**Author: M. A. Bishop**

# DevOps Cheat Sheet

---

## Conventional Commits: Recommended Types

- **feat**: A new feature
- **fix**: A bug fix
- **docs**: Documentation changes
- **style**: Formatting, missing semi-colons, etc.
- **refactor**: A code change that neither fixes a bug nor adds a feature
- **test**: Adding missing tests or correcting existing tests
- **chore**: Changes to the build process or auxiliary tools and libraries

## Git Terminology

- **Commit**: A snapshot of your entire project at a given time, identified by a unique hash
- **Index / Staging Area**: A list of files tracked by the repository and a collection of changes to be recorded in the next commit
- **Hash**: A unique ID (hexadecimal string) generated from a commit's data using the SHA-1 algorithm
- **Repository**: A sequence of commits and the .git directory that tracks all changes
- **Branch**: A separate version of the codebase, which has its own set of commits and allows multiple tracks of work in parallel.
- **Working Tree**: The directory containing the files and directories a developer is actively making changes to. It is one of the two main parts of a repository.
- **Remote Repository**: A copy of the repository located elsewhere that acts as a centralized hub for collaborators to send and receive updates.

## Git Best Practices & Policies

- Commit often and write meaningful commit messages
- Push often to the remote repository
- Rebase often from the main branch, but do not rebase the main branch itself
- Squash multiple commits into one before merging into the main branch
- Never commit generated files or share private SSH keys
- Use a .gitignore file to exclude local logs, credentials, and dependencies from the repository

# GitLab DevOps Professional Tips & Routines

## General:

- **Maximize Automation with CI/CD:** Leverage the platform's Continuous Integration and Continuous Deployment capabilities to automate builds, tests, and deployments, ensuring rapid and consistent software delivery.
- **Prioritize Continuous Testing:** Integrate automated tests (system, regression, UAT) into the pipeline, with the successful completion of one test automatically triggering the next one to maximize efficiency.
- **Implement Infrastructure as Code (IaC):** Use configuration management to provision environments and spin up servers through code, eliminating manual configuration and supporting repeatable, automated deployments.
- **Foster a Blameless Culture:** Adopt a culture of experimentation and shared responsibility, focusing on using rapid, automated feedback to identify and fix issues early rather than assigning individual blame.
- **Monitor Everything for Feedback:** Adhere to the measurement principle by monitoring all critical activities, infrastructure, and application performance to gather data and optimize processes.
- **Structure for Collaboration:** Organize around a single, cross-functional DevOps team that includes development, operations, and support roles for a product, promoting shared knowledge and responsibility.
- **Project Labels:** Standardize the use of project labels for categorization, priority, and automation to streamline filtering and reporting across all repository issues and merge requests.

## CI/CD Tips & Routines

- **Pipeline Optimization:** Regularly review and optimize CI/CD pipeline stages (e.g., caching dependencies, parallelizing tests) to minimize build and deployment times, adhering to the "Fast Feedback" principle.
- **Version Everything:** Treat all elements of the CI/CD configuration, including pipeline definitions and deployment scripts (IaC), as code and store them in Git.
- **Use Feature Flags:** Implement feature flags for all new features and major changes. This allows decoupling deployment from release and enables immediate rollback without a new deployment.
- **Small, Incremental Merges:** Ensure all changes are merged into the main branch frequently, ideally multiple times a day, to minimize merge conflicts and integration pain.
- **Standardized Environments:** Use the CI/CD process to create standardized, ephemeral environments for testing and review (Review Apps/Environments) that closely mirror production.
- **Security Scanning Integration:** Embed security testing (SAST, DAST, dependency scanning) directly into the CI pipeline as mandatory stages before deployment to ensure security is "shifted left."

## GitLab Project Features &Functions

- **Project Labels:** Create, assign, and manage custom labels (e.g., *bug*, *feature*, *priority::high*) to Issues, Merge Requests, and Epics to categorize and organize work. Labels are essential for filtering, reporting, and building powerful project automation.

- **Milestones:** Use Milestones to track progress on a group of Issues and Merge Requests toward a specific target date or goal (like a sprint or a major release). You associate a Milestone with an item by editing its sidebar in the issue/MR view.

- **Issues (Single Source of Truth):** Use Issues to record ideas, plan features, track bugs, and manage tasks. They serve as the single source of truth for discussion and collaboration on work to be completed. Create one by navigating to **Project** -> **Issues** -> **New Issue**.

- **Parent/Child Issues (Linking):** When creating or editing an Issue/Task/Incident, use the **Related Issues** section in the sidebar to define a parent-child relationship. This is typically done to break down a larger **Epic** or **Issue** into smaller, manageable **Tasks**. Use the syntax *related to #<IssueID>* or find the link option to set a hierarchical structure.
    - *blocked by #<IssueID>*, *duplicates #<IssueID>*

---

## Linking Examples

**Example Names:**

- **Epic:** E-100 "Implement Global Search Functionality"
- **Milestone:** M1.3 "Q2 Deployment" (Targeted for the end of Q2)
- **Issues/Tasks:** I-250 "Front-end UI for Search Bar," T-251 "Write search API unit tests," T-252 "Refactor database query service"

**Scenario 1: Epic to Task Breakdown (Parent/Child)**

This structure breaks down the high-level Epic into specific, actionable development work, all targeted for the same release milestone.

- **Epic E-100** (Parent) is targeted for **Milestone M1.3**.
    - **Issue I-250** (Child of E-100) is targeted for **Milestone M1.3**.
        - **Task T-252** (Child of I-250) is targeted for **Milestone M1.3**.

**Scenario 2: Cross-Project Dependency**

This demonstrates dependencies where a core infrastructure change (T-252) must be completed before the front-end work (I-250) can be finalized.

- **Issue I-250** *is blocked by* **Task T-252** (This ensures I-250 cannot proceed until T-252 is done).
- **Task T-251** (Unit Tests) *is related to* **Issue I-250** (A simple reference without enforcing a block).