Slip-1

Q1)create a simple project using any programming language and perform some operation on the project using git. Make your project as a git repository add your file staging area and commit changes with a descriptive message

.

1)Create folder and into that folder create Python file

# hello\_git.py

print("Hello, Git!")

2) Navigate to the project folder:

cd /path/to/project/folder

3)open cmd with your project path:

git init

git add hello\_git.py

git commit -m "Initial commit: Added hello\_git.py to print a welcome message"

git status

git log

python hello\_git.py

Q2)create a simple Java project using Maven. adding dependencies, and Configuring the project's POM file and compile code using maven tool.

1. Ensure you have Maven installed. Verify using:

mvn –version

1. Open your terminal and execute the following:

mvn archetype:generate -DgroupId=com.example -DartifactId=SimpleMavenProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

1. Navigate to the project folder:

cd SimpleMavenProject

1. Open pom.xml in your editor and configure dependencies.

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>SimpleMavenProject</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit Dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

1. Open src/main/java/com/example/App.java and modify it:

package com.example;

public class App {

public static void main(String[] args) {

System.out.println("Hello, Maven Project!");

}

}

1. Compile the Code

mvn compile

1. Package the project into a JAR file:

mvn package

1. To execute the generated JAR file:

java -cp target/SimpleMavenProject-1.0-SNAPSHOT.jar com.example.App

1. Verify Dependencies

mvn dependency:tree

Slip -2

Q1) Task 1: Dockerfile Creation and Build

Create a Dockerfile to containerize a simple HTML web page.

The Dockerfile should use an nginx base image and copy the HTML page to the default directory served by nginx.

Build the Docker image and tag it appropriately.

1) Verify Docker installation using:

docker –version

2) Create a project folder

mkdir nginx-html-docker

cd nginx-html-docker

3) Inside the folder, create an HTML file

echo "<!DOCTYPE html>

<html>

<head><title>My Dockerized Web Page</title></head>

<body><h1>Hello, Docker!</h1></body>

</html>" > index.html

4) Create a file named Dockerfile (no extension) in the same folder:

# Use the official nginx image as the base

FROM nginx:latest

# Copy the HTML file to the default nginx directory

COPY index.html /usr/share/nginx/html/

# Expose port 80

EXPOSE 80

5) Build the Docker image and tag it (e.g., my-nginx-web):

docker build -t my-nginx-web .

6) Verify the image is built:

docker images

Task 2: Running and Managing Containers

* + - Run the Docker container, mapping port 8080 on the host to the container's port 80.
    - Confirm the web page is accessible through localhost:8080.

Stop and remove the container after testing.

7) Start a container using the built image:

docker run -d -p 8080:80 --name my-nginx-container my-nginx-web

8) Verify the container is running:

docker ps

9) Open a browser and navigate to

http://localhost:8080

10) Stop and remove the running container

docker stop my-nginx-container

docker rm my-nginx-container

Q.2) create a simple project And push on remote server (like github ) using git. and perform some operation. And displays a chronological history of commits.

1) Create a new folder for your project:

mkdir simple-git-project

cd simple-git-project

2) Add a file to the project:

echo "Hello, Git and GitHub!" > hello.txt

3) Initialize a Git repository in the project directory:

git init

4) Add the file to the staging area:

git add hello.txt

5) Commit the changes with a descriptive message:

git commit -m "Initial commit: Added hello.txt with a welcome message"

6) Create a new repository on GitHub or any remote Git server (e.g., simple-git-project).

Add the remote repository:

git remote add origin https://github.com/your-username/simple-git-project.git

7) Push the changes to the remote repository:

git branch -M main

git push -u origin main

8) Modify the File:

echo "This is an update to demonstrate commit history." >> hello.txt

9) Stage and Commit the Changes:

git add hello.txt

git commit -m "Updated hello.txt with additional content"

10) Push the Changes:

git push

11) **View Commit History:** Display a chronological history of commits:

git log –oneline

Slip-3

Q1) Applying CI/CD Principles to Web Development Using Jenkins, Git, and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Q2) Create a simple project, push it to a remote repository on GitHub, and create a new branch. Merge this branch into the main branch and display a chronological history of commits.

**1)Create a new folder for your project:**

mkdir git-branch-demo

cd git-branch-demo

2) **Add a file to the project:**

echo "Hello, Git Branch Demo!" > hello.txt

3) Initialize a Git repository:

git init

4) Stage and commit the changes:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5) Create a new repository on GitHub (e.g., git-branch-demo).

Add the GitHub remote:

git remote add origin https://github.com/your-username/git-branch-demo.git

6) Push the main branch to the remote repository:

git branch -M main

git push -u origin main

7) Create a new branch

git checkout -b feature-branch

8) Make changes in the new branch:

echo "This is a new feature." >> hello.txt

9) Stage and commit the changes:

git add hello.txt

git commit -m "Added a new feature in hello.txt"

10) Push the branch to the remote repository:

git push -u origin feature-branch

11) Switch back to the main branch:

git checkout main

12) Merge the feature-branch into main:

git merge feature-branch

13) Push the updated main branch to the remote repository:

git push

14) Linear Commit History:

git log –oneline

15) Graphical Commit History:

git log --oneline --graph

Slip-4

Q1) Configure Maven to compile the code, run tests, and generate artifacts like

JAR files.

1) Use the Maven archetype to generate a project structure:

mvn archetype:generate -DgroupId=com.example -DartifactId=SampleMavenProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

2) Navigate to the project directory:

cd SampleMavenProject

3) Open pom.xml and ensure it has the following sections:

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>SampleMavenProject</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit Dependency for Testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

4) Create a simple Java class in src/main/java/com/example/App.java:

package com.example;

public class App {

public static void main(String[] args) {

System.out.println("Hello, Maven!");

}

}

5) Add a test class in src/test/java/com/example/AppTest.java

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AppTest {

@Test

public void shouldAnswerWithTrue() {

assertTrue(true);

}

}

6) **Compile the Code and run test:**

mvn compile

mvn test

mvn package

target/SampleMavenProject-1.0-SNAPSHOT.jar

mvn clean

7) Execute the JAR file

java -cp target/SampleMavenProject-1.0-SNAPSHOT.jar com.example.App

Q2) Create a simple project and use Git commands to check the status, view log history, see differences between the working directory and the last commit, make a commit and display a chronological history of commits.

1) Create a new project folder:

mkdir git-demo

cd git-demo

2) Add a file to the project:

echo "Hello, Git Demo!" > hello.txt

3) Initialize a Git repository:

git init

4) Add the file to the staging area and commit it:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5) To see the current status of your working directory:

git status

6) Modify the file:

echo "This is a change to demonstrate git commands." >> hello.txt

7) Check the status again:

git status

8) To see the commit history:

git log

9) To view a condensed log with commit messages:

git log –oneline

10)To see the differences between the working directory and the last commit

git diff

11) Stage the changes:

git add hello.txt

12) Commit the changes with a descriptive message:

git commit -m "Updated hello.txt to demonstrate git commands"

13) View a concise history:

git log –oneline

14) View a graphical history:

git log --oneline –graph

Slip-5

Q1) Create a simple project, push it to a remote repository on BitBucket, and create a new branch. Merge this branch into the main branch and display a chronological history of commits.

1) Create a Project Directory:

mkdir bitbucket-demo

cd bitbucket-demo

2) Add a File to the Project:

echo "Hello, Bitbucket!" > hello.txt

3) Initialize a Git Repository:

git init

4) Stage and Commit the File:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5)Create a Repository on Bitbucket:

Go to Bitbucket.

Create a new repository (e.g., bitbucket-demo).

6) Add the Remote Repository:

git remote add origin https://<username>@bitbucket.org/<username>/bitbucket-demo.git

7) Push the Main Branch to Bitbucket:

git branch -M main

git push -u origin main

8) Create and Switch to a New Branch:

git checkout -b feature-branch

9) Make Changes in the New Branch

echo "This is a new feature." >> hello.txt

10) Stage and Commit the Changes

git add hello.txt

git commit -m "Added a new feature in hello.txt"

11) Push the Branch to Bitbucket

git push -u origin feature-branch

12) Switch Back to the Main Branch:

git checkout main

13) Merge the Feature Branch into the Main Branch:

git merge feature-branch

14) Push the Updated Main Branch to Bitbucket:

git push

15) Linear Commit History

git log –oneline

16) **Graphical Commit History**

git log --oneline –graph

Q2) Create a simple project, push it to a remote repository on Github , and create a new branch. Merge this branch into the main branch and display a chronological history of commits.and Pull the changes on your local machine.

