SURYADATTA COLLEGE OF MANAGEMENT

INFORMATION RESEARCH & TECHNOLOGY

BAVDHAN ,PUNE-411021

LAB COURSE ON CS-612-MJ DevOps Fundamentals

Submitted by

- **NAME OF STUDENT**

1.D h r u v i B h a v s a r

Under the Guidance of

Miss.Mrunal gaikward mam

SUBMITTED IN PARTIAL

FULLFILLMENT OF MASTER OF SCIENCE(COMPUTER SCIENCE)

SEM-III

SAVITRIBAI PHULE PUNE UNIVERSITY

For Academic Year 2023-2024

CERTIFICATE

This is to certify that Mr./Ms.

student of MSC(CS) Semester____having Seat No. __at Surya Datta College of Management Information Research & Technology (SCMIRT), Pune, has successfully completed the assigned practical in

prescribed by the Savitribai Phule Pune University during the academic year_.

Internal Examiner External Examiner

Principal

Place: Pune Date:

1 Exploring Git Commands through Collaborative Coding.

Task 0 : Install Git

Output / Command ::

Verify Git installation with the version command.

```
C:\Users\Omkar>git version
git version 2.44.0.windows.1
```

Task1 : Setting Up Git Repository

● Open the command-line interface on your computer.

● Navigate to the directory where you want to create your Git repository.

● Run the basic git commands:

git init - This initialises a new Git repository in the current directory.

Commands/output:

```
PS C:\Users\Omkar\Desktop\new folder (2)> git init
Initialized empty Git repository in C:/Users/Omkar/Desktop/new folder (2)/.git/
```

Task2 : Creating and Committing Changes

● Create a new text file named "example.txt" using any text editor.

● Add some content to the "example.txt" file.

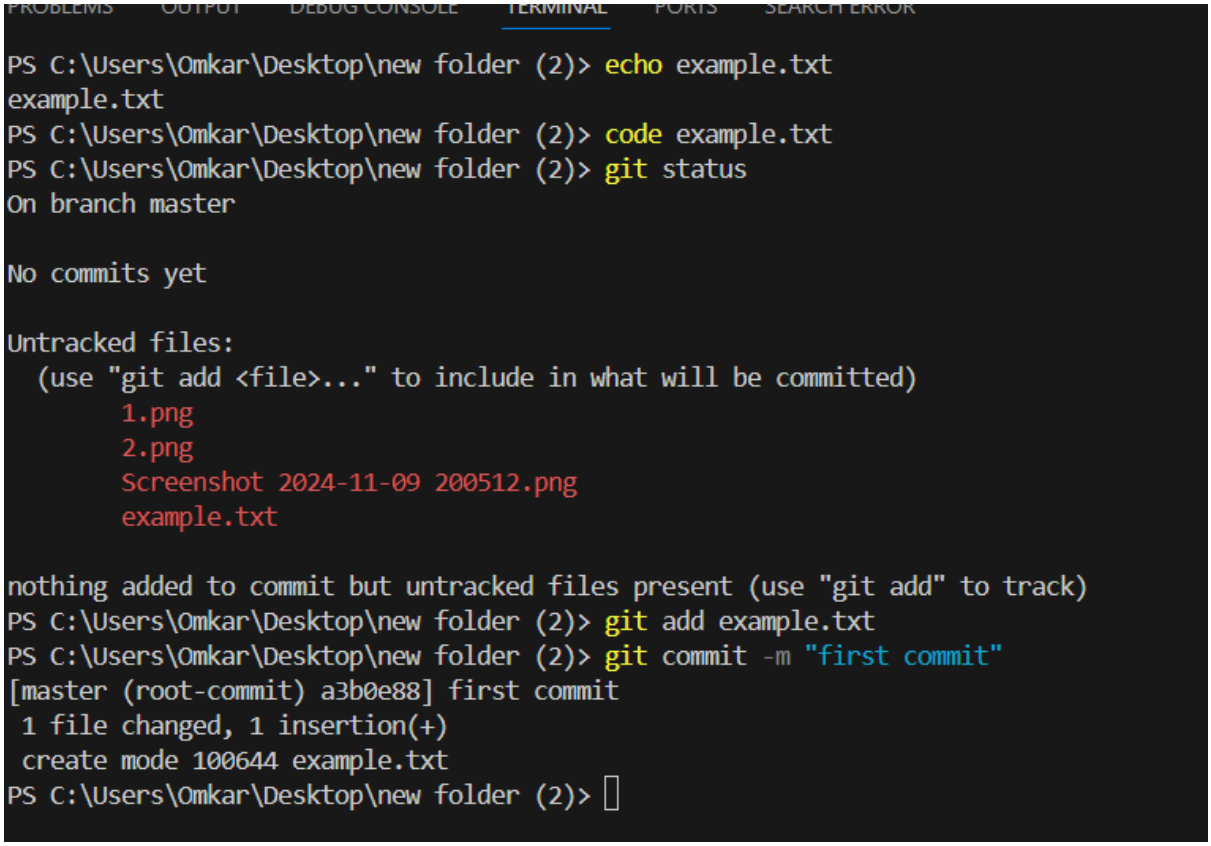
● In the command-line interface, run the following commands:

git status - This command shows the status of your working directory, highlighting untracked files.

git add example.txt - This stages the changes of the "example.txt" file for commit.

git commit -m "Add content to example.txt" - This commits the staged changes with a descriptive message.

Command /Output::



Task3 : Exploring History

Modify the content of "example.txt."

Run the following commands: git status - Notice the modified file is shown as "modified."

git diff - This displays the differences between the working directory and the last commit.

git log - This displays a chronological history of commits.

Command/output:

```
PS C:\Users\Omkar\Desktop\new folder (2)> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   example.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        1.png
        2.png
        Screenshot 2024-11-09 200512.png
        Screenshot 2024-11-09 201852.png

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\Omkar\Desktop\new folder (2)> git diff
diff --git a/example.txt b/example.txt
index 9854092..ef716e8 100644
--- a/example.txt
+++ b/example.txt
@@ -1 +1 @@
-Example text
\ No newline at end of file
+Example text: Apple
\ No newline at end of file
PS C:\Users\Omkar\Desktop\new folder (2)> git log
commit a3b0e88b5ca0a8ad593cf6e95bf2fd8f06e2de9b (HEAD -> master)
Author: omkar <omkarjoshi1231@gmail.com>
Date:   Sat Nov 9 20:18:25 2024 +0530

    first commit
PS C:\Users\Omkar\Desktop\new folder (2)>
```

Task4 : Branching and Merging

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

git branch feature, git checkout feature

or shorthand: git checkout -b feature

- Make changes to the "example.txt" file in the "feature" branch.
- Commit the changes in the "feature" branch.
- Switch back to the "master" branch: git checkout master
- Merge the changes from the "feature" branch into the "master" branch:

git merge feature

command/output

```
Already on 'feature'
PS C:\Users\Omkar\Desktop\new folder (2)> git add example.txt
PS C:\Users\Omkar\Desktop\new folder (2)> git commit -m "Update example.txt in feature branch"
[feature 95801c7] Update example.txt in feature branch
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\Omkar\Desktop\new folder (2)> git checkout master
Switched to branch 'master'
PS C:\Users\Omkar\Desktop\new folder (2)> git merge feature
Updating a3b0e88..95801c7
Fast-forward
 example.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\Omkar\Desktop\new folder (2)> git log --oneline
95801c7 (HEAD -> master, feature) Update example.txt in feature branch
a3b0e88 first commit
PS C:\Users\Omkar\Desktop\new folder (2)>
```

Task5 : Collaborating with Remote Repositories

● Create an account on a Git hosting service like GitHub (https://github.com/).

