

CGVR Lab

Software Development in Teams

- As soon as we start work with other people on a software project, we need to consider how to incorporate each-others changes.
- Other wise we can land in so-called merge hell, i.e. trying to merge all changes in the last few days before a deadline with a bunch of merge conflicts.
- Ideally, we always have a working state of our software.
 - Useful for presentations, reverting a bugged version, etc.
- Optimally, human error-prone processes could be automated.

CI / CD (just fyi)

- Continuous Integration
 - The practice of regularly merging to and from your main branch.
 - To catch and fix errors early.
 - via `merge` or `rebase`
 - Usually establish a useful branch structure and merging protocol.
 - `main` / `prod`, `dev`, `feature_...`
 - Changes are validated by automated build & test suites.

CI / CD (just fyi)

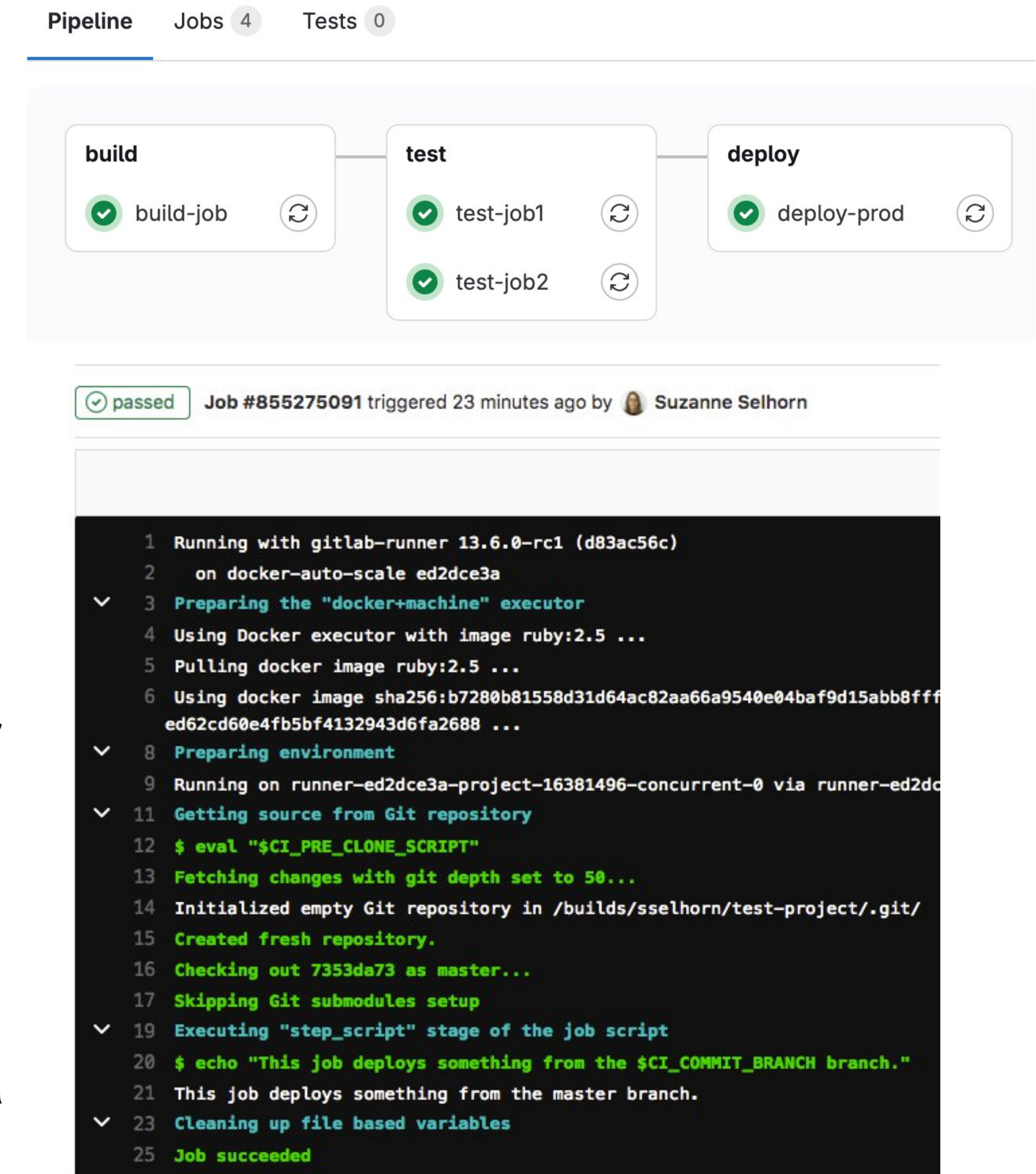
- Continuous Integration
 - The practice of regularly merging to and from your main branch.
 - Usually establish a useful branch structure and merging protocol.
 - Changes are validated by automated build & test suites.
- Continuous Delivery
 - Automating the release process (can otherwise be error-prone), so changes can be deployed via a button click at any time, e.g. daily or weekly.

