Task 1: Git Basic - Clone and Commit Changes

- **Objective**: Learn to clone a Git repository and make changes to it.
- **Steps**:
    1. Clone a repository from GitHub or GitLab using `git clone`.
    2. Make changes to any file in the repository.
    3. Stage and commit your changes using `git add <file>` and `git commit -m "commit message"`.
    4. Push the changes to the remote repository using `git push`.
- **Question**:
    - What happens when you run `git commit -m "message"` in your local repository? What is the purpose of the commit message?

Task 2: Git Branching and Merging

- **Objective**: Understand how to create branches and merge them into the main branch.
- **Steps**:
    1. Create a new branch using `git checkout -b <branch-name>`.
    2. Make some changes in the new branch.
    3. Commit your changes.
    4. Switch back to the main branch using `git checkout main`.
    5. Merge the new branch into the main branch using `git merge <branch-name>`.
    6. Push the changes to the remote repository.
- **Question**:
    - What is the difference between `git merge` and `git rebase`? Which one would you prefer for preserving the commit history?

Task 3: Git Rebase vs. Git Merge

- **Objective**: Learn the difference between `git merge` and `git rebase` and when to use them.
- **Steps**:
    1. Create a new branch and commit changes.
    2. Use `git merge` to combine the new branch with the main branch.
    3. Next, perform the same task using `git rebase` instead of merge.
    4. Compare the results of both approaches.
- **Question**:
    - What are the advantages and disadvantages of using `git rebase` over `git merge`?

Task 4: Git Stash to Save Changes Temporarily

- **Objective**: Understand how to save and retrieve uncommitted changes using `git stash`.
- **Steps**:
    1. Modify some files in the working directory without committing them.
    2. Use `git stash` to save the changes temporarily.
    3. Switch to a different branch and make some changes there.
    4. Use `git stash pop` to reapply the stashed changes back into the original branch.
- **Question**:
    - When should you use `git stash` instead of committing changes? What could be the risks of using it frequently?

Task 5: CI/CD - Set Up a Simple CI Pipeline with GitLab CI

- **Objective**: Learn how to set up a simple CI pipeline using GitLab CI.
- **Steps**:
    1. Create a new project in GitLab.
    2. Add a `.gitlab-ci.yml` file in the root of the repository with a simple job (e.g., echo a message).
    3. Commit and push the file.
    4. Observe the pipeline running in GitLab's CI/CD section.
- **Question**:
    - What is the role of `.gitlab-ci.yml` in a GitLab CI pipeline? How does it define the CI/CD process?

Task 6: Continuous Delivery (CD) with Jenkins

- **Objective**: Set up a Continuous Delivery (CD) pipeline using Jenkins.
- **Steps**:
    1. Install Jenkins on your local machine or use an online service.
    2. Create a Jenkins pipeline job.
    3. Configure the job to pull from a Git repository.
    4. Add steps to build and deploy your application to a test server.
    5. Trigger the pipeline and check the results.
- **Question**:
    - What are the key differences between Continuous Integration (CI) and Continuous Delivery (CD)? Why is CD important?
    -

Task 7: Setting Up Nginx as a Reverse Proxy

- **Objective**: Configure Nginx to act as a reverse proxy.
- **Steps**:
    1. Install Nginx on your server.
    2. Edit the Nginx configuration file (`/etc/nginx/nginx.conf`) to forward traffic to a backend server (e.g., `http://localhost:3000`).
    3. Restart Nginx.
    4. Test by accessing Nginx's public IP or domain in a browser to see if traffic is forwarded correctly.
- **Question**:
    - What is the role of `proxy_pass` in the Nginx reverse proxy configuration? How does it affect incoming requests?

Task 8: Load Balancing with Nginx

- **Objective**: Configure Nginx to perform load balancing between two or more backend servers.
- **Steps**:
    1. Set up two or more backend servers (you can use localhost with different ports).
    2. In Nginx's configuration file, configure the `upstream` directive to define the backend servers.
    3. Use the `proxy_pass` directive to forward incoming traffic to the upstream group.
    4. Restart Nginx and test load balancing by sending requests.
- **Question**:
    - How does Nginx handle load balancing by default? What algorithms can Nginx use for load balancing?

Task 9: Apache HTTPD Reverse Proxy Setup

- **Objective**: Learn to set up Apache HTTPD as a reverse proxy server.
- **Steps**:
    1. Install Apache HTTPD on your server.
    2. Enable `mod_proxy` and `mod_proxy_http` modules.
    3. Edit the Apache configuration file to route traffic to a backend server.
    4. Restart Apache and test the reverse proxy setup.
- **Question**:
    - What are the main differences between Apache HTTPD and Nginx when used as reverse proxies? Which one would you prefer for your project and why?

Task 10: Continuous Deployment with Docker and Jenkins

- **Objective**: Learn to set up a continuous deployment pipeline with Docker and Jenkins.
- **Steps**:
    1. Create a simple Docker containerized application (e.g., Node.js or Python app).
    2. Set up Jenkins to trigger a build when changes are pushed to Git.
    3. Add steps to Jenkins to build the Docker image and deploy it to a Docker registry (e.g., Docker Hub).
    4. Pull the image from the registry and deploy it to a server using Docker.
- **Question**:
    - How does Docker help in continuous deployment pipelines? What are the benefits of using Docker in this workflow?