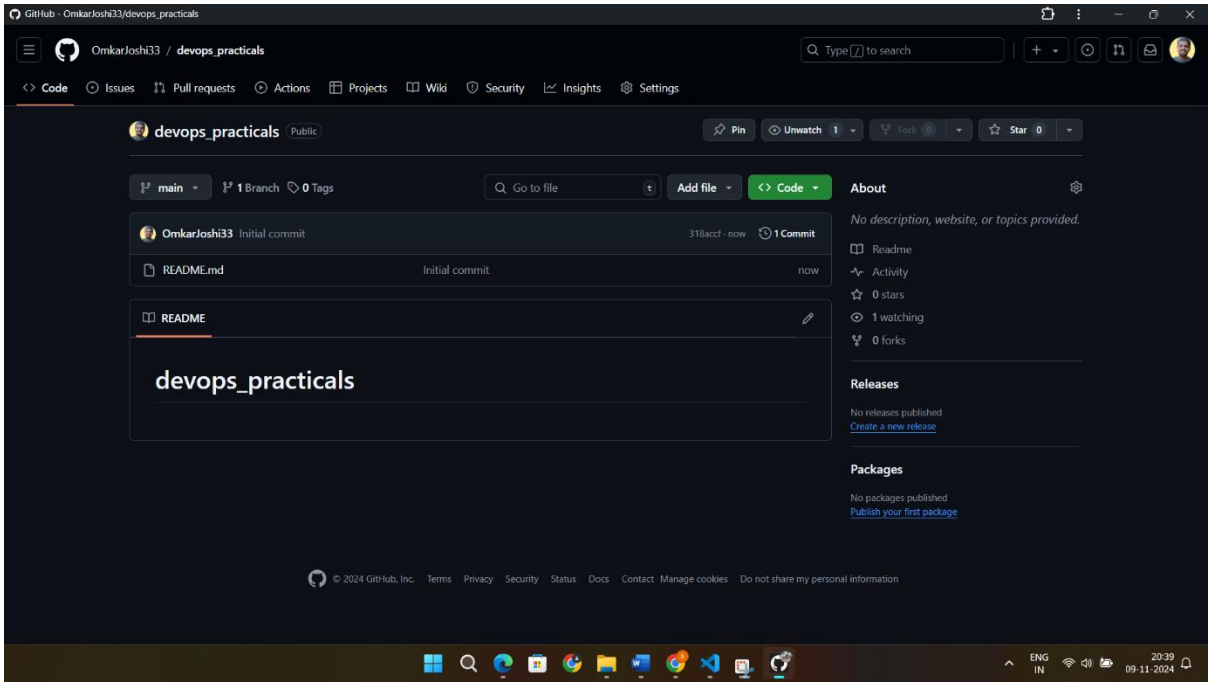
● Create a new repository on GitHub.

● Link your local repository to the remote repository:

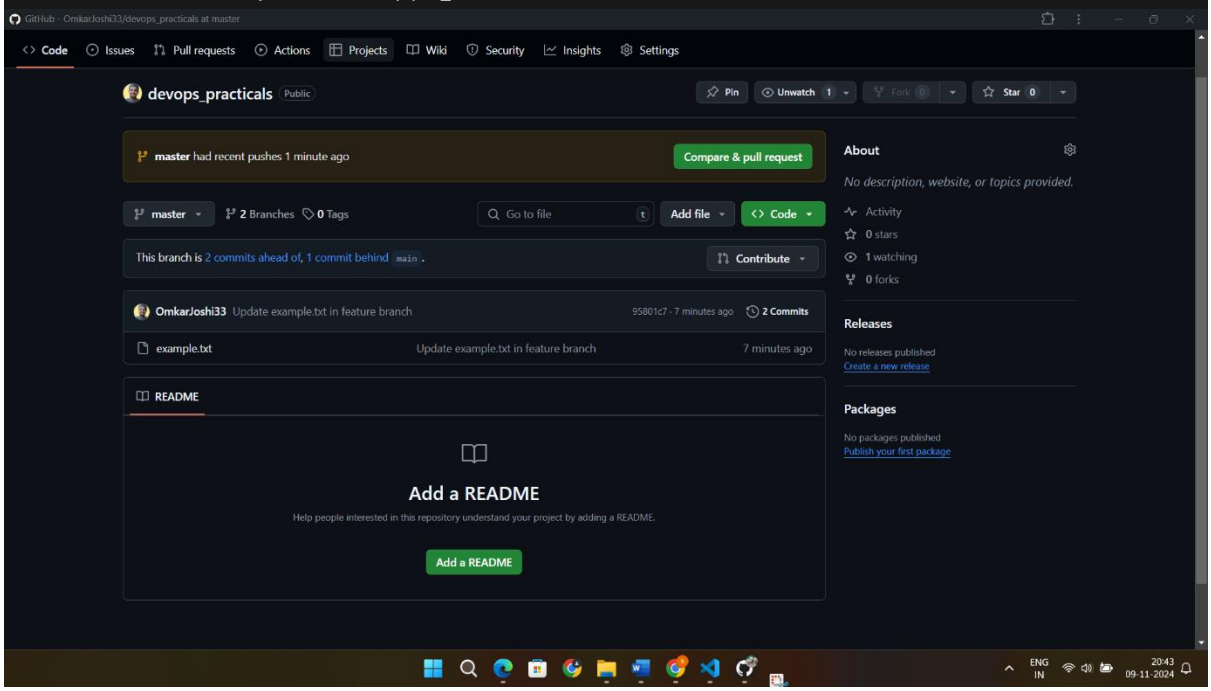
git remote add origin <repository_url>

● Push your local commits to the remote repository: git push origin master

Commands/ output::



```
PS C:\Users\Omkar\Desktop\new folder (2)> git remote add origin https://github.com/OmkarJoshi33/devops_practicals.git
error: remote origin already exists.
PS C:\Users\Omkar\Desktop\new folder (2)> git remote add master https://github.com/OmkarJoshi33/devops_practicals.git
PS C:\Users\Omkar\Desktop\new folder (2)> git push -u master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 499 bytes | 499.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/OmkarJoshi33/devops_practicals/pull/new/master
remote:
To https://github.com/OmkarJoshi33/devops_practicals.git
 * [new branch]      master -> master
branch 'master' set up to track 'master/master'.
PS C:\Users\Omkar\Desktop\new folder (2)>
```



2 Implement GitHub Operations using Git.

Task 1: Cloning a Repository

● Sign in to your GitHub account.

● Find a repository to clone (you can use a repository of your own or any public repository).

● Click the "Code" button and copy the repository URL.

● Open your terminal or command prompt.

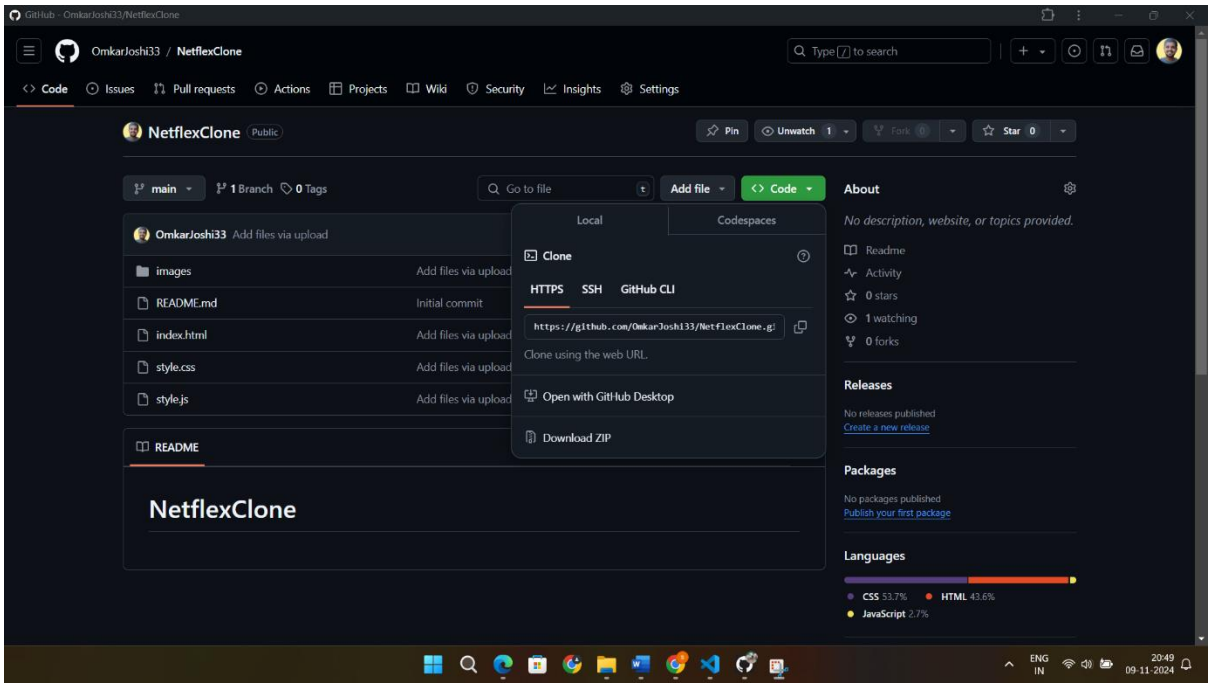
● Navigate to the directory where you want to clone the repository.

