

## Where to commit






A very important decision is **where to commit?**: LTS? Stable? Dev?

### General principle

The general principle is that everything goes to master (trunk), and once approved, cherry-picked (backported) to still supported branches (where more releases are planned). How far you can backport depends on the nature of the contribution.

### Where to commit

Commit status and order for each open branch:

Name	Currently this is git branch	What is allowed in this branch	Before committing here, first commit to
Dev	<a href="#">master (trunk)</a>  (future 28x)	Functional <a href="#">ENH</a> ancements and new features, <a href="#">FIX</a> es and <a href="#">TR</a> anslations Most development (new features) happens here. New features need to be functional, but don't need to be complete. In theory, should be releasable at any time. This is the place for <a href="#">RE</a> factoring or cosmetic changes. Also: Update language strings . If you commit to master, and after you want to commit to a stable branch, please see how to <a href="#">git cherry-pick</a> .	This is the first place to propose a merge request (MR)
Next Stable	<a href="#">branches/27.x</a> 	Exists only between the time Dev has been branched into the next stable, and the next stable .0 release has been released. Bug fixes and Translations only	Dev
Current Stable	<a href="#">branches/26.x</a> 	Bug fixes and minor safe enhancements	Dev, Next Stable (if any)
Previous Stable	None	Exists only until .1 release of Current Stable has been released. Then EoL or becomes the Previous Stable LTS	Dev, Current Stable
Previous Stable; LTS	<a href="#">branches/24.x</a> 	Minor safe enhancements, fixes and translations backported from Current Stable or Next Stable	Dev, Current Stable or Next Stable
Security fixes only; LTS	<a href="#">branches/21.x</a> 	Security fixes only	Dev, Current Stable or Next Stable, Previous Stable LTS

*The jQuery Sortable Tables feature must be activated for the sort feature to work.*

Legend:

- STS: Standard Term Support

- LTS: Long Term Support

The table above shows which branch is appropriate to commit what type of code. How close we are to the release also has an impact (ex.: don't start a major refactoring just before a release). Please see: [Freeze and Slush](#).

Please be extra careful about backporting changes to the database schema. Please see: [Database Schema Upgrade](#)

Please also see: [Versions](#) and [Git Workflow](#).

Sometimes, shared feature branches can be created for major things that are not stable enough yet, and require multiple developers to collaborate over a long period. These branches will never become a released branch directly.

For everything else, the author of the branch should create a [Merge Request](#) [↗](#) (MR) from his/her personal branch when it's ready (or [even better](#), a draft MR before it's ready).

The commit process (the human part)

Standard process

1. Create a merge request (MR) from your personal fork on GitLab against [master](#) [↗](#).
2. Use GitLab labels to state your intentions on where your commit will go.
  - Add [Tiki GitLab labels](#) [↗](#) such as needsCherryPicksTo26.x, needsCherryPicksTomaster or doNOTBackport as appropriate to the MR. It is each developer's responsibility to make sure these labels are created and removed for their own MR(s).
    - You need developer access to create the labels (Guest or Reporter level is not enough). If you do not yet have developer access, or you are an external contributor, add the info in the MR description and someone will do it for you. Please see [list of developers](#) [↗](#).
  - Once the initial MR is merged (and all pipelines are green), use [Git cherry-pick](#) to create additional MR(s) to cherry-pick into the appropriate branch(es). The **description should link to the initial MR** so we know it was backported.
  - Remove the backport label from the initial MR **once the additional MR(s) have been created** (do not wait for MRs to be merged in).

To see which merged MRs are still missing a cherry pick, use this GitLab query:

[https://gitlab.com/tikiwiki/tiki/-/merge\\_requests?scope=all&state=merged&label\\_name\[\]=needsCherryPicksTo%3A%3A\\*](https://gitlab.com/tikiwiki/tiki/-/merge_requests?scope=all&state=merged&label_name[]=needsCherryPicksTo%3A%3A*) [↗](#)

Alternate process

In some cases, it's more productive for the developer to work against a branch (not master), and later to cherry pick to higher branches all the way to master. In this case, please use the label

"needsCherryPicksTomaster"

[https://gitlab.com/tikiwiki/tiki/-/merge\\_requests?scope=all&state=merged&label\\_name\[\]=needsCherryPicksTo%3A%3Amaster](https://gitlab.com/tikiwiki/tiki/-/merge_requests?scope=all&state=merged&label_name[]=needsCherryPicksTo%3A%3Amaster) [↗](#)

