Sure, here are the steps to create a version control account on GitHub, create a repository, and push your code to GitHub using Git commands:

1. **Create a GitHub Account:**

   If you haven't already, go to [GitHub](https://github.com/) and sign up for an account.

2. **Install Git:**

   If you haven't already, install Git on your local machine. You can download it from the official [Git website](https://git-scm.com/).

3. **Configure Git:**

   Open a terminal or command prompt and configure Git with your username and email address using the following commands:

   ```
   git config --global user.name "Your GitHub Username"
   git config --global user.email "your_email@example.com"
   ```

4. **Create a New Repository on GitHub:**

   - Log in to your GitHub account.
   - Click on the "+" icon in the top right corner and select "New repository".
   - Enter a name for your repository, add a description if you want, choose whether it's public or private, and click on "Create repository".

5. **Initialize a Local Git Repository:**

   Open a terminal or command prompt, navigate to the directory where your code is located, and initialize a Git repository using the following command:

   ```
   git init
   ```

6. **Add Your Files to the Repository:**

Add the files you want to track to the repository using the following command:

```
git add .
```

7. **Commit Your Changes:**

Commit the files you've added to the repository with a descriptive message using the following command:

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

8. **Link Your Local Repository to the GitHub Repository:**

Copy the URL of your GitHub repository. Then, link your local repository to the GitHub repository using the following command:

```
git remote add origin <repository_url>
```

Replace `<repository_url>` with the URL of your GitHub repository.

9. **Push Your Code to GitHub:**

Push your committed changes to the GitHub repository using the following command:

```
git push -u origin master
```

If you are working on a branch other than master, replace `master` with the name of your branch.

10. **Authenticate with GitHub:**

If prompted, enter your GitHub username and password to authenticate and complete the push.

That's it! Your code should now be pushed to your GitHub repository. You can verify by visiting your repository on the GitHub website.

Git and GitHub are related but serve different purposes in the context of version control and collaborative software development:

1. **Git:**

- Git is a distributed version control system (DVCS) designed for tracking changes in source code during software development.

- It allows developers to work on projects locally, create branches for feature development or bug fixes, commit changes, merge branches, and revert to previous states.

- Git operates locally on a developer's machine, storing project history and metadata in a hidden `.git` directory within the project folder.

- It provides command-line tools and a set of commands for version control operations like commit, branch, merge, and more.

2. **GitHub:**

- GitHub is a web-based platform built around Git that provides hosting for Git repositories and adds additional features for collaboration, code review, and project management.

- It allows developers to host their Git repositories remotely on GitHub's servers, making it easier to share code with others and collaborate on projects.

- GitHub offers features like issue tracking, pull requests, code review tools, project boards, and wikis, facilitating collaboration and communication among team members.

- GitHub provides a graphical user interface (GUI) on top of Git, making it easier for users to interact with repositories, browse code, and manage projects.

- While Git is primarily a version control system, GitHub extends its functionality to provide a comprehensive platform for hosting, sharing, and collaborating on software projects.

In summary, Git is the version control system itself, while GitHub is a web-based hosting platform and collaboration tool built around Git. Developers use Git to manage their code locally, while GitHub provides a centralized platform for hosting Git repositories, facilitating collaboration, and managing software projects.

Creating a Docker container environment involves the following steps:

1. **Install Docker:**

   Ensure Docker is installed on your system. Docker provides installation instructions for various operating systems on its website.

2. **Write a Dockerfile:**

   Create a `Dockerfile` in your project directory. This file defines the steps needed to build your Docker image. It specifies the base image, copies files into the image, and configures the environment.

3. **Build the Docker Image:**

   Run the `docker build` command to build the Docker image based on the `Dockerfile` you created. For example:

   ```
   docker build -t my_image_name .
   ```

4. **Run a Docker Container:**

   Once the image is built, you can run a Docker container using the `docker run` command. Specify the image name and any additional options needed, such as port mappings or volume mounts. For example:

   ```
   docker run -d -p 8080:80 my_image_name
   ```

5. **Access the Container:**

You can interact with the running container through the Docker command-line interface or by accessing services running inside the container. For example, if your container runs a web server, you can access it by navigating to `http://localhost:8080` in your web browser.

6. **Manage Containers:**

Docker provides commands to manage containers, such as `docker ps` to list running containers, `docker stop` to stop a container, and `docker rm` to remove a container.

7. **Cleanup:**

Once you're done with your container environment, you can clean up by stopping and removing containers using the `docker stop` and `docker rm` commands, respectively. You can also remove unused Docker images using `docker rmi`.

This is a basic overview of creating a Docker container environment. Docker provides extensive documentation and resources for more advanced usage and best practices.