● Run the following command: git clone <repository_url>

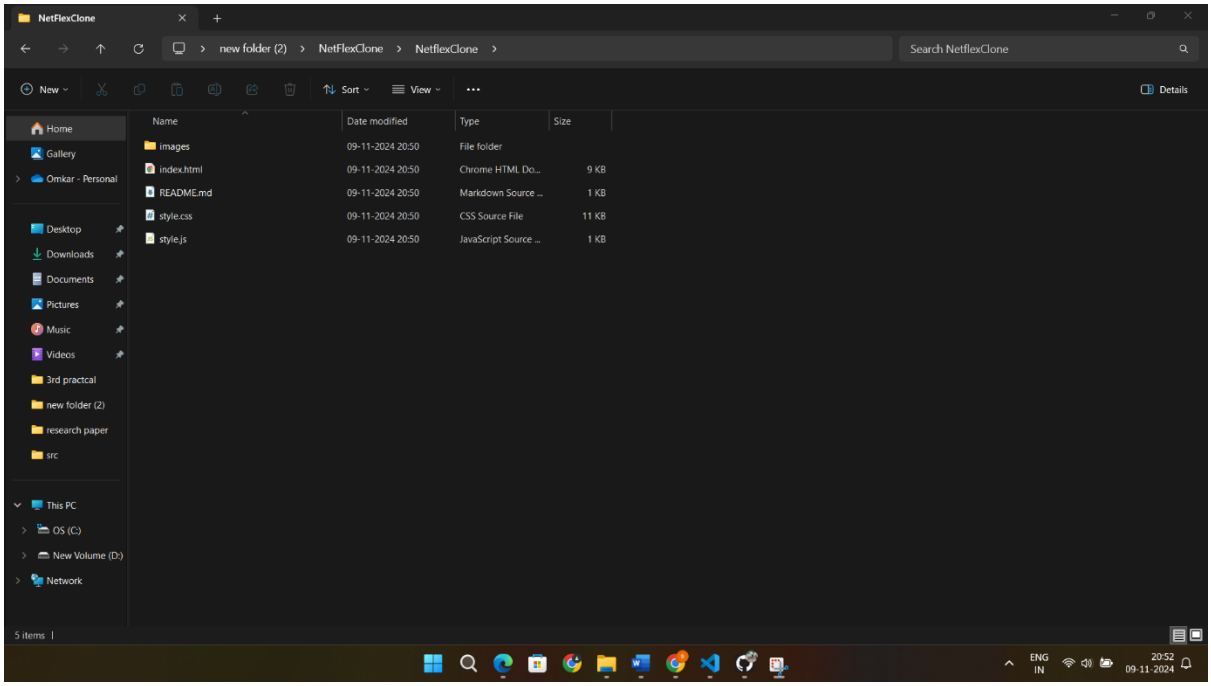
● Replace <repository_url> with the URL you copied from GitHub.

● This will clone the repository to your local machine.

Command/output




```
PS C:\Users\Omkar\Desktop\new folder (2)> cd NetFlexClone
PS C:\Users\Omkar\Desktop\new folder (2)\NetFlexClone> git clone https://github.com/OmkarJoshi33/NetflexClone.git
Cloning into 'NetflexClone'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 27 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (27/27), 936.84 KiB | 595.00 KiB/s, done.
PS C:\Users\Omkar\Desktop\new folder (2)> cd NetFlexClone
PS C:\Users\Omkar\Desktop\new folder (2)\NetFlexClone> git clone https://github.com/OmkarJoshi33/NetflexClone.git
Cloning into 'NetflexClone'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 27 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (27/27), 936.84 KiB | 595.00 KiB/s, done.
PS C:\Users\Omkar\Desktop\new folder (2)\NetFlexClone> git clone https://github.com/OmkarJoshi33/NetflexClone.git
Cloning into 'NetflexClone'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 27 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (27/27), 936.84 KiB | 595.00 KiB/s, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 27 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (27/27), 936.84 KiB | 595.00 KiB/s, done.
Resolving deltas: 100% (3/3), done.
PS C:\Users\Omkar\Desktop\new folder (2)\NetFlexClone> cd repository-name
```



Task 2: Making Changes and Creating a Branch

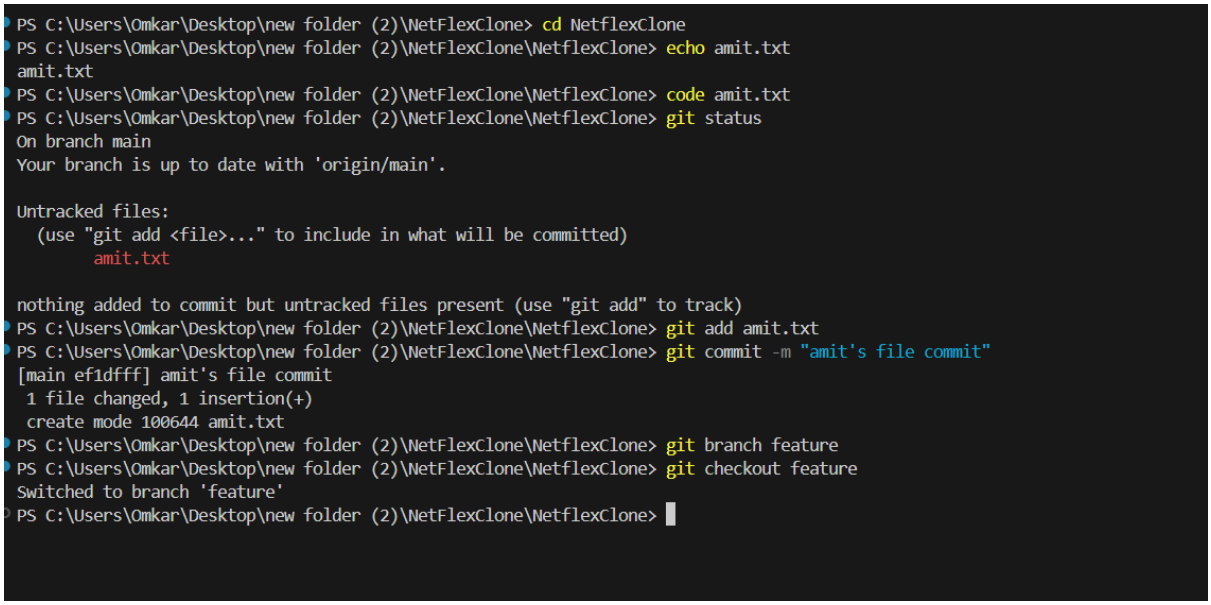
Navigate into the cloned repository: cd <repository_name>

- Create a new text file named "amit.txt" using a text editor.
- Add some content to the "amit.txt" file.
- Save the file and return to the command line.
- Check the status of the repository: git status
- Stage the changes for commit: git add amit.txt
- Commit the changes with a descriptive message:

git commit -m "Add content to amit.txt"

- Create a new branch named "feature": git branch feature
- Switch to the "feature" branch: git checkout feature

Command/output:



Task 3: Pushing Changes to GitHub

● Add Repository URL in a variable

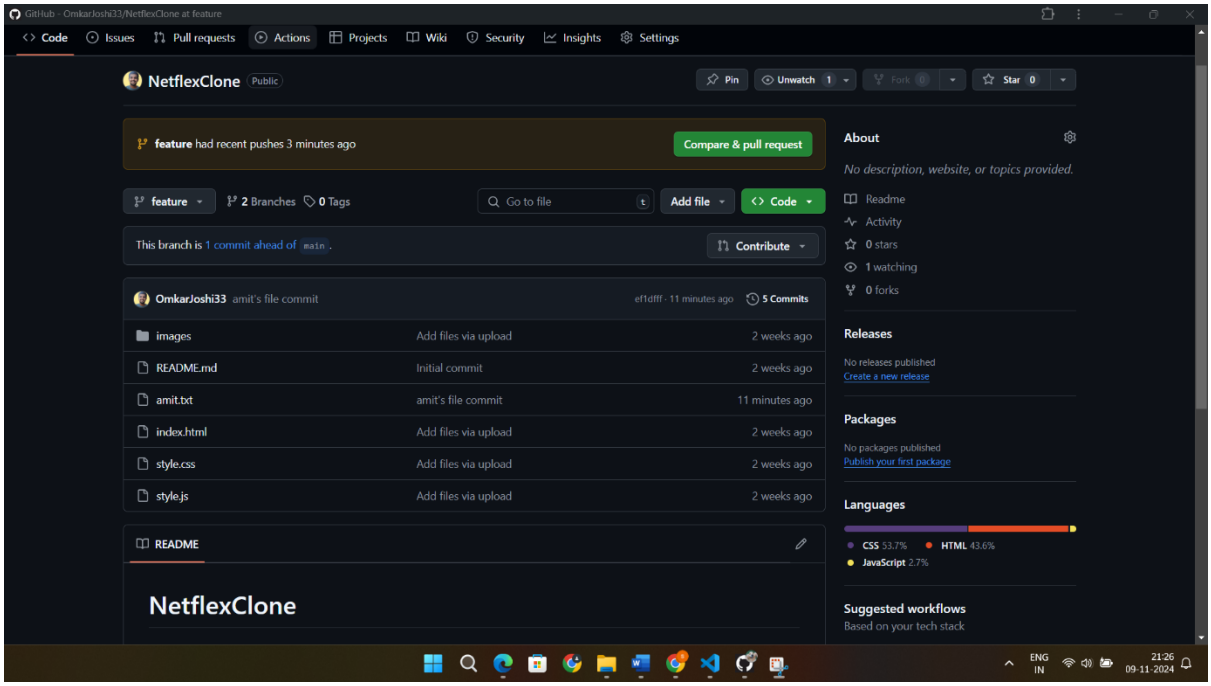
git remote add origin <repository_url>

● Replace <repository_url> with the URL you copied from GitHub.

