**EXP1. Demonstrate use of git commands to push and pull 3 html files from Github account.**

1)

GO to <https://git-scm.com/downloads/win> , download for windows , click on click here

Also add on the desktop

Use git and optional unix tools, next

2)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Home Page</title>

</head>

<body>

<h1>Welcome to My Website</h1>

<p>This is the homepage.</p>

<a href="file2.html">Go to About Page</a>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>About</title>

</head>

<body>

<h1>About Us</h1>

<p>We are learning how to use Git and GitHub.</p>

<a href="file3.html">Go to Contact Page</a>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Contact</title>

</head>

<body>

<h1>Contact Us</h1>

<p>Email: info@example.com</p>

<a href="file1.html">Back to Home</a>

</body>

</html>

3)

Make a repo public

4)

Use it command to run files

And gitbash on folder

5)

**Git init**

**Git status (red)**

**Git add .**

**Git status (green)**

**git config --global user.email “**[**vaibhavschavan26@gmail.com**](mailto:vaibhavschavan26@gmail.com)**”**

**git config –global user.name “Vaichavan”**

**git commit -m "first commit"**

**git branch -M main**

**git remote add origin https://github.com/Vaichavan/Expt1.git**

**git push -u origin main**

update file check **git status**

**Git add .**

**git commit -m "file updated"**

now make changes in h2 and then pull it

**git pull origin main**

**EXP2. Demonstrate Building java program (addition of 2 numbers) through Jenkins.**

Create Addition.java

**public class Addition {**

**public static void main(String[] args) {**

**int a = 10, b = 20;**

**int sum = a + b;**

**System.out.println("Sum: " + sum);**

**}**

**}**

**Manually add it OR**

**Git init**

**Git status (red)**

**Git add .**

**Git status (green)**

**git config --global user.email “**[**vaibhavschavan26@gmail.com**](mailto:vaibhavschavan26@gmail.com)**”**

**git config –global user.name “Vaichavan”**

**git commit -m "first commit"**

**git branch -M main**

**git remote add origin https://github.com/Vaichavan/Expt1.git**

**git push -u origin main**

3)

Plugin like git plugin and pipline

4) Create freestyle project , source git and no credentials as public= <https://github.com/Vaichavan/Expt1.git> , main

5) Build Steps choose **Execute Windows batch command** (Windows)

javac Addition.java

java Addition

6) save and build

**EXP3. Demonstrate Continuous Integration process in Jenkins. Build a java program every 2 minutes (addition of two numbers) residing in Github repository.**

**1) Same Above steps**

2). Scroll up until you find Triggers 12.In Triggers select Build Periodically and put the code

**H/2 \* \* \* \***

(This will build the project every 2 min)

13.Save and Apply and Build Now

**EXP4. Demonstrate CI/CD in Jenkins. Consider 3 html files in Github repository and write a scripted pipeline to deploy on Xampp server.**

1. Download xampp, keep apache start
2. Already have file in github with 3 html
3. Plugin

[SSH server](https://plugins.jenkins.io/sshd), [SSH Build Agents](https://plugins.jenkins.io/ssh-slaves) , [ShiningPanda Plugin](https://plugins.jenkins.io/shiningpanda), [Python Plugin](https://plugins.jenkins.io/python), [Pyenv Pipeline Plugin](https://plugins.jenkins.io/pyenv-pipeline) , [Pipeline: GitHub Groovy Libraries](https://plugins.jenkins.io/pipeline-github-lib), [Pipeline Graph View](https://plugins.jenkins.io/pipeline-graph-view), , [PipelineVersion](https://plugins.jenkins.io/workflow-aggregator), [Maven Integration](https://plugins.jenkins.io/maven-plugin), [Deploy to container Plugin](https://plugins.jenkins.io/deploy),

[Command Agent Launcher](https://plugins.jenkins.io/command-launcher)

[Build Pipeline Plugin](https://plugins.jenkins.io/build-pipeline-plugin)

[Ansible plugin](https://plugins.jenkins.io/ansible)

[Ansible Tower Plugin](https://plugins.jenkins.io/ansible-tower)

[AntVersion](https://plugins.jenkins.io/ant)

git, and pipeline

1. Choose pipeline
2. Scroll to **Pipeline** section
3.  Select **Definition**: *Pipeline script*
4.  Paste the following:

**node {**

**stage('Clone Repository') {**

**git branch: 'main', url: 'https://github.com/Vaichavan/Expt1.git'**

**}**

**stage('Build') {**

**echo 'No build required for static HTML files'**

**}**

**stage('Deploy to XAMPP') {**

**def xamppPath = 'C:/xampp/htdocs'**

**echo "Deploying HTML files to XAMPP..."**

**bat """**

**xcopy /Y /Q \*.html "${xamppPath}\\"**

**"""**

**}**

**}**

8) Go to your browser and open:

http://localhost/h1.html

**EXP5. Demonstrate CI/CD in Jenkins. Consider 3 html files in Github repository and write a declarative pipeline to deploy on Xampp server.**

Same as above all other steps are same

**pipeline {**

**agent any**

**environment {**

**DEPLOY\_DIR = 'C:/xampp/htdocs'**

**}**

**stages {**

**stage('Clone Repository') {**

**steps {**

**git branch: 'main', url: 'https://github.com/Vaichavan/Expt1.git'**

**}**

**}**

**stage('Build') {**

**steps {**

**echo 'No build required for HTML files'**

**}**

**}**

**stage('Deploy to XAMPP') {**

**steps {**

**echo 'Deploying files to XAMPP server...'**

**bat "xcopy /Y /Q \*.html \"${DEPLOY\_DIR}\\\""**

**}**

**}**

**}**

**post {**

**success {**

**echo 'Deployment successful!'**

**}**

**failure {**

**echo 'Deployment failed.'**

**}**

**}**

**}**

**EXP6. Write a Selenium script to perform automated testing.**

1. Jenkins freework , add git link then, main, BUILD STEPS = choose custom python build [Python Plugin](https://plugins.jenkins.io/python), [Pyenv Pipeline Plugin](https://plugins.jenkins.io/pyenv-pipeline)
2. Then where python in cmd and copy paste in loc
3. copy paste it in it

python -m venv env

call ./env/Scripts/activate.bat

pip install -r requirements.txt

sbase install chromedriver latest

python exp6.py

1. run

5) RUN IT ON VSCODE WITH selenium installed

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

import time

print("Sample test case started")

# Configure Chrome options

chrome\_options = Options()

chrome\_options.add\_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36")

chrome\_options.add\_argument("--disable-blink-features=AutomationControlled")  # Prevent detection

# Initialize WebDriver

driver = webdriver.Chrome(options=chrome\_options)

driver.maximize\_window()

driver.get("https://www.google.com/")

# Search for 'simplilearn'

search\_box = driver.find\_element(By.NAME, "q")

search\_box.send\_keys("simplilearn")

time.sleep(2)

search\_box.send\_keys(Keys.ENTER)

time.sleep(7)

# Get search result snippets

results = driver.find\_elements(By.CSS\_SELECTOR, ".tF2Cxc .VwiC3b")

# Print the first few snippets

for i, result in enumerate(results[:5]):  # Adjust the number as needed

    print(f"Result {i+1}: {result.text}\n")

# Close the browser

driver.quit()

print("Sample test case successfully completed")

**EXP7. Create an image of php project and push on Dockerhub repository.**

index.php file

<!DOCTYPE html>

<html>

  <head>

    <title>PHP in Docker</title>

  </head>

  <body>

    <?php echo "Hello world from PHP container"; ?>

  </body>

</html>

dockerfile

FROM php

COPY ./ ./

EXPOSE 3000

CMD ["php", "-S", "0.0.0.0:3000"]

Commands in same folder open terminal

1. **install docker wsl terminal press any then try again**
2. **first login in** [**vaibhavschavan26@gmail.com**](mailto:vaibhavschavan26@gmail.com)

**docker build -t php-app .**

**docker run -p 3000:3000 php-docker-app**

**docker images**

**http://localhost:3000**

**EXP8. Demonstrate use of docker commands to pull ubuntu official image, create a file and push updated image on Dockerhub.**

docker pull ubuntu

docker run -it ubuntu

echo "Hello from Docker container!" > /myfile.txt

exit

docker ps -a

**docker commit <container\_id> yourdockerhubusername/ubuntu-custom**

docker login

**docker push yourdockerhubusername/ubuntu-custom**

**Experiment 1: Demonstrate Use of Git Commands to Push and Pull 3 HTML Files from GitHub Account**

**✅ Aim:**

To explore and understand Git commands used to initialize a repository, track files, and perform push and pull operations between a local repository and a remote GitHub repository using 3 HTML files.