CI / CD (just fyi)

- Continuous Integration
 - The practice of regularly merging to and from your main branch.
 - Usually establish a useful branch structure and merging protocol.
 - Changes are validated by automated build & test suites.
- Continuous Delivery
 - Automating the release process (can otherwise be error-prone), so changes can be deployed via a button click at any time, e.g. daily or weekly.
- Continuous Deployment
 - Every change that passes the test & build suite, etc. is deployed to customers.

The Pipeline (just fyi)

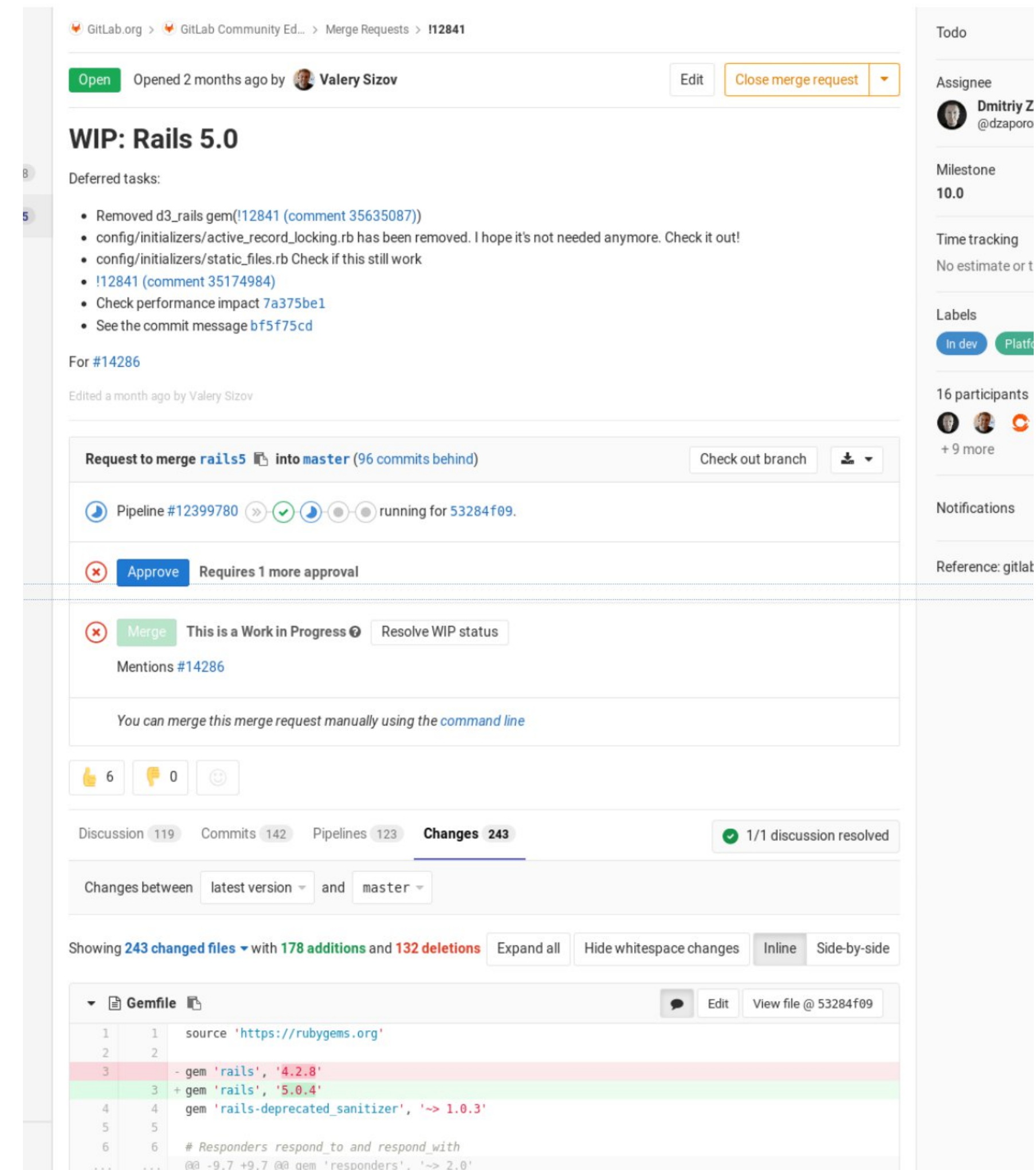
- In larger software projects usually an automated pipeline (for example, with GitLab CI) is set up to do e.g. the following things:
 - Run the projects test-suite after each push / merge to check for broken functionality.
 - Build the project in an isolated environment after each push / merge or on demand. This checks for build errors.
 - Deploying the application automatically, e.g. to a website or app store.



