GitLab Flow

Jadson Santos - jadsonjs@gmail.com

Last updated: 16/05/2021

Official Documentation

- Introduction to GitLab Flow

Introduction

Git allows you to create several development flows that meet the dynamics of your company.

You can have an enhancement branch, a maintenance branch, a branch that freezes the code to be approved before going into production, while new tasks for the next release can already be done, etc ...

When working as a team, it is important to have a well-defined git flow, to not generate a complex and inefficient flow.

You need to be sure that the team is in agreement on how the flow will be applied.

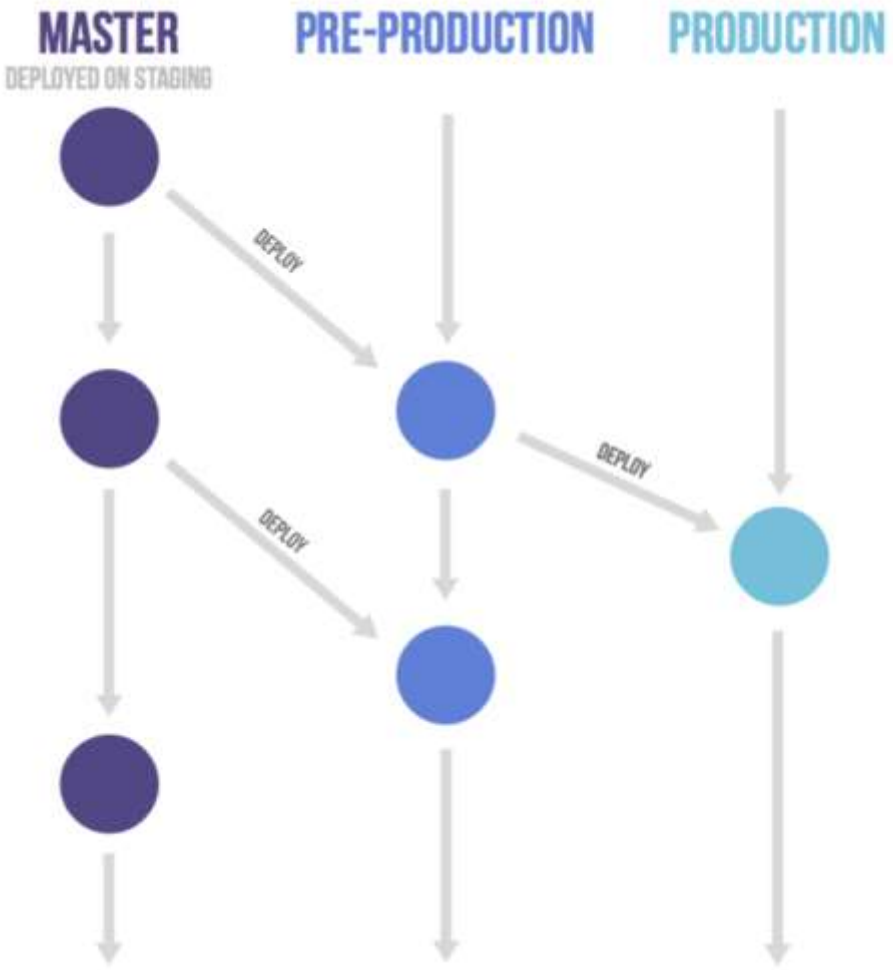
There are several cataloged workflows. There is no the best git flow, there is the one that best suits the reality of your team.

What can be considered a successful workflow?

- Does the workflow adapt to the size of the team?
- Is it easy to undo mistakes with this workflow?
- Does this workflow impose any unnecessary new overhead on the team?

A very popular flow is the Github flow The Github flow is a very simple flow, however, it assumes that you can deploy it in production whenever you merge a feature branch into the master.

GitLab flow tries to solve this problem by creating branches that represent the company's internal environments, there are more possibilities for testing and finding errors. Tests are carried out in all environments until production branch is reached.



GitLab flow has official documentation, but in my opinion, this documentation is just a flow description, it does not show in detail how the flow works, step by step. For this reason, I have created this document. This documentation tries to explain, in my understanding, each step you should execute to follow this flow. For enhancements and hotfix cycles.

Principles

- Create a new local branch for the task and periodically **push a branch of the same name on the server**
- When the task is finished, **request a Merge Request** for the master branch
- When the submitted changes were reviewed/approved, merge them with the master branch
- Once in the master, the code must be **integrated into the company's internal environments** branches, until reaching the production branch.
- When being merged into the master, **delete the branch** where the task was developed leaving the repository more organized.

Advantages

- This flow guarantees a clean state in the branches at any point in the project life cycle.
- It defines how to do Continuous Integration and Continuous Delivery
- It is very flexible according to the team's decisions
- It is less complex than GitFlow Workflow

Disadvantages

- It is more complex than GitHub Workflow
- Git's history becomes unreadable due to the various Merge Requests between branches.

GitLab Flow vs GitHub Flow

The biggest difference between GitLab Flow and GitHub Flow are the branches of the environment in GitLab Flow (for example, pre-production and production) GitHub flow assumes that if you are on the master you can go to production. GitLab flow allows the code to pass through internal environments before it reaches production.

