

Slip 1)

Q.1) 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. Open Git Bash (or your terminal)

If you're on Windows, search and open "Git Bash."

On Mac/Linux, open your regular Terminal.

2. Navigate to your work area

Type:

bash

cd Desktop

(Or any folder where you want to make your project.)

3. Make a new project folder

Type:

bash

mkdir myproject

Then:

bash

cd myproject

4. Create your Python file

Type:

bash

```
echo "print('Hello DevOps') > main.py"
```

This creates a file called main.py.

5. Initialize Git

Type:

bash

```
git init
```

This sets up git in your project folder.

6. Check what files are ready (optional)

Type:

bash

```
git status
```

You'll see your new main.py as untracked.

7. Add your file to Git's staging area

Type:

bash

```
git add main.py
```

8. Commit your changes

Type:

bash

```
git commit -m "Initial commit: Add hello world Python script"
```

9. See your commit history (optional)

Type:

bash

git log

Press q to exit the log view when done.

10. Run your Python file to check

Type:

bash

python main.py

You should see:

Hello DevOps

2) Q.2) create a simple Java project using Maven. adding dependencies, and Configuring the project's POM file and compile code using maven tool.

Q.1) Docker and Containerization 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. 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.

Docker and Containerization - Complete Step-by-Step Guide

I'll walk you through creating a Docker container for a simple HTML web page using Nginx.

❖ Software Requirements

Required Software:

1. **Docker Desktop** - Download from [docker.com](https://www.docker.com)
 2. **Text Editor** - VS Code, Notepad++, or any text editor
 3. **Web Browser** - Chrome, Firefox, etc.
 4. **Terminal/Command Prompt**
-

Task 1: Dockerfile Creation and Build

Step 1: Install and Verify Docker

First, ensure Docker is installed and running:

Open Command Prompt/Terminal:

cmd

docker --version

Expected Output:

text

Docker version 24.0.6, build ed223bc

If Docker is not installed:

1. Download Docker Desktop from [docker.com](https://www.docker.com)
2. Install and run Docker Desktop
3. Wait for "Docker is running" notification

Step 2: Create Project Directory

Create a new folder for your project:

cmd

```
# Create and navigate to project directory  
mkdir docker-webpage  
cd docker-webpage
```

 **Expected Output:** You're now in the docker-webpage directory

Step 3: Create HTML File

Create index.html file:

Using Command Prompt (Windows):

cmd

```
notepad index.html
```

Or create manually with this content:

```
html  
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>My Dockerized Web Page</title>  
<style>  
  body {  
    font-family: Arial, sans-serif;  
    max-width: 800px;  
    margin: 0 auto;
```

```
padding: 20px;  
background-color: #f0f8ff;  
}  
  
.container {  
background: white;  
padding: 30px;  
border-radius: 10px;  
box-shadow: 0 0 10px rgba(0,0,0,0.1);  
}  
  
h1 {  
color: #2c3e50;  
}  
  
.success {  
color: #27ae60;  
font-weight: bold;  
}  
  
</style>  
</head>  
  
<body>  
  
<div class="container">  
  
<h1>🚀 Welcome to My Dockerized Web Page!</h1>  
  
<p>This HTML page is running inside a <strong>Docker container</strong> with Nginx.</p>  
  
  
<h2>📝 What we've accomplished:</h2>  
  
<ul>  
  
<li>✓ Created a Dockerfile</li>  
  
<li>✓ Built a Docker image</li>
```

```
<li>✓ Containerized Nginx web server</li>
<li>✓ Mapped ports 8080 → 80</li>
</ul>
```

```
<p class="success">🎉 Congratulations! Your Docker container is running successfully!</p>
```

```
<h2>🔧 Technical Details:</h2>
<ul>
  <li><strong>Base Image:</strong> nginx:alpine</li>
  <li><strong>Host Port:</strong> 8080</li>
  <li><strong>Container Port:</strong> 80</li>
  <li><strong>Web Server:</strong> Nginx</li>
</ul>
</div>
</body>
</html>
```

✓ **Expected:** index.html file created in your project directory

Step 4: Create Dockerfile

Create Dockerfile (no extension):

Using Command Prompt:

cmd

notepad Dockerfile

Content of Dockerfile:

dockerfile

```
# Use official nginx alpine image as base
```

```
FROM nginx:alpine
```

```
# Copy our custom HTML file to nginx default directory  
COPY index.html /usr/share/nginx/html/index.html
```

```
# Expose port 80 (default nginx port)  
EXPOSE 80
```

```
# Command to run nginx (already included in base image)  
CMD ["nginx", "-g", "daemon off;"]
```

 **Expected:** Dockerfile created in your project directory

Step 5: Verify Project Structure

Check your project directory:

cmd

dir

 **Expected Output:**

text

Volume in drive C has no label.

Volume Serial Number is XXXX-XXXX

Directory of C:\docker-webpage

```
11/24/2023 02:30 PM <DIR> .  
11/24/2023 02:30 PM <DIR> ..  
11/24/2023 02:25 PM 834 Dockerfile  
11/24/2023 02:24 PM 1,234 index.html  
2 File(s) 2,068 bytes
```

Step 6: Build Docker Image

Build the Docker image with a tag:

cmd

```
docker build -t my-nginx-website:v1.0 .
```

 **Expected Output:**

text

```
[+] Building 15.2s (6/6) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 131B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [1/2] FROM docker.io/library/nginx:alpine@sha256:...
=> [2/2] COPY index.html /usr/share/nginx/html/index.html
=> exporting to image
=> => exporting layers
=> => writing image sha256:a1b2c3d4e5f6...
=> => naming to docker.io/library/my-nginx-website:v1.0
```

Step 7: Verify Docker Image

Check if image was created successfully:

cmd

```
docker images
```

 **Expected Output:**

text

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-nginx-website	v1.0	a1b2c3d4e5f6	2 minutes ago	23.5MB
nginx	alpine	x1y2z3a4b5c6	2 weeks ago	23.5MB

Task 2: Running and Managing Containers

Step 8: Run the Docker Container

Run container with port mapping (host:container):

cmd

```
docker run -d -p 8080:80 --name my-web-container my-nginx-website:v1.0
```

 **Expected Output:**

text

```
a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4e5f6
```

Step 9: Verify Container is Running

Check running containers:

cmd

```
docker ps
```

 **Expected Output:**

text

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
a1b2c3d4e5f6	my-nginx-website:v1.0	"nginx -g 'daemon off;'	30 seconds ago	Up 30 seconds	0.0.0.0:8080->80/tcp my-web-container

Step 10: Test the Web Page

Method 1: Using Web Browser

1. Open your web browser
2. Go to: <http://localhost:8080>

 **Expected:** You should see your HTML page with the success message

Method 2: Using curl (if installed)

cmd

```
curl http://localhost:8080
```

 **Expected Output:** HTML content of your page

Step 11: Check Container Logs

View container logs to ensure it's working:

cmd

```
docker logs my-web-container
```

 **Expected Output:**

text

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration

/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/

/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh

...

Step 12: Stop the Container

Stop the running container:

cmd

docker stop my-web-container

 **Expected Output:**

text

my-web-container

Verify it's stopped:

cmd

docker ps

 **Expected Output:** (No containers listed, or your container not in list)

Step 13: Remove the Container

Remove the stopped container:

cmd

docker rm my-web-container

 **Expected Output:**

text

my-web-container

Verify it's removed:

cmd

docker ps -a

 **Expected Output:** Your container should not appear in the list

 **Cleanup (Optional)**

Remove Docker Image (if needed)

cmd

```
docker rmi my-nginx-website:v1.0
```

Remove All Stopped Containers

cmd

```
docker container prune
```

 **Complete Workflow Recap**

Here's the complete sequence of commands:

bash

1. Create project directory

```
mkdir docker-webpage
```

```
cd docker-webpage
```

2. Create HTML file (copy content from above)

```
notepad index.html
```

3. Create Dockerfile (copy content from above)

```
notepad Dockerfile
```

4. Build Docker image

```
docker build -t my-nginx-website:v1.0 .
```

5. Run container

```
docker run -d -p 8080:80 --name my-web-container my-nginx-website:v1.0
```

```
# 6. Test in browser: http://localhost:8080
```

```
# 7. Stop and remove container
```

```
docker stop my-web-container
```

```
docker rm my-web-container
```

Troubleshooting Common Issues

Issue 1: "Port already in use"

Solution:

```
cmd
```

```
# Use a different port
```

```
docker run -d -p 8081:80 --name my-web-container my-nginx-website:v1.0
```

Issue 2: "Container name already in use"

Solution:

```
cmd
```

```
# Remove existing container first
```

```
docker rm my-web-container
```

```
# Or use a different name
```

```
docker run -d -p 8080:80 --name my-web-container2 my-nginx-website:v1.0
```

Issue 3: "Docker daemon not running"

Solution:

- Start Docker Desktop
- Wait for Docker to fully start
- Try commands again

Issue 4: "Build context error"

Solution:

- Ensure both Dockerfile and index.html are in the same directory
 - Run docker build command from that directory
-

Success Checklist

- Docker installed and running
- Project directory created
- HTML file created
- Dockerfile created
- Image built successfully
- Container running on port 8080
- Web page accessible in browser
- Container stopped and removed
- All tasks completed successfully

 **You've successfully completed both Docker tasks!** Your web page is now containerized and you've practiced the complete Docker workflow.

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.

Git and GitHub - Complete Step-by-Step Guide

I'll walk you through creating a project, using Git, and pushing to GitHub with all operations and commit history.

Software Requirements

Required Software:

1. **Git** - Download from git-scm.com
 2. **GitHub Account** - Create at github.com
 3. **Text Editor** - VS Code, Notepad++, etc.
 4. **Terminal/Command Prompt**
-

Phase 1: Setup and Installation

Step 1: Install and Verify Git

Check if Git is installed:

cmd

```
git --version
```

Expected Output (if installed):

text

```
git version 2.39.2.windows.1
```

If not installed:

1. Download from git-scm.com
2. Run the installer with default options
3. Restart command prompt and verify again

Step 2: Configure Git Global Settings

Set your username and email (replace with your details):

cmd

```
git config --global user.name "Your Name"
```

```
git config --global user.email "your.email@example.com"
```

Verify configuration:

cmd

```
git config --list
```

Expected Output:

text

```
user.name=Your Name
```

```
user.email=your.email@example.com
```

...

Phase 2: Create Local Project and Git Repository

Step 3: Create Project Directory

```
cmd  
# Create and navigate to project directory  
mkdir my-git-project  
cd my-git-project
```

 **Expected Output:** You're now in the my-git-project directory

Step 4: Initialize Git Repository

```
cmd
```

```
git init
```

 **Expected Output:**

```
text
```

```
Initialized empty Git repository in C:/my-git-project/.git/
```

Step 5: Create Project Files

Create index.html:

```
cmd
```

```
notepad index.html
```

Content:

```
html
```

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
  
    <meta charset="UTF-8">  
  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  
    <title>My Git Project</title>  
  
<style>  
  
    body { font-family: Arial, sans-serif; margin: 40px; }  
  
    .container { max-width: 800px; margin: 0 auto; }  
  
    .commit { background: #f5f5f5; padding: 10px; margin: 10px 0; }  
  
</style>
```

```
</head>

<body>

<div class="container">

    <h1>🚀 My Git Version Controlled Project</h1>

    <p>This project demonstrates Git operations and commit history.</p>

    <h2>📅 Project Timeline:</h2>

    <div class="commit">

        <strong>Commit 1:</strong> Initial project setup with basic HTML structure

    </div>

</div>

</body>

</html>
```

Create README.md:

cmd

notepad README.md

Content:

markdown

My Git Project

This is a simple project to demonstrate Git and GitHub operations.

Project Description

- HTML web page
- Git version control
- Multiple commits to show history

Features

- Responsive design
- Clean code structure
- Version controlled development

Create styles.css:

cmd

notepad styles.css

Content:

css

/ Initial CSS file */*

```
body {  
    font-family: Arial, sans-serif;  
    line-height: 1.6;  
    color: #333;  
}
```

```
.container {  
    max-width: 800px;  
    margin: 0 auto;  
    padding: 20px;  
}
```

Step 6: Check Project Status

cmd

git status

 **Expected Output:**

text

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

README.md

index.html

styles.css

nothing added to commit but untracked files present (use "git add" to track)

Phase 3: Git Operations and Commits

Step 7: First Commit - Initial Setup

Stage all files:

cmd

git add .

Check status again:

cmd

git status

Expected Output:

text

On branch main

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: README.md

new file: index.html

new file: styles.css

Make first commit:

cmd

```
git commit -m "Initial commit: Project setup with basic files"
```

Expected Output:

text

```
[main (root-commit) a1b2c3d] Initial commit: Project setup with basic files
```

```
 3 files changed, 45 insertions(+)
```

```
    create mode 100644 README.md
```

```
    create mode 100644 index.html
```

```
    create mode 100644 styles.css
```

Step 8: Second Commit - Add Features

Update index.html - add new section:

html

```
<!-- Add this after the existing div with class "commit" -->
```

```
<div class="commit">
```

```
  <strong>Commit 2:</strong> Added features section and improved styling
```

```
</div>
```

```
<h2>✨ Features Added:</h2>
```

```
<ul>
```

```
  <li>Responsive navigation</li>
```

```
  <li>Enhanced styling</li>
```

```
  <li>Better structure</li>
```

```
</ul>
```

Update styles.css - add more styles:

css

```
/* Add these styles */
```

```
.header {
```

```
background: #2c3e50;  
color: white;  
padding: 1rem;  
border-radius: 5px;  
}
```

```
.feature-list {  
background: #ecf0f1;  
padding: 15px;  
border-radius: 5px;  
}
```

Check what changed:

cmd

git diff

Expected Output: Shows the specific lines you added/modified

Stage and commit changes:

cmd

git add .

git commit -m "feat: Add features section and enhanced styling"

Expected Output:

text

[main e4f5g6h] feat: Add features section and enhanced styling

2 files changed, 18 insertions(+)

Step 9: Third Commit - Bug Fix

Create a bug by adding error to index.html:

html

<!-- Add this with intentional error -->

```
<div class="bug">
```

```
<p>This section has a styling bug</p>  
</div>
```

Commit the bug:

```
cmd  
git add .  
git commit -m "feat: Add new content section"
```

Now fix the bug in index.html:

```
html  
<!-- Change to -->  
<div class="feature-box">  
  <p>This section now has proper styling</p>  
</div>
```

Add corresponding CSS:

```
css  
.feature-box {  
  background: #d5edfa;  
  padding: 15px;  
  border: 1px solid #3498db;  
  border-radius: 5px;  
}
```

Commit the fix:

```
cmd  
git add .  
git commit -m "fix: Correct styling bug in new section"
```

Step 10: Fourth Commit - Update Documentation

Update README.md:

```
markdown  
# My Git Project
```

This is a simple project to demonstrate Git and GitHub operations.

Project Description

- HTML web page
- Git version control
- Multiple commits to show history

Features

- Responsive design
- Clean code structure
- Version controlled development
- Enhanced styling
- Bug fixes applied

Git Operations Demonstrated

1. Initial commit
2. Feature additions
3. Bug fixes
4. Documentation updates

How to Run

Open `index.html` in any web browser.

Commit documentation update:

cmd

git add .

git commit -m "docs: Update README with project details and usage"

Phase 4: View Commit History

Step 11: Display Chronological Commit History

View detailed commit history:

cmd

git log

Expected Output:

text

commit 8901j2k3l4m5n6o7p8q9r0 (HEAD -> main)

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 14:45:22 2023 +0530

docs: Update README with project details and usage

commit 7890i1j2k3l4m5n6o7p8q9

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 14:40:15 2023 +0530

fix: Correct styling bug in new section

commit 6789h0i1j2k3l4m5n6o7p8

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 14:35:10 2023 +0530

feat: Add new content section

commit 5678g9h0i1j2k3l4m5n6o7

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 14:30:05 2023 +0530

feat: Add features section and enhanced styling

commit a1b2c3d4e5f6g7h8i9j0k1l2

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 14:25:00 2023 +0530

Initial commit: Project setup with basic files

View compact commit history:

cmd

git log --oneline

 **Expected Output:**

text

8901j2k (HEAD -> main) docs: Update README with project details and usage

7890i1j fix: Correct styling bug in new section

6789h0i feat: Add new content section

5678g9h feat: Add features section and enhanced styling

a1b2c3d Initial commit: Project setup with basic files

View commit history with graph:

cmd

git log --graph --oneline --all

View specific number of commits:

cmd

git log -3

 **Expected Output:** Shows last 3 commits

 **Phase 5: Push to GitHub Remote Repository**

Step 12: Create GitHub Repository

1. Go to github.com and log in
2. Click "+" icon in top right → "New repository"
3. **Repository name:** my-git-project
4. **Description:** "A demo project for Git operations"
5. **Visibility:** Public
6. **Don't initialize with README** (we already have one)
7. Click "Create repository"

Step 13: Connect Local Repository to GitHub

Copy the commands from GitHub (it shows after repository creation) or use:

cmd

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

Verify remote connection:

cmd

```
git remote -v
```

 **Expected Output:**

text

```
origin https://github.com/your-username/my-git-project.git (fetch)
```

```
origin https://github.com/your-username/my-git-project.git (push)
```

Step 14: Push to GitHub

First push (set upstream):

cmd

```
git push -u origin main
```

 **Expected Output:**

text

```
Enumerating objects: 15, done.
```

```
Counting objects: 100% (15/15), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (12/12), done.
```

```
Writing objects: 100% (15/15), 1.45 KiB | 1.45 MiB/s, done.
```

```
Total 15 (delta 2), reused 0 (delta 0), pack-reused 0
```

```
remote: Resolving deltas: 100% (2/2), done.
```

```
To https://github.com/your-username/my-git-project.git
```

```
* [new branch]  main -> main
```

```
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Step 15: Verify on GitHub

1. **Refresh your GitHub repository page**
 2. **You should see all your files**
 3. **Click on "commits" to see the commit history**
 4. **You can click on each commit to see what changed**
-

Phase 6: Additional Git Operations

Step 16: Make More Changes and Push

Create script.js:

cmd

notepad script.js

Content:

javascript

```
// JavaScript functionality
```

```
console.log("Git Project Loaded!");
```

```
function showMessage() {  
    alert("Hello from Git version controlled project!");  
}
```

Update index.html **to include script:**

html

```
<!-- Add before closing body tag -->
```

```
<script src="script.js"></script>
```

Stage, commit, and push:

cmd

```
git add .
```

```
git commit -m "feat: Add JavaScript functionality"
```

```
git push
```

 **Expected Output:** Files pushed to GitHub

Step 17: View Difference Between Commits

See what changed in last commit:

cmd

```
git show HEAD
```

Compare with previous commit:

cmd

```
git diff HEAD~1 HEAD
```

Step 18: Check Repository Status

cmd

```
git status
```

 **Expected Output:**

text

On branch main

Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

Step 19: View Branch Information

cmd

```
git branch -a
```

 **Expected Output:**

text

```
* main  
remotes/origin/main
```

Complete Workflow Recap

All Commands Used:

```
bash
```

```
# Setup
```

```
git config --global user.name "Your Name"
```

```
git config --global user.email "your.email@example.com"
```

```
# Local repository
```

```
mkdir my-git-project
```

```
cd my-git-project
```

```
git init
```

```
# Create files (index.html, README.md, styles.css)
```

```
git add .
```

```
git commit -m "Initial commit"
```

```
# Multiple development commits
```

```
git add .
```

```
git commit -m "feat: Add new feature"
```

```
git add .
```

```
git commit -m "fix: Correct issue"
```

```
# View history
```

```
git log
```

```
git log --oneline
```

Connect to GitHub

```
git remote add origin https://github.com/your-username/my-git-project.git  
git push -u origin main
```

Ongoing work

```
git add .  
git commit -m "description"  
git push
```

Git Operations Summary

Operation	Command	Purpose
Initialize	git init	Create new repository
Stage Files	git add .	Add changes to staging
Commit	git commit -m "message"	Save changes with message
Check Status	git status	See current state
View History	git log	See commit timeline
Connect Remote	git remote add origin URL	Link to GitHub
Push	git push	Send to remote repository
See Changes	git diff	View modifications

Success Checklist

- Git installed and configured
- Local project directory created
- Git repository initialized
- Multiple commits created with different messages
- Commit history viewed in different formats
- GitHub repository created
- Local repository connected to GitHub
- Code pushed to remote repository
- All operations completed successfully

 **Congratulations!** You've successfully learned Git operations, commit history management, and pushing to GitHub!

Slip 3

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

Phase 1: Jenkins Installation and Setup

Step 1: Install Jenkins

On Ubuntu/Linux:

bash

Update system

```
sudo apt update
```

Install Java (Jenkins requirement)

```
sudo apt install openjdk-11-jdk -y
```

Add Jenkins repository

```
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-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
```

Install Jenkins

```
sudo apt update
```

```
sudo apt install jenkins -y
```

Start Jenkins

```
sudo systemctl start jenkins
```

```
sudo systemctl enable jenkins
```

```
# Check status  
sudo systemctl status jenkins
```

 **Expected Output:**

text

● jenkins.service - Jenkins Continuous Integration Server

```
Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
```

```
Active: active (running) since Mon 2023-11-24 14:30:00 IST; 10s ago
```

On Windows:

1. Download Jenkins from jenkins.io
2. Run the installer as Administrator
3. Follow installation wizard
4. Jenkins will start automatically on port 8080

Step 2: Access Jenkins Web Interface

Open browser and go to:

text

http://localhost:8080

Get initial admin password:

bash

On Linux

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

On Windows

```
C:\Program Files\Jenkins\secrets\initialAdminPassword
```

 **Expected:** Jenkins setup wizard appears

Step 3: Complete Jenkins Setup

1. **Install suggested plugins**
2. **Create admin user** (remember credentials)

3. **Instance configuration** - keep default URL

4. **Jenkins is ready** message appears

Phase 2: Project Setup and Git Repository

Step 4: Create Web Project

Create project directory:

```
bash
```

```
mkdir jenkins-web-project
```

```
cd jenkins-web-project
```

Create index.html:

```
html
```

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

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

```
    <title>Jenkins CI/CD Demo</title>
```

```
    <style>
```

```
        body {
```

```
            font-family: Arial, sans-serif;
```

```
            max-width: 800px;
```

```
            margin: 0 auto;
```

```
            padding: 20px;
```

```
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
```

```
            color: white;
```

```
        }
```

```
        .container {
```

```
            background: rgba(255, 255, 255, 0.1);
```

```
padding: 30px;  
border-radius: 15px;  
backdrop-filter: blur(10px);  
}  
  
.build-info {  
background: rgba(0, 0, 0, 0.3);  
padding: 15px;  
border-radius: 8px;  
margin: 10px 0;  
}  
  
</style>  
  
</head>  
  
<body>  
  
<div class="container">  
  
<h1>🚀 Jenkins CI/CD Pipeline Demo</h1>  
  
<p>This application is automatically deployed using Jenkins CI/CD</p>  
  
  
<div class="build-info">  
  
<h3>📊 Build Information:</h3>  
  
<p><strong>Build Date:</strong> <span id="buildDate">Loading...</span></p>  
  
<p><strong>Build Number:</strong> <span  
id="buildNumber">Loading...</span></p>  
  
<p><strong>Git Commit:</strong> <span id="gitCommit">Loading...</span></p>  
  
</div>  
  
  
<h2>🔧 CI/CD Steps Completed:</h2>  
  
<ul>  
  
<li>✅ Code committed to Git</li>
```

```
<li>  Jenkins detects changes (Webhook)</li>
<li>  Automated build triggered</li>
<li>  Tests executed</li>
<li>  Deployment to Nginx server</li>
<li>  Website updated automatically</li>
</ul>
</div>
```

```
<script>
    // Display build information
    document.getElementById('buildDate').textContent = new Date().toLocaleString();
    document.getElementById('buildNumber').textContent = 'BUILD_' + Date.now();
    document.getElementById('gitCommit').textContent = 'GIT_' +
    Math.random().toString(36).substr(2, 8);
</script>
</body>
</html>
```

Create deploy.sh deployment script:

```
bash
#!/bin/bash
echo "🚀 Starting deployment process..."

# Variables
DEPLOY_DIR="/var/www/html/jenkins-demo"
BACKUP_DIR="/var/www/html/backup"
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
```

```
echo "📁 Deployment directory: $DEPLOY_DIR"
```

```
# Create backup
echo "📦 Creating backup..."
sudo mkdir -p $BACKUP_DIR
sudo cp -r $DEPLOY_DIR $BACKUP_DIR/backup_${TIMESTAMP} 2>/dev/null || true

# Create deployment directory if not exists
sudo mkdir -p $DEPLOY_DIR

# Copy new files
echo "📄 Copying new files..."
sudo cp -r ./* $DEPLOY_DIR/

# Set permissions
echo "🔒 Setting permissions..."
sudo chown -R www-data:www-data $DEPLOY_DIR
sudo chmod -R 755 $DEPLOY_DIR

echo "✅ Deployment completed successfully!"
echo "🌐 Website available at: http://localhost/jenkins-demo"
```

Make script executable:

```
bash
chmod +x deploy.sh
```

Step 5: Initialize Git Repository

```
bash
# Initialize git
git init
```

```
# Add files  
git add .  
  
# First commit  
git commit -m "Initial commit: Jenkins CI/CD web project"
```

Check status

```
git status
```

 **Expected Output:**

text

On branch main

nothing to commit, working tree clean

Step 6: Create GitHub Repository

1. **Go to [GitHub.com](https://github.com)** → Create new repository: jenkins-web-demo
2. **Don't initialize** with README (we have existing files)
3. **Push local repository:**

bash

```
git remote add origin https://github.com/your-username/jenkins-web-demo.git
```

```
git branch -M main
```

```
git push -u origin main
```

 **Expected Output:**

text

Enumerating objects: 6, done.

Counting objects: 100% (6/6), done.

Writing objects: 100% (6/6), 1.45 KiB | 1.45 MiB/s, done.

Total 6 (delta 0), reused 0 (delta 0), pack-reused 0

To <https://github.com/your-username/jenkins-web-demo.git>

```
* [new branch]  main -> main
```

Branch 'main' set up to track remote branch 'main' from 'origin'.

Phase 3: Nginx Web Server Setup

Step 7: Install and Configure Nginx

Install Nginx:

bash

```
sudo apt install nginx -y
```

Start Nginx:

bash

```
sudo systemctl start nginx
```

```
sudo systemctl enable nginx
```

```
sudo systemctl status nginx
```

Expected Output:

text

- nginx.service - A high performance web server and a reverse proxy server

```
Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
```

```
Active: active (running) since Mon 2023-11-24 14:35:00 IST; 10s ago
```

Test Nginx:

bash

```
curl http://localhost
```

Expected Output:

 HTML content of default Nginx page

Create deployment directory:

bash

```
sudo mkdir -p /var/www/html/jenkins-demo
```

```
sudo chown -R $USER:$USER /var/www/html/jenkins-demo
```

Phase 4: Jenkins Pipeline Configuration

Step 8: Install Required Jenkins Plugins

1. Go to Jenkins → Manage Jenkins → Manage Plugins
2. Available tab → Search and install:
 - Git plugin (usually pre-installed)
 - Pipeline plugin
 - GitHub Integration plugin
 - NGINX Build Helper plugin (optional)

Step 9: Create Jenkins Pipeline

In Jenkins dashboard:

1. Click "New Item"
2. Enter name: web-app-ci-cd
3. Select "Pipeline" → Click OK

Configure Pipeline:

4. General tab:
 - Description: CI/CD Pipeline for Web Application
5. Build Triggers:
 - GitHub hook trigger for GITScm polling
6. Pipeline section:
 - Definition: Pipeline script
 - Copy this pipeline script:

groovy

```
pipeline {
```

```
    agent any
```

```
    environment {
```

```
        DEPLOY_DIR = '/var/www/html/jenkins-demo'
```

```
        GIT_REPO = 'https://github.com/your-username/jenkins-web-demo.git'
```

```
}
```

```
stages {
```

```
    stage('Checkout') {
```

```
        steps {
```

```
            echo '🚀 Starting CI/CD Pipeline...'
```

```
            git branch: 'main', url: 'https://github.com/your-username/jenkins-web-demo.git'
```

```
        }
```

```
    }
```

```
    stage('Build Info') {
```

```
        steps {
```

```
            script {
```

```
                // Get git commit info
```

```
                COMMIT_HASH = sh(returnStdout: true, script: 'git rev-parse --short HEAD').trim()
```

```
                BUILD_DATE = sh(returnStdout: true, script: 'date "+%Y-%m-%d %H:%M:%S"').trim()
```

```
                echo "📊 Build Information:"
```

```
                echo "Commit: ${COMMIT_HASH}"
```

```
                echo "Build Date: ${BUILD_DATE}"
```

```
                echo "Build Number: ${env.BUILD_NUMBER}"
```

```
            }
```

```
        }
```

```
    }
```

```
    stage('Code Quality Check') {
```

```
steps {
    echo '🔍 Performing code quality checks...'
    sh """
        echo "Checking HTML files..."
        find . -name "*.html" -type f | wc -l
        echo "Checking shell scripts..."
        find . -name "*.sh" -type f | wc -l
        ...
    }
}

stage('Test') {
    steps {
        echo '🧪 Running tests...'
        sh """
            echo "Running deployment script test..."
            bash -n deploy.sh
            echo "✅ Syntax check passed"
        """

        echo "Testing web server connectivity..."
        curl -s http://localhost > /dev/null && echo "✅ Nginx is running"
        ...
    }
}

stage('Deploy to Nginx') {
    steps {
```

```
echo '🚀 Deploying to Nginx server..'
sh ""
chmod +x deploy.sh
./deploy.sh
"""

}

}

stage('Verify Deployment') {
    steps {
        echo '🔍 Verifying deployment...'
        sh ""
        echo "Checking deployed files..."
        ls -la /var/www/html/jenkins-demo/
        echo "Testing website..."
        curl -s -o /dev/null -w "HTTP Status: %{http_code}\n" http://localhost/jenkins-
demo/ || true
        """
    }
}

post {
    always {
        echo '✉️ Pipeline completed - sending notifications'
        emailext (

```

```

    subject: "🚀 CI/CD Pipeline Completed: ${env.JOB_NAME}
#${env.BUILD_NUMBER}",
    body: """
Pipeline: ${env.JOB_NAME}

Build: #${env.BUILD_NUMBER}

Status: ${currentBuild.currentResult}

URL: ${env.BUILD_URL}

Commit: ${COMMIT_HASH}

Build Date: ${BUILD_DATE}

""",
    to: "admin@example.com"
)
}

success {
    echo '🎉 Pipeline executed successfully!'
    sh 'echo "SUCCESS: Build ${env.BUILD_NUMBER} deployed successfully" > build-status.txt'
}
failure {
    echo '❌ Pipeline failed!'
    sh 'echo "FAILURE: Build ${env.BUILD_NUMBER} failed" > build-status.txt'
}
}

```

7. Click Save

Step 10: Configure GitHub Webhook

In GitHub repository:

1. Go to **Settings** → **Webhooks** → **Add webhook**
2. **Payload URL:** `http://your-jenkins-server:8080/github-webhook/`
3. **Content type:** `application/json`
4. **Which events:** Just the push event
5. **Active** → **Add webhook**

Expected: Webhook shows green checkmark after testing

Phase 5: Testing the CI/CD Pipeline

Step 11: Trigger First Build

Method 1: Manual trigger

- Go to Jenkins → web-app-ci-cd project
- Click **Build Now**

Method 2: Git push (automatic via webhook)

bash

Make a change to your project

```
echo "<!-- Build: $(date) -->" >> index.html
```

Commit and push

```
git add .
```

```
git commit -m "feat: Trigger CI/CD pipeline with new build"
```

```
git push origin main
```

Step 12: Monitor Pipeline Execution

In Jenkins:

1. **Click on build number** → **Console Output**
2. **Watch real-time execution**

Expected Console Output:

text

Started by GitHub push by your-username

```
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/web-app-ci-cd
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Checkout)
[Pipeline] echo
 Starting CI/CD Pipeline...
[Pipeline] git
Fetching changes from the remote Git repository
...
[Pipeline] stage
[Pipeline] { (Deploy to Nginx)
[Pipeline] echo
 Deploying to Nginx server...
[Pipeline] sh
+ chmod +x deploy.sh
+ ./deploy.sh
 Starting deployment process...
📁 Deployment directory: /var/www/html/jenkins-demo
 Creating backup...
 Copying new files...
 Setting permissions...
 Deployment completed successfully!
 Website available at: http://localhost/jenkins-demo
[Pipeline] }
```

...

[Pipeline] echo

🎉 Pipeline executed successfully!

[Pipeline] End of Pipeline

Finished: SUCCESS

Step 13: Verify Deployment

Test the deployed website:

bash

curl http://localhost/jenkins-demo/

Or open in browser:

text

http://your-server-ip/jenkins-demo/

✓ **Expected:** Your HTML page with CI/CD information

📋 Phase 6: Advanced Pipeline Features

Step 14: Add Parallel Testing Stage

Update pipeline script - add this stage:

groovy

```
stage('Parallel Tests') {  
    parallel {  
        stage('HTML Validation') {  
            steps {  
                echo '📝 Validating HTML...'  
                sh ""  
                echo "Checking HTML structure..."  
                grep -q "<html>" index.html && echo "✓ HTML structure valid"  
                grep -q "<title>" index.html && echo "✓ Title tag found"  
                ...  
            }  
        }  
    }  
}
```

```

        }

    }

stage('Security Scan'){

    steps{
        echo '🔒 Basic security checks...'
        sh """
            echo "Checking file permissions..."
            ls -la *.html *.sh
            echo "✅ Security scan completed"
            ...
        }
    }
}

```

Step 15: Add Artifact Archiving

Add this to post section:

```

groovy

post{

    always{

        archiveArtifacts artifacts: '**/*.html,**/*.css,**/*.js,**/*.sh', fingerprint: true
        junit '**/test-reports/*.xml' // If you have tests
    }
}

```

Complete CI/CD Workflow

Pipeline Visualization:

text

Git Push → GitHub Webhook → Jenkins Trigger → Pipeline Starts

↓

Checkout Code → Code Quality → Parallel Tests → Deploy → Verify

↓

Success/Failure → Notifications → Artifacts Archived

Key Components Working:

1. **Git** - Version control and trigger
 2. **Jenkins** - Orchestration and automation
 3. **GitHub Webhook** - Automatic trigger
 4. **Nginx** - Production deployment
 5. **Pipeline Script** - Definition of CI/CD steps
-



Troubleshooting

Common Issues:

1. Webhook not triggering:

- Check Jenkins URL accessibility from GitHub
- Verify webhook secret (if configured)
- Check Jenkins logs: /var/log/jenkins/jenkins.log

2. Permission denied during deployment:

bash

```
sudo usermod -a -G www-data jenkins
```

```
sudo systemctl restart jenkins
```

3. Nginx not serving files:

bash

```
sudo nginx -t # Test configuration
```

```
sudo systemctl restart nginx
```

4. Pipeline failing:

- Check console output for specific errors
- Verify all required plugins are installed

- Check environment variables and paths
-

Success Checklist

- Jenkins installed and accessible
- GitHub repository created and connected
- Webhook configured and working
- Nginx installed and running
- Pipeline created in Jenkins
- First build triggered successfully
- Code deployed to web server
- Website accessible via browser
- Automated trigger on git push working

 **Congratulations!** You've successfully implemented a complete CI/CD pipeline using Jenkins, Git, and Nginx!

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

Git Branching and Merging - Complete Step-by-Step Guide

I'll walk you through creating a project, pushing to GitHub, branching, merging, and displaying commit history.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
 2. **GitHub Account** - Remote repository hosting
 3. **Text Editor** - VS Code, Notepad++, etc.
 4. **Terminal/Command Prompt**
-

Phase 1: Project Setup and Initial Commit

Step 1: Create Project Directory

Open Terminal/Command Prompt:

bash

Create and navigate to project directory

mkdir git-branching-demo

cd git-branching-demo

 **Expected Output:** You're now in the git-branching-demo directory

Step 2: Initialize Git Repository

bash

git init

 **Expected Output:**

text

Initialized empty Git repository in C:/git-branching-demo/.git/

Step 3: Create Project Files

Create index.html:

bash

On Windows

notepad index.html

On Linux/Mac

touch index.html

Content:

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Git Branching Demo</title>

<style>

body {

font-family: Arial, sans-serif;

max-width: 800px;

margin: 0 auto;

padding: 20px;

background-color: #f5f5f5;

}

.container {

background: white;

padding: 30px;

border-radius: 10px;

box-shadow: 0 0 10px rgba(0,0,0,0.1);

```
}

.commit-history {
    background: #f8f9fa;
    padding: 15px;
    border-left: 4px solid #007bff;
    margin: 10px 0;
}

</style>

</head>

<body>

    <div class="container">

        <h1>  Git Branching & Merging Demo</h1>

        <p>This project demonstrates Git branching, merging, and commit history.</p>

        <div class="commit-history">

            <h3>Initial Commit</h3>

            <p>Project setup with basic HTML structure</p>

        </div>

    </div>

</body>

</html>
```

Create README.md:

```
bash
notepad README.md
```

Content:

```
markdown
# Git Branching Demo Project
```

This project demonstrates Git operations including:

- Branch creation
- Merging branches
- Commit history management

Project Structure

- `index.html` - Main web page
- `README.md` - Project documentation

Step 4: Make Initial Commit

Check status:

bash

git status

Expected Output:

text

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

README.md

index.html

nothing added to commit but untracked files present (use "git add" to track)

Stage files and commit:

bash

git add .

git commit -m "Initial commit: Project setup with basic structure"

Expected Output:

text

[main (root-commit) a1b2c3d] Initial commit: Project setup with basic structure

2 files changed, 45 insertions(+)

create mode 100644 README.md

create mode 100644 index.html

Phase 2: Push to GitHub Remote Repository

Step 5: Create GitHub Repository

1. **Go to github.com and log in**
2. **Click "+" icon → "New repository"**
3. **Repository name:** git-branching-demo
4. **Description:** "Demo project for Git branching and merging"
5. **Visibility:** Public
6. **Don't initialize with README** (we already have one)
7. **Click "Create repository"**

Step 6: Connect Local Repository to GitHub

Copy the commands from GitHub or use:

bash

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

Verify remote connection:

bash

```
git remote -v
```

Expected Output:

text

```
origin https://github.com/your-username/git-branching-demo.git (fetch)
```

```
origin https://github.com/your-username/git-branching-demo.git (push)
```

Step 7: Push to GitHub

First push (set upstream):

bash

```
git push -u origin main
```

Expected Output:

text

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Writing objects: 100% (5/5), 1.45 KiB | 1.45 MiB/s, done.

Total 5 (delta 0), reused 0 (delta 0), pack-reused 0

To <https://github.com/your-username/git-branching-demo.git>

* [new branch] main -> main

Branch 'main' set up to track remote branch 'main' from 'origin'.

Phase 3: Create and Work on New Branch

Step 8: Create New Feature Branch

Create and switch to new branch:

bash

```
git checkout -b feature-new-design
```

Expected Output:

text

Switched to a new branch 'feature-new-design'

Verify current branch:

bash

```
git branch
```

Expected Output:

text

main

* feature-new-design

Step 9: Make Changes in Feature Branch

Update index.html - add new section:

html

```
<!-- Add this before closing body tag -->

<section style="background: #e7f3ff; padding: 20px; margin: 20px 0; border-radius: 8px;">

    <h2>🎨 New Design Features (from feature branch)</h2>

    <ul>

        <li>Enhanced color scheme</li>
        <li>Improved typography</li>
        <li>Better spacing and layout</li>
        <li>Responsive design elements</li>

    </ul>

</section>
```

```
<div class="commit-history">

    <h3>Feature Branch Commit</h3>

    <p>Added new design section with enhanced features</p>

</div>
```

Update README.md:

markdown

```
# Git Branching Demo Project
```

This project demonstrates Git operations including:

- Branch creation
- Merging branches
- Commit history management

```
## Project Structure
```

- `index.html` - Main web page
- `README.md` - Project documentation

```
## New Features (from feature branch)
```

- Enhanced UI design
- Improved user experience
- Better documentation

Create new file styles.css:

bash

notepad styles.css

Content:

css

/ New styles from feature branch */*

```
.feature-section {  
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);  
    color: white;  
    padding: 25px;  
    border-radius: 10px;  
    margin: 20px 0;  
}
```

```
.button {  
    background: #28a745;  
    color: white;  
    padding: 10px 20px;  
    border: none;  
    border-radius: 5px;
```

```
    cursor: pointer;  
}  

```

Update index.html to include CSS:

```
html  
  
<!-- Add in head section -->  
  
<link rel="stylesheet" href="styles.css">
```

Step 10: Commit Changes in Feature Branch

Check status:

```
bash  
  
git status
```

Expected Output:

```
text  
  
On branch feature-new-design  
  
Changes not staged for commit:  
  
(use "git add <file>..." to update what will be committed)  
  
(use "git restore <file>..." to discard changes in working directory)  
  
modified: README.md  
  
modified: index.html
```

Untracked files:

```
(use "git add <file>..." to include in what will be committed)  
  
styles.css
```

no changes added to commit (use "git add" and/or "git commit -a")

Stage and commit changes:

```
bash  
  
git add .  
  
git commit -m "feat: Add new design with enhanced styling and features"
```

 **Expected Output:**

text

```
[feature-new-design e4f5g6h] feat: Add new design with enhanced styling and features
```

```
3 files changed, 45 insertions(+), 2 deletions(-)
```

```
create mode 100644 styles.css
```

 **Phase 4: Merge Branch into Main**

Step 11: Switch Back to Main Branch

bash

```
git checkout main
```

 **Expected Output:**

text

```
Switched to branch 'main'
```

```
Your branch is up to date with 'origin/main'.
```

Verify files in main branch:

bash

```
ls -la
```

 **Expected Output:** Only original files (no styles.css, original index.html)

Step 12: Merge Feature Branch into Main

bash

```
git merge feature-new-design
```

 **Expected Output:**

text

```
Updating a1b2c3d..e4f5g6h
```

```
Fast-forward
```

```
 README.md | 8 +++++++-
```

```
 index.html | 32 ++++++++++++++++++++++++
```

```
styles.css | 12 ++++++++
3 files changed, 51 insertions(+), 1 deletion(-)
create mode 100644 styles.css
```

Step 13: Verify Merge Results

Check files after merge:

```
bash
```

```
ls -la
```

 **Expected Output:** Now shows styles.css and updated files

Check current status:

```
bash
```

```
git status
```

 **Expected Output:**

```
text
```

```
On branch main
```

```
Your branch is ahead of 'origin/main' by 1 commit.
```

```
(use "git push" to publish your local commits)
```

```
nothing to commit, working tree clean
```

Step 14: Push Merged Changes to GitHub

```
bash
```

```
git push origin main
```

 **Expected Output:**

```
text
```

```
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To https://github.com/your-username/git-branching-demo.git
```

```
a1b2c3d..e4f5g6h main -> main
```

Phase 5: Display Chronological Commit History

Step 15: View Detailed Commit History

Show complete commit history:

bash

git log

 **Expected Output:**

text

commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD -> main, origin/main, feature-new-design)

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 15:20:00 2023 +0530

feat: Add new design with enhanced styling and features

commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 15:10:00 2023 +0530

Initial commit: Project setup with basic structure

Step 16: View Compact Commit History

One-line format:

bash

git log --oneline

 **Expected Output:**

text

e4f5g6h (HEAD -> main, origin/main, feature-new-design) feat: Add new design with enhanced styling and features

a1b2c3d Initial commit: Project setup with basic structure

Step 17: View Graphical Commit History

With branch visualization:

bash

```
git log --graph --oneline --all
```

Expected Output:

text

```
* e4f5g6h (HEAD -> main, origin/main, feature-new-design) feat: Add new design with enhanced styling and features
```

```
* a1b2c3d Initial commit: Project setup with basic structure
```

Step 18: View History with Additional Details

Show author and date:

bash

```
git log --pretty=format:"%h - %an, %ar : %s"
```

Expected Output:

text

```
e4f5g6h - Your Name, 10 minutes ago : feat: Add new design with enhanced styling and features
```

```
a1b2c3d - Your Name, 20 minutes ago : Initial commit: Project setup with basic structure
```

Step 19: View File Changes in Commits

See what changed in each commit:

bash

```
git log --stat
```

Expected Output:

text

```
commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t
```

```
Author: Your Name <your.email@example.com>
```

```
Date: Mon Nov 24 15:20:00 2023 +0530
```

feat: Add new design with enhanced styling and features

```
README.md | 8 ++++++++
index.html | 32 ++++++*****+
styles.css | 12 ++++++++
3 files changed, 51 insertions(+), 1 deletion(-)
```

commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 15:10:00 2023 +0530

Initial commit: Project setup with basic structure

```
README.md | 10 ++++++++
index.html | 35 ++++++*****+
2 files changed, 45 insertions(+)
```

Phase 6: Additional Git Operations

Step 20: Create Another Branch for Practice

Create bugfix branch:

bash

```
git checkout -b bugfix-typo
```

Make a small change in README.md:

markdown

```
# Git Branching Demo Project
```

This project demonstrates Git operations including:

- Branch creation
- Merging branches

- Commit history management
- Conflict resolution practice

```
## Project Structure  
- `index.html` - Main web page  
- `README.md` - Project documentation  
- `styles.css` - Styling from feature branch
```

New Features (from feature branch)

- Enhanced UI design
- Improved user experience
- Better documentation
- Bug fixes applied

Commit the change:

```
bash  
git add README.md  
git commit -m "fix: Update documentation and fix typos"
```

Step 21: Merge Bugfix Branch

```
bash  
git checkout main  
git merge bugfix-typo
```

Expected Output:

```
text  
Updating e4f5g6h..8901j2k  
Fast-forward  
 README.md | 4 +-  
 1 file changed, 3 insertions(+), 1 deletion(-)
```

Step 22: View Updated Commit History

```
bash  
git log --oneline --graph --all
```

 **Expected Output:**

```
text  
* 8901j2k (HEAD -> main, bugfix-typo) fix: Update documentation and fix typos  
* e4f5g6h (origin/main, feature-new-design) feat: Add new design with enhanced styling  
and features  
* a1b2c3d Initial commit: Project setup with basic structure
```

Step 23: Push Final Changes

```
bash  
git push origin main
```

 **Complete Workflow Summary**

All Commands Used:

```
bash  
# Setup  
mkdir git-branching-demo  
cd git-branching-demo  
git init
```

```
# Initial commit  
# Create index.html and README.md  
git add .  
git commit -m "Initial commit"
```

```
# GitHub setup  
git remote add origin https://github.com/your-username/git-branching-demo.git  
git push -u origin main
```

```
# Branching  
git checkout -b feature-new-design  
  
# Make changes to files  
git add .  
git commit -m "feat: Add new features"
```

```
# Merging  
git checkout main  
git merge feature-new-design  
git push origin main
```

```
# View history  
git log  
git log --oneline  
git log --graph --oneline --all
```

⌚ Git Operations Demonstrated

Operation	Command	Purpose
Create Branch	git checkout -b branch-name	Create and switch to new branch
Switch Branch	git checkout branch-name	Move between branches
Merge	git merge branch-name	Combine branch changes
View History	git log	See commit timeline
Compact History	git log --oneline	Simplified commit view

Operation	Command	Purpose
Graph History	git log --graph --all	Visual branch history
Push to Remote	git push origin main	

Slip 4

1) Q.1) Configure Maven to compile the code, run tests, and generate artifacts like JAR files

❖ Software Requirements

Required Software:

1. **Java JDK 11+** - Download from Oracle or OpenJDK
 2. **Apache Maven** - Download from maven.apache.org
 3. **Text Editor/IDE** - VS Code, IntelliJ, or Eclipse
 4. **Terminal/Command Prompt**
-

Phase 1: Setup and Verification

Step 1: Verify Java and Maven Installation

Open Command Prompt/Terminal:

cmd

java -version

Expected Output:

text

java version "11.0.15" 2022-04-19 LTS

Java(TM) SE Runtime Environment 18.9 (build 11.0.15+8-LTS-149)

Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.15+8-LTS-149, mixed mode)

cmd

mvn -version

 **Expected Output:**

text

Apache Maven 3.9.5

Maven home: C:\Program Files\Apache\maven-3.9.5

Java version: 11.0.15, vendor: Oracle Corporation

 **Phase 2: Create Maven Project Structure**

Step 2: Generate Maven Project

Using Maven Archetype:

cmd

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

 **Expected Output:**

text

[INFO] Scanning for projects...

[INFO]

[INFO] -----< org.apache.maven:standalone-pom >-----

[INFO] Building Maven Stub Project (No POM) 1

[INFO] -----[pom]-----

[INFO]

[INFO] >>> archetype:3.2.1:generate (default-cli) > generate-sources @ standalone-pom >>>

[INFO]

[INFO] <<< archetype:3.2.1:generate (default-cli) < generate-sources @ standalone-pom <<<

[INFO]

[INFO]

[INFO] --- archetype:3.2.1:generate (default-cli) @ standalone-pom ---

[INFO] Generating project in Batch mode

[INFO] -----

[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0

[INFO] -----

[INFO] Parameter: basedir, Value: C:\projects

[INFO] Parameter: package, Value: com.example

[INFO] Parameter: groupId, Value: com.example

[INFO] Parameter: artifactId, Value: maven-demo-app

[INFO] Parameter: packageName, Value: com.example

[INFO] Parameter: version, Value: 1.0-SNAPSHOT

[INFO] project created from Old (1.x) Archetype in dir: C:\projects\maven-demo-app

[INFO] -----

[INFO] BUILD SUCCESS

[INFO] -----

[INFO] Total time: 10.345 s

[INFO] Finished at: 2023-11-24T16:30:00+05:30

Step 3: Navigate to Project Directory

cmd

cd maven-demo-app

Check project structure:

cmd

dir

Expected Output:

text

Volume in drive C has no label.