Source: [GitLab Docs](https://docs.gitlab.com/ee/ci/pipelines/)

Merging via Merge / Pull Request

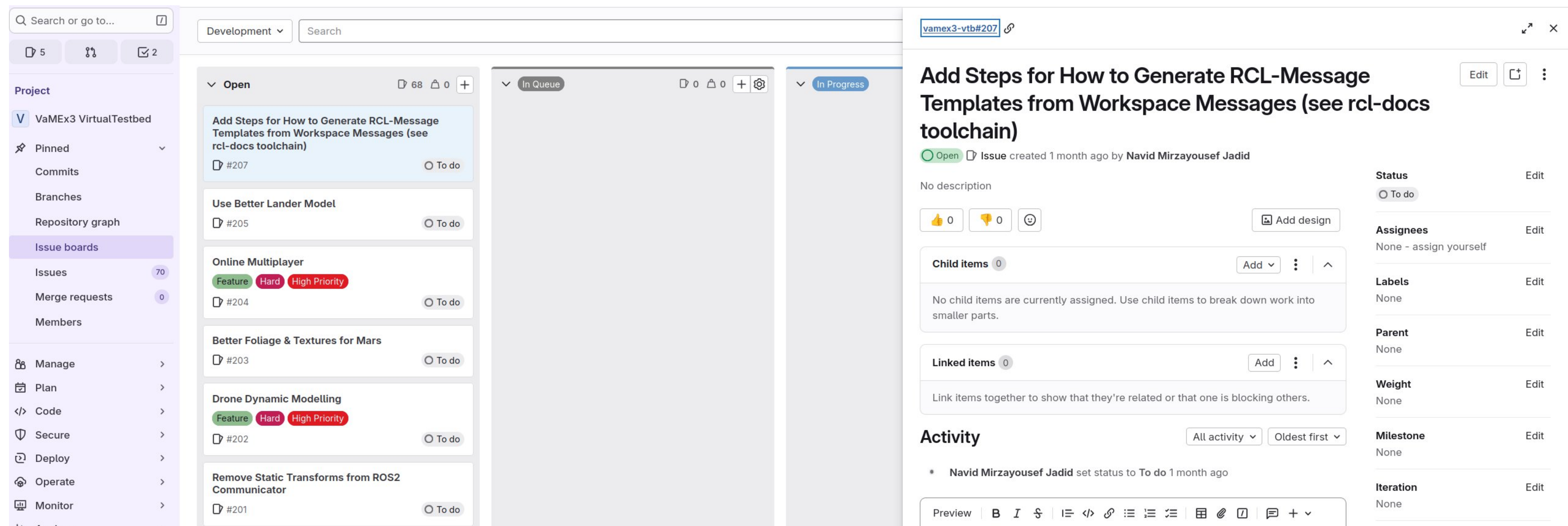
- Merges can be done locally but also online via a [Pull Request](#) (PR).
- Often used when merging changes from a fork of a repository.
- Can also merge branches of the same repository.
- Very common in open-source development but also commercial context (PR + Code Review).
- Changes are reviewed and optionally adjusted before the merge.



