**EXPERIMENT NO: 3.** Practice Source code management on GitHub. Experiment with the source code written in exercise 1

**Aim:** Practice Source code management on GitHub.

**Description:**
To practice source code management on GitHub, you can follow these steps:

- Create a GitHub account if you don't already have one.
- Create a new repository on GitHub.
- Clone the repository to your local machine:
  ```
  $ git clone <repository URL>
  ```
- Move to the repository directory:
  ```
  $ cd < repositoty-name>
  ```
- Create a new file in the repository and add the source code written in exercise 1.
- Stage the changes:
  ```
  $ git add <file-name>
  ```
- Commit the changes:
  ```
  $ git commit -m "Added source code for a simple user registration form"
  ```
- Push the changes to the remote repository:
  ```
  $ git push origin master
  ```
- Verify that the changes are reflected in the repository on GitHub.

These steps demonstrate how to use GitHub for source code management. You can use the same steps to manage any source code projects on GitHub. Additionally, you can also explore GitHub features such as pull requests, code review, and branch management to enhance your source code management workflow.

**EXPERIMENT NO: 4.**
Jenkins installation and setup, explore the environment

**Aim:** Jenkins installation and setup, explore the environment

**DESCRIPTION:**
Jenkins is a popular open-source tool for Continuous Integration and Continuous Deployment (CI/CD) in software development.
Here are the steps to install and set up Jenkins:

**Download and install Jenkins:**

- Download the Jenkins package for your operating system from the Jenkins website.
- Follow the installation instructions for your operating system to install Jenkins.

**Start the Jenkins service:**

- On Windows, use the Windows Services Manager to start the Jenkins service.
- On Linux, use the following command to start the Jenkins service:

```
$ sudo service jenkins start
```

**Access the Jenkins web interface:**

- Open a web browser and navigate to http://localhost:8080 to access the Jenkins web interface.
- If the Jenkins service is running, you will see the Jenkins login page.

**Initialize the Jenkins environment:**

- Follow the instructions on the Jenkins setup wizard to initialize the Jenkins environment.
- This process involves installing recommended plugins, setting up security, and creating the first admin user.

**Explore the Jenkins environment:**

- Once the Jenkins environment is set up, you can explore the various features and functionalities available in the web interface.
- Jenkins has a rich user interface that provides access to features such as build history, build statistics, and system information.

These are the basic steps to install and set up Jenkins. Depending on your use case, you may need to customize your Jenkins environment further.

## <u>EXPERIMENT NO: 5</u>

Demonstrate continuous integration and development using Jenkins.

**<u>Aim:</u>** To demonstrate continuous integration and development using Jenkins

**<u>DESCRIPTION:</u>** Continuous Integration (CI) and Continuous Development (CD) are important practices in software development that can be achieved using Jenkins.
Here's an example of how you can demonstrate CI/CD using Jenkins:

- Create a simple Java application:
- Create a simple Java application that you want to integrate with Jenkins.
- The application should have some basic functionality, such as printing "Hello World" or performing simple calculations.

**Commit the code to a Git repository:**

- Create a Git repository for the application and commit the code to the repository.
- Make sure that the Git repository is accessible from the Jenkins server.

**Create a Jenkins job:**

- Log in to the Jenkins web interface and create a new job.
- Configure the job to build the Java application from the Git repository.
- Specify the build triggers, such as building after every commit to the repository.

**Build the application:**

- Trigger a build of the application using the Jenkins job.
- The build should compile the code, run any tests, and produce an executable jar file.

**Monitor the build:**

- Monitor the build progress in the Jenkins web interface.
- The build should show the build log, test results, and the status of the build.

**Deploy the application:**

• If the build is successful, configure the Jenkins job to deploy the application to a production environment.

• The deployment could be as simple as copying the jar file to a production server or using a more sophisticated deployment process, such as using a containerization technology like Docker.

**Repeat the process:**

• Repeat the process for subsequent changes to the application

• Jenkins should automatically build and deploy the changes to the production environment. This is a basic example of how you can use Jenkins to demonstrate CI/CD in software development.

## **<u>EXPERIMENT NO.: 6.</u>**

Explore Docker commands for content management.

**<u>AIM:</u>** To explore Docker commands for content management.

**<u>DESCRIPTION</u>**: Docker is a containerization technology that is widely used for managing application containers. Here are some commonly used Docker commands for content management:

• Docker ps: List containers.

For example: $ docker ps

This command lists all running containers.

• Docker images: List images.

For example: $ docker images

This command lists all images stored locally on the host.

•Build the Docker image: Run the following command to build the Docker image

$ docker build -t myimage .

This command builds a new Docker image using the Dockerfile and tags the image with the name "myimage".

• Docker run: Run a command in a new container.

For example: $ docker run -d -p 8080:80 --name mycontainer

This command is used to create and start a new Docker container(my container) based on a Docker image.

**-d**: This flag stands for "detached mode". It means the container runs in the background (daemon mode), and the terminal prompt is returned to you immediately after starting the container.

**-p 8080:80**: This option specifies how Docker should map network ports between the Docker container and the host machine. It tells Docker to map port 80 inside the container to port 8080 on the host machine. This means that if you access port 8080 on your host machine, the traffic will be forwarded to port 80 inside the Docker container.

• Docker push: Push an image or a repository to a registry.

For example: $ docker push myimage

This command pushes a Docker image to a Docker registry, such as Docker Hub or a private registry.

• Docker pull: Pull an image or a repository from Docker registry.
For example: $ docker pull ubuntu:16.04
This command pulls the Ubuntu 16.04 image from the Docker Hub registry.

• Docker start: Start one or more stopped containers.
For example: $ docker start mycontainer
This command starts the container named "mycontainer".

• Docker stop: Stop one or more running containers.
For example: $ docker stop mycontainer
This command stops the container named "mycontainer".

• Docker rm: Remove one or more containers.
For example: $ docker rm mycontainer
This command removes the container named "mycontainer".

## EXPERIMENT NO.: 7

Develop a simple containerized application using Docker

**AIM:** To develop a simple containerized application using Docker

**DESCRIPTION :**

Here's an example of how you can develop a simple containerized application using Docker:

**Choose an application:**

• Choose a simple application that you want to containerize.
For example, create a simple Python script (`app.py` for example) that you want to run using Docker. Here's an example script that prints "Hello, Docker!" when executed:

**Create a Dockerfile:**

• Create a file named "Dockerfile" in the same directory as the application. The Dockerfile defines the environment in which your application will run and the steps needed to set up that environment.
In the Dockerfile, specify the base image, copy the application into the container, and specify the command to run the application.
Here's an example Dockerfile for a Python script:

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
```

```
COPY . /app

# Run the Python script when the container launches
CMD ["python", "./app.py"]
```

**Build the docker image:**
Now, build the Docker image using the Dockerfile. Open a terminal or command prompt, navigate to the directory where your Dockerfile and app.py are located, and run the following command:

```
docker build -t mypythonapp .
```

**Run the Docker Container:**
After successfully building the Docker image, you can run it as a container. Use the docker run command to start the container based on the image you just built:

```
docker run –name mycontainer mypythonapp
```

**Verify the output:**

Run the following command to verify the output of the container:

$ docker logs mycontainer This command displays the logs of the container and should show the "Hello World" output.

It can also be accessed from your local host, you need to ensure that the Docker container's port is mapped to your host machine.

**Run the Docker Container with Port Mapping:**
After successfully building the Docker image, run the container and map a port from the container to your host machine. This allows you to access the Python script running inside the container from your local host.

```
docker run -d -p 8086:80 mypythonapp
```