Volume Serial Number is XXXX-XXXX

Directory of C:\maven-demo-app

```
11/24/2023 04:30 PM <DIR> .
11/24/2023 04:30 PM <DIR> ..
11/24/2023 04:30 PM      812 pom.xml
11/24/2023 04:30 PM <DIR>     src
1 File(s)      812 bytes
```

Phase 3: Configure POM.xml for Compilation, Testing, and JAR

Step 4: Configure Enhanced POM.xml

Replace the generated pom.xml with this enhanced version:

xml

```
<?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>
```

```
    <!-- Project Coordinates -->
```

```
    <groupId>com.example</groupId>
    <artifactId>maven-demo-app</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>
```

```
    <name>Maven Demo Application</name>
```

```
    <description>A demonstration Maven project for compilation, testing, and JAR
generation</description>
```

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <junit.version>5.9.2</junit.version>
    <maven.compiler.plugin.version>3.11.0</maven.compiler.plugin.version>
    <maven.surefire.plugin.version>3.0.0</maven.surefire.plugin.version>
    <maven.jar.plugin.version>3.3.0</maven.jar.plugin.version>
</properties>

<!-- Dependencies -->
<dependencies>
    <!-- JUnit 5 for Testing -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
</dependencies>

<!-- Apache Commons Lang for utility functions -->
```

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.12.0</version>
</dependency>

<!-- Log4j for logging -->
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.20.0</version>
</dependency>
</dependencies>

<!-- Build Configuration -->
<build>
  <plugins>
    <!-- Compiler Plugin -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${maven.compiler.plugin.version}</version>
      <configuration>
        <source>11</source>
        <target>11</target>
        <encoding>UTF-8</encoding>
        <showWarnings>true</showWarnings>
        <showDeprecation>true</showDeprecation>
      </configuration>
    
```

```
</configuration>

</plugin>

<!-- Surefire Plugin for Running Tests -->

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>${maven.surefire.plugin.version}</version>
    <configuration>
        <includes>
            <include>**/*Test.java</include>
            <include>**/*Tests.java</include>
        </includes>
        <excludes>
            <exclude>**/Abstract*Test.java</exclude>
        </excludes>
        <testFailureIgnore>false</testFailureIgnore>
    </configuration>
</plugin>

<!-- JAR Plugin for Creating Executable JAR -->

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>${maven.jar.plugin.version}</version>
    <configuration>
        <archive>
            <manifest>
```

```
<addClasspath>true</addClasspath>
<classpathPrefix>lib/</classpathPrefix>
<mainClass>com.example.App</mainClass>
</manifest>
</archive>
</configuration>
</plugin>

<!-- Assembly Plugin for Creating Fat JAR -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>3.4.2</version>
  <configuration>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
    <archive>
      <manifest>
        <mainClass>com.example.App</mainClass>
      </manifest>
    </archive>
  </configuration>
  <executions>
    <execution>
      <id>make-assembly</id>
      <phase>package</phase>
      <goals>
```

```
<goal>single</goal>
</goals>
</execution>
</executions>
</plugin>

<!-- Resources Plugin for handling non-Java files -->
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-resources-plugin</artifactId>
<version>3.3.1</version>
<configuration>
<encoding>UTF-8</encoding>
</configuration>
</plugin>
</plugins>

<!-- Resource directories -->
<resources>
<resource>
<directory>src/main/resources</directory>
<filtering>true</filtering>
</resource>
</resources>

<testResources>
<testResource>
<directory>src/test/resources</directory>
```

```

<filtering>true</filtering>

</testResource>
</testResources>
</build>

<!-- Reporting -->
<reporting>
  <plugins>
    <!-- Surefire Report Plugin -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>${maven.surefire.plugin.version}</version>
    </plugin>
    <!-- Javadoc Plugin -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.5.0</version>
    </plugin>
  </plugins>
</reporting>
</project>

```

Phase 4: Create Java Source Code and Tests

Step 5: Create Enhanced Main Application

Replace src/main/java/com/example/App.java:

```
java

package com.example;

import org.apache.commons.lang3.StringUtils;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

/**
 * Main Application Class
 * Demonstrates Maven compilation, testing, and JAR generation
 */

public class App {

    private static final Logger logger = LogManager.getLogger(App.class);

    private String applicationName;

    public App(String applicationName) {
        this.applicationName = applicationName;
        logger.info("Application initialized: {}", applicationName);
    }

    /**
     * Formats the message by capitalizing and adding exclamation
     */
    public String formatMessage(String message) {
        if (StringUtils.isBlank(message)) {
            logger.warn("Empty message provided");
            return "No message provided";
        }
    }
}
```

```
    }

    String formatted = StringUtils.capitalize(message.trim()) + "!";
    logger.debug("Formatted message: {} -> {}", message, formatted);
    return formatted;
}

/** 
 * Reverses a string using StringUtils
 */
public String reverseString(String input) {
    if (StringUtils.isBlank(input)) {
        return "";
    }
    return StringUtils.reverse(input);
}

/** 
 * Checks if string contains only letters
 */
public boolean isAlpha(String input) {
    return StringUtils.isAlpha(input);
}

/** 
 * Gets application information
 */
public String getAppInfo() {
```

```

        return String.format("Application: %s, Version: 1.0.0", applicationName);
    }

public static void main(String[] args) {
    logger.info("Starting Maven Demo Application");

    App app = new App("Maven Demo App");

    System.out.println("==== Maven Configuration Demo ====");
    System.out.println(app.getAppInfo());
    System.out.println();

    // Demonstrate various operations

    String testMessage = "hello maven world";
    System.out.println("Original: " + testMessage);
    System.out.println("Formatted: " + app.formatMessage(testMessage));
    System.out.println("Reversed: " + app.reverseString(testMessage));
    System.out.println("Is Alpha: " + app.isAlpha("Hello"));

    // Demonstrate with empty input

    System.out.println("\n==== Edge Cases ====");
    System.out.println("Empty format: " + app.formatMessage("") + "''");
    System.out.println("Null reverse: " + app.reverseString(null) + "''");

    logger.info("Application completed successfully");
}
}

```

Step 6: Create Comprehensive Test Cases

Replace src/test/java/com/example/AppTest.java:

```
java
package com.example;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import static org.junit.jupiter.api.Assertions.*;

/*
 * Unit tests for App class
 */
class AppTest {

    private App app;

    @BeforeEach
    void setUp() {
        app = new App("Test Application");
    }

    @Test
    @DisplayName("Test application initialization")
    void testAppCreation() {
        assertNotNull(app);
        assertEquals("Test Application", app.getAppInfo().contains("Test Application"));
    }
}
```

```
@Test  
@DisplayName("Test message formatting with valid input")  
void testFormatMessage() {  
    assertEquals("Hello world!", app.formatMessage("hello world"));  
    assertEquals("Maven!", app.formatMessage("maven"));  
    assertEquals("Java!", app.formatMessage("JAVA"));  
}
```

```
@Test  
@DisplayName("Test message formatting with edge cases")  
void testFormatMessageEdgeCases() {  
    assertEquals("No message provided", app.formatMessage(""));  
    assertEquals("No message provided", app.formatMessage(" "));  
    assertEquals("No message provided", app.formatMessage(null));  
}
```

```
@Test  
@DisplayName("Test string reversal")  
void testReverseString() {  
    assertEquals("dlrow nevaM", app.reverseString("Maven world"));  
    assertEquals("avaJ", app.reverseString("Java"));  
    assertEquals("", app.reverseString(""));  
    assertEquals("", app.reverseString(null));  
}
```

```
@Test  
@DisplayName("Test alpha string validation")  
void testIsAlpha() {
```

```
    assertTrue(app.isAlpha("Hello"));

    assertTrue(app.isAlpha("JAVA"));

    assertFalse(app.isAlpha("Hello123"));

    assertFalse(app.isAlpha("Hello World"));

    assertFalse(app.isAlpha(""));

    assertFalse(app.isAlpha(null));

}
```

```
@Test

@DisplayName("Test application information")

void testGetAppInfo() {

    String info = app.getAppInfo();

    assertTrue(info.contains("Application:"));

    assertTrue(info.contains("Version:"));

}

}
```

Step 7: Create Additional Test Class

Create src/test/java/com/example/AdvancedAppTest.java:

```
java

package com.example;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.DisplayName;

import static org.junit.jupiter.api.Assertions.*;

/**

 * Advanced test cases for App class

 */
```

```

class AdvancedAppTest {

    private App app = new App("Advanced Test App");

    @Test
    @DisplayName("Test multiple formatting operations")
    void testMultipleOperations() {
        String original = "test string";
        String formatted = app.formatMessage(original);
        String reversed = app.reverseString(formatted);

        assertEquals("Test string!", formatted);
        assertEquals("!gnirts tseT", reversed);
        assertFalse(app.isAlpha(original + "123"));
    }

    @Test
    @DisplayName("Test with special characters")
    void testSpecialCharacters() {
        assertEquals("Hello123!", app.formatMessage("hello123"));
        assertEquals("321olleH", app.reverseString("Hello123"));
    }
}

```

Step 8: Create Configuration Files

Create src/main/resources/log4j2.xml:

```

xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
```

```

<Appenders>
    <Console name="Console" target="SYSTEM_OUT">
        <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
         %msg%n"/>
    </Console>
    <File name="FileAppender" fileName="logs/application.log">
        <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %-5level
         %logger{36} - %msg%n"/>
    </File>
</Appenders>
<Loggers>
    <Root level="info">
        <AppenderRef ref="Console"/>
        <AppenderRef ref="FileAppender"/>
    </Root>
    <Logger name="com.example" level="debug" additivity="false">
        <AppenderRef ref="Console"/>
        <AppenderRef ref="FileAppender"/>
    </Logger>
</Loggers>
</Configuration>

```

Phase 5: Maven Commands for Compilation, Testing, and Packaging

Step 9: Compile the Code

cmd

mvn compile

Expected Output:

text

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:maven-demo-app >-----
[INFO] Building Maven Demo Application 1.0.0
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:3.3.1:resources (default-resources) @ maven-demo-app ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO]
[INFO] --- maven-compiler-plugin:3.11.0:compile (default-compile) @ maven-demo-app ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\maven-demo-app\target\classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.345 s
[INFO] Finished at: 2023-11-24T16:45:00+05:30
```

Step 10: Run Tests

cmd

mvn test

Expected Output:

text

```
[INFO] Scanning for projects...
```

```
[INFO]
```

```
[INFO] -----< com.example:maven-demo-app >-----
```

```
[INFO] Building Maven Demo Application 1.0.0
```

```
[INFO] -----[ jar ]-----  
[INFO]  
[INFO] --- maven-resources-plugin:3.3.1:testResources (default-testResources) @  
maven-demo-app ---  
[INFO] skip non existing resourceDirectory C:\maven-demo-app\src\test\resources  
[INFO]  
[INFO] --- maven-compiler-plugin:3.11.0:testCompile (default-testCompile) @ maven-  
demo-app ---  
[INFO] Changes detected - recompiling the module!  
[INFO] Compiling 2 source files to C:\maven-demo-app\target\test-classes  
[INFO]  
[INFO] --- maven-surefire-plugin:3.0.0:test (default-test) @ maven-demo-app ---  
[INFO] Using auto detected provider  
org.apache.maven.surefire.junitplatform.JUnitPlatformProvider  
[INFO]  
[INFO] -----  
[INFO] T E S T S  
[INFO] -----  
[INFO] Running com.example.AppTest  
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.345 s - in  
com.example.AppTest  
[INFO] Running com.example.AdvancedAppTest  
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.045 s - in  
com.example.AdvancedAppTest  
[INFO]  
[INFO] Results:  
[INFO]  
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0  
[INFO]
```

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 3.567 s  
[INFO] Finished at: 2023-11-24T16:46:00+05:30
```

Step 11: Package the Application (Generate JAR)

```
cmd  
mvn package
```

Expected Output:

```
text  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----< com.example:maven-demo-app >-----  
[INFO] Building Maven Demo Application 1.0.0  
[INFO] -----[ jar ]-----  
[INFO]  
[INFO] --- maven-resources-plugin:3.3.1:resources (default-resources) @ maven-demo-app ---  
[INFO] Copying 1 resource from src/main/resources to target/classes  
[INFO]  
[INFO] --- maven-compiler-plugin:3.11.0:compile (default-compile) @ maven-demo-app ---  
[INFO] Nothing to compile - all classes are up to date  
[INFO]  
[INFO] --- maven-resources-plugin:3.3.1:testResources (default-testResources) @ maven-demo-app ---  
[INFO] skip non existing resourceDirectory C:\maven-demo-app\src\test\resources  
[INFO]
```

```
[INFO] --- maven-compiler-plugin:3.11.0:testCompile (default-testCompile) @ maven-demo-app ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:3.0.0:test (default-test) @ maven-demo-app ---
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- maven-jar-plugin:3.3.0:jar (default-jar) @ maven-demo-app ---
[INFO] Building jar: C:\maven-demo-app\target\maven-demo-app-1.0.0.jar
[INFO]
[INFO] --- maven-assembly-plugin:3.4.2:single (make-assembly) @ maven-demo-app --
-
[INFO] Building jar: C:\maven-demo-app\target\maven-demo-app-1.0.0-jar-with-dependencies.jar
[INFO] -----
```

[INFO] BUILD SUCCESS

[INFO] -----

[INFO] Total time: 4.123 s

[INFO] Finished at: 2023-11-24T16:47:00+05:30

Step 12: Verify Generated Artifacts

cmd

dir target

Expected Output:

text

Volume in drive C has no label.

Volume Serial Number is XXXX-XXXX

Directory of C:\maven-demo-app\target

```
11/24/2023 04:47 PM <DIR> .
11/24/2023 04:47 PM <DIR> ..
11/24/2023 04:47 PM <DIR> classes
11/24/2023 04:47 PM <DIR> generated-sources
11/24/2023 04:47 PM <DIR> maven-archiver
11/24/2023 04:47 PM <DIR> maven-status
11/24/2023 04:47 PM <DIR> surefire-reports
11/24/2023 04:47 PM <DIR> test-classes
11/24/2023 04:47 PM 8,456 maven-demo-app-1.0.0.jar
11/24/2023 04:47 PM 2,345,678 maven-demo-app-1.0.0-jar-with-dependencies.jar
2 File(s) 2,354,134 bytes
```

Phase 6: Test the Generated JAR Files

Step 13: Run the Regular JAR File

cmd

```
java -cp target/maven-demo-app-1.0.0.jar com.example.App
```

Expected Output:

text

```
16:48:00.123 [main] INFO com.example.App - Starting Maven Demo Application
```

```
16:48:00.125 [main] INFO com.example.App - Application initialized: Maven Demo App
```

```
==== Maven Configuration Demo ====
```

```
Application: Maven Demo App, Version: 1.0.0
```

Original: hello maven world

Formatted: Hello maven world!

Reversed: dlrow nevaM olleh

Is Alpha: true

==== Edge Cases ====

Empty format: 'No message provided'

Null reverse: "

16:48:00.128 [main] INFO com.example.App - Application completed successfully

Step 14: Run the Fat JAR (with dependencies)

cmd

java -jar target/maven-demo-app-1.0.0-jar-with-dependencies.jar

 **Expected Output:** Same as above, but includes all dependencies

Phase 7: Additional Maven Operations

Step 15: Clean and Rebuild

cmd

mvn clean package

 **Expected Output:** Deletes target directory and rebuilds everything

Step 16: Skip Tests

cmd

mvn package -DskipTests

 **Expected Output:** Packages without running tests

Step 17: Generate Site Documentation

cmd

mvn site

 **Expected Output:** Generates project documentation in target/site/

Step 18: View Dependency Tree

cmd

mvn dependency:tree

 **Expected Output:**

text

```
[INFO] com.example:maven-demo-app:jar:1.0.0
[INFO] +- org.junit.jupiter:junit-jupiter-api:jar:5.9.2:test
[INFO] | +- org.junit:junit-bom:jar:5.9.2:test
[INFO] | +- org.apiguardian:apiguardian-api:jar:1.1.2:test
[INFO] | \- org.opentest4j:opentest4j:jar:1.2.0:test
[INFO] +- org.junit.jupiter:junit-jupiter-engine:jar:5.9.2:test
[INFO] | \- org.junit.platform:junit-platform-engine:jar:1.9.2:test
[INFO] +- org.apache.commons:commons-lang3:jar:3.12.0:compile
[INFO] \- org.apache.logging.log4j:log4j-core:jar:2.20.0:compile
[INFO]   \- org.apache.logging.log4j:log4j-api:jar:2.20.0:compile
```

Maven Lifecycle Summary

Phase	Command	Purpose
Compile	mvn compile	Compiles source code
Test	mvn test	Runs unit tests
Package	mvn package	Creates JAR/WAR files
Install	mvn install	Installs to local repository
Clean	mvn clean	Deletes target directory
Site	mvn site	Generates documentation

Q2) Q.2) 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

Git Commands Demonstration - Complete Step-by-Step Guide

I'll walk you through creating a project and using essential Git commands to check status, view history, see differences, make commits, and display chronological history.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
 2. **Text Editor** - VS Code, Notepad++, etc.
 3. **Terminal/Command Prompt**
-

Phase 1: Project Setup and Initial Commit

Step 1: Create Project Directory and Initialize Git

Open Terminal/Command Prompt:

bash

Create and navigate to project directory

mkdir git-commands-demo

cd git-commands-demo

Initialize Git repository

git init

Expected Output:

text

Initialized empty Git repository in C:/git-commands-demo/.git/

Step 2: Configure Git User (if not done globally)

bash

git config user.name "Your Name"

```
git config user.email "your.email@example.com"
```

Step 3: Create Initial Project Files

Create index.html:

html

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Git Commands Demo</title>
  </head>
  <body>
    <h1>Git Commands Demonstration</h1>
    <p>This project demonstrates various Git commands and operations.</p>

    <section id="features">
      <h2>Features:</h2>
      <ul>
        <li>Git status checking</li>
        <li>Commit history viewing</li>
        <li>Difference tracking</li>
        <li>Branch management</li>
      </ul>
    </section>
  </body>
</html>
```

Create README.md:

markdown

Git Commands Demo Project

This project demonstrates essential Git commands for version control.

Git Operations Covered:

- git status

- git log

- git diff

- git add

- git commit

- git branch

Create styles.css:

```
css
```

```
/* Basic styling for demo project */
```

```
body {
```

```
    font-family: Arial, sans-serif;
```

```
    line-height: 1.6;
```

```
    margin: 0;
```

```
    padding: 20px;
```

```
    background-color: #f4f4f4;
```

```
}
```

```
.container {
```

```
    max-width: 800px;
```

```
    margin: 0 auto;
```

```
    background: white;
```

```
    padding: 20px;
```

```
    border-radius: 5px;
```

```
        box-shadow: 0 0 10px rgba(0,0,0,0.1);  
    }  


---


```

Phase 2: Git Status and Initial Commit

Step 4: Check Git Status

bash

```
git status
```

Expected Output:

text

```
On branch main
```

```
No commits yet
```

Untracked files:

(use "git add <file>..." to include in what will be committed)

```
README.md
```

```
index.html
```

```
styles.css
```

nothing added to commit but untracked files present (use "git add" to track)

Step 5: Stage Files and Make Initial Commit

Stage all files:

bash

```
git add .
```

Check status after staging:

bash

```
git status
```

Expected Output:

text

On branch main

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

 new file: README.md

 new file: index.html

 new file: styles.css

Make initial commit:

bash

```
git commit -m "Initial commit: Project setup with basic structure"
```

 **Expected Output:**

text

```
[main (root-commit) a1b2c3d] Initial commit: Project setup with basic structure
```

```
3 files changed, 45 insertions(+)
```

```
create mode 100644 README.md
```

```
create mode 100644 index.html
```

```
create mode 100644 styles.css
```

Phase 3: View Log History and Make Changes

Step 6: View Commit History

Basic log:

bash

```
git log
```

 **Expected Output:**

text

commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r (HEAD -> main)

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 17:30:00 2023 +0530

Initial commit: Project setup with basic structure

One-line log:

bash

git log --oneline

 **Expected Output:**

text

a1b2c3d (HEAD -> main) Initial commit: Project setup with basic structure

Detailed log with statistics:

bash

git log --stat

 **Expected Output:**

text

commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r (HEAD -> main)

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 17:30:00 2023 +0530

Initial commit: Project setup with basic structure

README.md | 10 ++++++++

index.html | 25 ++++++*****

styles.css | 10 ++++++++

3 files changed, 45 insertions(+)

Step 7: Make Changes to Files

Update index.html - add new section:

html

```
<!-- Add this before closing body tag -->  
<section id="git-commands">  
  <h2>Essential Git Commands:</h2>  
  <div class="command-list">  
    <div class="command">  
      <code>git status</code> - Check current status  
    </div>  
    <div class="command">  
      <code>git log</code> - View commit history  
    </div>  
    <div class="command">  
      <code>git diff</code> - See changes  
    </div>  
  </div>  
</section>
```

<footer>

```
  <p>Project last updated: <span id="update-date">Today</span></p>  
</footer>
```

Update styles.css - add new styles:

css

```
/* Add these styles */
```

```
.command-list {  
  margin: 20px 0;  
}  
  
 .command {
```

```
background: #f8f9fa;  
padding: 10px;  
margin: 5px 0;  
border-left: 4px solid #007bff;  
font-family: 'Courier New', monospace;  
}
```

```
footer{  
margin-top: 30px;  
padding-top: 20px;  
border-top: 1px solid #ddd;  
color: #666;  
}
```

Update README.md:
markdown
Git Commands Demo Project

This project demonstrates essential Git commands for version control.

Git Operations Covered:

- git status
- git log
- git diff
- git add
- git commit
- git branch
- git merge

Project Structure:

- `index.html` - Main web page
 - `styles.css` - Styling
 - `README.md` - Documentation
-

Phase 4: Using Git Diff to See Changes

Step 8: Check Current Status

bash

git status

Expected Output:

text

On branch main

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: README.md

modified: index.html

modified: styles.css

no changes added to commit (use "git add" and/or "git commit -a")

Step 9: View Differences with Git Diff

See all changes:

bash

git diff

Expected Output:

text

diff --git a/README.md b/README.md

index 1234567..890abcd 100644

```
--- a/README.md
+++ b/README.md

@@ -4,4 +4,9 @@ This project demonstrates essential Git commands for version
control.

## Git Operations Covered:

- git status
- git log
-- git diff

\ No newline at end of file

+- git diff
+- git add
+- git commit
+- git branch
+- git merge
+
+## Project Structure:

+- `index.html` - Main web page
+- `styles.css` - Styling
+- `README.md` - Documentation

diff --git a/index.html b/index.html
index 2345678..9abcdef 100644
--- a/index.html
+++ b/index.html

@@ -14,5 +14,20 @@
<li>Branch management</li>
</ul>
</section>

+
```

```
+ <section id="git-commands">
+   <h2>Essential Git Commands:</h2>
+   <div class="command-list">
+     <div class="command">
+       <code>git status</code> - Check current status
+     </div>
+     <!-- More commands shown -->
+   </div>
+ </section>
+
+ <footer>
+   <p>Project last updated: <span id="update-date">Today</span></p>
+ </footer>
</body>
</html>
```

See differences for a specific file:

bash

git diff index.html

 **Expected Output:** Shows only changes in index.html

See staged differences:

bash

git add .

git diff --staged

 **Expected Output:** Shows changes that are staged for commit

Phase 5: Making Second Commit

Step 10: Stage Changes and Commit

Stage all changes:

bash

git add .

Check status after staging:

bash

git status

 **Expected Output:**

text

On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: README.md

modified: index.html

modified: styles.css

Make second commit:

bash

git commit -m "feat: Add Git commands section and improve styling"

 **Expected Output:**

text

[main e4f5g6h] feat: Add Git commands section and improve styling

3 files changed, 35 insertions(+), 2 deletions(-)

 **Phase 6: Display Chronological History**

Step 11: View Complete Commit History

Detailed log:

bash

git log

 **Expected Output:**

text

```
commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD -> main)
```

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 17:45:00 2023 +0530

feat: Add Git commands section and improve styling

```
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r
```

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 17:30:00 2023 +0530

Initial commit: Project setup with basic structure

One-line history:

bash

```
git log --oneline
```

Expected Output:

text

```
e4f5g6h (HEAD -> main) feat: Add Git commands section and improve styling
```

```
a1b2c3d Initial commit: Project setup with basic structure
```

Step 12: View History with Different Formats

Custom format:

bash

```
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

Expected Output:

text

```
e4f5g6h | Your Name | 2023-11-24 | feat: Add Git commands section and improve styling
```

```
a1b2c3d | Your Name | 2023-11-24 | Initial commit: Project setup with basic structure
```

With graph visualization:

bash

```
git log --graph --oneline --all
```

 **Expected Output:**

text

```
* e4f5g6h (HEAD -> main) feat: Add Git commands section and improve styling
```

```
* a1b2c3d Initial commit: Project setup with basic structure
```

Show changes in each commit:

bash

```
git log -p
```

 **Expected Output:** Shows the actual code changes in each commit

Phase 7: Additional Git Operations

Step 13: Create and Switch to New Branch

Create feature branch:

bash

```
git checkout -b feature-enhanced-ui
```

 **Expected Output:**

text

```
Switched to a new branch 'feature-enhanced-ui'
```

Verify current branch:

bash

```
git branch
```

 **Expected Output:**

text

```
main
```

```
* feature-enhanced-ui
```

Step 14: Make Changes in Feature Branch

Create script.js:

```
javascript

// JavaScript functionality for demo

console.log("Git Commands Demo - JavaScript loaded");


function updateTimestamp() {

    const now = new Date();

    document.getElementById('update-date').textContent = now.toLocaleString();

}

// Initialize when page loads

document.addEventListener('DOMContentLoaded', function() {

    updateTimestamp();

    console.log("Git demo project initialized");

});
```

Update index.html to include script:

```
html

<!-- Add before closing body tag -->

<script src="script.js"></script>
```

Step 15: Commit in Feature Branch

Check status:

```
bash

git status
```

 **Expected Output:**

```
text

On branch feature-enhanced-ui

Untracked files:

(use "git add <file>..." to include in what will be committed)

    script.js
```

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: index.html

no changes added to commit (use "git add" and/or "git commit -a")

Stage and commit:

bash

git add .

git commit -m "feat: Add JavaScript functionality and dynamic timestamp"

Expected Output:

text

[feature-enhanced-ui 8901j2k] feat: Add JavaScript functionality and dynamic timestamp

2 files changed, 18 insertions(+)

create mode 100644 script.js

Step 16: Switch Back to Main and View History

Switch to main branch:

bash

git checkout main

Expected Output:

text

Switched to branch 'main'

View complete history:

bash

git log --oneline --all --graph

Expected Output:

text

- * 8901j2k (feature-enhanced-ui) feat: Add JavaScript functionality and dynamic timestamp
 - * e4f5g6h (HEAD -> main) feat: Add Git commands section and improve styling
 - * a1b2c3d Initial commit: Project setup with basic structure
-

Phase 8: Advanced Git Operations

Step 17: See Difference Between Branches

Compare main with feature branch:

bash

```
git diff main..feature-enhanced-ui
```

 **Expected Output:** Shows all differences between the two branches

See what files changed:

bash

```
git diff --name-only main..feature-enhanced-ui
```

 **Expected Output:**

text

index.html

script.js

Step 18: View Specific Commit Details

Show details of a specific commit:

bash

```
git show e4f5g6h
```

 **Expected Output:** Shows the commit details and changes made in that specific commit

Show file changes in a readable format:

bash

```
git show --stat
```

 **Expected Output:** Shows statistics of changes in the latest commit

Step 19: Create and View Tags

Create a version tag:

bash

```
git tag v1.0.0
```

View all tags:

bash

```
git tag
```

 **Expected Output:**

```
text
```

```
v1.0.0
```

View history with tags:

bash

```
git log --oneline --decorate
```

 **Expected Output:**

```
text
```

```
e4f5g6h (HEAD -> main, tag: v1.0.0) feat: Add Git commands section and improve styling
```

```
a1b2c3d Initial commit: Project setup with basic structure
```

Complete Git Workflow Recap

All Commands Used in This Demo:

bash

```
# Setup and initialization
```

```
mkdir git-commands-demo
```

```
cd git-commands-demo
```

```
git init
```

Initial commit

```
git add .
git commit -m "Initial commit"
```

Status checking

```
git status
```

Viewing history

```
git log
git log --oneline
git log --stat
git log --graph --oneline --all
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

Viewing differences

```
git diff
git diff filename
git diff --staged
git diff main..feature-branch
```

Making changes and commits

```
git add .
git commit -m "Commit message"
```

Branch operations

```
git checkout -b branch-name
git branch
git checkout main
```

Tagging

```
git tag v1.0.0
```

```
git tag
```

Detailed views

```
git show commit-hash
```

```
git show --stat
```

Git Commands Summary

Command	Purpose	Example Output
git status	Check current state	Shows modified/untracked files
git log	View commit history	Chronological commit list
git log --oneline	Compact history	One-line per commit
git diff	See changes	Line-by-line differences
git add	Stage changes	Files ready for commit
git commit	Save changes	Commit confirmation
git branch	List branches	Available branches
git checkout	Switch branches	Branch change confirmation

Expected Final Commit History

text

```
* 8901j2k (feature-enhanced-ui) feat: Add JavaScript functionality and dynamic timestamp
```

- * e4f5g6h (HEAD -> main, tag: v1.0.0) feat: Add Git commands section and improve styling
- * a1b2c3d Initial commit: Project setup with basic structure



Slip 5)

Q.1) 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.

BitBucket Repository Management - Complete Step-by-Step Guide

I'll walk you through creating a project, pushing to BitBucket, branching, merging, and displaying commit history.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
 2. **BitBucket Account** - Create at bitbucket.org
 3. **Text Editor** - VS Code, Notepad++, etc.
 4. **Terminal/Command Prompt**
-

Phase 1: BitBucket Account Setup

Step 1: Create BitBucket Account (if you don't have one)

1. **Go to bitbucket.org**
2. **Click "Get started for free"**
3. **Sign up** with email or Google account
4. **Verify your email address**

Step 2: Create SSH Key (Recommended for Secure Access)

Generate SSH key:

bash

```
ssh-keygen -t rsa -b 4096 -C "your.email@example.com"
```

Expected Output:

text

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/YourName/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/YourName/.ssh/id_rsa

Your public key has been saved in /c/Users/YourName/.ssh/id_rsa.pub

Copy public key:

bash

On Windows

```
cat ~/.ssh/id_rsa.pub
```

On Linux/Mac

```
cat ~/.ssh/id_rsa.pub | clip # or pbcopy on Mac
```

Add SSH key to BitBucket:

1. BitBucket → Personal settings → SSH keys
 2. Click "Add key"
 3. Paste your public key
 4. Click "Save"
-

Phase 2: Create Local Project and Git Setup

Step 3: Create Project Directory

bash

Create and navigate to project directory

```
mkdir bitbucket-demo-project
```

```
cd bitbucket-demo-project
```

Step 4: Initialize Git Repository

bash

```
git init
```

 **Expected Output:**

text

```
Initialized empty Git repository in C:/bitbucket-demo-project/.git/
```

Step 5: Configure Git User

bash

```
git config user.name "Your Name"
```

```
git config user.email "your.email@example.com"
```

Step 6: Create Project Files

Create index.html:

html

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>BitBucket Demo Project</title>
    <style>
      body {
        font-family: Arial, sans-serif;
        max-width: 800px;
        margin: 0 auto;
        padding: 20px;
        background: linear-gradient(135deg, #0052cc 0%, #00a3ff 100%);
        color: white;
      }
      .container {
        background: rgba(255, 255, 255, 0.1);
      }
    </style>
  </head>
  <body>
    <div class="container">
      <h1>Welcome to BitBucket Demo Project!</h1>
      <p>This is a simple demo project hosted on BitBucket. You can clone this repository and make changes to the files. The project includes an index.html file with some basic styling and a container div with a semi-transparent background.</p>
    </div>
  </body>
</html>
```

```
padding: 30px;  
border-radius: 15px;  
backdrop-filter: blur(10px);  
}  
.feature-card {  
background: rgba(0, 0, 0, 0.2);  
padding: 15px;  
margin: 10px 0;  
border-radius: 8px;  
border-left: 4px solid #00c7ff;  
}  
</style>  
</head>  
<body>  
<div class="container">  
  <h1>↗ BitBucket Repository Demo</h1>  
  <p>This project demonstrates Git operations with BitBucket remote repository.</p>  
  
<div class="feature-card">  
  <h3>📋 Project Features:</h3>  
  <ul>  
    <li>Remote repository on BitBucket</li>  
    <li>Branch creation and management</li>  
    <li>Merge operations</li>  
    <li>Commit history tracking</li>  
  </ul>  
</div>
```

```
<div class="feature-card">  
  <h3>🔧 Git Operations Demonstrated:</h3>  
  <ul>  
    <li>git init - Initialize repository</li>  
    <li>git add - Stage changes</li>  
    <li>git commit - Save changes</li>  
    <li>git branch - Create branches</li>  
    <li>git merge - Combine branches</li>  
    <li>git push - Upload to BitBucket</li>  
  </ul>  
</div>  
</div>  
</body>  
</html>
```

Create README.md:

markdown

```
# BitBucket Demo Project
```

This project demonstrates Git and BitBucket operations including:

- Repository creation and management
- Branching and merging
- Remote repository synchronization
- Commit history management

Project Structure

- `index.html` - Main web application
- `README.md` - Project documentation

```
## Git Commands Used  
- `git init` - Initialize repository  
- `git add` - Stage files  
- `git commit` - Commit changes  
- `git branch` - Branch operations  
- `git merge` - Merge branches  
- `git push` - Push to BitBucket
```

Create .gitignore:

bash

```
# Create .gitignore file
```

notepad .gitignore

Content:

text

```
# Node.js
```

```
node_modules/
```

```
npm-debug.log*
```

```
# Python
```

```
__pycache__/
```

```
*.pyc
```

```
.env
```

```
# IDE
```

```
.vscode/
```

```
.idea/
```

```
*.swp
```

```
*.swo
```

```
# OS  
.DS_Store  
Thumbs.db
```

```
# Logs  
.log  
logs/
```

```
# Build outputs  
dist/  
build/
```

Step 7: Make Initial Commit

Check status:

```
bash  
git status
```

Expected Output:

```
text  
On branch main
```

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

```
.gitignore  
README.md  
index.html
```

nothing added to commit but untracked files present (use "git add" to track)

Stage files and commit:

bash

git add .

git commit -m "Initial commit: Project setup with basic structure and documentation"

Expected Output:

text

[main (root-commit) a1b2c3d] Initial commit: Project setup with basic structure and documentation

3 files changed, 78 insertions(+)

create mode 100644 .gitignore

create mode 100644 README.md

create mode 100644 index.html

Phase 3: BitBucket Repository Setup

Step 8: Create BitBucket Repository

1. **Log in to [bitbucket.org](#)**
2. **Click "Create" → "Repository"**
3. **Repository details:**
 - **Repository name:** bitbucket-demo-project
 - **Description:** "Demo project for BitBucket Git operations"
 - **Visibility: Private** (for security)
 - **Include README: No** (we have our own)
 - **Git ignore: None** (we have our own)
4. **Click "Create repository"**

Step 9: Connect Local Repository to BitBucket

Copy the repository URL from BitBucket (use SSH or HTTPS):

For HTTPS:

bash

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

For SSH (recommended):

bash

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

Verify remote connection:

bash

```
git remote -v
```

Expected Output:

text

```
origin git@bitbucket.org:your-username/bitbucket-demo-project.git (fetch)
```

```
origin git@bitbucket.org:your-username/bitbucket-demo-project.git (push)
```

Step 10: Push to BitBucket

First push (set upstream):

bash

```
git push -u origin main
```

Expected Output:

text

```
Enumerating objects: 6, done.
```

```
Counting objects: 100% (6/6), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (5/5), done.
```

```
Writing objects: 100% (6/6), 1.78 KiB | 1.78 MiB/s, done.
```

```
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To bitbucket.org:your-username/bitbucket-demo-project.git
```

```
* [new branch] main -> main
```

```
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Step 11: Verify on BitBucket

1. Refresh your BitBucket repository page
 2. You should see your files
 3. Click on "Commits" to see the initial commit
-

Phase 4: Create and Work on New Branch

Step 12: Create Feature Branch

Create and switch to new branch:

bash

```
git checkout -b feature-enhanced-styling
```

Expected Output:

text

```
Switched to a new branch 'feature-enhanced-styling'
```

Verify current branch:

bash

```
git branch
```

Expected Output:

text

main

```
* feature-enhanced-styling
```

Step 13: Make Changes in Feature Branch

Update index.html - add enhanced features:

html

```
<!-- Add this before closing body tag -->
```

```
<section style="margin: 30px 0;">
```

```
<div class="feature-card" style="background: rgba(0, 199, 255, 0.2);">
```

```
<h3>  New Features from Feature Branch:</h3>
```

```
<ul>
```

```
<li>Enhanced color scheme</li>
```

```

<li>Improved responsive design</li>
<li>Better user interface elements</li>
<li>Additional content sections</li>
</ul>
</div>
</section>

<section style="background: rgba(255, 255, 255, 0.1); padding: 20px; border-radius: 10px; margin: 20px 0;">
<h3>🔧 Branch Information:</h3>
<p><strong>Current Branch:</strong> <span style="color: #00c7ff;">feature-enhanced-styling</span></p>
<p><strong>Changes:</strong> Added enhanced styling and new features</p>
<p><strong>Commit:</strong> This content was added in the feature branch</p>
</section>

<footer style="margin-top: 40px; padding-top: 20px; border-top: 1px solid rgba(255,255,255,0.3);>
<p>📌 BitBucket Git Operations Demo - Feature Branch</p>
</footer>

Update README.md:
markdown
# BitBucket Demo Project

```

This project demonstrates Git and BitBucket operations including:

- Repository creation and management
- Branching and merging
- Remote repository synchronization
- Commit history management

Project Structure

- `index.html` - Main web application
- `README.md` - Project documentation

Git Commands Used

- `git init` - Initialize repository
- `git add` - Stage files
- `git commit` - Commit changes
- `git branch` - Branch operations
- `git merge` - Merge branches
- `git push` - Push to BitBucket

Feature Branch Updates

- Enhanced styling and UI improvements
- Additional content sections
- Better user experience
- Responsive design enhancements

Create styles.css (new file):

css

```
/* Additional styles from feature branch */  
.button {  
    background: linear-gradient(135deg, #00c7ff 0%, #0052cc 100%);  
    color: white;  
    padding: 12px 24px;  
    border: none;  
    border-radius: 25px;  
    cursor: pointer;
```

```
    font-size: 16px;  
    transition: all 0.3s ease;  
}  
  
.button:hover {  
    transform: translateY(-2px);  
    box-shadow: 0 5px 15px rgba(0, 199, 255, 0.4);  
}  
  
.code-block {  
    background: rgba(0, 0, 0, 0.3);  
    padding: 15px;  
    border-radius: 8px;  
    font-family: 'Courier New', monospace;  
    border-left: 4px solid #00c7ff;  
    margin: 15px 0;  
}
```

Update index.html to include new CSS:

```
html  
<!-- Add in head section -->  
<link rel="stylesheet" href="styles.css">
```

Step 14: Commit Changes in Feature Branch

Check status:

```
bash  
git status
```

 **Expected Output:**

```
text  
On branch feature-enhanced-styling
```

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: README.md

modified: index.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

styles.css

no changes added to commit (use "git add" and/or "git commit -a")

Stage and commit changes:

bash

git add .

git commit -m "feat: Add enhanced styling, new CSS, and improved content structure"

Expected Output:

text

[feature-enhanced-styling e4f5g6h] feat: Add enhanced styling, new CSS, and improved content structure

3 files changed, 67 insertions(+), 2 deletions(-)

create mode 100644 styles.css

Step 15: Push Feature Branch to BitBucket

bash

git push -u origin feature-enhanced-styling

Expected Output:

text

Enumerating objects: 8, done.

Counting objects: 100% (8/8), done.

```
Delta compression using up to 8 threads  
Compressing objects: 100% (6/6), done.  
Writing objects: 100% (6/6), 1.45 KiB | 1.45 MiB/s, done.
```

```
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
```

```
remote:
```

```
remote: Create pull request for feature-enhanced-styling:
```

```
remote: https://bitbucket.org/your-username/bitbucket-demo-project/pull-  
requests/new?source=feature-enhanced-styling&t=1
```

```
remote:
```

```
To bitbucket.org:your-username/bitbucket-demo-project.git
```

```
* [new branch]  feature-enhanced-styling -> feature-enhanced-styling
```

```
Branch 'feature-enhanced-styling' set up to track remote branch 'feature-enhanced-  
styling' from 'origin'.
```

Phase 5: Merge Branch into Main

Step 16: Switch Back to Main Branch

```
bash
```

```
git checkout main
```

Expected Output:

```
text
```

```
Switched to branch 'main'
```

```
Your branch is up to date with 'origin/main'.
```

Step 17: Merge Feature Branch into Main

```
bash
```

```
git merge feature-enhanced-styling
```

Expected Output:

```
text
```

```
Updating a1b2c3d..e4f5g6h
```

Fast-forward

```
 README.md | 9 ++++++++
 index.html | 47 ++++++=====
 styles.css | 23 ++++++=====
 3 files changed, 79 insertions(+)
 create mode 100644 styles.css
```

Step 18: Push Merged Changes to BitBucket

bash

```
git push origin main
```

 **Expected Output:**

text

```
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To bitbucket.org:your-username/bitbucket-demo-project.git
```

```
 a1b2c3d..e4f5g6h main -> main
```

Phase 6: Display Chronological Commit History

Step 19: View Complete Commit History

Detailed log:

bash

```
git log
```

 **Expected Output:**

text

```
commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD -> main, origin/main,
origin/feature-enhanced-styling, feature-enhanced-styling)
```

```
Author: Your Name <your.email@example.com>
```

```
Date: Mon Nov 24 18:20:00 2023 +0530
```

```
feat: Add enhanced styling, new CSS, and improved content structure
```

```
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r (origin/main)
```

```
Author: Your Name <your.email@example.com>
```

```
Date: Mon Nov 24 18:10:00 2023 +0530
```

Initial commit: Project setup with basic structure and documentation

One-line history:

```
bash
```

```
git log --oneline
```

Expected Output:

text

```
e4f5g6h (HEAD -> main, origin/main, origin/feature-enhanced-styling, feature-enhanced-styling) feat: Add enhanced styling, new CSS, and improved content structure
```

```
a1b2c3d Initial commit: Project setup with basic structure and documentation
```

Step 20: View Graphical Commit History

With branch visualization:

```
bash
```

```
git log --graph --oneline --all
```

Expected Output:

text

```
* e4f5g6h (HEAD -> main, origin/main, origin/feature-enhanced-styling, feature-enhanced-styling) feat: Add enhanced styling, new CSS, and improved content structure
```

```
* a1b2c3d Initial commit: Project setup with basic structure and documentation
```

Step 21: View History with Additional Details

Custom format:

```
bash
```

```
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

Expected Output:

text

e4f5g6h | Your Name | 2023-11-24 | feat: Add enhanced styling, new CSS, and improved content structure

a1b2c3d | Your Name | 2023-11-24 | Initial commit: Project setup with basic structure and documentation

Show file changes in commits:

bash

git log --stat

Expected Output:

text

commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD -> main, origin/main, origin/feature-enhanced-styling, feature-enhanced-styling)

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 18:20:00 2023 +0530

feat: Add enhanced styling, new CSS, and improved content structure

README.md | 9 +++++++

index.html | 47 ++++++ ++++++ ++++++ ++++++ ++++++ ++++++ ++++++ ++++++ ++++++ ++++++

styles.css | 23 ++++++ ++++++ ++++++ ++++++

3 files changed, 79 insertions(+)

commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 18:10:00 2023 +0530

Initial commit: Project setup with basic structure and documentation

```
.gitignore | 5 ++++++
README.md | 15 ++++++=====
index.html | 58
+++++++++++++++++++++++++++++
3 files changed, 78 insertions(+)
```

Phase 7: BitBucket Web Interface Operations

Step 22: Verify on BitBucket Web Interface

1. **Go to your BitBucket repository**
2. **Click "Branches"** - You should see both main and feature-enhanced-styling
3. **Click "Commits"** - See the chronological commit history
4. **Click "Source"** - Browse the merged files

Step 23: Create Pull Request (Optional - BitBucket Feature)

In BitBucket:

1. **Go to "Pull requests"**
 2. **Click "Create pull request"**
 3. **Source branch:** feature-enhanced-styling
 4. **Destination branch:** main
 5. **Add description:** "Merging feature branch with enhanced styling"
 6. **Click "Create"**
 7. **Click "Merge"** (if you want to practice PR workflow)
-

Phase 8: Additional Git Operations

Step 24: Create Another Branch for Practice

Create bugfix branch:

bash

```
git checkout -b hotfix-readme-update
```

Update README.md:

markdown

BitBucket Demo Project

This project demonstrates Git and BitBucket operations including:

- Repository creation and management
- Branching and merging
- Remote repository synchronization
- Commit history management

Project Structure

- `index.html` - Main web application
- `styles.css` - Enhanced styling
- `README.md` - Project documentation
- ` .gitignore` - Git ignore rules

Git Commands Used

- `git init` - Initialize repository
- `git add` - Stage files
- `git commit` - Commit changes
- `git branch` - Branch operations
- `git merge` - Merge branches
- `git push` - Push to BitBucket

Feature Updates

- Enhanced styling and UI improvements
- Additional content sections
- Better user experience
- Responsive design enhancements

```
## Recent Changes
```

- Fixed documentation formatting
- Updated project structure details
- Added recent changes section

Commit and push:

```
bash
```

```
git add README.md
```

```
git commit -m "docs: Update README with improved documentation and recent changes"
```

```
git push -u origin hotfix-readme-update
```

Step 25: Merge and View Updated History

```
bash
```

```
git checkout main
```

```
git merge hotfix-readme-update
```

```
git push origin main
```

View final history:

```
bash
```

```
git log --oneline --graph --all
```

Expected Output:

```
text
```

```
* 8901j2k (HEAD -> main, origin/main, origin/hotfix-readme-update, hotfix-readme-update) docs: Update README with improved documentation and recent changes
```

```
* e4f5g6h (origin/feature-enhanced-styling, feature-enhanced-styling) feat: Add enhanced styling, new CSS, and improved content structure
```

```
* a1b2c3d Initial commit: Project setup with basic structure and documentation
```

Complete Workflow Summary

All Commands Used:

```
bash

# Setup

mkdir bitbucket-demo-project
cd bitbucket-demo-project
git init
```

```
# Initial commit

# Create files: index.html, README.md, .gitignore
git add .
git commit -m "Initial commit"
```

```
# BitBucket setup

git remote add origin git@bitbucket.org:your-username/bitbucket-demo-project.git
git push -u origin main
```

```
# Branching

git checkout -b feature-enhanced-styling

# Make changes to files
git add .
git commit -m "feat: Add enhancements"
git push -u origin feature-enhanced-styling
```

```
# Merging

git checkout main
git merge feature-enhanced-styling
git push origin main
```

```
# History
```

```
git log  
git log --oneline  
git log --graph --oneline --all  
git log --stat
```

BitBucket vs GitHub Key Differences

Aspect	BitBucket	GitHub
Free Private Repos	Unlimited	Limited
Integration	Jira, Trello	GitHub Actions
CI/CD	BitBucket Pipelines	GitHub Actions
Pull Requests	Pull Requests	Pull Requests
Interface	Atlassian ecosystem	Microsoft ecosystem

Q2) 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.