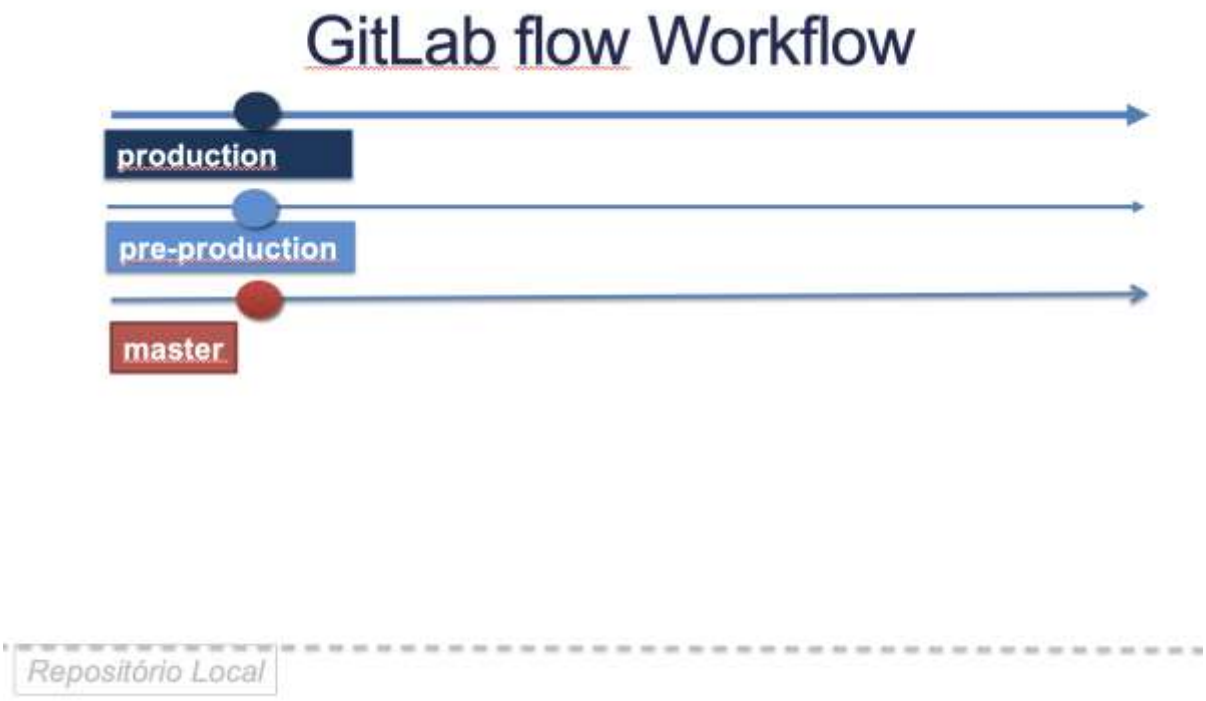
GitLab Flow vs GitFlow

GitLab flow is more favorable to the application of Continuous Integration then Gitfow. The division master/develop makes the Continuous Delivery and the Continuous Integration harder with Gitflow. In GitFlow the creation of branches such as hotfix and release can make integration complex.

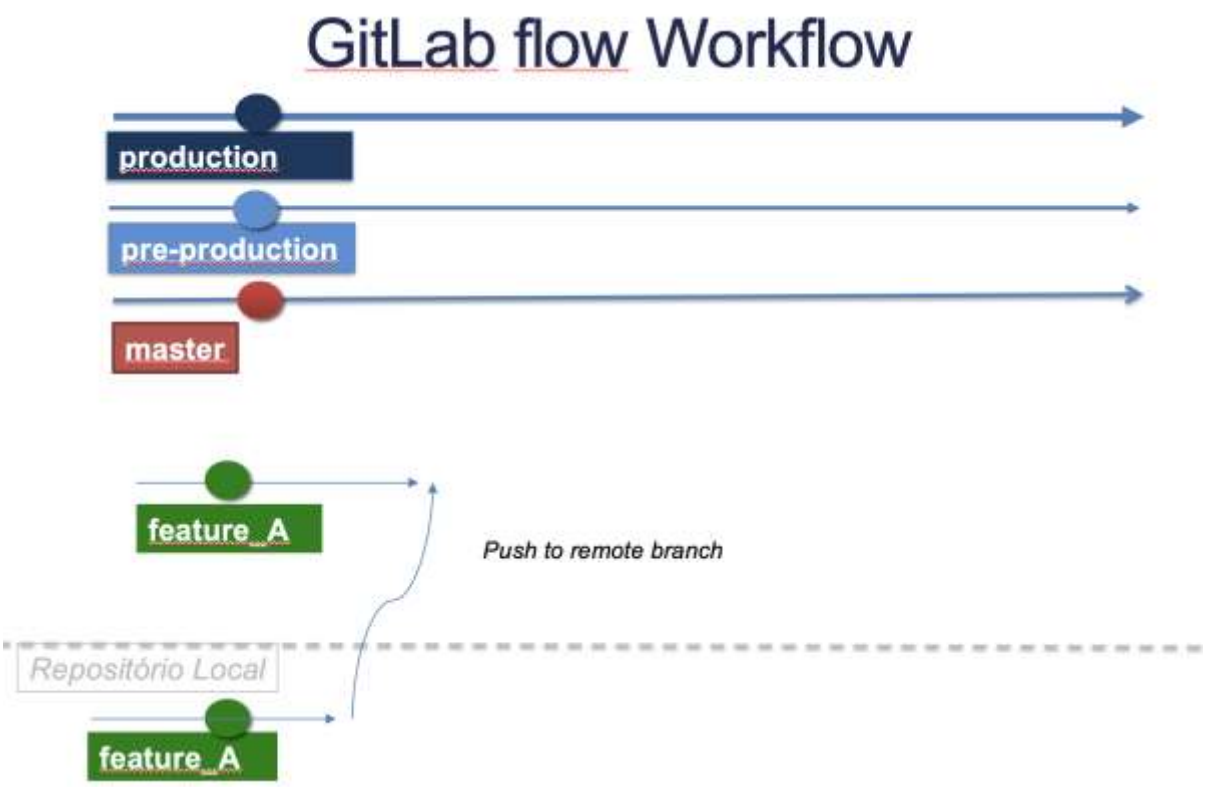
GitLab Flow in practice

Enhancement flow

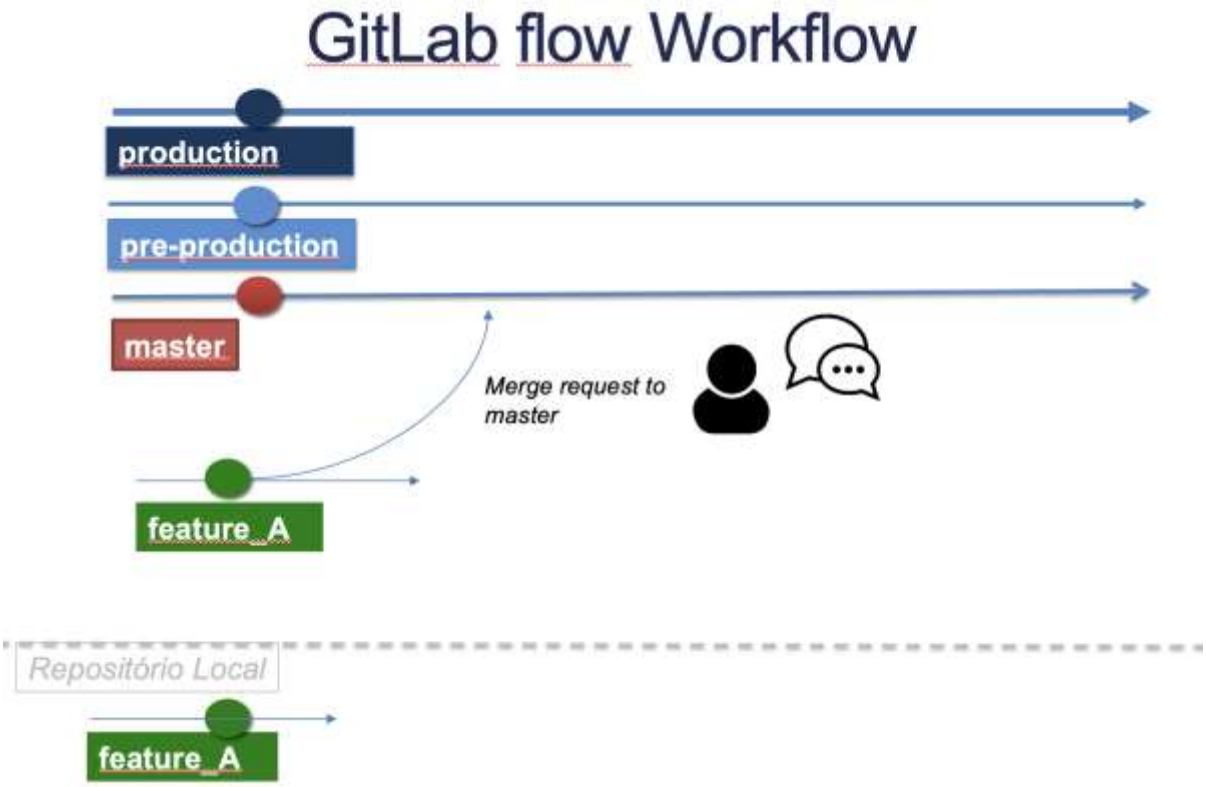
The flow starts with the **master** branch, the **pre-production** environment branch, and the **production** environment branch. All these branches should be protected, for developers do not commit directly to them.



