EXPERIMENT 1: Git Push/Pull HTML Files (Windows)

What You'll Do

- Push 3 HTML files from your Windows PC to GitHub.
- Pull them back to confirm success.

Step 1: Install Git

1. Download Git for Windows.
2. Run the installer. Keep all default settings (check "Git Bash" and "Windows Explorer Integration").

Step 2: Create HTML Files

1. Open **File Explorer**, go to `Desktop` , and create a folder named `devops-demo` .
2. Inside this folder, create 3 files:
    - `index.html`
    - `about.html`
    - `contact.html`
3. Right-click each file > **Open with Notepad** and add basic HTML code:
   xml
   ```
   <!-- Example for index.html --> <!DOCTYPE  html> <html> <body>   <h1>Home Page</h1> </body> </html>
   ```

Step 3: Push to GitHub

1. Open **Git Bash** (right-click in the folder > **Git Bash Here**).
2. Run these commands:
   bash
   ```
   # Initialize Git git init   # Link to GitHub (replace URL with your repo) git remote add origin https://github.com/your-username/your-repo.git
   ```

Step 4: Validate by Pulling

1. Delete the local `devops-demo` folder (to simulate a fresh pull).
2. Open **Git Bash** on the Desktop and run:
   bash
   ```
   git clone https://github.com/your-username/your-repo.git
   ```
3. Check if the 3 HTML files reappear in the cloned folder.

Troubleshooting

- **Permission Denied**: Run Git Bash as **Administrator**.
- **Authentication Failed**:
    - Use a GitHub Personal Access Token instead of your password.
- **Wrong Remote URL**: Fix with:
  bash
  ```
  git remote set-url origin https://github.com/your-username/your-repo.git
  ```

EXPERIMENT 2: Build a Java Program (Addition of 2 Numbers) via Jenkins

Objective: Automate compilation of a Java program using Jenkins on Windows.

Tools Required

1. Java JDK 11+ (Download)
2. Jenkins for Windows (Download)
3. GitHub account (to host your Java code)

Step-by-Step Guide

1. Install Java JDK

1. Download the **Windows x64 MSI Installer** from Adoptium.
2. Run the installer.
3. Set `JAVA_HOME` environment variable:
   - Press `Win + S` > Search for **Edit environment variables**.
   - Under **System variables**, add:
     - Variable name: `JAVA_HOME`
     - Variable value: `C:\Program Files\Eclipse Adoptium\jdk-11.0.xx.x-hotspot` (replace with your JDK path)
   - Add Java to `Path` :
     - Edit **Path** variable > Add `%JAVA_HOME%\bin` .

2. Install Jenkins

1. Download the **Windows installer** from jenkins.io.
2. Run the installer. Jenkins will start automatically on `http://localhost:8080` .
3. Unlock Jenkins:
   - Retrieve the initial admin password from `C:\ProgramData\Jenkins\.jenkins\secrets\initialAdminPassword` .
4. Install suggested plugins.

3. Prepare the Java Program

1. Create a file `AddNumbers.java` on your desktop:
   java
   ```
   public  class  AddNumbers  {   public  static  void  main(String[] args)  { int a =  10, b =  20; System.out.println("Sum: "  +  (a + b)); } }
   ```
2. Push this file to a GitHub repository (see Experiment 1 if needed).

4. Configure Jenkins Job

1. In Jenkins, click **New Item** > **Freestyle project** > Name it `Java-Build-Demo` .
2. Under **Source Code Management**:
   - Select **Git** > Enter your repository URL (e.g., `https://github.com/your-username/java-demo.git`).
3. Under **Build Steps**:
   - Click **Add build step** > **Execute Windows batch command**.
   - Enter:
     text
     ```
     javac AddNumbers.java java AddNumbers
     ```
4. Save and click **Build Now**.

5. Validate the Output

1. Check the **Console Output** of the build job.
2. You should see:
   text
   ```
   Sum: 30
   ```

Troubleshooting

- **‘javac’ not recognized**: Ensure `JAVA_HOME` and `Path` are configured correctly.
- **Build fails**: Check for typos in the Java code or GitHub URL.
- **Permission issues**: Run Jenkins as Administrator (right-click Jenkins shortcut > **Run as administrator**).

Let’s tackle **Experiment 3**! 🕒

EXPERIMENT 3: Continuous Integration in Jenkins (Build Every 2 Minutes)

Objective: Automatically build a Java program every 2 minutes from a GitHub repository.

Tools Required

- Jenkins (already installed from Experiment 2)
- Java JDK (configured in Experiment 2)
- GitHub repository with your Java code

Step-by-Step Guide

1. Update Your GitHub Repository

1. Ensure your `AddNumbers.java` file is in a GitHub repo (created in Experiment 2).
2. Add a `pom.xml` (Maven) file to automate builds. Create it with:
   xml
   ```
   <project>   <modelVersion>4.0.0</modelVersion> <groupId>com.example</groupId> <artifactId>add-numbers</artifactId> <version>1.0</version> </proj
   ```

2. Configure Jenkins Polling

1. Open your existing Jenkins job (`Java-Build-Demo` from Experiment 2).
2. Under **Build Triggers**, check **Poll SCM**.
3. In the schedule box, enter:
   text
   ```
   H/2 * * * *
   ```
   (This cron syntax triggers a poll every 2 minutes).