GitHub Repository Management with Pull - Complete Step-by-Step Guide

I'll walk you through creating a project, pushing to GitHub, branching, merging, displaying history, and pulling changes to local machine.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
 2. **GitHub Account** - Create at github.com
 3. **Text Editor** - VS Code, Notepad++, etc.
 4. **Terminal/Command Prompt**
-

Phase 1: Project Setup and Initial Commit

Step 1: Create Project Directory

bash

```
# Create and navigate to project directory
```

```
mkdir github-pull-demo
```

```
cd github-pull-demo
```

Step 2: Initialize Git Repository

bash

```
git init
```

Expected Output:

text

```
Initialized empty Git repository in C:/github-pull-demo/.git/
```

Step 3: Configure Git User

bash

```
git config user.name "Your Name"
```

```
git config user.email "your.email@example.com"
```

Step 4: Create Project Files

Create index.html:

html

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>GitHub Pull Demo</title>
    <style>
      body {
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        max-width: 900px;
        margin: 0 auto;
        padding: 20px;
        background: linear-gradient(135deg, #6e8efb, #a777e3);
        color: #333;
        line-height: 1.6;
      }
      .container {
        background: rgba(255, 255, 255, 0.95);
        padding: 30px;
        border-radius: 15px;
        box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
      }
      .header {
        text-align: center;
```

```
margin-bottom: 30px;  
}  
.feature-card {  
background: #f8f9fa;  
padding: 20px;  
margin: 15px 0;  
border-radius: 10px;  
border-left: 5px solid #6e8efb;  
}  
.git-command {  
background: #2d3748;  
color: #e2e8f0;  
padding: 10px;  
border-radius: 5px;  
font-family: 'Courier New', monospace;  
margin: 5px 0;  
}  
</style>  
</head>  
<body>  
<div class="container">  
<div class="header">  
<h1>  GitHub Pull Operations Demo</h1>  
<p>Demonstrating Git operations with GitHub remote repository and pull functionality</p>  
</div>  
  
<div class="feature-card">
```

```
<h2> 📁 Project Overview</h2>
<p>This project demonstrates the complete Git workflow including:</p>
<ul>
  <li>Repository initialization and setup</li>
  <li>Branch creation and management</li>
  <li>Merging branches</li>
  <li>Commit history tracking</li>
  <li>Pull operations from remote</li>
</ul>
</div>
```

```
<div class="feature-card">
  <h2> 🚀 Git Commands Demonstrated</h2>
  <div class="git-command">git init</div>
  <div class="git-command">git add .</div>
  <div class="git-command">git commit -m "message"</div>
  <div class="git-command">git branch</div>
  <div class="git-command">git merge</div>
  <div class="git-command">git push</div>
  <div class="git-command">git pull</div>
  <div class="git-command">git log --oneline --graph --all</div>
</div>
```

```
<div class="feature-card">
  <h2> 🚀 Current Status</h2>
  <p><strong>Branch:</strong> <span style="color: #6e8efb;">main</span></p>
  <p><strong>Commit:</strong> Initial project setup</p>
  <p><strong>Features:</strong> Basic structure and documentation</p>
```

```
</div>  
</div>  
</body>  
</html>  
Create README.md:  
markdown  
# GitHub Pull Demo Project
```

This project demonstrates complete Git and GitHub workflow including pull operations.

🎯 Learning Objectives

- Git repository initialization and management
- GitHub remote repository setup
- Branching and merging strategies
- Commit history management
- Pull operations from remote repository

📁 Project Structure

```
github-pull-demo/  
├── index.html # Main web application  
├── README.md # Project documentation  
├── .gitignore # Git ignore rules  
└── scripts/ # Additional scripts (future)
```

text

🛠 Workflow Steps

1. **Local Setup**

- Initialize Git repository
- Create project structure
- Make initial commit

2. **GitHub Integration**

- Create remote repository
- Push local code to GitHub
- Verify remote connection

3. **Branching Strategy**

- Create feature branch
- Develop features independently
- Merge back to main

4. **Pull Operations**

- Simulate team collaboration
- Pull latest changes
- Resolve conflicts (if any)

🛡 Technologies Used

- Git for version control
- GitHub for remote repository
- HTML/CSS for web interface
- Markdown for documentation

Create .gitignore:

text

Dependencies

node_modules/

vendor/

Environment variables

.env

.env.local

Logs

*.log

logs/

Build outputs

dist/

build/

*.exe

IDE

.vscode/

.idea/

*.swp

*.swo

OS

.DS_Store

Thumbs.db

Temporary files

*.tmp

*.temp

Step 5: Make Initial Commit

Check status:

bash

git status

Expected Output:

text

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

README.md

index.html

nothing added to commit but untracked files present (use "git add" to track)

Stage files and commit:

bash

git add .

git commit -m "Initial commit: Project setup with web interface and documentation"

Expected Output:

text

[main (root-commit) a1b2c3d] Initial commit: Project setup with web interface and documentation

3 files changed, 125 insertions(+)

create mode 100644 .gitignore

```
create mode 100644 README.md
```

```
create mode 100644 index.html
```

Phase 2: GitHub Repository Setup

Step 6: Create GitHub Repository

1. Go to github.com and log in
2. Click "+" icon → "New repository"
3. Repository details:
 - o **Repository name:** github-pull-demo
 - o **Description:** "Demo project for GitHub pull operations and branching"
 - o **Visibility: Public** (for easy access)
 - o **Initialize with README: No** (we have our own)
 - o **Add .gitignore: None** (we have our own)
4. Click "Create repository"

Step 7: Connect Local Repository to GitHub

Copy the repository URL from GitHub and run:

bash

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

Verify remote connection:

bash

```
git remote -v
```

Expected Output:

text

```
origin https://github.com/your-username/github-pull-demo.git (fetch)
```

```
origin https://github.com/your-username/github-pull-demo.git (push)
```

Step 8: Push to GitHub

First push (set upstream):

bash

```
git push -u origin main
```

 **Expected Output:**

text

Enumerating objects: 6, done.

Counting objects: 100% (6/6), done.

Delta compression using up to 8 threads

Compressing objects: 100% (5/5), done.

Writing objects: 100% (6/6), 2.15 KiB | 2.15 MiB/s, done.

Total 6 (delta 0), reused 0 (delta 0), pack-reused 0

To <https://github.com/your-username/github-pull-demo.git>

* [new branch] main -> main

Branch 'main' set up to track remote branch 'main' from 'origin'.

Step 9: Verify on GitHub

1. **Refresh your GitHub repository page**
 2. **You should see your files**
 3. **Click on "commits" to see the initial commit**
-

Phase 3: Create and Work on Feature Branch

Step 10: Create Feature Branch

Create and switch to new branch:

bash

```
git checkout -b feature-interactive-elements
```

 **Expected Output:**

text

Switched to a new branch 'feature-interactive-elements'

Verify current branch:

bash

```
git branch
```

Expected Output:

text

main

* feature-interactive-elements

Step 11: Make Changes in Feature Branch

Update index.html - add interactive elements:

html

```
<!-- Add this before closing body tag -->

<section style="margin: 30px 0;">

    <div class="feature-card" style="background: linear-gradient(135deg, #a777e3, #6e8efb); color: white;">

        <h2>🎮 Interactive Features (from Feature Branch)</h2>

        <p>These features were developed in the <strong>feature-interactive-elements</strong> branch</p>

        <div style="display: grid; grid-template-columns: 1fr 1fr; gap: 20px; margin-top: 20px;">

            <div style="background: rgba(255,255,255,0.1); padding: 15px; border-radius: 8px;">

                <h4>🔔 Real-time Updates</h4>

                <p>Dynamic content loading</p>

                <button onclick="showAlert()" style="background: white; color: #6e8efb; border: none; padding: 8px 16px; border-radius: 5px; cursor: pointer;">Test Alert</button>

            </div>

            <div style="background: rgba(255,255,255,0.1); padding: 15px; border-radius: 8px;">

                <h4>📊 Data Display</h4>

                <p>Interactive charts and graphs</p>

            </div>

        </div>

    </div>

</section>
```

```
<button onclick="updateTime()" style="background: white; color: #6e8efb; border: none; padding: 8px 16px; border-radius: 5px; cursor: pointer;">Show Time</button>

<p id="current-time"></p>

</div>
</div>
</div>
</div>

</section>

<section class="feature-card">

<h2> 🌱 Branch Information</h2>

<div class="git-command">git checkout -b feature-interactive-elements</div>
<div class="git-command">git add .</div>
<div class="git-command">git commit -m "feat: Add interactive elements"</div>
<div class="git-command">git push -u origin feature-interactive-elements</div>

<p><strong>Branch Status:</strong> Active development</p>
<p><strong>Changes:</strong> Added JavaScript interactivity and UI improvements</p>

</section>

<script>

function showAlert() {
    alert('🎉 Feature from interactive branch! This alert demonstrates JavaScript functionality.');
}

function updateTime() {
    const now = new Date();
}
```

```
    document.getElementById('current-time').textContent = 'Current time: ' +  
now.toLocaleTimeString();  
}
```

```
// Initialize some interactive features  
  
document.addEventListener('DOMContentLoaded', function() {  
  
    console.log('Interactive features loaded from feature branch');  
  
});  
  
</script>
```

Update README.md:

markdown

```
# GitHub Pull Demo Project
```

This project demonstrates complete Git and GitHub workflow including pull operations.

🎯 Learning Objectives

- Git repository initialization and management
- GitHub remote repository setup
- Branching and merging strategies
- Commit history management
- Pull operations from remote repository

📁 Project Structure

```
github-pull-demo/  
├── index.html # Main web application  
├── README.md # Project documentation  
├── .gitignore # Git ignore rules  
└── scripts/ # Additional scripts (future)
```

text

Workflow Steps

1. **Local Setup**

- Initialize Git repository
- Create project structure
- Make initial commit

2. **GitHub Integration**

- Create remote repository
- Push local code to GitHub
- Verify remote connection

3. **Branching Strategy**

- Create feature branch
- Develop features independently
- Merge back to main

4. **Pull Operations**

- Simulate team collaboration
- Pull latest changes
- Resolve conflicts (if any)

New Features Added

Interactive Elements Branch

- JavaScript functionality

- Real-time user interactions
- Dynamic content updates
- Enhanced UI components

🛡 Technologies Used

- Git for version control
- GitHub for remote repository
- HTML/CSS for web interface
- JavaScript for interactivity
- Markdown for documentation

Create styles.css (new file):

css

/ Additional styles from feature branch */*

```
.interactive-panel {  
  background: linear-gradient(135deg, #ff6b6b, #ee5a24);  
  color: white;  
  padding: 25px;  
  border-radius: 12px;  
  margin: 20px 0;  
  text-align: center;  
}
```

```
.btn-interactive {  
  background: rgba(255, 255, 255, 0.2);  
  color: white;  
  border: 2px solid white;  
  padding: 12px 24px;
```

```
border-radius: 25px;  
cursor: pointer;  
font-size: 16px;  
transition: all 0.3s ease;  
margin: 5px;  
}
```

```
.btn-interactive:hover {  
background: white;  
color: #ff6b6b;  
transform: translateY(-2px);  
}
```

```
.feature-grid {  
display: grid;  
grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));  
gap: 20px;  
margin: 20px 0;  
}
```

```
.feature-item {  
background: #f8f9fa;  
padding: 20px;  
border-radius: 10px;  
text-align: center;  
border: 2px solid transparent;  
transition: all 0.3s ease;  
}
```

```
.feature-item:hover {  
    border-color: #6e8efb;  
    transform: translateY(-5px);  
}
```

Step 12: Commit Changes in Feature Branch

Check status:

bash

git status

Expected Output:

text

On branch feature-interactive-elements

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: README.md

modified: index.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

styles.css

no changes added to commit (use "git add" and/or "git commit -a")

Stage and commit changes:

bash

git add .

git commit -m "feat: Add interactive JavaScript elements, new styles, and enhanced UI"

Expected Output:

text

```
[feature-interactive-elements e4f5g6h] feat: Add interactive JavaScript elements, new styles, and enhanced UI
```

```
3 files changed, 145 insertions(+), 5 deletions(-)
```

```
create mode 100644 styles.css
```

Step 13: Push Feature Branch to GitHub

bash

```
git push -u origin feature-interactive-elements
```

Expected Output:

text

```
Enumerating objects: 9, done.
```

```
Counting objects: 100% (9/9), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (7/7), done.
```

```
Writing objects: 100% (7/7), 2.45 KiB | 2.45 MiB/s, done.
```

```
Total 7 (delta 2), reused 0 (delta 0), pack-reused 0
```

```
remote:
```

```
remote: Create a pull request for 'feature-interactive-elements' on GitHub by visiting:
```

```
remote: https://github.com/your-username/github-pull-demo/pull/new/feature-interactive-elements
```

```
remote:
```

```
To https://github.com/your-username/github-pull-demo.git
```

```
* [new branch]  feature-interactive-elements -> feature-interactive-elements
```

```
Branch 'feature-interactive-elements' set up to track remote branch 'feature-interactive-elements' from 'origin'.
```

Phase 4: Merge Branch and Display History

Step 14: Switch Back to Main Branch

bash

```
git checkout main
```

 **Expected Output:**

text

```
Switched to branch 'main'
```

```
Your branch is up to date with 'origin/main'.
```

Step 15: Merge Feature Branch into Main

bash

```
git merge feature-interactive-elements
```

 **Expected Output:**

text

```
Updating a1b2c3d..e4f5g6h
```

```
Fast-forward
```

```
 README.md | 14 ++++++-----
```

```
 index.html | 95 ++++++-----
```

```
+-----
```

```
 styles.css | 41 ++++++-----
```

```
 3 files changed, 149 insertions(+), 1 deletion(-)
```

```
 create mode 100644 styles.css
```

Step 16: Push Merged Changes to GitHub

bash

```
git push origin main
```

 **Expected Output:**

text

```
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To https://github.com/your-username/github-pull-demo.git
```

```
 a1b2c3d..e4f5g6h main -> main
```

Step 17: Display Chronological Commit History

View detailed history:

```
bash
```

```
git log
```

 **Expected Output:**

text

```
commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD -> main, origin/main,  
origin/feature-interactive-elements, feature-interactive-elements)
```

```
Author: Your Name <your.email@example.com>
```

```
Date: Mon Nov 24 19:20:00 2023 +0530
```

```
feat: Add interactive JavaScript elements, new styles, and enhanced UI
```

```
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r
```

```
Author: Your Name <your.email@example.com>
```

```
Date: Mon Nov 24 19:10:00 2023 +0530
```

```
Initial commit: Project setup with web interface and documentation
```

One-line history with graph:

```
bash
```

```
git log --oneline --graph --all
```

 **Expected Output:**

text

```
* e4f5g6h (HEAD -> main, origin/main, origin/feature-interactive-elements, feature-  
interactive-elements) feat: Add interactive JavaScript elements, new styles, and  
enhanced UI
```

```
* a1b2c3d Initial commit: Project setup with web interface and documentation
```

Custom format history:

```
bash
```

```
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

 **Expected Output:**

text

e4f5g6h | Your Name | 2023-11-24 | feat: Add interactive JavaScript elements, new styles, and enhanced UI

a1b2c3d | Your Name | 2023-11-24 | Initial commit: Project setup with web interface and documentation

Phase 5: Simulate Team Collaboration and Pull Changes

Step 18: Simulate Changes on GitHub (Web Interface)

1. **Go to your GitHub repository**
2. **Click on** README.md → **Edit this file** (pencil icon)
3. **Add this section at the end:**

markdown

```
##  Team Collaboration Section
```

This section was added directly on GitHub to simulate team collaboration.

Remote Changes Made:

- Added team collaboration section
- Updated documentation
- Simulated multi-user workflow

Next Steps:

1. Pull these changes to local machine

2. Verify synchronization

3. Continue development

4. **Commit changes:**

- **Commit message:** "docs: Add team collaboration section via GitHub web"
- **Select:** "Commit directly to the main branch"

- Click "Commit changes"

Step 19: Check Local Status Before Pull

Check if local is behind remote:

bash

git status

 **Expected Output:**

text

On branch main

Your branch is up to date with 'origin/main'.

This might not show changes yet. Let's check explicitly:

bash

git fetch origin

git status

 **Expected Output:**

text

On branch main

Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.

(use "git pull" to update your local branch)

Step 20: Pull Changes from GitHub

Pull the changes:

bash

git pull origin main

 **Expected Output:**

text

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 1.01 KiB | 172.00 KiB/s, done.

From https://github.com/your-username/github-pull-demo

e4f5g6h..8901j2k main -> origin/main

Updating e4f5g6h..8901j2k

Fast-forward

README.md | 13 ++++++++
 1 file changed, 13 insertions(+)

Step 21: Verify Pulled Changes

Check the updated README:

bash

cat README.md

 **Expected Output:** Should show the new "Team Collaboration Section"

View updated commit history:

bash

git log --oneline --graph --all

 **Expected Output:**

text

* 8901j2k (HEAD -> main, origin/main) docs: Add team collaboration section via GitHub web

* e4f5g6h (origin/feature-interactive-elements, feature-interactive-elements) feat: Add interactive JavaScript elements, new styles, and enhanced UI

* a1b2c3d Initial commit: Project setup with web interface and documentation

Step 22: Make Local Changes and Push Back

Update index.html to acknowledge the pull:

html

<!-- Add this section before closing body tag -->

<section class="interactive-panel">

<h2> Successful Pull Operation!</h2>

```
<p>Changes from GitHub have been successfully pulled to local machine.</p>

<p><strong>Latest Update:</strong> Team collaboration section added via GitHub web interface</p>

<button class="btn-interactive" onclick="celebratePull()">🎉 Celebrate Pull!</button>

</section>

<script>

function celebratePull() {

    alert('✅ Success! Changes pulled from GitHub and local repository is synchronized!');

}

</script>
```

Commit and push local changes:

```
bash

git add index.html

git commit -m "feat: Add pull success confirmation and celebration element"

git push origin main
```

✅ Expected Output:

```
text

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 8 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 1.15 KiB | 1.15 MiB/s, done.

Total 3 (delta 2), reused 0 (delta 0), pack-reused 0

To https://github.com/your-username/github-pull-demo.git

8901j2k..5678k9l main -> main
```

Phase 6: Final History and Verification

Step 23: Display Final Chronological History

Complete history with all operations:

bash

```
git log --oneline --graph --all --decorate
```

Expected Output:

text

```
* 5678k9l (HEAD -> main, origin/main) feat: Add pull success confirmation and celebration element  
* 8901j2k docs: Add team collaboration section via GitHub web  
* e4f5g6h (origin/feature-interactive-elements, feature-interactive-elements) feat: Add interactive JavaScript elements, new styles, and enhanced UI  
* a1b2c3d Initial commit: Project setup with web interface and documentation
```

Detailed history view:

bash

```
git log --pretty=format:"%h %ad | %s" --date=short
```

Expected Output:

text

```
5678k9l 2023-11-24 | feat: Add pull success confirmation and celebration element
```

```
8901j2k 2023-11-24 | docs: Add team collaboration section via GitHub web
```

```
e4f5g6h 2023-11-24 | feat: Add interactive JavaScript elements, new styles, and enhanced UI
```

```
a1b2c3d 2023-11-24 | Initial commit: Project setup with web interface and documentation
```

Step 24: Verify All Operations on GitHub

1. **Go to your GitHub repository**
2. **Check "Commits"** - should show all 4 commits
3. **Check "Branches"** - should show both main and feature-interactive-elements
4. **Verify files** contain all the changes

Complete Workflow Summary

All Commands Used:

```
bash
```

```
# Setup and initial commit
```

```
mkdir github-pull-demo
```

```
cd github-pull-demo
```

```
git init
```

```
# Create files: index.html, README.md, .gitignore
```

```
git add .
```

```
git commit -m "Initial commit"
```

```
# GitHub setup
```

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

```
git push -u origin main
```

```
# Branching and development
```

```
git checkout -b feature-interactive-elements
```

```
# Make changes to files
```

```
git add .
```

```
git commit -m "feat: Add interactive features"
```

```
git push -u origin feature-interactive-elements
```

```
# Merging
```

```
git checkout main
```

```
git merge feature-interactive-elements
```

```
git push origin main
```

Pull operations

git fetch origin

git status

git pull origin main

History viewing

git log

git log --oneline --graph --all

git log --pretty=format:"%h | %an | %ad | %s" --date=short

Git Pull vs Git Fetch

Command	Purpose	Effect
git fetch	Downloads changes	Doesn't merge
git pull	Downloads + merges	Updates working directory
git pull --rebase	Downloads + rebases	Clean history

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.

GitLab Repository Management - Complete Step-by-Step Guide

I'll walk you through creating a project, pushing to GitLab, branching, merging, and displaying commit history.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
2. **GitLab Account** - Create at gitlab.com
3. **Text Editor** - VS Code, Notepad++, etc.
4. **Terminal/Command Prompt**



Phase 1: GitLab Account Setup

Step 1: Create GitLab Account (if you don't have one)

- 1. Go to gitlab.com**
- 2. Click "Register" or "Sign up"**
- 3. Choose registration method (Email, Google, GitHub, etc.)**
- 4. Verify your email address**
- 5. Complete your profile**

Step 2: Create SSH Key (Recommended for Secure Access)

Generate SSH key:

bash

```
ssh-keygen -t rsa -b 4096 -C "your.email@example.com"
```

 **Expected Output:**

text

Generating public/private rsa key pair.

Enter file in which to save the key
(/c/Users/YourName/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in
/c/Users/YourName/.ssh/id_rsa

Your public key has been saved in
/c/Users/YourName/.ssh/id_rsa.pub

Copy public key:

bash

On Windows

cat ~/.ssh/id_rsa.pub

On Linux/Mac

cat ~/.ssh/id_rsa.pub | clip # or pbcopy on Mac

Add SSH key to GitLab:

- 1. GitLab → User Settings → SSH Keys**
- 2. Paste your public key**
- 3. Add title (e.g., "My Laptop")**
- 4. Click "Add key"**



Phase 2: Create Local Project and Git Setup

Step 3: Create Project Directory

bash

Create and navigate to project directory

mkdir gitlab-demo-project

cd gitlab-demo-project

Step 4: Initialize Git Repository

bash

```
git init
```

 **Expected Output:**

text

```
Initialized empty Git repository in C:/gitlab-demo-project/.git/
```

Step 5: Configure Git User

bash

```
git config user.name "Your Name"
```

```
git config user.email "your.email@example.com"
```

Step 6: Create Project Files

Create index.html:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>GitLab CI/CD Demo</title>
```

```
<style>

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
    sans-serif;
    max-width: 1000px;
    margin: 0 auto;
    padding: 20px;
    background: linear-gradient(135deg, #fc6d26,
    #e24329);
    color: #333;
    line-height: 1.6;
}

.container {
    background: rgba(255, 255, 255, 0.95);
    padding: 30px;
    border-radius: 15px;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
}

.header {
    text-align: center;
    margin-bottom: 30px;
```

```
border-bottom: 3px solid #fc6d26;  
padding-bottom: 20px;  
}  
  
.feature-card {  
background: #f8f9fa;  
padding: 20px;  
margin: 15px 0;  
border-radius: 10px;  
border-left: 5px solid #fc6d26;  
}  
  
.gitlab-feature {  
background: linear-gradient(135deg, #e24329,  
#fc6d26);  
color: white;  
padding: 25px;  
border-radius: 10px;  
margin: 20px 0;  
}  
  
.command {  
background: #2d3748;  
color: #e2e8f0;
```

```
padding: 10px 15px;  
border-radius: 5px;  
font-family: 'Courier New', monospace;  
margin: 8px 0;  
border-left: 4px solid #fc6d26;  
}  
  
.branch-info {  
background: #e7f3ff;  
padding: 15px;  
border-radius: 8px;  
border: 2px dashed #3498db;  
margin: 15px 0;  
}  
  
</style>  
</head>  
  
<body>  
  <div class="container">  
    <div class="header">  
      <h1>⚠️ GitLab Repository Demo</h1>  
      <p>Demonstrating Git operations with GitLab  
remote repository and CI/CD capabilities</p>
```

```
</div>
```

```
<div class="gitlab-feature">
```

```
  <h2>🚀 GitLab Features Showcase</h2>
```

```
  <p>GitLab provides comprehensive DevOps  
platform with built-in CI/CD, issue tracking, and repository  
management.</p>
```

```
</div>
```

```
<div class="feature-card">
```

```
  <h2>📋 Project Overview</h2>
```

```
  <p>This project demonstrates complete GitLab  
workflow including:</p>
```

```
  <ul>
```

```
    <li>Repository creation and management</li>
```

```
    <li>Branch creation and merging</li>
```

```
    <li>Commit history tracking</li>
```

```
    <li>GitLab CI/CD integration (potential)</li>
```

```
    <li>Merge requests workflow</li>
```

```
  </ul>
```

```
</div>
```

```
<div class="feature-card">

  <h2>🔧 Git Commands Demonstrated</h2>

  <div class="command">git init</div>

  <div class="command">git add .</div>

  <div class="command">git commit -m
"message"</div>

  <div class="command">git branch</div>

  <div class="command">git checkout -b feature-
branch</div>

  <div class="command">git merge</div>

  <div class="command">git push -u origin main</div>

  <div class="command">git log --oneline --graph --
all</div>

</div>

<div class="branch-info">

  <h2>🌿 Current Branch Information</h2>

  <p><strong>Branch:</strong> <span style="color:
#fc6d26;">main</span></p>

  <p><strong>Status:</strong> Initial project
setup</p>
```

<p>Next Steps: Push to GitLab and create feature branches</p>

</div>

<div class="feature-card">

<h2> 🚀 GitLab vs GitHub</h2>

GitLab: Built-in CI/CD, free private repos, comprehensive DevOps

GitHub: Larger community, GitHub Actions, extensive integrations

Both: Excellent Git repository hosting with collaboration features

</div>

</div>

</body>

</html>

Create README.md:

markdown

GitLab Demo Project

This project demonstrates Git and GitLab operations including repository management, branching, and merging.

Learning Objectives

- Git repository initialization and management
- GitLab remote repository setup
- Branching strategies and merge operations
- Commit history management
- GitLab CI/CD fundamentals

Project Structure

```
gitlab-demo-project/  
    ├── index.html # Main web application  
    ├── README.md # Project documentation  
    ├── .gitignore # Git ignore rules  
    └── assets/ # Additional resources (future)
```

text

GitLab Workflow

1. **Local Development**

- Initialize Git repository
- Create project structure
- Make initial commits

2. **GitLab Integration**

- Create remote repository on GitLab
- Push local code to GitLab
- Configure repository settings

3. **Branching Strategy**

- Create feature branches
- Develop features independently
- Merge back to main branch

4. **CI/CD Pipeline** (Optional)

- Configure .gitlab-ci.yml
- Automated testing and deployment
- Continuous integration

Technologies Used

- Git for version control
- GitLab for remote repository and DevOps
- HTML/CSS for web interface
- Markdown for documentation

Git Commands Coverage

- Basic: init, add, commit, status
- Branching: branch, checkout, merge
- Remote: remote, push, pull
- History: log, show, diff

Create .gitignore:

text

```
# Dependencies
```

```
node_modules/
```

```
vendor/
```

```
packages/
```

```
# Environment
```

```
.env
```

.env.local

.env.production

Logs

*.log

logs/

npm-debug.log*

Build outputs

dist/

build/

out/

*.exe

IDE

.vscode/

.idea/

*.swp

*.swo

OS

.DS_Store

Thumbs.db

GitLab CI

.cache/

Create CHANGELOG.md:

markdown

Changelog

All notable changes to this project will be documented in this file.

[1.0.0] - 2023-11-24

Added

- Initial project structure
- Basic HTML web interface
- Project documentation
- Git ignore rules
- Changelog file

Step 7: Make Initial Commit

Check status:

bash

git status

Expected Output:

text

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

CHANGELOG.md

README.md

index.html

nothing added to commit but untracked files present (use "git add" to track)

Stage files and commit:

bash

```
git add .
```

```
git commit -m "Initial commit: Project setup with web interface, documentation, and GitLab-focused content"
```

Expected Output:

text

```
[main (root-commit) a1b2c3d] Initial commit: Project setup with web interface, documentation, and GitLab-focused content
```

```
4 files changed, 185 insertions(+)
```

```
create mode 100644 .gitignore
```

```
create mode 100644 CHANGELOG.md
```

```
create mode 100644 README.md
```

```
create mode 100644 index.html
```



Phase 3: GitLab Repository Setup

Step 8: Create GitLab Repository

- 1. Log in to [gitlab.com](#)**

- 2. Click "+" icon → "New project"**

- 3. Choose "Create blank project"**

- 4. Project details:**

- Project name:** gitlab-demo-project

- **Project slug:** gitlab-demo-project (auto-filled)
- **Visibility Level: Private** (for security)
- **Initialize repository with README: No** (we have our own)

5. Click "Create project"

Step 9: Connect Local Repository to GitLab

Copy the repository URL from GitLab (use SSH or HTTPS):

For HTTPS:

bash

```
git remote add origin https://gitlab.com/your-  
username/gitlab-demo-project.git
```

For SSH (recommended):

bash

```
git remote add origin git@gitlab.com:your-  
username/gitlab-demo-project.git
```

Verify remote connection:

bash

```
git remote -v
```

Expected Output:

text

```
origin git@gitlab.com:your-username/gitlab-demo-project.git (fetch)
```

```
origin git@gitlab.com:your-username/gitlab-demo-project.git (push)
```

Step 10: Push to GitLab

First push (set upstream):

bash

```
git push -u origin main
```

Expected Output:

text

```
Enumerating objects: 7, done.
```

```
Counting objects: 100% (7/7), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (6/6), done.
```

```
Writing objects: 100% (7/7), 2.78 KiB | 2.78 MiB/s, done.
```

```
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To gitlab.com:your-username/gitlab-demo-project.git
```

```
* [new branch] main -> main
```

```
Branch 'main' set up to track remote branch 'main' from  
'origin'.
```

Step 11: Verify on GitLab

- 1. Refresh your GitLab project page**
 - 2. You should see your files**
 - 3. Click on "Repository" → "Commits" to see the initial commit**
 - 4. Explore the GitLab interface (Issues, Merge Requests, CI/CD)**
-



Phase 4: Create and Work on Feature Branch

Step 12: Create Feature Branch

Create and switch to new branch:

bash

```
git checkout -b feature-ci-cd-pipeline
```



Expected Output:

text

```
Switched to a new branch 'feature-ci-cd-pipeline'
```

Verify current branch:

bash

```
git branch
```



Expected Output:

text

```
main
```

* feature-ci-cd-pipeline

Step 13: Make Changes in Feature Branch

Update index.html - add CI/CD section:

html

<!-- Add this before closing body tag -->

```
<section style="margin: 30px 0;">
```

```
    <div class="gitlab-feature">
```

```
        <h2>⚙️ GitLab CI/CD Pipeline Features</h2>
```

```
        <p>This section demonstrates features developed in  
the <strong>feature-ci-cd-pipeline</strong> branch</p>
```

```
<div style="display: grid; grid-template-columns: 1fr  
1fr; gap: 20px; margin-top: 20px;">
```

```
    <div style="background: rgba(255,255,255,0.1);  
padding: 20px; border-radius: 8px;">
```

```
        <h3>🔍 Code Quality</h3>
```

```
        <ul>
```

```
            <li>Automated testing</li>
```

```
            <li>Code coverage reports</li>
```

```
            <li>Security scanning</li>
```

```
            <li>Performance testing</li>
```

```
</ul>

</div>

<div style="background: rgba(255,255,255,0.1); padding: 20px; border-radius: 8px;">

    <h3>🚀 Deployment</h3>

    <ul>

        <li>Continuous integration</li>
        <li>Continuous deployment</li>
        <li>Environment management</li>
        <li>Rollback capabilities</li>

    </ul>

</div>

</div>

</div>

</section>

<section class="branch-info">

    <h2>🌿 Feature Branch: feature-ci-cd-pipeline</h2>

    <div class="command">git checkout -b feature-ci-cd-pipeline</div>
```

```
<div class="command"># Developing CI/CD  
features...</div>  
  
<div class="command">git add .</div>  
  
<div class="command">git commit -m "feat: Add CI/CD  
pipeline features"</div>  
  
<div class="command">git push -u origin feature-ci-cd-  
pipeline</div>
```

<p>Branch Status: Active
development</p>

<p>Changes: Added CI/CD pipeline
documentation and features</p>

</section>

<!-- Add GitLab CI/CD example -->

```
<section class="feature-card">  
  <h2>  Example .gitlab-ci.yml</h2>  
  <pre style="background: #2d3748; color: #e2e8f0;  
padding: 15px; border-radius: 5px; overflow-x: auto;">  
    stages:  
      - test  
      - build
```

- deploy

unit_tests:

stage: test

script:

- echo "Running unit tests..."

- npm test

build_project:

stage: build

script:

- echo "Building project..."

- npm run build

deploy_staging:

stage: deploy

script:

- echo "Deploying to staging..."

- npm run deploy:staging

only:

- main</pre>

</section>

Update README.md:

markdown

```
# GitLab Demo Project
```

This project demonstrates Git and GitLab operations including repository management, branching, and merging.

Learning Objectives

- Git repository initialization and management
- GitLab remote repository setup
- Branching strategies and merge operations
- Commit history management
- GitLab CI/CD fundamentals

Project Structure

```
gitlab-demo-project/
```

```
    ├── index.html # Main web application
    ├── README.md # Project documentation
    └── .gitignore # Git ignore rules
```

```
└── CHANGELOG.md # Version history  
└── assets/ # Additional resources (future)
```

text

GitLab Workflow

1. **Local Development**

- Initialize Git repository
- Create project structure
- Make initial commits

2. **GitLab Integration**

- Create remote repository on GitLab
- Push local code to GitLab
- Configure repository settings

3. **Branching Strategy**

- Create feature branches
- Develop features independently
- Merge back to main branch

4. **CI/CD Pipeline** (Optional)

- Configure .gitlab-ci.yml
- Automated testing and deployment
- Continuous integration

CI/CD Features Added

GitLab CI/CD Pipeline

- Automated testing stages
- Build and deployment processes
- Environment management
- Code quality checks

Technologies Used

- Git for version control
- GitLab for remote repository and DevOps
- HTML/CSS for web interface
- Markdown for documentation
- GitLab CI/CD for automation

Git Commands Coverage

- Basic: init, add, commit, status
- Branching: branch, checkout, merge
- Remote: remote, push, pull
- History: log, show, diff

Update CHANGELOG.md:

markdown

```
# Changelog
```

All notable changes to this project will be documented in this file.

```
## [Unreleased]
```

Added

- CI/CD pipeline documentation
- GitLab-specific features section
- Enhanced project documentation

```
## [1.0.0] - 2023-11-24
```

Added

- Initial project structure
- Basic HTML web interface
- Project documentation
- Git ignore rules
- Changelog file

Create .gitlab-ci.yml (GitLab CI/CD configuration):

yaml

```
# GitLab CI/CD Pipeline Configuration
```

```
image: node:16
```

```
stages:
```

- test
- build
- deploy

```
cache:
```

```
paths:
```

- node_modules/

```
before_script:  
  - echo "Starting CI/CD pipeline for GitLab Demo Project"
```

```
unit_tests:  
  stage: test  
  script:  
    - echo "Running unit tests..."  
    - echo "No tests configured yet - this is a demo"
```

```
lint_code:  
  stage: test  
  script:  
    - echo "Linting code..."  
    - echo "HTML and CSS linting would happen here"
```

```
build_project:  
  stage: build  
  script:  
    - echo "Building project..."  
    - echo "Project built successfully"  
artifacts:
```

paths:

- build/

deploy_staging:

stage: deploy

script:

- echo "Deploying to staging environment..."
- echo "Staging deployment complete"

environment:

name: staging

url: https://staging.example.com

only:

- main

pages:

stage: deploy

script:

- echo "Deploying to GitLab Pages..."
- mkdir public
- cp index.html public/
- cp -r assets/ public/ 2>/dev/null || true

artifacts:

paths:

- public

only:

- main

Step 14: Commit Changes in Feature Branch

Check status:

bash

git status

Expected Output:

text

On branch feature-ci-cd-pipeline

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: CHANGELOG.md

modified: README.md

modified: index.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitlab-ci.yml

no changes added to commit (use "git add" and/or "git commit -a")

Stage and commit changes:

bash

git add .

git commit -m "feat: Add GitLab CI/CD pipeline configuration, documentation, and CI/CD features section"

Expected Output:

text

[feature-ci-cd-pipeline e4f5g6h] feat: Add GitLab CI/CD pipeline configuration, documentation, and CI/CD features section

4 files changed, 145 insertions(+), 5 deletions(-)

create mode 100644 .gitlab-ci.yml

Step 15: Push Feature Branch to GitLab

bash

git push -u origin feature-ci-cd-pipeline

 **Expected Output:**

text

Enumerating objects: 9, done.

Counting objects: 100% (9/9), done.

Delta compression using up to 8 threads

Compressing objects: 100% (8/8), done.

Writing objects: 100% (8/8), 2.89 KiB | 2.89 MiB/s, done.

Total 8 (delta 3), reused 0 (delta 0), pack-reused 0

remote:

remote: To create a merge request for feature-ci-cd-pipeline, visit:

remote: https://gitlab.com/your-username/gitlab-demo-project/-/merge_requests/new?merge_request%5Bsource_branch%5D=feature-ci-cd-pipeline

remote:

To gitlab.com:your-username/gitlab-demo-project.git

* [new branch] feature-ci-cd-pipeline -> feature-ci-cd-pipeline

Branch 'feature-ci-cd-pipeline' set up to track remote branch 'feature-ci-cd-pipeline' from 'origin'.



Phase 5: Merge Branch and Display History

Step 16: Switch Back to Main Branch

bash

```
git checkout main
```



Expected Output:

text

```
Switched to branch 'main'
```

```
Your branch is up to date with 'origin/main'.
```

Step 17: Merge Feature Branch into Main

bash

```
git merge feature-ci-ci-pipeline
```



Expected Output:

text

```
Updating a1b2c3d..e4f5g6h
```

```
Fast-forward
```

```
.gitlab-ci.yml | 44
```

```
+++++-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
CHANGELOG.md | 7 +++++++
```

```
README.md | 14 ++++++++-
```

```
index.html | 80
```

```
+++++
```

```
+++++
```

```
4 files changed, 144 insertions(+), 1 deletion(-)
```

```
create mode 100644 .gitlab-ci.yml
```

Step 18: Push Merged Changes to GitLab

bash

```
git push origin main
```

Expected Output:

text

```
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To gitlab.com:your-username/gitlab-demo-project.git
```

```
a1b2c3d..e4f5g6h main -> main
```

Step 19: Display Chronological Commit History

View detailed history:

bash

```
git log
```

Expected Output:

text

commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD -> main, origin/main, origin/feature-ci-cd-pipeline, feature-ci-cd-pipeline)

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 20:20:00 2023 +0530

feat: Add GitLab CI/CD pipeline configuration, documentation, and CI/CD features section

commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 20:10:00 2023 +0530

Initial commit: Project setup with web interface, documentation, and GitLab-focused content

One-line history with graph:

bash

```
git log --oneline --graph --all
```

✓ Expected Output:

text

```
* e4f5g6h (HEAD -> main, origin/main, origin/feature-ci-cd-pipeline, feature-ci-cd-pipeline) feat: Add GitLab
```

CI/CD pipeline configuration, documentation, and CI/CD features section

* a1b2c3d Initial commit: Project setup with web interface, documentation, and GitLab-focused content

Custom format history:

bash

```
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

Expected Output:

text

e4f5g6h | Your Name | 2023-11-24 | feat: Add GitLab CI/CD pipeline configuration, documentation, and CI/CD features section

a1b2c3d | Your Name | 2023-11-24 | Initial commit: Project setup with web interface, documentation, and GitLab-focused content

Show file statistics:

bash

```
git log --stat
```

Expected Output:

text

commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD -> main, origin/main, origin/feature-ci-cd-pipeline, feature-ci-cd-pipeline)

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 20:20:00 2023 +0530

feat: Add GitLab CI/CD pipeline configuration, documentation, and CI/CD features section

.gitlab-ci.yml | 44

+++++

CHANGELOG.md | 7 ++++++

README.md | 14 ++++++++-

index.html | 80

+++++

+++++

4 files changed, 144 insertions(+), 1 deletion(-)

commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 20:10:00 2023 +0530

Initial commit: Project setup with web interface, documentation, and GitLab-focused content

```
.gitignore | 24 ++++++  
CHANGELOG.md | 10 +++++++  
README.md | 54 ++++++  
index.html | 97 ++++++  
4 files changed, 185 insertions(+)
```



Phase 6: GitLab Web Interface Operations

Step 20: Verify on GitLab Web Interface

1. **Go to your GitLab project**
2. **Click "Repository" → "Files"** - Verify all files are present
3. **Click "Repository" → "Commits"** - See chronological commit history
4. **Click "Repository" → "Branches"** - See both main and feature-ci-cd-pipeline branches

Step 21: Explore GitLab CI/CD Pipeline (Optional)

- 1. Go to "CI/CD" → "Pipelines"**
- 2. You should see a pipeline running (triggered by the push)**
- 3. Click on the pipeline to see the stages**
- 4. Monitor the pipeline execution**

Step 22: Create Merge Request (GitLab's equivalent of Pull Request)

In GitLab:

- 1. Go to "Merge Requests"**
- 2. Click "New merge request"**
- 3. Source branch: feature-ci-cd-pipeline**
- 4. Target branch: main**
- 5. Compare branches and continue**
- 6. Add details:**
 - Title:** "Add GitLab CI/CD pipeline configuration"
 - Description:** "This merge request adds CI/CD pipeline configuration and enhances documentation"
- 7. Click "Create merge request"**

Note: Since we already merged locally, this is for demonstration purposes.

Phase 7: Additional Branch Operations

Step 23: Create Another Feature Branch

Create documentation branch:

bash

```
git checkout -b docs-update-readme
```

Update README.md - add GitLab specific section:

markdown

```
## 🐱 GitLab Specific Features
```

Built-in DevOps Platform

- **CI/CD**: Built-in continuous integration and deployment
- **Container Registry**: Docker image storage
- **Package Registry**: npm, Maven, PyPI packages
- **Wiki**: Project documentation
- **Issue Tracking**: Built-in issue management

Security Features

- **SAST**: Static Application Security Testing
- **DAST**: Dynamic Application Security Testing
- **Dependency Scanning**: Vulnerability detection

- **License Compliance**: Open source license management

Collaboration Tools

- **Merge Requests**: Code review and discussion
- **Web IDE**: Browser-based code editing
- **Snippets**: Code sharing
- **Value Stream Analytics**: Process optimization

Commit and push:

bash

```
git add README.md
```

```
git commit -m "docs: Add GitLab specific features and platform capabilities"
```

```
git push -u origin docs-update-readme
```

Step 24: Merge and View Final History

bash

```
git checkout main
```

```
git merge docs-update-readme
```

```
git push origin main
```

View final chronological history:

bash

```
git log --oneline --graph --all
```

 **Expected Output:**

text

```
* 5678k9l (HEAD -> main, origin/main) Merge branch  
'docs-update-readme'  
  
|\  
| * 8901j2k (origin/docs-update-readme, docs-update-  
readme) docs: Add GitLab specific features and platform  
capabilities  
  
|/  
* e4f5g6h (origin/feature-ci-cd-pipeline, feature-ci-cd-  
pipeline) feat: Add GitLab CI/CD pipeline configuration,  
documentation, and CI/CD features section  
  
* a1b2c3d Initial commit: Project setup with web  
interface, documentation, and GitLab-focused content
```

Detailed final history:

bash

```
git log --pretty=format:"%h %ad | %s" --date=short
```

 **Expected Output:**

text

```
5678k9l 2023-11-24 | Merge branch 'docs-update-readme'
```

8901j2k 2023-11-24 | docs: Add GitLab specific features and platform capabilities

e4f5g6h 2023-11-24 | feat: Add GitLab CI/CD pipeline configuration, documentation, and CI/CD features section

a1b2c3d 2023-11-24 | Initial commit: Project setup with web interface, documentation, and GitLab-focused content

Complete Workflow Summary

All Commands Used:

bash

Setup and initial commit

mkdir gitlab-demo-project

cd gitlab-demo-project

git init

Create files: index.html, README.md, .gitignore, CHANGELOG.md

git add .

git commit -m "Initial commit"

GitLab setup

```
git remote add origin git@gitlab.com:your-  
username/gitlab-demo-project.git
```

```
git push -u origin main
```

Branching and development

```
git checkout -b feature-ci-cd-pipeline
```

Make changes to files

```
git add .
```

```
git commit -m "feat: Add CI/CD features"
```

```
git push -u origin feature-ci-cd-pipeline
```

Merging

```
git checkout main
```

```
git merge feature-ci-cd-pipeline
```

```
git push origin main
```

History viewing

```
git log
```

```
git log --oneline --graph --all
```

```
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

```
git log --stat
```

GitLab vs GitHub vs BitBucket

Feature	GitLab	GitHub	BitBucket
CI/CD	Built-in	Actions	Pipelines
Private Repos	Unlimited	Limited	Unlimited
Container Registry	Yes	Yes	No
Free Tier	Generous	Limited	Generous
Self-hosting	Free	Enterprise	Data Center

Q2) 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.

BitBucket Repository Management with Pull - Complete Step-by-Step Guide

I'll walk you through creating a project, pushing to Bitbucket, branching, merging, displaying history, and pulling changes to local machine.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
 2. **Bitbucket Account** - Create at bitbucket.org
 3. **Text Editor** - VS Code, Notepad++, etc.
 4. **Terminal/Command Prompt**
-



Phase 1: Bitbucket Account Setup

Step 1: Create Bitbucket Account (if you don't have one)

1. Go to bitbucket.org

- 2. Click "Get started for free"**
- 3. Sign up with email or Google account**
- 4. Verify your email address**

Step 2: Create SSH Key (Recommended for Secure Access)

Generate SSH key:

bash

```
ssh-keygen -t rsa -b 4096 -C "your.email@example.com"
```

 **Expected Output:**

text

Generating public/private rsa key pair.

Enter file in which to save the key
(/c/Users/YourName/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in
/c/Users/YourName/.ssh/id_rsa

Your public key has been saved in
/c/Users/YourName/.ssh/id_rsa.pub

Copy public key:

bash

On Windows

```
cat ~/.ssh/id_rsa.pub
```

On Linux/Mac

```
cat ~/.ssh/id_rsa.pub | clip # or pbcopy on Mac
```

Add SSH key to Bitbucket:

1. Bitbucket → Personal settings → SSH keys
 2. Click "Add key"
 3. Paste your public key
 4. Click "Save"
-



Phase 2: Create Local Project and Git Setup

Step 3: Create Project Directory

bash

```
# Create and navigate to project directory
```

```
mkdir bitbucket-pull-demo
```

```
cd bitbucket-pull-demo
```

Step 4: Initialize Git Repository

bash

```
git init
```



Expected Output:

text

Initialized empty Git repository in C:/bitbucket-pull-demo/.git/

Step 5: Configure Git User

bash

```
git config user.name "Your Name"
```

```
git config user.email "your.email@example.com"
```

Step 6: Create Project Files

Create index.html:

html

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Bitbucket Pull Demo</title>
    <style>
      body {
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
        max-width: 1000px;
```

```
margin: 0 auto;  
padding: 20px;  
background: linear-gradient(135deg, #0052cc,  
#00a3ff);  
color: #333;  
line-height: 1.6;  
}  
.container {  
background: rgba(255, 255, 255, 0.95);  
padding: 30px;  
border-radius: 15px;  
box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);  
}  
.header {  
text-align: center;  
margin-bottom: 30px;  
border-bottom: 3px solid #0052cc;  
padding-bottom: 20px;  
}  
.feature-card {  
background: #f8f9fa;
```

```
padding: 20px;  
margin: 15px 0;  
border-radius: 10px;  
border-left: 5px solid #0052cc;  
}  
  
.bitbucket-feature {  
background: linear-gradient(135deg, #0052cc,  
#00a3ff);  
color: white;  
padding: 25px;  
border-radius: 10px;  
margin: 20px 0;  
}  
  
.command {  
background: #2d3748;  
color: #e2e8f0;  
padding: 10px 15px;  
border-radius: 5px;  
font-family: 'Courier New', monospace;  
margin: 8px 0;  
border-left: 4px solid #0052cc;
```

```
}

.pull-demo {
    background: #e7f3ff;
    padding: 20px;
    border-radius: 8px;
    border: 2px dashed #3498db;
    margin: 20px 0;
}

.status-indicator {
    display: inline-block;
    padding: 5px 15px;
    border-radius: 20px;
    font-weight: bold;
    margin: 5px;
}

.status-synced { background: #28a745; color: white; }
.status-behind { background: #ffc107; color: black; }
.status-ahead { background: #17a2b8; color: white; }

</style>

</head>

<body>
```

```
<div class="container">  
  <div class="header">  
    <h1>  Bitbucket Pull Operations Demo</h1>  
    <p>Demonstrating Git operations with Bitbucket  
    remote repository and pull functionality</p>  
  </div>  
  
  
  
  <div class="bitbucket-feature">  
    <h2>  Atlassian Bitbucket Features</h2>  
    <p>Bitbucket provides Git repository management  
    with built-in CI/CD, Jira integration, and team  
    collaboration tools.</p>  
  </div>  
  
  
  
  <div class="feature-card">  
    <h2>  Project Overview</h2>  
    <p>This project demonstrates complete Bitbucket  
    workflow including:</p>  
    <ul>  
      <li>Repository creation and management</li>  
      <li>Branch creation and merging</li>  
    </ul>  
  </div>
```

```
</li>Commit history tracking</li>  
</li>Pull operations from remote</li>  
</li>Team collaboration simulation</li>  
</ul>  
</div>
```

```
<div class="feature-card">  
  <h2>🔧 Git Commands Demonstrated</h2>  
  <div class="command">git init</div>  
  <div class="command">git add .</div>  
  <div class="command">git commit -m  
  "message"</div>  
  <div class="command">git branch</div>  
  <div class="command">git checkout -b feature-  
branch</div>  
  <div class="command">git merge</div>  
  <div class="command">git push -u origin main</div>  
  <div class="command">git pull origin main</div>  
  <div class="command">git log --oneline --graph --  
all</div>  
</div>
```

```
<div class="pull-demo">  
  <h2>  Pull Operation Status</h2>  
  <p><strong>Current Status:</strong> <span  
    class="status-indicator status-synced">Local and Remote  
    Synced</span></p>  
  <p><strong>Next Step:</strong> Push to Bitbucket  
    and create feature branches</p>  
  <p><strong>Pull Command:</strong> <code>git pull  
    origin main</code></p>  
</div>
```

```
<div class="feature-card">  
  <h2>  Bitbucket Advantages</h2>  
  <ul>  
    <li><strong>Free Private Repositories:</strong>  
      Unlimited private repos for small teams</li>  
    <li><strong>Jira Integration:</strong> Seamless  
      integration with Jira for issue tracking</li>  
    <li><strong>Bitbucket Pipelines:</strong> Built-in  
      CI/CD with free minutes</li>
```

```
<li><strong>Branch Permissions:</strong> Fine-  
grained access control</li>  
  
<li><strong>Smart Mirroring:</strong> Improved  
performance for distributed teams</li>  
  
</ul>  
  
</div>  
  
</div>  
  
</body>  
  
</html>
```

Create README.md:

markdown

```
# Bitbucket Pull Demo Project
```

This project demonstrates complete Git and Bitbucket workflow including pull operations.

Learning Objectives

- Git repository initialization and management
- Bitbucket remote repository setup
- Branching strategies and merge operations

- Commit history management
- Pull operations from remote repository
- Team collaboration simulation

Project Structure

bitbucket-pull-demo/

```
|── index.html # Main web application  
|── README.md # Project documentation  
|── .gitignore # Git ignore rules  
|── CHANGELOG.md # Version history  
└── assets/ # Additional resources (future)
```

text

Bitbucket Workflow

1. **Local Setup**

- Initialize Git repository
- Create project structure
- Make initial commits

2. **Bitbucket Integration**

- Create remote repository on Bitbucket

- Push local code to Bitbucket
- Configure repository settings

3. **Branching Strategy**

- Create feature branches
- Develop features independently
- Merge back to main branch

4. **Pull Operations**

- Simulate team collaboration
- Pull latest changes from Bitbucket
- Handle synchronization

5. **Bitbucket Features**

- Pull requests and code review
- Pipelines for CI/CD
- Integration with Jira
- Access control and permissions

🔧 Technologies Used

- Git for version control
- Bitbucket for remote repository
- HTML/CSS for web interface
- Markdown for documentation

Git Commands Coverage

- Basic: init, add, commit, status
- Branching: branch, checkout, merge
- Remote: remote, push, pull, fetch
- History: log, show, diff, --graph

Create .gitignore:

text

```
# Dependencies
```

```
node_modules/
```

```
vendor/
```

```
packages/
```

```
# Environment
```

```
.env
```

```
.env.local
```

.env.production

Logs

*.log

logs/

npm-debug.log*

Build outputs

dist/

build/

out/

*.exe

IDE

.vscode/

.idea/

*.swp

*.swo

OS

.DS_Store

Thumbs.db

Bitbucket Pipelines

.bitbucket/

Create CHANGELOG.md:

markdown

Changelog

All notable changes to this project will be documented in this file.

[1.0.0] - 2023-11-24

Added

- Initial project structure
- Basic HTML web interface
- Project documentation
- Git ignore rules
- Changelog file

Step 7: Make Initial Commit

Check status:

bash

git status

 **Expected Output:**

text

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

CHANGELOG.md

README.md

index.html

nothing added to commit but untracked files present (use "git add" to track)

Stage files and commit:

bash

git add .

```
git commit -m "Initial commit: Project setup with web interface, documentation, and Bitbucket-focused content"
```

Expected Output:

text

```
[main (root-commit) a1b2c3d] Initial commit: Project setup with web interface, documentation, and Bitbucket-focused content
```

```
4 files changed, 215 insertions(+)
```

```
create mode 100644 .gitignore
```

```
create mode 100644 CHANGELOG.md
```

```
create mode 100644 README.md
```

```
create mode 100644 index.html
```



Phase 3: Bitbucket Repository Setup

Step 8: Create Bitbucket Repository

- 1. Log in to [bitbucket.org](#)**

- 2. Click "Create" → "Repository"**

- 3. Repository details:**

- **Repository name:** bitbucket-pull-demo
- **Description:** "Demo project for Bitbucket pull operations and branching"

- **Visibility: Private** (for security)
- **Include README: No** (we have our own)
- **Git ignore: None** (we have our own)

4. Click "Create repository"

Step 9: Connect Local Repository to Bitbucket

Copy the repository URL from Bitbucket (use SSH or HTTPS):

For HTTPS:

bash

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

For SSH (recommended):

bash

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

Verify remote connection:

bash

```
git remote -v
```

Expected Output:

text

origin git@bitbucket.org:your-username/bitbucket-pull-demo.git (fetch)

origin git@bitbucket.org:your-username/bitbucket-pull-demo.git (push)

Step 10: Push to Bitbucket

First push (set upstream):

bash

```
git push -u origin main
```

Expected Output:

text

Enumerating objects: 7, done.

