORKDIR /app

# 3. Copy dependency file to container
COPY requirements.txt .

# 4. Install dependencies
```

```
RUN pip install --no-cache-dir -r requirements.txt

# 5. Copy the rest of the application code
COPY app.py .

# 6. Expose the port the app runs on
EXPOSE 5000

# 7. Command to run the application
CMD ["python", "app.py"]
```

Explanation of instructions:

Instruction	Purpose
FROM	Base image for your container
WORKDIR	Directory inside container where commands run
COPY	Copy files from host to container
RUN	Run commands during build (install dependencies)
EXPOSE	Tell Docker which port the container listens on
CMD	Default command to run when container starts

Step 4: Build the Docker Image

```
docker build -t my-flask-app:1.0 .
```

- -t my-flask-app:1.0 → name and version tag for the image.
- . → current directory is context (Dockerfile and app files).

Check the image:

```
docker images
```

Step 5: Run the Docker Container

```
docker run -d -p 5000:5000 my-flask-app:1.0
```

- -d → detached mode (runs in background)
- -p 5000:5000 → maps host port 5000 to container port 5000

Test in browser:

```
http://localhost:5000
```

You should see **“Hello, Docker!”**.

Step 6: Manage Docker Containers

- List running containers:

```
docker ps
```

- Stop a container:

```
docker stop <container_id>
```

- Remove a container:

```
docker rm <container_id>
```

Step 7: Optional - Test Changes

- Make changes to app.py.
- Rebuild the image:

```
docker build -t my-flask-app:1.1 .
```

- Run the new image.

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Chetana M. Kawale**

Roll No :