GitLab Cheat Sheet

A comprehensive cheat sheet for using GitLab, covering essential commands, features, and workflows for version control and collaboration.

GitLab Basics

Core Concepts

Repository: A centralized storage location for code, files, and version history.

Branch: An independent line of development, allowing for parallel work and feature isolation.

Merge Request (MR): A proposal to merge changes from one branch into another. It includes code review, discussions, and automated checks.

Pipeline: An automated workflow that defines the steps required to build, test, and deploy code.

CI/CD: Continuous Integration and Continuous Delivery/Deployment. A set of practices to automate the software release process.

GitLab Workflow

1. **Create a Branch:** Start a new feature or bug fix by creating a branch from the main branch.
2. **Develop and Commit:** Make changes on your branch and commit them with descriptive messages.
3. **Push to GitLab:** Push your branch to the remote GitLab repository.
4. **Create a Merge Request:** Open a merge request to propose your changes to the main branch.
5. **Review and Discuss:** Collaborate with reviewers to address feedback and improve the code.
6. **Run pipelines:** Configure CI/CD pipelines to validate changes before merging.
7. **Merge:** Once approved, merge the changes into the main branch.

Basic Git Commands

`git clone `	Clone a repository from GitLab to your local machine.
`git checkout -b `	Create and switch to a new branch.
`git add .`	Stage all changes for commit.
`git commit -m ""`	Commit staged changes with a descriptive message.
`git push origin `	Push the local branch to the remote GitLab repository.
`git pull origin `	Pull the latest changes from the remote branch to your local branch.

Merge Requests & Code Review

Creating Merge Requests

1. **Push your branch:** After committing changes locally, push your branch to the remote GitLab repository using `git push origin <branch_name>`.
2. **Navigate to GitLab:** Go to your project on GitLab and you should see a prompt to create a merge request for your recently pushed branch.
3. **Fill in details:** Provide a title and description for your merge request. Explain the changes you've made and why they are necessary.
4. **Assign reviewers:** Choose one or more reviewers to review your code. Consider assigning individuals with expertise in the affected areas.
5. **Submit the merge request:** Once you've filled in all the necessary details, submit the merge request.

Code Review Process

1. **Receive notification:** Reviewers will receive a notification about the new merge request.
2. **Review the code:** Reviewers should carefully examine the changes, looking for potential bugs, security vulnerabilities, and adherence to coding standards.
3. **Provide feedback:** Use GitLab's commenting features to provide feedback directly on the code. Be clear and constructive in your comments.
4. **Iterate and improve:** The author should address the feedback and make necessary changes. Push the updated code to the branch, which will automatically update the merge request.
5. **Approve or request changes:** Once the reviewers are satisfied with the changes, they can approve the merge request. If further changes are needed, they can request them.

Merge Request Commands

`git fetch origin `	Fetch the remote branch to your local machine.
`git merge origin/`	Merge the remote branch into your current branch (after fetching).
`git rebase origin/`	Rebase your current branch onto the remote branch (alternative to merging).

GitLab CI/CD

CI/CD Pipeline Configuration

GitLab CI/CD is configured using a `.gitlab-ci.yml` file at the root of your repository. This file defines the stages, jobs, and scripts that make up your pipeline.

Stages: Define the order in which jobs are executed (e.g., build, test, deploy).

Jobs: Define the tasks to be performed in each stage (e.g., compiling code, running tests, deploying to a server).

Scripts: Define the commands to be executed within each job.

Variables: Define environment variables that can be used in your scripts.

Example .gitlab-ci.yml

```yaml
stages:
- build
- test
- deploy

build_job:
stage: build
script:
- echo "Building..."
- ./build.sh

test_job:
stage: test
script:
- echo "Testing..."
- ./test.sh

deploy_job:
stage: deploy
script:
- echo "Deploying..."
- ./deploy.sh
only:
- main
```

CI/CD Variables

`CI_COMMIT_BRANCH`	The branch or tag name for which the pipeline is running.
`CI_COMMIT_SHA`	The commit SHA for which the pipeline is running.
`CI_PROJECT_ID`	The ID of the GitLab project.
`CI_PROJECT_NAME`	The name of the GitLab project.
`CI_PIPELINE_ID`	The ID of the current pipeline.

GitLab Advanced Features

GitLab Pages

GitLab Pages allows you to host static websites directly from your GitLab repository. You can use it to create personal or project websites, documentation, or blogs.

To set up GitLab Pages, you need to create a `.gitlab-ci.yml` file that builds your website and publishes it to the `public` directory.

GitLab will automatically deploy your website to a GitLab Pages domain (e.g., `username.gitlab.io/projectname`).

GitLab Issues

GitLab Issues are used to track bugs, feature requests, and other tasks related to your project. They provide a central place to discuss and manage work.

You can assign issues to team members, set milestones, add labels, and track progress.

Issues can be linked to merge requests to track the code changes that address them.

GitLab Security

GitLab provides various security features to help you identify and address vulnerabilities in your code.

Static Application Security Testing (SAST): Analyzes your source code for potential vulnerabilities.

Dynamic Application Security Testing (DAST): Tests your running application for vulnerabilities.

Dependency Scanning: Identifies vulnerabilities in your project's dependencies.

Container Scanning: Scans your Docker images for vulnerabilities.