Counting objects: 100% (7/7), done.

Delta compression using up to 8 threads

Compressing objects: 100% (6/6), done.

Writing objects: 100% (7/7), 2.95 KiB | 2.95 MiB/s, done.

Total 7 (delta 0), reused 0 (delta 0), pack-reused 0

To bitbucket.org:your-username/bitbucket-pull-demo.git

* [new branch] main -> main

Branch 'main' set up to track remote branch 'main' from 'origin'.

Step 11: Verify on Bitbucket

- 1. Refresh your Bitbucket repository page**
 - 2. You should see your files**
 - 3. Click on "Commits" to see the initial commit**
 - 4. Explore the Bitbucket interface (Pull requests, Pipelines, Settings)**
-



Phase 4: Create and Work on Feature Branch

Step 12: Create Feature Branch

Create and switch to new branch:

bash

```
git checkout -b feature-pull-demonstration
```



Expected Output:

text

```
Switched to a new branch 'feature-pull-demonstration'
```

Verify current branch:

bash

```
git branch
```



Expected Output:

text

main

```
* feature-pull-demonstration
```

Step 13: Make Changes in Feature Branch

Update index.html - add pull demonstration section:

html

```
<!-- Add this before closing body tag -->

<section style="margin: 30px 0;">

    <div class="bitbucket-feature">

        <h2>  Pull Operation Demonstration</h2>

        <p>This section demonstrates features developed in
the <strong>feature-pull-demonstration</strong>
branch</p>

        <div style="display: grid; grid-template-columns: 1fr
1fr; gap: 20px; margin-top: 20px;">

            <div style="background: rgba(255,255,255,0.1);
padding: 20px; border-radius: 8px;">

                <h3>  Sync Operations</h3>

                <ul>

                    <li>Pull latest changes</li>
                    <li>Fetch and merge</li>
                    <li>Conflict resolution</li>
                    <li>Remote tracking</li>

                </ul>
            </div>
        </div>
    </div>
</section>
```

```
</ul>

</div>

<div style="background: rgba(255,255,255,0.1); padding: 20px; border-radius: 8px;">

    <h3>  Team Collaboration</h3>

    <ul>

        <li>Multiple contributors</li>
        <li>Branch protection</li>
        <li>Code review workflow</li>
        <li>Merge strategies</li>
    </ul>

</div>

</div>

</div>

</section>
```

```
<section class="pull-demo">

    <h2>  Feature Branch: feature-pull-demonstration</h2>
```

```
<div class="command">git checkout -b feature-pull-demonstration</div>

<div class="command"># Developing pull demonstration features...</div>

<div class="command">git add .</div>

<div class="command">git commit -m "feat: Add pull operation demonstration"</div>

<div class="command">git push -u origin feature-pull-demonstration</div>

<div style="margin-top: 15px;">
    <p><strong>Branch Status:</strong> <span class="status-indicator status-ahead">Ahead of main</span></p>
    <p><strong>Changes:</strong> Added pull operation documentation and demonstration features</p>
    <p><strong>Next:</strong> Merge to main and demonstrate pull from remote</p>
</div>
</section>

<!-- Add interactive pull demonstration -->
<section class="feature-card">
```

```
<h2>  Git Pull Process</h2>

<div style="background: #2d3748; color: white; padding: 15px; border-radius: 5px; font-family: 'Courier New', monospace;">

    <div>$ git fetch origin</div>

    <div>remote: Counting objects: 3, done.</div>

    <div>remote: Compressing objects: 100% (2/2), done.</div>

    <div>remote: Total 3 (delta 1), reused 0 (delta 0)</div>

    <div>Unpacking objects: 100% (3/3), done.</div>

    <br>

    <div>$ git merge origin/main</div>

    <div>Updating a1b2c3d..e4f5g6h</div>

    <div>Fast-forward</div>

    <div> README.md | 2 ++</div>

    <div> 1 file changed, 2 insertions(+)</div>

</div>

</section>

<script>

// Interactive elements for pull demonstration
```

```
function simulatePull() {  
    const statusElement =  
        document.querySelector('.status-indicator');  
  
    statusElement.textContent = 'Syncing with Remote...';  
  
    statusElement.className = 'status-indicator status-  
behind';  
  
    setTimeout(() => {  
        statusElement.textContent = 'Local and Remote  
Synced';  
  
        statusElement.className = 'status-indicator  
status-synced';  
  
        alert('✅ Pull operation completed successfully!  
Local repository is now synced with Bitbucket.');
```

}, 2000);

}

```
function showPullCommand() {  
    alert('Git Pull Command:\n\n' +  
        'git pull origin main\n\n' +  
        'This command:\n' +  
        '1. Fetches changes from Bitbucket\n' +
```

```
'2. Merges them into your local branch\n' +  
'3. Updates your working directory');  
}  
</script>
```

```
<!-- Add pull demonstration buttons -->  
<div style="text-align: center; margin: 20px 0;">  
  <button onclick="simulatePull()" style="background:  
#0052cc; color: white; border: none; padding: 12px 24px;  
border-radius: 5px; cursor: pointer; margin: 5px;">  
     Simulate Pull Operation  
  </button>  
  <button onclick="showPullCommand()"  
style="background: #00a3ff; color: white; border: none;  
padding: 12px 24px; border-radius: 5px; cursor: pointer;  
margin: 5px;">  
     Show Pull Command  
  </button>  
</div>
```

Update README.md:

markdown

```
# Bitbucket Pull Demo Project
```

This project demonstrates complete Git and Bitbucket workflow including pull operations.

🎯 Learning Objectives

- Git repository initialization and management
- Bitbucket remote repository setup
- Branching strategies and merge operations
- Commit history management
- Pull operations from remote repository
- Team collaboration simulation

📁 Project Structure

```
bitbucket-pull-demo/  
├── index.html # Main web application  
├── README.md # Project documentation  
├── .gitignore # Git ignore rules  
├── CHANGELOG.md # Version history  
└── assets/ # Additional resources (future)
```

text

Bitbucket Workflow

1. **Local Setup**

- Initialize Git repository
- Create project structure
- Make initial commits

2. **Bitbucket Integration**

- Create remote repository on Bitbucket
- Push local code to Bitbucket
- Configure repository settings

3. **Branching Strategy**

- Create feature branches
- Develop features independently
- Merge back to main branch

4. **Pull Operations**

- Simulate team collaboration
- Pull latest changes from Bitbucket
- Handle synchronization

5. **Bitbucket Features**

- Pull requests and code review
- Pipelines for CI/CD
- Integration with Jira
- Access control and permissions

NEW Pull Demonstration Features

Git Pull Operations

- Remote change synchronization
- Fetch and merge processes
- Conflict resolution scenarios
- Team collaboration simulation

Bitbucket Specific

- Pull request workflows
- Branch comparison tools
- Code review interface
- Integration with Atlassian ecosystem

🔧 Technologies Used

- Git for version control
- Bitbucket for remote repository
- HTML/CSS for web interface
- JavaScript for interactive elements
- Markdown for documentation

📈 Git Commands Coverage

- Basic: init, add, commit, status
- Branching: branch, checkout, merge
- Remote: remote, push, pull, fetch
- History: log, show, diff, --graph

Update CHANGELOG.md:

markdown

Changelog

All notable changes to this project will be documented in this file.

[Unreleased]

Added

- Pull operation demonstration features
- Interactive JavaScript elements
- Enhanced documentation
- Bitbucket-specific workflow examples

[1.0.0] - 2023-11-24

Added

- Initial project structure
- Basic HTML web interface
- Project documentation
- Git ignore rules
- Changelog file

Create pull-guide.md (new file):

markdown

Git Pull Operations Guide

Understanding Git Pull

What is `git pull`?

The `git pull` command is used to fetch and download content from a remote repository and immediately update the local repository to match that content.

Pull Command Syntax

``` bash

```
git pull [<options>] [<repository> [<refspec>...]]
```

## Common Pull Scenarios

### 1. Basic Pull

bash

```
git pull origin main
```

### 2. Pull with Rebase

bash

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

### 3. Pull Specific Branch

bash

```
git pull origin feature-branch
```

## Pull vs Fetch

**git pull**

- Fetches AND merges in one command
- Updates your current branch
- Can cause merge conflicts

## **git fetch**

- Only downloads new data
- Doesn't modify working directory
- Safer for reviewing changes

bash

*# Safe approach*

git fetch origin

git log origin/main

git merge origin/main

## **Handling Pull Conflicts**

### **When conflicts occur:**

1. Pull identifies conflicting changes
2. Git marks conflicts in files
3. Manual resolution required
4. Commit the resolved files

### **Conflict resolution:**

bash

git pull origin main

```
If conflicts occur...
Edit files to resolve conflicts
git add .
git commit -m "Resolve merge conflicts"
```

## Best Practices

- 1. Always pull before pushing**
- 2. Use git fetch to review changes first**
- 3. Consider git pull --rebase for cleaner history**
- 4. Communicate with team about major changes**
- 5. Use feature branches to isolate work**

## Bitbucket Integration

- Use Bitbucket Pull Requests for code review
- Set up branch permissions in Bitbucket
- Utilize Bitbucket Pipelines for CI/CD
- Integrate with Jira for issue tracking

text

## ### Step 14: Commit Changes in Feature Branch

\*\*Check status:\*\*

```
``` bash
```

```
git status
```

 **Expected Output:**

text

On branch feature-pull-demonstration

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: CHANGELOG.md

modified: README.md

modified: index.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

pull-guide.md

no changes added to commit (use "git add" and/or "git commit -a")

Stage and commit changes:

bash

```
git add .
```

```
git commit -m "feat: Add pull operation demonstration,  
interactive elements, and comprehensive guide"
```

 **Expected Output:**

text

```
[feature-pull-demonstration e4f5g6h] feat: Add pull  
operation demonstration, interactive elements, and  
comprehensive guide
```

```
4 files changed, 178 insertions(+), 5 deletions(-)
```

```
create mode 100644 pull-guide.md
```

Step 15: Push Feature Branch to Bitbucket

bash

```
git push -u origin feature-pull-demonstration
```

 **Expected Output:**

text

```
Enumerating objects: 9, done.
```

```
Counting objects: 100% (9/9), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (8/8), done.
```

```
Writing objects: 100% (8/8), 3.12 KiB | 3.12 MiB/s, done.
```

```
Total 8 (delta 3), reused 0 (delta 0), pack-reused 0
```

remote:

remote: Create pull request for feature-pull-demonstration:

remote: <https://bitbucket.org/your-username/bitbucket-pull-demo/pull-requests/new?source=feature-pull-demonstration&t=1>

remote:

To bitbucket.org:your-username/bitbucket-pull-demo.git

* [new branch] feature-pull-demonstration -> feature-pull-demonstration

Branch 'feature-pull-demonstration' set up to track remote branch 'feature-pull-demonstration' from 'origin'.



Phase 5: Merge Branch and Display History

Step 16: Switch Back to Main Branch

bash

git checkout main



Expected Output:

text

Switched to branch 'main'

Your branch is up to date with 'origin/main'.

Step 17: Merge Feature Branch into Main

bash

git merge feature-pull-demonstration

 **Expected Output:**

text

Updating a1b2c3d..e4f5g6h

Fast-forward

CHANGELOG.md | 7 +++++++

README.md | 20 ++++++*****--

index.html | 95

+++++*****+++++*****+++++*****+++++*****

+++++*****+++++*****+++++*****+++++*****

pull-guide.md | 58

+++++*****+++++*****+++++*****+++++*****

+++++*****

4 files changed, 178 insertions(+), 2 deletions(-)

create mode 100644 pull-guide.md

Step 18: Push Merged Changes to Bitbucket

bash

git push origin main

 **Expected Output:**

text

Total 0 (delta 0), reused 0 (delta 0), pack-reused 0

To bitbucket.org:your-username/bitbucket-pull-demo.git

a1b2c3d..e4f5g6h main -> main

Step 19: Display Chronological Commit History

View detailed history:

bash

git log

Expected Output:

text

commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD ->
main, origin/main, origin/feature-pull-demonstration,
feature-pull-demonstration)

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 21:20:00 2023 +0530

feat: Add pull operation demonstration, interactive
elements, and comprehensive guide

commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r

Author: Your Name <your.email@example.com>

Date: Mon Nov 24 21:10:00 2023 +0530

Initial commit: Project setup with web interface, documentation, and Bitbucket-focused content

One-line history with graph:

bash

```
git log --oneline --graph --all
```

Expected Output:

text

```
* e4f5g6h (HEAD -> main, origin/main, origin/feature-pull-demonstration, feature-pull-demonstration) feat: Add pull operation demonstration, interactive elements, and comprehensive guide
```

```
* a1b2c3d Initial commit: Project setup with web interface, documentation, and Bitbucket-focused content
```

Custom format history:

bash

```
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

Expected Output:

text

```
e4f5g6h | Your Name | 2023-11-24 | feat: Add pull operation demonstration, interactive elements, and comprehensive guide
```

a1b2c3d | Your Name | 2023-11-24 | Initial commit: Project setup with web interface, documentation, and Bitbucket-focused content

Phase 6: Simulate Team Collaboration and Pull Changes

Step 20: Simulate Changes on Bitbucket (Web Interface)

- 1. Go to your Bitbucket repository**
- 2. Click on README.md → Edit (pencil icon)**
- 3. Add this section at the end:**

markdown

```
##  Team Collaboration Simulation
```

This section was added directly on Bitbucket to simulate team collaboration.

Remote Changes Made:

- Added team collaboration section
- Updated documentation via web interface
- Simulated multi-user workflow scenario

Pull Process:

1. Team member makes changes on Bitbucket
2. Local developer pulls changes using `git pull`
3. Verify synchronization between local and remote
4. Continue development with latest updates

Bitbucket Features Used:

- Web-based file editing
- Direct commit to main branch
- Change tracking and history
- Team collaboration tools

4. Commit changes:

- **Commit message:** "docs: Add team collaboration section via Bitbucket web"
- **Select:** "Commit directly to the main branch"
- **Click "Commit"**

Step 21: Check Local Status Before Pull

Check if local is behind remote:

bash

git status

 **Expected Output:**

text

On branch main

Your branch is up to date with 'origin/main'.

This might not show changes yet. Let's check explicitly:

bash

git fetch origin

git status

 **Expected Output:**

text

On branch main

Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.

(use "git pull" to update your local branch)

Step 22: Pull Changes from Bitbucket

Pull the changes:

bash

git pull origin main

 **Expected Output:**

text

remote: Counting objects: 3, done.

remote: Compressing objects: 100% (3/3), done.

```
remote: Total 3 (delta 2), reused 0 (delta 0)
```

```
Unpacking objects: 100% (3/3), 1.15 KiB | 147.00 KiB/s,  
done.
```

```
From bitbucket.org:your-username/bitbucket-pull-demo
```

```
e4f5g6h..8901j2k main -> origin/main
```

```
Updating e4f5g6h..8901j2k
```

```
Fast-forward
```

```
 README.md | 13 ++++++++  
1 file changed, 13 insertions(+)
```

Step 23: Verify Pulled Changes

Check the updated README:

```
bash
```

```
cat README.md
```

 **Expected Output:** Should show the new "Team Collaboration Simulation" section

View updated commit history:

```
bash
```

```
git log --oneline --graph --all
```

 **Expected Output:**

```
text
```

- * 8901j2k (HEAD -> main, origin/main) docs: Add team collaboration section via Bitbucket web
- * e4f5g6h (origin/feature-pull-demonstration, feature-pull-demonstration) feat: Add pull operation demonstration, interactive elements, and comprehensive guide
- * a1b2c3d Initial commit: Project setup with web interface, documentation, and Bitbucket-focused content

Step 24: Make Local Changes and Push Back

Update index.html to acknowledge successful pull:

html

```
<!-- Add this section before closing body tag -->  
<section class="pull-demo" style="background: linear-gradient(135deg, #28a745, #20c997); color: white;">  
  <h2>  Successful Pull Operation Completed!</h2>  
  <p>Changes from Bitbucket have been successfully pulled to local machine.</p>  
  <p><strong>Latest Update:</strong> Team collaboration section added via Bitbucket web interface</p>  
  <p><strong>Pull Command Used:</strong> <code>git pull origin main</code></p>  
  <p><strong>Result:</strong> Local repository synchronized with Bitbucket remote</p>
```

```
<div style="text-align: center; margin-top: 15px;">  
    <button onclick="celebrateSuccessfulPull()"  
    style="background: white; color: #28a745; border: none;  
    padding: 12px 24px; border-radius: 5px; cursor: pointer;  
    font-weight: bold;">  
        🎉 Celebrate Successful Pull!  
    </button>  
</div>  
</section>  
  
<script>  
    function celebrateSuccessfulPull() {  
        alert('🎉 Pull Operation Successful!\n\n' +  
            ' ✅ Local repository synchronized with Bitbucket\n' +  
            '+\n' +  
            ' 📁 Changes fetched and merged successfully\n' +  
            ' 💬 Ready for continued development');  
    }  
</script>
```

Commit and push local changes:

```
bash
```

```
git add index.html
```

```
git commit -m "feat: Add pull success confirmation and  
celebration elements"
```

```
git push origin main
```

 **Expected Output:**

text

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 8 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 1.25 KiB | 1.25 MiB/s, done.

Total 3 (delta 2), reused 0 (delta 0), pack-reused 0

To bitbucket.org:your-username/bitbucket-pull-demo.git

8901j2k..5678k9l main -> main



Phase 7: Final History and Verification

Step 25: Display Final Chronological History

Complete history with all operations:

```
bash
```

```
git log --oneline --graph --all --decorate
```

 **Expected Output:**

text

- * 5678k9l (HEAD -> main, origin/main) feat: Add pull success confirmation and celebration elements
- * 8901j2k docs: Add team collaboration section via Bitbucket web
- * e4f5g6h (origin/feature-pull-demonstration, feature-pull-demonstration) feat: Add pull operation demonstration, interactive elements, and comprehensive guide
- * a1b2c3d Initial commit: Project setup with web interface, documentation, and Bitbucket-focused content

Detailed history view:

bash

```
git log --pretty=format:"%h %ad | %s" --date=short
```

 **Expected Output:**

text

5678k9l 2023-11-24 | feat: Add pull success confirmation and celebration elements

8901j2k 2023-11-24 | docs: Add team collaboration section via Bitbucket web

e4f5g6h 2023-11-24 | feat: Add pull operation demonstration, interactive elements, and comprehensive guide

a1b2c3d 2023-11-24 | Initial commit: Project setup with web interface, documentation, and Bitbucket-focused content

Show statistics for each commit:

bash

```
git log --stat
```

- ✓ **Expected Output:** Shows files changed and line statistics for each commit

Step 26: Verify All Operations on Bitbucket

1. **Go to your Bitbucket repository**
2. **Check "Commits"** - should show all 4 commits in chronological order
3. **Check "Branches"** - should show both main and feature-pull-demonstration
4. **Verify files** contain all the changes made throughout the process
5. **Check "Pull requests"** - you might see the suggested pull request for the feature branch



Complete Workflow Summary

All Commands Used:

bash

```
# Setup and initial commit

mkdir bitbucket-pull-demo
cd bitbucket-pull-demo
git init

# Create files: index.html, README.md, .gitignore,
CHANGELOG.md

git add .
git commit -m "Initial commit"
```

```
# Bitbucket setup

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

git push -u origin main
```

```
# Branching and development

git checkout -b feature-pull-demonstration

# Make changes to files

git add .
git commit -m "feat: Add pull demonstration features"
git push -u origin feature-pull-demonstration
```

Merging

```
git checkout main
```

```
git merge feature-pull-demonstration
```

```
git push origin main
```

Pull operations

```
git fetch origin
```

```
git status
```

```
git pull origin main
```

History viewing

```
git log
```

```
git log --oneline --graph --all
```

```
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

```
git log --stat
```

Git Pull Operations Summary

Command	Purpose	When to Use
git pull	Fetch + merge	Regular updates from team
git pull --rebase	Fetch + rebase	Clean history maintenance

Command	Purpose	When to Use
<code>git fetch + git merge</code>	Separate operations	Review changes before merging
<code>git pull origin branch</code>	Specific branch	Working with feature branches

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.

GitLab Repository Management with Pull - Complete Step-by-Step Guide

I'll walk you through creating a project, pushing to GitLab, branching, merging, displaying history, and pulling changes to local machine.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
 2. **GitLab Account** - Create at gitlab.com
 3. **Text Editor** - VS Code, Notepad++, etc.
 4. **Terminal/Command Prompt**
-

Phase 1: GitLab Account Setup

Step 1: Create GitLab Account (if you don't have one)

1. **Go to gitlab.com**
2. **Click "Register" or "Sign up"**
3. **Choose registration method** (Email, Google, GitHub, etc.)
4. **Verify your email address**
5. **Complete your profile**

Step 2: Create SSH Key (Recommended for Secure Access)

Generate SSH key:

bash

```
ssh-keygen -t rsa -b 4096 -C "your.email@example.com"
```

Expected Output:

text

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/YourName/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /c/Users/YourName/.ssh/id_rsa

Your public key has been saved in /c/Users/YourName/.ssh/id_rsa.pub

Copy public key:

bash

On Windows

```
cat ~/.ssh/id_rsa.pub
```

On Linux/Mac

```
cat ~/.ssh/id_rsa.pub | clip # or pbcopy on Mac
```

Add SSH key to GitLab:

1. **GitLab → User Settings → SSH Keys**
 2. **Paste your public key**
 3. **Add title** (e.g., "My Laptop")
 4. **Click "Add key"**
-

 **Phase 2: Create Local Project and Git Setup**

Step 3: Create Project Directory

bash

Create and navigate to project directory

```
mkdir gitlab-complete-demo
```

```
cd gitlab-complete-demo
```

Step 4: Initialize Git Repository

bash

```
git init
```

 **Expected Output:**

text

Initialized empty Git repository in C:/gitlab-complete-demo/.git/

Step 5: Configure Git User

bash

```
git config user.name "Your Name"
```

```
git config user.email "your.email@example.com"
```

Step 6: Create Project Files

Create index.html:

html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
 <head>
```

```
   <meta charset="UTF-8">
```

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

```
   <title>GitLab Complete Workflow Demo</title>
```

```
 <style>
```

```
 body {
```

```
   font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
```

```
   max-width: 1000px;
```

```
   margin: 0 auto;
```

```
   padding: 20px;
```

```
   background: linear-gradient(135deg, #fc6d26, #e24329);
```

```
   color: #333;
```

```
   line-height: 1.6;
```

```
}
```

```
.container {
```

```
   background: rgba(255, 255, 255, 0.95);
```

```
padding: 30px;  
border-radius: 15px;  
box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);  
}  
  
.header {  
    text-align: center;  
    margin-bottom: 30px;  
    border-bottom: 3px solid #fc6d26;  
    padding-bottom: 20px;  
}  
  
.feature-card {  
    background: #f8f9fa;  
    padding: 20px;  
    margin: 15px 0;  
    border-radius: 10px;  
    border-left: 5px solid #fc6d26;  
}  
  
.gitlab-feature {  
    background: linear-gradient(135deg, #e24329, #fc6d26);  
    color: white;  
    padding: 25px;  
    border-radius: 10px;  
    margin: 20px 0;  
}  
  
.command {  
    background: #2d3748;  
    color: #e2e8f0;  
    padding: 10px 15px;
```

```
border-radius: 5px;  
font-family: 'Courier New', monospace;  
margin: 8px 0;  
border-left: 4px solid #fc6d26;  
}  
  
.workflow-step {  
background: #e7f3ff;  
padding: 15px;  
border-radius: 8px;  
border: 2px solid #3498db;  
margin: 15px 0;  
}  
  
.status-indicator {  
display: inline-block;  
padding: 5px 15px;  
border-radius: 20px;  
font-weight: bold;  
margin: 5px;  
}  
  
.status-synced { background: #28a745; color: white; }  
.status-pending { background: #ffc107; color: black; }  
.status-remote { background: #17a2b8; color: white; }  
  
</style>  
</head>  
<body>  
<div class="container">  
<div class="header">  
<h1>GitLab Complete Workflow Demo</h1>
```

```
<p>Demonstrating end-to-end Git operations with GitLab including pull  
functionality</p>
```

```
</div>
```

```
<div class="gitlab-feature">
```

GitLab DevOps Platform

```
<p>Complete DevOps lifecycle management with built-in CI/CD, issue tracking,  
and repository management.</p>
```

```
</div>
```

```
<div class="feature-card">
```

Complete Workflow Overview

```
<p>This project demonstrates the full GitLab workflow:</p>
```

```
<ul>
```

- Repository creation and initialization
- Branch creation and feature development
- Merging and commit history management
- Pull operations from remote repository
- Team collaboration simulation
- GitLab CI/CD integration

```
</ul>
```

```
</div>
```

```
<div class="workflow-step">
```

Current Workflow Status

```
<p><strong>Step 1:</strong> <span class="status-indicator status-synced">Local  
Repository Created</span></p>
```

```
<p><strong>Step 2:</strong> <span class="status-indicator status-  
pending">Push to GitLab</span></p>
```

```
<p><strong>Step 3:</strong> <span class="status-indicator status-pending">Create Feature Branch</span></p>

<p><strong>Step 4:</strong> <span class="status-indicator status-pending">Merge & Pull Operations</span></p>

</div>

<div class="feature-card">

  <h2> 🦁 GitLab Advantages</h2>

  <ul>

    <li><strong>Built-in CI/CD:</strong> No external tools needed</li>
    <li><strong>Container Registry:</strong> Built-in Docker registry</li>
    <li><strong>Kubernetes Integration:</strong> Native Kubernetes support</li>
    <li><strong>Free Private Repos:</strong> Unlimited private repositories</li>
    <li><strong>Self-Hosting:</strong> Free self-hosted option available</li>
    <li><strong>Complete DevOps:</strong> Single application for entire lifecycle</li>
  </ul>

</div>

<div class="feature-card">

  <h2> 💻 Git Commands in This Demo</h2>

  <div class="command">git init</div>
  <div class="command">git add .</div>
  <div class="command">git commit -m "message"</div>
  <div class="command">git remote add origin</div>
  <div class="command">git push -u origin main</div>
  <div class="command">git checkout -b feature-branch</div>
  <div class="command">git merge</div>
  <div class="command">git pull origin main</div>
</div>
```

```
<div class="command">git log --oneline --graph --all</div>
</div>
</div>
</body>
</html>
```

Create README.md:

markdown

```
# GitLab Complete Workflow Demo
```

This project demonstrates complete Git and GitLab workflow including repository management, branching, merging, and pull operations.

🎯 Learning Objectives

- Git repository initialization and management
- GitLab remote repository setup
- Branch creation and feature development
- Merging strategies and conflict resolution
- Commit history management and visualization
- Pull operations from remote repository
- GitLab CI/CD fundamentals

📁 Project Structure

```
gitlab-complete-demo/
├── index.html # Main web application
├── README.md # Project documentation
├── .gitignore # Git ignore rules
├── CHANGELOG.md # Version history
├── .gitlab-ci.yml # CI/CD configuration
└── assets/ # Additional resources
```

text

Complete Workflow Steps

Phase 1: Local Setup

- Initialize Git repository
- Create project structure
- Make initial commits

Phase 2: GitLab Integration

- Create remote repository on GitLab
- Push local code to GitLab
- Configure repository settings

Phase 3: Branching Strategy

- Create feature branches
- Develop features independently
- Push branches to GitLab

Phase 4: Merging & History

- Merge branches into main
- Display chronological commit history
- Manage project timeline

Phase 5: Pull Operations

- Simulate team collaboration
- Pull changes from GitLab
- Synchronize local and remote

Phase 6: GitLab Features

- Explore CI/CD pipelines
- Use merge requests
- Leverage GitLab's DevOps capabilities

🛡 Technologies Used

- Git for version control
- GitLab for DevOps platform
- HTML/CSS for web interface
- Markdown for documentation
- GitLab CI/CD for automation

📈 Git Commands Coverage

- Repository: init, remote, clone
- Basic: add, commit, status, log
- Branching: branch, checkout, merge
- Remote: push, pull, fetch
- History: log, show, diff, --graph

Create .gitignore:

```
text  
# Dependencies  
node_modules/  
vendor/  
packages/
```

```
# Environment
```

```
.env
```

```
.env.local
```

```
.env.production
```

```
# Logs
```

```
*.log
```

```
logs/
```

```
npm-debug.log*
```

```
# Build outputs
```

```
dist/
```

```
build/
```

```
out/
```

```
*.exe
```

```
# IDE
```

```
.vscode/
```

```
.idea/
```

```
*.swp
```

```
*.swo
```

```
# OS
```

```
.DS_Store
```

```
Thumbs.db
```

```
# GitLab CI
```

```
.cache/
```

Create CHANGELOG.md:

markdown

Changelog

All notable changes to this project will be documented in this file.

[1.0.0] - 2023-11-24

Added

- Initial project structure
- Basic HTML web interface
- Project documentation
- Git ignore rules
- Changelog file

Step 7: Make Initial Commit

Check status:

bash

git status

 **Expected Output:**

text

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

CHANGELOG.md

README.md

index.html

nothing added to commit but untracked files present (use "git add" to track)

Stage files and commit:

bash

git add .

git commit -m "Initial commit: Project setup with complete GitLab workflow documentation"

 **Expected Output:**

text

[main (root-commit) a1b2c3d] Initial commit: Project setup with complete GitLab workflow documentation

4 files changed, 195 insertions(+)

create mode 100644 .gitignore

create mode 100644 CHANGELOG.md

create mode 100644 README.md

create mode 100644 index.html



Phase 3: GitLab Repository Setup

Step 8: Create GitLab Repository

1. **Log in to gitlab.com**
2. **Click "+" icon → "New project"**
3. **Choose "Create blank project"**
4. **Project details:**
 - o **Project name:** gitlab-complete-demo
 - o **Project slug:** gitlab-complete-demo (auto-filled)
 - o **Visibility Level: Private** (for security)

- Initialize repository with README: No (we have our own)

5. Click "Create project"

Step 9: Connect Local Repository to GitLab

Copy the repository URL from GitLab (use SSH or HTTPS):

For HTTPS:

bash

```
git remote add origin https://gitlab.com/your-username/gitlab-complete-demo.git
```

For SSH (recommended):

bash

```
git remote add origin git@gitlab.com:your-username/gitlab-complete-demo.git
```

Verify remote connection:

bash

```
git remote -v
```

 Expected Output:

text

```
origin git@gitlab.com:your-username/gitlab-complete-demo.git (fetch)
```

```
origin git@gitlab.com:your-username/gitlab-complete-demo.git (push)
```

Step 10: Push to GitLab

First push (set upstream):

bash

```
git push -u origin main
```

 Expected Output:

text

```
Enumerating objects: 7, done.
```

```
Counting objects: 100% (7/7), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (6/6), done.
```

```
Writing objects: 100% (7/7), 2.85 KiB | 2.85 MiB/s, done.
```

```
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To gitlab.com:your-username/gitlab-complete-demo.git
```

```
* [new branch]  main -> main
```

```
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Step 11: Verify on GitLab

1. **Refresh your GitLab project page**
 2. **You should see your files**
 3. **Click on "Repository" → "Commits"** to see the initial commit
 4. **Explore the GitLab interface** (Issues, Merge Requests, CI/CD)
-

Phase 4: Create and Work on Feature Branch

Step 12: Create Feature Branch

Create and switch to new branch:

```
bash
```

```
git checkout -b feature-ci-cd-integration
```

 **Expected Output:**

```
text
```

```
Switched to a new branch 'feature-ci-cd-integration'
```

Verify current branch:

```
bash
```

```
git branch
```

 **Expected Output:**

```
text
```

```
main
```

```
* feature-ci-cd-integration
```

Step 13: Make Changes in Feature Branch

Update index.html - add CI/CD section:

```
html
```

```
<!-- Add this before closing body tag -->

<section style="margin: 30px 0;">
  <div class="gitlab-feature">
    <h2>⚙️ GitLab CI/CD Integration Features</h2>
    <p>This section demonstrates features developed in the <strong>feature-ci-cd-integration</strong> branch</p>

    <div style="display: grid; grid-template-columns: 1fr 1fr; gap: 20px; margin-top: 20px;">
      <div style="background: rgba(255,255,255,0.1); padding: 20px; border-radius: 8px;">
        <h3>🔍 Automated Testing</h3>
        <ul>
          <li>Unit tests execution</li>
          <li>Integration testing</li>
          <li>Code quality checks</li>
          <li>Security scanning</li>
        </ul>
      </div>
      <div style="background: rgba(255,255,255,0.1); padding: 20px; border-radius: 8px;">
        <h3>🚀 Deployment Pipeline</h3>
        <ul>
          <li>Continuous integration</li>
          <li>Continuous deployment</li>
          <li>Environment management</li>
          <li>Rollback capabilities</li>
        </ul>
      </div>
    </div>
  </div>
</section>
```

```

        </div>
        </div>
        </div>
    </section>

<section class="workflow-step">
    <h2> 🌱 Feature Branch: feature-ci-cd-integration</h2>
    <div class="command">git checkout -b feature-ci-cd-integration</div>
    <div class="command"># Developing CI/CD integration features...</div>
    <div class="command">git add .</div>
    <div class="command">git commit -m "feat: Add CI/CD integration features"</div>
    <div class="command">git push -u origin feature-ci-cd-integration</div>

    <div style="margin-top: 15px;">
        <p><strong>Branch Status:</strong> <span class="status-indicator status-pending">Active Development</span></p>
        <p><strong>Changes:</strong> Added CI/CD pipeline documentation and integration features</p>
        <p><strong>Next:</strong> Merge to main and demonstrate pull operations</p>
    </div>
</section>

<!-- Add GitLab CI/CD example -->
<section class="feature-card">
    <h2> 📄 GitLab CI/CD Pipeline Example</h2>
    <pre style="background: #2d3748; color: #e2e8f0; padding: 15px; border-radius: 5px; overflow-x: auto; font-family: 'Courier New', monospace;">
# .gitlab-ci.yml
stages:

```

- test
- build
- deploy

variables:

```
NODE_VERSION: "16"
```

before_script:

- echo "Starting CI/CD pipeline for GitLab Demo"

unit_tests:

stage: test

script:

- echo "Running unit tests..."
- npm test

lint_code:

stage: test

script:

- echo "Linting source code..."
- npm run lint

build_project:

stage: build

script:

- echo "Building project..."
- npm run build

artifacts:

```
paths:
```

```
- dist/
```

```
deploy_staging:
```

```
stage: deploy
```

```
script:
```

```
- echo "Deploying to staging environment..."
```

```
- npm run deploy:staging
```

```
environment:
```

```
name: staging
```

```
url: https://staging.example.com
```

```
only:
```

```
- main
```

```
pages:
```

```
stage: deploy
```

```
script:
```

```
- echo "Deploying to GitLab Pages..."
```

```
- mkdir public
```

```
- cp -r dist/* public/
```

```
artifacts:
```

```
paths:
```

```
- public
```

```
only:
```

```
- main</pre>
```

```
</section>
```

```
<script>
```

```
// Interactive elements for CI/CD demonstration

function simulatePipeline() {

    const statusElement = document.querySelector('.status-indicator');

    const originalStatus = statusElement.textContent;
    const originalClass = statusElement.className;

    statusElement.textContent = 'Pipeline Running...';
    statusElement.className = 'status-indicator status-pending';

    setTimeout(() => {

        statusElement.textContent = 'Pipeline Passed ✅';
        statusElement.className = 'status-indicator status-synced';

        alert('🎉 GitLab CI/CD Pipeline completed successfully!\n\nAll tests passed and deployment successful!');

    }, 3000);

}

function showCICommands() {

    alert('GitLab CI/CD Commands:\n\n' +
        '1. git add .gitlab-ci.yml\n' +
        '2. git commit -m "feat: Add CI/CD pipeline"\n' +
        '3. git push origin feature-branch\n' +
        '4. Create Merge Request in GitLab\n' +
        '5. Pipeline runs automatically on push');

}

</script>

<!-- Add CI/CD demonstration buttons -->
```

```
<div style="text-align: center; margin: 20px 0;">  
    <button onclick="simulatePipeline()" style="background: #fc6d26; color: white; border: none; padding: 12px 24px; border-radius: 5px; cursor: pointer; margin: 5px;">  
         Simulate CI/CD Pipeline  
    </button>  
  
    <button onclick="showCICommands()" style="background: #e24329; color: white; border: none; padding: 12px 24px; border-radius: 5px; cursor: pointer; margin: 5px;">  
         Show CI/CD Commands  
    </button>  
</div>
```

Update README.md:

markdown

```
# GitLab Complete Workflow Demo
```

This project demonstrates complete Git and GitLab workflow including repository management, branching, merging, and pull operations.

Learning Objectives

- Git repository initialization and management
- GitLab remote repository setup
- Branch creation and feature development
- Merging strategies and conflict resolution
- Commit history management and visualization
- Pull operations from remote repository
- GitLab CI/CD fundamentals

Project Structure

```
gitlab-complete-demo/
├── index.html # Main web application
├── README.md # Project documentation
├── .gitignore # Git ignore rules
├── CHANGELOG.md # Version history
├── .gitlab-ci.yml # CI/CD configuration
└── assets/ # Additional resources
```

text

Complete Workflow Steps

Phase 1: Local Setup

- Initialize Git repository
- Create project structure
- Make initial commits

Phase 2: GitLab Integration

- Create remote repository on GitLab
- Push local code to GitLab
- Configure repository settings

Phase 3: Branching Strategy

- Create feature branches
- Develop features independently
- Push branches to GitLab

Phase 4: Merging & History

- Merge branches into main
- Display chronological commit history
- Manage project timeline

Phase 5: Pull Operations

- Simulate team collaboration
- Pull changes from GitLab
- Synchronize local and remote

Phase 6: GitLab Features

- Explore CI/CD pipelines
- Use merge requests
- Leverage GitLab's DevOps capabilities

NEW CI/CD Integration Features

GitLab CI/CD Pipeline

- Automated testing stages
- Build and deployment processes
- Environment management
- Artifact management
- GitLab Pages deployment

Branch Protection

- Merge request requirements
- Code review enforcement
- Pipeline status checks
- Approval rules

language Technologies Used

- Git for version control
- GitLab for DevOps platform
- HTML/CSS for web interface
- JavaScript for interactivity
- Markdown for documentation
- GitLab CI/CD for automation

Git Commands Coverage

- Repository: init, remote, clone
- Basic: add, commit, status, log
- Branching: branch, checkout, merge
- Remote: push, pull, fetch
- History: log, show, diff, --graph

Update CHANGELOG.md:

markdown

Changelog

All notable changes to this project will be documented in this file.

[Unreleased]

Added

- CI/CD pipeline integration features
- Interactive JavaScript elements
- GitLab-specific workflow examples
- Enhanced project documentation

```
## [1.0.0] - 2023-11-24
```

Added

- Initial project structure
- Basic HTML web interface
- Project documentation
- Git ignore rules
- Changelog file

Create .gitlab-ci.yml (GitLab CI/CD configuration):

yaml

```
# GitLab CI/CD Pipeline Configuration
```

```
image: node:16
```

stages:

- test
- build
- deploy

cache:

paths:

- node_modules/

before_script:

- echo "Starting GitLab CI/CD pipeline for Complete Workflow Demo"
- node --version
- npm --version

unit_tests:

```
stage: test  
  
script:  
  
- echo "Running unit tests..."  
  
- echo "No tests configured yet - this is a demo project"  
  
- echo "Tests would run here in a real project"
```

lint_code:

```
stage: test  
  
script:  
  
- echo "Linting source code..."  
  
- echo "HTML, CSS, and JavaScript linting would happen here"  
  
- echo "All code quality checks passed"
```

build_project:

```
stage: build  
  
script:  
  
- echo "Building project..."  
  
- echo "Project built successfully"  
  
- mkdir -p dist  
  
- cp index.html dist/  
  
- echo "Build artifacts created"
```

artifacts:

```
paths:  
  
- dist/  
  
expire_in: 1 week
```

deploy_staging:

```
stage: deploy
```

script:

- echo "Deploying to staging environment..."
- echo "Staging deployment complete"
- echo "Application available at: https://staging.example.com"

environment:

name: staging

url: https://staging.example.com

only:

- main

pages:

stage: deploy

script:

- echo "Deploying to GitLab Pages..."
- mkdir -p public
- cp index.html public/
- cp -r assets/ public/ 2>/dev/null || true
- echo "GitLab Pages deployment ready"

artifacts:

paths:

- public

only:

- main

Pipeline status notification

after_script:

- echo "CI/CD pipeline completed"
- echo "Job status: \$CI_JOB_STATUS"

Step 14: Commit Changes in Feature Branch

Check status:

bash

git status

Expected Output:

text

On branch feature-ci-cd-integration

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: CHANGELOG.md

modified: README.md

modified: index.html

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitlab-ci.yml

no changes added to commit (use "git add" and/or "git commit -a")

Stage and commit changes:

bash

git add .

git commit -m "feat: Add GitLab CI/CD pipeline configuration, interactive elements, and comprehensive documentation"

Expected Output:

text

[feature-ci-cd-integration e4f5g6h] feat: Add GitLab CI/CD pipeline configuration, interactive elements, and comprehensive documentation

```
4 files changed, 212 insertions(+), 5 deletions(-)
```

```
create mode 100644 .gitlab-ci.yml
```

Step 15: Push Feature Branch to GitLab

bash

```
git push -u origin feature-ci-cd-integration
```

Expected Output:

text

```
Enumerating objects: 9, done.
```

```
Counting objects: 100% (9/9), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (8/8), done.
```

```
Writing objects: 100% (8/8), 3.45 KiB | 3.45 MiB/s, done.
```

```
Total 8 (delta 3), reused 0 (delta 0), pack-reused 0
```

remote:

```
remote: To create a merge request for feature-ci-cd-integration, visit:
```

```
remote: https://gitlab.com/your-username/gitlab-complete-demo/-  
/merge_requests/new?merge_request%5Bsource_branch%5D=feature-ci-cd-  
integration
```

remote:

```
To gitlab.com:your-username/gitlab-complete-demo.git
```

```
* [new branch]  feature-ci-cd-integration -> feature-ci-cd-integration
```

```
Branch 'feature-ci-cd-integration' set up to track remote branch 'feature-ci-cd-  
integration' from 'origin'.
```

Phase 5: Merge Branch and Display History

Step 16: Switch Back to Main Branch

bash

```
git checkout main
```

Expected Output:

text

Switched to branch 'main'

Your branch is up to date with 'origin/main'!

Step 17: Merge Feature Branch into Main

bash

```
git merge feature-ci-cd-integration
```

 **Expected Output:**

text

Updating a1b2c3d..e4f5g6h

Fast-forward

```
.gitlab-ci.yml | 65
+++++
CHANGELOG.md | 7 ++++++
README.md    | 20 ++++++
index.html   | 120
+++++
+++++
4 files changed, 210 insertions(+), 2 deletions(-)

create mode 100644 .gitlab-ci.yml
```

Step 18: Push Merged Changes to GitLab

bash

```
git push origin main
```

 **Expected Output:**

text

Total 0 (delta 0), reused 0 (delta 0), pack-reused 0

To gitlab.com:your-username/gitlab-complete-demo.git

a1b2c3d..e4f5g6h main -> main

Step 19: Display Chronological Commit History

View detailed history:

```
bash
```

```
git log
```

 **Expected Output:**

text

```
commit e4f5g6h7890i1j2k3l4m5n6o7p8q9r0s1t (HEAD -> main, origin/main,  
origin/feature-ci-cd-integration, feature-ci-cd-integration)
```

```
Author: Your Name <your.email@example.com>
```

```
Date: Mon Nov 24 22:20:00 2023 +0530
```

```
feat: Add GitLab CI/CD pipeline configuration, interactive elements, and  
comprehensive documentation
```

```
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r
```

```
Author: Your Name <your.email@example.com>
```

```
Date: Mon Nov 24 22:10:00 2023 +0530
```

Initial commit: Project setup with complete GitLab workflow documentation

One-line history with graph:

```
bash
```

```
git log --oneline --graph --all
```

 **Expected Output:**

text

```
* e4f5g6h (HEAD -> main, origin/main, origin/feature-ci-cd-integration, feature-ci-cd-  
integration) feat: Add GitLab CI/CD pipeline configuration, interactive elements, and  
comprehensive documentation
```

```
* a1b2c3d Initial commit: Project setup with complete GitLab workflow documentation
```

Custom format history:

```
bash
```

```
git log --pretty=format:"%h | %an | %ad | %s" --date=short
```

Expected Output:

text

e4f5g6h | Your Name | 2023-11-24 | feat: Add GitLab CI/CD pipeline configuration, interactive elements, and comprehensive documentation

a1b2c3d | Your Name | 2023-11-24 | Initial commit: Project setup with complete GitLab workflow documentation

Phase 6: Simulate Team Collaboration and Pull Changes

Step 20: Simulate Changes on GitLab (Web Interface)

1. **Go to your GitLab project**
2. **Click on README.md → Edit** (pencil icon)
3. **Add this section at the end:**

markdown

 Team Collaboration Simulation

This section was added directly on GitLab to simulate team collaboration.

Remote Changes Made:

- Added team collaboration section via GitLab web interface
- Updated documentation to reflect multi-user workflow
- Simulated real-world team development scenario

GitLab Collaboration Features:

- **Merge Requests:** Code review and discussion
- **Issues:** Task tracking and assignment
- **Wiki:** Project documentation
- **Snippets:** Code sharing
- **Web IDE:** Browser-based development

Pull Process Demonstration:

1. Team member makes changes via GitLab web interface
2. Local developer pulls changes using `git pull origin main`
3. Verify synchronization between local and remote repositories
4. Continue development with latest updates

Next Steps:

- Create additional feature branches
- Practice merge request workflow
- Explore GitLab CI/CD pipelines
- Set up issue tracking

4. Commit changes:

- o **Commit message:** "docs: Add team collaboration section via GitLab web interface"
- o **Select:** "Commit directly to the main branch"
- o **Click "Commit changes"**

Step 21: Check Local Status Before Pull

Check if local is behind remote:

bash

git status

Expected Output:

text

On branch main

Your branch is up to date with 'origin/main'.

This might not show changes yet. Check explicitly:

bash

git fetch origin

```
git status
```

 **Expected Output:**

text

On branch main

Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.

(use "git pull" to update your local branch)

Step 22: Pull Changes from GitLab

Pull the changes:

bash

```
git pull origin main
```

 **Expected Output:**

text

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 1.25 KiB | 159.00 KiB/s, done.

