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

Here's a step-by-step guide on how to push and pull 3 HTML files from a Github account using Git commands:

Assuming you have already created a Github repository and cloned it to your local machine:

1. Create 3 HTML files in your local repository. Let's call them file1.html, file2.html, and file3.html.

2. Use the command `git add .` to stage all changes made in the repository.

3. Use the command `git commit -m "Added 3 HTML files"` to commit the changes with a descriptive message.

4. Use the command `git push origin main` to push the committed changes to the remote

repository on Github.

Note: You may need to enter your Github username and password to authenticate the push.

5. To pull any changes made to the repository from Github, use the command `git pull origin main`.

Note: This command will fetch and merge any changes made on the remote repository to your local repository.

6. To verify that the HTML files were successfully pushed and pulled, navigate to your Github repository's webpage and check that the files are present.

That's it! You've successfully pushed and pulled 3 HTML files from your Github account using Git commands.

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

Sure, here's a step-by-step guide on how to build a Java program (addition of two numbers) through Jenkins:

1. Install Jenkins on your local machine or a server.

2. Create a new Jenkins job by clicking "New Item" on the Jenkins dashboard and giving it a name.

3. Under "General" settings, give a description of the job, select "Build a free-style software project" and click "OK".

4. Under "Source Code Management", select "Git" and provide the repository URL where the Java program is stored.

5. Under "Build", select "Add build step" and

choose "Execute Windows batch command" (assuming you are running Jenkins on a Windows machine).

6. In the "Command" field, enter the following code:

```

javac AddTwoNumbers.java

java AddTwoNumbers

```

This code will compile and run the Java program "AddTwoNumbers.java".

7. Save the job settings and click "Build Now" to build the Java program.

8. Once the build is complete, click on the build number under "Build History" and select "Console Output" to view the output of the Java program.

That's it! You've successfully built a Java

program (addition of two numbers) through Jenkins. Make sure that the Java program is properly stored in the Git repository before building it in Jenkins.

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

Sure, here's a step-by-step guide on how to demonstrate a continuous integration process in Jenkins by building a Java program every 2 minutes (addition of two numbers) residing in a Github repository:

1. Install Jenkins on your local machine or a server.

2. Create a new Jenkins job by clicking "New Item" on the Jenkins dashboard and giving it a name.

3. Under "General" settings, give a description of the job, select "Build a free-style software project" and click "OK".

4. Under "Source Code Management", select "Git" and provide the repository URL where the Java program is stored.

5. Under "Build Triggers", select "Build periodically" and enter "\*/2 \* \* \* \*" in the "Schedule" field. This means that the job will build every 2 minutes.

6. Under "Build", select "Add build step" and choose "Execute Windows batch command" (assuming you are running Jenkins on a Windows machine).

7. In the "Command" field, enter the following code:

```

cd C:\path\to\java\program

javac AddTwoNumbers.java

java AddTwoNumbers

```

This code will navigate to the directory where the Java program is stored, compile it, and run it.

8. Save the job settings and click "Build Now" to build the Java program for the first time.

9. Once the build is complete, click on the build number under "Build History" and select "Console Output" to view the output of the Java program.

10. To test the continuous integration process, make a change to the Java program in the Github repository and commit the changes.

11. Wait for 2 minutes and then check the Jenkins job to see if it automatically builds

the new version of the Java program.

That's it! You've successfully demonstrated a continuous integration process in Jenkins by building a Java program every 2 minutes residing in a Github repository. This process can be extended to run automated tests, generate reports, and perform other tasks to ensure the quality of the software being developed**.**

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

Sure, here's a step-by-step guide on how to demonstrate CI/CD in Jenkins by deploying 3 HTML files from a Github repository on a XAMPP server using a declarative pipeline:

1. Install Jenkins on your local machine or a server.

2. Install XAMPP server and make sure it is

running.