Source: [GitLab Docs](#)

Issue Lists / Boards

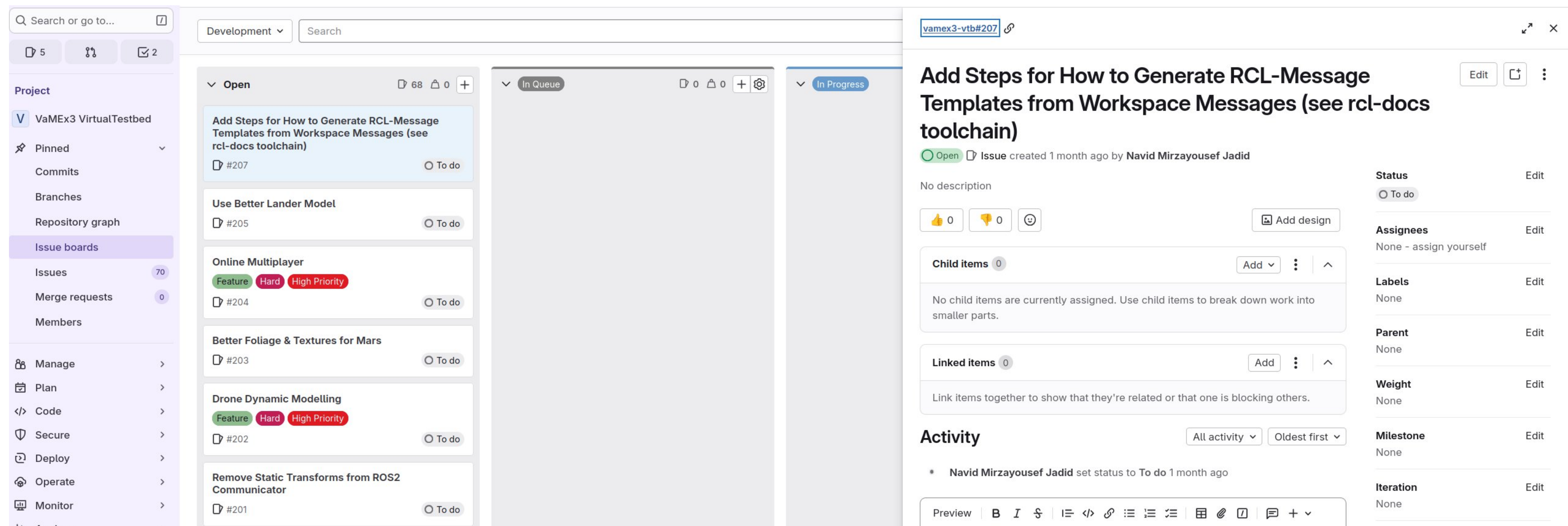
- All git providers offer issue trackers or boards.
- Help you to organize your work and assign tasks / responsibilities:



The screenshot displays a Git issue tracker interface. On the left, a sidebar shows the project structure with options like 'Project', 'Pinned', 'Commits', 'Branches', 'Repository graph', 'Issue boards' (selected), 'Issues' (70), 'Merge requests' (0), and 'Members'. The main area shows a list of issues under the 'Open' tab, including 'Add Steps for How to Generate RCL-Message Templates from Workspace Messages (see rcl-docs toolchain)' (#207), 'Use Better Lander Model' (#205), 'Online Multiplayer' (#204), 'Better Foliage & Textures for Mars' (#203), 'Drone Dynamic Modelling' (#202), and 'Remove Static Transforms from ROS2 Communicator' (#201). The right panel shows a detailed view of issue #207, titled 'Add Steps for How to Generate RCL-Message Templates from Workspace Messages (see rcl-docs toolchain)'. It includes a status of 'Open', a description, a 'Child items' section, a 'Linked items' section, and an 'Activity' section. The right sidebar shows various metadata fields like 'Status', 'Assignees', 'Labels', 'Parent', 'Weight', 'Milestone', and 'Iteration'.