**📚 Theory:**

Git is a **distributed version control system** that allows multiple developers to work on a codebase simultaneously. Git tracks every modification made to the files, enabling easy rollbacks, branching, and collaboration. GitHub is a cloud-based hosting service for Git repositories that provides additional features such as collaboration tools, issue tracking, and code review.

By using push and pull, developers can sync local changes with the central GitHub repository and keep all team members updated.

**📘 Key Definitions:**

* **Repository (Repo):** A directory that contains all project files and version history.
* **Staging Area:** Area where changes are reviewed before committing.
* **Commit:** A snapshot of changes with a message.
* **Push:** Uploading commits from the local repo to GitHub.
* **Pull:** Fetching and merging changes from GitHub to local repo.

**💡 Real-World Example:**

A team of web developers is building a static site. Each member works on separate HTML pages and uses Git and GitHub to merge changes without overwriting each other's work.

**🔧 Steps:**

1. **Install Git** (if not already installed).

bash

CopyEdit

sudo apt install git

1. **Configure Git for your identity**:

bash

CopyEdit

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

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

1. **Create 3 HTML files** in a folder:
   * index.html, about.html, contact.html
2. **Initialize a Git repository**:

bash

CopyEdit

git init

1. **Add files to staging area**:

bash

CopyEdit

git add .

1. **Commit changes**:

bash

CopyEdit

git commit -m "Added 3 HTML files"

1. **Create a repository on GitHub** (e.g., html-demo).
2. **Link local repo to GitHub**:

bash

CopyEdit

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

1. **Push changes to GitHub**:

bash

CopyEdit

git push -u origin main

1. **Clone on another machine or folder**:

bash

CopyEdit

git clone https://github.com/username/html-demo.git

1. **Make a change (e.g., add a paragraph), commit, and push again**:

bash

CopyEdit

git add .

git commit -m "Updated index.html"

git push

1. **Pull latest changes elsewhere**:

bash

CopyEdit

git pull origin main

**🧠 Learning Outcomes:**

* Understand the concept of version control.
* Perform Git operations (init, add, commit, push, pull).
* Use GitHub to collaborate and synchronize code.

**📝 Conclusion:**

Git and GitHub provide powerful tools for managing and sharing code. Mastery of basic Git commands is foundational for collaborative development and CI/CD pipelines.

**🔹 Experiment 2: Demonstrate Building a Java Program (Addition of 2 Numbers) through Jenkins**

**✅ Aim:**

To configure Jenkins to automatically compile and execute a simple Java program that adds two numbers, thereby demonstrating the automation of the build process.

**📚 Theory:**

**Jenkins** is an open-source automation server used to implement Continuous Integration. It supports automation of code building, testing, and deployment. By creating Jenkins jobs, developers can automate repetitive tasks and ensure consistent builds.

In this experiment, Jenkins is used to fetch code (Java program) and build it using a shell script.

**📘 Key Concepts:**

* **Freestyle Project:** A basic job type in Jenkins for simple automation tasks.
* **Build Step:** A section where commands/scripts are added to execute during the job.
* **Console Output:** The output log showing job execution steps.

**💡 Java Code Example – Add.java:**

java

CopyEdit

public class Add {

public static void main(String[] args) {

int a = 20;

int b = 30;

int sum = a + b;

System.out.println("Sum = " + sum);

}

}

**🔧 Steps:**

1. **Install Jenkins and Java**:

bash

CopyEdit

sudo apt update

sudo apt install openjdk-11-jdk

sudo apt install jenkins

1. **Start Jenkins**:

bash

CopyEdit

sudo systemctl start jenkins

1. **Access Jenkins**: Go to http://localhost:8080
2. **Unlock Jenkins** using the admin password from:

bash

CopyEdit

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

1. **Install Suggested Plugins** during setup.
2. **Create a Freestyle Project**:
   * Name: Java-Addition-Build
3. **Configure Git repository** (host your Add.java on GitHub):
   * Add GitHub repo under Source Code Management.
4. **Add Build Step**:
   * Choose "Execute Shell"
   * Enter:

bash

CopyEdit

javac Add.java

java Add

1. **Save and build the project.**
2. **Check Console Output** for Sum = 50.