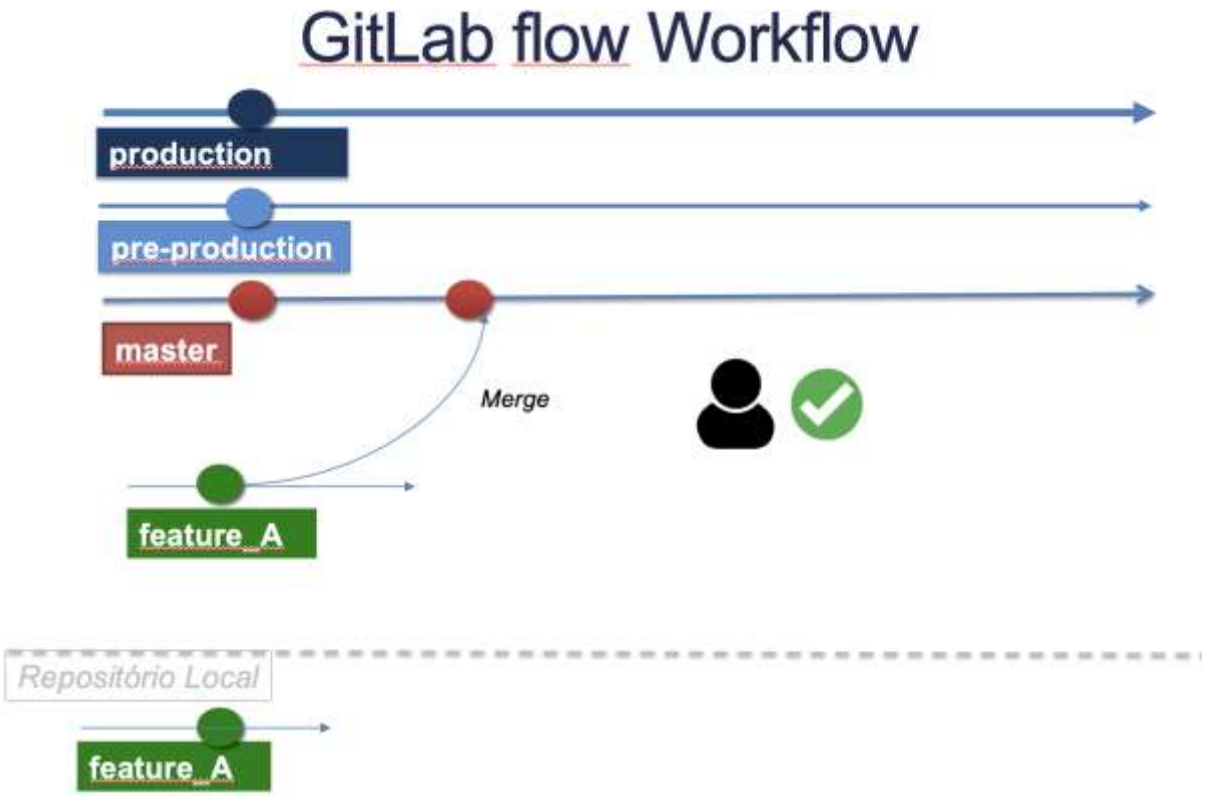
In the same way as in GitHub flow, to start a new development demand, you must create a specific branch for this demand and periodically perform pushes for branch of the same name to the remote repository.



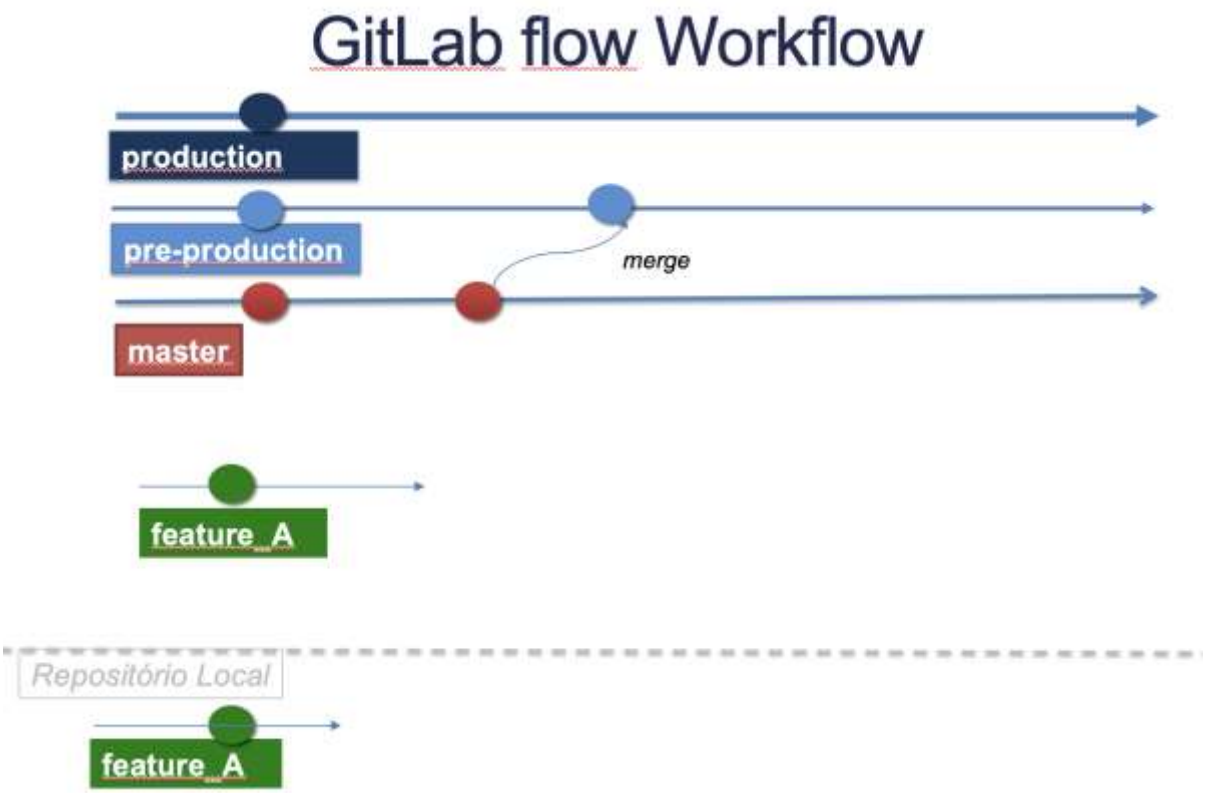
Upon finalizing the demand, a *Merge Request* for the master is requested. A code review can be opened in GitLab and a discussion about the change can be started