3. Modify the Build Step

1. Under **Build**, replace the existing batch command with:
   text
   ```
   mvn clean compile
   ```
   (This uses Maven to compile the Java program).

4. Test Automatic Builds

1. Make a small change to `AddNumbers.java` (e.g., modify the numbers to `a = 15, b = 25`).
2. Commit and push the change to GitHub:
   bash
   ```
   git  add AddNumbers.java git commit -m "Trigger CI build" git push origin main
   ```
3. Wait up to 2 minutes. Jenkins will detect the change and auto-trigger a build.

5. Validate Results

1. Check the **Build History** in Jenkins for new builds.
2. View the **Console Output** of the latest build. You should see:
   text
   ```
   [INFO] BUILD SUCCESS
   ```

Troubleshooting

- **Build Not Triggered**:
  - Verify the cron syntax in **Poll SCM**.
  - Ensure Jenkins has read access to your GitHub repo.
- **Maven Not Found**: Install the Maven plugin in Jenkins.
- **No Changes Detected**: Add a dummy file to your repo to force a change.

4. CI/CD Scripted Pipeline for XAMPP Deployment

Objective: Automatically deploy 3 HTML files from GitHub to XAMPP server using Jenkins.

Tools Required

- Jenkins (Windows)
- XAMPP Server (`C:\xampp\htdocs`)
- GitHub repository with 3 HTML files

Jenkinsfile (Scripted Pipeline)

```
node {
    stage('Checkout') {
        // Clone GitHub repo
        git 'https://github.com/your-username/html-repo.git'
    }

    stage('Deploy') {
        bat """
            xcopy /Y /E "${WORKSPACE}\\*.html" "C:\\xampp\\htdocs\\"
            net stop Apache2.4
            net start Apache2.4
        """
    }
}
```

Setup Instructions

1. Configure XAMPP

1. Install XAMPP (Download) to `C:\xampp`.
2. Start Apache via XAMPP Control Panel.

2. Create Jenkins Pipeline Job

1. In Jenkins, create a **New Item** > **Pipeline**.
2. Under **Pipeline**:
    - Select **Pipeline script from SCM** > **Git**.
    - Add your GitHub repo URL.
    - Set **Script Path** to `Jenkinsfile`.

3. Run the Pipeline

1. Click **Build Now** to trigger manually.
2. Jenkins will:
    - Clone the repo.
    - Copy HTML files to `C:\xampp\htdocs`.
    - Restart Apache.

Validation

1. Open browser and navigate to:
    - `http://localhost/index.html`
    - Replace `index.html` with your actual filenames.

Troubleshooting

- **Permission Denied**:
    - Run Jenkins **as Administrator** (right-click shortcut > **Run as administrator**).
- **Apache Fails to Restart**:
    - Manually start Apache via XAMPP Control Panel.
- **Files Not Copied**:
    - Check workspace path in Jenkins (`%JENKINS_HOME%\workspace\job-name`).

Key Modifications for Windows

- Use `bat` instead of `sh` for Windows commands.
- Escape backslashes in paths (`C:\\xampp\\htdocs`).
- XAMPP's Apache service name is `Apache2.4` (use `net stop/start`).

Here's a **declarative pipeline** to deploy HTML files from GitHub to XAMPP on Windows, optimized for readability and using Jenkins best practices from the search results:

5. CI/CD Declarative Pipeline for XAMPP Deployment

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git branch: 'main',
                url: 'https://github.com/your-username/html-repo.git'
            }
        }

        stage('Deploy') {
            steps {
                bat '''
                    xcopy /Y "%WORKSPACE%\\*.html" "C:\\xampp\\htdocs\\"
                    net stop Apache2.4
                    net start Apache2.4
                '''
            }
        }
    }
}
```

Key Components Explained

1. **Checkout Stage**:
    - Uses Jenkins' native `git` step ([Search Result 3][4]) to clone the repository.
    - Replace `your-username/html-repo` with your actual GitHub URL.
2. **Deploy Stage**:
    - `xcopy` transfers HTML files from Jenkins workspace to XAMPP's `htdocs` directory.
    - Apache service restart ensures changes take effect immediately.

Windows-Specific Modifications

1. **Path Handling**:
    - Use double backslashes (`\\`) in paths
    - `%WORKSPACE%` environment variable points to Jenkins job directory
2. **Service Management**:
    - `net stop/start` commands require **Admin privileges**
    - Run Jenkins service as Administrator (via Windows Services)

Validation

1. Access deployed files at:

```
http://localhost/index.html
http://localhost/about.html
http://localhost/contact.html
```

2. Check Jenkins console output for:

```
[xcopy] 3 File(s) copied
[Apache] The Apache2.4 service was started successfully
```

Troubleshooting

- **Permission Denied**:

  ```
  post {
    always {
      bat 'icacls "C:\\xampp\\htdocs" /grant Everyone:(F)'
    }
  }
  ```

- **Apache Not Restarting**:
  - Manually verify service status in XAMPP Control Panel
  - Add `timeout 10` between stop/start commands

Experiment 6: Selenium Automated Testing (Google Search)

Objective: Automate Google search for "RAIT Dypatil" using Python Selenium

Tools Required

1. Python 3.6+
2. Chrome Browser
3. Required Packages:

   ```
   pip install selenium webdriver-manager
   ```

Step-by-Step Implementation

1. Create Python Script

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from webdriver_manager.chrome import ChromeDriverManager

# Initialize browser
driver = webdriver.Chrome(ChromeDriverManager().install())
wait = WebDriverWait(driver, 10)

try:
    # Step 1: Navigate to Google
    driver.get("https://www.google.com")

    # Step 2: Find search box
    search_box = wait.until(
        EC.presence_of_element_located((By.NAME, "q"))
    )

    # Step 3: Enter search query
    search_box.send_keys("RAIT Dypatil")

    # Step 4: Submit search
    search_box.send_keys(Keys.RETURN)

    # Step 5: Wait for results
    wait.until(
        EC.presence_of_element_located((By.ID, "search"))
    )

    # Keep browser open for 10 seconds
    input("Press Enter to close the browser...")

finally:
    # Step 6: Clean up
    driver.quit()
```

2. Run the Script

```
python rait_search.py
```

Key Components Explained

1. **Browser Initialization**
   - Uses `webdriver-manager` for automatic ChromeDriver setup[1][3]
   - Explicit wait (10 seconds) for reliable element loading[2][5]
2. **Element Interaction**
   - Locates search box using `By.NAME` selector[4][5]
   - Uses `send_keys()` for text input and form submission[2][4]
3. **Validation**
   - Waits for search results container (`id="search"`) to load
   - Manual verification via console input

Expected Workflow

1. Chrome browser opens automatically
2. Navigates to Google.com
3. Types "RAIT Dypatil" in search box
4. Displays search results
5. Browser remains open until user presses Enter

Troubleshooting

Issue	Solution
Browser doesn't open	Check Chrome/WebDriver versions
"Element not found"	Increase wait time from 10 to 15 seconds
ChromeDriver errors	Run `pip install --upgrade webdriver-manager`
Permission issues	Run script as Administrator

Best Practices

1. Use explicit waits instead of `time.sleep()` [3][5]
2. Always include `driver.quit()` to prevent zombie processes
3. Consider headless mode for background execution:

```
options = webdriver.ChromeOptions()
options.add_argument("--headless=new")
driver = webdriver.Chrome(options=options)
```

Here's the simplest way to Dockerize a PHP project and push to Docker Hub:

Experiment 7 Create an image of php project and push on Dockerhub repository

1. **Create `Dockerfile`** (no extensions):

```
FROM php:8-apache
COPY . /var/www/html/
```

2. **Build Image**:

```
docker build -t yourusername/php-project .
```

3. **Login to Docker Hub**:

```
docker login
```

4. **Push to Docker Hub**:

```
docker push yourusername/php-project
```

Example

```
# For a project named "myapp"
docker build -t raitstudent/myapp .
docker push raitstudent/myapp
```

Done!
Access it later anywhere with:

```
docker run -p 80:80 raitstudent/myapp
```

Here's a concise guide for **Experiment 8** using Docker commands on Windows:

8. Docker Image Modification, Pull and Push

Objective: Modify Ubuntu image and push to Docker Hub

Step-by-Step Commands

1. **Pull Official Ubuntu Image**:

   ```
   docker pull ubuntu:latest
   ```

2. **Create Container and Add File**:

   ```
   docker run -it --name temp-container ubuntu /bin/bash
   ```

   Inside container:

   ```
   touch /testfile
   exit
   ```

3. **Commit Changes to New Image**:

   ```
   docker commit temp-container your-dockerhub-username/ubuntu-modified
   ```

4. **Tag Image** (if needed):

   ```
   docker tag your-dockerhub-username/ubuntu-modified:latest your-dockerhub-username/ubuntu-modified:v1
   ```

5. **Push to Docker Hub**:

   ```
   docker login
   docker push your-dockerhub-username/ubuntu-modified
   ```

Verification

1. Check Docker Hub for your new repository.
2. Pull the image elsewhere:

   ```
   docker pull your-dockerhub-username/ubuntu-modified
   docker run -it --rm your-dockerhub-username/ubuntu-modified ls /
   ```

You should see `testfile` in the root directory.

Key Notes

- Replace `your-dockerhub-username` with your actual Docker ID
- Create the repository on Docker Hub first (via website) if it doesn't exist
- Use `docker ps -a` to find container ID if you closed the terminal

Time: Entire process takes <5 minutes.