**🧠 Learning Outcomes:**

* Understand Jenkins job configuration.
* Automate Java compilation using Jenkins.
* Read and interpret Jenkins job logs.

**📝 Conclusion:**

Jenkins streamlines the software build process, reducing manual effort and increasing reliability in project compilation.

**🔹 Experiment 3: Demonstrate Continuous Integration Process in Jenkins – Build a Java Program Every 2 Minutes from GitHub**

**✅ Aim:**

To demonstrate Continuous Integration (CI) using Jenkins by polling a GitHub repository every 2 minutes and automatically building a Java program.

**📚 Theory:**

**Continuous Integration (CI)** ensures that changes pushed to a repository are automatically tested and built. Jenkins achieves CI through jobs that monitor repositories and trigger builds based on defined intervals or events.

By polling GitHub every 2 minutes, Jenkins can compile the Java code and alert developers if any build fails, thereby maintaining code integrity.

**📘 Key Concepts:**

* **Poll SCM:** A Jenkins trigger to check source control (GitHub) at regular intervals.
* **Build Trigger:** A mechanism to automate the build without manual intervention.
* **CI Pipeline:** A sequence that includes fetch → build → test.

**💡 Example:**

Same Java program from Exp 2 is hosted on GitHub. Jenkins will fetch and build it every 2 minutes.

**🔧 Steps:**

1. **Use previously created Java project or create a new GitHub repo** with Add.java.
2. **Create Freestyle Project in Jenkins**:
   * Name: CI-Java-Build
3. **Configure GitHub Repo** under Source Code Management.
4. **Set Build Triggers**:
   * Tick Poll SCM and enter:

CopyEdit

H/2 \* \* \* \*

* + This means: Run job every 2 minutes.

1. **Build Step (Shell)**:

bash

CopyEdit

javac Add.java

java Add

1. **Save the job.**
2. **Push a new change to GitHub (e.g., change values).**
3. **Wait for Jenkins to auto-detect changes and build.**
4. **View build logs and success/failure status.**

**🧠 Learning Outcomes:**

* Understand the concept of CI.
* Configure Jenkins for periodic polling.
* Automate build verification for every repository change.

**📝 Conclusion:**

Automated and scheduled builds through Jenkins ensure that code is always up-to-date and working. CI reduces manual testing, improves collaboration, and detects issues early.

Would you like the detailed write-up for Experiments 4 to 8 as well?

4o

**You said:**

Would you like the detailed write-up for Experiments 4 to 8 as well?

**ChatGPT said:**

Great! Here's the **detailed write-up for Experiments 4 to 8** from your DevOps Lab Manual, with full **Aim, Theory, Definitions, Examples, Steps, Learning Outcomes, and Conclusions**:

**🔹 Experiment 4: Demonstrate CI/CD in Jenkins — Scripted Pipeline to Deploy 3 HTML Files on XAMPP Server**

**✅ Aim:**

To implement a **scripted Jenkins pipeline** that automates fetching 3 HTML files from a GitHub repository and deploys them to the XAMPP server's htdocs directory.

**📚 Theory:**

A **scripted pipeline** in Jenkins uses Groovy-based code to define custom stages and logic for CI/CD. This gives developers more control and flexibility over the build and deployment process. Combined with **XAMPP**, a local web server stack (Apache, MySQL, PHP, Perl), developers can test HTML deployments locally.

**📘 Definitions:**

* **CI/CD:** Continuous Integration/Continuous Deployment.
* **Scripted Pipeline:** Jenkins pipeline defined using Groovy script in Jenkinsfile.
* **XAMPP Server:** Cross-platform Apache server package used to host web files locally.

**💡 Example Scenario:**

Deploying static website files from GitHub to a local XAMPP server for testing before moving to production.