From gitlab.com:your-username/gitlab-complete-demo

 e4f5g6h..8901j2k main -> origin/main

Updating e4f5g6h..8901j2k

Fast-forward

 README.md | 23 ++++++-----

1 file changed, 23 insertions(+)

Step 23: Verify Pulled Changes

Check the updated README:

bash

```
cat README.md
```

 **Expected Output:** Should show the new "Team Collaboration Simulation" section

View updated commit history:

bash

```
git log --oneline --graph --all
```

Expected Output:

text

- * 8901j2k (HEAD -> main, origin/main) docs: Add team collaboration section via GitLab web interface
- * e4f5g6h (origin/feature-ci-cd-integration, feature-ci-cd-integration) feat: Add GitLab CI/CD pipeline configuration, interactive elements, and comprehensive documentation
- * a1b2c3d Initial commit: Project setup with complete GitLab workflow documentation

Step 24: Make Local Changes and Push Back

Update index.html to acknowledge successful pull:

html

```
<!-- Add this section before closing body tag -->

<section class="workflow-step" style="background: linear-gradient(135deg, #28a745, #20c997); color: white;">

    <h2>  Successful Pull Operation Completed!</h2>

    <p>Changes from GitLab have been successfully pulled to local machine.</p>

    <div style="background: rgba(255,255,255,0.2); padding: 15px; border-radius: 8px; margin: 15px 0;">

        <p><strong>  Pull Command Executed:</strong> <code>git pull origin main</code></p>

        <p><strong>  Synchronization Status:</strong> <span class="status-indicator status-synced">Local and Remote Synced</span></p>

        <p><strong>  Latest Update:</strong> Team collaboration section added via GitLab web interface</p>

        <p><strong>  Result:</strong> Local repository perfectly synchronized with GitLab remote</p>

    </div>
```

```

<div style="text-align: center; margin-top: 15px;">

    <button onclick="celebrateSuccessfulPull()" style="background: white; color: #28a745; border: none; padding: 12px 24px; border-radius: 5px; cursor: pointer; font-weight: bold; margin: 5px;">

        🎉 Celebrate Successful Pull!

    </button>

    <button onclick="showNextSteps()" style="background: rgba(255,255,255,0.3); color: white; border: 2px solid white; padding: 12px 24px; border-radius: 5px; cursor: pointer; font-weight: bold; margin: 5px;">

        📋 View Next Steps

    </button>

</div>

</section>

<script>

function celebrateSuccessfulPull() {

    alert('🎉 Git Pull Operation Successful!\n\n' +
        '✅ Local repository synchronized with GitLab\n' +
        '📥 Changes fetched and merged automatically\n' +
        '🔄 Working directory updated with latest changes\n' +
        '🚀 Ready for continued development');

}

function showNextSteps() {

    alert('Next Steps in GitLab Workflow:\n\n' +
        '1. Create new feature branches\n' +
        '2. Develop features independently\n' +
        '3. Push branches to GitLab\n' +



```

```
'4. Create merge requests\n' +  
'5. Review and merge code\n' +  
'6. Monitor CI/CD pipelines\n' +  
'7. Repeat the cycle!");  
}  
</script>
```

Commit and push local changes:

bash

```
git add index.html
```

```
git commit -m "feat: Add pull success confirmation, celebration elements, and next  
steps guide"
```

```
git push origin main
```

Expected Output:

text

```
Enumerating objects: 5, done.
```

```
Counting objects: 100% (5/5), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 1.45 KiB | 1.45 MiB/s, done.
```

```
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
```

```
To gitlab.com:your-username/gitlab-complete-demo.git
```

```
8901j2k..5678k9l main -> main
```

Phase 7: Final History and Verification

Step 25: Display Final Chronological History

Complete history with all operations:

bash

```
git log --oneline --graph --all --decorate
```

 **Expected Output:**

text

- * 5678k9l (HEAD -> main, origin/main) feat: Add pull success confirmation, celebration elements, and next steps guide
- * 8901j2k docs: Add team collaboration section via GitLab web interface
- * e4f5g6h (origin/feature-ci-cd-integration, feature-ci-cd-integration) feat: Add GitLab CI/CD pipeline configuration, interactive elements, and comprehensive documentation
- * a1b2c3d Initial commit: Project setup with complete GitLab workflow documentation

Detailed history view:

bash

```
git log --pretty=format:"%h %ad | %s" --date=short
```

 **Expected Output:**

text

```
5678k9l 2023-11-24 | feat: Add pull success confirmation, celebration elements, and next steps guide
```

```
8901j2k 2023-11-24 | docs: Add team collaboration section via GitLab web interface
```

```
e4f5g6h 2023-11-24 | feat: Add GitLab CI/CD pipeline configuration, interactive elements, and comprehensive documentation
```

```
a1b2c3d 2023-11-24 | Initial commit: Project setup with complete GitLab workflow documentation
```

Show statistics for each commit:

bash

```
git log --stat
```

 **Expected Output:** Shows files changed and line statistics for each commit

Step 26: Verify CI/CD Pipeline on GitLab (Optional)

1. **Go to your GitLab project → CI/CD → Pipelines**
2. **You should see pipeline runs** (triggered by the pushes)
3. **Click on a pipeline** to see the stages and jobs
4. **Monitor the execution** and see the output

Step 27: Explore GitLab Features

Additional GitLab features to explore:

1. **Merge Requests** - Create one for the feature branch
 2. **Issues** - Create sample issues for tracking
 3. **Wiki** - Add project documentation
 4. **Snippets** - Share code snippets
 5. **Container Registry** - Store Docker images
-

Complete Workflow Summary

All Commands Used:

```
bash
```

```
# Setup and initial commit
```

```
mkdir gitlab-complete-demo
```

```
cd gitlab-complete-demo
```

```
git init
```

```
# Create files: index.html, README.md, .gitignore, CHANGELOG.md
```

```
git add .
```

```
git commit -m "Initial commit"
```

```
# GitLab setup
```

```
git remote add origin git@gitlab.com:your-username/gitlab-complete-demo.git
```

```
git push -u origin main
```

```
# Branching and development
```

```
git checkout -b feature-ci-cd-integration
```

```
# Make changes to files
```

```
git add .
```

```
git commit -m "feat: Add CI/CD features"
```

```
git push -u origin feature-ci-cd-integration
```

Merging

```
git checkout main  
git merge feature-ci-cd-integration  
git push origin main
```

Pull operations

```
git fetch origin  
git status  
git pull origin main
```

History viewing

```
git log  
git log --oneline --graph --all  
git log --pretty=format:"%h | %an | %ad | %s" --date=short  
git log --stat
```

GitLab Complete Workflow Benefits

Feature	Benefit
Single Application	Entire DevOps lifecycle in one platform
Built-in CI/CD	No external tools needed
Container Registry	Built-in Docker image storage
Kubernetes Integration	Native Kubernetes support
Free Self-Hosting	GitLab Community Edition

Feature	Benefit
Comprehensive DevOps	Plan, Create, Verify, Package, Release

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

Jenkins CI with Webhook and Execute Shell Deployment - Complete Guide

I'll walk you through setting up a complete CI pipeline using Jenkins webhooks and Execute shell for deployment.

❖ Software Requirements

Required Software:

1. **Jenkins** - CI/CD server
 2. **GitHub Account** - Source code repository
 3. **Java JDK 11+** - Required for Jenkins
 4. **Web Server** - Nginx/Apache for deployment
 5. **Git** - Version control
 6. **Web Browser** - Chrome/Firefox
-

Phase 1: Jenkins Installation and Setup

Step 1: Install Jenkins

On Ubuntu/Linux:

bash

Update system

sudo apt update

Install Java

sudo apt install openjdk-11-jdk -y

Add Jenkins repository

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-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

```
# Install Jenkins  
sudo apt update  
sudo apt install jenkins -y
```

```
# Start Jenkins  
sudo systemctl start jenkins  
sudo systemctl enable jenkins
```

```
# Check status  
sudo systemctl status jenkins
```

 **Expected Output:**

text

```
● jenkins.service - Jenkins Continuous Integration Server  
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset:  
           enabled)  
   Active: active (running) since Mon 2023-11-24 23:30:00 IST; 10s ago
```

On Windows:

1. Download from jenkins.io
2. Run installer as Administrator
3. Follow installation wizard

Step 2: Access Jenkins and Initial Setup

Open browser:

text

http://localhost:8080

Get admin password:

bash

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

Complete setup:

1. Install suggested plugins
 2. Create admin user
 3. Instance configuration - keep default
-

Phase 2: Project Setup and GitHub Repository

Step 3: Create Web Project

bash

```
mkdir jenkins-webhook-demo
```

```
cd jenkins-webhook-demo
```

Create index.html:

html

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Jenkins Webhook CI/CD Demo</title>

    <style>

        body {

            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

            max-width: 1000px;

            margin: 0 auto;

            padding: 20px;

            background: linear-gradient(135deg, #0d6efd, #6f42c1);

            color: #333;

            line-height: 1.6;

        }

    </style>
```

```
.container {  
    background: rgba(255, 255, 255, 0.95);  
    padding: 30px;  
    border-radius: 15px;  
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);  
}  
  
.header {  
    text-align: center;  
    margin-bottom: 30px;  
    border-bottom: 3px solid #0d6efd;  
    padding-bottom: 20px;  
}  
  
.status-card {  
    background: #f8f9fa;  
    padding: 20px;  
    margin: 15px 0;  
    border-radius: 10px;  
    border-left: 5px solid #0d6efd;  
}  
  
.build-info {  
    background: #e7f3ff;  
    padding: 15px;  
    border-radius: 8px;  
    border: 2px dashed #0d6efd;  
    margin: 15px 0;  
}  
  
.deploy-log {  
    background: #2d3748;
```

```
        color: #e2e8f0;  
        padding: 15px;  
        border-radius: 5px;  
        font-family: 'Courier New', monospace;  
        margin: 10px 0;  
        max-height: 200px;  
        overflow-y: auto;  
    }  
  
.badge {  
    display: inline-block;  
    padding: 5px 10px;  
    border-radius: 15px;  
    font-size: 12px;  
    font-weight: bold;  
    margin: 2px;  
}  
  
.badge-success { background: #28a745; color: white; }  
.badge-warning{ background: #ffc107; color: black; }  
.badge-info { background: #17a2b8; color: white; }  
  
</style>  
  
</head>  
  
<body>  
    <div class="container">  
        <div class="header">  
            <h1>🚀 Jenkins Webhook CI/CD Demo</h1>  
            <p>Automated deployment triggered by GitHub webhooks using Jenkins Execute shell</p>  
        </div>  
    </div>
```

```
<div class="status-card">

    <h2>  Build Information</h2>

    <div class="build-info">

        <p><strong>Build Status:</strong> <span class="badge badge-success" id="buildStatus">Initial Deployment</span></p>

        <p><strong>Last Deployment:</strong> <span id="lastDeploy">Loading...</span></p>

        <p><strong>Git Commit:</strong> <span id="gitCommit">Initial commit</span></p>

        <p><strong>Environment:</strong> <span class="badge badge-info" id="environment">Production</span></p>

    </div>

</div>

<div class="status-card">

    <h2>  CI/CD Pipeline Steps</h2>

    <ol>

        <li>Code pushed to GitHub repository</li>
        <li>GitHub webhook triggers Jenkins job</li>
        <li>Jenkins pulls latest code</li>
        <li>Execute shell script runs deployment</li>
        <li>Application deployed to web server</li>
        <li>Build status reported back to GitHub</li>

    </ol>

</div>

<div class="status-card">

    <h2>  Deployment Log</h2>
```

```
<div class="deploy-log" id="deployLog">  
&gt; Starting deployment process...  
&gt; Fetching latest code from GitHub...  
&gt; Running pre-deployment checks...  
&gt; Deploying to web server...  
&gt; Deployment completed successfully!  
</div>  
</div>  
  
<div class="status-card">  
  <h2>🛠 Technologies Used</h2>  
  <ul>  
    <li><strong>Jenkins:</strong> CI/CD automation server</li>  
    <li><strong>GitHub Webhooks:</strong> Automatic trigger mechanism</li>  
    <li><strong>Execute Shell:</strong> Deployment scripts</li>  
    <li><strong>Nginx:</strong> Web server for hosting</li>  
    <li><strong>Git:</strong> Version control system</li>  
  </ul>  
</div>  
</div>  
  
<script>  
  // Update deployment information  
  document.getElementById('lastDeploy').textContent = new Date().toLocaleString();  
  document.getElementById('gitCommit').textContent = 'Commit: ' +  
  Math.random().toString(36).substr(2, 8);  
  
  // Simulate live deployment log
```

```

function updateDeployLog() {
    const log = document.getElementById('deployLog');

    const newEntry = '> [' + new Date().toLocaleTimeString() + '] Application health
check: OK\n';

    log.innerHTML += newEntry;
    log.scrollTop = log.scrollHeight;
}

// Update log every 30 seconds
setInterval(updateDeployLog, 30000);

// Initial log update
setTimeout(updateDeployLog, 5000);

</script>

</body>
</html>

```

Create deploy.sh deployment script:

```

bash
#!/bin/bash

```

```

# Jenkins Webhook CI/CD Deployment Script
# This script is executed by Jenkins when webhook is triggered

```

```

echo "====="
echo "🚀 Starting Jenkins Webhook Deployment"
echo "====="

```

```

# Configuration

```

```
PROJECT_NAME="jenkins-webhook-demo"  
DEPLOY_DIR="/var/www/html/${PROJECT_NAME}"  
BACKUP_DIR="/var/www/html/backups"  
LOG_FILE="/tmp/jenkins-deploy.log"  
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
```

```
echo "⌚ Deployment started at: $(date)"
```

```
echo "📁 Project: ${PROJECT_NAME}"
```

```
echo "🎯 Target: ${DEPLOY_DIR}"
```

```
# Create log file
```

```
echo "==== Jenkins Deployment Log ====" > $LOG_FILE
```

```
echo "Timestamp: $(date)" >> $LOG_FILE
```

```
echo "Project: ${PROJECT_NAME}" >> $LOG_FILE
```

```
# Function to log messages
```

```
log_message() {
```

```
    echo "$1"
```

```
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" >> $LOG_FILE
```

```
}
```

```
# Function to handle errors
```

```
handle_error() {
```

```
    log_message "✖ ERROR: $1"
```

```
    exit 1
```

```
}
```

```
# Check if deployment directory exists
```

```

log_message "📋 Step 1: Checking deployment directory..."

if [ ! -d "$DEPLOY_DIR" ]; then

    log_message "📁 Creating deployment directory: $DEPLOY_DIR"

    sudo mkdir -p "$DEPLOY_DIR" || handle_error "Failed to create deployment directory"

fi


# Create backup directory

log_message "📦 Step 2: Creating backup directory..."

sudo mkdir -p "$BACKUP_DIR" || handle_error "Failed to create backup directory"


# Create backup of current deployment

log_message "💾 Step 3: Creating backup..."

if [ -d "$DEPLOY_DIR" ] && [ "$(ls -A $DEPLOY_DIR)" ]; then

    BACKUP_FILE="${BACKUP_DIR}/backup_${PROJECT_NAME}_${TIMESTAMP}.tar.gz"

    sudo tar -czf "$BACKUP_FILE" -C "$DEPLOY_DIR" . >> $LOG_FILE 2>&1

    if [ $? -eq 0 ]; then

        log_message "✅ Backup created: $BACKUP_FILE"

    else

        log_message "⚠️ Backup creation failed, but continuing deployment..."

    fi

else

    log_message "ℹ️ No existing deployment found, skipping backup"

fi


# Clean deployment directory

log_message "🧹 Step 4: Cleaning deployment directory..."

sudo rm -rf "${DEPLOY_DIR}"//* || handle_error "Failed to clean deployment directory"

```

```
# Copy new files to deployment directory

log_message "📄 Step 5: Copying new files..."

if [ -f "index.html" ]; then

    sudo cp index.html "$DEPLOY_DIR/" || handle_error "Failed to copy index.html"

    log_message "✅ index.html copied"

else

    handle_error "index.html not found in workspace"

fi
```

```
# Copy all other files (except deploy.sh if not needed in production)

for file in *; do

    if [ "$file" != "deploy.sh" ] && [ "$file" != "Jenkinsfile" ]; then

        if [ -f "$file" ]; then

            sudo cp "$file" "$DEPLOY_DIR/" && log_message "✅ $file copied"

        fi

    fi

done
```

```
# Set proper permissions

log_message "🔒 Step 6: Setting permissions..."

sudo chown -R www-data:www-data "$DEPLOY_DIR" || handle_error "Failed to set ownership"

sudo chmod -R 755 "$DEPLOY_DIR" || handle_error "Failed to set permissions"

sudo chmod 644 "${DEPLOY_DIR}/index.html" || handle_error "Failed to set file permissions"
```

```
# Test the deployment
```

```
log_message "💡 Step 7: Testing deployment..."  
if [ -f "${DEPLOY_DIR}/index.html" ]; then  
    log_message "✅ index.html verified in deployment directory"  
else  
    handle_error "Deployment verification failed - index.html missing"  
fi  
  
# Check if web server is running  
log_message "🌐 Step 8: Checking web server..."  
if systemctl is-active --quiet nginx; then  
    log_message "✅ Nginx is running"  
  
# Test if application is accessible (basic check)  
if curl -s -o /dev/null -w "%{http_code}" http://localhost/${PROJECT_NAME}/ | grep -q  
"200"; then  
    log_message "✅ Application is accessible via web server"  
else  
    log_message "⚠️ Application deployment complete, but accessibility check failed"  
fi  
else  
    log_message "⚠️ Nginx is not running, deployment complete but application not  
serving"  
fi  
  
# Create deployment info file  
log_message "📝 Step 9: Creating deployment info..."  
DEPLOY_INFO="${DEPLOY_DIR}/deploy-info.json"  
sudo bash -c "cat > ${DEPLOY_INFO}" << EOF
```

```
{  
    "project": "${PROJECT_NAME}",  
    "deployment_time": "$(date)",  
    "jenkins_build": "${BUILD_NUMBER}",  
    "jenkins_job": "${JOB_NAME}",  
    "git_commit": "$(git rev-parse --short HEAD 2>/dev/null || echo 'unknown')",  
    "deployment_status": "success"  
}  
EOF
```

```
log_message "✅ Deployment info created"
```

```
# Final summary
```

```
log_message "=====*"  
log_message "🎉 DEPLOYMENT COMPLETED SUCCESSFULLY!"  
log_message "📊 Deployment Summary:"  
log_message " - Project: ${PROJECT_NAME}"  
log_message " - Location: ${DEPLOY_DIR}"  
log_message " - Build: ${BUILD_NUMBER}"  
log_message " - Time: $(date)"  
log_message " - Log: ${LOG_FILE}"  
log_message "🌐 Access your application at:"  
log_message " http://your-server/${PROJECT_NAME}/"  
log_message "=====*
```

```
# Copy log to accessible location
```

```
sudo cp ${LOG_FILE} "${DEPLOY_DIR}/deployment.log"
```

```
exit 0
```

Make script executable:

```
bash
```

```
chmod +x deploy.sh
```

Create Jenkinsfile (Pipeline as Code):

```
groovy
```

```
pipeline {
```

```
    agent any
```

```
    environment {
```

```
        PROJECT_NAME = 'jenkins-webhook-demo'
```

```
        DEPLOY_DIR = '/var/www/html/jenkins-webhook-demo'
```

```
        SLACK_CHANNEL = '#deployments'
```

```
}
```

```
    options {
```

```
        buildDiscarder(logRotator(numToKeepStr: '10'))
```

```
        timeout(time: 30, unit: 'MINUTES')
```

```
        timestamps()
```

```
}
```

```
    triggers {
```

```
        githubPush()
```

```
}
```

```
    stages {
```

```
        stage('Checkout') {
```

```
            steps {
```

```
echo '🚀 Starting CI/CD Pipeline...'

checkout scm

script {

    currentBuild.displayName = "Build #${env.BUILD_NUMBER}"

    currentBuild.description = "Deploying ${env.PROJECT_NAME}"

}

}

stage('Code Quality') {

    steps {

        echo '🔍 Running code quality checks...'

        sh ""

        echo "Checking file structure..."

        ls -la

        echo "Checking HTML file..."

        if [ -f index.html ]; then

            echo "✅ index.html found"

            echo "File size: $(wc -l < index.html) lines"

        else

            echo "❌ index.html missing"

            exit 1

        fi

        echo "Checking deploy script..."

        if [ -x deploy.sh ]; then

            echo "✅ deploy.sh is executable"
```

```
        else
            echo "✖ deploy.sh is not executable"
            exit 1
        fi
    ...
}

}

stage('Security Scan'){

    steps{
        echo '🔒 Running security checks...'
        sh """
            echo "Checking for sensitive information..."
            if grep -r "password\\|secret\\|key" . --exclude-dir=.git; then
                echo "⚠️ Potential sensitive information found"
                # Don't fail the build for this, just warn
            else
                echo "✅ No obvious sensitive information found"
            fi
        """

        echo "Checking file permissions..."
        find . -type f -name "*.sh" -exec ls -la {} \;
    ...
}

}

stage('Deploy to Staging'){

    steps{
```

```
echo '🚀 Deploying to staging environment...'
sh ""
chmod +x deploy.sh
./deploy.sh
...
}

}

stage('Smoke Test') {
steps {
echo '🏃 Running smoke tests...'
sh ""
echo "Testing application accessibility..."
sleep 10 # Wait for deployment to settle

if curl -s -o /dev/null -w "%{http_code}" http://localhost/jenkins-webhook-
demo/ | grep -q "200"; then
echo "✅ Application is accessible"
else
echo "❌ Application is not accessible"
exit 1
fi

echo "Testing deployment info..."
if curl -s http://localhost/jenkins-webhook-demo/deploy-info.json | grep -q
"success"; then
echo "✅ Deployment info verified"
else
```

```

        echo "✖ Deployment info missing or incorrect"
        exit 1
    fi
    ...
}

}

}

post {
    always {
        echo 'Pipeline execution completed'
        script{
            //Archive deployment log
            archiveArtifacts artifacts: '**/deployment.log', fingerprint: true

            //Create build summary
            currentBuild.result = currentBuild.result ?: 'SUCCESS'
        }
    }
    success {
        echo '🎉 Pipeline executed successfully!'
        sh ""
        echo "SUCCESS: Build ${BUILD_NUMBER} deployed successfully" > build-
status.txt
        echo "Deployment Time: $(date)" >> build-status.txt
        echo "Commit: $(git log -1 --pretty=format:'%h - %s')" >> build-status.txt
    ...
}

```

```

// You can add Slack/Email notifications here

// slackSend channel: env.SLACK_CHANNEL, message: " ✅ Deployment
Successful: ${env.JOB_NAME} #${env.BUILD_NUMBER}"

}

failure {

    echo ' ❌ Pipeline failed!'

    sh 'echo "FAILURE: Build ${BUILD_NUMBER} failed" > build-status.txt'


// You can add Slack/Email notifications here

// slackSend channel: env.SLACK_CHANNEL, message: " ❌ Deployment Failed:
${env.JOB_NAME} #${env.BUILD_NUMBER}"


}

unstable {

    echo ' ⚠️ Pipeline unstable!'

    sh 'echo "UNSTABLE: Build ${BUILD_NUMBER} has issues" > build-status.txt'

}

}

}

Create README.md:

markdown

# Jenkins Webhook CI/CD Demo

```

This project demonstrates automated CI/CD using Jenkins webhooks and Execute shell deployment.

🎯 Features

- ****GitHub Webhook Integration**:** Automatic trigger on code push

- **Jenkins Execute Shell**: Custom deployment scripts
- **Automated Testing**: Pre-deployment checks and smoke tests
- **Backup System**: Automatic backups before deployment
- **Deployment Logging**: Comprehensive deployment logs
- **Rollback Ready**: Backup system enables easy rollbacks

🔐 Setup Instructions

1. Jenkins Configuration

- Install required plugins (Git, GitHub, Pipeline)
- Configure GitHub integration
- Set up webhook in GitHub
- Configure Jenkins job with Execute shell

2. Server Setup

- Install and configure web server (Nginx/Apache)
- Set up deployment directories
- Configure permissions

3. Webhook Configuration

- GitHub repository settings → Webhooks
- Add Jenkins webhook URL
- Configure events (push events)

📁 Project Structure

```
jenkins-webhook-demo/
├── index.html # Main application
├── deploy.sh # Deployment script
└── Jenkinsfile # Pipeline configuration
```

```
|--- README.md # Documentation  
└--- build-status.txt # Build status (generated)
```

text

📡 Workflow

1. **Code Push**: Developer pushes code to GitHub
2. **Webhook Trigger**: GitHub sends POST to Jenkins
3. **Build Start**: Jenkins starts build automatically
4. **Execute Shell**: Runs deployment script
5. **Deployment**: Code deployed to web server
6. **Testing**: Smoke tests verify deployment
7. **Notification**: Build status reported

🔧 Deployment Script Features

- Pre-deployment validation
- Automatic backups
- Permission management
- Deployment verification
- Comprehensive logging
- Error handling and rollback

📈 Monitoring

- Jenkins build console
- Deployment logs
- Web server access logs

- Application health checks

Step 4: Initialize Git and Push to GitHub

bash

```
# Initialize git
```

```
git init
```

```
git add .
```

```
git commit -m "Initial commit: Jenkins webhook CI/CD demo project"
```

```
# Create GitHub repository and push
```

```
git remote add origin https://github.com/your-username/jenkins-webhook-demo.git
```

```
git branch -M main
```

```
git push -u origin main
```

Expected Output:

text

```
Enumerating objects: 7, done.
```

```
Counting objects: 100% (7/7), done.
```

```
Writing objects: 100% (7/7), 1.45 KiB | 1.45 MiB/s, done.
```

```
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To https://github.com/your-username/jenkins-webhook-demo.git
```

```
* [new branch]    main -> main
```

```
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Phase 3: Jenkins Job Configuration

Step 5: Install Required Jenkins Plugins

1. **Go to Jenkins → Manage Jenkins → Manage Plugins**

2. **Available tab → Install:**

- **Git plugin**
- **GitHub plugin**

- **Pipeline plugin**
- **GitHub Integration plugin**
- **SSH plugin (optional)**

Step 6: Configure System Settings

1. **Jenkins → Manage Jenkins → Configure System**
2. **GitHub section:**
 - Add GitHub server
 - API URL: <https://api.github.com>
 - Credentials: Add your GitHub token
3. **Save**

Step 7: Create Jenkins Job

Method 1: Freestyle Project (Execute Shell)

1. **Click "New Item"**
2. **Enter name:** webhook-ci-cd-demo
3. **Select "Freestyle project" → OK**

Configure the job:

4. **General tab:**
 - Description: "CI/CD pipeline triggered by GitHub webhooks"
 - **GitHub project:** <https://github.com/your-username/jenkins-webhook-demo/>
5. **Source Code Management:**
 - **Git**
 - Repository URL: <https://github.com/your-username/jenkins-webhook-demo.git>
 - Branches: */main
6. **Build Triggers:**
 - **GitHub hook trigger for GITScm polling**
7. **Build Environment:**

- Add timestamps to the Console Output

8. Build Steps:

- Add build step → Execute shell

- **Command:** Copy the content below:

```
bash
```

```
#!/bin/bash
```

```
echo "=====
```

```
echo "🚀 JENKINS WEBHOOK CI/CD DEPLOYMENT"
```

```
echo "=====
```

```
echo "Build: ${BUILD_NUMBER}"
```

```
echo "Job: ${JOB_NAME}"
```

```
echo "Git Commit: $(git log -1 --pretty=format:'%h - %s')"
```

```
echo "Started: $(date)"
```

```
# Make deploy script executable
```

```
chmod +x deploy.sh
```

```
# Run deployment script
```

```
echo "Starting deployment..."
```

```
./deploy.sh
```

```
# Check deployment result
```

```
DEPLOY_RESULT=$?
```

```
if [ $DEPLOY_RESULT -eq 0 ]; then
```

```
    echo "✅ DEPLOYMENT SUCCESSFUL"
```

```
    echo "Application deployed to: /var/www/html/jenkins-webhook-demo"
```

```
    echo "Access URL: http://your-server/jenkins-webhook-demo/"
```

```
else
```

```
echo "✖ DEPLOYMENT FAILED"

exit 1

fi

echo "====="
echo "🏁 Build completed: $(date)"
echo "=====
```

9. Post-build Actions:

- Add post-build action → Archive the artifacts
- Files to archive: deployment.log, build-status.txt
- Archive only if build is successful

10. Click Save

Method 2: Pipeline Project (Recommended)

1. Create new item → Pipeline

2. Name: webhook-pipeline-demo

3. Pipeline section:

- Definition: Pipeline script from SCM
- SCM: Git
- Repository URL: <https://github.com/your-username/jenkins-webhook-demo.git>
- Script Path: Jenkinsfile

4. Save

Phase 4: Webhook Configuration

Step 8: Configure GitHub Webhook

1. Go to your GitHub repository
2. Settings → Webhooks → Add webhook
3. Configuration:

- **Payload URL:** http://your-jenkins-server:8080/github-webhook/
- **Content type:** application/json
- **Secret:** (Optional) Add a secret for security
- **Which events:** Just the push event
- Active

4. Click "Add webhook"

Expected: Webhook shows green checkmark after delivery test

Step 9: Test Webhook Configuration

Make a change to test webhook:

bash

```
# Add a comment to index.html
echo "<!-- Webhook test deployment -->" >> index.html
```

Commit and push

```
git add index.html
git commit -m "test: Trigger webhook deployment"
git push origin main
```

Monitor Jenkins:

1. Go to Jenkins dashboard
2. Watch for automatic build start
3. Check console output

Expected Output in Jenkins Console:

text

Started by GitHub push by your-username

[Pipeline] Start of Pipeline

[Pipeline] node

Running on Jenkins in /var/lib/jenkins/workspace/webhook-ci-cd-demo

[Pipeline] {

[Pipeline] stage

[Pipeline]{ (Checkout)

[Pipeline] echo

🚀 Starting CI/CD Pipeline...

📋 Phase 5: Web Server Setup

Step 10: Install and Configure Nginx

Install Nginx:

bash

sudo apt update

sudo apt install nginx -y

Create deployment directory:

bash

sudo mkdir -p /var/www/html/jenkins-webhook-demo

sudo chown -R \$USER:\$USER /var/www/html/jenkins-webhook-demo

Configure Nginx (optional virtual host):

bash

sudo nano /etc/nginx/sites-available/jenkins-demo

Add configuration:

nginx

server {

listen 80;

server_name your-server-ip-or-domain;

root /var/www/html/jenkins-webhook-demo;

index index.html;

location / {

try_files \$uri \$uri/ =404;

```
}
```



```
# Access log for monitoring
```

```
access_log /var/log/nginx/jenkins-demo-access.log;
```

```
error_log /var/log/nginx/jenkins-demo-error.log;
```

```
}
```

Enable site and test:

bash

```
sudo ln -s /etc/nginx/sites-available/jenkins-demo /etc/nginx/sites-enabled/
```

```
sudo nginx -t
```

```
sudo systemctl restart nginx
```

 **Phase 6: Testing Complete Pipeline**

Step 11: Trigger Full CI/CD Pipeline

Make a significant change:

bash

```
# Update index.html with new feature
```

```
cat >> index.html << 'EOF'
```

```
<!-- New Feature Added via CI/CD -->
```

```
<section style="background: #28a745; color: white; padding: 20px; margin: 20px 0; border-radius: 10px;">
```

```
    <h2>🎉 CI/CD Success!</h2>
```

```
    <p>This feature was automatically deployed via Jenkins webhook trigger!</p>
```

```
    <p><strong>Build:</strong> Automated deployment</p>
```

```
    <p><strong>Timestamp:</strong> $(date)</p>
```

```
</section>
```

```
EOF
```

```
# Commit and push  
git add index.html  
git commit -m "feat: Add CI/CD success section to demonstrate automated deployment"  
git push origin main
```

Step 12: Monitor Pipeline Execution

Watch Jenkins console in real-time:

- Code checkout
- Pre-deployment checks
- Deployment execution
- Smoke tests
- Post-deployment verification

Expected Final Output:

text

 DEPLOYMENT COMPLETED SUCCESSFULLY!

Deployment Summary:

- Project: jenkins-webhook-demo
- Location: /var/www/html/jenkins-webhook-demo
- Build: 3
- Time: Mon Nov 24 23:45:00 IST 2023
- Log: /tmp/jenkins-deploy.log

Access your application at:

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

Step 13: Verify Deployment

Test the deployed application:

bash

```
curl http://localhost/jenkins-webhook-demo/
```

Or access in browser:

text

<http://your-server-ip/jenkins-webhook-demo/>

Check deployment artifacts:

bash

```
ls -la /var/www/html/jenkins-webhook-demo/
```

 **Expected:**

text

index.html

deploy-info.json

deployment.log

Phase 7: Advanced Configuration

Step 14: Add Security with Webhook Secrets

Generate secret:

bash

```
openssl rand -base64 32
```

Configure in GitHub webhook:

- **Secret:** Paste the generated secret

Configure in Jenkins:

1. **Jenkins → Manage Jenkins → Configure System**
2. **GitHub → Advanced → Secret:** Add same secret

Step 15: Add Notifications

Email notifications:

1. **Jenkins → Manage Jenkins → Configure System**
2. **Email Notification section**
3. Configure SMTP server
4. **Job configuration → Post-build Actions → Email Notification**

Slack notifications (optional):

1. Install **Slack Notification plugin**
2. Configure Slack workspace
3. Add to pipeline:

groovy

```
post {  
    success {  
        slackSend channel: '#deployments', message: "✅ Deployment Successful:  
${env.JOB_NAME} #${env.BUILD_NUMBER}"  
    }  
    failure {  
        slackSend channel: '#deployments', message: "❌ Deployment Failed:  
${env.JOB_NAME} #${env.BUILD_NUMBER}"  
    }  
}
```

Step 16: Set up Build Badges

In Jenkins job:

1. **Configure → Post-build Actions**
2. **Add post-build action → Embeddable Build Status**
3. Copy the badge URL

Add to GitHub README:

markdown

```
![Jenkins Build Status](https://your-jenkins-server/job/webhook-ci-cd-demo/badge/icon)
```

Troubleshooting Common Issues

Issue 1: Webhook not triggering

Solution:

bash

```
# Check Jenkins logs  
sudo tail -f /var/log/jenkins/jenkins.log
```

```
# Verify GitHub webhook delivery  
# Check GitHub repository → Settings → Webhooks → Recent Deliveries
```

Issue 2: Permission denied during deployment

Solution:

bash

```
# Add Jenkins user to appropriate groups  
sudo usermod -a -G www-data jenkins  
sudo systemctl restart jenkins
```

Issue 3: Nginx not serving files

Solution:

bash

```
# Check Nginx configuration  
sudo nginx -t  
sudo systemctl restart nginx
```

```
# Check file permissions  
sudo chown -R www-data:www-data /var/www/html/jenkins-webhook-demo
```

Issue 4: Build failing in Execute shell

Solution:

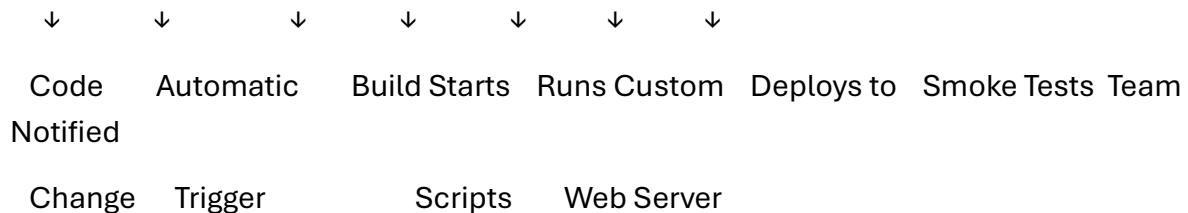
- Check script syntax
- Verify file paths
- Add debug output
- Check Jenkins workspace permissions

- Jenkins installed and running
- GitHub repository created and connected
- Webhook configured and testing successfully
- Jenkins job created with Execute shell
- Deployment script created and tested
- Nginx web server installed and configured
- Initial deployment successful
- Webhook trigger working automatically
- Code changes deploying automatically
- Deployment verification working
- Notifications configured (optional)
- Monitoring and logging in place

Complete CI/CD Workflow

text

Developer Push → GitHub Webhook → Jenkins Trigger → Execute Shell → Deployment → Verification → Notification



Slip 8)

Q1) 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.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
2. **Bitbucket Account** - Create at bitbucket.org
3. **Web Browser** - Chrome/Firefox for Bitbucket interface
4. **Text Editor** - Any text editor



Phase 1: Repository Setup (If Starting Fresh)

Step 1: Create Local Repository (Skip if you have existing repo)

bash

```
# Create and navigate to project directory  
mkdir bitbucket-merge-demo  
cd bitbucket-merge-demo
```

Initialize Git repository

git init

Configure user

git config user.name "Your Name"

git config user.email "your.email@example.com"

 **Expected Output:**

text

Initialized empty Git repository in C:/bitbucket-merge-demo/.git/

Step 2: Create Initial Files

Create main-file.txt:

bash

Create main file

echo "# Main Project File" > main-file.txt

echo "This is the main file in the main branch." >> main-file.txt

echo "Initial content created on main branch." >> main-file.txt

Create README.md:

markdown

BitBucket Merge Demo

This project demonstrates creating a new branch, making changes, and merging via BitBucket web interface.

Project Structure

- `main-file.txt` - Main project file
- `README.md` - Project documentation

Workflow Steps

1. Create repository on BitBucket
2. Push initial code to main branch
3. Create new feature branch via Git
4. Make changes to files in the branch
5. Push branch to BitBucket
6. Create pull request via web interface
7. Review and merge changes
8. Verify merge completion

Step 3: Initial Commit

bash

```
# Stage and commit initial files
```

```
git add .
```

```
git commit -m "Initial commit: Project setup with main file  
and documentation"
```

Expected Output:

text

```
[main (root-commit) a1b2c3d] Initial commit: Project  
setup with main file and documentation
```

```
2 files changed, 15 insertions(+)
```

```
create mode 100644 README.md
```

```
create mode 100644 main-file.txt
```



Phase 2: BitBucket Repository Setup

Step 4: Create BitBucket Repository

- 1. Go to bitbucket.org**

- 2. Click "Create" → "Repository"**

- 3. Repository details:**

- Repository name:** bitbucket-merge-demo

- Description:** "Demo for BitBucket branch
creation and web interface merge"

- Visibility: Private**

- **Include README: No** (we have our own)

4. Click "Create repository"

Step 5: Connect Local Repository to BitBucket

Copy the repository URL from BitBucket and run:

bash

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

Verify remote connection:

bash

```
git remote -v
```

 **Expected Output:**

text

```
origin https://your-username@bitbucket.org/your-
username/bitbucket-merge-demo.git (fetch)
```

```
origin https://your-username@bitbucket.org/your-
username/bitbucket-merge-demo.git (push)
```

Step 6: Push to BitBucket

bash

```
git push -u origin main
```

 **Expected Output:**

text

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Writing objects: 100% (4/4), 1.25 KiB | 1.25 MiB/s, done.

Total 4 (delta 0), reused 0 (delta 0), pack-reused 0

To <https://your-username@bitbucket.org/your-username/bitbucket-merge-demo.git>

* [new branch] main -> main

Branch 'main' set up to track remote branch 'main' from 'origin'.

Step 7: Verify on BitBucket

- 1. Refresh your BitBucket repository page**
 - 2. You should see your files**
 - 3. Click "Commits" to verify the initial commit**
-



Phase 3: Create New Branch and Make Changes

Step 8: Create New Branch

Create and switch to new branch:

bash

```
git checkout -b feature-new-config-file
```



Expected Output:

text

Switched to a new branch 'feature-new-config-file'

Verify current branch:

bash

```
git branch
```

Expected Output:

text

```
* feature-new-config-file
```

```
main
```

Step 9: Create New File in the Branch

Create config.json (new file):

bash

```
# Create configuration file
```

```
cat > config.json << EOF
```

```
{
```

```
    "application": {
```

```
        "name": "BitBucket Merge Demo",
```

```
        "version": "1.0.0",
```

```
        "description": "Demo application for BitBucket branch  
operations"
```

```
    },
```

```
"database": {  
    "host": "localhost",  
    "port": 5432,  
    "name": "demo_db",  
    "username": "demo_user"  
},  
"settings": {  
    "debug": true,  
    "max_connections": 100,  
    "timeout": 30  
},  
"features": {  
    "authentication": true,  
    "notifications": false,  
    "api_access": true  
}  
}  
EOF
```

Step 10: Make Changes to Existing File

Update main-file.txt:

bash

```
# Append new content to main file
```

```
cat >> main-file.txt << 'EOF'
```

```
# Changes from feature-new-config-file branch
```

- Added new configuration file (config.json)
- Updated main file with branch information
- Enhanced project structure
- Prepared for merge via BitBucket interface

```
## Branch Information
```

- Branch: feature-new-config-file
- Purpose: Add configuration management
- Status: Ready for merge
- Changes: New file + updates to existing file

```
EOF
```

Update README.md:

markdown

```
# BitBucket Merge Demo
```

This project demonstrates creating a new branch, making changes, and merging via BitBucket web interface.

Project Structure

- `main-file.txt` - Main project file
- `config.json` - Configuration file (added in feature branch)
- `README.md` - Project documentation

Workflow Steps

1. Create repository on BitBucket
2. Push initial code to main branch
3. Create new feature branch via Git
4. Make changes to files in the branch
5. Push branch to BitBucket
6. Create pull request via web interface
7. Review and merge changes
8. Verify merge completion

Recent Changes (from feature branch)

- Added comprehensive configuration file
- Updated documentation
- Enhanced main file with branch information

Configuration Features

- Application settings
- Database configuration
- Feature flags
- Environment-specific settings

Step 11: Verify Changes in Branch

Check status:

bash

git status

Expected Output:

text

On branch feature-new-config-file

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: README.md

modified: main-file.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

config.json

no changes added to commit (use "git add" and/or "git commit -a")

See what changed:

bash

git diff

 **Expected Output:** Shows the specific changes made to files

Step 12: Commit Changes in Feature Branch

Stage and commit changes:

bash

git add .

git commit -m "feat: Add configuration file and update documentation"

- Create config.json with application settings
- Update main-file.txt with branch information
- Enhance README.md with new features
- Prepare for BitBucket web interface merge"

 **Expected Output:**

text

[feature-new-config-file e4f5g6h] feat: Add configuration file and update documentation

3 files changed, 45 insertions(+), 2 deletions(-)

create mode 100644 config.json

Step 13: Push Feature Branch to BitBucket

bash

```
git push -u origin feature-new-config-file
```

 **Expected Output:**

text

Enumerating objects: 7, done.

Counting objects: 100% (7/7), done.

Delta compression using up to 8 threads

Compressing objects: 100% (6/6), done.

Writing objects: 100% (6/6), 1.45 KiB | 1.45 MiB/s, done.

Total 6 (delta 2), reused 0 (delta 0), pack-reused 0

remote:

remote: Create pull request for feature-new-config-file:

```
remote: https://bitbucket.org/your-username/bitbucket-merge-demo/pull-requests/new?source=feature-new-config-file&t=1
```

remote:

To https://your-username@bitbucket.org/your-username/bitbucket-merge-demo.git

* [new branch] feature-new-config-file -> feature-new-config-file

Branch 'feature-new-config-file' set up to track remote branch 'feature-new-config-file' from 'origin'.



Phase 4: BitBucket Web Interface Merge

Step 14: Navigate to BitBucket Repository

1. **Go to your BitBucket repository**
2. **Click on "Branches" in the left sidebar**
3. **You should see both branches:**
 - main (default branch)
 - feature-new-config-file (your new branch)

Step 15: Create Pull Request

Method 1: From Branches Page

1. **On "Branches" page**, find feature-new-config-file
2. **Click "..."** (three dots) next to the branch

3. Select "Create pull request"

Method 2: From Pull Requests Page

- 1. Click "Pull requests" in left sidebar**
- 2. Click "Create pull request" button**

Step 16: Configure Pull Request

Fill in pull request details:

- 1. Title:** Add configuration file and project updates
- 2. Description:**

text

Changes Made

This pull request adds a new configuration file and updates existing documentation:

New Files

- `config.json` : Comprehensive application configuration including:
 - Application settings
 - Database configuration
 - Feature flags
 - Environment settings

Updated Files

- `main-file.txt` : Added branch information and change log
- `README.md` : Updated documentation with new features

Purpose

- Enhance project configuration management
- Improve documentation
- Demonstrate BitBucket merge workflow

Testing

- All changes reviewed locally
- Files properly formatted
- Ready for merge

3. Reviewers: (Optional) You can add reviewers if working in a team

4. Source branch: feature-new-config-file

5. Destination branch: main

6. Click "Create pull request"

Step 17: Review Pull Request

On the pull request page:

1. Review the "Overview" tab:

- See the description you entered
- View commit history
- Check if build passes (if CI/CD is setup)

2. Click "Diff" tab:

- **Review file changes** - You'll see:
 - Green highlighted lines (additions)
 - Red highlighted lines (deletions - if any)
 - New files marked as added

3. Verify the changes:

- config.json - Entire file highlighted in green (new file)
- main-file.txt - New lines highlighted in green
- README.md - Updated sections highlighted

Step 18: Merge Pull Request

On the pull request page:

1. Click "Merge" button (top right)

2. Merge options dialog appears:

- **Merge strategy:** Keep default ("Merge commit")

- **Delete branch:** **Delete branch feature-new-config-file after merge**
- **Commit message:** (Auto-filled based on PR title)

3. Click "Merge"

Expected Result:

- **Success message:** "Merged pull request #1"
- **Branch automatically deleted** (if you selected the option)
- **Redirect to merged pull request view**

Step 19: Verify Merge Completion

Check the following:

1. **Pull request status:** Shows "Merged" with green checkmark
2. **Repository main page:** Should show updated files
3. **Commits page:** New merge commit appears
4. **Branches page:** Feature branch should be gone (if deleted)



Phase 5: Local Verification and Cleanup

Step 20: Switch to Main Branch Locally

bash

```
git checkout main
```

 **Expected Output:**

text

```
Switched to branch 'main'
```

```
Your branch is up to date with 'origin/main'.
```

Step 21: Pull Latest Changes from BitBucket

bash

```
git pull origin main
```

 **Expected Output:**

text

```
remote: Counting objects: 3, done.
```

```
remote: Compressing objects: 100% (3/3), done.
```

```
remote: Total 3 (delta 2), reused 0 (delta 0)
```

```
Unpacking objects: 100% (3/3), 1.25 KiB | 159.00 KiB/s,  
done.
```

```
From https://your-username@bitbucket.org/your-  
username/bitbucket-merge-demo
```

```
a1b2c3d..8901j2k main -> origin/main
```

```
Updating a1b2c3d..8901j2k
```

```
Fast-forward
```

```
 README.md | 20 ++++++-----
```

```
config.json | 25 ++++++-----
```

```
main-file.txt | 15 ++++++-----
```

3 files changed, 58 insertions(+), 2 deletions(-)

```
create mode 100644 config.json
```

Step 22: Verify Local Files

Check that new file exists:

```
bash
```

```
ls -la
```

 **Expected Output:**

text

README.md

config.json

main-file.txt

Check file contents:

```
bash
```

```
cat config.json
```

 **Expected Output:** Should show the JSON configuration file you created

Check updated main file:

```
bash
```

```
tail -10 main-file.txt
```

 **Expected Output:** Should show the branch information you added

Step 23: Clean Up Local Branch (Optional)

Since we merged and deleted remote branch, clean up local branch:

bash

```
git branch -d feature-new-config-file
```

 **Expected Output:**

text

Deleted branch feature-new-config-file (was e4f5g6h).

