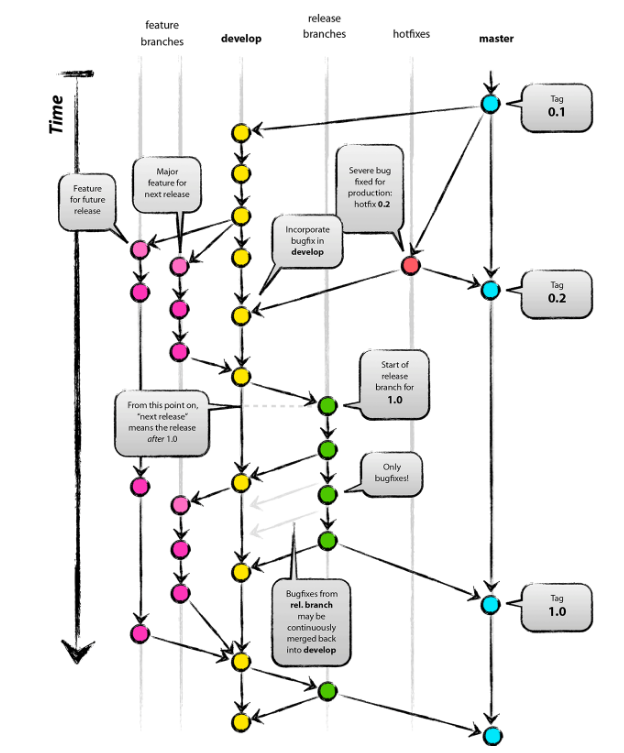
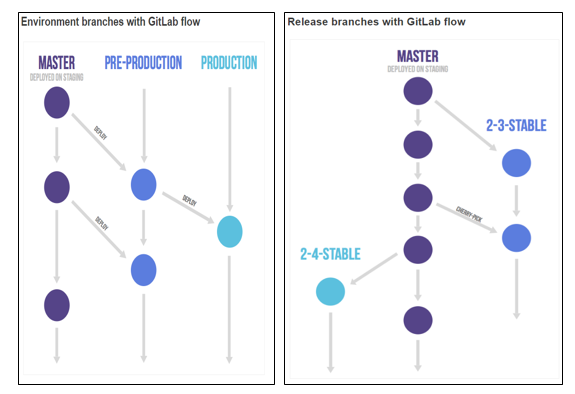
# Gitflow: Branching Strategy



# Gitflow: Branching Explained

* **Master** is considered to be the main branch where the source code of HEAD always reflects a production-ready state.
* **Develop** is considered to be the branch where the source code of HEAD always reflects a state with the latest delivered development changes for the next release. Some would call this the “integration branch”. This is where any automatic nightly builds are built from. When the source code in the develop branch reaches a stable point and is ready to be released, all of the changes should be merged back into master somehow and then tagged with a release number.
* **Feature** branches (or sometimes called topic branches) are used to develop new features for the upcoming or a distant future release. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).
* **Release** branches support preparation of a new production release. The key moment to branch off a new release branch from develop is when develop (almost) reflects the desired state of the new release. At least all features that are targeted for the release-to-be-built must be merged in to develop at this point in time. All features targeted at future releases may not—they must wait until after the release branch is branched off.
* **Hotfix** branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version. The essence is that work of team members (on the develop branch) can continue, while another person is preparing a quick production fix.

# GitLab: Branching strategy

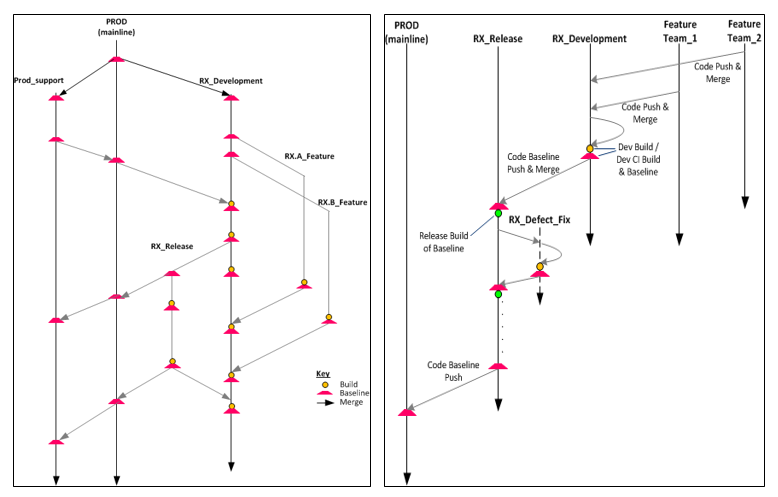


# GitLab flow: Branching Explained

Suppose you have a staging environment, a pre-production environment and a production environment. In this case the master branch is deployed on staging. When someone wants to deploy to pre-production they create a merge request from the master branch to the pre-production branch. And going live with code happens by merging the pre-production branch into the production branch. This workflow where commits only flow downstream ensures that everything has been tested on all environments. If you need to cherry-pick a commit with a hotfix it is common to develop it on a feature branch and merge it into master with a merge request, do not delete the feature branch. If master is good to go (it should be if you are practicing continuous delivery) you then merge it to the other branches.

Only in case you need to release software to the outside world you need to work with release branches. In this case, each branch contains a minor version (2-3-stable, 2-4-stable, etc.). The stable branch uses master as a starting point and is created as late as possible. By branching as late as possible you minimize the time you have to apply bug fixes to multiple branches. After a release branch is announced, only serious bug fixes are included in the release branch. If possible these bug fixes are first merged into master and then cherry-picked into the release branch. This way you can't forget to cherry-pick them into master and encounter the same bug on subsequent releases.

# SVN: Branching Strategy



# SVN: Branching Explained

* **Trunk**: The code in trunk is always maintained as close to release state as possible. Any developer making changes to the trunk must be absolutely certain that his or her part can be coded, tested and is ready to deploy within 2 days (can vary depending on your length of deployment cycle). If it takes more than 2 days, they are not allowed to directly change the code in the trunk. They have to create a branch.
* **Developer** branch: Each developer must create his or her own branch if their code will take more time to program than your normal deployment cycle. It is the responsibility of the developer to regularly (timeframe depends on your development environment) merge changes from trunk to his branch.
* **Feature** branch: Create a feature branch if 2 or more developers are going to work on a new feature that will take considerable time to complete. It is the responsibility of the team lead to merge changes from trunk to this feature branch on a regular basis.

It is always beneficial to merge changes from trunk frequently. Because, after a couple of days conflict between trunk version and branch version can get out of hand and merging will practically be impossible.

* **Release** branch: A Release branch may be used to finalize a release through test and fix.

When the developer branch or feature branch is ready for release, merge changes back from the branch to Release and Release to trunk.

* **Tags**: Tag is similar to branch. When you create a branch simply rename the folder from branch/branches to tag. Use Tags to tag a release whenever a new release version is deployed. This will come in handy if a very minor patch has to be made on the release version when your trunk is temporarily dirty. Ideally, you do not merge from trunk to the tag.