When approved, the change must be integrated into the master branch.

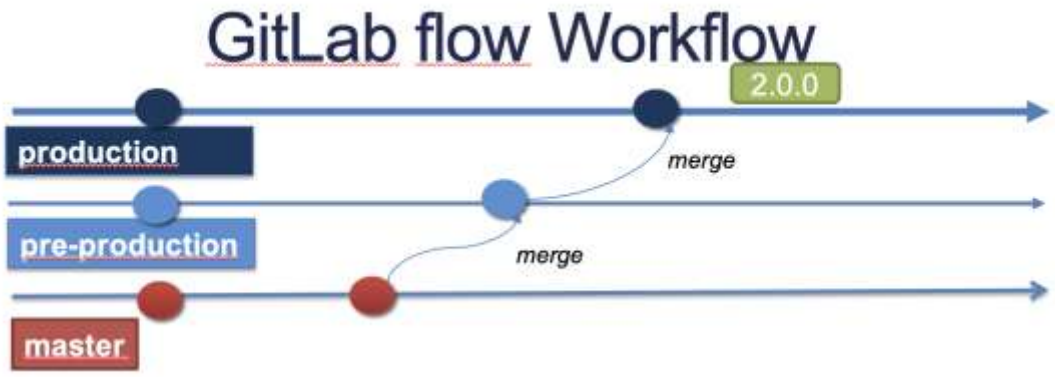


A merge must then be made between the **master** branch and the pre-production environment branch. A pipeline should be executed to build the project and run the automated tests.



When passing in the automated tests, a merge must be done for the production branch. A pipeline should be executed again, to run the automated tests one more time and deploy to production.

A tag must be created to mark a stable version of the system and the feature branch must be removed to make the repository more organized.



