git

GitLab Flow

GitFlow

A complex but successful system to handle project trough git branching system:
https://nvie.com/posts/a-successful-git-branching-model/

The whole model is bit excessive for a single person or small team work, but is the foundation of **frequent release** practices and **Continuous Integration/Continuous Deployment**.

GitLab Flow is the GitaLab implementation of it:
https://about.gitlab.com/topics/version-control/what-is-gitlab-flow/

GitFlow

GitFlow involves **several types of branches**: feature, release, hotfix, master, and develop branches. Each branch serves a **specific purpose** in the software development process.

Versioning and **tagging** are crucial for tracking the progress of the project.

Feature branches are used to develop **new features** or **enhancements**. **Release branches** prepare the code for the next production release. **Hotfix branches** address **critical issues** in the production environment.

Isolation of features and bug fixes allows for **parallel development**, **improved collaboration**, better version control and **easy tracking** of the **project's progress**, effective release management, making it easier to deploy stable versions.

GitFlow and CI/CD

Continuous Integration (**CI**) is a software development practice where code **changes** are **automatically built**, **tested**, and **integrated** into a shared repository. GitFlow encourages CI by promoting a structured workflow and automated testing.

CI tools like Jenkins, Travis CI, and CircleCI can be integrated with GitFlow for automated testing and validation.

Continuous Deployment (**CD**) is the practice of **automatically deploying** code changes **to production**. CD is closely related to CI, as CI ensures that code is always in a deployable state.

GitFlow supports CD by providing a clear release process and version management.

GitLab Flow

The **GitLab Flow** foresee the creation of at least a **development branch** and a **branch for each environment** being used:

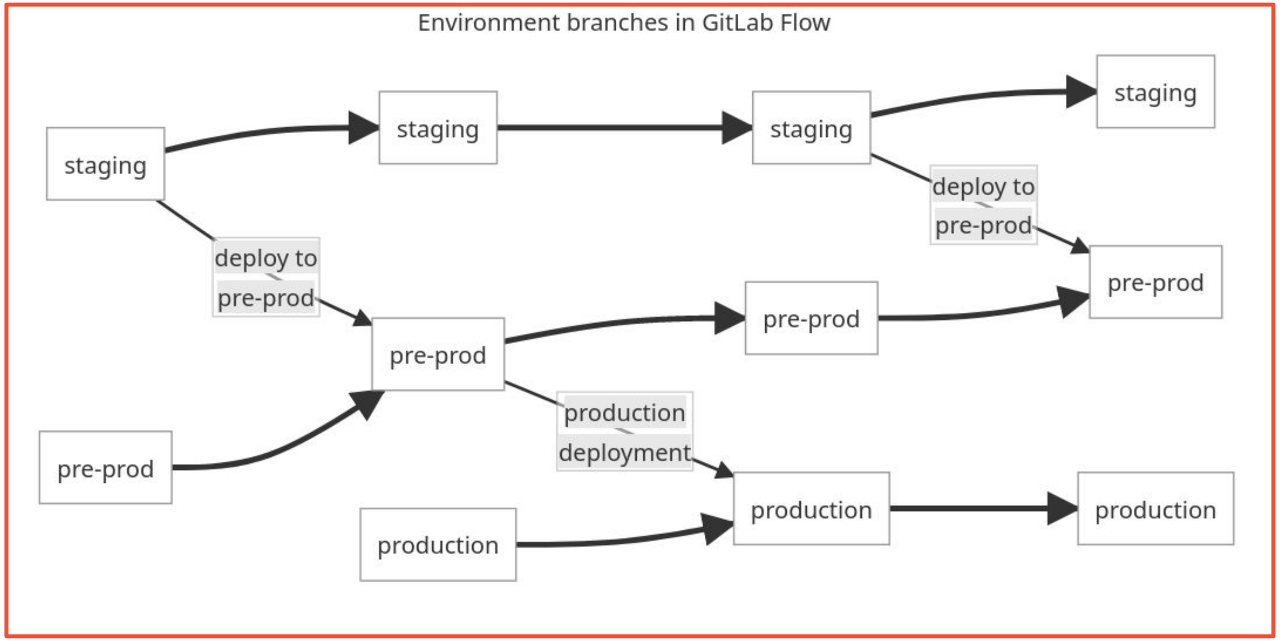
- production/main
- pre-production/beta
- staging/alpha

A merge on one of these branches is a **release** of the respective environment

GitLab Flow

Aside from emergency fix, merge on environment branches should **happen in order**:

- devel → staging/alpha
- staging/alpha → pre-prod/beta
- pre-prod/beta → prod/main