● Push the "feature" branch to GitHub: git push origin feature

● Check your GitHub repository to confirm that the new branch "feature" is available.

Command/output::




```
PS C:\Users\Omkar\Desktop\new folder (2)\NetFlexClone\NetflexClone> git remote add origin https://github.com/OmkarJoshi33/devops_practicals.git
error: remote origin already exists.
PS C:\Users\Omkar\Desktop\new folder (2)\NetFlexClone\NetflexClone> git push origin feature
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 317 bytes | 317.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'feature' on GitHub by visiting:
remote:      https://github.com/OmkarJoshi33/NetflexClone/pull/new/feature
remote:
To https://github.com/OmkarJoshi33/NetflexClone.git
 * [new branch]      feature -> feature
PS C:\Users\Omkar\Desktop\new folder (2)\NetFlexClone\NetflexClone>
```

Task 4: Collaborating through Pull Requests

● After the pull request is merged, update your local repository:

git checkout main. git pull origin main

● Create a pull request on GitHub: ● Go to the repository on GitHub.

● Click on "Pull Requests" and then "New Pull Request."

● Choose the base branch (usually "main" or "master") and the compare branch ("feature").

● Review the changes and click "Create Pull Request."

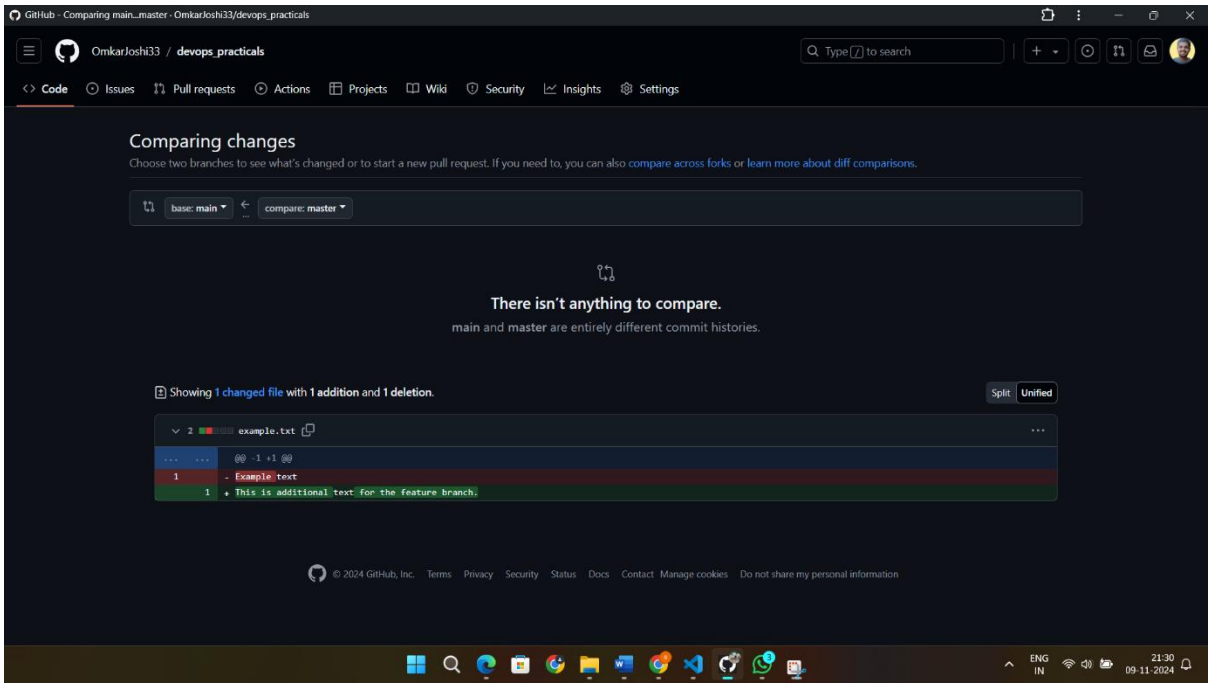
● Review and merge the pull request:

● Add a title and description for the pull request.

● Assign reviewers if needed.

● Once the pull request is approved, merge it into the base branch.

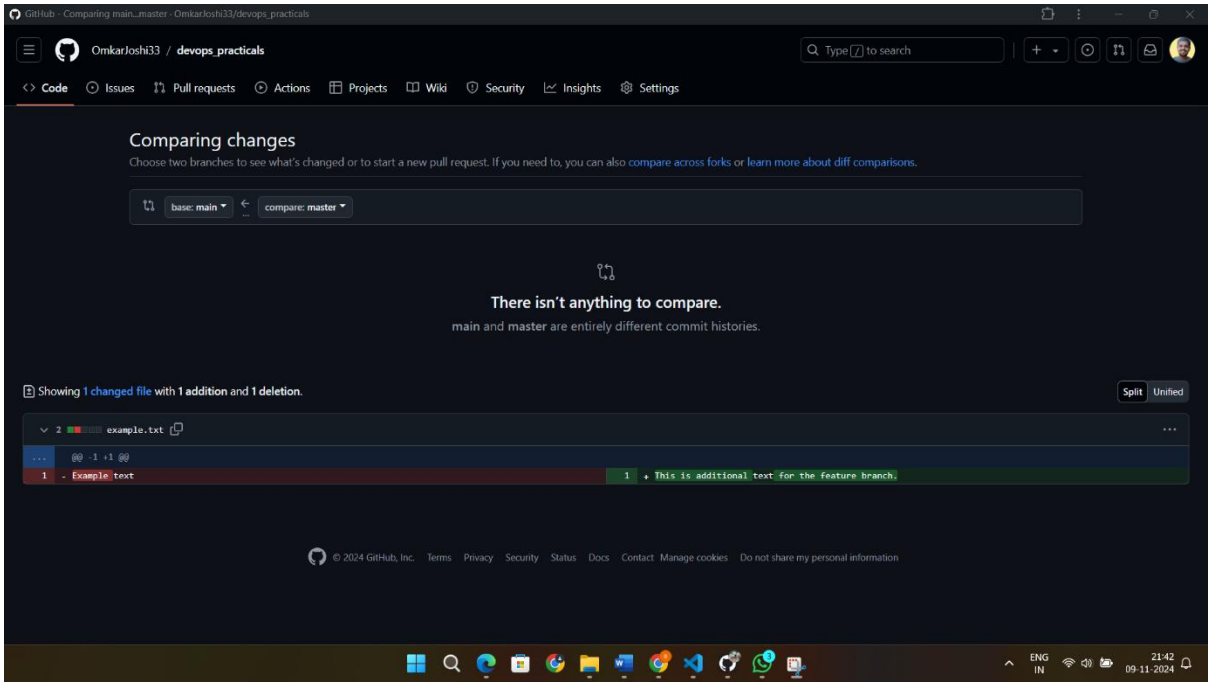
Command/output



Task 5: Syncing Changes

● After the pull request is merged, update your local repository: git checkout main. git pull origin main

Output::