3. Create a new Jenkins job by clicking "New Item" on the Jenkins dashboard and giving it a name.

4. Under "General" settings, give a description of the job, select "Pipeline" and click "OK".

5. Under "Pipeline", select "Definition" as "Pipeline script" and paste the following code:

```

pipeline {

agent any

stages {

stage('Clone repository') {

steps {

git 'https://github.com/username/repository.git'

}

}

stage('Deploy to XAMPP') {

steps {

bat 'xcopy /E /Y C:\\path\\to\\repository C:\\path\\to\\xampp\\htdocs'

}

}

}

}

```

This code creates a pipeline with two stages: "Clone repository" and "Deploy to XAMPP". The first stage clones the Github repository, while the second stage deploys the 3 HTML files from the repository to the XAMPP server.

Note: Replace "username/repository" with the URL of your Github repository and "C:\\path\\to\\repository" and "C:\\path\\to\\xampp\\htdocs" with the actual paths on your system.

6. Save the job settings and click "Build Now" to deploy the 3 HTML files on the XAMPP server.

7. Once the deployment is complete, check the XAMPP server to ensure that the HTML files are properly deployed.

8. To test the CI/CD process, make a change to one of the HTML files in the Github repository and commit the changes.

9. Wait for the Jenkins job to automatically trigger the pipeline and deploy the new version of the HTML file on the XAMPP server.

That's it! You've successfully demonstrated CI/CD in Jenkins by deploying 3 HTML files from a Github repository on a XAMPP server using a declarative pipeline. This process can be extended to include other stages such as automated testing, quality checks, and deployment to production servers.

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

Sure, here's a step-by-step guide on how to demonstrate CI/CD in Jenkins by deploying 3 HTML files from a Github repository on a XAMPP server using a declarative pipeline:

1. Install Jenkins on your local machine or a server.

2. Install XAMPP server and make sure it is running.

3. Create a new Jenkins job by clicking "New Item" on the Jenkins dashboard and giving it a name.

4. Under "General" settings, give a description of the job, select "Pipeline" and click "OK".

5. Under "Pipeline", select "Definition" as "Pipeline script" and paste the following code:

```

pipeline {

agent any

stages {

stage('Clone repository') {

steps {

git 'https://github.com/username/repository.git'

}

}

stage('Deploy to XAMPP') {

steps {

bat 'xcopy /E /Y C:\\path\\to\\repository C:\\path\\to\\xampp\\htdocs'

}

}

}

}

```

This code creates a pipeline with two stages: "Clone repository" and "Deploy to XAMPP". The first stage clones the Github repository, while the second stage deploys the 3 HTML files from the repository to the XAMPP server.

Note: Replace "username/repository" with the URL of your Github repository and "C:\\path\\to\\repository" and "C:\\path\\to\\xampp\\htdocs" with the actual paths on your system.

6. Save the job settings and click "Build Now" to deploy the 3 HTML files on the XAMPP server.

7. Once the deployment is complete, check the XAMPP server to ensure that the HTML files are properly deployed.

8. To test the CI/CD process, make a change to one of the HTML files in the Github repository and commit the changes.

9. Wait for the Jenkins job to automatically trigger the pipeline and deploy the new version of the HTML file on the XAMPP server.

That's it! You've successfully demonstrated CI/CD in Jenkins by deploying 3 HTML files from a Github repository on a XAMPP server using a declarative pipeline. This process can be extended to include other stages such as automated testing, quality checks, and deployment to production servers.

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

Sure! Here's an example Selenium script in Python to perform automated testing on a website:

```python

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

# create a new instance of the Chrome driver

driver = webdriver.Chrome()

# navigate to the website to be tested

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

# find the search box element and enter a query

search\_box = driver.find\_element\_by\_name("q")

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

# submit the search query

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

# verify that the search results page is displayed

assert "Google Search Results" in driver.title

# find the first search result link and click on it

search\_result = driver.find\_element\_by\_css\_selector("#search .g:nth-of-type(1) a")

search\_result.click()

# verify that the correct page is displayed

assert "Selenium" in driver.title

# close the browser window

driver.close()

```

This script uses the Selenium WebDriver API to open a Chrome browser window, navigate to a website, enter a search query, submit the query, and verify that the correct search results page and page title are displayed. It then clicks on the first search result link and verifies that the correct page is displayed, before closing the browser window.