**🔧 Steps:**

1. **Ensure XAMPP is installed** and running. Note the path:
   * Typically C:\xampp\htdocs\ (Windows)
   * Or /opt/lampp/htdocs/ (Linux)
2. **Create a GitHub repo** containing 3 HTML files.
3. **Create a Jenkinsfile** in that repo:

groovy

CopyEdit

node {

stage('Clone Repo') {

git 'https://github.com/username/html-demo.git'

}

stage('Deploy to XAMPP') {

sh 'cp -r \*.html /opt/lampp/htdocs/'

}

}

1. **Create a new Pipeline job** in Jenkins.
2. **Choose "Pipeline script from SCM"**, connect your GitHub repo.
3. **Save and run the pipeline**.
4. **Verify deployment** by opening http://localhost/index.html.

**🧠 Learning Outcome:**

Students understand how to create scripted pipelines and perform automatic deployments to a local web server.

**📝 Conclusion:**

Scripted Jenkins pipelines enable precise control over CI/CD stages, and integration with XAMPP helps simulate real-world web deployment environments.

**🔹 Experiment 5: Demonstrate CI/CD in Jenkins — Declarative Pipeline to Deploy 3 HTML Files on XAMPP Server**

**✅ Aim:**

To create a **declarative Jenkins pipeline** that automates the deployment of HTML files to the XAMPP server from a GitHub repository.

**📚 Theory:**

A **declarative pipeline** uses a predefined syntax to describe the CI/CD workflow more simply and readably than scripted pipelines. It’s the recommended format for most use cases due to its structure and ease of use.

**📘 Definitions:**

* **Declarative Pipeline:** A Jenkins pipeline using a simplified syntax block (pipeline { ... }).
* **Deploy Stage:** The part of the pipeline where files are copied to the server.
* **Post Block:** Used to define steps to execute after the pipeline completes.

**💡 Use Case:**

A front-end team wants to deploy updated HTML files to a local server every time a change is pushed to GitHub.

**🔧 Steps:**

1. **Set up XAMPP** as in Exp 4.
2. **Create a GitHub repo** with HTML files and a Jenkinsfile:

groovy

CopyEdit

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git 'https://github.com/username/html-demo.git'

}

}

stage('Deploy to XAMPP') {

steps {

sh 'cp -r \*.html /opt/lampp/htdocs/'

}

}

}

post {

success {

echo 'Deployment completed successfully!'

}

failure {

echo 'Deployment failed.'

}

}

}

1. **Create a Pipeline job in Jenkins**, select "Pipeline script from SCM".
2. **Configure GitHub repository**, and save.
3. **Build the pipeline**.
4. **Access the deployed site** via http://localhost/.

**🧠 Learning Outcome:**

Students will learn declarative pipeline syntax and how to implement cleaner, more maintainable CI/CD pipelines.

**📝 Conclusion:**

Declarative pipelines simplify CI/CD automation while maintaining reliability. Integrating with XAMPP helps test deployments in real-time.

**🔹 Experiment 6: Write a Selenium Script to Perform Automated Testing**

**✅ Aim:**

To write a **Selenium WebDriver script** that automates browser-based testing of a website.

**📚 Theory:**

**Selenium** is a tool used to automate web browser actions like clicking buttons, entering text, and verifying page content. It supports various programming languages like Java, Python, and JavaScript.

**📘 Definitions:**

* **WebDriver:** API used to automate browser actions.
* **Locator:** A method to identify web elements (e.g., by ID, name, XPath).
* **Automation Testing:** Testing without manual interaction.

**💡 Example Scenario:**

Automating login form validation on a website.

**🔧 Steps (Python Example):**

1. **Install Selenium and WebDriver**:

bash

CopyEdit

pip install selenium

1. **Download ChromeDriver** and set path.
2. **Write a test script:**

python

CopyEdit

from selenium import webdriver

from selenium.webdriver.common.by import By

driver = webdriver.Chrome()

driver.get("http://example.com/login")

driver.find\_element(By.ID, "username").send\_keys("admin")

driver.find\_element(By.ID, "password").send\_keys("admin123")

driver.find\_element(By.ID, "submit").click()

assert "Dashboard" in driver.title

driver.quit()

1. **Run the test** and verify output.