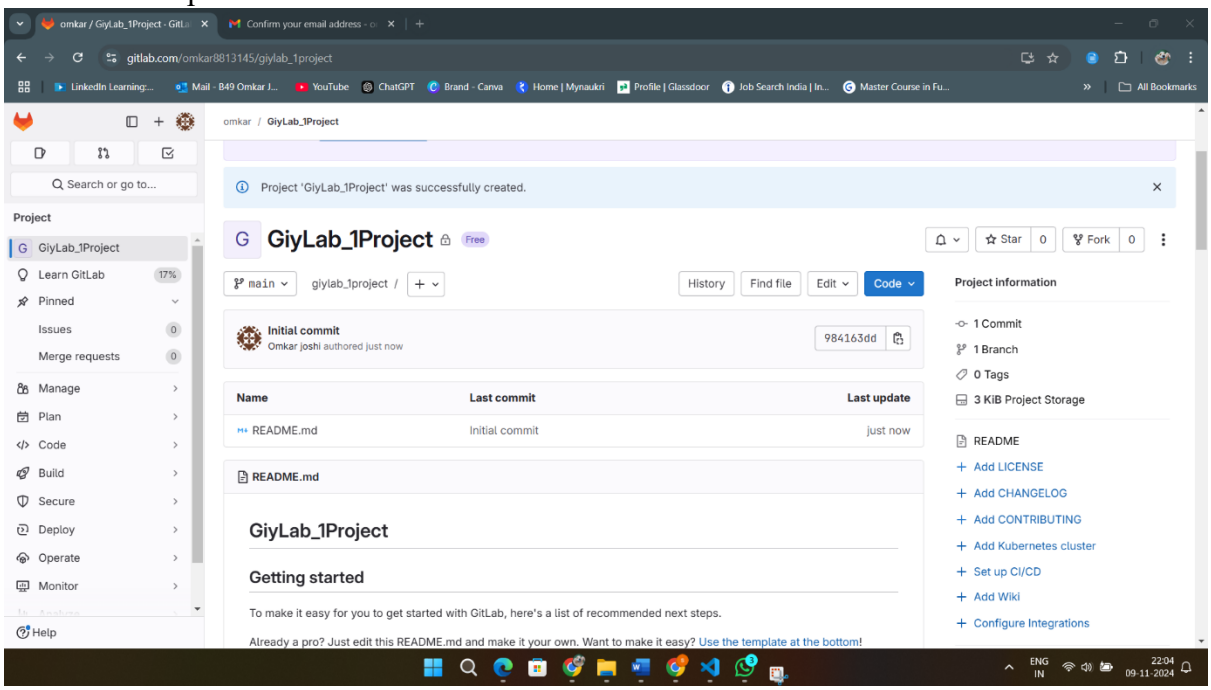


Implement GitLab Operations using Git.

Task 1: Creating a Repository

- Sign in to your GitLab account.
- Click the "New" button to create a new project.
- Choose a project name, visibility level (public, private), and other settings.
- Click "Create project."

Command/output



Task 2: Cloning a Repository

● Open your terminal or command prompt.

● Navigate to the directory where you want to clone the repository.

● Copy the repository URL from GitLab.

● Run the following command: git clone <repository_url>

● Replace <repository_url> with the URL you copied from GitLab.

● This will clone the repository to your local machine.

Command/output:

Task 3: Making Changes and Creating a Branch

● Navigate into the cloned repository: cd <repository_name>

● Create a new text file named "example.txt" using a text editor.

● Add some content to the "example.txt" file.

● Save the file and return to the command line.

● Check the status of the repository: git status

● Stage the changes for commit: git add example.txt

● Commit the changes with a descriptive message:

git commit -m "Add content to example.txt"

● Create a new branch named "feature": git branch feature

● Switch to the "feature" branch: git checkout feature

Command/output::

```
PS C:\Users\Omkar\Desktop\new folder (2)\random> cd recloth
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth> echo example.txt
example.txt
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth> code example.txt
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth> git status
On branch master
Your branch is up to date with 'origin/master'.

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

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth> git add exaple.txt
fatal: pathspec 'exaple.txt' did not match any files
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth> git add example.txt
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth> git commit -m "1st commit"
[master 3136b31] 1st commit
 1 file changed, 2 insertions(+)
 create mode 100644 example.txt
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth> git branch feature
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth> git checkout feature
Switched to branch 'feature'
PS C:\Users\Omkar\Desktop\new folder (2)\random\recloth>
```

Task 4: Pushing Changes to GitLab

● Add Repository URL in a variable git remote add origin <repository_url>

● Replace <repository_url> with the URL you copied from GitLab.

● Push the "feature" branch to GitLab: git push origin feature

● Check your GitLab repository to confirm that the new branch "feature" is available.

● After the merge request is merged, update your local repository:

git checkout main, git pull origin main

Command/output::

Task 5: Collaborating through Merge Requests

1. Create a merge request on GitLab:

 Go to the repository on GitLab.

 Click on "Merge Requests" and then "New Merge Request."

 Choose the source branch ("feature") and the target branch ("main" or master").

 Review the changes and click "Submit merge request."

2. Review and merge the merge request:

 Add a title and description for the merge request.

 Assign reviewers if needed.

 Once the merge request is approved, merge it into the target branch.

Task 6: Syncing Changes

4

Implement BitBucket Operations using Git.

Task 1: Creating a Repository

● Sign in to your Bitbucket account.

● Click the "Create" button to create a new repository.

● Choose a repository name, visibility (public or private), and other settings.

● Click "Create repository."

Task 2: Cloning a Repository

● Open your terminal or command prompt.

● Navigate to the directory where you want to clone the repository.

● Copy the repository URL from Bitbucket.

● Run the following command: git clone <repository_url>

● Replace <repository_url> with the URL you copied from Bitbucket.

● This will clone the repository to your local machine.

Task 3: Making Changes and Creating a Branch

● Navigate into the cloned repository: cd <repository_name>

● Create a new text file named "example.txt" using a text editor.

● Add some content to the "example.txt" file.

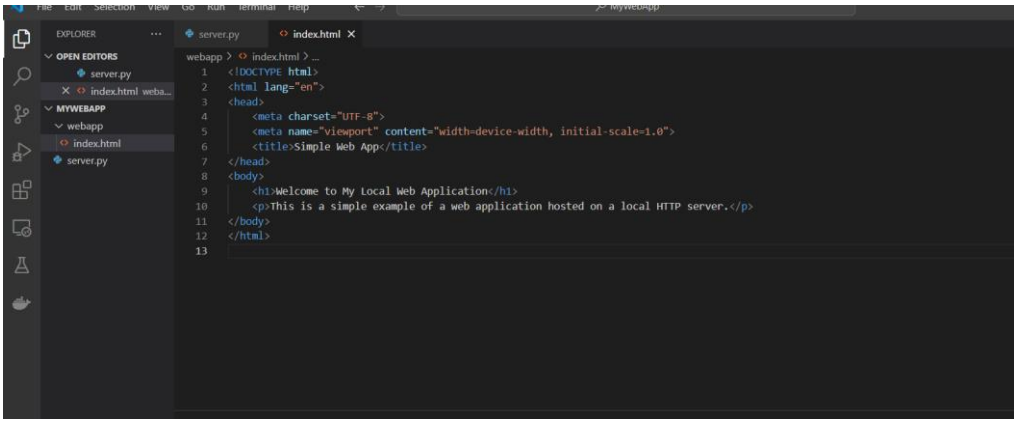
● Save the file and return to the command line.

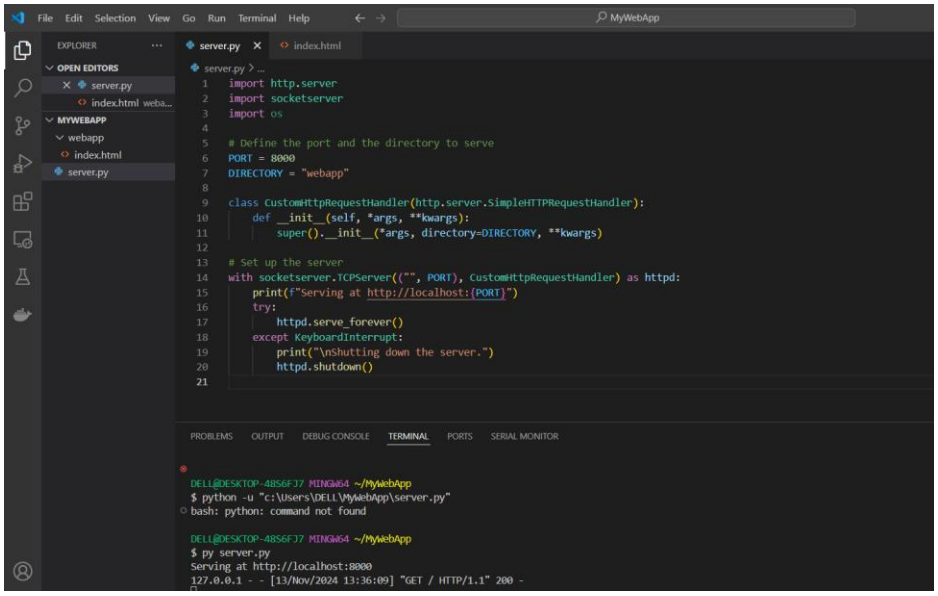
● Check the status of the repository: git status