Issue Lists / Boards

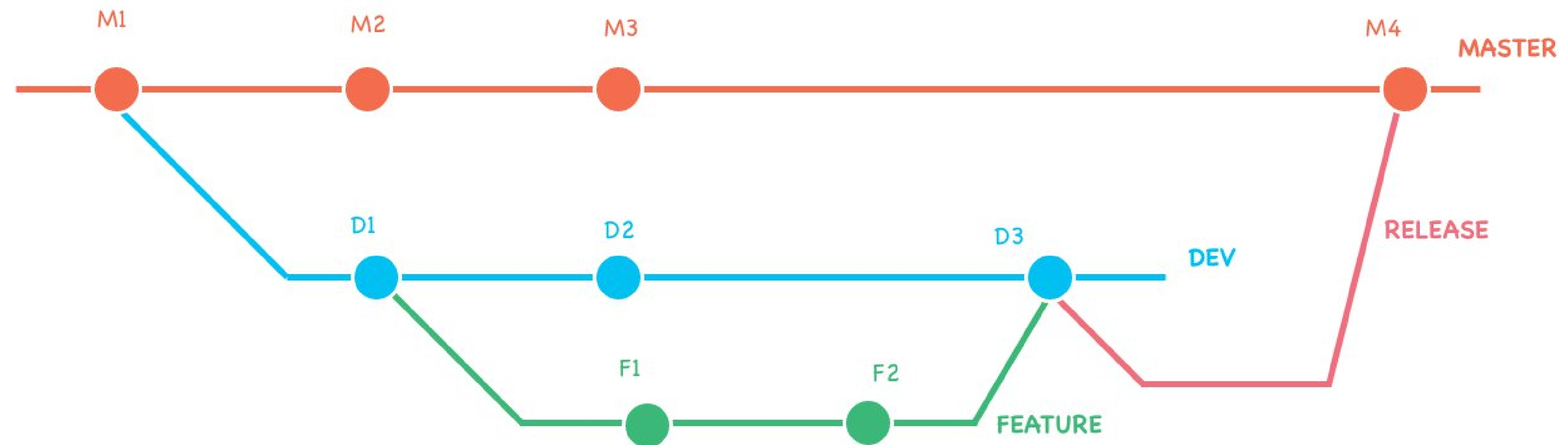
- All git providers offer issue trackers or boards.
- Help you to organize your work and assign tasks / responsibilities:



The screenshot displays a Git issue tracker interface. On the left, a sidebar shows the project structure with options like 'Project', 'Pinned', 'Commits', 'Branches', 'Repository graph', 'Issue boards' (selected), 'Issues' (70), 'Merge requests' (0), and 'Members'. The main area shows a list of issues under the 'Open' tab, including 'Add Steps for How to Generate RCL-Message Templates from Workspace Messages (see rcl-docs toolchain)' (#207), 'Use Better Lander Model' (#205), 'Online Multiplayer' (#204), 'Better Foliage & Textures for Mars' (#203), 'Drone Dynamic Modelling' (#202), and 'Remove Static Transforms from ROS2 Communicator' (#201). The right panel shows a detailed view of issue #207, titled 'Add Steps for How to Generate RCL-Message Templates from Workspace Messages (see rcl-docs toolchain)'. It includes a status of 'Open', a description, a 'Child items' section, a 'Linked items' section, and an 'Activity' section. The right sidebar shows various metadata fields like 'Status', 'Assignees', 'Labels', 'Parent', 'Weight', 'Milestone', and 'Iteration'.

What to do in Your Projects?