**🧠 Learning Outcome:**

Students will understand how to write automated test scripts, interact with web elements, and validate web pages.

**📝 Conclusion:**

Selenium makes it possible to test websites quickly and reliably, reducing the need for manual testing and improving software quality.

**🔹 Experiment 7: Create an Image of PHP Project and Push to DockerHub**

**✅ Aim:**

To containerize a simple PHP project using Docker, create a Docker image, and upload it to DockerHub.

**📚 Theory:**

Docker allows packaging applications with their dependencies in a container. A **Dockerfile** defines how the image is built. DockerHub is a registry for storing and sharing Docker images.

**📘 Definitions:**

* **Dockerfile:** Script to build a Docker image.
* **Image:** Blueprint for containers.
* **Container:** Running instance of an image.

**💡 Example Project:**

A PHP project with index.php that displays “Hello World”.

**🔧 Steps:**

1. **Create index.php:**

php

CopyEdit

<?php echo "Hello World!"; ?>

1. **Create Dockerfile:**

dockerfile

CopyEdit

FROM php:7.4-apache

COPY . /var/www/html/

1. **Build the image**:

bash

CopyEdit

docker build -t username/php-demo .

1. **Test the container locally**:

bash

CopyEdit

docker run -d -p 8080:80 username/php-demo

1. **Login to DockerHub**:

bash

CopyEdit

docker login

1. **Push image to DockerHub**:

bash

CopyEdit

docker push username/php-demo

**🧠 Learning Outcome:**

Students will learn how to containerize applications, build Docker images, and push them to DockerHub for sharing.

**📝 Conclusion:**

Docker makes application deployment portable and consistent. Pushing to DockerHub allows global access and reuse of container images.

**🔹 Experiment 8: Demonstrate Use of Docker Commands — Pull Ubuntu Image, Create a File, Push Updated Image**

**✅ Aim:**

To use Docker commands to pull the official Ubuntu image, modify it by adding a file, and push the updated image to DockerHub.

**📚 Theory:**

Docker enables fast provisioning of environments. Modifying a base image and pushing it as a new version ensures repeatability and standardization in DevOps pipelines.

**📘 Definitions:**

* **Base Image:** A starting point for new Docker images.
* **Layers:** Each instruction in a Dockerfile creates a layer.
* **Tag:** Label used to identify Docker images.

**🔧 Steps:**

1. **Pull Ubuntu image**:

bash

CopyEdit

docker pull ubuntu

1. **Create Dockerfile**:

dockerfile

CopyEdit

FROM ubuntu

RUN apt update && apt install -y nano

RUN echo "DevOps Test File" > /myfile.txt

1. **Build updated image**:

bash

CopyEdit

docker build -t username/ubuntu-updated .

1. **Run container and verify file**:

bash

CopyEdit

docker run -it username/ubuntu-updated bash

cat /myfile.txt

1. **Push to DockerHub**:

bash

CopyEdit

docker push username/ubuntu-updated

**🧠 Learning Outcome:**

Students will understand image modification, tagging, and version management using Docker.

**📝 Conclusion:**

Using Docker, system images can be customized and reused across teams. Pushing to DockerHub promotes version control and distribution.

Would you like these compiled into a formatted PDF or Word document?

4o

**You said:**

separatedly give indepth theory for 4-8

**ChatGPT said:**

Updated saved memory

Here are the **in-depth theory explanations** for **Experiments 4 to 8** in your DevOps lab manual:

**🔹 Experiment 4: CI/CD in Jenkins — Scripted Pipeline to Deploy HTML Files on XAMPP**

**🧠 In-Depth Theory:**

A **scripted pipeline** in Jenkins is defined using Groovy-based scripting, offering maximum control over logic, flow, and execution. Unlike declarative pipelines, scripted pipelines are more flexible and can include complex logic such as conditional statements and loops.

In CI/CD, **Continuous Integration (CI)** ensures code changes are automatically merged, tested, and validated. **Continuous Deployment (CD)** extends this to automatically push updates to a live environment. Combining Jenkins with a **scripted pipeline** allows developers to design custom deployment workflows.

**XAMPP** is a lightweight web server stack used for hosting applications locally. It includes Apache (HTTP server), MySQL (database), PHP, and Perl. By deploying HTML files to the XAMPP htdocs directory, the files become accessible via localhost, simulating a real server environment.