33

● Stage the changes for commit: git add example.txt
● Commit the changes with a descriptive message: git commit -m "Add content to example.txt"
● Create a new branch named "feature": git branch feature
● Switch to the "feature" branch: git checkout feature
Task 4: Pushing Changes to Bitbucket
● Add Repository URL in a variable: git remote add origin <repository_url>
● Replace <repository_url> with the URL you copied from Bitbucket.
● Push the "feature" branch to Bitbucket: git push origin feature
● Check your Bitbucket repository to confirm that the new branch "feature" is available.
Task 5: Collaborating through Pull Requests
1. Create a pull request on Bitbucket:
● After the pull request is merged, update your local repository:
git checkout main, git pull origin main
○ Go to the repository on Bitbucket. ○ Click on "Create pull request."
○ Choose the source branch ("feature") and the target branch ("main" or "master").
○ Review the changes and click "Create pull request."
2. Review and merge the pull request:
○ Add a title and description for the pull request.
○ Assign reviewers if needed.
○ Once the pull request is approved, merge it into the target branch.
Task 6: Syncing Changes
5 Applying CI/CD Principles to Web Development Using Jenkins, Git, and Local HTTP Server
Step 1: Set Up the Web Application and Local HTTP Server
● Create a simple web application or use an existing one. Ensure it can be hosted by a local HTTP server.

● Set up a local HTTP server (e.g., Apache or Nginx) to serve the web application. Ensure it's configured correctly and running.



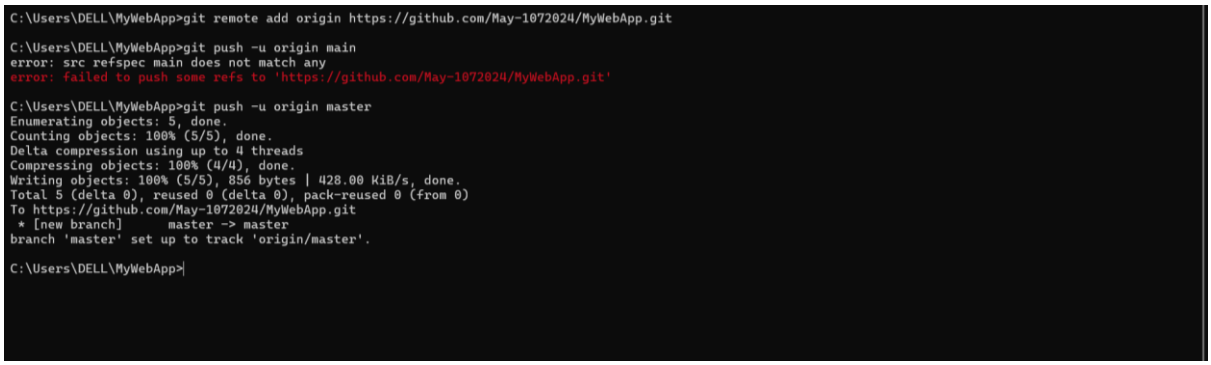


Step 2: Set Up a Git Repository

Create a Git repository for your web application. Initialize it with the following commands: git init, git add, git commit -m "Initial commit"

● Configure Jenkins to work with Git by setting up your Git credentials in the Jenkins Credential Manager.

● Create a remote Git repository (e.g., on GitHub or Bitbucket) to push your code to later.

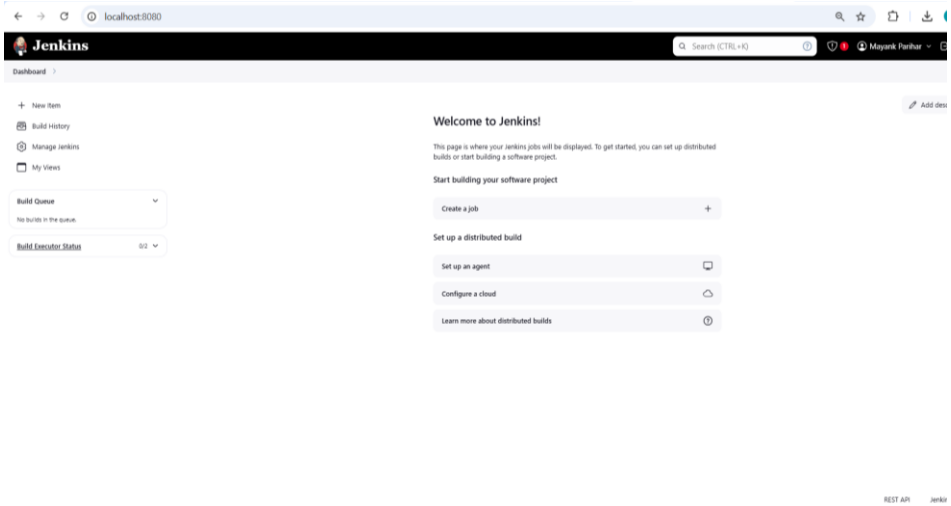


Step 3: Install and Configure Jenkins

● Download and install Jenkins following the instructions for your operating system (https://www.jenkins.io/download/).

● Open Jenkins in your web browser (usually at http://localhost:8080) and complete the initial setup.

● Install the necessary plugins for Git integration, job scheduling, and webhook support.

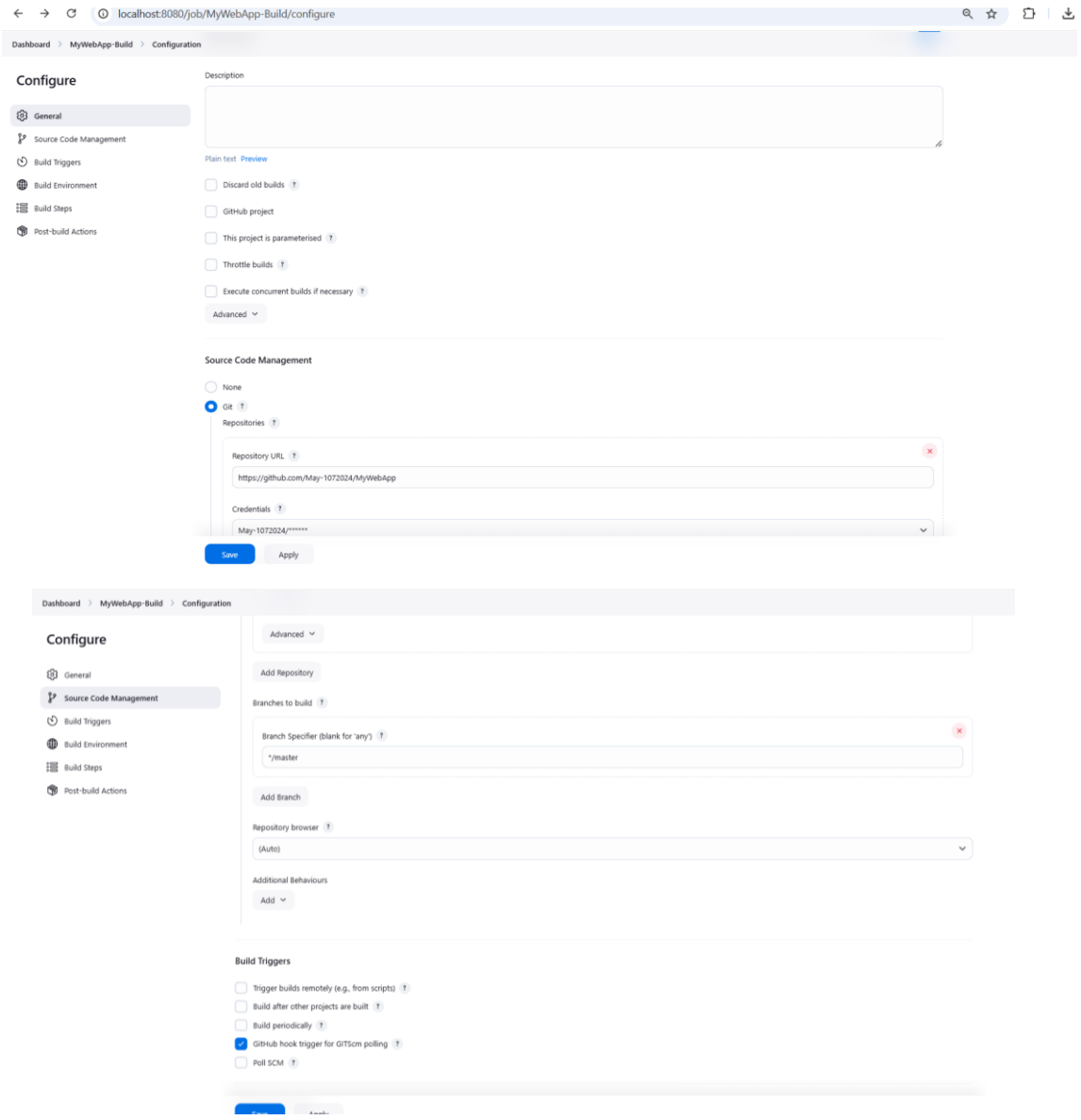


Step 4: Create a Jenkins Job

● Create a new Jenkins job using the "Freestyle project" type.