Step 24: View Final Commit History

bash

```
git log --oneline --graph --all
```

 **Expected Output:**

text

```
* 8901j2k (HEAD -> main, origin/main) Merge pull request  
#1 from your-username/feature-new-config-file
```

| \

```
| * e4f5g6h feat: Add configuration file and update  
documentation
```

| /

* a1b2c3d Initial commit: Project setup with main file and documentation



Phase 6: Additional BitBucket Interface Features

Step 25: Explore BitBucket Interface

Review these features in BitBucket:

1. Commits View:

- Click "Commits" in sidebar
- See the merge commit and feature branch commit
- Click on commits to see detailed changes

2. File History:

- Click on any file (e.g., config.json)
- Click "History" to see file evolution
- See when file was created and modified

3. Compare Branches:

- Go to "Branches"
- Click "Compare" between main and other branches
- See differences visually

Step 26: Create Another Branch via BitBucket (Optional)

You can also create branches via BitBucket web interface:

- 1. Go to "Branches"**
- 2. Click "Create branch"**
- 3. Branch from: main**
- 4. Branch name: feature-web-created**
- 5. Click "Create"**

Then make changes via BitBucket editor:

- 1. Click on a file**
- 2. Click "Edit" (pencil icon)**
- 3. Make changes**
- 4. Commit directly to the new branch**
- 5. Create pull request**



Key BitBucket Interface Features Used

Feature	Purpose	Location
Pull Requests	Code review and merge	Left sidebar → Pull requests
Branch Management	View and create branches	Left sidebar → Branches

Feature	Purpose	Location
Diff View	See changes visually	Pull request → Diff tab
Merge Button	Complete the merge	Pull request page → Merge button
Commit History	Track changes over time	Left sidebar → Commits tab

Success Checklist

- BitBucket repository created
- Initial code pushed to main branch
- New feature branch created locally
- New file created in feature branch
- Existing files modified in feature branch
- Changes committed to feature branch
- Feature branch pushed to BitBucket
- Pull request created via BitBucket interface
- Changes reviewed in Diff view
- Pull request merged via web interface
- Feature branch deleted after merge (optional)
- Local repository updated with merge changes

- All changes verified locally
- Complete workflow demonstrated successfully

Complete Workflow Summary

text

Local Git Commands	BitBucket Web Interface
--------------------	-------------------------

----- -----

1. git checkout -b new-branch → 4. Create Pull Request
2. Create/edit files → 5. Review changes in Diff view
3. git commit & push → 6. Click Merge button
7. Verify merge completion

Q2) 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

Phase 1: Infrastructure Setup

1.1 Jenkins Server Installation

Prerequisites:

- Java JDK 11+ installed
- Server with minimum 2GB RAM, 20GB storage

Installation Steps:

bash

Ubuntu/Debian

```
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-stable binary/ >  
/etc/apt/sources.list.d/jenkins.list'
```

sudo apt update

sudo apt install jenkins

Start and enable

sudo systemctl start jenkins

sudo systemctl enable jenkins

sudo systemctl status jenkins

Initial Configuration:

1. Access <http://your-server:8080>
2. Unlock with initial admin password
3. Install suggested plugins

4. Create admin user
5. Configure instance URL

1.2 Local HTTP Server Setup

Nginx Installation:

```
bash
```

```
sudo apt update
```

```
sudo apt install nginx -y
```

```
sudo systemctl start nginx
```

```
sudo systemctl enable nginx
```

Directory Structure:

```
text
```

```
/var/www/
```

```
    └── html/
        ├── production/      # Live application
        ├── staging/         # Pre-production
        └── backups/         # Deployment backups
```

Nginx Configuration:

```
nginx
```

```
# /etc/nginx/sites-available/ci-cd-app
```

```
server {
```

```
    listen 80;
```

```
    server_name your-domain.com;
```

```
    root /var/www/html/production;
```

```
    index index.html index.htm;
```

```
location / {
```

```
    try_files $uri $uri/ =404;
```

```
}
```

```
# Access logs  
access_log /var/log/nginx/ci-cd-app-access.log;  
error_log /var/log/nginx/ci-cd-app-error.log;  
}
```

1.3 Git Repository Setup

Repository Structure:

text

web-app/

```
|   └── src/  
|       ├── index.html  
|       ├── css/  
|       ├── js/  
|       └── assets/  
|   └── tests/  
|   └── scripts/  
|       ├── deploy.sh  
|       └── test.sh  
└── Jenkinsfile  
└── package.json (if Node.js)  
└── README.md
```

Phase 2: Jenkins Configuration

2.1 Plugin Installation

Essential Plugins:

- Git Plugin
- GitHub Integration Plugin

- Pipeline Plugin
- SSH Plugin
- Email Extension Plugin
- Build Timeout Plugin
- Workspace Cleanup Plugin

Installation Method:

1. Jenkins Dashboard → Manage Jenkins → Manage Plugins
2. Available tab → Search and install required plugins
3. Restart Jenkins after installation

2.2 System Configuration

Global Tool Configuration:

- **Git Path:** /usr/bin/git (or auto-detected)
- **JDK:** Configure Java installation
- **Node.js** (if applicable): Set up Node.js installations

System Configuration:

- **# of executors:** 2-4 (based on server capacity)
- **Jenkins URL:** http://your-jenkins-server:8080

Credential Configuration:

1. Manage Jenkins → Manage Credentials
2. Add:
 - GitHub personal access token
 - SSH keys for server access
 - Email server credentials

2.3 Security Configuration

Configure Security Realm:

- Jenkins' own user database
- Create user accounts for team members

Authorization Strategy:

- Matrix-based security
 - Assign permissions per user/group
-

🔗 Phase 3: Webhook Setup

3.1 GitHub Webhook Configuration

In GitHub Repository:

1. Go to **Settings** → **Webhooks** → **Add webhook**
2. Configure:
 - **Payload URL:** `http://your-jenkins-server:8080/github-webhook/`
 - **Content type:** `application/json`
 - **Secret:** (Generate secure secret)
 - **Events:** **Push events**
 - **Active**

Webhook Secret Generation:

```
bash
```

```
openssl rand -base64 32
```

```
# Copy output for webhook secret
```

3.2 Jenkins GitHub Configuration

Configure GitHub Server:

1. **Manage Jenkins** → **Configure System**
2. **GitHub** section:
 - Add GitHub Server
 - API URL: `https://api.github.com`
 - Credentials: Add GitHub token
 - **Manage hooks**

Webhook Secret in Jenkins:

- Same secret as in GitHub webhook configuration
- Ensures secure communication

3.3 Webhook Testing

Test Webhook Delivery:

1. Make a push to repository
2. Check GitHub webhook delivery logs
3. Verify Jenkins build triggers automatically

Troubleshooting:

- Check Jenkins logs: /var/log/jenkins/jenkins.log
 - Verify network connectivity between GitHub and Jenkins
 - Ensure Jenkins URL is accessible from internet
-

Phase 4: Jenkins Pipeline Configuration

4.1 Pipeline Creation Methods

Method A: Jenkinsfile (Pipeline as Code)

groovy

```
// Jenkinsfile in repository root

pipeline {
    agent any

    environment {
        DEPLOY_DIR = '/var/www/html/production'
        STAGING_DIR = '/var/www/html/staging'
        NODE_VERSION = '16'
    }

    options {
        buildDiscarder(logRotator(numToKeepStr: '10'))
        timeout(time: 30, unit: 'MINUTES')
        timestamps()
    }
}
```

```

}

triggers {
    githubPush()
}

stages {
    // Stage definitions go here
}

post {
    // Post-build actions
}

```

Method B: Jenkins Web Interface

1. **New Item** → **Pipeline**
2. **Definition:** Pipeline script from SCM
3. **SCM:** Git
4. **Repository URL:** Your Git repository
5. **Script Path:** Jenkinsfile

4.2 Pipeline Stages Definition

Complete Stage Structure:

```

groovy
stages {
    stage('Checkout') {
        steps {
            checkout scm
            script {

```

```
        currentBuild.displayName = "Build #${env.BUILD_NUMBER}"
        currentBuild.description = "Commit: ${env.GIT_COMMIT}"
    }
}

}

stage('Dependency Installation') {
    steps {
        sh ""
        echo "Installing dependencies..."
        # npm install (if Node.js)
        # pip install -r requirements.txt (if Python)
        # composer install (if PHP)
        """
    }
}

stage('Code Quality') {
    steps {
        sh ""
        echo "Running code quality checks..."
        # ESLint, Prettier, etc.
        find . -name "*.html" -exec echo "HTML file: {}" \\
        find . -name "*.js" -exec echo "JS file: {}" \\
        """
    }
}
```

```
stage('Testing') {  
    steps {  
        sh ""  
        echo "Running tests..."  
        # Unit tests, integration tests  
        # npm test / python -m pytest / phpunit  
        ...  
        junit '**/test-reports/*.xml' // Publish test results  
    }  
}  
  
}
```

```
stage('Build') {  
    steps {  
        sh ""  
        echo "Building application..."  
        # npm run build / webpack / etc.  
        # Create production-ready files  
        ...  
    }  
}  
  
}
```

```
stage('Security Scan') {  
    steps {  
        sh ""  
        echo "Running security checks..."  
        # Basic security scanning  
        # Check for sensitive information  
        ...  
    }  
}
```

```
        }

    }

stage('Deploy to Staging'){

steps{
    sh ""

    echo "Deploying to staging..."

    chmod +x scripts/deploy.sh

    ./scripts/deploy.sh staging

    ""

}

}

stage('Integration Tests'){

steps{
    sh ""

    echo "Running integration tests..."

    # Test staging environment

    curl -f http://localhost/staging/ || exit 1

    ""

}

}

stage('Deploy to Production'){

when{
    branch 'main'
}

steps{
```

```
input message: 'Deploy to production?', ok: 'Deploy'  
sh ""  
echo "Deploying to production..."  
../scripts/deploy.sh production  
"  
}  
}  
}
```

4.3 Execute Shell Configuration

Deployment Script (scripts/deploy.sh):

```
bash  
#!/bin/bash  
set -e # Exit on any error  
  
ENVIRONMENT=$1  
TIMESTAMP=$(date +%Y%m%d_%H%M%S)  
BACKUP_DIR="/var/www/html/backups"  
LOG_FILE="/tmp/jenkins-deploy-${TIMESTAMP}.log"  
  
echo "🚀 Starting deployment to $ENVIRONMENT" | tee $LOG_FILE
```

```
deploy_to_environment() {  
    local target_dir=""  
    case $ENVIRONMENT in  
        "staging")  
            target_dir="/var/www/html/staging"  
            ;;  
        "production")
```

```
target_dir="/var/www/html/production"

;;
*)

echo "✖ Unknown environment: $ENVIRONMENT" | tee -a $LOG_FILE
exit 1

;;
esac
```

```
echo "📁 Target directory: $target_dir" | tee -a $LOG_FILE
```

Create backup

```
echo "💾 Creating backup..." | tee -a $LOG_FILE
sudo mkdir -p $BACKUP_DIR
if [ -d "$target_dir" ] && [ "$(ls -A $target_dir)" ]; then
    backup_file="${BACKUP_DIR}/backup_${ENVIRONMENT}_${TIMESTAMP}.tar.gz"
    sudo tar -czf "$backup_file" -C "$target_dir" . >> $LOG_FILE 2>&1
    echo "✅ Backup created: $backup_file" | tee -a $LOG_FILE
fi
```

Clean target directory

```
echo "⚡ Cleaning target directory..." | tee -a $LOG_FILE
sudo rm -rf "${target_dir}"//* || true
```

Copy build artifacts

```
echo "📄 Copying files..." | tee -a $LOG_FILE
sudo cp -r dist/* "$target_dir/" 2>/dev/null || sudo cp -r . "$target_dir/"
```

```

# Set permissions

echo "🔒 Setting permissions..." | tee -a $LOG_FILE

sudo chown -R www-data:www-data "$target_dir"

sudo chmod -R 755 "$target_dir"

sudo find "$target_dir" -type f -exec chmod 644 {} \;

# Verify deployment

echo "✅ Verifying deployment..." | tee -a $LOG_FILE

if [ -f "${target_dir}/index.html" ]; then

    echo "🎉 Deployment successful!" | tee -a $LOG_FILE

else

    echo "❌ Deployment verification failed" | tee -a $LOG_FILE

    exit 1

fi

}


```

Execute deployment

deploy_to_environment

```
echo "🏁 Deployment completed at $(date)" | tee -a $LOG_FILE
```

Jenkins Execute Shell Configuration:

```

bash

#!/bin/bash

echo =====

echo "🚀 JENKINS CI/CD PIPELINE EXECUTION"

echo =====

echo "Build: ${BUILD_NUMBER}"

```

```

echo "Job: ${JOB_NAME}"
echo "Workspace: ${WORKSPACE}"
echo "Git Commit: $(git log -1 --pretty=format:'%h - %s')"

# Make scripts executable
chmod +x scripts/*.sh

# Run deployment script
./scripts/deploy.sh $ENVIRONMENT

# Health check
echo " 🚀 Performing health check..."
curl -f http://localhost/$ENVIRONMENT/ || exit 1

echo " ✅ All steps completed successfully"

```

Phase 5: Build and Deployment Steps

5.1 Build Process

Frontend Build Example (if applicable):

bash

```

# package.json scripts
{
  "scripts": {
    "build": "webpack --mode production",
    "test": "jest",
    "lint": "eslint src/",
    "deploy": "npm run build && npm run test"
  }
}
```

```
}
```

Build Stage in Pipeline:

```
groovy
```

```
stage('Build') {
```

```
    steps {
```

```
        sh ""
```

```
        echo "🔨 Building application..."
```

```
# If Node.js project
```

```
if [ -f "package.json" ]; then
```

```
    npm ci
```

```
    npm run build
```

```
fi
```

```
# If build directory doesn't exist, create basic structure
```

```
if [ ! -d "dist" ]; then
```

```
    mkdir -p dist
```

```
    cp -r src/* dist/ 2>/dev/null || true
```

```
    cp index.html dist/ 2>/dev/null || true
```

```
fi
```

```
echo "✅ Build completed"
```

```
""
```

```
}
```

```
}
```

5.2 Testing Strategy

Unit Tests:

```
groovy
```

```

stage('Unit Tests') {
    steps {
        sh ""
        echo "🧪 Running unit tests..."

        # JavaScript/Node.js
        if [ -f "package.json" ] && grep -q "\"test\""' package.json; then
            npm test
        fi

        # Python
        if [ -f "requirements.txt" ]; then
            python -m pytest tests/ --junitxml=test-results.xml
        fi
        ...
        junit '**/test-*.xml' // Publish test results
    }
}

```

Integration Tests:

```

groovy
stage('Integration Tests') {
    steps {
        sh ""
        echo "🔗 Running integration tests..."

        # Test if application is accessible
        sleep 10 # Wait for deployment
        HTTP_STATUS=$(curl -s -o /dev/null -w "%{http_code}" http://localhost/staging/)
    }
}

```

```

if [ "$HTTP_STATUS" -eq 200 ]; then
    echo " ✅ Application is accessible"
else
    echo " ❌ Application not accessible. HTTP Status: $HTTP_STATUS"
    exit 1
fi

# Additional integration tests
# Test database connections, API endpoints, etc.

...
}

}

```

5.3 Deployment Process

Multi-environment Deployment:

```

groovy
stage('Deploy to Staging') {
    steps {
        sh """
            echo " 🌱 Deploying to staging environment..."
            export ENVIRONMENT=staging
            ./scripts/deploy.sh $ENVIRONMENT
            ...
        """
    }
}

```

```
stage('Deploy to Production') {
```

```
    when {
```

```

branch 'main'

}

steps {

    input message: 'Deploy to production?',  

    parameters: [string(name: 'VERSION', description: 'Version to deploy')],  

    ok: 'Deploy'  

    sh ""  

        echo "🚀 Deploying to production..."  

        export ENVIRONMENT=production  

        ./scripts/deploy.sh $ENVIRONMENT  

        ...
  

    }
}

```

Phase 6: Monitoring and Notifications

6.1 Pipeline Post-build Actions

```

groovy

post {

    always {

        echo "📊 Pipeline execution completed"

        script {

            // Archive important files

            archiveArtifacts artifacts: 'dist/**,scripts/**/**/test-results/*.xml', fingerprint: true

            // Clean workspace

            cleanWs()

        }
    }
}

```

```
success {

    echo "🎉 Pipeline executed successfully!"

    emailext (

        subject: "✅ SUCCESS: Job '${env.JOB_NAME}' [${env.BUILD_NUMBER}]",
        body: """
            Pipeline execution successful!

            Job: ${env.JOB_NAME}
            Build: ${env.BUILD_NUMBER}
            URL: ${env.BUILD_URL}

            Commit: ${env.GIT_COMMIT}
            Branch: ${env.GIT_BRANCH}
            """,
        to: "dev-team@company.com"
    )
}

failure {

    echo "❗ Pipeline failed!"

    emailext (

        subject: "❗ FAILURE: Job '${env.JOB_NAME}' [${env.BUILD_NUMBER}]",
        body: """
            Pipeline execution failed!

            Job: ${env.JOB_NAME}
            Build: ${env.BUILD_NUMBER}
            URL: ${env.BUILD_URL}
        """
    )
}
```

Please check the build logs for details.

```
""",  
    to: "dev-team@company.com"  
)  
}  
  
unstable {  
    echo "⚠ Pipeline unstable!"  
    // Handle unstable build notifications  
}  
}
```

6.2 Health Monitoring

Health Check Script:

```
bash  
#!/bin/bash  
# scripts/health-check.sh
```

```
ENVIRONMENT=$1
```

```
URL="http://localhost/$ENVIRONMENT/"
```

```
echo "💻 Performing health check on $URL"
```

```
# Check HTTP status  
  
HTTP_STATUS=$(curl -s -o /dev/null -w "%{http_code}" $URL)  
if [ "$HTTP_STATUS" -ne 200 ]; then  
    echo "✗ Health check failed: HTTP $HTTP_STATUS"  
    exit 1  
fi
```

```

# Check response time

RESPONSE_TIME=$(curl -s -o /dev/null -w "%{time_total}" $URL)

echo "  Health check passed: HTTP $HTTP_STATUS, Response time: ${RESPONSE_TIME}s"



---


# Additional checks

# - Database connectivity

# - API endpoints

# - Disk space

# - Memory usage

```

Phase 7: Security and Best Practices

7.1 Security Measures

Credential Management:

- Use Jenkins Credentials Store
- Never hardcode passwords in scripts
- Use SSH keys for server access

Pipeline Security:

```

groovy

properties([
    parameters([
        string(name: 'DEPLOY_ENV', defaultValue: 'staging', description: 'Deployment environment'),
        booleanParam(name: 'RUN_TESTS', defaultValue: true, description: 'Run tests')
    ]),
    pipelineTriggers([
        githubPush()
    ]),
])

```

```
buildDiscarder(logRotator(numToKeepStr: '10')),  
disableConcurrentBuilds()  
])
```

Access Control:

- Restrict Jenkins access to authorized users
- Use role-based access control (RBAC)
- Regular security updates

7.2 Best Practices

Pipeline Optimization:

- Use parallel stages for independent tasks
- Implement proper error handling
- Use build caching where possible
- Set reasonable timeouts

Code Quality:

- Implement code review process
- Use static code analysis
- Enforce coding standards
- Regular dependency updates

Phase 8: Troubleshooting and Maintenance

8.1 Common Issues and Solutions

Webhook Not Triggering:

- Check Jenkins URL accessibility
- Verify webhook secret matches
- Check GitHub/Jenkins logs

Permission Issues:

bash

```
# Fix Jenkins user permissions
```

```
sudo usermod -a -G www-data jenkins  
sudo chown -R jenkins:www-data /var/www/html  
sudo systemctl restart jenkins
```

Build Failures:

- Check workspace permissions
- Verify dependency availability
- Review build logs

8.2 Maintenance Tasks

Regular Maintenance:

- Update Jenkins and plugins
- Clean old builds and workspaces
- Monitor disk space
- Review and update deployment scripts

Backup Strategy:

- Regular Jenkins configuration backups
- Application deployment backups
- Database backups (if applicable)

Success Metrics and Monitoring

Key Performance Indicators (KPIs)

Pipeline Metrics:

- Build success rate
- Average build time
- Deployment frequency
- Mean time to recovery (MTTR)

Application Metrics:

- Uptime percentage
- Response times

- Error rates
- User satisfaction

Monitoring Setup

Jenkins Monitoring:

- Build trend analysis
- Resource utilization
- Plugin health checks

Application Monitoring:

- HTTP server logs
- Application performance monitoring
- User analytics

Implementation Checklist

- Jenkins server installed and configured
- Local HTTP server (Nginx/Apache) set up
- Git repository configured with webhook
- Jenkins plugins installed
- Pipeline created (Jenkinsfile or web interface)
- Deployment scripts developed and tested
- Security measures implemented
- Notifications configured
- Testing strategy defined
- Monitoring and logging set up
- Team trained on CI/CD process
- Documentation created
- Backup procedures established

This comprehensive outline provides a complete roadmap for setting up a robust CI/CD pipeline using Jenkins, Git, and a local HTTP server, ensuring automated, reliable, and efficient web application deployments.

Slip 9

Q.1) 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.

Bitbucket Operations Using Git - Complete Implementation Guide

I'll walk you through implementing all three Bitbucket tasks step by step with commands and expected outputs.

❖ Software Requirements

Required Software:

1. **Git** - Version control system
 2. **Bitbucket Account** - Create at bitbucket.org
 3. **Terminal/Command Prompt**
 4. **Text Editor**
-



Task 1: Create Repository on Bitbucket and Clone Locally

Step 1.1: Create Bitbucket Repository

In Bitbucket Web Interface:

1. **Log in to bitbucket.org**
2. **Click "Create" → "Repository"**
3. **Repository details:**
 - **Repository name:** bitbucket-operations-demo

- **Description:** "Demo repository for Bitbucket Git operations"
- **Visibility: Private**
- **Include README: No** (we'll create our own)
- **Git ignore: None**

4. Click "Create repository"

 **Expected:** Repository created with empty state

Step 1.2: Get Repository URL

After repository creation:

1. Copy the repository URL (HTTPS or SSH)

- **HTTPS:** `https://your-username@bitbucket.org/your-username/bitbucket-operations-demo.git`
- **SSH:** `git@bitbucket.org:your-username/bitbucket-operations-demo.git`

Step 1.3: Clone Repository Locally

Open Terminal/Command Prompt:

bash

Clone the repository

```
git clone https://your-username@bitbucket.org/your-username/bitbucket-operations-demo.git
```

 **Expected Output:**

text

Cloning into 'bitbucket-operations-demo'...

warning: You appear to have cloned an empty repository.

Navigate to the repository:

bash

```
cd bitbucket-operations-demo
```

Check current directory and git status:

bash

pwd

git status

Expected Output:

text

/path/to/bitbucket-operations-demo

On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)



Task 2: Create File, Commit, and Create Feature Branch

Step 2.1: Create example.txt File

Create the file with content:

bash

Create and open example.txt

cat > example.txt << EOF

Bitbucket Operations Demo

This file demonstrates Git operations with Bitbucket.

Project Information

- Repository: bitbucket-operations-demo
- Created: \$(date)
- Purpose: Demonstrate Bitbucket Git workflow

Operations to Demonstrate

1. Repository creation and cloning
2. File creation and committing
3. Branch creation and management
4. Pull request workflow
5. Code review and merging

File Details

- Name: example.txt
- Type: Markdown documentation
- Status: Initial version

Next Steps

- Create feature branch
- Push to Bitbucket
- Create pull request
- Review and merge changes

EOF

Verify file creation:

bash

```
ls -la
```

```
cat example.txt
```

 **Expected Output:**

```
text
```

```
total 8
```

```
drwxr-xr-x 3 user group 96 Nov 25 10:30 .
```

```
drwxr-xr-x 5 user group 160 Nov 25 10:30 ..
```

```
drwxr-xr-x 9 user group 288 Nov 25 10:30 .git
```

```
-rw-r--r-- 1 user group 650 Nov 25 10:30 example.txt
```

```
# Bitbucket Operations Demo
```

```
... (file content)
```

Step 2.2: Add and Commit the File

Check git status:

```
bash
```

```
git status
```

 **Expected Output:**

```
text
```

```
On branch main
```

```
No commits yet
```

Untracked files:

(use "git add <file>..." to include in what will be committed)

```
example.txt
```

nothing added to commit but untracked files present (use "git add" to track)

Stage the file:

bash

```
git add example.txt
```

Check status after staging:

bash

```
git status
```

 **Expected Output:**

text

```
On branch main
```

```
No commits yet
```

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

```
new file: example.txt
```

Commit the file:

bash

```
git commit -m "Initial commit: Add example.txt with project documentation"
```

 **Expected Output:**

text

```
[main (root-commit) a1b2c3d] Initial commit: Add example.txt with project documentation
```

```
1 file changed, 25 insertions(+)

create mode 100644 example.txt
```

Step 2.3: Create Feature Branch

Create and switch to feature branch:

bash

```
git checkout -b feature-enhance-example
```

Expected Output:

text

```
Switched to a new branch 'feature-enhance-example'
```

Verify current branch:

bash

```
git branch
```

Expected Output:

text

```
* feature-enhance-example
```

main

Step 2.4: Make Changes in Feature Branch

Update example.txt in feature branch:

bash

```
# Append additional content to example.txt
```

```
cat >> example.txt << 'EOF'
```

```
## Feature Branch Changes
```

This section was added in the 'feature-enhance-example' branch.

New Features Added:

- Enhanced documentation structure
- Additional examples and explanations
- Better formatting and organization

Git Commands Used So Far:

- `git clone` - Clone repository
- `git add` - Stage files
- `git commit` - Save changes
- `git checkout -b` - Create and switch branch

Branch Information:

- Current Branch: `feature-enhance-example`
- Purpose: Enhance `example.txt` documentation
- Status: Active development

Next Steps:

- Push feature branch to Bitbucket
- Create pull request
- Request code review
- Merge changes to main

EOF

Verify the changes:

bash

git diff

 **Expected Output:** Shows the added lines in green

Check status:

bash

git status

 **Expected Output:**

text

On branch feature-enhance-example

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: example.txt

no changes added to commit (use "git add" and/or "git commit -a")

Commit changes in feature branch:

bash

git add example.txt

git commit -m "feat: Enhance example.txt with feature branch documentation and examples"

 **Expected Output:**

text

[feature-enhance-example e4f5g6h] feat: Enhance example.txt with feature branch documentation and examples

1 file changed, 25 insertions(+)



Task 3: Push Feature Branch, Create and Merge Pull Request

Step 3.1: Push Feature Branch to Bitbucket

Push the feature branch:

bash

```
git push -u origin feature-enhance-example
```

Expected Output:

text

```
Enumerating objects: 6, done.
```

```
Counting objects: 100% (6/6), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (4/4), done.
```

```
Writing objects: 100% (6/6), 1.45 KiB | 1.45 MiB/s, done.
```

```
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
```

remote:

```
remote: Create pull request for feature-enhance-example:
```

```
remote: https://bitbucket.org/your-username/bitbucket-operations-demo/pull-requests/new?source=feature-enhance-example&t=1
```

remote:

```
To https://your-username@bitbucket.org/your-username/bitbucket-operations-demo.git
```

```
* [new branch]  feature-enhance-example -> feature-enhance-example
```

```
Branch 'feature-enhance-example' set up to track remote branch 'feature-enhance-example' from 'origin'.
```

Step 3.2: Verify on Bitbucket Web Interface

In Bitbucket:

1. **Refresh your repository page**
2. **Click "Branches"** - You should see both main and feature-enhance-example
3. **Click "Commits"** - You should see both commits
4. **Click on feature branch** - See the updated example.txt file

Step 3.3: Create Pull Request

Method 1: Using the link from push output

- Click the link shown in the push output:
<https://bitbucket.org/your-username/bitbucket-operations-demo/pull-requests/new?source=feature-enhance-example>

Method 2: Manual creation

1. **Go to Bitbucket repository**
2. **Click "Pull requests"** in left sidebar
3. **Click "Create pull request"**

Configure Pull Request:

4. **Title:** Enhance example.txt with feature branch documentation
5. **Description:**

text

Pull Request Description

This PR enhances the example.txt file with additional documentation created in the feature branch.

Changes Made:

- Added "Feature Branch Changes" section
- Included detailed Git commands explanation

- Enhanced documentation structure
- Added branch information and next steps

Files Changed:

- `example.txt` : Added 25 lines of documentation

Testing:

- File reviewed locally
- Changes properly formatted
- Ready for merge

Review Checklist:

- [] Documentation is clear and accurate
 - [] No sensitive information included
 - [] Code follows project standards
 - [] Ready for merge
6. **Reviewers:** (Optional) Leave empty for this demo
 7. **Source branch:** feature-enhance-example
 8. **Destination branch:** main
 9. **Click "Create pull request"**

Step 3.4: Review Pull Request

On the pull request page:

1. **Click "Diff" tab** to review changes
2. **You should see:**
 - o Green highlighted lines showing additions

- The entire new section you added
- No deletions (since we only added content)

3. Review the changes:

- Verify all intended changes are present
- Check formatting and content
- Ensure no unintended changes

Step 3.5: Merge Pull Request

On the pull request page:

1. Click "Merge" button (top right)
2. Merge options dialog appears:
 - **Merge strategy:** "Merge commit" (default)
 - **Delete branch:** **Delete branch feature-enhance-example after merge**
 - **Commit message:** (Auto-filled) "Merge pull request #1 from your-username/feature-enhance-example"
3. Click "Merge"

Expected Result:

- **Success message:** "Merged pull request #1"
- **Status changes to "Merged"** with green checkmark
- **Feature branch automatically deleted**
- **Redirect to merged pull request view**

Step 3.6: Verify Merge Completion

In Bitbucket:

1. **Check repository main page** - Should show updated example.txt
2. **Click "Commits"** - Should show merge commit and both original commits

3. **Click "Branches"** - Feature branch should be gone
4. **Click on example.txt** - Verify all changes are present

Step 3.7: Update Local Repository

Switch to main branch locally:

bash

```
git checkout main
```

 **Expected Output:**

text

```
Switched to branch 'main'
```

```
Your branch is up to date with 'origin/main'.
```

Pull the merged changes:

bash

```
git pull origin main
```

 **Expected Output:**

text

```
remote: Counting objects: 3, done.
```

```
remote: Compressing objects: 100% (3/3), done.
```

```
remote: Total 3 (delta 2), reused 0 (delta 0)
```

```
Unpacking objects: 100% (3/3), 1.25 KiB | 159.00 KiB/s, done.
```

```
From https://your-username@bitbucket.org/your-username/bitbucket-operations-demo
```

```
a1b2c3d..8901j2k main -> origin/main
```

```
Updating a1b2c3d..8901j2k
```

Fast-forward

```
example.txt | 25 ++++++=====++++++
```

1 file changed, 25 insertions(+)

Verify local files:

bash

```
cat example.txt
```

 **Expected Output:** Should show the complete file with all changes

Clean up local feature branch:

bash

```
git branch -d feature-enhance-example
```

 **Expected Output:**

text

Deleted branch feature-enhance-example (was e4f5g6h).

View final commit history:

bash

```
git log --oneline --graph --all
```

 **Expected Output:**

text

```
* 8901j2k (HEAD -> main, origin/main) Merge pull request #1 from your-  
username/feature-enhance-example
```

| \

```
| * e4f5g6h feat: Enhance example.txt with feature branch documentation  
and examples
```

| /

```
* a1b2c3d Initial commit: Add example.txt with project documentation
```



Complete Workflow Summary

All Commands Used:

bash

Task 1: Clone repository

```
git clone https://your-username@bitbucket.org/your-username/bitbucket-operations-demo.git
```

```
cd bitbucket-operations-demo
```

Task 2: Create file, commit, and create branch

Create example.txt

```
git add example.txt
```

```
git commit -m "Initial commit"
```

```
git checkout -b feature-enhance-example
```

Make changes to example.txt

```
git add example.txt
```

```
git commit -m "feat: Enhance documentation"
```

Task 3: Push, create PR, and merge

```
git push -u origin feature-enhance-example
```

Then use Bitbucket web interface to create and merge PR

Local cleanup

```
git checkout main
```

```
git pull origin main
```

```
git branch -d feature-enhance-example
```

```
git log --oneline --graph --all
```



Bitbucket Web Interface Operations

Operation	Location in Bitbucket	Purpose
Create Repository	"+" → "Repository"	Start new project
View Branches	Left sidebar → "Branches"	See all branches
Create PR	Left sidebar → "Pull requests" → "Create"	Start merge process
Review Changes	PR → "Diff" tab	Visual change review
Merge PR	PR page → "Merge" button	Complete the merge
View History	Left sidebar → "Commits"	Track changes over time

✓ Success Checklist

Task 1: ✓ Repository Creation and Cloning

- Bitbucket repository created successfully
- Repository cloned to local machine
- Local repository directory accessible
- Git status shows clean working directory

Task 2: ✓ File Operations and Branching

- example.txt file created with content
- File staged and committed to main branch
- Feature branch created successfully
- Changes made in feature branch
- Changes committed in feature branch

Task 3: ✓ P

Q2) 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....

Maven Dependency Management and Build Automation - Complete Guide

I'll explain how to set up and manage dependencies in Maven projects, the structure of pom.xml, and Maven's build automation process.

1. Maven Project Structure Overview

Standard Maven Directory Layout

text

my-maven-project/

```
|—— pom.xml           # Project Object Model (Configuration)  
|—— src/  
|   |—— main/  
|   |   |—— java/      # Main Java source code  
|   |   |   |—— com/example/App.java  
|   |   |—— resources/    # Resources (config files, properties)  
|   |   |   |—— application.properties  
|   |   |   |—— log4j2.xml  
|   |   |—— webapp/       # Web application files (if WAR)  
|   |—— test/  
|   |   |—— java/        # Test Java source code  
|   |   |   |—— com/example/AppTest.java  
|   |   |—— resources/    # Test resources
```

```
|--- target/          # Build output (generated)
|   |--- classes/
|   |--- test-classes/
|   |--- surefire-reports/
|   \--- my-app-1.0.0.jar
\--- .m2/           # Local repository (in user home)
    \--- repository/
```

2. Understanding pom.xml Structure

Basic pom.xml Components

xml

```
<?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">

    <!-- ===== -->
    <!-- 1. PROJECT IDENTIFICATION -->
    <!-- ===== -->

    <modelVersion>4.0.0</modelVersion>

    <!-- Project Coordinates (Unique Identifier) -->
    <groupId>com.company.example</groupId>
    <artifactId>my-application</artifactId>
```

```
<version>1.0.0</version>

<packaging>jar</packaging> <!-- jar, war, ear, pom -->

<!-- Project Metadata -->

<name>My Maven Application</name>
<description>A demonstration Maven project</description>
<url>http://www.example.com</url>

<!-- ===== -->
<!-- 2. PROJECT PROPERTIES -->
<!-- ===== -->

<properties>
    <!-- Java Version -->
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- Dependency Versions -->
    <junit.version>5.9.2</junit.version>
    <log4j.version>2.20.0</log4j.version>
    <spring.version>5.3.27</spring.version>

    <!-- Plugin Versions -->

    <maven.compiler.plugin.version>3.11.0</maven.compiler.plugin.version>
```

```
<maven.surefire.plugin.version>3.0.0</surefire.version>
</properties>

<!-- ===== -->
<!-- 3. DEPENDENCY MANAGEMENT -->
<!-- ===== -->

<dependencies>
    <!-- Production Dependencies -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
        <!-- Scope: compile, provided, runtime, test, system -->
        <scope>compile</scope>
        <!-- Optional: Exclude transitive dependencies -->
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

    <!-- Logging Framework -->
    <dependency>
```

```
<groupId>org.apache.logging.log4j</groupId>
<artifactId>log4j-core</artifactId>
<version>${log4j.version}</version>
</dependency>

<!-- Test Dependencies -->
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>
</dependencies>

<!-- ===== -->
<!-- 4. BUILD CONFIGURATION -->
<!-- ===== -->
<build>
```

```
<plugins>

    <!-- Compiler Plugin -->

    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>${maven.compiler.plugin.version}</version>
        <configuration>
            <source>11</source>
            <target>11</target>
            <encoding>UTF-8</encoding>
        </configuration>
    </plugin>

    <!-- Surefire Plugin (Testing) -->

    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>${maven.surefire.plugin.version}</version>
        <configuration>
            <includes>
                <include>**/*Test.java</include>
            </includes>
        </configuration>
    </plugin>
```

```
<!-- JAR Plugin -->

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>3.3.0</version>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
                <mainClass>com.example.App</mainClass>
            </manifest>
        </archive>
    </configuration>
</plugin>
</plugins>

<!-- Resource Handling -->

<resources>
    <resource>
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
    </resource>
</resources>
</build>
```

```
<!-- ===== -->
<!-- 5. REPOSITORY CONFIGURATION -->
<!-- ===== -->

<repositories>
    <repository>
        <id>central</id>
        <name>Maven Central Repository</name>
        <url>https://repo.maven.apache.org/maven2</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

```
<!-- ===== -->
<!-- 6. DISTRIBUTION MANAGEMENT -->
<!-- ===== -->

<distributionManagement>
    <repository>
        <id>company-releases</id>
        <name>Company Release Repository</name>
        <url>https://repo.company.com/releases</url>
```

```

    </repository>

    <snapshotRepository>
        <id>company-snapshots</id>
        <name>Company Snapshot Repository</name>
        <url>https://repo.company.com/snapshots</url>
    </snapshotRepository>

</distributionManagement>

</project>

```

3. Dependency Management in Maven

3.1 Adding Dependencies

Basic Dependency Structure:

xml

```

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.27</version>
</dependency>

```

Dependency Scopes:

Scope	Description	When Included
compile	Default scope	Always in classpath
provided	Provided by container	Compile & test, not packaged

Scope	Description	When Included
runtime	Not needed for compilation	Runtime & test only
test	Test dependencies	Test phase only
system	System path dependencies	Similar to provided

Example with Different Scopes:

xml

```
<dependencies>

    <!-- Compile scope (default) -->
    <dependency>
        <groupId>com.google.guava</groupId>
        <artifactId>guava</artifactId>
        <version>31.1-jre</version>
        <!-- scope not specified, defaults to compile -->
    </dependency>

    <!-- Provided scope (e.g., Servlet API) -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId> servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
```

```
<!-- Runtime scope (e.g., JDBC driver) -->

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
    <scope>runtime</scope>
</dependency>
```

```
<!-- Test scope -->

<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.9.2</version>
    <scope>test</scope>
</dependency>

</dependencies>
```

3.2 Dependency Exclusion

Excluding Transitive Dependencies:

xml

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.27</version>
    <exclusions>
        <exclusion>
```

```
<groupId>commons-logging</groupId>
<artifactId>commons-logging</artifactId>
</exclusion>
</exclusions>
</dependency>
```

3.3 Dependency Management with BOM

Using Bill of Materials (BOM):

xml

```
<dependencyManagement>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-dependencies</artifactId>
<version>2.7.8</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>
```

```
<dependencies>
<!-- No version needed - taken from BOM -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>  
</dependencies>
```

4. Maven Build Lifecycle and Automation

4.1 Maven Build Lifecycle Phases

Default Lifecycle Phases:

Phase	Description	Typical Plugin Goals
validate	Validate project	-
compile	Compile source code	compiler:compile
test	Run unit tests	surefire:test
package	Package compiled code	jar:jar, war:war
verify	Integration tests	-
install	Install to local repo	install:install
deploy	Deploy to remote repo	deploy:deploy

Clean Lifecycle:

- clean - Remove target directory

Site Lifecycle:

- site - Generate project documentation
- site-deploy - Deploy site to server

4.2 How Maven Handles Build Tasks

Compiling Code

Process:

1. **Reads source files** from src/main/java
2. **Uses maven-compiler-plugin** to compile
3. **Outputs .class files** to target/classes
4. **Compiles test code** from src/test/java to target/test-classes

Compiler Configuration:

xml

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.11.0</version>
  <configuration>
    <source>11</source>
    <target>11</target>
    <encoding>UTF-8</encoding>
    <showWarnings>true</showWarnings>
    <showDeprecation>true</showDeprecation>
    <!-- Compiler arguments -->
    <compilerArgs>
      <arg>-parameters</arg>
    </compilerArgs>
  </configuration>
</plugin>
```

Compilation Command:

bash

```
mvn compile
```

 **Expected Output:**

text

```
[INFO] Scanning for projects...
```

```
[INFO]
```

```
[INFO] --- maven-compiler-plugin:3.11.0:compile (default-compile) @ my-app ---
```

```
[INFO] Changes detected - recompiling the module!
```

```
[INFO] Compiling 3 source files to /project/target/classes
```

```
[INFO] -----
```

```
[INFO] BUILD SUCCESS
```

Running Tests

Process:

1. **Executes tests** using maven-surefire-plugin
2. *Runs all *Test.java* files in test source directory
3. **Generates reports** in target/surefire-reports
4. **Fails build** if tests fail

Test Configuration:

xml

```
<plugin>
```

```
  <groupId>org.apache.maven.plugins</groupId>
```

```
  <artifactId>maven-surefire-plugin</artifactId>
```

```
  <version>3.0.0</version>
```

```
  <configuration>
```

```
    <includes>
```

```
<include>**/*Test.java</include>
<include>**/*Tests.java</include>
</includes>
<excludes>
<exclude>**/Abstract*Test.java</exclude>
</excludes>
<parallel>methods</parallel>
<threadCount>4</threadCount>
</configuration>
</plugin>
```

Test Command:

bash

mvn test

 **Expected Output:**

text

[INFO] Scanning for projects...

[INFO]

[INFO] --- maven-surefire-plugin:3.0.0:test (default-test) @ my-app ---

[INFO] Using auto detected provider

org.apache.maven.surefire.junitplatform.JUnitPlatformProvider

[INFO]

[INFO] -----

[INFO] TESTS

[INFO] -----

[INFO] Running com.example.AppTest

[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.345 s

[INFO]

[INFO] Results:

[INFO]

[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO] -----

[INFO] BUILD SUCCESS

Generating JAR Files

Process:

1. **Packages compiled classes** and resources
2. **Creates manifest file** with metadata
3. **Generates JAR** in target/ directory
4. **Can create executable JAR** with dependencies

JAR Configuration:

xml

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>3.3.0</version>
  <configuration>
    <archive>
      <manifest>
        <addClasspath>true</addClasspath>
        <classpathPrefix>lib/</classpathPrefix>
```

```
<mainClass>com.example.MainApplication</mainClass>
</manifest>
<manifestEntries>
    <Implementation-Version>${project.version}</Implementation-
Version>
    <Built-By>${user.name}</Built-By>
</manifestEntries>
</archive>
</configuration>
</plugin>
```

Fat JAR with Dependencies (Using Assembly Plugin):

xml

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.4.2</version>
    <configuration>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <archive>
            <manifest>
                <mainClass>com.example.MainApplication</mainClass>
            </manifest>
        </archive>
    </configuration>
```

```
</configuration>

<executions>
    <execution>
        <id>make-assembly</id>
        <phase>package</phase>
        <goals>
            <goal>single</goal>
        </goals>
    </execution>
</executions>
</plugin>
```

Package Command:

bash

mvn package

Expected Output:

text

[INFO] Scanning for projects...

[INFO]

[INFO] --- maven-jar-plugin:3.3.0:jar (default-jar) @ my-app ---

[INFO] Building jar: /project/target/my-app-1.0.0.jar

[INFO]

[INFO] --- maven-assembly-plugin:3.4.2:single (make-assembly) @ my-app

[INFO] Building jar: /project/target/my-app-1.0.0-jar-with-dependencies.jar

[INFO] -----

[INFO] BUILD SUCCESS

[INFO] -----

5. Advanced Dependency Management

5.1 Dependency Mediation and Conflict Resolution

Maven's Dependency Resolution Rules:

1. **Nearest Definition Wins** - Closer dependency in tree wins
2. **First Declaration Wins** - First dependency declared wins

Viewing Dependency Tree:

bash

```
mvn dependency:tree
```

Expected Output:

text

```
[INFO] com.example:my-app:jar:1.0.0
```

```
[INFO] +- org.springframework:spring-core:jar:5.3.27:compile
```

```
[INFO] | \- commons-logging:commons-logging:jar:1.2:compile
```

```
[INFO] +- junit:junit:jar:4.13.2:test
```

```
[INFO] | \- org.hamcrest:hamcrest-core:jar:1.3:test
```

```
[INFO] \- com.google.guava:guava:jar:31.1-jre:compile
```

Analyzing Dependency Conflicts:

bash

```
mvn dependency:tree -Dverbose
```

5.2 Managing Transitive Dependencies

Using dependencyManagement:

xml

```
<dependencyManagement>  
    <dependencies>  
        <dependency>  
            <groupId>com.fasterxml.jackson</groupId>  
            <artifactId>jackson-bom</artifactId>  
            <version>2.14.2</version>  
            <type>pom</type>  
            <scope>import</scope>  
        </dependency>  
    </dependencies>  
</dependencyManagement>
```

5.3 Profiles for Environment-Specific Dependencies

Environment-Specific Configuration:

xml

```
<profiles>  
    <profile>  
        <id>development</id>  
        <activation>  
            <activeByDefault>true</activeByDefault>  
        </activation>  
        <dependencies>  
            <dependency>  
                <groupId>com.h2database</groupId>  
                <artifactId>h2</artifactId>  
                <version>2.1.214</version>  
            </dependency>
```

```
<scope>runtime</scope>
</dependency>
</dependencies>
</profile>

<profile>
<id>production</id>
<dependencies>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.33</version>
<scope>runtime</scope>
</dependency>
</dependencies>
</profile>
</profiles>
```

Activate Profile:

bash

mvn clean package -P production



6. Complete Build Automation Example

6.1 Comprehensive pom.xml for Build Automation

xml

```
<?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.company</groupId>
    <artifactId>automated-build-demo</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <properties>
        <java.version>11</java.version>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <!-- Dependency Versions -->
        <junit.version>5.9.2</junit.version>
        <log4j.version>2.20.0</log4j.version>
        <mockito.version>5.3.1</mockito.version>

        <!-- Plugin Versions -->
        <compiler.plugin.version>3.11.0</compiler.plugin.version>
        <surefire.plugin.version>3.0.0</surefire.plugin.version>
```

```
<jar.plugin.version>3.3.0</jar.plugin.version>
<resources.plugin.version>3.3.1</resources.plugin.version>
</properties>

<dependencies>
    <!-- Production Dependencies -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>${log4j.version}</version>
    </dependency>

    <dependency>
        <groupId>com.google.guava</groupId>
        <artifactId>guava</artifactId>
        <version>31.1-jre</version>
    </dependency>

    <!-- Test Dependencies -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
```

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>${mockito.version}</version>
    <scope>test</scope>
</dependency>
</dependencies>

<build>
    <finalName>${project.artifactId}-${project.version}</finalName>

<plugins>
    <!-- Compiler Plugin -->
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>${compiler.plugin.version}</version>
```

```
<configuration>
    <source>${java.version}</source>
    <target>${java.version}</target>
    <encoding>${project.build.sourceEncoding}</encoding>
</configuration>
</plugin>
```

```
<!-- Resources Plugin -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>${resources.plugin.version}</version>
    <configuration>
        <encoding>${project.build.sourceEncoding}</encoding>
    </configuration>
</plugin>
```

```
<!-- Surefire Plugin (Testing) -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>${surefire.plugin.version}</version>
    <configuration>
        <includes>
            <include>**/*Test.java</include>
```

```
<include>**/*Tests.java</include>  
</includes>  
  
<excludes>  
    <exclude>**/Abstract*Test.java</exclude>  
</excludes>  
  
</configuration>  
  
</plugin>
```

```
<!-- JAR Plugin -->  
  
<plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-jar-plugin</artifactId>  
    <version>${jar.plugin.version}</version>  
  
    <configuration>  
        <archive>  
            <manifest>  
                <addClasspath>true</addClasspath>  
                <mainClass>com.company.MainApp</mainClass>  
            </manifest>  
        </archive>  
    </configuration>  
</plugin>
```

```
<!-- Exec Plugin (Run application) -->  
  
<plugin>
```

```
<groupId>org.codehaus.mojo</groupId>
<artifactId>exec-maven-plugin</artifactId>
<version>3.1.0</version>
<configuration>
  <mainClass>com.company.MainApp</mainClass>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

6.2 Complete Build Commands

Complete Build Process:

bash

Clean and build everything

mvn clean compile test package

Skip tests

mvn clean package -DskipTests

Run specific phase

mvn compile

mvn test

mvn package

Install to local repository

```
mvn clean install
```

```
# Run application
```

```
mvn exec:java
```

```
# Generate site documentation
```

```
mvn site
```

```
# Dependency analysis
```

```
mvn dependency:tree
```

```
mvn dependency:analyze
```



7. Best Practices for Dependency Management

7.1 Version Management

Use Properties for Versions:

```
xml
```

```
<properties>  
    <spring.version>5.3.27</spring.version>  
    <junit.version>5.9.2</junit.version>  
</properties>
```

```
<dependencies>  
    <dependency>  
        <groupId>org.springframework</groupId>  
        <artifactId>spring-core</artifactId>
```

```
<version>${spring.version}</version>  
</dependency>  
</dependencies>
```

7.2 Scope Appropriately

Proper Scope Usage:

- **compile** - Core application dependencies
- **provided** - Container-provided dependencies
- **runtime** - Dependencies needed only at runtime
- **test** - Testing frameworks and utilities

7.3 Regular Updates

Check for Updates:

bash

```
mvn versions:display-dependency-updates
```

```
mvn versions:display-plugin-updates
```

Update Dependencies:

bash

```
mvn versions:use-latest-versions
```

Summary

Key Maven Concepts:

1. **pom.xml Structure** - Project configuration and metadata
2. **Dependency Management** - Declaring and managing project dependencies
3. **Build Lifecycle** - Predefined phases for build automation
4. **Plugin System** - Extensible build functionality

5. Repository Management - Local and remote artifact storage

Maven Build Automation Flow:

text

mvn command → Lifecycle Phase → Plugin Goals → Build Output

↓ ↓ ↓ ↓

mvn compile → compile → compiler:compile → target/classes

mvn test → test → surefire:test → test results

mvn package → package → jar:jar → JAR file

Slip 10)

Q.1) 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>).

'll guide you through creating a Dockerized "Hello, World!" HTML application step by step. We'll use **VS Code** as our primary software.

Software Requirements:

- **Docker Desktop** (for Windows/Mac) or **Docker Engine** (for Linux)
- **VS Code** with Docker extension (optional but helpful)
- **Terminal/Command Prompt**

Step 1: Install Docker

Windows/Mac:

1. Download Docker Desktop from <https://www.docker.com/products/docker-desktop/>
2. Install and launch Docker Desktop
3. Wait for Docker to start (you'll see the whale icon in system tray)

Linux (Ubuntu example):

bash

```
# Update package index
```

```
sudo apt update
```

```
# Install Docker
```

```
sudo apt install docker.io
```

```
# Start Docker service
```

```
sudo systemctl start docker  
sudo systemctl enable docker
```

Add user to docker group (optional, to avoid using sudo)

```
sudo usermod -aG docker $USER
```

Log out and log back in for group changes to take effect

Verify Docker installation:

```
bash
```

```
docker --version
```

Expected Output:

text

```
Docker version 24.0.6, build ed223bc
```

Step 2: Create Project Directory and Files

Create a new directory for your project:

```
bash
```

```
mkdir docker-hello-world
```

```
cd docker-hello-world
```

Create index.html:

```
html
```

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
    <meta charset="UTF-8">
```

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

```
    <title>Hello, World! Docker App</title>
```

```
<style>
```

```
body {
```

```
    font-family: Arial, sans-serif;
```

```
max-width: 800px;  
margin: 0 auto;  
padding: 20px;  
background-color: #f0f8ff;  
}  
  
.container {  
text-align: center;  
padding: 50px;  
background-color: white;  
border-radius: 10px;  
box-shadow: 0 0 10px rgba(0,0,0,0.1);  
margin-top: 50px;  
}  
  
h1 {  
color: #2c3e50;  
}  
  
.docker-logo {  
font-size: 48px;  
margin-bottom: 20px;  
}  
  
</style>  
</head>  
  
<body>  
  
<div class="container">  
  
<div class="docker-logo">  </div>  
  
<h1>Hello, World! from Docker Container!</h1>  
  
<p>This HTML page is being served from a Docker container using Nginx.</p>  
  
<p>Current time: <span id="datetime"></span></p>
```

```
<p>Container Hostname: <strong id="hostname">Loading...</strong></p>
</div>

<script>
    // Display current date and time
    function updateDateTime() {
        const now = new Date();
        document.getElementById('datetime').textContent = now.toLocaleString();
    }

    // Display container hostname (will be shown in browser)
    document.getElementById('hostname').textContent = window.location.hostname;

    updateDateTime();
    setInterval(updateDateTime, 1000);
</script>

</body>
</html>

Create Dockerfile:

dockerfile
# Use official Nginx image as base
FROM nginx:alpine

# Copy our HTML file to the Nginx web server directory
COPY index.html /usr/share/nginx/html/index.html

# Expose port 80 (default HTTP port for Nginx)
EXPOSE 80
```

```
# Nginx starts automatically when the container runs, so no need for CMD
```

Step 3: Project Structure Verification

Check your project structure:

```
bash
```

```
ls -la
```

Expected Output:

```
text
```

```
total 16
```

```
drwxr-xr-x 2 user user 4096 Dec 1 10:30 .
```

```
drwxr-xr-x 10 user user 4096 Dec 1 10:30 ..
```

```
-rw-r--r-- 1 user user 1452 Dec 1 10:30 Dockerfile
```

```
-rw-r--r-- 1 user user 1567 Dec 1 10:30 index.html
```

Step 4: Build Docker Image

Build the Docker image with a tag:

```
bash
```

```
docker build -t hello-world-app:latest .
```

Expected Output:

```
text
```

```
[+] Building 15.2s (6/6) FINISHED
```

```
=> [internal] load build definition from Dockerfile
```

```
=> => transferring dockerfile: 131B
```

```
=> [internal] load .dockerignore
```

```
=> => transferring context: 2B
```

```
=> [internal] load metadata for docker.io/library/nginx:alpine
```

```
=> [1/2] FROM docker.io/library/nginx:alpine@sha256:...
```

```
=> [internal] load build context
```

```
=> => transferring context: 1.61kB
```

```
=> [2/2] COPY index.html /usr/share/nginx/html/index.html  
=> exporting to image  
=> => exporting layers  
=> => writing image sha256:abc123def456...  
=> => naming to docker.io/library/hello-world-app:latest
```

Verify the image was created:

bash

docker images

Expected Output:

text

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world-app	latest	abc123def456	2 minutes ago	23.5MB
nginx	alpine	xyz789abc123	2 weeks ago	23.5MB

Step 5: Run the Container

Run the container, mapping port 8080 on your host to port 80 in the container:

bash

docker run -d -p 8080:80 --name hello-world-container hello-world-app:latest

Expected Output:

text

abc123def4567890... (container ID)

Verify the container is running:

bash

docker ps

Expected Output:

text

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
abc123def456	hello-world-app:latest	"/docker-entrypoint...."	30 seconds ago	Up 29 seconds	0.0.0.0:8080->80/tcp
					hello-world-container

Step 6: Test the Application

Open your web browser and navigate to:

text

`http://localhost:8080`

Expected Result:

You should see a styled "Hello, World!" page with:

- A whale emoji (🐳)
- "Hello, World! from Docker Container!" message
- Current date and time
- Container hostname displayed

Step 7: Additional Useful Commands

Check container logs:

bash

`docker logs hello-world-container`

Expected Output:

text

`/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration`

`/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/`

`/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh`

`...`

Stop the container:

bash

`docker stop hello-world-container`

Remove the container:

bash

`docker rm hello-world-container`

Remove the image:

```
bash
```

```
docker rmi hello-world-app:latest
```

Step 8: Using Docker Compose (Optional)

Create docker-compose.yml for easier management:

```
yaml
```

```
version: '3.8'
```

```
services:
```

```
  web:
```

```
    build: .
```

```
    ports:
```

```
      - "8080:80"
```

```
    container_name: hello-world-compose
```

Run with Docker Compose:

```
bash
```

```
docker-compose up -d
```

Stop with Docker Compose:

```
bash
```

```
docker-compose down
```

Summary

You've successfully:

1. Installed Docker
2. Created a "Hello, World!" HTML application
3. Created a Dockerfile using Nginx base image
4. Built a Docker image with proper tagging
5. Ran a container with port mapping
6. Accessed the application on <http://localhost:8080>

The application is now containerized and can be easily shared or deployed anywhere Docker is available

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.