1)Create a Project Directory:

mkdir github-demo

cd github-demo

2) Add a File to the Project:

echo "Hello, GitHub!" > hello.txt

3)Initialize a Git Repository:

git init

4)Stage and Commit the File:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5) Create a Repository on GitHub:

Go to GitHub.

Create a new repository (e.g., github-demo).

6) Add the Remote Repository:

git remote add origin https://github.com/<your-username>/github-demo.git

7) Push the Main Branch to GitHub:

git branch -M main

git push -u origin main

8) Create and Switch to a New Branch:

git checkout -b feature-branch

9)Make Changes in the New Branch:

echo "This is a new feature." >> hello.txt

10)Stage and Commit the Changes:

git add hello.txt

git commit -m "Added a new feature in hello.txt"

11)Push the Branch to GitHub:

git push -u origin feature-branch

12)Switch Back to the Main Branch:

git checkout main

13)Merge the Feature Branch into the Main Branch:

git merge feature-branch

14)Push the Updated Main Branch to GitHub:

git push

15) Ensure you are on the main branch locally:

git checkout main

16)Pull the latest changes from the remote repository:

git pull

17) Linear Commit History:

git log --oneline

18) Graphical Commit History:

git log --oneline –graph

Slip-6

Q.1 ) Create a simple project, push it to a remote repository on GitLab, and create a new branch. Merge this branch into the main branch and display a chronological history of commits

1) Create a Project Directory: Open a terminal and create a new directory for your project:

mkdir gitlab-demo

cd gitlab-demo

2) Add a File to the Project: Create a simple text file (hello.txt) in the project directory:

echo "Hello, GitLab!" > hello.txt

3) Initialize a Git Repository: Initialize a Git repository in the project directory:

git init

4) Stage and Commit the File: Stage the file and make an initial commit:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5) Create a Repository on GitLab:

Go to GitLab.

Create a new repository (e.g., gitlab-demo).

6) Add the Remote GitLab Repository: Add your GitLab repository as a remote origin:

git remote add origin https://gitlab.com/<your-username>/gitlab-demo.git

7)Push the Main Branch to GitLab:

git branch -M main

8)Push the main branch to GitLab:

git push -u origin main

9) Create and Switch to a New Branch: Create a new branch (e.g., feature-branch) and switch to it:

git checkout -b feature-branch

10) Make Changes in the New Branch: Edit the hello.txt file:

echo "This is a new feature." >> hello.txt

11) Stage and Commit the Changes: Stage and commit the changes to the new branch:

git add hello.txt

git commit -m "Added a new feature in hello.txt"

12) Push the Feature Branch to GitLab: Push the new branch to GitLab:

git push -u origin feature-branch

13) Switch Back to the Main Branch: Checkout the main branch:

git checkout main

14)Merge the Feature Branch into the Main Branch: Merge the changes from feature-branch into main:

git merge feature-branch

15)Push the Updated Main Branch to GitLab: Push the merged changes to GitLab:

git push

16) View the Commit History

git log –oneline

17) View the Commit History in a Graphical Format

git log --oneline –graph

18) Pull Changes on Your Local Machine

git checkout main

19) Pull the Latest Changes

git pull origin main

Q.2) Create a simple project, push it to a remote repository on Bitbucket , and create a new branch. Merge this branch into the main branch and display a chronological history of commits.and Pull the changes on your local machine.

1)Create a new folder for your project:

mkdir bitbucket-demo

cd bitbucket-demo

2)Add a file to the project:

echo "Hello, Bitbucket Demo!" > hello.txt

3)Initialize a Git repository:

git init

4)Add the file to the staging area and commit it:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5) Create a new repository on Bitbucket:

Go to Bitbucket.

Create a new repository (e.g., bitbucket-demo).

6)Add the Bitbucket remote URL:

Get the URL from your Bitbucket repository page (e.g., https://your-bitbucket-username@bitbucket.org/your-bitbucket-username/bitbucket-demo.git).

Run the following command to add the remote:

git remote add origin https://your-bitbucket-username@bitbucket.org/your-bitbucket-username/bitbucket-demo.git

7)Push the main branch to Bitbucket:

git branch -M main

git push -u origin main

8) Create a new branch (e.g., feature-branch):

git checkout -b feature-branch

9) Make changes in the new branch:

echo "This is a feature change." >> hello.txt

10) Stage and commit the changes:

git add hello.txt

git commit -m "Added feature change in hello.txt"

11) Push the branch to Bitbucket:

git push -u origin feature-branch

12) Switch back to the main branch:

git checkout main

13) Merge the feature-branch into main:

git merge feature-branch

14) Push the updated main branch to Bitbucket:

git push

15) View a chronological history of commits:

git log –oneline

16) View a graphical history:

git log --oneline –graph

17) Pull the latest changes from Bitbucket:

git pull origin main

Slip-7

Q.1 ) Create a simple project, push it to a remote repository on GitLab, and create a new branch. Merge this branch into the main branch and display a chronological history of commits. Pull the changes on your local machine.

1) Create a Project Directory: Open a terminal and create a new directory for your project:

mkdir gitlab-demo

cd gitlab-demo

2) Add a File to the Project: Create a simple text file (hello.txt) in the project directory:

echo "Hello, GitLab!" > hello.txt

3) Initialize a Git Repository: Initialize a Git repository in the project directory:

git init

4) Stage and Commit the File: Stage the file and make an initial commit:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5) Create a Repository on GitLab:

Go to GitLab.

Create a new repository (e.g., gitlab-demo).

6) Add the Remote GitLab Repository: Add your GitLab repository as a remote origin:

git remote add origin https://gitlab.com/<your-username>/gitlab-demo.git

7)Push the Main Branch to GitLab:

git branch -M main

8)Push the main branch to GitLab:

git push -u origin main

9) Create and Switch to a New Branch: Create a new branch (e.g., feature-branch) and switch to it:

git checkout -b feature-branch

10) Make Changes in the New Branch: Edit the hello.txt file:

echo "This is a new feature." >> hello.txt

11) Stage and Commit the Changes: Stage and commit the changes to the new branch:

git add hello.txt

git commit -m "Added a new feature in hello.txt"

12) Push the Feature Branch to GitLab: Push the new branch to GitLab:

git push -u origin feature-branch

13) Switch Back to the Main Branch: Checkout the main branch:

git checkout main

14)Merge the Feature Branch into the Main Branch: Merge the changes from feature-branch into main:

git merge feature-branch

15)Push the Updated Main Branch to GitLab: Push the merged changes to GitLab:

git push

16) View the Commit History

git log –oneline

17) View the Commit History in a Graphical Format

git log --oneline –graph

18) Pull Changes on Your Local Machine

git checkout main

19) Pull the Latest Changes

git pull origin main

Q.2) Create CI using Webhook and deploy a project using Jenkins Execute shell.

1) Create a project folder

mkdir my-project

cd my-project

2)Create a simple file in the project:

echo "echo Hello, Jenkins!" > deploy.sh

chmod +x deploy.sh

3)Initialize a Git repository and commit the project:

git init

git add .

git commit -m "Initial commit with deploy script"

4) Push your project to a remote repository (e.g., GitHub, GitLab):

Create a repository on GitHub or GitLab.

Push your project to the remote repository:

git remote add origin <your-repo-url>

git push -u origin master

5) Access Jenkins Dashboard:

Open your Jenkins instance in the browser (http://localhost:8080).

Create a New Job:

In the Jenkins dashboard, click on New Item → Enter a project name (e.g., MyProject-CI) → Select Freestyle project → Click OK.

Configure the Job:

Under the Source Code Management section, select Git.

Enter the Git repository URL (e.g., https://github.com/your-username/my-project.git).

Optionally, provide credentials if required for private repositories.

Set the branch to master (or the default branch of your repository).

Add Build Step:

Under Build, click Add build step → Select Execute shell.

In the Command box, add the shell command to deploy the project (e.g., running a deploy script):

./deploy.sh

Save the Job:

Click Save to save your job configuration.

6) Configure Webhook in Your Git Repository:

If using GitHub:

Go to your GitHub repository → Settings → Webhooks → Add webhook.

Set the Payload URL to your Jenkins server's URL:

perl

http://<your-jenkins-server>/github-webhook/

Set Content Type to application/json.

Choose Just the push event to trigger the build when you push changes.

Click Add webhook.