● Configure the job to use a webhook trigger. You can do this by selecting the "GitHub

hook trigger for GITScm polling" option in the job's settings.



Step 5: Set Up the Jenkins Job (Using Execute Shell)

● In the job configuration, go to the "Build" section.

● Add a build step of type "Execute shell."

● In the "Command" field, define the build and deployment steps using shell commands. For example:

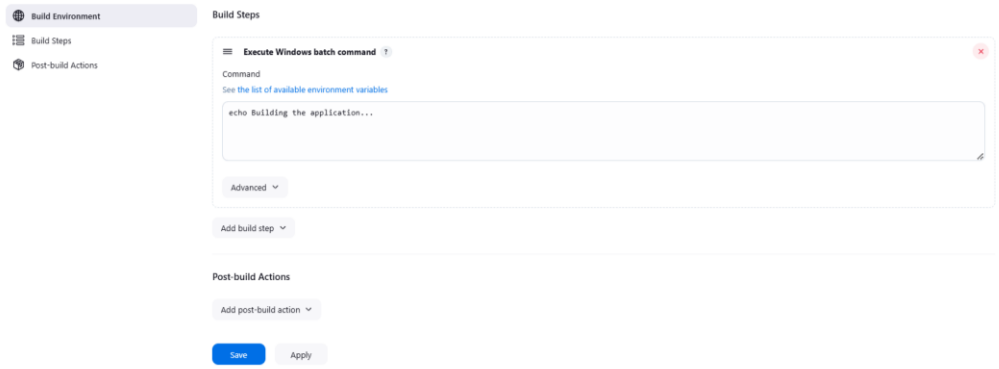
Checkout code from Git

Build your web application (e.g., npm install, npm run build)

Copy the build artefacts to the local HTTP server directory

rm -rf /var/www/html/webdirectory/*

cp -r * /var/www/html/webdirectory/



Step 6: Set Up a Webhook in Git Repository

Visit the URL of your local HTTP server to verify that the web application has been updated with the latest changes.

● In your Git repository (e.g., on GitHub), go to "Settings" and then "Webhooks."

● Create a new webhook, and configure it to send a payload to the Jenkins webhook

URL (usually http://jenkins-server/github-webhook/). Make sure to set the content type to "application/json."

● OR use “GitHub hook trigger for GITScm polling?” Under Build Trigger

Step 7: Trigger the CI/CD Pipeline

● Push changes to your Git repository. The webhook should trigger the Jenkins job automatically, executing the build and deployment steps defined in the "Execute Shell" build step.

● Monitor the Jenkins job's progress in the Jenkins web interface.

Step 8: Verify the CI/CD Pipeline

6 Exploring Containerization and Application Deployment with Docker

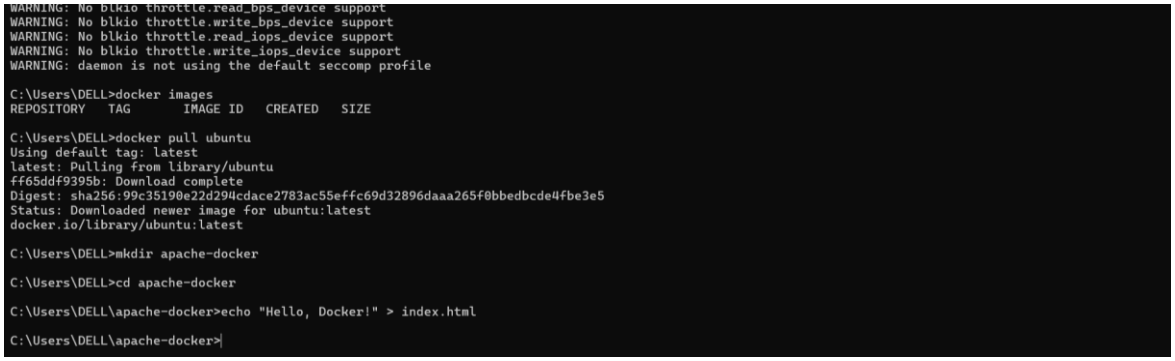
Task 1: Install Docker

● If you haven't already, install Docker on your computer by following the instructions provided on the Docker website (https://docs.docker.com/get-docker/).

Task 2: Create a Simple HTML Page

● Create a directory for your web server project.

● Inside this directory, create a file named index.html with a simple "Hello, Docker!" message. This will be the content served by your Apache web server.



Task 3: Create a Dockerfile

● Create a Dockerfile in the same directory as your web server project. The Dockerfile defines how your Apache web server application will be packaged into a Docker container.

Here's an example: //teacher can use other example for demo

Dockerfile

Use an official Apache image as the base image

FROM httpd:2.4

Copy your custom HTML page to the web server's document root

COPY index.html /usr/local/apache2/htdocs/

```
C:\Users\DELL\apache-docker>touch Dockerfile
'touch' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\DELL\apache-docker>echo. > Dockerfile

C:\Users\DELL\apache-docker>
```

Task 4: Build the Docker Image

● Build the Docker image by running the following command in the same directory as your Dockerfile: docker build -t my-apache-server .

● Replace my-apache-server with a suitable name for your image.

```
C:\Users\DELL\apache-docker>docker build -t apache-hello .
[+] Building 13.2s (8/8) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                 0.1s
 => => transferring dockerfile: 105B                                                                                                 0.0s
 => [internal] load metadata for docker.io/library/httpd:latest                                                                      3.6s
 => [auth] library/httpd:pull token for registry-1.docker.io                                                                         0.0s
 => [internal] load .dockerignore                                                                                                    0.1s
 => => transferring context: 2B                                                                                                      0.0s
 => [internal] load build context                                                                                                    0.1s
 => => transferring context: 56B                                                                                                     0.0s
 => [1/2] FROM docker.io/library/httpd:latest@sha256:6bdbdf5ac16ac3d6ef543a693fd5dfafae2428b4b0cdc52a480166603a069136               9.1s
 => => resolve docker.io/library/httpd:latest@sha256:6bdbdf5ac16ac3d6ef543a693fd5dfafae2428b4b0cdc52a480166603a069136               0.1s
 => => sha256:334a67c7f78bddd148eb4d24df67ed1ed4f3856c0d7edd9679a6a400dc54783c 291B / 291B                                           0.2s
 => => sha256:0062038102c990223e92353e4bab9ae1ee778f28f80a051879c5768dda84b10d 26.04MB / 26.04MB                                     5.6s
 => => sha256:3ed0d9182dde2c70e0817dc7ef1f02aad4da2eef2656098597257098a3bc8b5a 4.20MB / 4.20MB                                       1.9s
 => => sha256:d675ed392a91a9d52b3d1b8873c6b44be3738bc6d1b322fb64c1404de8d54072 145B / 145B                                           0.9s
 => => sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B                                             0.8s
 => => sha256:2d429b9e73a6cf90a5bb85105c8118b30a1b2deedeae3ea9587055ffcb80eb45 29.13MB / 29.13MB                                     6.1s
 => => extracting sha256:2d429b9e73a6cf90a5bb85105c8118b30a1b2deedeae3ea9587055ffcb80eb45                                            1.0s
 => => extracting sha256:d675ed392a91a9d52b3d1b8873c6b44be3738bc6d1b322fb64c1404de8d54072                                            0.0s
 => => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1                                            0.0s
 => => extracting sha256:3ed0d9182dde2c70e0817dc7ef1f02aad4da2eef2656098597257098a3bc8b5a                                            0.2s
 => => extracting sha256:0062038102c990223e92353e4bab9ae1ee778f28f80a051879c5768dda84b10d                                            0.5s
 => => extracting sha256:334a67c7f78bddd148eb4d24df67ed1ed4f3856c0d7edd9679a6a400dc54783c                                            0.0s
 => [2/2] COPY ./index.html /usr/local/apache2/htdocs/                                                                               0.2s
 => exporting to image                                                                                                               0.4s
 => => exporting layers                                                                                                              0.2s
 => => exporting manifest sha256:c324d95543843fab298ef1bab1a710868b5b00b488c59065cc8bc028dccd6fe8                                    0.0s
 => => exporting config sha256:38d815f762cd2bbf0f11522b7e63ae66f33dcb758cc7b25397634758 31f9f817                                     0.0s
 => => exporting attestation manifest sha256:cb7e0b24af6fc82795d134b97c82398bde5f6ff047c4e78a04a17e7f5ab9a3fc                        0.0s
 => => exporting manifest list sha256:6cd287b9ac0be2309f76fd50e95de6c55a6541466a4ae8003fa8975fc3743a5e                               0.0s
 => => naming to docker.io/library/apache-hello:latest                                                                               0.0s
 => => unpacking to docker.io/library/apache-hello:latest                                                                            0.0s

C:\Users\DELL\apache-docker>
```