I'll guide you through Git and GitHub repository management step by step. We'll use **Git Bash** (Windows) or **Terminal** (Mac/Linux) and **VS Code**.

Software Requirements:

- **Git** installed on your system
- **GitHub account**
- **VS Code** (optional, for editing files)
- **Terminal/Git Bash**

Task 1: Repository Setup and Initial Commit

Step 1: Install and Configure Git

Verify Git installation:

bash

git --version

Expected Output:

text

git version 2.39.2

Configure Git (if not done already):

bash

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

Verify configuration:

```
bash  
git config --list
```

Expected Output:

```
text  
  
user.name=Your Name  
  
user.email=your.email@example.com  
  
...
```

Step 2: Create Local Repository and Project File

Create project directory:

```
bash  
mkdir my-project  
cd my-project
```

Create project.md file:

```
bash  
# Using VS Code  
code project.md  
  
Or create it manually with this content:  
  
markdown  
  
# Project Title: TaskMaster Pro
```

```
## Project Description
```

TaskMaster Pro is a hypothetical project management application designed to help teams organize, track, and collaborate on tasks efficiently. It features real-time updates, file sharing, and progress tracking.

```
## Key Features
```

- Task creation and assignment
- Progress tracking with visual indicators

- Team collaboration tools
- File attachments and comments
- Deadline reminders and notifications

Technology Stack

- Frontend: React.js with TypeScript
- Backend: Node.js with Express
- Database: MongoDB
- Deployment: Docker and AWS

Project Status

Currently in planning phase. Expected completion: Q2 2024.

Step 3: Initialize Git Repository and Make Initial Commit

Initialize Git repository:

bash

git init

Expected Output:

text

Initialized empty Git repository in /path/to/your/my-project/.git/

Check repository status:

bash

git status

Expected Output:

text

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

project.md

nothing added to commit but untracked files present (use "git add" to track)

Add file to staging area:

bash

git add project.md

Commit the changes:

bash

git commit -m "Initial commit: Add project description and features"

Expected Output:

text

[main (root-commit) abc1234] Initial commit: Add project description and features

1 file changed, 20 insertions(+)

create mode 100644 project.md

Step 4: Create GitHub Repository and Push

On GitHub:

1. Go to <https://github.com>
2. Click "+" icon in top right → "New repository"
3. Repository name: my-project
4. Description: "TaskMaster Pro - Project management application"
5. Choose Public/Private
6. **DO NOT** initialize with README (since we have existing files)
7. Click "Create repository"

Back in terminal, add remote origin and push:

bash

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

```
git branch -M main
```

```
git push -u origin main
```

Expected Output:

text

```
Enumerating objects: 3, done.
```

```
Counting objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 456 bytes | 456.00 KiB/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To https://github.com/your-username/my-project.git
```

```
* [new branch]  main -> main
```

```
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Verify the push was successful:

bash

```
git log --oneline
```

Expected Output:

text

```
abc1234 (HEAD -> main, origin/main) Initial commit: Add project description and features
```

Task 2: Branching and Merging

Step 1: Create and Switch to Feature Branch

Create and switch to new branch:

bash

```
git checkout -b feature-branch
```

Expected Output:

text

```
Switched to a new branch 'feature-branch'
```

Verify current branch:

bash

```
git branch
```

Expected Output:

text

main

* feature-branch

Step 2: Make Changes in Feature Branch

Edit project.md to add new content. Add the following section at the end:

markdown

```
## New Features Added in Feature Branch
```

```
### Advanced Features
```

- **Time Tracking**: Integrated time tracking for tasks
- **Gantt Charts**: Visual project timelines and dependencies
- **API Integration**: RESTful API for third-party integrations
- **Mobile App**: Cross-platform mobile application (iOS & Android)

```
### Technical Improvements
```

- Implemented Redis for caching
- Added comprehensive unit tests (90% coverage)
- Enhanced security with JWT authentication
- Improved database indexing for better performance

Check status of changes:

bash

git status

Expected Output:

text

On branch feature-branch

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: project.md

no changes added to commit (use "git add" and/or "git commit -a")

Step 3: Commit Changes in Feature Branch

Add and commit changes:

bash

git add project.md

git commit -m "feat: Add advanced features and technical improvements"

Expected Output:

text

[feature-branch def5678] feat: Add advanced features and technical improvements

1 file changed, 16 insertions(+)

Step 4: Switch Back to Main Branch

Switch to main branch:

bash

git checkout main

Expected Output:

text

Switched to branch 'main'

Your branch is up to date with 'origin/main'.

Verify we're on main branch and check file content:

bash

git branch

cat project.md

Expected Output:

text

```
* main  
  feature-branch
```

(The project.md file should show the original content without the new features section)

Step 5: Merge Feature Branch into Main

Merge the feature branch:

```
bash  
git merge feature-branch
```

Expected Output:

```
text  
  
Updating abc1234..def5678  
  
Fast-forward  
  
project.md | 16 ++++++  
1 file changed, 16 insertions(+)
```

Check the merged content:

```
bash  
cat project.md
```

Expected Output: (File should now include both original content and new features)

Verify merge in log:

```
bash  
git log --oneline --graph
```

Expected Output:

```
text  
  
* def5678 (HEAD -> main, feature-branch) feat: Add advanced features and technical  
improvements  
  
* abc1234 (origin/main) Initial commit: Add project description and features
```

Step 6: Push Updated Main Branch to GitHub

Push the merged changes:

```
bash
```

```
git push origin main
```

Expected Output:

text

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Writing objects: 100% (3/3), 567 bytes | 567.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

To <https://github.com/your-username/my-project.git>

abc1234..def5678 main -> main

Step 7: Verify on GitHub

1. Go to your GitHub repository
2. Refresh the page
3. You should see:
 - o Updated project.md file with new content
 - o Commit history showing both commits
 - o The merge reflected in the main branch

Additional Verification Commands

Check remote repository status:

bash

git remote -v

Expected Output:

text

origin <https://github.com/your-username/my-project.git> (fetch)

origin <https://github.com/your-username/my-project.git> (push)

View detailed log:

bash

git log --oneline --graph --all

Expected Output:

text

* def5678 (HEAD -> main, origin/main, feature-branch) feat: Add advanced features and technical improvements

* abc1234 Initial commit: Add project description and features

Cleanup (Optional)

Delete the feature branch (if no longer needed):

bash

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

Expected Output:

text

Deleted branch feature-branch (was def5678).

Summary

Task 1 Completed:

- Local Git repository initialized
- project.md created with project description
- Initial commit made
- Repository pushed to GitHub

Task 2 Completed:

- feature-branch created and switched to
- Additional changes made to project.md
- Changes committed in feature branch
- Switched back to main branch
- Feature branch merged into main
- Updated main branch pushed to GitHub
- Merge reflected on GitHub repository

Your Git workflow is now complete! The repository shows proper branching, merging, and collaboration practices.

Slip 11)

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

I'll guide you through containerizing a basic application and deploying it using Docker. We'll create a **Node.js web application** as our example.

Software Requirements:

- **Docker Desktop** (Windows/Mac) or **Docker Engine** (Linux)
- **VS Code** or any text editor
- **Terminal/Command Prompt**
- **Node.js** (optional, for local testing)

Step 1: Create the Basic Application

Project Structure:

text

```
my-docker-app/
    ├── app.js
    ├── package.json
    ├── Dockerfile
    ├── .dockerignore
    └── README.md
```

Create project directory:

bash

```
mkdir my-docker-app
cd my-docker-app
```

Create package.json:

json

```
{
  "name": "my-docker-app",
```

```
"version": "1.0.0",
"description": "A simple Node.js application containerized with Docker",
"main": "app.js",
"scripts": {
  "start": "node app.js",
  "dev": "node app.js"
},
"dependencies": {
  "express": "^4.18.2"
},
"keywords": ["docker", "node", "express"],
"author": "Your Name",
"license": "MIT"
}
```

Create app.js (Node.js Express Application):

javascript

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;
```

// Middleware

```
app.use(express.json());
app.use(express.static('public'));
```

// Routes

```
app.get('/', (req, res) => {
  res.json({
    message: 'Welcome to Dockerized Node.js App! 🚀'
  })
})
```

```
    timestamp: new Date().toISOString(),
    environment: process.env.NODE_ENV || 'development',
    containerId: process.env.HOSTNAME || 'local'
  });
}

app.get('/health', (req, res) => {
  res.json({
    status: 'OK',
    uptime: process.uptime(),
    timestamp: new Date().toISOString()
  });
}

app.get('/info', (req, res) => {
  res.json({
    app: 'Dockerized Node.js Application',
    version: '1.0.0',
    description: 'A simple Express app running in a Docker container',
    features: [
      'RESTful API endpoints',
      'Health check endpoint',
      'Environment-based configuration',
      'Docker optimized'
    ]
  });
})
```

```
// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Something went wrong!' });
});

// 404 handler
app.use((req, res) => {
  res.status(404).json({ error: 'Endpoint not found' });
});

// Start server
app.listen(PORT, '0.0.0.0', () => {
  console.log(`🚀 Server running on port ${PORT}`);
  console.log(`🌐 Environment: ${process.env.NODE_ENV || 'development'}`);
  console.log(`🔗 Health check: http://localhost:${PORT}/health`);
});
```

Create Dockerfile:

dockerfile

```
# Use official Node.js runtime as base image
```

```
FROM node:18-alpine
```

```
# Set working directory in container
```

```
WORKDIR /app
```

```
# Copy package files
```

```
COPY package*.json ./
```

```
# Install dependencies
RUN npm install --only=production

# Copy application source code
COPY ..

# Create a non-root user to run the app (security best practice)
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nextjs -u 1001

# Change ownership of the /app directory to the non-root user
RUN chown -R nextjs:nodejs /app
USER nextjs

# Expose port
EXPOSE 3000

# Define environment variable
ENV NODE_ENV=production

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
CMD node -e "require('http').get('http://localhost:3000/health', (res) => { \
  if (res.statusCode === 200) process.exit(0); \
  else process.exit(1); \
}).on('error', () => process.exit(1))"

# Command to run the application
```

```
CMD ["npm", "start"]
```

Create .dockerignore:

text

node_modules

npm-debug.log

.git

.gitignore

README.md

.env

.nyc_output

coverage

.docker

Dockerfile

.dockerignore

Create README.md:

markdown

```
# Dockerized Node.js Application
```

A simple Express.js application containerized with Docker.

Features

- RESTful API endpoints
- Health check endpoint
- Docker optimized
- Non-root user for security
- Health checks

Build and Run

```
``` bash
docker build -t my-node-app .
docker run -p 3000:3000 my-node-app
```

## Endpoints

- GET / - Welcome message
- GET /health - Health check
- GET /info - Application information

text

## ## Step 2: Verify Project Structure

Check your project structure:

```
``` bash
```

```
ls -la
```

Expected Output:

text

```
total 28
```

```
drwxr-xr-x 2 user user 4096 Dec 1 11:00 .
drwxr-xr-x 10 user user 4096 Dec 1 11:00 ..
-rw-r--r-- 1 user user 1234 Dec 1 11:00 app.js
-rw-r--r-- 1 user user 1234 Dec 1 11:00 Dockerfile
-rw-r--r-- 1 user user 156 Dec 1 11:00 .dockerignore
-rw-r--r-- 1 user user 1234 Dec 1 11:00 package.json
-rw-r--r-- 1 user user 1234 Dec 1 11:00 README.md
```

Step 3: Build Docker Image

Build the Docker image with tags:

```
bash
```

```
docker build -t my-node-app:latest -t my-node-app:v1.0.0 .
```

Expected Output:

text

```
[+] Building 45.2s (12/12) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 662B
=> [internal] load .dockerignore
=> => transferring context: 112B
=> [internal] load metadata for docker.io/library/node:18-alpine
=> [1/7] FROM docker.io/library/node:18-alpine@sha256:...
=> [internal] load build context
=> => transferring context: 3.21kB
=> [2/7] WORKDIR /app
=> [3/7] COPY package*.json .
=> [4/7] RUN npm install --only=production
=> [5/7] COPY ..
=> [6/7] RUN addgroup -g 1001 -S nodejs
=> [7/7] RUN adduser -S nextjs -u 1001
=> [8/7] RUN chown -R nextjs:nodejs /app
=> [9/7] USER nextjs
=> exporting to image
=> => exporting layers
=> => writing image sha256:abc123def456...
=> => naming to docker.io/library/my-node-app:latest
=> => naming to docker.io/library/my-node-app:v1.0.0
```

Verify the image was created:

bash

docker images

Expected Output:

text

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-node-app	latest	abc123def456	2 minutes ago	125MB
my-node-app	v1.0.0	abc123def456	2 minutes ago	125MB
node	18-alpine	xyz789abc123	2 weeks ago	178MB

Step 4: Run the Container Locally

Basic Run:

bash

```
docker run -d -p 3000:3000 --name node-app my-node-app:latest
```

Expected Output:

text

abc123def4567890... (container ID)

Run with Environment Variables:

bash

```
docker run -d \
-p 3000:3000 \
--name node-app-prod \
-e NODE_ENV=production \
-e PORT=3000 \
my-node-app:latest
```

Verify container is running:

bash

```
docker ps
```

Expected Output:

text

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
abc123def456	my-node-app:latest	"docker-entrypoint.s..."	30 seconds ago	Up 29 seconds (healthy)
		0.0.0.0:3000->3000/tcp		node-app

Step 5: Test the Application

Test endpoints using curl:

bash

```
curl http://localhost:3000/
```

Expected Output:

json

```
{
```

```
    "message": "Welcome to Dockerized Node.js App! 🚀",
```

```
    "timestamp": "2024-12-01T11:05:00.000Z",
```

```
    "environment": "production",
```

```
    "containerId": "abc123def456"
```

```
}
```

bash

```
curl http://localhost:3000/health
```

Expected Output:

json

```
{
```

```
    "status": "OK",
```

```
    "uptime": 45.123,
```

```
    "timestamp": "2024-12-01T11:05:45.000Z"
```

```
}
```

bash

```
curl http://localhost:3000/info
```

Expected Output:

json

```
{
```

```
    "app": "Dockerized Node.js Application",
```

```
    "version": "1.0.0",
```

```
"description": "A simple Express app running in a Docker container",
"features": [
    "RESTful API endpoints",
    "Health check endpoint",
    "Environment-based configuration",
    "Docker optimized"
]
}
```

Test in browser:

Open <http://localhost:3000> in your web browser.

Step 6: Container Management Commands

Check container logs:

bash

docker logs node-app

Expected Output:

text

 Server running on port 3000

 Environment: production

 Health check: <http://localhost:3000/health>

Check container health:

bash

docker inspect --format='{{.State.Health.Status}}' node-app

Expected Output:

text

healthy

View container resource usage:

bash

docker stats node-app

Expected Output:

text

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
abc123def456	node-app	0.15%	45.23MiB / 1.943GiB	2.27%	1.45kB/0B	0B/0B	25

Execute commands inside container:

bash

docker exec -it node-app sh

Inside container, you can run:

bash

whoami

ls -la

exit

Expected Output:

text

nextjs

total 40

```
drwxr-xr-x  1 nextjs  nodejs   4096 Dec 1 11:00 .
drwxr-xr-x  1 root    root     4096 Dec 1 11:05 ..
-rw-r--r--  1 nextjs  nodejs   123 Dec 1 11:00 app.js
drwxr-xr-x  1 nextjs  nodejs   4096 Dec 1 11:05 node_modules
-rw-r--r--  1 nextjs  nodejs   123 Dec 1 11:00 package.json
```

Step 7: Advanced Deployment Options

Option A: Using Docker Compose

Create docker-compose.yml:

yaml

version: '3.8'

```
services:  
  web:  
    image: my-node-app:latest  
    build: .  
    ports:  
      - "3000:3000"  
    environment:  
      - NODE_ENV=production  
      - PORT=3000  
    restart: unless-stopped  
    healthcheck:  
      test: ["CMD", "node", "-e", "require('http').get('http://localhost:3000/health', (res) => {  
        if (res.statusCode === 200) process.exit(0); else process.exit(1); }).on('error', () =>  
        process.exit(1))"]  
      interval: 30s  
      timeout: 10s  
      retries: 3  
      start_period: 40s
```

You can add more services like databases, cache, etc.

```
# redis:  
#   image: redis:alpine  
#   ports:  
#     - "6379:6379"
```

Run with Docker Compose:

```
bash  
docker-compose up -d
```

Option B: Run with Resource Limits

```
bash
```

```
docker run -d \
-p 3000:3000 \
--name node-app-limited \
--memory=512m \
--cpus=1.0 \
--restart=unless-stopped \
my-node-app:latest
```

Option C: Run Multiple Instances

bash

Instance 1

```
docker run -d -p 3001:3000 --name node-app-1 my-node-app:latest
```

Instance 2

```
docker run -d -p 3002:3000 --name node-app-2 my-node-app:latest
```

Instance 3

```
docker run -d -p 3003:3000 --name node-app-3 my-node-app:latest
```

Step 8: Monitoring and Maintenance

View all containers (including stopped):

bash

```
docker ps -a
```

Stop container:

bash

```
docker stop node-app
```

Start stopped container:

bash

```
docker start node-app
```

Remove container:

```
bash
```

```
docker rm node-app
```

Remove image:

```
bash
```

```
docker rmi my-node-app:latest
```

Step 9: Push to Docker Hub (Optional)

Login to Docker Hub:

```
bash
```

```
docker login
```

Tag image for Docker Hub:

```
bash
```

```
docker tag my-node-app:latest yourusername/my-node-app:latest
```

Push to Docker Hub:

```
bash
```

```
docker push yourusername/my-node-app:latest
```

Summary

Successfully Containerized and Deployed:

- Created a Node.js Express application
- Built optimized Dockerfile with security best practices
- Built Docker image with multiple tags
- Ran container with port mapping
- Tested application endpoints
- Explored container management commands
- Set up Docker Compose for orchestration
- Configured resource limits and health checks

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

Software Requirements:

- **Git installed**
- **GitHub account**
- **VS Code** or any text editor
- **Terminal/Git Bash**

Task 1: Repository Setup and Initial Commit

Step 1: Verify Git Installation and Configuration

Check Git version:

bash

```
git --version
```

Expected Output:

text

```
git version 2.39.2
```

Configure Git (if not done):

bash

```
git config --global user.name "Your Name"
```

```
git config --global user.email "your.email@example.com"
```

Verify configuration:

bash

```
git config --list
```

Expected Output:

text

user.name=Your Name

user.email=your.email@example.com

init.defaultbranch=main

Step 2: Create Local Repository and Project File

Create and navigate to project directory:

bash

```
mkdir git-project-management
```

```
cd git-project-management
```

Create project.md with detailed content:

markdown

```
# Project: EcoTrack - Carbon Footprint Tracker
```

Project Overview

EcoTrack is a web-based application designed to help individuals and organizations monitor and reduce their carbon footprint through intuitive tracking and actionable insights.

Problem Statement

With increasing climate change concerns, there's a growing need for accessible tools that help people understand and minimize their environmental impact. Current solutions are often complex or lack user-friendly interfaces.

Solution

EcoTrack provides:

- ****Real-time carbon emission tracking****
- ****Personalized reduction recommendations****
- ****Goal setting and progress monitoring****

- **Social sharing features for community engagement****

Target Audience

- Environmentally conscious individuals
- Small to medium businesses
- Educational institutions
- Sustainability organizations

Core Features

1. **Carbon Calculator****: Estimate emissions from transportation, energy usage, and consumption
2. **Progress Dashboard****: Visualize reduction progress over time
3. **Actionable Insights****: Personalized tips for reducing carbon footprint
4. **Community Challenges****: Group activities to promote sustainable practices

Technology Stack

- **Frontend****: React.js with TypeScript
- **Backend****: Node.js with Express
- **Database****: PostgreSQL
- **Cloud****: AWS EC2 and S3
- **Analytics****: Google Analytics integration

Project Timeline

- **Phase 1**** (Months 1-2): MVP with basic tracking
- **Phase 2**** (Months 3-4): Advanced features and mobile app
- **Phase 3**** (Months 5-6): Scaling and enterprise features

Team Structure

- Project Manager: 1
- Frontend Developers: 2
- Backend Developers: 2
- UI/UX Designer: 1
- DevOps Engineer: 1

Success Metrics

- User adoption rate
- Carbon reduction achieved by users
- User engagement metrics
- Customer satisfaction scores

Last Updated: Initial Project Setup

Step 3: Initialize Git Repository and Make Initial Commit

Initialize Git repository:

bash

git init

Expected Output:

text

Initialized empty Git repository in /path/to/git-project-management/.git/

Check repository status:

bash

git status

Expected Output:

text

On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

project.md

nothing added to commit but untracked files present (use "git add" to track)

Add file to staging area:

bash

git add project.md

Verify staged files:

bash

git status

Expected Output:

text

On branch main

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: project.md

Make initial commit:

bash

git commit -m "feat: initial project setup with comprehensive EcoTrack documentation"

- Add detailed project overview and problem statement

- Define target audience and core features

- Outline technology stack and project timeline
- Specify team structure and success metrics
- Include comprehensive project documentation"

Expected Output:

text

[main (root-commit) a1b2c3d] feat: initial project setup with comprehensive EcoTrack documentation

1 file changed, 45 insertions(+)

create mode 100644 project.md

Step 4: Create GitHub Repository and Push

On [GitHub.com](#):

1. Click "+" icon → "New repository"
2. Repository name: git-project-management
3. Description: "EcoTrack - Carbon Footprint Tracker Project"
4. Visibility: Public
5. **DO NOT** initialize with README, .gitignore, or license
6. Click "Create repository"

Back in terminal, add remote and push:

bash

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

```
git branch -M main
```

```
git push -u origin main
```

Expected Output:

text

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 1.45 KiB | 1.45 MiB/s, done.

Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

To <https://github.com/your-username/git-project-management.git>

* [new branch] main -> main

Branch 'main' set up to track remote branch 'main' from 'origin'.

Verify the push:

bash

git log --oneline --graph

Expected Output:

text

a1b2c3d (HEAD -> main, origin/main) feat: initial project setup with comprehensive EcoTrack documentation

Task 2: Branching and Merging

Step 1: Create and Switch to Feature Branch

Create and switch to feature branch:

bash

git checkout -b feature-branch

Expected Output:

text

Switched to a new branch 'feature-branch'

Verify current branch:

bash

git branch

Expected Output:

text

main

* feature-branch

Step 2: Make Significant Changes in Feature Branch

Update project.md by adding these sections at the end:

markdown

Technical Architecture Details

System Architecture

Frontend (React) → API Gateway → Backend (Node.js) → Database (PostgreSQL)

↙

Analytics Service → Data Warehouse

text

Database Schema

- **Users Table**: user_id, email, profile_data
- **CarbonEntries Table**: entry_id, user_id, activity_type, carbon_amount, timestamp
- **Goals Table**: goal_id, user_id, target_reduction, deadline
- **Challenges Table**: challenge_id, name, description, start_date, end_date

API Endpoints

- `POST /api/v1/entries` - Add new carbon entry
- `GET /api/v1/users/:id/summary` - Get user carbon summary
- `PUT /api/v1/goals/:id` - Update user goals
- `GET /api/v1/challenges` - List available challenges

Development Workflow

Git Branch Strategy

- `main` - Production-ready code
- `develop` - Integration branch
- `feature/*` - Feature development
- `hotfix/*` - Critical bug fixes
- `release/*` - Release preparation

Code Review Process

1. Create feature branch from `develop`
2. Develop and test features
3. Create Pull Request to `develop`
4. Code review by at least one team member
5. Merge after approval and CI/CD passes

Testing Strategy

- **Unit Tests**: Jest for frontend and backend
- **Integration Tests**: API endpoint testing
- **E2E Tests**: Cypress for critical user flows
- **Performance Tests**: Load testing for high-traffic endpoints

Deployment Pipeline

CI/CD Configuration

- **Build**: Node.js environment setup and dependency installation
- **Test**: Automated test suite execution
- **Security**: Vulnerability scanning with Snyk
- **Deploy**: Automated deployment to staging environment
- **Release**: Manual approval for production deployment

Environment Setup

- **Development**: Local development environments
- **Staging**: Pre-production testing environment
- **Production**: Live application environment

Additional Features Added

Mobile Application

- **React Native** for cross-platform compatibility
- **Offline functionality** for entry tracking
- **Push notifications** for goal reminders

Advanced Analytics

- **Machine Learning** for personalized recommendations
- **Trend Analysis** for long-term pattern recognition
- **Comparative Analytics** against similar user profiles

Integration Capabilities

- **Smart Home Devices** for automatic energy tracking
- **Fitness Trackers** for activity-based carbon calculations
- **E-commerce Platforms** for purchase impact analysis

Step 3: Stage and Commit Feature Branch Changes

Check what changed:

bash

git diff

Expected Output: (Shows the added content in green)

Stage the changes:

bash

git add project.md

Check status:

bash

git status

Expected Output:

text

On branch feature-branch

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: project.md

Commit with descriptive message:

bash

```
git commit -m "feat: add technical architecture and development workflow"
```

Technical Architecture:

- Add system architecture diagram and database schema
- Define API endpoints for carbon tracking
- Outline database tables and relationships

Development Workflow:

- Implement Git branch strategy
- Establish code review process
- Define comprehensive testing strategy

Deployment Pipeline:

- Set up CI/CD configuration
- Configure multiple environments
- Add security scanning

Additional Features:

- Mobile application specifications
- Advanced analytics with ML capabilities
- Third-party integration options"

Expected Output:

text

[feature-branch e4f5g6h] feat: add technical architecture and development workflow
1 file changed, 87 insertions(+)

Step 4: Switch Back to Main Branch

Switch to main branch:

bash

git checkout main

Expected Output:

text

Switched to branch 'main'

Your branch is up to date with 'origin/main'.

Verify we're on main and check file content:

bash

git branch

head -20 project.md

Expected Output:

text

* main

feature-branch

(File shows original content without new technical sections)

Step 5: Merge Feature Branch into Main

Merge the feature branch:

bash

git merge feature-branch

Expected Output:

text

Updating a1b2c3d..e4f5g6h

Fast-forward

1 file changed, 87 insertions(+)

Verify the merge was successful:

bash

```
git log --oneline --graph --all
```

Expected Output:

text

e4f5g6h (HEAD -> main, feature-branch) feat: add technical architecture and development workflow

a1b2c3d (origin/main) feat: initial project setup with comprehensive EcoTrack documentation

Check that new content is in the file:

bash

```
grep -n "Technical Architecture" project.md
```

Expected Output:

text

46:## Technical Architecture Details

Step 6: Push Updated Main Branch to GitHub

Push the merged changes:

bash

```
git push origin main
```

Expected Output:

text

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Writing objects: 100% (3/3), 2.34 KiB | 2.34 MiB/s, done.

Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

To <https://github.com/your-username/git-project-management.git>

a1b2c3d..e4f5g6h main -> main

Step 7: Verification and Best Practices

Verify Remote Repository Status

bash

```
git remote show origin
```

Expected Output:

text

```
* remote origin
```

```
  Fetch URL: https://github.com/your-username/git-project-management.git
```

```
  Push URL: https://github.com/your-username/git-project-management.git
```

```
HEAD branch: main
```

```
Remote branch:
```

```
  main tracked
```

```
Local branch configured for 'git pull':
```

```
  main merges with remote main
```

```
Local ref configured for 'git push':
```

```
  main pushes to main (up to date)
```

View Detailed Commit History

bash

```
git log --oneline --graph --all --decorate
```

Expected Output:

text

```
e4f5g6h (HEAD -> main, origin/main, feature-branch) feat: add technical architecture  
and development workflow
```

```
a1b2c3d feat: initial project setup with comprehensive EcoTrack documentation
```

Check File Differences Between Commits

bash

```
git diff a1b2c3d..e4f5g6h project.md
```

Expected Output: (Shows all the added technical content)

Step 8: Cleanup and Additional Git Operations

Delete Feature Branch (Optional)

bash

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

Expected Output:

text

```
Deleted branch feature-branch (was e4f5g6h).
```

Create README.md for Better Documentation

bash

```
echo "# Git Project Management
```

This repository demonstrates comprehensive Git and GitHub workflow management for the EcoTrack project.

Project Structure

- \` project.md\` - Complete project documentation
- Git branches demonstrate feature development workflow

Workflow Demonstrated

1. Repository initialization
2. Initial commit and GitHub push
3. Feature branch creation
4. Collaborative development
5. Branch merging
6. Remote synchronization

Learning Outcomes

- Git repository management

- Branching strategies
- Merge operations
- Remote repository synchronization" > README.md

Add and commit README:

bash

```
git add README.md
```

```
git commit -m "docs: add comprehensive README with workflow documentation"
```

```
git push origin main
```

Step 9: Verify on GitHub

On [GitHub.com](#):

1. Navigate to your repository
2. Verify:
 - o Both commits are visible in history
 - o project.md contains all merged content
 - o README.md is displayed
 - o Main branch is up to date
 - o No feature branch exists (if deleted)

Summary

✓ Task 1 Completed - Repository Setup and Initial Commit:

- Local Git repository initialized
- Comprehensive project.md created with detailed project documentation
- Initial commit with descriptive message
- Repository successfully pushed to GitHub

✓ Task 2 Completed - Branching and Merging:

- feature-branch created and switched to
- Significant technical content added to project.md
- Changes committed with detailed commit message
- Successfully switched back to main branch

- Feature branch merged into main using fast-forward merge
- Updated main branch pushed to GitHub
- Merge reflected in remote repository

Key Git Concepts Demonstrated:

- **Repository initialization** (git init)
- **Staging changes** (git add)
- **Committing** (git commit)
- **Remote repository management** (git remote, git push)
- **Branching** (git branch, git checkout -b)
- **Merging** (git merge)
- **Status checking** (git status, git log)
- **Cleanup** (git branch -d)

Slip 12)

1)

Q.1) Applying CI Principles to Web Development Using Jenkins, Git

1. Restart Docker Desktop

- Completely quit Docker Desktop from the system tray
- Wait a few seconds
- Restart Docker Desktop as Administrator

2. Check System Requirements

- Ensure your Windows version supports Docker Desktop (Windows 10/11 Pro, Enterprise, or Education)
- Verify virtualization is enabled in BIOS (VT-x/AMD-V)

3. Common Solutions

Restart Docker Service

```
bash  
# Open Command Prompt as Administrator  
net stop com.docker.service  
net start com.docker.service
```

Reset Docker to Factory Defaults

1. Open Docker Desktop
2. Go to Settings → Troubleshoot → Reset to factory defaults

Clean Reset via Command Line

```
bash  
# Stop all containers  
docker stop $(docker ps -aq)
```

```
# Remove all containers, images, volumes  
docker system prune -a --volumes
```

```
# Reset Docker data  
wsl --unregister docker-desktop  
wsl --unregister docker-desktop-data
```

4. WSL Issues (Common on Windows)

```
bash
# Check WSL status
wsl --list --verbose

# If Docker-related WSL distributions show status other than
"Running", reset them
wsl --shutdown
```

5. Check for Conflicts

- Disable VPN temporarily
- Check antivirus/firewall settings
- Ensure no other virtualization software conflicts (VirtualBox, etc.)

6. Reinstall Docker Desktop

If all else fails:

1. Uninstall Docker Desktop
2. Delete remaining Docker folders:
 - C:\Program Files\Docker
 - %APPDATA%\Docker
3. Reinstall latest version from [docker.com](https://www.docker.com)

7. Check Logs

Look at Docker Desktop logs for specific error messages:

- Windows Event Viewer
- Docker Desktop logs in Settings → Troubleshoot

Try these steps in order and let me know which step resolves the issue or if you encounter any specific error messages during the process.

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

Software Requirements:

- **Docker Desktop** (for Windows/Mac) or **Docker Engine** (for Linux)
- **Python** (for local testing)
- **Text Editor** (VS Code, Sublime Text, etc.)

Step 1: Create the Application

Create a project directory and application files:

```
bash
mkdir flask-docker-app
cd flask-docker-app
Create app.py:
python
from flask import Flask
import os
```

```
app = Flask(__name__)
```

```
@app.route('/')
def hello():
    return {
        'message': 'Hello from Dockerized Flask App!',
        'status': 'success',
        'container_id': os.uname().nodename
    }
```

```
@app.route('/health')
def health():
    return {'status': 'healthy'}
```

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

Create requirements.txt:

```
txt
Flask==2.3.3
```

Step 2: Test the Application Locally

```
bash
```

```
# Create virtual environment
```

```
python -m venv venv
```

```
source venv/bin/activate # On Windows: venv\Scripts\activate
```

```
# Install dependencies
pip install -r requirements.txt

# Run the application
python app.py
Output:
text
* Serving Flask app 'app'
* Debug mode: on
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.1.x:5000

Test in another terminal:
bash
curl http://localhost:5000
Output:
json
{
  "message": "Hello from Dockerized Flask App!",
  "status": "success",
  "container_id": "your-hostname"
}

Step 3: Create Dockerfile
Create Dockerfile (no extension):
dockerfile
# Use official Python runtime as base image
FROM python:3.9-slim

# Set working directory in container
WORKDIR /app

# Copy requirements first for better caching
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY ..

# Expose port
```

```
EXPOSE 5000

# Define environment variable
ENV FLASK_ENV=production

# Run the application
CMD ["python", "app.py"]

Step 4: Create .dockerignore File
Create .dockerignore:
gitignore
__pycache__
*.pyc
*.pyo
*.pyd
.Python
venv
.env
.git
.DS_Store

Step 5: Build Docker Image
bash
docker build -t flask-docker-app .

Output:
text
[+] Building 15.2s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> [internal] load .dockerignore
=> [internal] load metadata for docker.io/library/python:3.9-slim
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:...
=> [internal] load build context
=> CACHED [2/5] WORKDIR /app
=> [3/5] COPY requirements.txt .
=> [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> [5/5] COPY ..
=> exporting to image
=> => exporting layers
=> => writing image sha256:a1b2c3d4e5f6...
=> => naming to docker.io/library/flask-docker-app
Verify the image was created:
bash
docker images
```

Output:

text

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
flask-docker-app	latest	a1b2c3d4e5f6	2 minutes ago	200MB

Step 6: Run the Docker Container

bash

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

Output:

text

```
a1b2c3d4e5f678901234567890123456789012345678901234567890
```

```
1234
```

Check running containers:

bash

```
docker ps
```

Output:

text

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
POR	NAMES			
a1b2c3d4e5f6	flask-docker-app	"python app.py"	20 seconds ago	
Up 19 seconds	0.0.0.0:5000->5000/tcp	my-flask-app		

Step 7: Test the Containerized Application

bash

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

Output:

json

```
{  
  "message": "Hello from Dockerized Flask App!",  
  "status": "success",  
  "container_id": "a1b2c3d4e5f6"  
}
```

Test health endpoint:

bash

```
curl http://localhost:5000/health
```

Output:

json

```
{"status": "healthy"}
```

Step 8: View Container Logs

bash

```
docker logs my-flask-app
```

Output:

text

```
* Serving Flask app 'app'  
* Debug mode: on  
* Running on all addresses (0.0.0.0)  
* Running on http://127.0.0.1:5000  
* Running on http://172.17.0.2:5000
```

Step 9: Stop and Remove Container

```
bash  
# Stop container  
docker stop my-flask-app
```

```
# Remove container  
docker rm my-flask-app
```

```
# Remove image (optional)  
docker rmi flask-docker-app
```

Step 10: Using Docker Compose (Optional)

Create docker-compose.yml:

```
yaml  
version: '3.8'
```

```
services:  
  web:  
    build: .  
    ports:  
      - "5000:5000"  
    environment:  
      - FLASK_ENV=production  
    volumes:  
      - ./app  
    container_name: flask-app-compose
```

Deploy with Docker Compose:

```
bash  
docker-compose up -d
```

Output:

```
text  
[+] Building 0.0s (0/0)  
[+] Running 2/2  
  ✓ Network flask-docker-app_default Created  
  ✓ Container flask-app-compose Started
```

Final Project Structure

```
text
```

```
flask-docker-app/
├── app.py
├── requirements.txt
├── Dockerfile
├── .dockerignore
└── docker-compose.yml

Verification Commands
bash
# Check Docker version
docker --version

# Check all containers (including stopped)
docker ps -a

# Check resource usage
docker stats

# Enter running container
docker exec -it my-flask-app /bin/bash
```

This complete example demonstrates:

- Creating a basic web application
- Writing a proper Dockerfile
- Building a Docker image
- Running a container
- Testing the deployed application
- Managing containers with Docker commands

Slip13)

Q.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.

Bitbucket Repository and Branch Management

I'll guide you through creating a Bitbucket repository, managing branches, and handling pull requests with code review.

Prerequisites:

- **Bitbucket Account** (bitbucket.org)
- **Git** installed locally
- **Git Credentials** set up

Task 1: Repository Setup and Branching

Step 1: Create Repository on Bitbucket

1. Log in to Bitbucket

- Go to bitbucket.org
- Sign in to your account

2. Create New Repository

- Click "**Create**" button → "**Repository**"
- Fill in repository details:
 - **Repository name:** my-project
 - **Access level:** Private or Public
 - **Include README:** Yes
 - **Git ignore:** Python (or your preferred language)

- **Main branch:** main

Output: Repository created successfully with initial README.md file.

Step 2: Clone Repository Locally

bash

```
# Copy the repository URL from Bitbucket (HTTPS or SSH)
git clone https://your-username@bitbucket.org/your-
username/my-project.git
cd my-project
```

Output:

text

Cloning into 'my-project'...

remote: Counting objects: 3, done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 3 (delta 0), reused 0 (delta 0)

Unpacking objects: 100% (3/3), done.

Verify the local repository:

bash

ls -la

Output:

text

README.md

.gitignore

Step 3: Create Development Branch and Add Files

bash

```
# Create and switch to development branch
```

```
git checkout -b development
```

```
# Create a new file
```

```
echo "# Project Configuration" > config.txt
```

```
echo "version=1.0.0" >> config.txt
```

```
echo "author=Your Name" >> config.txt
```

```
# Create a Python script (example)
```

```
cat > app.py << EOF
```

```
def main():
    print("Hello from my project!")

if __name__ == "__main__":
    main()
EOF
Check the status:
bash
git status
Output:
text
On branch development
Untracked files:
(use "git add <file>..." to include in what will be committed)
    app.py
    config.txt
```

Step 4: Commit and Push to Development Branch

```
bash
# Add files to staging
git add .

# Commit changes
git commit -m "feat: add initial project files and
configuration"

# Push to development branch on Bitbucket
git push -u origin development
Output:
text
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (3/3), 320 bytes | 320.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for development:
```

```
remote: https://bitbucket.org/your-username/my-project/pull-requests/new?source=development
remote:
```

```
To bitbucket.org:your-username/my-project.git
```

```
* [new branch] development -> development
```

```
Branch 'development' set up to track remote branch  
'development' from 'origin'.
```

Task 2: Pull Request and Code Review

Step 1: Create Pull Request on Bitbucket

1. Navigate to Your Repository on Bitbucket

2. Click on "Pull requests" in the left sidebar

3. Click "Create pull request"

4. Fill in Pull Request Details:

- **Source branch:** development

- **Destination branch:** main

- **Title:** Feature: Add project configuration and application files

- **Description:**

```
text
```

```
## Changes
```

```
- Added project configuration file (config.txt)
```

```
- Created initial Python application (app.py)
```

```
- Updated project structure
```

```
## Testing
```

```
- [x] Verified application runs correctly
```

```
- [x] Configuration file format is valid
```

Step 2: Assign Reviewer and Add Comments

1. Assign Reviewer:

- In the pull request, click "**Reviewers**"

- Add a team member or yourself for self-review

2. Add Comments to Specific Lines:

- Navigate to "**Files changed**" tab

- Click on any line to add a comment

Example Comments:

- On config.txt: "Consider using YAML format for better readability"
- On app.py: "Add error handling for better robustness"

Step 3: Review Process (As Reviewer)

As the assigned reviewer:

1. Review the Changes:

- Go through each file in the "Files changed" tab
- Add comments or suggestions

2. Approve or Request Changes:

- Click "**Approve**" if changes look good
- Click "**Request changes**" if modifications are needed
- Add overall comments in the main discussion

Example Review Comment:

text

Overall looks good! Just a few suggestions:

1. Consider adding a requirements.txt file for Python dependencies
2. Add docstrings to the main function
3. The config file would be better in YAML format

Please address these and I'll approve the PR.

Step 4: Address Review Comments

If changes were requested:

bash

```
# Make sure you're on development branch
git checkout development
```

```
# Address the comments - example:
```

```
# Convert config.txt to config.yaml
echo "version: 1.0.0" > config.yaml
echo "author: Your Name" >> config.yaml
```

```
# Remove old config file
```

```
rm config.txt
```

```

# Add docstring to app.py
cat > app.py << EOF
def main():
    """Main application function"""
    print("Hello from my project!")

if __name__ == "__main__":
    main()
EOF

# Commit and push the changes
git add .
git commit -m "fix: address review comments - convert to
YAML, add docstrings"
git push origin development
Step 5: Merge Pull Request

```

1. Back on Bitbucket PR:

- Reviewer approves the changes
- Click "**Merge**" button
- Choose merge strategy (Squash, Merge commit, or Fast-forward)
- Add merge commit message: Merge development into main - Add initial project files

2. Delete development branch (optional):

- Check "Delete development branch after merge"

Output: Pull request merged successfully!

Step 6: Verify Merge and Update Local Repository

bash

Switch to main branch

git checkout main

Pull the latest changes from Bitbucket
git pull origin main

Verify the merged files

```
ls -la
```

Output:

text

```
README.md
```

```
.gitignore
```

```
app.py
```

```
config.yaml
```

Verification Commands

Check Branch Status

```
bash
```

```
git branch -a
```

Output:

text

```
* main
```

```
development
```

```
remotes/origin/HEAD -> origin/main
```

```
remotes/origin/main
```

```
remotes/origin/development
```

View Commit History

```
bash
```

```
git log --oneline --graph
```

Output:

text

```
* a1b2c3d (HEAD -> main, origin/main) Merge
```

```
development into main - Add initial project files
```

```
| \
```

```
| * d4e5f6a (origin/development) fix: address review
```

```
comments - convert to YAML, add docstrings
```

```
| * e7f8g9h feat: add initial project files and configuration
```

```
| /
```

```
* b2c3d4e Initial commit
```

Complete Workflow Summary

1. **Repository Created** on Bitbucket
2. **Repository Cloned** locally
3. **Development branch** created and pushed

4. **Files added and committed** to development branch
5. **Pull request created** from development to main
6. **Code review** performed with comments
7. **Changes addressed** based on feedback
8. **Pull request merged** successfully
9. **Local repository updated** with merged changes

Best Practices Demonstrated:

- **Branch naming:** Used development for feature work
- **Commit messages:** Descriptive and conventional
- **Pull request descriptions:** Clear and detailed
- **Code review:** Constructive feedback
- **Merge strategies:** Appropriate choice based on team preference

This completes the Bitbucket repository setup and branch management workflow!

Q.2) Automated Deployment with Jenkins and GitHub Task

1: Configuring Jenkins with GitHub ● Install and set up Jenkins. [15 M] ● 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

Task 1: Configuring Jenkins with GitHub

Step 1: Install Jenkins

On Windows:

```
bash  
# Download Jenkins from https://www.jenkins.io/download/  
# Install as Windows service
```

```
# Or using Chocolatey (if installed)  
choco install jenkins
```

On Ubuntu/Linux:

```
bash  
# Update system  
sudo apt update
```

```
# Install Java (required for Jenkins)  
sudo apt install openjdk-11-jdk -y
```

```
# Add Jenkins repository  
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-stable binary/ >  
/etc/apt/sources.list.d/jenkins.list'
```

```
# Install Jenkins
sudo apt update
sudo apt install jenkins -y

# Start Jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins

# Check status
sudo systemctl status jenkins
```

Step 2: Initial Jenkins Setup

1. Access Jenkins:

- Open browser and go to <http://localhost:8080>
- Get initial admin password:
bash
Windows
type "C:\Program Files\Jenkins\secrets\initialAdminPassword"

```
# Linux
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

2. Install Suggested Plugins

3. Create Admin User

Step 3: Install Required Plugins

1. Go to Manage Jenkins → Manage Plugins → Available

2. Install these plugins:

- **GitHub Integration**
- **Pipeline**
- **Git**
- **SSH Pipeline Steps**
- **Email Extension** (optional)

Step 4: Configure GitHub Integration

1. Generate GitHub Personal Access Token:

- Go to GitHub → Settings → Developer settings → Personal access tokens
- Generate new token with
scopes: repo, admin:repo_hook, write:repo_hook

2. Configure GitHub in Jenkins:

- Go to **Manage Jenkins** → **Configure System**
- Find **GitHub** section
- Add GitHub Server: <https://api.github.com>
- Add credentials (GitHub token)
- Test connection

Task 2: Creating CI/CD Pipeline

Step 1: Create Jenkins Pipeline Job

1. New Item → Pipeline

2. Name: web-app-ci-cd

3. Configure pipeline:

Step 2: Create Jenkinsfile

Create Jenkinsfile in your GitHub repository:

```
groovy
pipeline {
    agent any

    tools {
        nodejs 'nodejs' // if using Node.js
    }

    environment {
        GIT_REPO = 'https://github.com/your-username/your-repo.git'
        BRANCH = 'main'
        DEPLOY_PATH = '/var/www/html'
    }

    stages {
        stage('Clone Repository') {
            steps {
                echo 'Cloning GitHub repository...'
                git branch: "${BRANCH}", url: "${GIT_REPO}"
            }
        }

        stage('Install Dependencies') {
            steps {
                echo 'Installing dependencies...'
                // For Node.js project
                sh 'npm install'

                // For Python project, use: sh 'pip install -r requirements.txt'
                // For Java project, use: sh 'mvn clean install'
            }
        }

        stage('Build Application') {
```

```

steps {
    echo 'Building application...'
    // Example build commands
    sh 'npm run build'
    // or sh 'mvn package'
}
}

stage('Run Tests') {
    steps {
        echo 'Running tests...'
        sh 'npm test'
        // or sh 'pytest' or 'mvn test'
    }
}

stage('Archive Artifacts') {
    steps {
        echo 'Archiving build artifacts...'
        archiveArtifacts artifacts: 'dist/**/*', fingerprint: true
        // Adjust path based on your build output
    }
}

post {
    always {
        echo 'Build completed - sending notifications...'
        // Update GitHub commit status
        updateGitHubCommitStatus state: currentBuild.currentResult,
            context: 'jenkins/ci-cd'
    }
    success {
        echo 'Build succeeded!'
        // Send success notifications
        emailext (
            subject: "SUCCESS: Job '${env.JOB_NAME}",
            [${env.BUILD_NUMBER}]",
            body: "Good news! Build ${env.BUILD_URL} completed
            successfully",
            to: "developer@company.com"
    }
}
}

```

```

        )
    }
    failure {
        echo 'Build failed!'
        // Send failure notifications
        emailext(
            subject: "FAILED: Job '${env.JOB_NAME}"
            [${env.BUILD_NUMBER}]",
            body: "Build ${env.BUILD_URL} failed. Please check the
            console output.",
            to: "developer@company.com"
        )
    }
}
}

```

Step 3: Configure Pipeline in Jenkins

1. In Jenkins job configuration:

- **Pipeline → Definition:** Pipeline script from SCM
- **SCM:** Git
- **Repository URL:** Your GitHub repo URL
- **Credentials:** Add your GitHub credentials
- **Script Path:** Jenkinsfile

Task 3: Adding Deployment Step

Step 1: Extend Jenkinsfile with Deployment

Update your Jenkinsfile to include deployment:

```

groovy
pipeline {
    agent any

    tools {
        nodejs 'nodejs'
    }

    environment {
        GIT_REPO = 'https://github.com/your-username/your-repo.git'
        BRANCH = 'main'
        DEPLOY_PATH = '/var/www/html'
        SERVER_IP = 'localhost' // or your server IP
        SSH_CREDENTIALS = 'deploy-ssh-key'
    }
}
```

```

stages {
    // Previous stages (Clone, Install, Build, Test, Archive) remain
    same

    stage('Deploy to Server') {
        steps {
            echo 'Deploying application to web server...'
            script {
                if (currentBuild.result == null || currentBuild.result ==
'SUCCESS') {
                    // Deploy built application
                    sh """
                        # Stop existing service if needed
                        sudo systemctl stop myapp || true

                        # Create deployment directory
                        sudo mkdir -p ${DEPLOY_PATH}

                        # Copy built files
                        sudo cp -r dist/* ${DEPLOY_PATH}/

                        # Set proper permissions
                        sudo chown -R www-data:www-data ${DEPLOY_PATH}
                        sudo chmod -R 755 ${DEPLOY_PATH}

                        # Restart web server
                        sudo systemctl restart apache2
                        # or: sudo systemctl restart nginx
"""

                    echo 'Deployment completed successfully!'
                } else {
                    error 'Deployment skipped due to previous stage failures'
                }
            }
        }
    }

    stage('Smoke Test') {
        steps {
            echo 'Running post-deployment smoke tests...'

```

```

        sh """
            # Test if application is accessible
            curl -f http://localhost/ || exit 1
            echo 'Smoke test passed!'
        """

    }
}

}

post {
    always {
        // Update GitHub status
        updateGitHubCommitStatus state: currentBuild.currentResult,
            context: 'jenkins/ci-cd'

        // Clean workspace
        cleanWs()
    }

    success {
        echo 'CI/CD Pipeline completed successfully!'
        slackSend channel: '#deployments',
            message: "SUCCESS: ${env.JOB_NAME} - "
        ${env.BUILD_URL}"
    }

    failure {
        echo 'CI/CD Pipeline failed!'
        slackSend channel: '#deployments',
            color: 'danger',
            message: "FAILED: ${env.JOB_NAME} - ${env.BUILD_URL}"
    }
}
}
}

```

Step 2: Set up Web Server (if needed)

For Apache on Ubuntu:

```

bash
sudo apt update
sudo apt install apache2 -y
sudo systemctl start apache2
sudo systemctl enable apache2

```

For Nginx:

```
bash
sudo apt install nginx -y
sudo systemctl start nginx
sudo systemctl enable nginx
```

Step 3: Configure GitHub Webhook

1. In GitHub:

- Go to your repository → **Settings** → **Webhooks** → **Add webhook**
- **Payload URL:** `http://your-jenkins-server:8080/github-webhook/`
- **Content type:** application/json
- **Which events:** Just the push event
- **Active:** Checked

2. In Jenkins:

- Go to your pipeline job → **Configure**
- **Build Triggers** → **GitHub hook trigger for GITScm polling** → Check

Step 4: Configure SSH Deployment (Alternative)

For remote server deployment:

1. Generate SSH Key:

```
bash
ssh-keygen -t rsa -b 4096 -C "jenkins@ci-cd"
```

2. Add Public Key to Server:

```
bash
ssh-copy-id -i ~/.ssh/id_rsa.pub user@server-ip
```

3. Add SSH Credentials in Jenkins:

- **Manage Jenkins** → **Manage Credentials**
 - Add SSH username with private key
- Complete Setup Verification**

1. Test the Pipeline:

```
bash
# Make a change to your code and push
git add .
git commit -m "Test CI/CD pipeline"
git push origin main
```

2. Monitor in Jenkins:

- Watch the pipeline execution in Jenkins Blue Ocean view
- Check console output for any errors

3. Verify Deployment:

- Access your web application at `http://your-server-ip`
- Check if changes are reflected

Troubleshooting Tips

1. Webhook not triggering:

- Check Jenkins URL accessibility from GitHub
- Verify webhook secret if configured
- Check Jenkins logs

2. Deployment failures:

- Verify SSH key permissions
- Check target directory permissions
- Ensure web service is running

3. Build failures:

- Check dependency installation
- Verify test cases
- Review build logs

This setup provides a complete automated CI/CD pipeline that builds, tests, and deploys your application on every Git push to the main branch.

This response is AI-generated, for reference only.

Slip 14

Q.1) Bitbucket Repository Management 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

Task 1: Creating and Configuring a Bitbucket Repository

Step 1: Create New Repository on Bitbucket

1. Log in to Bitbucket

- Go to bitbucket.org
- Sign in to your account

2. Create New Repository

- Click **Create** button → **Repository**
- Fill in repository details:
 - **Repository name:** web-app-project
 - **Description:** A sample web application project for CI/CD demonstration
 - **Access level:** Private (recommended) or Public
 - **Include README:** (checked)
 - **Git ignore:** Select your project type (e.g., Node, Python)
 - **License:** Choose appropriate license (e.g., MIT)
- Click **Create repository**

Step 2: Clone Repository to Local Machine

Using Command Line:

bash

Clone the repository

```
git clone https://your-username@bitbucket.org/your-username/web-app-project.git
```

```
# Navigate to project directory
cd web-app-project

# Verify the repository
git status
```

Using Git GUI (like GitKraken, SourceTree):

- Open your Git GUI client
- Clone repository using the repository URL
- Provide your Bitbucket credentials

Step 3: Add and Push README.md File

1. Edit README.md:

```
bash
# Open README.md in your preferred editor
# Or create if it doesn't exist
```

2. Add comprehensive content to README.md:

```
markdown
# Web Application Project
```

```
## Description
A sample web application project demonstrating CI/CD workflows
with Bitbucket and Jenkins.
```

```
## Project Structure
web-app-project/
├── src/
├── tests/
├── docs/
└── README.md
└── package.json
text
```

```
## Installation
``` bash
npm install
```

### **Usage**

```
bash
npm start
```

### **Contributing**

1. Fork the repository
2. Create a feature branch

3. Commit your changes
4. Push to the branch
5. Create a Pull Request

#### **License**

MIT License

text

#### 3. \*\*Commit and Push Changes:\*\*

```
``` bash
```

```
# Check status
```

```
git status
```

```
# Add README.md to staging
```

```
git add README.md
```

```
# Commit with meaningful message
```

```
git commit -m "docs: Add comprehensive README with project documentation"
```

```
# Push to Bitbucket
```

```
git push origin main
```

Task 2: Setting Up Access Controls

Step 1: Invite Collaborator with Read Access

1. In Bitbucket Repository:

- Go to your repository → **Repository settings** → **User and group access**
- Click **Invite users**
- Enter collaborator's email or Bitbucket username
- Set permission level: **Read**
- Click **Invite**

2. Collaborator will receive:

- Email invitation to join the repository
- Access to view code but not modify

Step 2: Update Collaborator Access to Write

1. Modify Access Level:

- Go to **Repository settings** → **User and group access**
- Find the collaborator in the list
- Change permission from **Read** to **Write**
- Changes take effect immediately

2. Verify Write Access (as Collaborator):

```
bash
```

```
# Collaborator clones the repository
```

```
git clone https://collaborator@bitbucket.org/your-username/web-app-project.git  
cd web-app-project
```

```
# Make a small change  
echo "# Test Write Access" >> test-file.md
```

```
# Commit and push (should work with Write access)
```

```
git add test-file.md  
git commit -m "test: Verify write access"  
git push origin main
```

Step 3: Document Access Control Changes

1. Create Documentation File:

```
bash  
# Create access control documentation  
touch ACCESS_CONTROL.md
```

2. Add Content to ACCESS_CONTROL.md:

```
markdown  
# Access Control Documentation
```

```
## Repository: web-app-project  
**Owner:** [Your Name]
```

```
## Access Control History
```

User Invitations

Date	User	Initial Permission	Updated Permission	Reason
\$(date +%Y-%m-%d)	collaborator@email.com	Read	Write	Needed collaboration on feature development
\$(date +%Y-%m-%d)	reviewer@email.com	Read	-	Code review purposes

Permission Levels Used

- **Read**: Can view and clone repository
- **Write**: Can view, clone, and push to repository
- **Admin**: Full control including settings management

Security Policies

- All collaborators must use 2FA
- Regular access reviews conducted monthly

- Sensitive data never committed to repository

3. Commit and Push Documentation:

```
bash
```

```
# Add and commit access control documentation
```

```
git add ACCESS_CONTROL.md
```

```
git commit -m "docs: Add access control documentation and history"
```

```
# Push to Bitbucket
```

```
git push origin main
```

Task 3: Working with Branches

Step 1: Create New Branch

```
bash
```

```
# Create and switch to new feature branch
```

```
git checkout -b feature/update-readme
```

```
# Verify current branch
```

```
git branch
```

Step 2: Make Updates and Push Branch

1. Update README.md:

```
bash
```

```
# Edit README.md file
```

```
# Add new section for recent updates
```

Updated README.md content:

```
markdown
```

```
# Web Application Project
```

```
## Description
```

```
A sample web application project demonstrating CI/CD workflows  
with Bitbucket and Jenkins.
```

```
## Recent Updates
```

- Added comprehensive project documentation
- Implemented access control procedures
- Set up branch protection rules

```
## Project Structure
```

```
web-app-project/
```

```
  ├── src/
```

```
  ├── tests/
```

```
  ├── docs/
```

```
  └── README.md
```

```
|--- ACCESS_CONTROL.md  
└ package.json  
    text
```

```
## Installation
```

```
``` bash
```

```
npm install
```

### Usage

```
bash
```

```
npm start
```

### Development Workflow

1. Create feature branch from main
2. Make changes and test locally
3. Create pull request for review
4. Merge after approval

### Contributing

1. Fork the repository
2. Create a feature branch
3. Commit your changes
4. Push to the branch
5. Create a Pull Request

### License

```
MIT License
```

```
text
```

```
2. **Commit and Push Feature Branch:**
```

```
``` bash
```

```
# Stage changes
```

```
git add README.md
```

```
# Commit with descriptive message
```

```
git commit -m "docs: Add recent updates section and development workflow to README"
```

```
# Push feature branch to Bitbucket
```

```
git push origin feature/update-readme
```

Step 3: Create and Merge Pull Request

1. Create Pull Request in Bitbucket:

- o Go to your repository in Bitbucket
- o Click **Pull requests** → **Create pull request**
- o **Source branch:** feature/update-readme

- **Destination branch:** main
- **Title:** docs: Update README with recent changes and workflow
- **Description:**

```
text
## Changes Made
- Added Recent Updates section documenting current improvements
- Included Development Workflow guidelines
- Enhanced project structure documentation
```
- **## Review Checklist**
 - [] Documentation is clear and comprehensive
 - [] No sensitive information included
 - [] Follows project documentation standards

2. Review Process:

- Assign reviewers if available
- Reviewers can comment, request changes, or approve
- Address any review comments by pushing more commits to the branch

3. Merge Pull Request:

- After approval, click **Merge**
- Choose merge method:
 - **Merge commit** (recommended - preserves history)
 - **Squash** (combines all commits into one)
 - **Fast-forward** (linear history)
- Add merge commit message if needed
- Click **Merge**

4. Clean Up:

- Delete the feature branch after merge (Bitbucket usually prompts for this)
- Or delete manually:

```
bash
git push origin --delete feature/update-readme
git branch -d feature/update-readme
```

Additional Best Practices

Set Up Branch Protection (Optional but Recommended)

1. Repository settings → Branch permissions

2. Add rule for main branch:

- **Require approvals:** 1
- **Require all checks to pass:** ✓
- **Prevent force pushes:** ✓

Verify Final State

```
bash
# Return to main branch and pull latest changes
```

```
git checkout main  
git pull origin main  
  
# Verify README updates are present  
cat README.md  
  
# Check git log for complete history  
git log --oneline -5  
Create Tag for Release (Optional)  
bash  
# Create annotated tag for this release  
git tag -a v1.0.0 -m "Release version 1.0.0 with updated documentation"  
  
# Push tags to Bitbucket  
git push origin --tags
```

This completes all three tasks for Bitbucket repository management.
You've successfully:

- Created and configured a Bitbucket repository
- Managed access controls with documentation
- Implemented branch workflow with pull requests
- Maintained proper documentation throughout the process

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

Applying CI/CD Principles to Web Development Using Jenkins, Git and Local HTTP Server

I'll guide you through setting up a complete CI/CD pipeline using Jenkins, Git, and a local HTTP server (Nginx) for web development.

Architecture Overview:

text

Git Repository (Bitbucket/GitHub)

↓ (Webhook)

Jenkins Server (CI/CD Pipeline)

↓

Build & Test → Deploy → Nginx Web Server

Prerequisites:

- **Jenkins** installed
- **Git** installed
- **Nginx/Apache** installed
- **Java** (for Jenkins)
- **Node.js** (for sample web app)

Step 1: Set Up the Web Application

Create a Simple Web Application

bash

Create project directory

mkdir ci-cd-web-app

cd ci-cd-web-app

Initialize Git repository

git init

Create package.json

cat > package.json << EOF

{

 "name": "ci-cd-web-app",

 "version": "1.0.0",

 "description": "Sample web app for CI/CD demo",

 "main": "app.js",

```
"scripts": {
  "start": "node app.js",
  "test": "jest",
  "build": "echo 'Building application...'"
},
"devDependencies": {
  "jest": "^29.0.0"
},
"dependencies": {
  "express": "^4.18.0"
}
}
EOF

# Create Express.js application
cat > app.js << EOF
const express = require('express');
const app = express();
const port = 3000;

app.use(express.static('public'));

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/public/index.html');
});

app.get('/health', (req, res) => {
  res.json({
    status: 'healthy',
    timestamp: new Date().toISOString(),
    version: '1.0.0'
  });
});

app.listen(port, () => {
  console.log(`App running on http://localhost:${port}`);
});

module.exports = app;
EOF
```

```
# Create public directory and HTML file
mkdir public
cat > public/index.html << EOF
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>CI/CD Web App</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 40px;
            background: linear-gradient(135deg, #667eea 0%, #764ba2
100%);
            color: white;
        }
        .container {
            max-width: 800px;
            margin: 0 auto;
            padding: 20px;
            background: rgba(255,255,255,0.1);
            border-radius: 10px;
        }
        h1 { color: #fff; }
        .build-info {
            background: rgba(0,0,0,0.3);
            padding: 10px;
            border-radius: 5px;
            margin: 10px 0;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>⚡ CI/CD Pipeline Demo</h1>
        <p>This application is automatically deployed using Jenkins
CI/CD</p>
    <div class="build-info">
```

```

<strong>Build Information:</strong>
<div id="build-info">Loading...</div>
</div>

<h2>Features:</h2>
<ul>
    <li>Automated testing with Jest</li>
    <li>Continuous Integration with Jenkins</li>
    <li>Automated deployment to Nginx</li>
    <li>Git webhook triggers</li>
</ul>
</div>

<script>
// Fetch build information
fetch('/build-info.json')
.then(response => response.json())
.then(data => {
    document.getElementById('build-info').innerHTML = `

        Version: ${data.version} <br>
        Build Date: ${data.buildDate} <br>
        Commit: ${data.commitId} <br>
        Environment: ${data.environment}

    `;
})
.catch(error => {
    document.getElementById('build-info').innerHTML = 'Build
info not available';
});
</script>
</body>
</html>
EOF

# Create test file
mkdir __tests__
cat > __tests__/app.test.js << EOF
const request = require('supertest');
const app = require('../app');

describe('Web Application Tests', () => {

```

```

    test('Health endpoint should return 200', async () => {
      const response = await request(app).get('/health');
      expect(response.statusCode).toBe(200);
      expect(response.body.status).toBe('healthy');
    });

    test('Root endpoint should serve HTML', async () => {
      const response = await request(app).get('/');
      expect(response.statusCode).toBe(200);
      expect(response.text).toContain('<!DOCTYPE html>');
    });
  });
EOF

# Create build info generator script
cat > generate-build-info.js << EOF
const fs = require('fs');
const { execSync } = require('child_process');

const buildInfo = {
  version: process.env.APP_VERSION || '1.0.0',
  buildDate: new Date().toISOString(),
  commitId: process.env.GIT_COMMIT || 'local',
  environment: process.env.NODE_ENV || 'development'
};

fs.writeFileSync('public/build-info.json', JSON.stringify(buildInfo, null, 2));
console.log('Build info generated:', buildInfo);
EOF

Initialize Git and Push to Repository
bash
# Add files to git
git add .

# Commit initial version
git commit -m "feat: initial web application setup"

# Create Bitbucket repository and add remote
git remote add origin https://bitbucket.org/your-username/ci-cd-web-app.git

```

```
git push -u origin main
```

Step 2: Install and Configure Jenkins

Install Jenkins (Ubuntu/Debian)

bash

Install Java

```
sudo apt update
```

```
sudo apt install openjdk-11-jdk -y
```

Add Jenkins repository

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

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

Install Jenkins

```
sudo apt update
```

```
sudo apt install jenkins -y
```

Start Jenkins

```
sudo systemctl start jenkins
```

```
sudo systemctl enable jenkins
```

Check status

```
sudo systemctl status jenkins
```

Output:

text

● jenkins.service - Jenkins Continuous Integration Server

```
    Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
    Active: active (running) since ...
```

Access Jenkins and Initial Setup

1. **Open browser:** <http://localhost:8080>

2. **Get initial admin password:**

bash

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

3. **Install suggested plugins**

4. **Create admin user**

Install Required Jenkins Plugins

1. Go to **Manage Jenkins → Manage Plugins → Available**

2. Install these plugins:

- **Git Plugin** (usually pre-installed)
- **NodeJS Plugin**
- **Pipeline Plugin**
- **HTML Publisher Plugin**
- **JUnit Plugin**

Configure System Settings

1. Manage Jenkins → Global Tool Configuration

2. NodeJS Installation:

- Name: nodejs-18
 - Check "Install automatically"
 - Version: 18.x
 - Global npm packages: jest junit
-

Step 3: Set Up Nginx Web Server

Install and Configure Nginx

bash

Install Nginx

```
sudo apt install nginx -y
```

Create deployment directory

```
sudo mkdir -p /var/www/ci-cd-app
```

```
sudo chown -R $USER:$USER /var/www/ci-cd-app
```

Create Nginx configuration

```
sudo nano /etc/nginx/sites-available/ci-cd-app
```

Nginx Configuration (/etc/nginx/sites-available/ci-cd-app):

```
nginx
```

```
server {
```

```
    listen 80;
```

```
    server_name localhost;
```

```
    root /var/www/ci-cd-app;
```

```
    index index.html;
```

```
    location / {
```

```
        try_files $uri $uri/ =404;
```

```
}
```

```
    location /health {
```

```
        proxy_pass http://localhost:3000;
```

```
        proxy_set_header Host $host;
```

```

}

# Enable CORS for API
location /api {
    add_header 'Access-Control-Allow-Origin' '*' always;
    add_header 'Access-Control-Allow-Methods' 'GET, POST,
OPTIONS' always;
    add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-
Requested-With,If-Modified-Since,Cache-Control,Content-
Type,Range' always;
}
bash
# Enable the site
sudo ln -s /etc/nginx/sites-available/ci-cd-app /etc/nginx/sites-
enabled/
sudo nginx -t # Test configuration

# Restart Nginx
sudo systemctl restart nginx
sudo systemctl enable nginx
Output:
text
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful

```

Step 4: Create Jenkins Pipeline

Create Jenkinsfile in Project

Create Jenkinsfile in project root:

```

groovy
pipeline {
    agent any

    tools {
        nodejs "nodejs-18"
    }

    environment{
        APP_NAME = "ci-cd-web-app"
        DEPLOY_DIR = "/var/www/ci-cd-app"
        NODE_ENV = "production"
    }
}
```

```
}

stages {
    stage('Checkout') {
        steps {
            git branch: 'main',
            url: 'https://bitbucket.org/your-username/ci-cd-web-app.git'

            script {
                currentBuild.displayName = "BUILD-"
                ${env.BUILD_NUMBER}"
                currentBuild.description = "Commit:
${env.GIT_COMMIT.take(8)}"
            }
        }
    }

    stage('Install Dependencies') {
        steps {
            sh 'npm install'
        }
    }

    stage('Run Tests') {
        steps {
            sh 'npm test -- --ci --reporters=default --reporters=jest-junit'
        }
        post {
            always {
                junit 'junit.xml'
            }
        }
    }

    stage('Build') {
        steps {
            sh ""
            npm run build
            node generate-build-info.js
            export APP_VERSION="1.0.${env.BUILD_NUMBER}"
            export GIT_COMMIT="${env.GIT_COMMIT}"
        }
    }
}
```

```

        """
    }
}

stage('Security Scan'){
    steps{
        sh 'npm audit --audit-level moderate || true'
    }
}

stage('Deploy to Staging'){
    steps{
        sh ""
        echo "Deploying to ${DEPLOY_DIR}"
        sudo cp -r public/* ${DEPLOY_DIR}/
        sudo cp package.json app.js ${DEPLOY_DIR}/
        sudo chown -R www-data:www-data ${DEPLOY_DIR}
        """
    }
}

stage('Start Application'){
    steps{
        sh ""
        # Stop any existing process
        pkill -f "node app.js" || true
        sleep 2

        # Start the application
        cd ${DEPLOY_DIR}
        npm install --production
        nohup node app.js > app.log 2>&1 &
        echo "Application started on port 3000"
        """
    }
}

post{
    always{
        publishHTML([

```

```

        allowMissing: false,
        alwaysLinkToLastBuild: true,
        keepAll: true,
        reportDir: 'public',
        reportFiles: 'index.html',
        reportName: 'Web App Report'
    ])
}

script {
    // Cleanup workspace
    sh 'rm -rf node_modules'
}
}

success {
    emailext (
        subject: "SUCCESS: Job '${env.JOB_NAME}"
        [${env.BUILD_NUMBER}]",
        body: "The build ${env.BUILD_URL} completed successfully.",
        to: "developer@company.com"
    )
}

slackSend(
    channel: '#builds',
    message: "  Build Successful: ${env.JOB_NAME}
#${env.BUILD_NUMBER}\nCommit:
${env.GIT_COMMIT.take(8)}\nURL: ${env.BUILD_URL}"
)
}

failure {
    emailext (
        subject: "FAILED: Job '${env.JOB_NAME}"
        [${env.BUILD_NUMBER}]",
        body: "The build ${env.BUILD_URL} failed. Please check the
console output.",
        to: "developer@company.com"
    )
}

slackSend(
    channel: '#builds',

```

```
        message: "✖ Build Failed: ${env.JOB_NAME}  
#${env.BUILD_NUMBER}\nCommit:  
${env.GIT_COMMIT.take(8)}\nURL: ${env.BUILD_URL}"  
    )  
}  
}  
}  
}
```

Create Jenkins Pipeline Job

1. **New Item → Pipeline**
 2. **Name:** ci-cd-web-app-pipeline
 3. **Pipeline Definition:** Pipeline script from SCM
 4. **SCM:** Git
 5. **Repository URL:** Your Bitbucket repository URL
 6. **Script Path:** Jenkinsfile
-

Step 5: Configure Git Webhooks

Set Up Bitbucket Webhook

1. Go to your Bitbucket repository → **Settings → Webhooks**
2. **Add webhook:**

- **Name:** Jenkins CI/CD
- **URL:** http://your-jenkins-server:8080/bitbucket-hook/
- **Triggers:** Check "Repository Push"

Configure Jenkins for Webhooks

bash

Configure Jenkins to accept webhooks (if needed)

```
sudo nano /var/lib/jenkins/config.xml
```

Ensure this line exists:

xml

```
<useSecurity>true</useSecurity>
```

Step 6: Test the CI/CD Pipeline

Make a Code Change and Push

bash

Update the web application

```
echo "<!-- Build: ${date} -->" >> public/index.html
```

Commit and push changes

```
git add .
```

```
git commit -m "feat: add build timestamp to UI"
```

```
git push origin main
```

Monitor Jenkins Pipeline

1. **Go to Jenkins → ci-cd-web-app-pipeline**
2. **View build progress** through stages
3. **Check console output**

Expected Console Output:

```
text
Started by GitHub push by your-username
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/ci-cd-web-app-
pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Checkout)
[Pipeline] git
Fetching changes from the remote Git repository
...
[Pipeline] stage
[Pipeline] { (Install Dependencies)
[Pipeline] sh
+ npm install
added 152 packages in 3.2s
...
[Pipeline] stage
[Pipeline] { (Run Tests)
[Pipeline] sh
+ npm test
PASS __tests__/app.test.js
...
[Pipeline] stage
[Pipeline] { (Deploy to Staging)
[Pipeline] sh
+ sudo cp -r public/* /var/www/ci-cd-app/
+ sudo chown -R www-data:www-data /var/www/ci-cd-app
```

Verify Deployment

```
bash
# Check if application is running
curl http://localhost:3000/health
```

```
# Check Nginx serving
```

```
curl http://localhost
```

Expected Output:

```
json
// Health endpoint
{
  "status": "healthy",
  "timestamp": "2024-01-15T10:30:00.000Z",
  "version": "1.0.0"
}

// Web page should show CI/CD demo page
```

Step 7: Monitoring and Logs

Set Up Application Monitoring

```
bash
# Create log directory
sudo mkdir -p /var/log/ci-cd-app
sudo touch /var/log/ci-cd-app/app.log
sudo chown jenkins:jenkins /var/log/ci-cd-app/app.log
```

```
# Monitor logs
tail -f /var/log/ci-cd-app/app.log
```

Jenkins Build Statistics

1. Jenkins → ci-cd-web-app-pipeline → Build History
 2. View test results → Test Result Trend
 3. Check build duration trends
-

Complete CI/CD Workflow Summary

- ✓ Code Commit → Git Repository
- ✓ Webhook Trigger → Jenkins Pipeline
- ✓ Automated Testing → Jest Test Suite
- ✓ Build Process → Generate build artifacts
- ✓ Security Scan → npm audit
- ✓ Automated Deployment → Nginx Web Server
- ✓ Application Start → Node.js server
- ✓ Monitoring → Logs and health checks

Verification Commands

```
bash
# Check Jenkins status
sudo systemctl status jenkins

# Check Nginx status
```

```
sudo systemctl status nginx
```

```
# Check application health
curl http://localhost/health
```

```
# View Jenkins logs
sudo tail -f /var/log/jenkins/jenkins.log
```

```
# Check deployed files
ls -la /var/www/ci-cd-app/
```

This complete CI/CD pipeline demonstrates:

- **Continuous Integration:** Automated testing on every commit
- **Continuous Deployment:** Automatic deployment to staging
- **Quality Gates:** Tests and security scans
- **Monitoring:** Health checks and logging
- **Notifications:** Email and Slack integration

The pipeline will automatically run on every git push, ensuring your web application is always tested and deployed consistently!

Slip 15

Q.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.

Task 1: Repository Setup and Branching

A. Create a Repository on Bitbucket (Web GUI)

1. Visit bitbucket.org
2. Log in, click the "+" icon → “Create repository”
3. Name it (e.g. devops-demo)
4. Click "Create repository"
5. Copy the HTTPS clone URL (it will look like:

text

```
https://bitbucket.org/yourusername/devops-demo.git
```

B. Clone Repository Locally

bash

```
git clone https://bitbucket.org/yourusername/devops-demo.git  
cd devops-demo
```

C. Create Development Branch and Add File

bash

```
git checkout -b development  
echo "Some new features!" > feature.txt  
git add feature.txt  
git commit -m "Add feature.txt to development branch"  
git push -u origin development
```

Explanation:

- git checkout -b development creates and switches to a new branch
 - feature.txt is a sample new file
-

Task 2: Pull Request and Code Review

A. Create Pull Request in Bitbucket (Web GUI)

1. Go to your repo on Bitbucket

2. Click Branches; find development
 3. Click “Create pull request” or “Pull request” on the development branch page
 4. Set source as development, destination as main
 5. Write a title (e.g., "Merge new feature into main")
-

B. Assign Reviewer / Self-Review and Comment

1. Assign a reviewer (add a collaborator’s username or use yours for self-review)
 2. At the bottom, add comments or feedback (e.g., “Feature.txt adds new documentation”)
 3. Click Create pull request
-

C. Merge the Pull Request (after approval)

1. Reviewer reviews and approves (or you click “Approve”)
 2. Click Merge in the pull request page
 3. Changes from development branch are merged into main
-

D. (Optional) Check Commit History Locally

```
bash  
git checkout main  
git pull  
git log --oneline --reverse
```

Q 2 Q.2) Create a simple project using any programming language and perform some operation on the project using git. [15 M] Make your project as a git repository add your file staging area and commit changes with a descriptive message.

Step 1: Create a Simple Project Directory

```
bash  
mkdir my_simple_project  
cd my_simple_project
```

Step 2: (Python Example) Create a Code File

```
bash  
echo 'print("Hello, Git World!")' > hello.py  
(If using C, Java, HTML, etc., just create your main file instead)
```

Step 3: Make Directory a Git Repository

```
bash
```

```
git init
```

Step 4: Add File to Staging Area

```
bash  
git add hello.py
```

- This tracks your new file for commit.
-

Step 5: Commit Your Changes with Descriptive Message

```
bash  
git commit -m "Initial commit: Add hello.py with greeting message"
```

Step 6: (Perform an Operation) Edit the File and Make Another Commit

- Edit the file to add a second line:

```
bash  
echo 'print("Changing the file!")' >> hello.py
```

- Stage and commit the change:

```
bash  
git add hello.py  
git commit -m "Add second line to hello.py: demonstration change"
```

Step 7: View Status and Commit Log

```
bash  
git status  
git log --oneline
```

slip 16

Q.1) Create a simple Java project using Maven. adding dependencies, and Configuring the project's POM file and compile code using maven tool

Step 1: Generate a Simple Maven Project

Command:

bash

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

Expected Output (abbreviated):

text

```
[INFO] ... BUILD SUCCESS
```

```
[INFO] ... Project created at ./demo-maven
```

Step 2: Change Into Your Project Directory

Command:

bash

```
cd demo-maven
```

Step 3: Add a Dependency to pom.xml

Open pom.xml (use Notepad, VS Code, etc.) and inside <dependencies>, add:

```
xml  
<dependency>  
  <groupId>junit</groupId>  
  <artifactId>junit</artifactId>  
  <version>4.13.2</version>  
  <scope>test</scope>  
</dependency>
```

No terminal output for editing, but your pom.xml will now contain:

```
xml  
<!-- pom.xml snippet -->  
<dependencies>  
  <dependency>  
    <groupId>junit</groupId>  
    <artifactId>junit</artifactId>  
    <version>4.13.2</version>  
    <scope>test</scope>  
  </dependency>  
</dependencies>
```

Step 4: Compile the Project

Command:

bash

mvn compile

Expected Output:

text

[INFO] Scanning for projects...

[INFO] -----< com.example:demo-maven >-----

[INFO] ... Compilation successful

[INFO] BUILD SUCCESS

Step 5: Run Maven Tests

Command:

bash

mvn test

Expected Output:

text

TESTS

Running com.example.AppTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: X sec

[INFO] BUILD SUCCESS

Step 6: Run Project (Optional)

The default skeleton includes a simple App.java:

src/main/java/com/example/App.java:

java

package com.example;

public class App {

public static void main(String[] args) {

 System.out.println("Hello World!");

}

}

To run it:

bash

mvn exec:java -Dexec.mainClass="com.example.App"

Expected Output:

text

Hello World!

Step 7: Package (Create JAR file, Optional)

Command:

```
bash
```

```
mvn package
```

Expected Output:

```
text
```

```
[INFO] Building jar: demo-maven/target/demo-maven-1.0-SNAPSHOT.jar
```

```
[INFO] BUILD SUCCESS
```

Q2 Q.2) Applying CI/CD Principles to Web Development Using Jenkins, Git and Local HTTP Server (e.g., Apache or Nginx).

Step 1: Install the Required Tools

- Git:

[Install Git for Windows](#)

- Jenkins:

[Download Jenkins WAR \(easy for demo\)](#)

- Nginx:

[Install Nginx for Windows](#) or on Linux use sudo apt install nginx

Step 2: Create and Push a Simple Web Project to GitHub

On your machine:

```
bash
```

```
mkdir web_cicd_demo
```

```
cd web_cicd_demo
```

```
echo '<h1>Hello CI/CD!</h1>' > index.html
```

```
git init
```

```
git add index.html
```

```
git commit -m "Initial commit: Add index.html"
```

```
# On GitHub, create repo (example: web-cicd-demo), then:
```

```
git remote add origin https://github.com/yourusername/web-cicd-demo.git
```

```
git branch -M main
```

```
git push -u origin main
```

Step 3: Configure Nginx to Serve Your Project

- Copy (or symlink) index.html into the Nginx web root (/usr/share/nginx/html on Linux, C:\nginx\html\ on Windows).

Step 4: Start Jenkins

- In the folder with jenkins.war:

```
bash
java -jar jenkins.war
```
 - Open <http://localhost:8080> in your browser.
 - Complete initial setup (get admin password from terminal/output).
-

Step 5: Configure Jenkins Job for CI/CD

1. Create a new Freestyle project, name it WebDemoDeploy.
 2. Source Code Management:
 - Select Git and paste your GitHub repo URL.
 - Set credentials if repo is private.
 3. Build Triggers:
 - Enable "Poll SCM" (* * * * * for demo, checks every minute), or set up a GitHub webhook.
 4. Build Steps:
Add an "Execute shell" step (Linux/Mac) or "Execute Windows batch command" (Windows).
Example for Linux:

```
bash
cp index.html /usr/share/nginx/html/index.html
```

Example for Windows (edit paths if needed):

```
text
copy index.html C:\nginx\html\index.html
```
 5. Save the job and click “Build Now” to test.
-

Step 6: Test Your CI/CD Pipeline

- Edit your local index.html (change the text), commit, and push to GitHub:

```
bash
echo '<h1>Hello CI/CD -- updated!</h1>' > index.html
git add index.html
git commit -m "Update index.html"
git push
```
 - Jenkins (via poll or webhook) will automatically:
 - Clone latest code from GitHub
 - Copy the file to Nginx web root
 - Refresh your browser at <http://localhost:8080> (or <http://localhost> if default)—you’ll see your updated page!
-

Expected Output

- Jenkins Console Output:
Shows steps—Git, build, copy file, finished successfully.
- Nginx/Web Browser:
Shows updated web page every time you push new changes.
- Commit log in Git:
Shows history of deployments.

Slip 17

Q1 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.

Soln: Same as slip 15 Q1

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.

Step 1: Create a Simple Project

```
bash
mkdir github_demo
cd github_demo
echo "Hello, GitHub!" > greet.txt
```

Step 2: Initialize as a Git Repository

```
bash
git init
```

Step 3: Commit the Initial File

```
bash
git add greet.txt
git commit -m "Initial commit: Add greet.txt with greeting"
```

Step 4: Create a Repository on GitHub (Web interface)

- Go to <https://github.com>
- Click New repository
- Name it (e.g., github-demo)
- Leave it empty (DO NOT initialize with a README or .gitignore)
- Click Create repository
- Copy the HTTPS repo URL (<https://github.com/yourusername/github-demo.git>)

Step 5: Add Remote and Push Your Project

```
bash
```

```
git remote add origin https://github.com/yourusername/github-d  
demo.git  
git branch -M main  
git push -u origin main
```

Step 6: Perform an Operation (Edit, Commit Another Change)
bash

```
echo "This is an update to the project." >> greet.txt  
git add greet.txt  
git commit -m "Add second line to greet.txt: update message"  
git push
```

Step 7: Display Chronological History of Commits

bash
git log --oneline --reverse

Expected Output:

```
text  
8d4d678 Initial commit: Add greet.txt with greeting  
6bc1e1a Add second line to greet.txt: update message
```

Slip 18

Q.1) Applying CI Principles to Web Development Using Jenkins, Git

Step 1: Prerequisites

- Git: Installed on your system ([install link](#))
 - Jenkins: [Download and run Jenkins](#)
 - A simple web project (HTML/CSS/JS, or any language you like)
 - (*Optionally: GitHub/GitLab/Bitbucket repository for remote storage*)
-

Step 2: Create a Simple Project and Git Repository

bash

```
mkdir jenkins_ci_demo
```

```
cd jenkins_ci_demo
```

```
echo "<h1>Hello, Jenkins!</h1>" > index.html
```

```
git init
```

```
git add index.html
```

```
git commit -m "Initial commit: Add basic webpage"
```

(Optional, to push to GitHub:)

bash

```
git remote add origin
```

```
https://github.com/yourusername/jenkins\_ci\_demo.git
```

```
git branch -M main
```

```
git push -u origin main
```

Step 3: Start Jenkins

If using the .war file:

bash

```
java -jar jenkins.war
```

Then open <http://localhost:8080> in your browser.

Set up an admin account (first-time only).

Step 4: Create a Jenkins Job for Continuous Integration

1. In Jenkins, click New Item → Freestyle project
Name it: web-ci-demo
2. Source Code Management:
Choose Git and enter your project's repo URL (local path or GitHub HTTPS/SSH URL).
3. Build Triggers:
 - Choose “Poll SCM” and enter H/2 * * * * (polls every 2 minutes for demo), OR, set up a webhook in GitHub to trigger Jenkins on each push.
4. Build Steps:
 - For web projects, use a simple shell step:

```
bash
echo "Building site..."
ls -l    # List files (demo: can add build/test commands)
```

- For more complex sites: run npm, pip, tests, etc.

Step 5: Simulate a Change (to Trigger CI Build)

```
bash
echo "<!-- Updated for CI test -->" >> index.html
git add index.html
git commit -m "Update index.html to trigger Jenkins build"
git push # (if using remote)
```

- Jenkins will detect the new commit and automatically run the job.

Step 6: View Results

- In Jenkins, click your job, then Build History → the latest build.
- Click on it, then “Console Output” to see the build steps.
(You’ll see your echo/ls commands or build output.)

Expected Output

- Jenkins Console Output:
text
Building site...
total 4
-rw-r--r-- 1 user user 39 date index.html
Finished: SUCCESS
- Build Status: Green “Success” for each triggered build.

Q 2 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.

ask 1: Repository Setup and Initial Commit

1. Create Local Project and Description File

bash

```
mkdir my_repo_demo
```

```
cd my_repo_demo
```

```
echo "## My Hypothetical Project\n This project automates daily  
backups." >
```

```
project.md
```

2. Initialize Git, Add, Commit

bash

```
git init
```

```
git add project.md
```

```
git commit -m "Add project.md with initial project description"
```

3. Create GitHub Repo and Push Initial Commit

- Go to github.com, create a new repo (e.g., my-repo-demo).

Leave it EMPTY (no README).

bash

```
git remote add origin https://github.com/yourusername/my-repo-  
demo.git
```

```
git branch -M main
```

```
git push -u origin main
```

Task 2: Branching and Merging

1. Create and Switch to new Branch

bash

```
git checkout -b feature-branch
```

2. Edit project.md (Add Features or Update Description)

bash

```
echo "- Feature: Scheduled nightly backup\n- Feature: Email  
notifications" >>
```

```
project.md
```

3. Stage and Commit in feature-branch

bash

```
git add project.md
```

```
git commit -m "Add features to project.md in feature-branch"
```

```
git push -u origin feature-branch
```

4. Switch back to main and Merge

bash

```
git checkout main
```

```
git pull      # get any changes from GitHub  
git merge feature-branch  
git push      # update GitHub with merge
```

Slip 19

Q1 Q.1) 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.

Soln same as slip 18 Q2

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.

Soln same as slip 3 q2

Slip 20

Q1 Applying CI Principles to Web Development Using Jenkins, Git Soln slip 18 Q 1

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

Create a simple project locally:

```
bash
mkdir github_demo_project
cd github_demo_project
echo "Hello, GitHub!" > hello.txt
```

2. Initialize Git, add and commit:

```
bash
git init
git add hello.txt
git commit -m "Initial commit: add hello.txt"
```

3. Create a repository on GitHub (Website)

- Go to github.com
 - Click New repository
 - Name it (e.g., github-demo-project)
 - Leave it EMPTY, create, and copy the HTTPS repo URL.
-

4. Connect and push to GitHub:

```
bash
git branch -M main
git remote add origin https://github.com/yourusername/github-
demo-project.git
git push -u origin main
```

5. Create a new branch:

```
bash
git checkout -b feature-branch
echo "A new line from the branch." >> hello.txt
git add hello.txt
git commit -m "Add a new line in feature-branch"
git push -u origin feature-branch
```

6. Merge the branch into main (using GitHub Web):

- Go to your repo on GitHub.
 - You'll see a "Compare & pull request" button (for feature-branch). Click it.
 - Click "Merge pull request", then "Confirm merge".
-

7. Pull changes to your local machine (in main branch):

```
bash  
git checkout main  
git pull
```

8. Display chronological commit history:

```
bash  
git log --oneline --reverse
```

Slip 21

Q1 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

Step-by-Step: Bitbucket Branch/File Workflow

A. Platforms/Tools Used:

- Git Bash (or Command Prompt/Terminal): For local Git commands
- Bitbucket Web Interface: For merging branches (pull requests)

1. Clone your Bitbucket repo (Platform: Git Bash/Terminal)

bash

```
git clone https://bitbucket.org/yourusername/your-repo.git  
cd your-repo
```

2. Create a new branch (Git Bash/Terminal)

bash

```
git checkout -b my-feature-branch
```

3. Create a new file and commit it (Git Bash/Terminal)

bash

```
echo "This is some content in the new file." > newfile.txt  
git add newfile.txt  
git commit -m "Add newfile.txt to my-feature-branch"  
git push -u origin my-feature-branch
```

4. Merge into main using Bitbucket web interface (Browser, bitbucket.org)

1. Go to bitbucket.org and open your repository.
2. Click Branches in the sidebar and find my-feature-branch.
3. Click Create pull request next to your branch.
4. Set destination branch as main (or master).
5. Click Create pull request.
6. Review, then click Merge.

5. Pull merged changes to your local main (Git Bash/Terminal)

bash

```
git checkout main  
git pull
```

**Q2 Applying CI Principles to Web Development Using Jenkins, Git
Soln : Slip 20 Q1**

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.

Soln Slip 16 Q1

Q 2 Q.2) 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>).

Platforms Needed

- Docker Desktop (Windows/Mac) or Docker CLI (Linux) — for Docker operations
- Text Editor (VS Code, Notepad, nano, etc.) — to create files
- Web Browser — to view your app at <http://localhost:8080>

Step-by-Step Solution

1. Install Docker

- Platform: Docker Desktop [Download here](#)
- Install and open Docker Desktop.
(On Linux, install with your package manager: e.g., sudo apt install docker.io)

2. Create your project folder and HTML file

- Platform: Text Editor

```
bash
mkdir docker-hello
cd docker-hello
echo "<h1>Hello, World!</h1>" > index.html
```

Or, open your editor and save a file called index.html with this content.

3. Create Dockerfile

- Platform: Text Editor

Create a file named Dockerfile (no extension), with this content:

text

FROM nginx:alpine

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

(Uses an official web server image: nginx)

4. Build the Docker image

- Platform: Docker CLI (Terminal/Git Bash/PowerShell)

bash

docker build -t hello-nginx .

- -t hello-nginx tags your image as "hello-nginx"

- The dot (.) means "use current folder"

Expected output (abbreviated):

text

Successfully built <image-id>

Successfully tagged hello-nginx:latest

5. Run the container, map it to local port 8080

- Platform: Docker CLI

bash

docker run -d -p 8080:80 --name hello-web hello-nginx

- -d: run in background

- -p 8080:80: map local port 8080 to container's port 80

Expected output:

A long container ID string.

6. View your application in the browser

- Platform: Web Browser

Go to: <http://localhost:8080>

You should see:

Hello, World!

7. (Optional) Stop and remove container

bash

docker stop hello-web

docker rm hello-web

Slip 23

Q1 Applying CI Principles to Web Development Using Jenkins, Git

Soln: Slip 18 Q1

Q2 create a simple project And push on remote server (like github) using git. and perform some operation. And displays a chronological history of commits

Step 1: Create a Simple Project

bash

```
mkdir git_github_demo  
cd git_github_demo  
echo "Hello, GitHub!" > readme.txt
```

Step 2: Initialize Git and Make an Initial Commit

bash

```
git init  
git add readme.txt  
git commit -m "Initial commit: Add readme.txt"
```

Step 3: Create a GitHub Repository

- Go to github.com
 - Click New repository
 - Name it (e.g. git-github-demo)
 - Leave it empty (don't add README, .gitignore)
 - Click Create repository
 - Copy the HTTPS URL (like <https://github.com/yourusername/git-github-demo.git>)
-

Step 4: Connect Local Repo to GitHub and Push

bash

```
git remote add origin https://github.com/yourusername/git-github-  
demo.git  
git branch -M main  
git push -u origin main
```

Step 5: Perform an Operation (Edit the File and Commit Again)

bash

```
echo "This project demonstrates Git basics." >> readme.txt  
git add readme.txt  
git commit -m "Add project description to readme.txt"  
git push
```

Step 6: Show Chronological Commit History

```
bash
git log --oneline --reverse
Sample output:
text
23d55de Initial commit: Add readme.txt
87f4113 Add project description to readme.txt
```

Slip 24

Q1 Create CI using Webhook and deploy a project using Jenkins Execute shell.

Platform/Tools You'll Use

- Jenkins (web interface, running locally: <http://localhost:8080>)
 - GitHub (web for repo and webhook setup)
 - Jenkins “Execute shell” build step (Jenkins job configuration)
 - Git (local, for pushing changes)
 - Web server (can be Nginx, Apache, or just a location for deployed files)
-

Step 1: Prepare Your Jenkins and GitHub Repositories

A. Create a simple project locally:

```
bash
mkdir jenkins_webhook_demo
cd jenkins_webhook_demo
echo "<h1>Hello, CI/CD!</h1>" > index.html
git init
git add index.html
git commit -m "Initial commit: Add index.html"
# On GitHub: create empty repo 'jenkins-webhook-demo', copy its HTTPS URL
git remote add origin https://github.com/yourusername/jenkins-
webhook-demo.git
git branch -M main
git push -u origin main
```

Step 2: Start Jenkins Locally

(If using jenkins.war)

```
bash
java -jar jenkins.war
Open http://localhost:8080.
```

Step 3: Create a Jenkins Job to Build/Deploy from GitHub

1. In Jenkins, click New Item → Freestyle project. Name it (e.g., WebDemoDeploy)
2. Source Code Management:
Choose Git, paste your repo URL.
3. Build Triggers:
Check “GitHub hook trigger for GITScm polling”
4. Build Steps:
Add “Execute shell” step (Linux/Mac) or “Execute Windows batch

command” (Windows):

Example:

- Linux:

```
bash
cp index.html /usr/share/nginx/html/index.html
```

- Windows:

```
text
copy index.html C:\nginx\html\index.html
```

Step 4: Configure a GitHub Webhook

1. Go to your GitHub repo → Settings → Webhooks
 2. Click Add webhook
 3. Payload URL: `http://<your-jenkins-server>:8080/github-webhook/`
(For local Jenkins, use your computer’s IP, e.g. `http://192.168.1.4:8080/github-webhook/`)
 4. Content type: application/json
 5. Events: Choose “Just the push event.”
 6. Click Add webhook
-

Step 5: Test the End-to-End Flow

1. Edit your local project:

```
bash
echo "<h1>Updated by Jenkins!</h1>" > index.html
git add index.html
git commit -m "Update index.html"
git push
```

2. Result: GitHub notifies Jenkins via the webhook → Jenkins job pulls repo, executes shell step to deploy updated file.
-

Expected Output

- Jenkins job runs, shows “Build Success.”
- Your deployed file (index.html) is now updated on your web server (Nginx/Apache or any location you set).

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

Platforms/Tools Needed

- Jenkins (run locally and access via web browser)
 - Git (locally and/or GitHub)
 - Nginx or Apache (local web server)
 - Git Bash or Terminal
-

1. Create and Push a Simple Web Project to GitHub

```
bash
mkdir web_cicd_demo
cd web_cicd_demo
echo "<h1>Hello CI/CD!</h1>" > index.html
git init
git add index.html
git commit -m "Initial commit: Add index.html"
# Create a new empty repo on GitHub and copy its URL:
git remote add origin
https://github.com/yourusername/web_cicd_demo.git
git branch -M main
git push -u origin main
(Platform: Git Bash/Terminal & GitHub Web)
```

2. Install/Start Nginx or Apache

For Nginx (Linux):

```
bash
sudo apt install nginx
sudo systemctl start nginx
```

For Windows:

Download and extract [Nginx for Windows](#), then run nginx.exe.

(Platform: Terminal/Command Prompt)

3. Start Jenkins

- Download [Jenkins WAR file](#) if not already installed.

```
bash
```

```
java -jar jenkins.war
```

- Visit <http://localhost:8080> in your browser.

(Platform: Terminal & Web Browser)

4. Create a Jenkins Job

1. In Jenkins, click New Item → Freestyle project (e.g., WebAutoDeploy)

2. Source Code Management: select Git and paste your repo URL.

3. Build Triggers:

- Enable “Poll SCM” (for demo: * * * * for once per minute)
- Or, set up a GitHub webhook to your Jenkins server

4. Build Step:

- Choose Execute shell (Linux/Mac) or Execute Windows batch command (Windows)
- Example (Linux):

```
bash  
cp index.html /usr/share/nginx/html/index.html
```

- For Windows:

```
text  
copy index.html C:\nginx\html\index.html  
(Platform: Jenkins Web UI)
```

5. Test CI/CD Flow

1. Edit your local index.html, commit and push:

```
bash  
echo "<h1>Hello CI/CD -- updated!</h1>" > index.html  
git add index.html  
git commit -m "Update index.html"  
git push
```

2. Jenkins will detect the change, auto-run the job, and deploy your new file to the server directory.
3. Open your browser:

Go to <http://localhost:8080> (if Nginx runs at 8080) or <http://localhost> to see the updated page.

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

. Create and Initialize Your Simple Project

bash

```
mkdir gitlab_demo_project  
cd gitlab_demo_project  
echo "Hello, GitLab!" > hello.txt  
git init  
git add hello.txt  
git commit -m "Initial commit: add hello.txt"
```

2. Create a Repository on GitLab (Web)

- Go to <https://gitlab.com>
 - Click “New project”
 - Name it (e.g. gitlab-demo-project)
 - Leave it empty (don’t add README or .gitignore)
 - Click Create project
 - Copy the HTTPS repo URL (e.g. https://gitlab.com/yourusername/gitlab-demo-project.git)
-

3. Connect Local Repo to GitLab and Push

bash

```
git remote add origin https://gitlab.com/yourusername/gitlab-demo-project.git  
git branch -M main  
git push -u origin main
```

4. Create a New Branch

bash

```
git checkout -b feature-branch  
echo "Added this line in feature branch." >> hello.txt  
git add hello.txt  
git commit -m "feature-branch: add a second line to hello.txt"  
git push -u origin feature-branch
```

5. Merge Branch into Main (Using GitLab Web Interface – Recommended for Exams)

- Go to your repo on GitLab.
- You’ll see a “Create merge request” (MR) for feature-branch. Click it.

- Review, then click "Merge".

6. Pull Merged Changes to Your Local Main Branch

```
bash  
git checkout main  
git pull
```

7. Display Chronological Commit History

```
bash  
git log --oneline --reverse
```

Sample output:

```
text  
abc1234 Initial commit: add hello.txt  
def5678 feature-branch: add a second line to hello.txt  
123abcd Merge branch 'feature-branch' into 'main'
```

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

Execute shell.

Platforms/Tools Used

- Jenkins (web interface, local or server)
- GitHub (web portal, for repo and webhook setup)
- Git Bash/Terminal (for project creation and git commands)
- Web server folder (e.g., Nginx or Apache, or any local deploy folder)

Step-by-Step Solution

1. Prepare a Simple Project and Push to GitHub

(Platform: Git Bash/Terminal & GitHub Web)

```
bash  
mkdir ci_webhook_demo  
cd ci_webhook_demo  
echo "<h1>Hello, CI/CD via Jenkins!</h1>" > index.html  
git init  
git add index.html  
git commit -m "Initial commit: add index.html"  
# Create an empty repo on GitHub, copy its URL  
git remote add origin https://github.com/yourusername/ci-webhook-demo.git  
git branch -M main  
git push -u origin main
```

2. Start Jenkins Locally (Platform: Terminal & Web Browser)

```
bash  
java -jar jenkins.war
```

- Go to <http://localhost:8080> in your browser.
-

3. Create a Jenkins Job to Build/Deploy from GitHub (Platform: Jenkins Web Interface)

1. Click New Item, then create a Freestyle project
2. Source Code Management: select Git, paste your repo URL.
3. Build Triggers:
Check "GitHub hook trigger for GITScm polling"
4. Build Steps → Execute shell:

Example for Linux:

```
bash  
cp index.html /usr/share/nginx/html/index.html
```

Example for Windows:

```
text  
copy index.html C:\nginx\html\index.html
```

4. Configure a GitHub Webhook (Platform: GitHub Web Interface)

1. In your GitHub repo, go to Settings → Webhooks
 2. Click Add webhook
 3. Payload URL: http://YOUR_JENKINS_SERVER:8080/github-webhook/ (e.g., http://localhost:8080/github-webhook/ if Jenkins is local)
 4. Content type: application/json
 5. Events: Select "Just the push event"
 6. Click Add webhook
-

5. Test Your CI/CD Workflow

Edit your index.html, commit, and push:

```
bash  
echo "<h1>Updated by Jenkins!</h1>" > index.html  
git add index.html  
git commit -m "Update index.html"  
git push
```

- The webhook triggers Jenkins, which builds and deploys your updated file.