**DAY 4 TASKS**

**Task 1:** Git Basic - Clone and Commit Changes

**Objective**: Learn to clone a Git repository and make changes to it**.**

**Steps:**

* Clone a repository from GitHub or GitLab using git clone.
* Make changes to any file in the repository.
* Stage and commit your changes using git add and git commit -m "commit message".
* Push the changes to the remote repository using git push.

A computer screen shot of a program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Question:** What happens when you run git commit -m "message" in your local repository? What is the purpose of the commit message?

**Ans:** When we run git commit -m "message" in your local repository, the following happens:

* **Staging Area to Local Repository:**

Git takes the changes from the staging area and saves them as a snapshot in your local repository

* **Commit Creation:**

A commit object is created in the repository.

* **Local Changes are Saved:**

The commit acts as a save point for changes, allowing you to track the project history and revert to this state if needed.

The purpose of the commit message are :

* **Documentation** : Provides context and explains why the changes were made.
* **Collaboration :** Helps team members to understand the changes without looking at the code.
* **History Tracking:** Facilities reviewing and debugging by offering a clear record of project’s development history.
* **Future Reference :** Allows us to quickly identify changes or bug fixes when revisiting the codebase later.

**Task 2:** Git Branching and Merging

**Objective:** Understand how to create branches and merge them into the main branch.

**Steps:**

* Create a new branch:

git checkout -b feature-branch

* Modify a file (e.g., add new content to README.md).
* Stage and commit the changes:

git add README.md

git commit -m "Added feature details"

* Switch back to the main branch:

git checkout main

* Merge the feature-branch into main:

git merge feature-branch

* Push the changes to the remote repository:

git push origin main

**Question**:

**What is the difference between git merge and git rebase? Which one would you prefer for preserving the commit history?**

git merge combines changes from different branches into the current branch. It merges the commit histories of both branches.

**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:

git checkout -b feature-branch

echo "Some feature changes" > feature.txt

git add feature.txt

git commit -m "Added feature changes"

1. Merge the branch:

git checkout main

git merge feature-branch

1. Rebase the branch:

git checkout feature-branch

git rebase main

1. Compare the commit histories:

git log --oneline

**Question**:

What are the advantages and disadvantages of using git rebase over git merge?

git merge combines the branches, preserving both histories. In contrast, git rebase rewrites commit history, making it appear as though the changes were applied directly on top of the target branch, resulting in a cleaner, linear history.

**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 but do not commit (e.g., edit feature.txt).
2. Stash the changes:

git stash

1. Switch to another branch:

git checkout main

1. Return to the original branch and apply the stashed changes:

git checkout feature-branch

git stash pop

**Question**:

When should you use git stash instead of committing changes? What could be the risks of using it frequently?

git stash temporarily saves your changes, allowing you to switch branches without committing them. You can later reapply the changes using git stash pop.

**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 GitLab repository.
2. Add the .gitlab-ci.yml file to the root of the repository with the following content:

stages:

- build

build\_job:

stage: build

script:

-echo "Hello, GitLab CI!"

1. Commit and push the file:

git add .gitlab-ci.yml

git commit -m "Add GitLab CI configuration"

git push origin main

1. In GitLab, go to **CI/CD > Pipelines** to see the pipeline running.

**Question**:

o What is the role of .gitlab-ci.yml in a GitLab CI pipeline? How does it define the CI/CD process?

.gitlab-ci.yml defines the pipeline stages and jobs. The build job runs a simple script to print "Hello, GitLab CI!". This file automates your CI process by defining the steps to execute when code is pushed to the repository.

**Task 6: Continuous Delivery (CD) with Jenkins**

**Objective**: Set up a Continuous Delivery (CD) pipeline using Jenkins.

**Steps**:

1. Install Jenkins and start it on your local machine.
2. .Create a new Jenkins pipeline job.
3. Configure the pipeline to pull from your Git repository (e.g., using a GitHub webhook).
4. Add the build and deploy steps :

A computer code with black text

Description automatically generated

1. Trigger the pipeline manually or automatically after a commit.

**Question**:

What are the key differences between Continuous Integration (CI) and Continuous Delivery (CD)? Why is CD important?

**Continuous Integration (CI)** focuses on automating the building and testing of code changes. Developers frequently integrate their code changes into a shared repository, triggering automated builds and tests. This ensures early detection of integration issues and promotes code quality.

**Continuous Delivery (CD)** builds upon CI by automating the entire software release process, from build to deployment. It involves automating the testing, building, and deployment of code changes to production environments. This enables faster and more frequent software releases.

**CD is Important:**

* **Faster Time to Market:** CD accelerates the delivery of new features and bug fixes to customers.
* **Improved Quality:** Automated testing catches defects early in the development cycle, leading to higher-quality software.
* **Reduced Risk:** Automated deployments minimize human error and reduce the risk of deployment failures.
* **Increased Efficiency:** Automation streamlines the development and release process, freeing up developers to focus on innovation.
* **Enhanced Customer Satisfaction:** Frequent, reliable releases lead to happier customers.

**Task 7: Setting Up Nginx as a Reverse Proxy**

**Objective**: Configure Nginx to act as a reverse proxy.

**Steps**:

A screenshot of a computer program

Description automatically generated

**Question**:

o What is the role of proxy\_pass in the Nginx reverse proxy configuration? How does it affect incoming requests?

The proxy\_pass directive forwards requests to the backend server. The proxy\_set\_header directives ensure that relevant information (such as the original IP address) is passed along to the backend.

**Task 8: Load Balancing with Nginx**

**Objective**: Configure Nginx to perform load balancing between two or more backend servers.

**Steps**:

A screenshot of a computer program

Description automatically generated

**Question**:

How does Nginx handle load balancing by default? What algorithms can Nginx use for load balancing?

Nginx uses round-robin by default for load balancing, distributing traffic evenly across multiple backend servers. You can also configure other algorithms like IP hash or least\_conn.

**Task 9: Apache HTTPD Reverse Proxy Setup**

**Objective**: Learn to set up Apache HTTPD as a reverse proxy server.

**Steps**:

A computer screen shot of a computer code

Description automatically generated

A close-up of a computer code

Description automatically generated

**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?

Apache uses ProxyPass and ProxyPassReverse to forward client requests to a backend server. ProxyPass forwards requests, and ProxyPassReverse ensures the correct response headers are returned to the client.

**Task 10: Continuous Deployment with Docker and Jenkins**

**Objective**: Learn to set up a continuous deployment pipeline with Docker and Jenkins.

**Steps**:

A screenshot of a computer program

Description automatically generated

**Question**:

How does Docker help in continuous deployment pipelines? What are the benefits of using Docker in this workflow?

Docker containers provide a consistent environment for application deployment. Jenkins automates building and deploying the Docker container, ensuring the latest changes are deployed efficiently to the production server.