Teamwork

Join the Git Cult

Romain Eggermont

Summary

- Introduction
- Continuous Integration (CI)
- Going further
  - Work valuation
  - Git Hooks
  - & more

Introduction

- In the three previous courses, you saw how to work with git and its ecosystem

Continuous Integration

Problem: Several developers works together on the same project. How can I assure the code base is correct and viable?

Solution: Continuous Integration (CI)

Main goals: Lint and Test at each step of the project

How does it work?

- Branches usually hooked. When a change is detected, a pipeline of multiple jobs is triggered

- CI tool answers with a status code to inform the git repository manager (GitLab / GitHub)

Tools comparison

Tools	GitLab CI	GitHub Actions	Jenkins	Travis CI
Requirements	None / Runner (for self-hosted)	None	Server (self-hosted only)	$$$
Pricing	Free	Free (Limited)	Free	$$$
Features	++	+	+++	+++

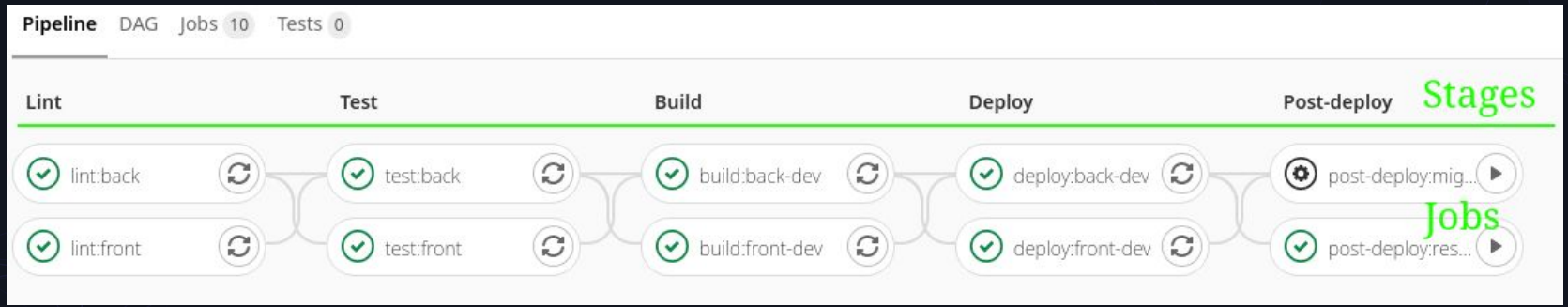
GitLab CI

- Native support for GitLab, file written in YAML
- At each change, GitLab check for a `.gitlab-ci.yml` file, analyze it and run pipeline and jobs related to the changes made
- Based on Docker image: `image: node:lts`
- Test your yaml online at `/-/ci/lint`

```yaml
image: node:lts

stages:
  - lint
  - test

code-lint: # Job n°1 declaration
  stage: lint
  only:
    refs:
      - master
      - merge_requests
  scripts:
    # The script

code-test: # Job n°2 declaration
  stage: test
  only:
    refs:
      - master
      - merge_requests
  scripts:
    # The script

```

Stages and jobs

- Documentation for stages and jobs
- Define specific stages to order, separate and optionally parallelize jobs
- Jobs = single task



GitHub Actions

- Native support for GitHub, file written in YAML
- At each change, GitHub check for a yaml file in .github/workflows/, analyze it and run pipeline and jobs related to the changes made
- CI results can be seen in /actions

```yaml
name: CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Test check 1
        run: echo "Test 1"

      - name: Test check 2
        run: echo "Test 2"

```

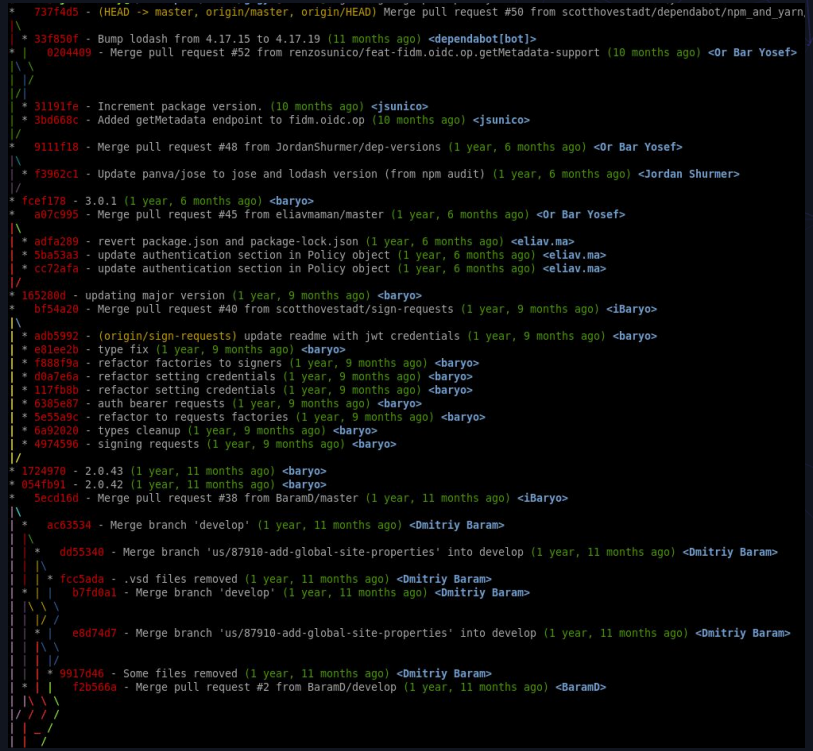
Going further

The following slides are a non-exhaustive list of tips & tricks possible with Git

You can find a big list in the Git book

Graphical log representation

- Following is an example, find the one that fit your tastes
- `git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative`
- Format cheatsheet
- Git Log documentation



Work valuation

- Thanks to Git, you can easily make reports of work made by the team

- Here, we just give two commit hashes to a custom script that generates the stats

```
> ./scripts/release/review_stats.sh 846d29d cf5956a
Number of commits: 195
Summary of changes:  213 files changed, 1367 insertions(+), 127178 deletions(-)
Number or merged PRs: 5
```

Submodules

- Submodules is the package manager for git repositories

- Syntax: git submodule add [-b <branch>] <repository> [<path>]

- Git Submodule documentation

Git Hooks

- Run custom scripts when specific important git actions occur.
- Two types:
  - Client-side hooks: pre-commit, post-commit, pre-push, ...
  - Server-side hooks: pre-receive, post-receive, update, ...
- Skip hooks with `--no-verify` option
- To be edited in the `.git/hooks` folder, where you can find hooks samples
- Standard: Put them in `.githooks` folder to version it and run `git config --local core.hooksPath .githooks/`
- Git Hooks Documentation

Repositories synchronisation

- The goal is to save code at multiple sources (GitHub + GitLab for example)
- Usually works as a post-receive hook, as seen before.
- Can be made manually as a hook, or thanks to GitLab or GitHub
- GitLab mirroring documentation