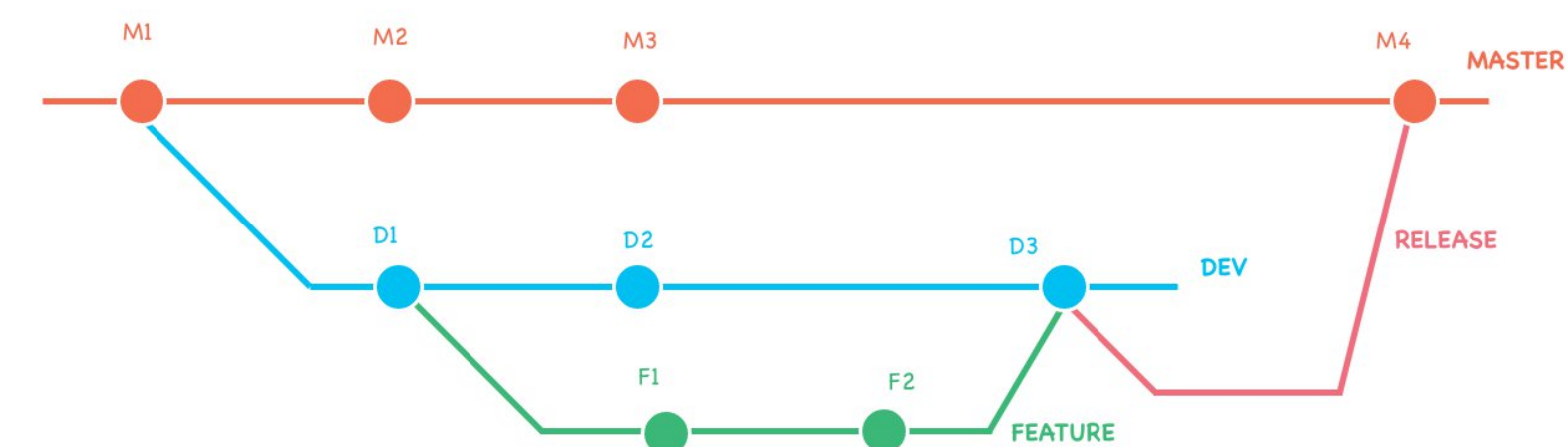
- Use a smart branch structure:



Source: [Substack](#)

What to do in Your Projects?

- Use a smart branch structure:
 - `main` / `prod`: Always in a working state; should be protected.
 - `dev`: Branch to merge your new features into and to branch from for feature branches. Avoid pushing to this branch directly. Should ideally also always be in a working state.
 - If not: We recommend fixing it before continuing.
 - feature branches: Branches where you work on features before merging them.
 - Can be deleted after they have been merged into `dev`.
 - Regularly prune unused / deprecated branches.



Source: [Substack](#)

What to do in Your Projects?

- Use a smart branch structure: `main` / `prod` , `dev`, feature branches, etc.
- Use an issue board for task / issue tracking.

What to do in Your Projects?

- Use a smart branch structure: `main` / `prod` , `dev`, feature branches, etc.
- Use an issue board for task / issue tracking.
- Set up regular meetings (e.g. weekly, bi-weekly) to check on the state of the project and make adjustments accordingly (e.g. assigning to more people to a task or redistributing tasks).
 - Find a good middle ground: Both too many and too few meetings / check ups can hinder the progress of your project.
 - Start early and work incrementally.
 - Be very conservative with time estimates. It is very easy to be overconfident with time estimates.

References & Additional Resources

- Atlassian – Continuous Delivery Principles (<https://www.atlassian.com/continuous-delivery/principles>)
- Atlassian – Continuous Integration vs Delivery vs Deployment (<https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>)
- GitLab - CI/CD Pipelines (<https://docs.gitlab.com/ci/pipelines/>)
- GitLab – Tutorial: Create and Run your First GitLab CI/CD pipeline (https://docs.gitlab.com/ci/quick_start/)
- GitLab CI/CD Pipeline Tutorial for Beginners ([Youtube](#))
- GitLab – Merge Request Docs (https://docs.gitlab.com/user/project/merge_requests/)