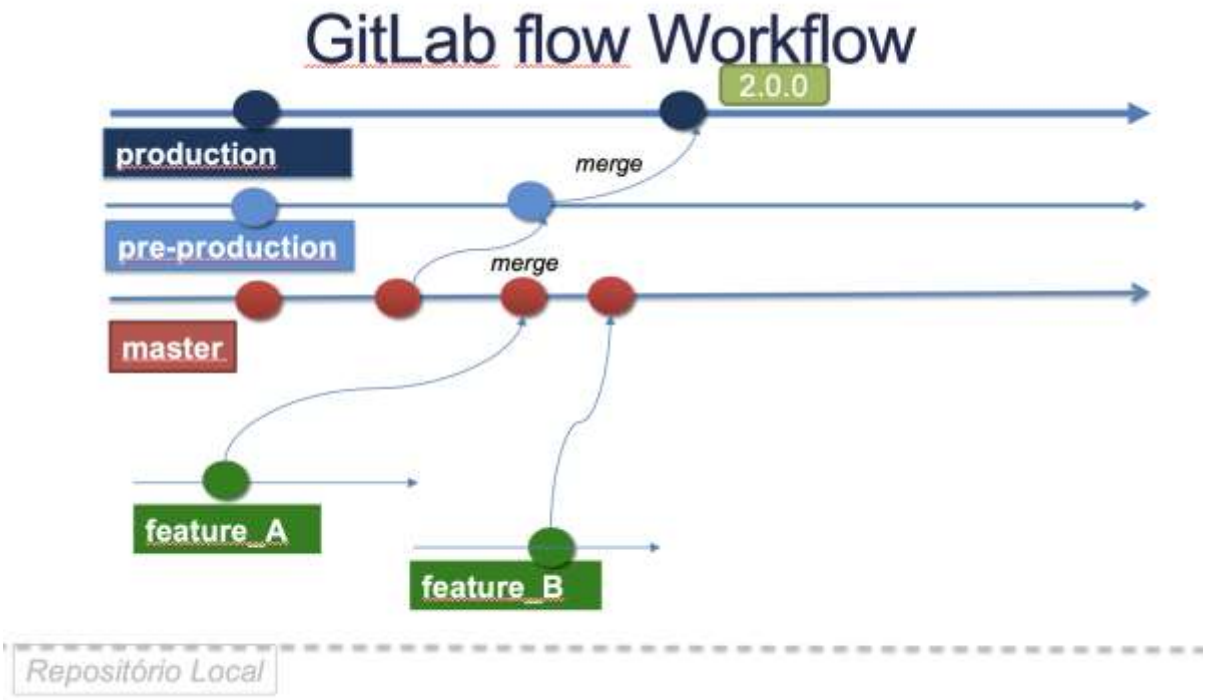
Repositório Local

Hotfix flow

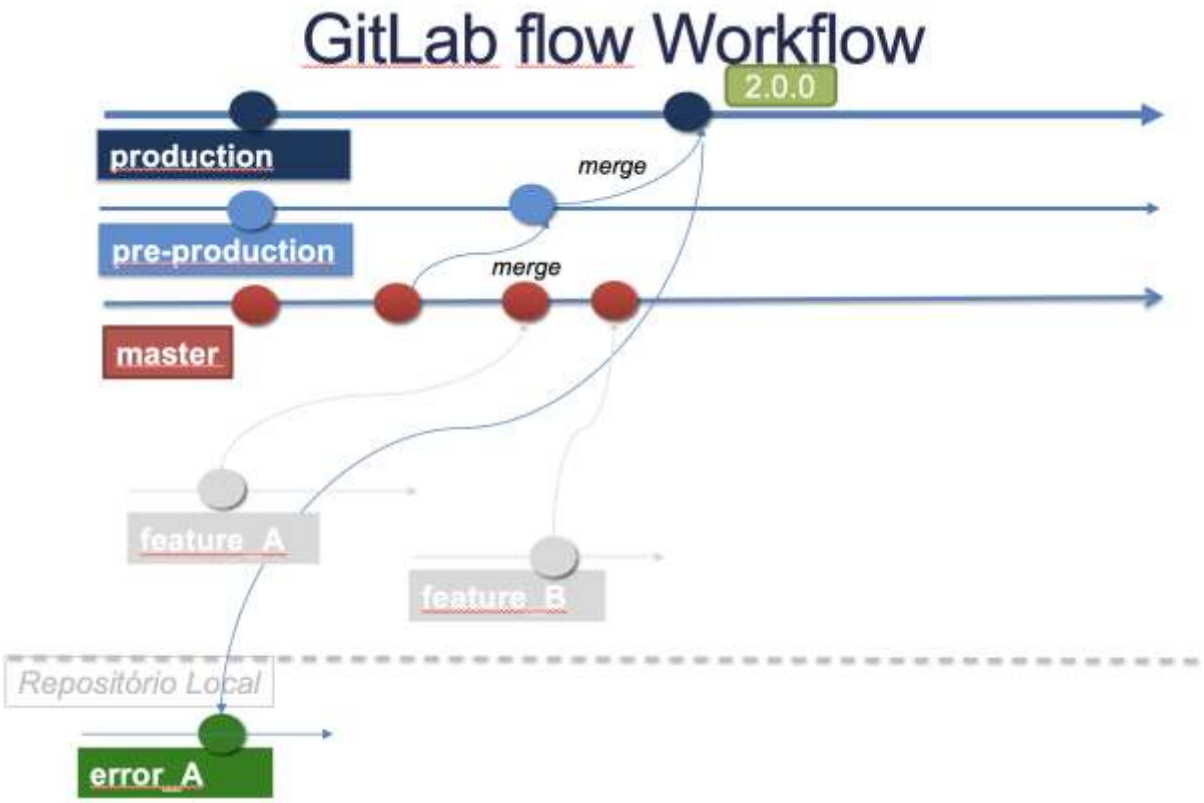
In the event of an urgent change that cannot wait for the master to be merge in production. It is possible to create a branch to realize the urgent task from the production branch. This would be a flow equivalent to the "hotfix" branch of GitFlow.

Usually, these urgent changes are error corrections. And few commits are made.

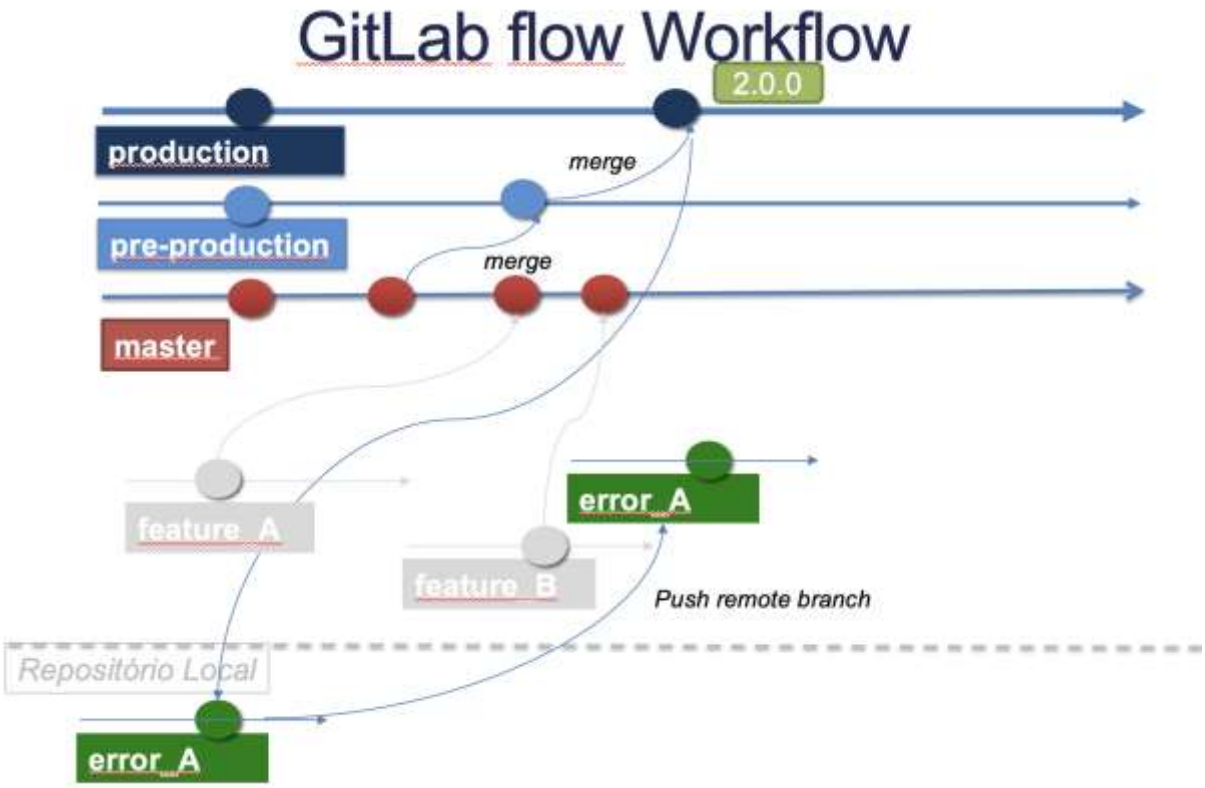
Let's start with a repository where there are 2 tasks committed to the master branch, but which have not yet been made available in production. Suddenly, an error arises to be corrected in production as soon as possible.