7) If using GitLab:

Go to your GitLab repository → Settings → Webhooks.

Set the URL to your Jenkins webhook URL (e.g., http://<your-jenkins-server>/gitlab-webhook/).

Choose the push event to trigger the build.

Click Add webhook.

Verify Webhook:

Make sure that your repository is pushing events to Jenkins. You can check this in Jenkins under Manage Jenkins → System Log.

8) Make a Change and Push:

Make a change in your project (e.g., modify deploy.sh).

Commit and push the changes:

git add .

git commit -m "Updated deploy script"

git push

Slip-8

Q.1 ) Create a new file on a separate branch, make some changes to this file, and then merge these changes into the main branch using bitBucket interface.

1) Log in to Bitbucket: Go to Bitbucket and log into your account.

Create a new repository:

Click on Create repository.

Give your repository a name (e.g., my-bitbucket-project).

Choose whether the repository should be public or private.

Click Create repository.

2) **Clone the Repository** to your local machine

git clone https://your-username@bitbucket.org/your-username/my-bitbucket-project.git

cd my-bitbucket-project

3) Create a new branch:

You can create and switch to a new branch using the following Git commands:

git checkout -b new-feature-branch

4) Create a new file in your project folder:

echo "This is a new file on a separate branch" > newfile.txt

5) Add and commit the new file:

git add newfile.txt

git commit -m "Added newfile.txt on a new branch"

6) Push the changes to Bitbucket:

git push -u origin new-feature-branch

7) Edit the new file: Open newfile.txt in your editor and make some changes.

echo "This is an updated version of newfile.txt" > newfile.txt

8) Commit the changes:

git add newfile.txt

git commit -m "Updated newfile.txt"

9) Push the changes to Bitbucket:

git push origin new-feature-branch

10) Go to Bitbucket: Open your repository on Bitbucket (e.g., https://bitbucket.org/your-username/my-bitbucket-project).

Navigate to Pull Requests:

On the left sidebar, click on Pull requests.

Click the Create pull request button.

Create the Pull Request:

Select the source branch (e.g., new-feature-branch) and the destination branch (e.g., main).

Add a title and description for the pull request, explaining what changes you made.

Click Create pull request.

Review and Merge the Pull Request:

After creating the pull request, Bitbucket will show the changes made in the new-feature-branch relative to the main branch.

You or someone else can review the changes.

Once ready, click Merge to merge the new-feature-branch into main.

Complete the Merge:

After merging, Bitbucket will show the success message and you can see the merged changes in the main branch.

11) Pull Changes

git checkout main

git pull origin main

Q.2) Outline the process of setting up a CI/CD pipeline for a web application using Jenkins, Git, and a local HTTP server. Include the configuration of Jenkins, the webhook setup, and the execution of build and deployment steps.

1) Install Jenkins:

Install Jenkins on a server or local machine. Refer to the official Jenkins installation guide for detailed steps.

After installation, start Jenkins and open the web interface by navigating to http://localhost:8080.

2) Unlock Jenkins:

When you first access Jenkins, it will prompt you for an unlock key.

Find the unlock key in the secrets/initialAdminPassword file in the Jenkins installation directory. Copy the key and paste it into the web interface.

3) Install Suggested Plugins:

During the initial setup, Jenkins will ask you to install plugins. Select "Install suggested plugins" to get essential plugins for a typical CI/CD pipeline.

4) Create an Admin User:

After plugin installation, set up an admin user and complete the setup.

5) Create a New Jenkins Job:

In the Jenkins dashboard, click on New Item.

Enter a project name (e.g., webapp-ci-pipeline).

Select Freestyle project and click OK.

6) Configure the Job:

Source Code Management:

Select Git.

Enter the Git repository URL (e.g., https://github.com/your-username/webapp.git).

If your repository is private, enter the required credentials for Git.

7) Build Triggers (Webhook configuration will go here later):

You can enable Poll SCM to poll the repository for changes (e.g., H/5 \* \* \* \*), but it's better to use a webhook, which we'll configure later.

8) Build Environment (Optional):

If you need to set environment variables, you can do that here.

9) Add Build Step:

Click on Add build step and select Execute shell.

In the Command section, enter the build and deployment commands for your web application. For example, if you're using a Node.js application:

# Install dependencies

npm install

# Run tests

npm test

# Build the application (this step will vary depending on your project)

npm run build

# Copy the build files to the web server directory (for Apache/Nginx)

cp -r ./build/\* /var/www/html/

10) Configure the Webhook in GitHub:

Go to your repository on GitHub.

Click on Settings → Webhooks → Add webhook.

In the Payload URL, enter the URL for your Jenkins server's webhook endpoint. The URL should look like:

http://<jenkins-server>/github-webhook/

Set the Content type to application/json.

Under Which events would you like to trigger this webhook?, select Just the push event (or you can select Let me select individual events and choose events like push).

Click Add webhook to save.

If you are using GitLab, follow a similar process:

Go to your repository in GitLab → Settings → Webhooks.

Add the Jenkins webhook URL (http://<jenkins-server>/gitlab-webhook/).

Verify the Webhook:

Ensure that the webhook triggers a Jenkins build by making a change to your repository and pushing it. Jenkins should automatically start the build process when the push occurs.

11) Install Apache or Nginx on the server (if not already installed):

For Apache (on Ubuntu/Debian):

sudo apt update

sudo apt install apache2

For Nginx:

sudo apt update

sudo apt install nginx

Set up the web server directory:

For Apache:

The default web root is /var/www/html/, so your application should be deployed there (as shown in the Jenkins shell script above).

For Nginx:

The default web root is usually /usr/share/nginx/html/.

Ensure the web server is running:

Start the web server:

Apache: sudo systemctl start apache2

Nginx: sudo systemctl start nginx

Grant the necessary permissions to the web server to write to the web root directory:

sudo chown -R www-data:www-data /var/www/html

Slip 9

Q1) Implement Bitbucket Operations Using Git

* + - Task 1: Create a new repository on Bitbucket and clone it locally.
    - Task 2: Create a file example.txt, add and commit it, and create a branch feature.
    - Task 3: Push the feature branch to Bitbucket and create a pull request. Review and merge the pull request.

Create a New Repository on Bitbucket:

Go to Bitbucket and log in with your credentials.

Click on Create repository from the Bitbucket dashboard.

Give your repository a name (e.g., my-bitbucket-project).

Set the repository as either Private or Public, depending on your needs.

Click on Create repository.

Clone the Repository Locally:

After creating the repository, you'll be redirected to the repository's page.

Copy the repository URL (HTTPS or SSH).

Open your terminal and clone the repository using Git:

git clone https://your-username@bitbucket.org/your-username/my-bitbucket-project.git

cd my-bitbucket-project

Create a New File:

echo "This is a new example file" > example.txt

Add the File to Git:

Stage the example.txt file to be committed:

git add example.txt

Commit the Changes:

Commit the file with a descriptive commit message:

git commit -m "Add example.txt"

Create a New Branch (feature):

Create and switch to a new branch called feature:

git checkout -b feature

Push the feature Branch to Bitbucket:

Push the new branch to Bitbucket:

git push -u origin feature

Create a Pull Request on Bitbucket:

Go to your Bitbucket repository on the web.

Click on Pull requests in the left-hand sidebar.

Click the Create pull request button.

In the Source section, select the feature branch.

In the Destination section, select the main branch (or the branch you want to merge into).

Add a title and description for the pull request, explaining the changes.

Click Create pull request.

Review and Merge the Pull Request:

Review the changes in the pull request.

After reviewing, click Merge to merge the changes from the feature branch into the main branch.

Optionally, you can delete the feature branch after the merge to keep the repository clean.

Q.2) Explain how to set up and manage dependencies in a Maven project. Describe the structure of a pom.xml file, adding dependencies, and how Maven handles build automation for tasks like compiling code, running tests, and generating JAR files.

1) To create a new Maven project, use the following command:

mvn archetype:generate -DgroupId=com.example -DartifactId=my-maven-project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

2) Structure of the pom.xml File

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>my-maven-project</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<dependencies>

<!-- Example of adding a dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Build plugins (optional) -->

</plugins>

</build>

</project>

3) Adding Dependencies to pom.xml

<dependencies>

<!-- JUnit for running tests -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

4) Compile Code

mvn compile

5) Run Tests

mvn test

6) Generate JAR File

mvn package

7) Install the JAR file into the local Maven repository:

mvn install

8) Clean the project (delete previous build files):

mvn clean

Slip-10

Q1) Install Docker on your system and create a simple "Hello, World!" application using HTML.