Collaborating

1. Use MRs. Even core developers use them for their own commits (but frequently self merge them). This has a lot of benefits with little overhead:
  1. You can make sure you didn't break the CI (Auto-merge and forget)
  2. If you did break something, whoever notices has a place to discuss the change
  3. The commits on a MR can be grouped using arbitrary criterias (regardless of whether or not you intend to ultimately squash it) so it's least disruptive to your flow. Examples include:
    1. Daily work MR (very useful for seniors fixing a bunch of small independent things without waiting for CI constantly)

2. MR for collaborating with a specific person(s). Essentially micro-topic branches. Be careful not to force-push too much if you collaborate with other people on a MR.
  3. Draft MR containing your own small feature branch so you can get feedback, or force the CI to run.
2. Don't hesitate to put a MR back in Draft (either as the author or the reviewer)
    1. It doesn't mean it's bad. Rather, it's a signal to reviewers not to re-review this until they are asked to, or the MR is out of draft.
  3. When you fixed **something in the code** on a MR, don't just wait! Do something to inform the reviewer(s): (comment, resolve a thread, take the MR out of draft). MRs are rebased frequently, so the mere fact you pushed code isn't enough information for reviewers to notice.
    1. When a reviewer comments or asks you a question, please try to answer quickly (so it's fresh in the reviewer's mind).

## For MR reviewers

### What to review

- Does this make sense in Tiki?
  - Does it duplicate functionality or code?
  - If a new library was added: [How to pick a software library](#)
  - Will it break things for current users?
- Code quality
  - Is the code quality higher than the average in Tiki?
  - Is there some copy-pasted code?
- Does it seem like this was actually tested?
  - As Victor wrote: "[code reviewers usually do code quality review and note specific areas to improve but we are not compilers. You have to test every place of code you touch before committing](#) ↗."

### Risk assessment: Keep context in mind.

1. For an initial MR
  1. How is the code (does it improve the general quality of the code (not is it perfect...)? Is it understandable? Does it seem to you the developer may be unaware of a standardized way to do a similar thing in Tiki, etc.
  2. What are the risks? Aside from the obvious "This may lead to bugs", for a MR to master, important questions are
    - Could this result in data corruption/ambiguity?
    - Could this make the code much harder to refactor in the future?
2. For a MR on a branch
  1. The trade-offs are different. On a branch, change is inherently more risky than on master. The MR has already been reviewed and approved once, so your question is more "What are the risks related to the difference between the two branches".
    1. During a branch stabilization period, the risk is almost inexistant for most changes. The further master has diverged from the branch, the more thought must be put into merging cherry-picks.
  2. During stabilization periods, it is acceptable for more senior developers to cherry pick directly into the "Next Stable" branch, and remove labels as they do so
  3. For the same reason, but for a longer period, one can group multiple unrelated cherry-picks in a single branch MR to backport them.

### How to merge it

1. Rebase the MR
2. Check if the MR should be squashed, and if so, merge the text of the commit message so it represents the whole.

3. Merging without pipelines after a rebase. When you are reviewing and merging multiple MRs in a session, this can save a lot of time if you feel you know it won't break the pipeline. Just make sure it's not the last thing you do in the day, so you still have time to fix the pipeline.

See also: [Tools for Merge Request reviewers](#)

When is this supposed to be released?

See [Version lifecycle](#)

### Definition of "security-only" phase

- The "security-only" of the LTS period is intended for security fixes, but could include a few bug fixes as well.
  - We will review security vulnerabilities reported to the [Security Team](#)
  - Publish a fix or a way to deactivate the feature.
    - If the included code doesn't have a patch for that version
- What if a security vulnerability requires major code changes, that are not suitable for LTS?
  - We'll disable the feature via [System Configuration](#) so you can choose to use it knowing the risks, decide not to use it, or upgrade.
- The documentation at doc.tiki.org is kept up to date for more recent versions, so expect to see there some documentation about features not available in your Tiki.

### Other notes

- If you're working on Cypht Webmail, please see <https://github.com/cypht-org/cypht/wiki/Lifecycle> [↗](#) and [How to upgrade Cypht within Tiki via Composer](#)
- If you must change the English version (but are not changing the meaning and so the translations are still valid, please use [Mass spelling correction](#). If you can't use that, just add to [Pending text corrections](#)
- If we are close to a release, and you have a change with a risk of regression, try to consult the [release manager](#).
- There are some things that are black and white and there are many shades of gray. In case of doubt, ask on the [Dev Mailing List](#)
- <https://trunkbaseddevelopment.com/> [↗](#)

### Related

- [Commit Tags](#)
- [Git Workflow](#)
- [Commit Guidelines](#)
- [Backport guidelines](#)
- [Git cherry-pick](#)

Alias names of this page:

[WhereToCommit](#) | [Where](#)