Lab sheet: 6

Objective:
To understand and implement version control using GitLab by creating a repository, managing branches, merging changes, and setting up a basic CI/CD pipeline.

Key Features of GitLab:

- Version Control: GitLab provides version control capabilities using Git, allowing developers to track changes to source code over time. This enables collaboration, change tracking, and code history maintenance.
- Repositories: Repositories on GitLab are collections of files, code, documentation, and assets related to a project. Each repository can have multiple branches and tags, allowing developers to work on different features simultaneously.
- Continuous Integration/Continuous Deployment (CI/CD): GitLab offers robust CI/CD capabilities. It automates the building, testing, and deployment of code changes, ensuring that software is delivered rapidly and reliably.
- Merge Requests: Merge requests in GitLab are similar to pull requests in other platforms. They enable developers to propose code changes, collaborate, and get code reviewed before merging it into the main codebase.
- Issues and Project Management: GitLab includes tools for managing project tasks, bugs, and enhancements. Issues can be assigned, labeled, and tracked, while project boards help visualize and manage work.
- Container Registry: GitLab includes a container registry that allows users to store and manage Docker images for applications.
- Code Review and Collaboration: Built-in code review tools facilitate collaboration among team members. Inline comments, code discussions, and code snippets are integral parts of this process.
- Wiki and Documentation: GitLab provides a space for creating project wikis and documentation, ensuring that project information is easily accessible and well-documented.
- Security and Compliance: GitLab offers security scanning, code analysis, and compliance features to help identify and address security vulnerabilities and ensure code meets compliance standards.
- GitLab Pages: Similar to GitHub Pages, GitLab Pages lets users publish static websites directly from a GitLab repository.

Benefits of Using GitLab:

- End-to-End DevOps: GitLab offers an integrated platform for the entire software development and delivery process, from code writing to deployment.
- Simplicity: GitLab provides a unified interface for version control, CI/CD, and project management, reducing the need to use multiple tools.
- Customizability: GitLab can be self-hosted on-premises or used through GitLab's cloud service. This flexibility allows organizations to choose the hosting option that best suits their needs.
- Security: GitLab places a strong emphasis on security, with features like role-based access control (RBAC), security scanning, and compliance checks.
- Open Source and Enterprise Versions: GitLab offers both a free, open-source Community Edition and a paid, feature-rich Enterprise Edition, making it suitable for individual developers and large enterprises alike.

Prerequisites:

- Basic knowledge of Git commands and version control concepts
- A GitLab account
- Git installed on your local machine

Experiment Steps:

Step 1: Create a GitLab Repository

- Log in to GitLab (https://gitlab.com/).
- Click on **New Project** and select **Create Blank Project**.
- Provide a project name (e.g., gitlab-demo) and set it to **Public/Private** as required.
- Click **Create Project**.

Step 2: Clone the Repository to Your Local Machine

- Copy the repository URL from GitLab.
- Open a terminal and run:
- git clone <repository_url>
- cd gitlab-demo

Step 3: Create and Manage Branches

- Create a new branch:
- git checkout -b feature-branch
- Make changes in a file (e.g., README.md).
- Add and commit changes:
- git add README.md
- git commit -m "Updated README with feature details"
- Push the changes to GitLab:
- git push origin feature-branch

Step 4: Merge Changes Using Merge Requests

- Go to GitLab and navigate to **Merge Requests**.
- Click **New Merge Request** and select feature-branch as the source branch and main as the target branch.
- Click **Compare Branches and Continue**.
- Add a description and click **Create Merge Request**.
- Once reviewed, click **Merge** to merge the branch into main.

Step 5: Set Up a Basic CI/CD Pipeline

- In the root directory of your project, create a file named .gitlab-ci.yml.
- Add the following content to automate a simple build process:

```
stages:
  - test

test-job:
  stage: test
  script:
    - echo "Running Tests..."
    - echo "Tests Completed Successfully"
```

- Commit and push the .gitlab-ci.yml file to GitLab.
- Navigate to **CI/CD > Pipelines** in GitLab to see the pipeline running.

Step 6: Collaborate Using Issues and Milestones

- Go to **Issues** in your GitLab project.
- Click **New Issue** and provide a title and description.
- Assign the issue to a team member and add labels.
- Track progress and close the issue upon completion.

Expected Outcome:

- Successfully created and managed a GitLab repository.
- Used branching and merging efficiently.
- Implemented a simple CI/CD pipeline.
- Used GitLab's issue tracker for collaboration.

Conclusion:

This lab experiment provides hands-on experience in using GitLab for version control, collaboration, and CI/CD automation, which are essential skills in DevOps and modern software development.