Create a Dockerfile to containerize the application, using an official web server image as the base. Build the Docker image, tag it, and run a container, making the application accessible on a local port (e.g., http://localhost:8080).

1) Verify Docker installation using:

docker –version

2) Create a project folder

mkdir nginx-html-docker

cd nginx-html-docker

3) Inside the folder, create an HTML file

echo "<!DOCTYPE html>

<html>

<head><title>My Dockerized Web Page</title></head>

<body><h1>Hello, Docker!</h1></body>

</html>" > index.html

4) Create a file named Dockerfile (no extension) in the same folder:

# Use the official nginx image as the base

FROM nginx:latest

# Copy the HTML file to the default nginx directory

COPY index.html /usr/share/nginx/html/

# Expose port 80

EXPOSE 80

5) Build the Docker image and tag it (e.g., my-nginx-web):

docker build -t my-nginx-web .

6) Verify the image is built:

docker images

Stop and remove the container after testing.

7) Start a container using the built image:

docker run -d -p 8080:80 --name my-nginx-container my-nginx-web

8) Verify the container is running:

docker ps

9) Open a browser and navigate to

http://localhost:8080

10) Stop and remove the running container

docker stop my-nginx-container

docker rm my-nginx-container

Q.2) Git and GitHub Repository Management.

Task 1: Repository Setup and Initial Commit

* Set up a local Git repository and create a file named project.md with a brief description of a hypothetical project.
* Initialize the repository, add project.md, commit the changes, and push to a GitHub repository.

Task 2: Branching and Merging

* Create a new branch called feature-branch and make additional changes to project.md.
* Commit the changes in the feature-branch, switch back to main, and merge feature-branch into main.
* Push the updated main branch to GitHub, ensuring the merge is reflected.

Task 1: Repository Setup and Initial Commit

Set up a Local Git Repository: First, open your terminal (or Git Bash) and navigate to the folder where you want to create your project.

mkdir my-git-project

cd my-git-project

git init

2) Create project.md with a Brief Description

echo "# My Hypothetical Project" > project.md

echo "This is a brief description of the project." >> project.md

3) Add project.md to Git: Stage the file for commit:

git add project.md

4) Commit the Changes

git commit -m "Initial commit with project description"

5) Push the Repository to GitHub:

Create a new repository on GitHub (visit GitHub and create a new repository).

Copy the repository URL (HTTPS or SSH) from GitHub.

git remote add origin https://github.com/your-username/my-git-project.git

git branch -M main # Ensure you're on the main branch

git push -u origin main

Task 2: Branching and Merging

1) Create a New Branch (feature-branch) and Make Changes

git checkout -b feature-branch

echo "Additional details for the project can be added here." >> project.md

2) Commit the Changes in feature-branch

git add project.md

git commit -m "Added more details to project description in feature-branch"

3) Switch Back to the main Branch

git checkout main

4) Merge feature-branch into main

git merge feature-branch

5) Push the Updated main Branch to GitHub

git push origin main

Slip-11

Q.1 ) Containerize a basic application and deploy it using Docker.

1) Create a project folder: First, create a directory for your application.

mkdir my-docker-app

cd my-docker-app

2) Create the index.html file

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>My Basic Web Application</title>

</head>

<body>

<h1>Welcome to My Basic Web Application!</h1>

<p>This is a simple application running inside a Docker container.</p>

</body>

</html>

3) Create a Dockerfile

# Use nginx as the base image

FROM nginx:alpine

# Copy the index.html file to the default nginx directory

COPY index.html /usr/share/nginx/html/index.html

# Expose port 80 (default for nginx)

EXPOSE 80

4) Build the Docker Image

docker build -t my-web-app .

5) Run the Docker Container

docker run -d -p 8080:80 --name web-app-container my-web-app

6) Access the Application

http://localhost:8080

7) Stop the Docker Container:

docker stop web-app-container

8)Remove the Docker Container:

docker rm web-app-container

Q.2) Git and GitHub Repository Management

Objective: To manage repositories, branching, and merging in Git and GitHub.

Task 1: Repository Setup and Initial Commit

* Set up a local Git repository and create a file named project.md with a brief description of a hypothetical project.
* Initialize the repository, add project.md, commit the changes, and push to a GitHub repository.

Task 2: Branching and Merging

* Create a new branch called feature-branch and make additional changes to project.md.
* Commit the changes in the feature-branch, switch back to main, and merge feature-branch into main.
* Push the updated main branch to GitHub, ensuring the merge is reflected.

Task 1: Repository Setup and Initial Commit

Set up a Local Git Repository: First, open your terminal (or Git Bash) and navigate to the folder where you want to create your project.

mkdir my-git-project

cd my-git-project

git init

2) Create project.md with a Brief Description

echo "# My Hypothetical Project" > project.md

echo "This is a brief description of the project." >> project.md

3) Add project.md to Git: Stage the file for commit:

git add project.md

4) Commit the Changes

git commit -m "Initial commit with project description"

5) Push the Repository to GitHub:

Create a new repository on GitHub (visit GitHub and create a new repository).

Copy the repository URL (HTTPS or SSH) from GitHub.

git remote add origin https://github.com/your-username/my-git-project.git

git branch -M main # Ensure you're on the main branch

git push -u origin main

Task 2: Branching and Merging

1) Create a New Branch (feature-branch) and Make Changes

git checkout -b feature-branch

echo "Additional details for the project can be added here." >> project.md

2) Commit the Changes in feature-branch

git add project.md

git commit -m "Added more details to project description in feature-branch"

3) Switch Back to the main Branch

git checkout main

4) Merge feature-branch into main

git merge feature-branch

5) Push the Updated main Branch to GitHub

git push origin main

Slip-12

Q.1 ) Applying CI/CD Principles to Web Development Using Jenkins, Git, and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Q.2) Containerize a basic application and deploy it using Docker.

1) Create a project folder: First, create a directory for your application.

mkdir my-docker-app

cd my-docker-app

2) Create the index.html file

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>My Basic Web Application</title>

</head>

<body>

<h1>Welcome to My Basic Web Application!</h1>

<p>This is a simple application running inside a Docker container.</p>

</body>

</html>

3) Create a Dockerfile

# Use nginx as the base image

FROM nginx:alpine

# Copy the index.html file to the default nginx directory

COPY index.html /usr/share/nginx/html/index.html

# Expose port 80 (default for nginx)

EXPOSE 80

4) Build the Docker Image

docker build -t my-web-app .

5) Run the Docker Container

docker run -d -p 8080:80 --name web-app-container my-web-app

6) Access the Application

http://localhost:8080

7) Stop the Docker Container:

docker stop web-app-container

8)Remove the Docker Container:

docker rm web-app-container

Slip-13

* 1. ) Bitbucket Repository and Branch Management Task 1: Repository Setup and Branching.
     + Create a repository on Bitbucket and clone it locally.
     + Create a branch development, add a new file, commit the changes, and push it to the development branch on Bitbucket.

Task 2: Pull Request and Code Review

* + - In Bitbucket, create a pull request to merge development into main.
    - Assign a reviewer (or self-review) and comment on any changes before merging.

1) Log in to Bitbucket: Go to Bitbucket and log into your account.

Create a new repository:

Click on Create repository.

Give your repository a name (e.g., my-bitbucket-project).

Choose whether the repository should be public or private.

Click Create repository.

2) **Clone the Repository** to your local machine

git clone https://your-username@bitbucket.org/your-username/my-bitbucket-project.git

cd my-bitbucket-project

3) Create a new branch:

You can create and switch to a new branch using the following Git commands:

git checkout -b new-feature-branch

4) Create a new file in your project folder:

echo "This is a new file on a separate branch" > newfile.txt

5) Add and commit the new file:

git add newfile.txt

git commit -m "Added newfile.txt on a new branch"

6) Push the changes to Bitbucket:

git push -u origin new-feature-branch

7) Edit the new file: Open newfile.txt in your editor and make some changes.

echo "This is an updated version of newfile.txt" > newfile.txt

8) Commit the changes:

git add newfile.txt

git commit -m "Updated newfile.txt"

9) Push the changes to Bitbucket:

git push origin new-feature-branch

10) Go to Bitbucket: Open your repository on Bitbucket (e.g., https://bitbucket.org/your-username/my-bitbucket-project).

Navigate to Pull Requests:

On the left sidebar, click on Pull requests.

Click the Create pull request button.

