The 11 Rules of GitLab Flow

Version management with Git is an improvement over methods used before Git in just about every way. However, many organizations end up with messy workflows, or overly complex ones. This is particularly a problem for organizations who have transitioned from another version control system.

In this post we're laying out 11 rules for the GitLab Workflow, to help simplify and clean it up. The major benefit of the rules (or so we hope) is that it simplifies the process and produces a more efficient and cleaner outcome.

We think there's always room for improvement, and everything is a draft. As always, **everyone can contribute**! Feedback and opinions are very welcome.

1. Use feature branches, no direct commits on master.

If you're coming over from SVN, for example, you'll be used to a trunk-based workflow. When using Git you should create a **branch** for whatever you're working on, so that you end up doing a code review before you merge.

2. Test all commits, not only ones on master.

Some people set up their CI to only test what has been merged into **master**. This is too late; people should feel confident that **master** always has green tests. It doesn't make sense for people to have to test **master** before they start developing new features, for example. CI isn't expensive, so it makes the best sense to do it this way.

3. Run all the tests on all commits (if your tests run longer than 5 minutes have them run in parallel).

If you're working on a feature branch and you add new commits, run tests then and there. If the tests are taking a long time, try running them in parallel. Do this server-side in merge requests, running the complete test suite. If you have a test suite for development and another that you only run for new versions; it's worthwhile to set up parallel tests and run them all.

4. Perform code reviews before merges into master, not afterwards.

Don't test everything at the end of your week. Do it on the spot, because you'll be more likely to catch things that could cause problems and others will also be working to come up with solutions.

5. Deployments are automatic, based on branches or tags.

If you don't want to deploy **master** every time, you can create a **production branch**; but there's no reason why you should use a script or log in somewhere to do it manually. Have everything automated, or a specific branch that triggers a production deploy.

6. Tags are set by the user, not by CI.

A user sets a **tag** and, based on that, the CI will perform an action. You shouldn't have the CI change the repository. If you need very detailed metrics, you should have a server report detailing new versions.

7. Releases are based on tags.

If you tag something, that creates a new release.

8. Pushed commits are never rebased.

If you push to a public branch you shouldn't rebase it since that makes it hard to follow what you're improving, what the test results were, and it breaks cherrypicking. We sometimes sin against this rule ourselves when we ask a contributor to squash and rebase at the end of a review process to make something easier to revert. But in general the guideline is: code should be clean, history should be realistic.

9. Everyone starts from master, and targets master.

This means you don't have any long branches. You check out **master**, build your feature, create your merge request, and target **master** again. You should do your complete review **before** you merge, and not have any intermediate stages.

10. Fix bugs in master first and release branches second.

If you find a bug, the **worst** thing you can do is fix it in the just-released version, and not fix it in **master**. To avoid it, you always fix forward. Fix it in **master**, then **cherry-pick** it into another patch-release branch.

11. Commit messages reflect intent.

You should not only say what you did, but also why you did it. It's even more useful if you explain why you did this over any other options.

Read more at the GitLab Flow documentation.

Follow @GitLab and stay tuned for the next post!