

Repository Management



- [Package Names](#)
 - [Package Versions](#)
 - [Milestones](#)
 - [Labels](#)
 - [Groups Of Labels](#)
 - [Command](#)
 - [Scope](#)
 - [State](#)
 - [Required Labels](#)
 - [Bug](#)
 - [Upstream Bug](#)
 - [Good First Issue](#)
 - [Commits](#)
 - [Pull Requests](#)
 - [Merge Pre-Conditions](#)
 - [Unmergeable Contributions](#)
 - [Merge Conditions](#)
 - [Releases](#)
 - [WP-CLI Releases](#)
 - [Package Releases](#)
 - [Tips & Tools](#)
 - [GitHub Searches](#)
-

Package Names

Package names for the official `wp-cli` GitHub organization need to be discussed upfront with the [maintainers](#).

Before creating a new repository, it's important to discuss the repository first. This discussion helps establish consensus on what purpose the codebase serves, and where it lives best in the WP-CLI ecosystem.

Package names for WP-CLI commands must end with the `-command` suffix for clarity, and also to distinguish them from the non-command repositories.

Before new repositories are created for commands, a decision needs to be made whether the command deserves its own repository or whether it should be included within an existing repository instead.

Example: `wp-cli/scaffold-package-command`.

Package Versions

Package versions follow the [Semantic Versioning Specification \(SemVer\)](#).

In layman's terms for us, this typically means:

- *Patch releases* (e.g. 1.0.x) are used for bug fixes, documentation changes, small enhancements, etc.
- *Minor releases* (e.g. 1.x.0) are used when a new command is introduced to a package, or a major enhancement is made to an existing command.
- *Major releases* (e.g. x.0.0) are used when changes to the framework change the WP-CLI API in a breaking way.

Milestones

Open milestones usually point to the next version to be released.

As the release notes are built using the milestones, both pull requests and issues that are to be merged need to be set to the upcoming milestone, so that these pull requests will be part of the release notes.

Labels

Labels are an important organizational tool to communicate the state and progress of the work being done. [Committers](#) are required to keep these up-to-date.

Keeping issues and pull requests accurately labeled helps us make sense of issue tracker usage in aggregate and more easily search closed issues and pull requests for past discussion.

Groups Of Labels

Labels can be part of a label group, a concept that is applicable to all packages. The actual labels that are available can depend on the actual package, but the groups always have the same semantic role.

Command

The labels that define what specific command a given issue/pull request is related to are prefixed with `command:`.

Some examples:

- `command:core` - Relates to one or more of the subcommands under the `wp core` parent command.
- `command:cli-check-update` - Relates to the `wp cli check-update` command.
- `command:post-meta-update` - Relates to the `wp post meta update` command.

Scope

The labels that define what scope the current issue/pull request applies to are prefixed with `scope:`.

Used scopes:

- `scope:bootstrap` - Part of the bootstrap process, which loads both WP-CLI as well as WordPress Core.
- `scope:distribution` - Part of the distribution process, where the Phar is being built and releases are produced.
- `scope:documentation` - Part of the handbook, command reference or inline help.
- `scope:framework` - Part of the WP-CLI framework itself, which provides the architecture, API and helper functions to make commands possible.
- `scope:testing` - Part of the unit or functional tests.
- `scope:website` - Part of the website infrastructure on `wp-cli.org` or `make.wordpress.org`.

State

The labels that define what state a given issue/pull request is in are prefixed with `state:`.

Used states:

- `state:unconfirmed` - The bug/problem in the issue could not be replicated yet or might be related to the reporter's environment.
- `state:unsupported` - The issue is outside of the scope of a bug report and cannot be supported on GitHub. The reporter should be pointed towards one of the [support channels](#).

Required Labels

Some labels have a special meaning and/or might be used for automated workflows down the road. These are required across all official packages.

Bug

The `bug` label denotes an issue that is a confirmed bug where the code does not produce the expected result.

Bugs are specific to unexpected behavior of existing functionality. Functionality perceived to be missing is not a bug, but an enhancement.

Upstream Bug

The `upstream-bug` label denotes an issue that is a bug in upstream software (e.g. WordPress core, PHP, etc.) that won't be fixed in WP-CLI.