Create the Pull Request:

Select the source branch (e.g., new-feature-branch) and the destination branch (e.g., main).

Add a title and description for the pull request, explaining what changes you made.

Click Create pull request.

Review and Merge the Pull Request:

After creating the pull request, Bitbucket will show the changes made in the new-feature-branch relative to the main branch.

You or someone else can review the changes.

Once ready, click Merge to merge the new-feature-branch into main.

Complete the Merge:

After merging, Bitbucket will show the success message and you can see the merged changes in the main branch.

11) Pull Changes

git checkout main

git pull origin main

Q2) Task 1: Configuring Jenkins with GitHub

* Install and set up Jenkins.
* Integrate Jenkins with a GitHub repository, ensuring Jenkins triggers a build on every push.

Task 2: Creating a CI/CD Pipeline

* Create a Jenkins pipeline that clones your GitHub repository, builds a simple web application, and archives the build artifacts.
* Configure a post-build action to notify your GitHub repository of the build status.

Task 3: Adding Deployment Step

* Extend the pipeline to deploy the built application to a local web server (use shell commands for deployment).
* Set up a webhook in GitHub to trigger the Jenkins job automatically upon a code push.

1. Install Jenkins:

Use the following commands to install Jenkins on your local machine (Ubuntu example)

sudo apt update

sudo apt install openjdk-11-jdk -y

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt update

sudo apt install jenkins -y

2. Start Jenkins:

sudo systemctl start jenkins

sudo systemctl enable jenkins

3. Access Jenkins:

Open your browser and navigate to `http://localhost:8080`.

Retrieve the initial admin password:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

- Log in, install suggested plugins, and create an admin user.

2 Install Required Plugins:

- Go to `Manage Jenkins > Manage Plugins`.

- Install:

1) Git Plugin

2) GitHub Integration Plugin

3) Pipeline Plugin

3) Add GitHub Credentials:

- Go to `Manage Jenkins > Credentials > System > Global Credentials`.

- Add your GitHub credentials (username/password or personal access token).

4. Configure Jenkins Job to Use GitHub:

- Create a new Freestyle project in Jenkins.

- Under the Source Code Management section, select `Git` and add your repository URL.

- Use the credentials you added earlier.

4. Configure Build Triggers:

- In the project configuration, go to Build Triggers.

- Check GitHub hook trigger for GITScm polling.

5. Set Up a GitHub Webhook:

- Go to your GitHub repository settings.

- Navigate to Webhooks > Add Webhook.

- Add the Jenkins URL as the payload URL (e.g., `http://your-jenkins-url/github-webhook/`).

- Set the content type to `application/json`.

- Select Just the push event and click Add Webhook.

Task 2: Creating a CI/CD Pipeline

1. Go to Jenkins and create a new Pipeline project.

2. In the project configuration, add the following Jenkinsfile as the pipeline script:

```groovy

pipeline {

agent any

stages {

stage('Clone Repository') {

steps {

git branch: 'main', url: 'https://github.com/your-username/your-repo.git'

}

}

stage('Build Application') {

steps {

echo 'Building application...'

sh 'mkdir build && echo "Build Successful!" > build/output.txt'

}

}

stage('Archive Artifacts') {

steps {

archiveArtifacts artifacts: 'build/output.txt', fingerprint: true

}

}

}

post {

always {

echo 'Notifying GitHub of build status...'

// Add a GitHub notification script or plugin configuration

}

}

}

3. Save the pipeline and trigger a build.

Task 3: Adding Deployment Step

Step 1: Extend the Pipeline for Deployment

Modify the `Jenkinsfile` to add a deployment step:

```groovy

pipeline {

agent any

stages {

stage('Clone Repository') {

steps {

git branch: 'main', url: 'https://github.com/your-username/your-repo.git'

}

}

stage('Build Application') {

steps {

echo 'Building application...'

sh 'mkdir build && echo "Build Successful!" > build/output.txt'

}

}

stage('Archive Artifacts') {

steps {

archiveArtifacts artifacts: 'build/output.txt', fingerprint: true

}

}

stage('Deploy Application') {

steps {

echo 'Deploying application...'

sh '''

cp -r build /var/www/html/

sudo systemctl restart apache2

'''

}

}

}

post {

always {

echo 'Notifying GitHub of build status...'

// Add a GitHub notification script or plugin configuration

}

}

}

Step 2: Set Up Web Server

1. Install Apache or Nginx:

sudo apt update

sudo apt install apache2 -y

2. Verify the web server is running:

sudo systemctl start apache2

sudo systemctl enable apache2

3. The deployed application files will be copied to `/var/www/html`.

Slip-14

Q1) Task 1: Creating and Configuring a Bitbucket Repository

* + - Create a new repository on Bitbucket with a meaningful name and description.
    - Clone the repository to your local machine and add a README.md file.
    - Push the README.md file to the Bitbucket repository.

Task 2: Setting Up Access Controls

* + - Invite a collaborator to the repository with "Read" access only.
    - Update the collaborator's access to "Write" and confirm they can push to the repository.
    - Document the access control changes in a separate file and push it to the repository.

Task 3: Working with Branches

* + - * Create a new branch named feature/update-readme.
      * Make a minor update to the README.md file on this branch and push it.
      * Create a pull request (PR) for merging this branch into the main branch and merge it after review.

1) Create a Repository on Bitbucket

Log in to your Bitbucket account and click Create repository.

Provide a meaningful name (e.g., bitbucket-repo-demo) and description.

Set the repository type to public or private as per your requirements.

Click Create repository.

2) Copy the HTTPS or SSH clone URL from the repository page.

Clone the repository to your local machine:

git clone https://bitbucket.org/your-username/bitbucket-repo-demo.git

cd bitbucket-repo-demo

3) Create a README.md file:

echo "# Bitbucket Repository Demo" > README.md

4) Add and commit the file:

git add README.md

git commit -m "Added README.md file"

5)Push the changes to the Bitbucket repository:

git push origin main

6) Invite a Collaborator

Go to the repository settings on Bitbucket.

Navigate to the User and group access section.

Add the collaborator’s Bitbucket username or email and set their access to Read.

7) Update Collaborator Access

Change the collaborator's access from Read to Write in the same settings.

Ask the collaborator to push a test file to verify their access.

8) Create a file named access\_controls.md

echo "1. Invited collaborator with Read access.\n2. Updated collaborator access to Write.\n3. Confirmed collaborator can push changes." > access\_controls.md

9) Add, commit, and push the file:

git add access\_controls.md

git commit -m "Documented access control changes"

git push origin main

10) Create a new branch named feature/update-readme

git checkout -b feature/update-readme

11) Make an update to the README.md file

echo "\n## Update: Added feature branch details" >> README.md

12) Commit and push the changes

git add README.md

git commit -m "Updated README.md with feature branch details"

git push origin feature/update-readme

13) Create a Pull Request (PR)

Go to the Bitbucket repository and navigate to the Pull Requests tab.

Click Create pull request.

Select the source branch (feature/update-readme) and the target branch (main).

Add a meaningful title and description.

Review the changes and approve the PR.

Merge the PR into the main branch.

Q.2) Applying CI/CD Principles to Web Development Using Jenkins, Git and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Slip-15

* 1. ) Bitbucket Repository and Branch Management Task 1: Repository Setup and Branching
     + Create a repository on Bitbucket and clone it locally.
     + Create a branch development, add a new file, commit the changes, and push it to the development branch on Bitbucket.

Task 2: Pull Request and Code Review

* + - In Bitbucket, create a pull request to merge development into main.
    - Assign a reviewer (or self-review) and comment on any changes before merging.

1) Log in to Bitbucket: Go to Bitbucket and log into your account.

Create a new repository:

Click on Create repository.

Give your repository a name (e.g., my-bitbucket-project).

Choose whether the repository should be public or private.

Click Create repository.

2) **Clone the Repository** to your local machine

git clone https://your-username@bitbucket.org/your-username/my-bitbucket-project.git

cd my-bitbucket-project

3) Create a new branch:

You can create and switch to a new branch using the following Git commands:

git checkout -b new-feature-branch

4) Create a new file in your project folder:

echo "This is a new file on a separate branch" > newfile.txt

5) Add and commit the new file:

git add newfile.txt

git commit -m "Added newfile.txt on a new branch"

6) Push the changes to Bitbucket:

git push -u origin new-feature-branch

7) Edit the new file: Open newfile.txt in your editor and make some changes.

echo "This is an updated version of newfile.txt" > newfile.txt

8) Commit the changes:

git add newfile.txt

git commit -m "Updated newfile.txt"

9) Push the changes to Bitbucket:

git push origin new-feature-branch

10) Go to Bitbucket: Open your repository on Bitbucket (e.g., https://bitbucket.org/your-username/my-bitbucket-project).