Task 5: Run the Docker Container

Start a Docker container from the image you built:

docker run -p 8080:80 -d my-apache-server

● Optionally, remove the container and the Docker image:

docker rm <container_id>, docker rmi my-apache-server

● This command maps port 80 in the container to port 8080 on your host machine and runs the container in detached mode.

```
C:\Users\DELL\apache-docker>docker run -d -p 8080:80 apache-hello
f928b4a101863769339e5d46a0e6e22c18a6a2946b04ebed8c23a0bd38d389f3
docker: Error response from daemon: Ports are not available: exposing port TCP 0.0.0.0:8080 -> 0.0.0.0:0: listen tcp4 0.0.0.0:8080: bind: An attempt was mad
e to access a socket in a way forbidden by its access permissions.

C:\Users\DELL\apache-docker>docker run -d -p 8080:80 apache-hello
7a57c296c3e1644821b20f4738e2ced9febe3de668ae8a0f062ad1d435efe8d0

C:\Users\DELL\apache-docker>
```

Task 6: Access Your Apache Web Server

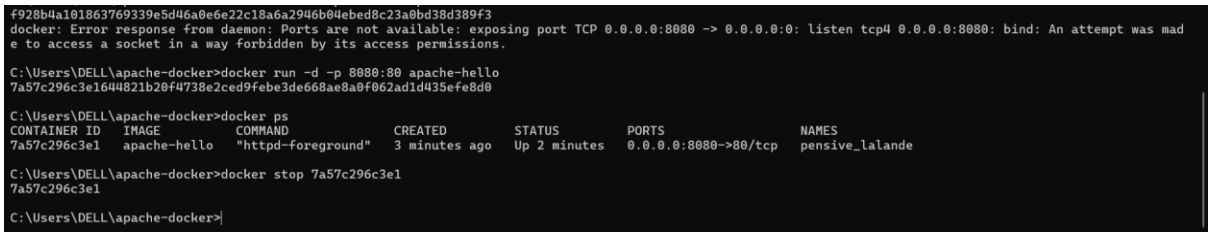
Access your Apache web server by opening a web browser and navigating to http://localhost:8080. You should see the "Hello, Docker!" message served by your

Apache web server running within the Docker container.

Task 7: Cleanup

Stop the running Docker container: docker stop <container_id>

● Replace <container_id> with the actual ID of your running container.



7 Applying CI/CD Principles to Web Development Using Jenkins, Git, usingDocker Containers

Task 1: Set Up the Web Application and Git Repository

● Create a simple web application or use an existing one. Ensure it can be hosted in a Docker container.

● Initialise a Git repository for your web application and push it to GitHub.

Task 2: Install and Configure Jenkins

● Install Jenkins on your computer or server following the instructions for your operating system (https://www.jenkins.io/download/).

● Open Jenkins in your web browser (usually at http://localhost:8080) and complete the initial setup, including setting up an admin user and installing necessary plugins.

● Set Branches to build -> Branch Specifier to the working Git branch (ex

*/master)

● Configure Jenkins to work with Git by setting up Git credentials in the Jenkins Credential Manager.

Task 3: Create a Jenkins Job

● Create a new Jenkins job using the "Freestyle project" type.

● In the job configuration, specify a name for your job and choose "This project is parameterized."

● Add a "String Parameter" named GIT_REPO_URL and set its default value to your Git repository URL.

● In the job configuration, go to the "Build Triggers" section and select the "GitHub hook trigger for GITScm polling" option. This enables Jenkins to listen for GitHub webhook triggers.

Task 4: Configure Build Steps

● In the job configuration, go to the "Build" section.

● Add build steps to execute Docker commands for building and deploying the containerized web application. Use the following commands:

Remove the existing container if it exists: docker rm --force container1

Build a new Docker image: docker build -t nginx-image1 .

Run the Docker container: docker run -d -p 8081:80 --name=container1 nginx image1

Access your web application by opening a web browser and navigating to http://localhost:8081 (or the appropriate URL if hosted elsewhere).

● These commands remove the existing container (if any), build a Docker image named "nginx-image1," and run a Docker container named "container1" on port 8081.

Task 5: Set Up a GitHub Webhook

● In your GitHub repository, navigate to "Settings" and then "Webhooks."

● Create a new webhook, and configure it to send a payload to the Jenkins webhook URL (usually http://jenkins-server/github-webhook/). Set the content type to "application/json."

Task 6: Trigger the CI/CD Pipeline

● Push changes to your GitHub repository. The webhook will trigger the Jenkins job automatically, executing the build and deployment steps defined in the job configuration.

● Monitor the Jenkins job's progress in the Jenkins web interface.

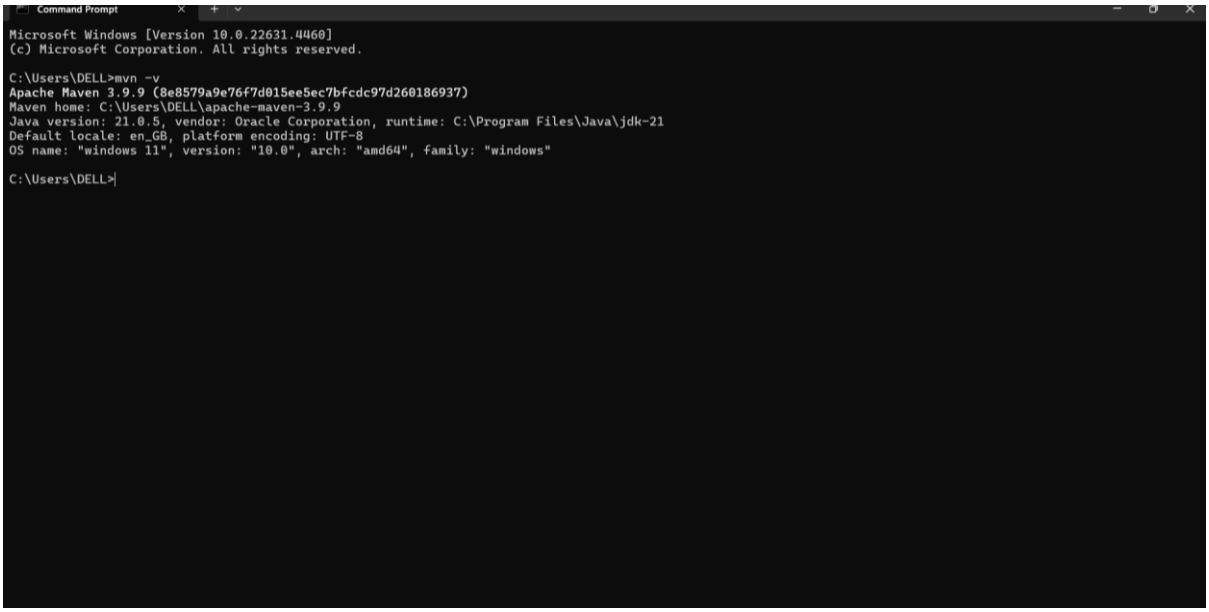
Task 7: Verify the Deployment

8 Practical Maven Assignment

Task 1: Setting up Maven Environment

 installing Maven on local machines and configuring it properly.

 understand the directory structure and how to create a basic Maven project.



Task 2: Creating a Simple Maven Project

 create a simple Java project using Maven.

 This could involve creating classes, adding dependencies, and configuring the project's POM (Project Object Model).

Task 3: Dependency Management

 Introduce dependency management in Maven.

 Assign tasks such as adding external dependencies to the project using Maven Central repository and managing version conflicts.

Task 4: Build Automation

 set up automated builds using Maven.

 configure Maven to compile the code, run tests, and generate artifacts like JAR files.