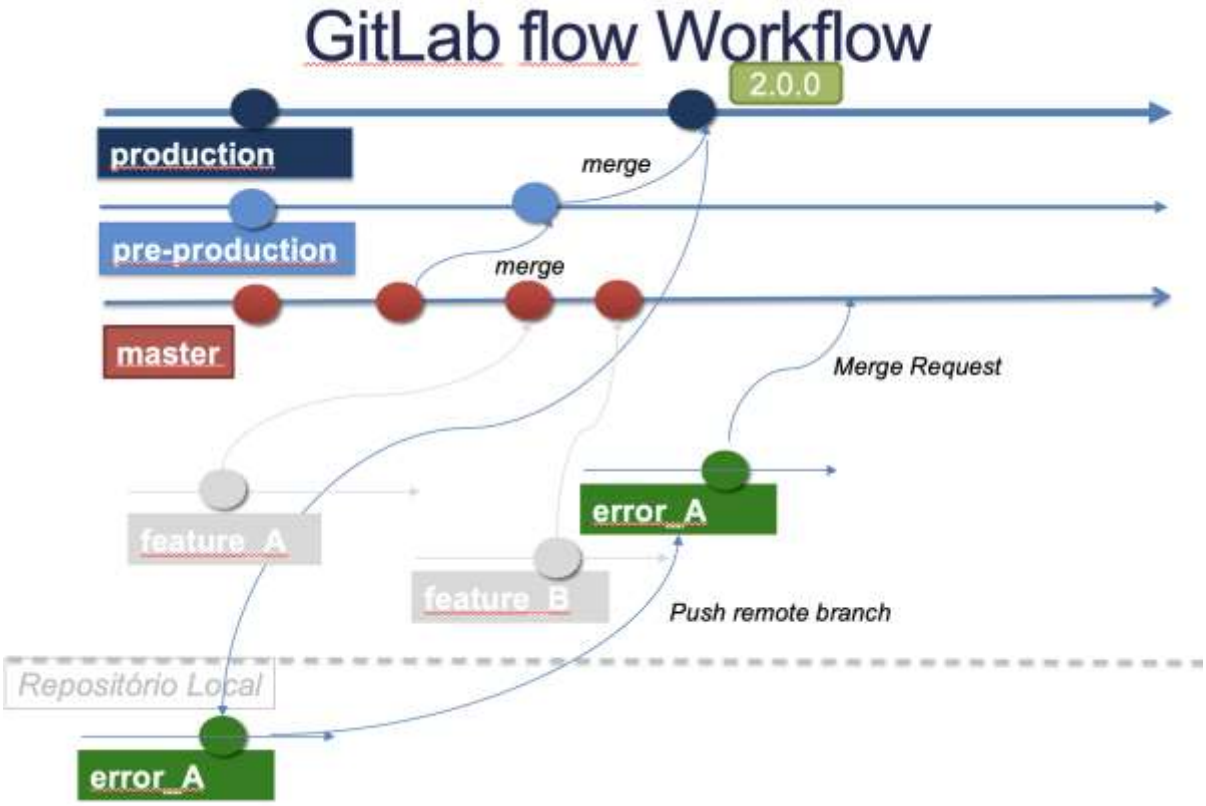
Create an error correction branch from the production branch.



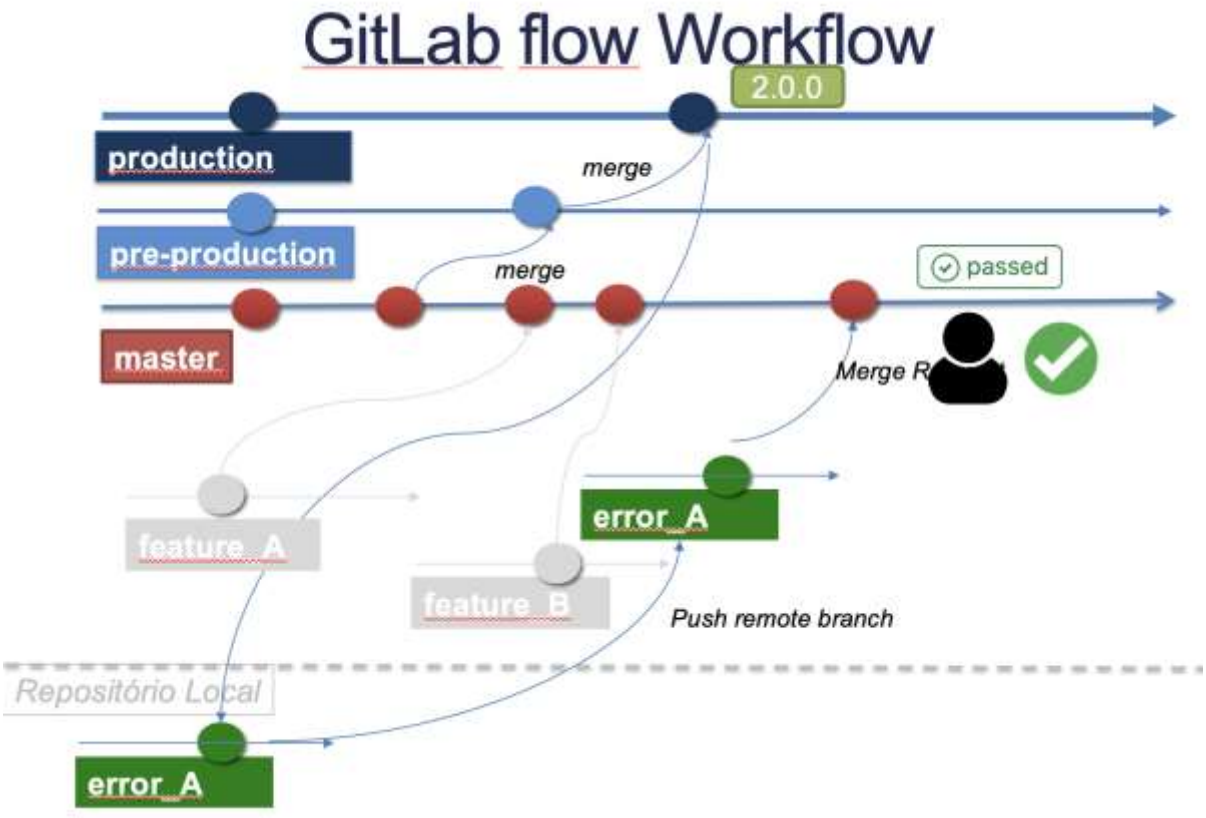
Push the branch to the remote repository.



Request a *Merge Request* for the master branch. Usually, as in the enhancement flow.