Good First Issue

The `good-first-issue` label denotes an issue that is a good entry point for new contributors wanting to get their feet wet.

Good first issues are small in scope and don't require extensive technical expertise or historical project knowledge to get started.

Commits

No direct commits are to be done against the default (`main`/`master`) branch of the packages. All code changes need to go through a pull request workflow.

Pull Requests

All code changes go through a pull request workflow.

Every submitted pull request needs to go through a [code review](#), and needs to be approved by at least one of the [committers](#).

Merge Pre-Conditions

Non-trivial pull requests should be preceded by a related issue that defines the problem to solve and allows for discussion of the most appropriate solution before actually writing code.

If a pull request is submitted by one of the [committers](#), the submitter should set the "Reviewers" for that pull request to `wp-cli/committers` if a general code review is needed, or to one or more specific committer profiles if the expertise of a specific person is needed/wanted.

If a pull request is submitted by an external contributor, the [committers](#) should be responsive and provide quick feedback to encourage further contributions.

Unmergeable Contributions

Sometimes, a pull request may not be mergeable, no matter how much additional effort is applied to it (e.g. out of scope, etc.). In these cases, it's best to let the contributor down as softly and firmly as possible, both to encourage future involvement and avoid flame wars.

Make sure to:

1. Thank the contributor for their time and effort.
2. Fully explain the reasoning behind the decision to close the pull request.
3. Link to as much supporting documentation as possible.

If you'd like a template to follow:

“ Thanks _____ for the time you've spent on this pull request.

I'm closing this pull request because _____. To clarify further, _____.

For more details, please see _____ and _____.

Merge Conditions

Apart from needing to be approved, pull requests also need to have their tests in a passing state before they can be merged. When both of these conditions are true, any [committer](#) can merge the

pull request by:

1. Ensuring that all applicable labels have been set.
2. Ensuring that the correct milestone has been set.
3. Ensuring that the branch is deleted after the merge (if applicable).
4. Adapting the title of the pull-request as needed. The ideal pull request title is one that can be copied and pasted into release notes as is.

Once a pull request is merged, the committer should make sure any corresponding issues are also closed and assigned to the correct milestone.

Releases

WP-CLI Releases

WP-CLI releases are scheduled and communicated well in advance by the [maintainers](#).

When the release is ready, it is prepared and executed according to the [release checklist](#) by one of the [maintainers](#).

Package Releases

Package releases happen on an as-needed basis and are executed by the [maintainers](#). A package is tagged for a release when:

- It has a reasonable number of changes that have been sitting for a while (couple weeks or more) and would benefit from landing in the nightly build.
- It has a substantial enhancement (e.g. a new command) that's worth getting into the nightly build for pre-release validation.

Tagging a release should be postponed (for a few to several days) if there's an open pull request that makes sense to include in the release.

Package release notes should include the subject of and a link to each merged pull request. It's not necessary to include minor documentation, coding standard, and test suite changes. Feel free to edit the subject for clarity. See [wp-cli/scaffold-command v1.0.6](#) for an example.

Importantly, tagging a package release is also an opportunity to perform a final review of product quality:

- Each merged pull request makes sense to ship, and has sufficient test coverage and documentation.
- The package's README.md is up-to-date.
- No breaking changes were introduced, unless explicitly intended.

Sometime after a new package version is tagged, the `wp-make-coffee` bot submits a pull request against `wp-cli/wp-cli` with the results of `composer update`. When this pull request is merged, the package release will end up in the WP-CLI nightly build.

Tips & Tools

GitHub Searches

There are a lot of useful ways to search GitHub. These searches can be put into bookmarks or embedded into pages as a link.

Here are some examples:

- [Open issues & pull requests across all WP-CLI repositories, sorted by last updated](#)
- [Merged pull requests across all WP-CLI repositories that have no milestone, sorted by last updated](#)
- [Closed issues & pull requests across all WP-CLI repositories that have no label, sorted by last updated](#)

	#3	
	cocobook-sync 	2025-06-25 18:05:32 UTC
	cocobook-sync 	2026-10-06 22:24:37 UTC