Navigate to Pull Requests:

On the left sidebar, click on Pull requests.

Click the Create pull request button.

Create the Pull Request:

Select the source branch (e.g., new-feature-branch) and the destination branch (e.g., main).

Add a title and description for the pull request, explaining what changes you made.

Click Create pull request.

Review and Merge the Pull Request:

After creating the pull request, Bitbucket will show the changes made in the new-feature-branch relative to the main branch.

You or someone else can review the changes.

Once ready, click Merge to merge the new-feature-branch into main.

Complete the Merge:

After merging, Bitbucket will show the success message and you can see the merged changes in the main branch.

11) Pull Changes

git checkout main

git pull origin main

Q.2) Create a simple project using any programming language and perform some operation on the project using git. Make your project as a git repository add your file staging area and commit changes with a descriptive message.

1)Create folder and into that folder create Python file

# hello\_git.py

print("Hello, Git!")

2) Navigate to the project folder:

cd /path/to/project/folder

3)open cmd with your project path:

git init

git add hello\_git.py

git commit -m "Initial commit: Added hello\_git.py to print a welcome message"

git status

git log

python hello\_git.py

Slip-16

Q1)Create a simple Java project using Maven. adding dependencies, and Configuring the project's POM file and compile code using maven tool.

1) Ensure you have Maven installed. Verify using:

mvn –version

2) Open your terminal and execute the following:

mvn archetype:generate -DgroupId=com.example -DartifactId=SimpleMavenProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

3) Navigate to the project folder:

cd SimpleMavenProject

4) Open pom.xml in your editor and configure dependencies.

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>SimpleMavenProject</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit Dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

5) Open src/main/java/com/example/App.java and modify it:

package com.example;

public class App {

public static void main(String[] args) {

System.out.println("Hello, Maven Project!");

}

}

6) Compile the Code

mvn compile

7) Package the project into a JAR file:

mvn package

8) To execute the generated JAR file:

java -cp target/SimpleMavenProject-1.0-SNAPSHOT.jar com.example.App

9) Verify Dependencies

mvn dependency:tree

Q.2) Applying CI/CD Principles to Web Development Using Jenkins, Git and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Slip-17

* 1. ) Bitbucket Repository and Branch Management Task 1: Repository Setup and Branching
     + Create a repository on Bitbucket and clone it locally.
     + Create a branch development, add a new file, commit the changes, and push it to the development branch on Bitbucket.

Task 2: Pull Request and Code Review

* + - In Bitbucket, create a pull request to merge development into main.
    - Assign a reviewer (or self-review) and comment on any changes before merging.

1) Log in to Bitbucket: Go to Bitbucket and log into your account.

Create a new repository:

Click on Create repository.

Give your repository a name (e.g., my-bitbucket-project).

Choose whether the repository should be public or private.

Click Create repository.

2) **Clone the Repository** to your local machine

git clone https://your-username@bitbucket.org/your-username/my-bitbucket-project.git

cd my-bitbucket-project

3) Create a new branch:

You can create and switch to a new branch using the following Git commands:

git checkout -b new-feature-branch

4) Create a new file in your project folder:

echo "This is a new file on a separate branch" > newfile.txt

5) Add and commit the new file:

git add newfile.txt

git commit -m "Added newfile.txt on a new branch"

6) Push the changes to Bitbucket:

git push -u origin new-feature-branch

7) Edit the new file: Open newfile.txt in your editor and make some changes.

echo "This is an updated version of newfile.txt" > newfile.txt

8) Commit the changes:

git add newfile.txt

git commit -m "Updated newfile.txt"

9) Push the changes to Bitbucket:

git push origin new-feature-branch

10) Go to Bitbucket: Open your repository on Bitbucket (e.g., https://bitbucket.org/your-username/my-bitbucket-project).

Navigate to Pull Requests:

On the left sidebar, click on Pull requests.

Click the Create pull request button.

Create the Pull Request:

Select the source branch (e.g., new-feature-branch) and the destination branch (e.g., main).

Add a title and description for the pull request, explaining what changes you made.

Click Create pull request.

Review and Merge the Pull Request:

After creating the pull request, Bitbucket will show the changes made in the new-feature-branch relative to the main branch.

You or someone else can review the changes.

Once ready, click Merge to merge the new-feature-branch into main.

Complete the Merge:

After merging, Bitbucket will show the success message and you can see the merged changes in the main branch.

11) Pull Changes

git checkout main

git pull origin main

Q.2) Create a simple project And push on remote server (like github ) using git. and perform some operation. And displays a chronological history of commits.

1) Create a new folder for your project:

mkdir simple-git-project

cd simple-git-project

2) Add a file to the project:

echo "Hello, Git and GitHub!" > hello.txt

3) Initialize a Git repository in the project directory:

git init

4) Add the file to the staging area:

git add hello.txt

5) Commit the changes with a descriptive message:

git commit -m "Initial commit: Added hello.txt with a welcome message"

6) Create a new repository on GitHub or any remote Git server (e.g., simple-git-project).

Add the remote repository:

git remote add origin https://github.com/your-username/simple-git-project.git

7) Push the changes to the remote repository:

git branch -M main

git push -u origin main

8) Modify the File:

echo "This is an update to demonstrate commit history." >> hello.txt

9) Stage and Commit the Changes:

git add hello.txt

git commit -m "Updated hello.txt with additional content"

10) Push the Changes:

git push

11) **View Commit History:** Display a chronological history of commits:

git log –oneline

Slip-18

Q1) Applying CI/CD Principles to Web Development Using Jenkins, Git and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Q.2) Git and GitHub Repository Management.

Task 1: Repository Setup and Initial Commit

* Set up a local Git repository and create a file named project.md with a brief description of a hypothetical project.
* Initialize the repository, add project.md, commit the changes, and push to a GitHub repository.

Task 2: Branching and Merging

* Create a new branch called feature-branch and make additional changes to project.md.
* Commit the changes in the feature-branch, switch back to main, and merge feature-branch into main.
* Push the updated main branch to GitHub, ensuring the merge is reflected.

Task 1: Repository Setup and Initial Commit

Set up a Local Git Repository: First, open your terminal (or Git Bash) and navigate to the folder where you want to create your project.

mkdir my-git-project

cd my-git-project

git init

2) Create project.md with a Brief Description

echo "# My Hypothetical Project" > project.md

echo "This is a brief description of the project." >> project.md

3) Add project.md to Git: Stage the file for commit:

git add project.md

4) Commit the Changes

git commit -m "Initial commit with project description"

5) Push the Repository to GitHub:

Create a new repository on GitHub (visit GitHub and create a new repository).

Copy the repository URL (HTTPS or SSH) from GitHub.

git remote add origin https://github.com/your-username/my-git-project.git

git branch -M main # Ensure you're on the main branch

git push -u origin main

Task 2: Branching and Merging

1) Create a New Branch (feature-branch) and Make Changes

git checkout -b feature-branch

echo "Additional details for the project can be added here." >> project.md

2) Commit the Changes in feature-branch

git add project.md

git commit -m "Added more details to project description in feature-branch"

3) Switch Back to the main Branch

git checkout main

4) Merge feature-branch into main

git merge feature-branch

5) Push the Updated main Branch to GitHub

git push origin main

Slip-19

Q.2) Git and GitHub Repository Management.

Task 1: Repository Setup and Initial Commit

* Set up a local Git repository and create a file named project.md with a brief description of a hypothetical project.
* Initialize the repository, add project.md, commit the changes, and push to a GitHub repository.

Task 2: Branching and Merging

* Create a new branch called feature-branch and make additional changes to project.md.
* Commit the changes in the feature-branch, switch back to main, and merge feature-branch into main.
* Push the updated main branch to GitHub, ensuring the merge is reflected.

Task 1: Repository Setup and Initial Commit

Set up a Local Git Repository: First, open your terminal (or Git Bash) and navigate to the folder where you want to create your project.

mkdir my-git-project

cd my-git-project

git init

2) Create project.md with a Brief Description

echo "# My Hypothetical Project" > project.md

echo "This is a brief description of the project." >> project.md

3) Add project.md to Git: Stage the file for commit:

git add project.md

4) Commit the Changes

git commit -m "Initial commit with project description"

5) Push the Repository to GitHub:

Create a new repository on GitHub (visit GitHub and create a new repository).