Note that the script assumes that the

ChromeDriver executable is installed and added to the PATH environment variable, and that the website being tested has a search box with name "q" and search result links with CSS selector "#search .g:nth-of-type(1) a". You may need to modify the script accordingly for your specific testing scenario**.**

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

Sure! Here's a step-by-step guide on how to create a Docker image of a PHP project and push it to Dockerhub:

1. Install Docker on your local machine or server.

2. Create a new directory for your PHP project and add all the necessary files (e.g., index.php, config.php, etc.).

3. Create a new file named "Dockerfile" in the project directory and add the following code:

```

# Use an official PHP runtime as a parent image

FROM php:7.4-apache

# Set the working directory to /var/www/html

WORKDIR /var/www/html

# Copy the current directory contents into the container at /var/www/html

COPY . /var/www/html

# Install any needed packages

RUN apt-get update && apt-get install -y \

git \

zip \

unzip \

&& rm -rf /var/lib/apt/lists/\*

# Install Composer

RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --

filename=composer

# Install dependencies

RUN composer install --no-dev --no-interaction

# Expose port 80 for Apache

EXPOSE 80

# Start the Apache web server

CMD ["apache2-foreground"]

```

This code creates a Docker image using the official PHP 7.4 runtime with Apache, sets the working directory to "/var/www/html", copies the contents of the current directory to the container, installs necessary packages, installs Composer, installs project dependencies, exposes port 80 for Apache, and starts the Apache web server.

4. Build the Docker image by running the following command in the project directory:

```

docker build -t <username>/<repository>:<tag> .

```

5. After the image is built, push it to your Dockerhub repository by running the following command:

```

docker push <username>/<repository>:<tag>

```

This command pushes the Docker image to your Dockerhub repository.

6. Once the push is complete, you can verify that the image is available on Dockerhub by logging into Dockerhub, navigating to your project repository, and checking the "Tags" tab.

That's it! You've successfully created a Docker image of a PHP project and pushed it to Dockerhub. You can now use this image to run your project in a container on any system with Docker installed**.**

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

Sure! Here's a step-by-step guide on how to pull the official Ubuntu Docker image, create a file in a container, and push the updated image to Dockerhub:

1. Install Docker on your local machine or server.

2. Pull the official Ubuntu Docker image by running the following command:

```

docker pull ubuntu

```

This command downloads the latest version of the Ubuntu Docker image from the Docker Hub registry.

3. Start a new container from the Ubuntu image by running the following command:

```

docker run -it --name mycontainer ubuntu /bin/bash

```

This command starts a new container with a name of "mycontainer", using the Ubuntu image, and opens a Bash shell in the container. The "-it" option is used to create an interactive terminal session.

4. Once inside the container, create a new file by running the following command:

```

echo "Hello, world!" > myfile.txt

```

This command creates a new file named "myfile.txt" in the current directory and writes the text "Hello, world!" to it.

5. Exit the container by running the following command:

```

exit

```

6. Commit the changes made to the container to create a new Docker image by running the following command:

```

docker commit mycontainer <username>/<repository>:<tag>

```

7. Push the updated Docker image to your Dockerhub repository by running the following command:

```

docker push <username>/<repository>:<tag>

```

This command pushes the Docker image to your Dockerhub repository.

8. Once the push is complete, you can verify that the image is available on Dockerhub by logging into Dockerhub, navigating to your project repository, and checking the "Tags" tab.

That's it! You've successfully pulled the official Ubuntu Docker image, created a file in a container, and pushed the updated image to Dockerhub.