**Key Concepts:**

* node {} block defines where the pipeline runs.
* Jenkins uses agents (nodes) to execute jobs.
* Shell commands inside sh blocks allow direct interaction with the underlying OS.
* Scripted pipelines are ideal for teams that need fine-grained control over the build/deploy process.

**🔹 Experiment 5: CI/CD in Jenkins — Declarative Pipeline to Deploy HTML Files on XAMPP**

**🧠 In-Depth Theory:**

A **declarative pipeline** is a simplified syntax for defining Jenkins CI/CD workflows using the pipeline {} structure. It is more readable and maintainable than scripted pipelines and is recommended for most use cases due to its structure, validation, and error handling features.

The declarative approach divides a pipeline into stages (e.g., Checkout, Build, Deploy) using a clear and strict format. This makes pipelines easier to version, troubleshoot, and maintain.

Using Jenkins' **"Pipeline from SCM"** feature, the Jenkinsfile is stored in the source code repository itself (usually GitHub), ensuring synchronization between code and automation logic.

Deployment to **XAMPP** is the same in principle as in Experiment 4 — by copying files to the htdocs folder, Jenkins enables local testing and deployment of static web files.

**Key Features of Declarative Pipelines:**

* Built-in validation and error handling.
* post block for defining steps after execution (success, failure).
* Easy integration with shared libraries, credentials, and environment variables.
* Better support in Jenkins UI (visualization and editor plugins).

**🔹 Experiment 6: Selenium Automation Testing**

**🧠 In-Depth Theory:**

**Selenium** is an open-source framework for automating browser interactions. It allows developers and testers to simulate user actions like clicking buttons, filling out forms, navigating pages, and verifying content. Selenium supports multiple languages: Python, Java, C#, JavaScript, and Ruby.

Selenium WebDriver is the most popular component. It interacts directly with the browser using native automation APIs. Unlike Selenium IDE (record-playback), WebDriver is fully programmable, making it ideal for integration into automated test suites and CI/CD pipelines.

Automated testing reduces human error and speeds up the development cycle by running tests in parallel and on different browsers. It also integrates well with Jenkins, enabling automated test runs after each code change.

**Common Locators:**

* By.id
* By.name
* By.className
* By.xpath
* By.cssSelector

**Test Types:**

* Smoke Testing: Basic functionality check.
* Regression Testing: Ensures new changes don’t break existing features.
* End-to-End Testing: Simulates complete user workflows.

**🔹 Experiment 7: Dockerize PHP Project and Push to DockerHub**

**🧠 In-Depth Theory:**

**Docker** is a containerization platform that packages an application along with its environment into a single container image. This makes it portable, consistent, and easy to run on any system with Docker installed.

A **Dockerfile** is a blueprint for creating Docker images. It includes:

* Base image (FROM)
* Application files (COPY)
* Instructions for building the image (RUN, CMD)

By building an image from a PHP project and pushing it to **DockerHub**, developers make it available publicly (or privately) for deployment on any host. This improves collaboration, version control, and deployment speed.

**DockerHub** is the central registry where images can be stored, tagged, and shared. Once pushed, others can pull the image using a single command.

**Benefits:**

* Eliminates "it works on my machine" issues.
* Supports microservices architecture.
* Simplifies deployment and rollback processes.

**🔹 Experiment 8: Docker Commands — Modify Ubuntu Image and Push to DockerHub**

**🧠 In-Depth Theory:**

In this experiment, students explore Docker’s ability to modify existing images and create new ones. Starting from a base **Ubuntu image**, they add custom files or packages and then commit those changes as a new image.

This is achieved by writing a **Dockerfile**, which defines the steps for creating a new image based on another. For example, adding a file or installing software using RUN instructions.

Once the modified image is tested, it is tagged and pushed to **DockerHub**, making it portable and sharable.

This mirrors real-world practices in DevOps where base images (e.g., Ubuntu, Alpine, CentOS) are customized with application code or system settings and used in automated deployments.

**Important Docker Commands:**

* docker pull: Download base image.
* docker build: Create new image.
* docker tag: Assign repository and version.
* docker push: Upload image to DockerHub.