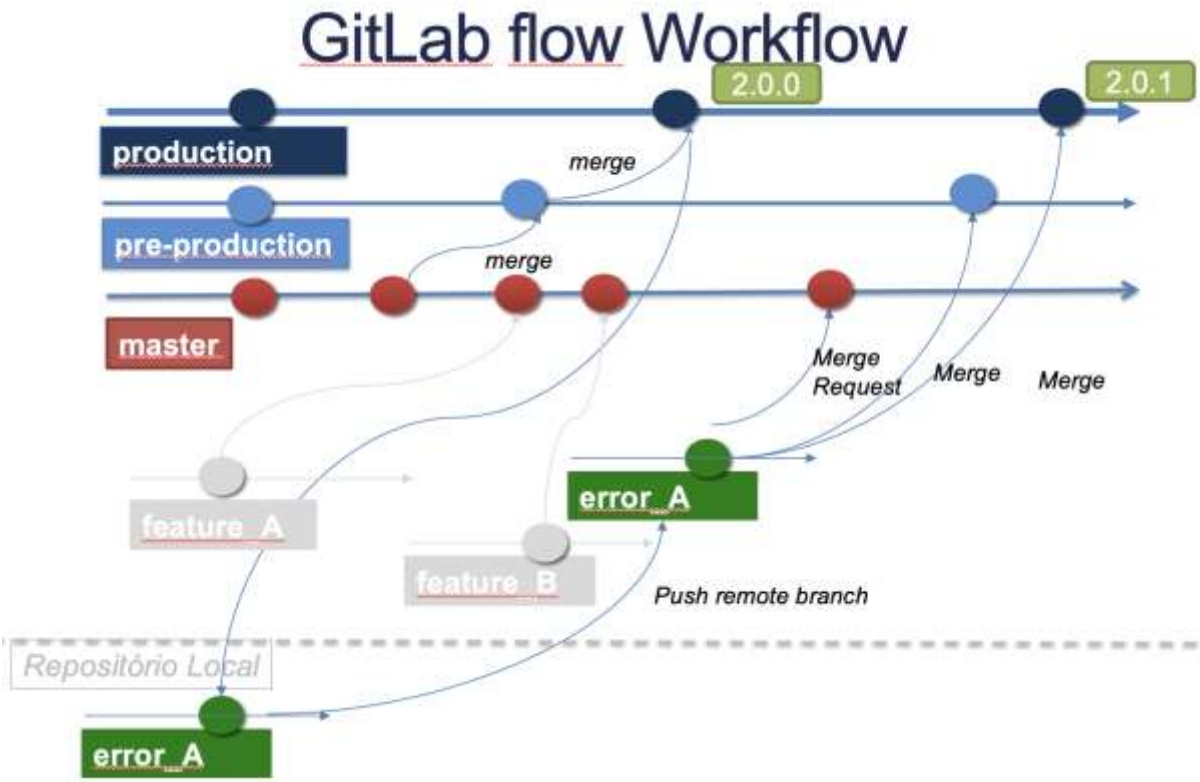
Wait for the *Merge Request* to be approved and the build of the master branch to pass, indicating that there are no errors in your commit.



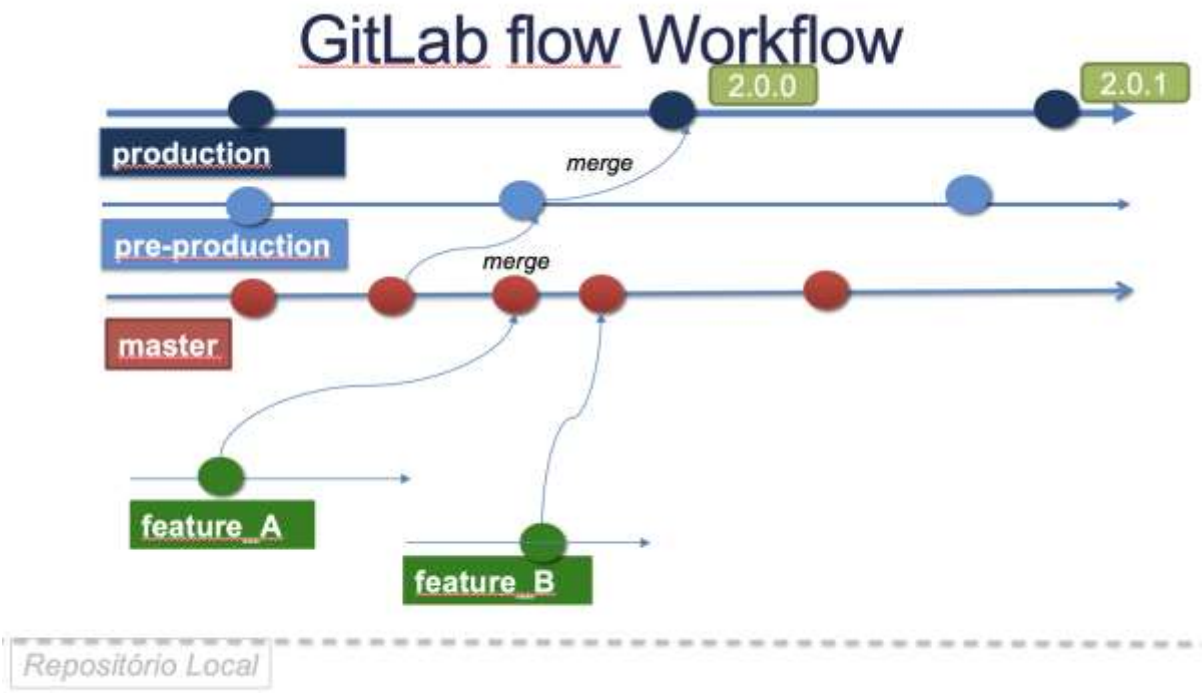
When automatic tests pass in the master branch, due to urgency, you should not wait for the master branch to be merged in other environments branches. Make a *merge / merge request* **from your feature branch** to **all** other branches of internal environments.

As your feature branch is the production branch with your new commit, usually just a unique commit will be merged to the production branch. Make sure to merge these changes to all environments branches. Or the error correction / urgent change can be lost.

Create a tag, to mark a new system version.