Copy the repository URL (HTTPS or SSH) from GitHub.

git remote add origin https://github.com/your-username/my-git-project.git

git branch -M main # Ensure you're on the main branch

git push -u origin main

Task 2: Branching and Merging

1) Create a New Branch (feature-branch) and Make Changes

git checkout -b feature-branch

echo "Additional details for the project can be added here." >> project.md

2) Commit the Changes in feature-branch

git add project.md

git commit -m "Added more details to project description in feature-branch"

3) Switch Back to the main Branch

git checkout main

4) Merge feature-branch into main

git merge feature-branch

5) Push the Updated main Branch to GitHub

git push origin main

Q.2) Create a simple project, push it to a remote repository on GitHub, and create a new branch. Merge this branch into the main branch and display a chronological history of commits.

1) Create a new folder for your project:

mkdir simple-git-project

cd simple-git-project

2) Add a file to the project:

echo "Hello, Git and GitHub!" > hello.txt

3) Initialize a Git repository in the project directory:

git init

4) Add the file to the staging area:

git add hello.txt

5) Commit the changes with a descriptive message:

git commit -m "Initial commit: Added hello.txt with a welcome message"

6) Create a new repository on GitHub or any remote Git server (e.g., simple-git-project).

Add the remote repository:

git remote add origin https://github.com/your-username/simple-git-project.git

7) Push the changes to the remote repository:

git branch -M main

git push -u origin main

8) Modify the File:

echo "This is an update to demonstrate commit history." >> hello.txt

9) Stage and Commit the Changes:

git add hello.txt

git commit -m "Updated hello.txt with additional content"

10) Push the Changes:

git push

11) **View Commit History:** Display a chronological history of commits:

git log –oneline

Slip-20

Q1) Applying CI/CD Principles to Web Development Using Jenkins, Git and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Q.2) Create a simple project, push it to a remote repository on Github , and create a new branch. Merge this branch into the main branch and display a chronological history of commits. and Pull the changes on your local machine.

1)Create a Project Directory:

mkdir github-demo

cd github-demo

2) Add a File to the Project:

echo "Hello, GitHub!" > hello.txt

3)Initialize a Git Repository:

git init

4)Stage and Commit the File:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5) Create a Repository on GitHub:

Go to GitHub.

Create a new repository (e.g., github-demo).

6) Add the Remote Repository:

git remote add origin https://github.com/<your-username>/github-demo.git

7) Push the Main Branch to GitHub:

git branch -M main

git push -u origin main

8) Create and Switch to a New Branch:

git checkout -b feature-branch

9)Make Changes in the New Branch:

echo "This is a new feature." >> hello.txt

10)Stage and Commit the Changes:

git add hello.txt

git commit -m "Added a new feature in hello.txt"

11)Push the Branch to GitHub:

git push -u origin feature-branch

12)Switch Back to the Main Branch:

git checkout main

13)Merge the Feature Branch into the Main Branch:

git merge feature-branch

14)Push the Updated Main Branch to GitHub:

git push

15) Ensure you are on the main branch locally:

git checkout main

16)Pull the latest changes from the remote repository:

git pull

17) Linear Commit History:

git log --oneline

18) Graphical Commit History:

git log --oneline –graph

Q.1 ) Create a new file on a separate branch, make some changes to this file, and then merge these changes into the main branch using bitBucket interface.

1) Log in to Bitbucket: Go to Bitbucket and log into your account.

Create a new repository:

Click on Create repository.

Give your repository a name (e.g., my-bitbucket-project).

Choose whether the repository should be public or private.

Click Create repository.

2) **Clone the Repository** to your local machine

git clone https://your-username@bitbucket.org/your-username/my-bitbucket-project.git

cd my-bitbucket-project

3) Create a new branch:

You can create and switch to a new branch using the following Git commands:

git checkout -b new-feature-branch

4) Create a new file in your project folder:

echo "This is a new file on a separate branch" > newfile.txt

5) Add and commit the new file:

git add newfile.txt

git commit -m "Added newfile.txt on a new branch"

6) Push the changes to Bitbucket:

git push -u origin new-feature-branch

7) Edit the new file: Open newfile.txt in your editor and make some changes.

echo "This is an updated version of newfile.txt" > newfile.txt

8) Commit the changes:

git add newfile.txt

git commit -m "Updated newfile.txt"

9) Push the changes to Bitbucket:

git push origin new-feature-branch

10) Go to Bitbucket: Open your repository on Bitbucket (e.g., https://bitbucket.org/your-username/my-bitbucket-project).

Navigate to Pull Requests:

On the left sidebar, click on Pull requests.

Click the Create pull request button.

Create the Pull Request:

Select the source branch (e.g., new-feature-branch) and the destination branch (e.g., main).

Add a title and description for the pull request, explaining what changes you made.

Click Create pull request.

Review and Merge the Pull Request:

After creating the pull request, Bitbucket will show the changes made in the new-feature-branch relative to the main branch.

You or someone else can review the changes.

Once ready, click Merge to merge the new-feature-branch into main.

Complete the Merge:

After merging, Bitbucket will show the success message and you can see the merged changes in the main branch.

11) Pull Changes

git checkout main

git pull origin main

Q1) Applying CI/CD Principles to Web Development Using Jenkins, Git and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Slip-22

Q1) create a simple Java project using Maven. adding dependencies, and Configuring the project's POM file and compile code using maven tool.

1. Ensure you have Maven installed. Verify using:

mvn –version

1. Open your terminal and execute the following:

mvn archetype:generate -DgroupId=com.example -DartifactId=SimpleMavenProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

1. Navigate to the project folder:

cd SimpleMavenProject

1. Open pom.xml in your editor and configure dependencies.

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>SimpleMavenProject</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit Dependency -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

1. Open src/main/java/com/example/App.java and modify it:

package com.example;

public class App {

public static void main(String[] args) {

System.out.println("Hello, Maven Project!");

}

}

1. Compile the Code

mvn compile

1. Package the project into a JAR file:

mvn package

1. To execute the generated JAR file:

java -cp target/SimpleMavenProject-1.0-SNAPSHOT.jar com.example.App

1. Verify Dependencies

mvn dependency:tree

Q2) Install Docker on your system and create a simple "Hello, World!" application using HTML.Create a Dockerfile to containerize the application, using an official web server image as the base. Build the Docker image, tag it, and run a container, making the application accessible on a local port (e.g., http://localhost:8080).

1) Verify Docker installation using:

docker –version

2) Create a project folder

mkdir nginx-html-docker

cd nginx-html-docker

3) Inside the folder, create an HTML file

echo "<!DOCTYPE html>

<html>

<head><title>My Dockerized Web Page</title></head>

<body><h1>Hello, Docker!</h1></body>

</html>" > index.html

4) Create a file named Dockerfile (no extension) in the same folder:

# Use the official nginx image as the base

FROM nginx:latest

# Copy the HTML file to the default nginx directory

COPY index.html /usr/share/nginx/html/

# Expose port 80

EXPOSE 80

5) Build the Docker image and tag it (e.g., my-nginx-web):

docker build -t my-nginx-web .

6) Verify the image is built:

docker images

Stop and remove the container after testing.

7) Start a container using the built image:

docker run -d -p 8080:80 --name my-nginx-container my-nginx-web

8) Verify the container is running:

docker ps

9) Open a browser and navigate to

http://localhost:8080

10) Stop and remove the running container

docker stop my-nginx-container

docker rm my-nginx-container

Slip-23

Q1) Applying CI/CD Principles to Web Development Using Jenkins, Git and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Q.2) create a simple project And push on remote server (like github ) using git. and perform some operation. And displays a chronological history of commits.

1) Create a new folder for your project:

mkdir simple-git-project

cd simple-git-project

2) Add a file to the project:

echo "Hello, Git and GitHub!" > hello.txt

3) Initialize a Git repository in the project directory:

git init

4) Add the file to the staging area:

git add hello.txt

5) Commit the changes with a descriptive message:

git commit -m "Initial commit: Added hello.txt with a welcome message"

6) Create a new repository on GitHub or any remote Git server (e.g., simple-git-project).

Add the remote repository:

git remote add origin https://github.com/your-username/simple-git-project.git

7) Push the changes to the remote repository:

git branch -M main

git push -u origin main

8) Modify the File:

echo "This is an update to demonstrate commit history." >> hello.txt

9) Stage and Commit the Changes:

git add hello.txt

git commit -m "Updated hello.txt with additional content"

10) Push the Changes:

git push

11) **View Commit History:** Display a chronological history of commits:

git log –oneline

Slip-24

Q.1) Create CI using Webhook and deploy a project using Jenkins Execute shell.