GitLab Flow

1. Flow start from **issues**: it describe the “problem” to solve or the feature to develop. Usually an issue is **assigned** to a person or a team
2. Person or team working on it, **create a specific branch** starting from development one
3. **Development** happens as required by the issue
4. At the end a **pull request(PR)** is opened. PR correspond with sharing, test and revision of person/team’s work. If it goes south start again from step 3. PR may stay in draft state until feedback and approval has been received
5. PR tested and approved is **accepted**: code changes goes on development branch

GitLab Flow

- GitHub, GitLab ecc. offers the possibility to properly assign **rights on available instruments**: protection of important branches and selection of specific person for PR approvals, among others
- **Issues and Pull Request can referrer each other**. In small project possibly a PR for each issue
- **automatic test, CI/CD** can be integrated in the process: i.e. PR cannot be approved if automatic test doesn't pass
- To have a **new release**: a new PR that specify source branch and destination branch (i.e. merge from staging/alpha to pre-prod/beta is a release to pre-prod/beta)

Gitlab Flow

RECAP

Step	**Comandi principali**
Open an **issue** that describe necessary changes	-
Create a **branch** for the issue (from development)	`git switch -c <nome branch>`

Code and make significative commits	`git add <nome file>`
	`git commit`

Pull request	`git push -o merge_request.create -o merge_request.target=develop`
PR approval	-

GitLab CI/CD

GitLab CI/CD

GitLab CI/CD is a robust, built-in solution for automating your software development pipeline

Key benefits: automation, consistency, and efficiency

`.gitlab-ci.yml` is the foundation of GitLab CI/CD flow

In the **.gitlab-ci.yml** file, you can define:

- The tasks you want to complete
- Other configuration files and templates you want to include
- Dependencies and caches
- The commands you want to run in sequence and those you want to run in parallel
- The location to deploy your application to
- Whether you want to run the scripts automatically or trigger any of them manually

Runners (agents that run on physical or virtual machines) run your jobs, usually in a containerized way

https://docs.gitlab.com/ee/ci/

Useful tools

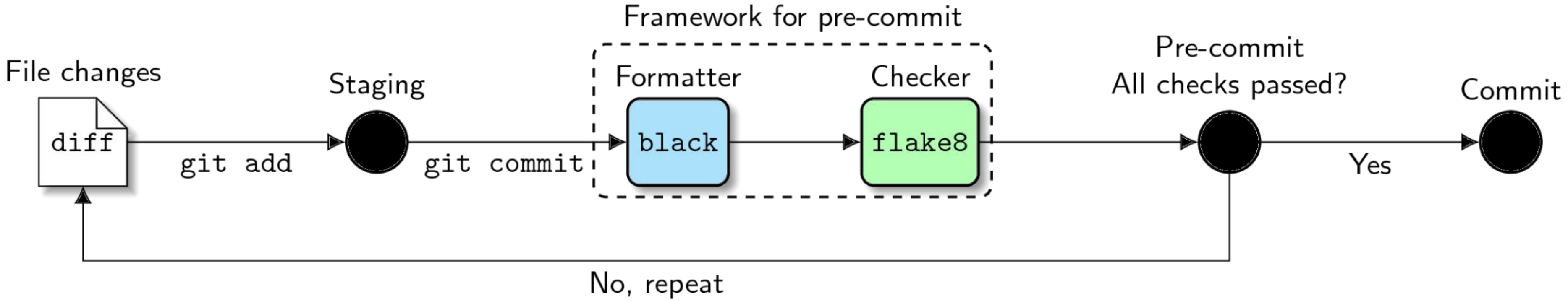
UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Pre-commit

Git offers **hooks** to execute script before and after events like commit and pull

Common practice to have local script being executed **before a commit** take place

Pre-commit (https://pre-commit.com/) is a tool that simplify handling this type of scripts



Script get executed before commit, if they fail commit is not executed

https://ljvmiranda921.github.io/notebook/2018/06/21/precommits-using-black-and-flake8/

SonarQube

SonarQube is an open-source platform designed to continuously inspect and analyze code for code quality, security, and maintainability issues

Key Features:

- Code Quality Analysis: evaluates code quality against industry-standard coding rules and best practices
- Security Vulnerability Detection: identifies security vulnerabilities in your code
- Maintainability Checks: helps ensure code maintainability and readability.
- Code Duplication Detection: finds duplicate code snippets that can be refactored.

https://docs.sonarsource.com/sonarqube/latest/devops-platform-integration/gitlab-integration/

Ansible

Ansible is an open-source automation tool that simplifies the management and orchestration of IT infrastructure

Key Features:

- Agentless: doesn't require agents, making it lightweight and easy to set up
- Infrastructure as Code: uses YAML files to define tasks and configurations
- Idempotent: ensures that tasks can be run multiple times without changing the system's state if it's already compliant
- Extensible: is extensible through custom modules and plugins

https://www.ansible.com/