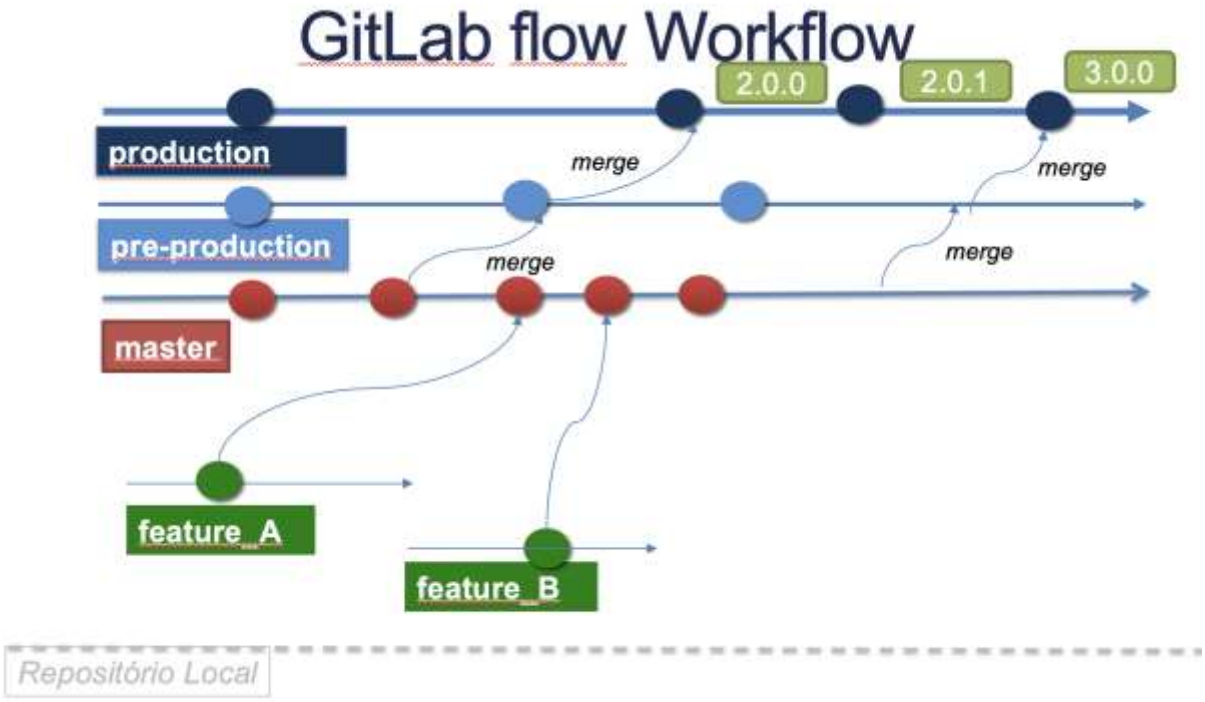


Only now delete the error correction branch. The enhancement branches remain in the repository and your correction is in all environments of the system.

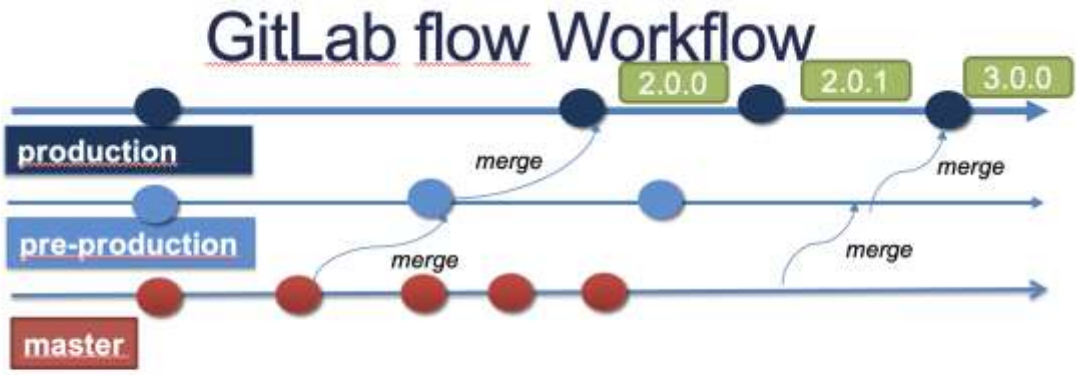


When the rest of the changes are completed, the improvements will be made available in production without erasing the error correction, as it was already integrated into the master branch of the project.

A version tag with improvements is marked.

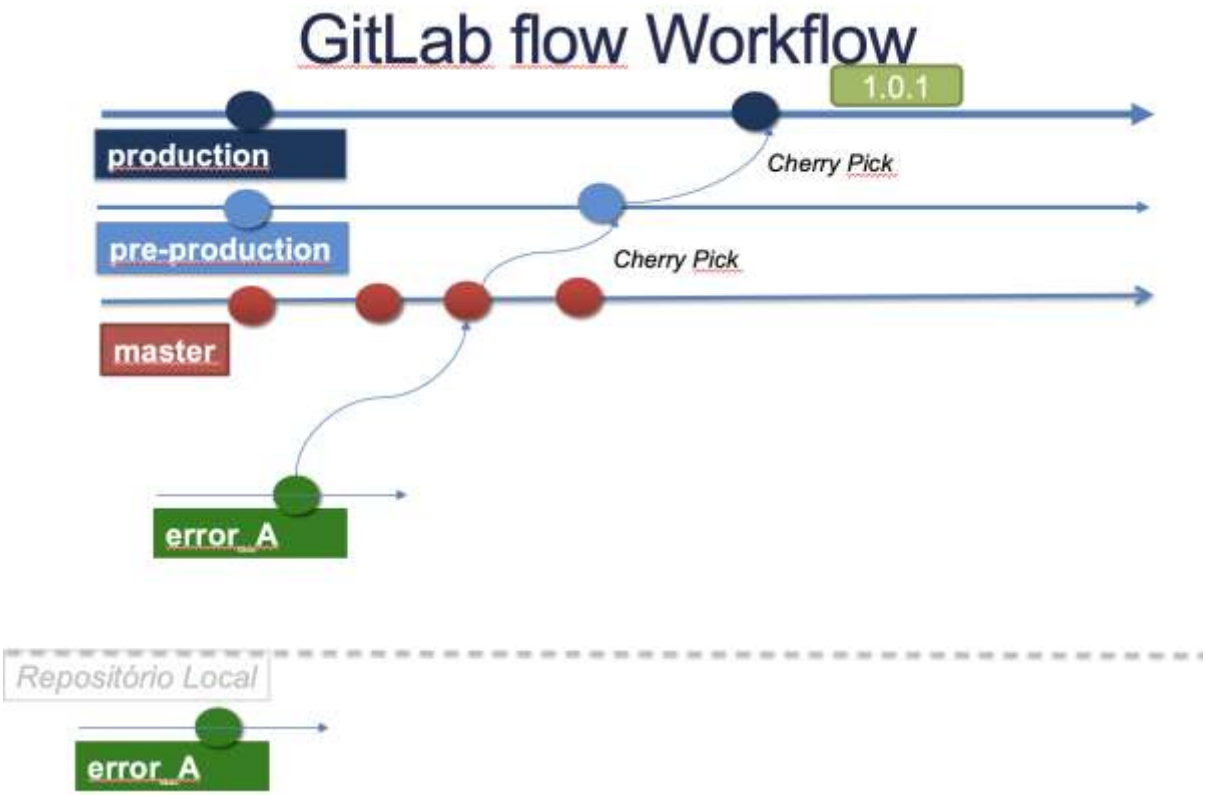


At the end of the cycle, all changes are published in production. All non-permanent branches are deleted. The flow is finished and a new cycle begins.



Repositório Local

Alternatively, for urgent tasks, you can use the **cherry-pick** command to get just a specific commit and send it to the other system environments. So, for example, you can send only one specific commit to production while the others are still waiting for manual tests to finish, in the pre-production branch.



An important thing in this flow is that commits should always be integrated from master to production. Never in the opposite way. As said in the original GitLab flow documentation: **"This workflow, where commits only flow downstream, ensures that everything is tested in all environments. "**

Master -> Pre-Production -> Production (OK)

Production -> Pre-production -> Master (ERROR)

Despite the name, the GitLab flow is not a flow to be applied just on gitlab.com. Can be used with any repository the uses Git.