1) Create a project folder

mkdir my-project

cd my-project

2)Create a simple file in the project:

echo "echo Hello, Jenkins!" > deploy.sh

chmod +x deploy.sh

3)Initialize a Git repository and commit the project:

git init

git add .

git commit -m "Initial commit with deploy script"

4) Push your project to a remote repository (e.g., GitHub, GitLab):

Create a repository on GitHub or GitLab.

Push your project to the remote repository:

git remote add origin <your-repo-url>

git push -u origin master

5) Access Jenkins Dashboard:

Open your Jenkins instance in the browser (http://localhost:8080).

Create a New Job:

In the Jenkins dashboard, click on New Item → Enter a project name (e.g., MyProject-CI) → Select Freestyle project → Click OK.

Configure the Job:

Under the Source Code Management section, select Git.

Enter the Git repository URL (e.g., https://github.com/your-username/my-project.git).

Optionally, provide credentials if required for private repositories.

Set the branch to master (or the default branch of your repository).

Add Build Step:

Under Build, click Add build step → Select Execute shell.

In the Command box, add the shell command to deploy the project (e.g., running a deploy script):

./deploy.sh

Save the Job:

Click Save to save your job configuration.

6) Configure Webhook in Your Git Repository:

If using GitHub:

Go to your GitHub repository → Settings → Webhooks → Add webhook.

Set the Payload URL to your Jenkins server's URL:

perl

http://<your-jenkins-server>/github-webhook/

Set Content Type to application/json.

Choose Just the push event to trigger the build when you push changes.

Click Add webhook.

7) If using GitLab:

Go to your GitLab repository → Settings → Webhooks.

Set the URL to your Jenkins webhook URL (e.g., http://<your-jenkins-server>/gitlab-webhook/).

Choose the push event to trigger the build.

Click Add webhook.

Verify Webhook:

Make sure that your repository is pushing events to Jenkins. You can check this in Jenkins under Manage Jenkins → System Log.

8) Make a Change and Push:

Make a change in your project (e.g., modify deploy.sh).

Commit and push the changes:

git add .

git commit -m "Updated deploy script"

git push

Q2) Applying CI/CD Principles to Web Development Using Jenkins, Git, and Local HTTP Server (e.g., Apache or Nginx).

1) **Install Git**:

sudo apt update

sudo apt install git –y

2) Verify:

git –version

3) **Install Jenkins**:

http://localhost:8080

4) For Apache:

sudo apt install apache2 –y

5) For Nginx

sudo apt install nginx –y

6) **Create Project Files**:

mkdir web-ci-cd

cd web-ci-cd

echo "<!DOCTYPE html>

<html>

<head><title>CI/CD Demo</title></head>

<body><h1>Welcome to CI/CD with Jenkins, Git, and Nginx!</h1></body>

</html>" > index.html

7) **Initialize Git Repository**:

git init

git add index.html

git commit -m "Initial commit: Added index.html"

8) **Push to Remote Repository**:

git remote add origin https://github.com/your-username/web-ci-cd.git

git branch -M main

git push -u origin main

9) **Create a New Job**:

* Go to Jenkins Dashboard → New Item → Select "Pipeline" → Name it Web-CI-CD.

10) **Configure the Pipeline**:

* Under "Pipeline Definition," select "Pipeline script from SCM."
* Choose Git as the SCM and enter the repository URL

https://github.com/your-username/web-ci-cd.git

11) **Add a Pipeline Script (Jenkinsfile)**: Create a Jenkinsfile in the project directory:

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/your-username/web-ci-cd.git'

}

}

stage('Build') {

steps {

echo 'Building the project...'

}

}

stage('Deploy') {

steps {

script {

sh 'sudo cp index.html /var/www/html/'

}

}

}

}

}

12) Commit and push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Added Jenkinsfile for CI/CD"

git push

13) Go to the Jenkins Dashboard and select your pipeline job.

Click "Build Now".

Monitor the pipeline stages:

Checkout: Clones the repository.

Build: Simulates a build step.

Deploy: Copies the index.html to the HTTP server's root directory.

14) Open your browser and navigate to:

http://localhost/

Slip-25

Q.1 ) Create a simple project, push it to a remote repository on GitLab, and create a new branch. Merge this branch into the main branch and display a chronological history of commits

1) Create a Project Directory: Open a terminal and create a new directory for your project:

mkdir gitlab-demo

cd gitlab-demo

2) Add a File to the Project: Create a simple text file (hello.txt) in the project directory:

echo "Hello, GitLab!" > hello.txt

3) Initialize a Git Repository: Initialize a Git repository in the project directory:

git init

4) Stage and Commit the File: Stage the file and make an initial commit:

git add hello.txt

git commit -m "Initial commit: Added hello.txt"

5) Create a Repository on GitLab:

Go to GitLab.

Create a new repository (e.g., gitlab-demo).

6) Add the Remote GitLab Repository: Add your GitLab repository as a remote origin:

git remote add origin https://gitlab.com/<your-username>/gitlab-demo.git

7)Push the Main Branch to GitLab:

git branch -M main

8)Push the main branch to GitLab:

git push -u origin main

9) Create and Switch to a New Branch: Create a new branch (e.g., feature-branch) and switch to it:

git checkout -b feature-branch

10) Make Changes in the New Branch: Edit the hello.txt file:

echo "This is a new feature." >> hello.txt

11) Stage and Commit the Changes: Stage and commit the changes to the new branch:

git add hello.txt

git commit -m "Added a new feature in hello.txt"

12) Push the Feature Branch to GitLab: Push the new branch to GitLab:

git push -u origin feature-branch

13) Switch Back to the Main Branch: Checkout the main branch:

git checkout main

14)Merge the Feature Branch into the Main Branch: Merge the changes from feature-branch into main:

git merge feature-branch

15)Push the Updated Main Branch to GitLab: Push the merged changes to GitLab:

git push

16) View the Commit History

git log –oneline

17) View the Commit History in a Graphical Format

git log --oneline –graph

18) Pull Changes on Your Local Machine

git checkout main

19) Pull the Latest Changes

git pull origin main

Q.1) Create CI using Webhook and deploy a project using Jenkins Execute shell.

1) Create a project folder

mkdir my-project

cd my-project

2)Create a simple file in the project:

echo "echo Hello, Jenkins!" > deploy.sh

chmod +x deploy.sh

3)Initialize a Git repository and commit the project:

git init

git add .

git commit -m "Initial commit with deploy script"

4) Push your project to a remote repository (e.g., GitHub, GitLab):

Create a repository on GitHub or GitLab.

Push your project to the remote repository:

git remote add origin <your-repo-url>

git push -u origin master

5) Access Jenkins Dashboard:

Open your Jenkins instance in the browser (http://localhost:8080).

Create a New Job:

In the Jenkins dashboard, click on New Item → Enter a project name (e.g., MyProject-CI) → Select Freestyle project → Click OK.

Configure the Job:

Under the Source Code Management section, select Git.

Enter the Git repository URL (e.g., https://github.com/your-username/my-project.git).

Optionally, provide credentials if required for private repositories.

Set the branch to master (or the default branch of your repository).

Add Build Step:

Under Build, click Add build step → Select Execute shell.

In the Command box, add the shell command to deploy the project (e.g., running a deploy script):

./deploy.sh

Save the Job:

Click Save to save your job configuration.

6) Configure Webhook in Your Git Repository:

If using GitHub:

Go to your GitHub repository → Settings → Webhooks → Add webhook.

Set the Payload URL to your Jenkins server's URL:

perl

http://<your-jenkins-server>/github-webhook/

Set Content Type to application/json.

Choose Just the push event to trigger the build when you push changes.

Click Add webhook.

7) If using GitLab:

Go to your GitLab repository → Settings → Webhooks.

Set the URL to your Jenkins webhook URL (e.g., http://<your-jenkins-server>/gitlab-webhook/).

Choose the push event to trigger the build.

Click Add webhook.

Verify Webhook:

Make sure that your repository is pushing events to Jenkins. You can check this in Jenkins under Manage Jenkins → System Log.

8) Make a Change and Push:

Make a change in your project (e.g., modify deploy.sh).

Commit and push the changes:

git add .

git commit -m "Updated deploy script"

git push