

Node.js collaborator guide

Contents


- [Issues and pull requests](#)
 - [Welcoming first-time contributors](#)
 - [Closing issues and pull requests](#)
 - [Author ready pull requests](#)
 - [Handling own pull requests](#)
 - [Security issues](#)
- [Accepting modifications](#)
 - [Code reviews](#)
 - [Consensus seeking](#)
 - [Waiting for approvals](#)
 - [Testing and CI](#)
 - [Useful Jenkins CI jobs](#)
 - [Starting a Jenkins CI job](#)
 - [Internal vs. public API](#)
 - [Breaking changes](#)
 - [Breaking changes and deprecations](#)
 - [Breaking changes to internal elements](#)
 - [Unintended breaking changes](#)
 - [Reverting commits](#)
 - [Introducing new modules](#)
 - [Additions to Node-API](#)
 - [Deprecations](#)
 - [Involving the TSC](#)
- [Landing pull requests](#)
 - [Using the commit queue GitHub labels](#)
 - [Using `git-node`](#)
 - [Technical HOWTO](#)
 - [Troubleshooting](#)
 - [I made a mistake](#)
 - [Long Term Support](#)
 - [What is LTS?](#)
 - [How are LTS branches managed?](#)
 - [How can I help?](#)
- [Who to CC in the issue tracker](#)

This document explains how collaborators manage the Node.js project. Collaborators should understand the [guidelines for new contributors](#) and the [project governance model](#).

Issues and pull requests

Mind these guidelines, the opinions of other collaborators, and guidance of the [TSC](#). Notify other qualified parties for more input on an issue or a pull request. See [Who to CC in the issue tracker](#).

Welcoming first-time contributors

Always show courtesy to individuals submitting issues and pull requests. Be welcoming to first-time contributors, identified by the GitHub  First-time contributor badge.

For first-time contributors, check if the commit author is the same as the pull request author. This way, once their pull request lands, GitHub will show them as a *Contributor*. Ask if they have configured their git [username](#) and [email](#) to their liking.

Closing issues and pull requests

Collaborators can close any issue or pull request that is not relevant to the future of the Node.js project. Where this is unclear, leave the issue or pull request open for several days to allow for discussion. Where this does not yield evidence that the issue or pull request has relevance, close it. Remember that issues and pull requests can always be re-opened if necessary.

Author ready pull requests

A pull request is *author ready* when:

- There is a CI run in progress or completed.
- There is at least one collaborator approval.
- There are no outstanding review comments.

Please always add the `author ready` label to the pull request in that case. Please always remove it again as soon as the conditions are not met anymore.

Handling own pull requests

When you open a pull request, [start a CI](#) right away. Later, after new code changes or rebasing, start a new CI.

As soon as the pull request is ready to land, please do so. This allows other collaborators to focus on other pull requests. If your pull request is not ready to land but is [author ready](#), add the `author ready` label. If you wish to land the pull request yourself, use the "assign yourself" link to self-assign it.

Managing security issues

Use the process outlined in [SECURITY.md](#) to report security issues. If a user opens a security issue in the public repository:

- Ask the user to submit a report through HackerOne as outlined in [SECURITY.md](#).
- Move the issue to the private repository called [premature-disclosures](#).
- For any related pull requests, create an associated issue in the `premature-disclosures` repository. Add a copy of the patch for the pull request to the issue. Add screenshots of discussion from the pull request to the issue.
- [Open a ticket with GitHub](#) to delete the pull request using Node.js (team) as the account organization.
- Open a new issue in the public repository with the title `FYI - pull request deleted #YYYY`. Include an explanation for the user:

`FYI @xxxx we asked GitHub to delete your pull request while we work on releases in private.`

- Email `tsc@iojs.org` with links to the issues in the `premature-disclosures` repository.

Accepting modifications

Contributors propose modifications to Node.js using GitHub pull requests. This includes modifications proposed by TSC members and other collaborators. A pull request must pass code review and CI before landing into the codebase.

Code reviews

At least two collaborators must approve a pull request before the pull request lands. One collaborator approval is enough if the pull request has been open for more than seven days.

Approving a pull request indicates that the collaborator accepts responsibility for the change.

Approval must be from collaborators who are not authors of the change.

In some cases, it might be necessary to summon a GitHub team to a pull request for review by @-mention. See [Who to CC in the issue tracker](#).

If you are the first collaborator to approve a pull request that has no CI yet, please [start one](#). Please also start a new CI if the pull request creator pushed new code since the last CI run.

Consensus seeking

A pull request can land if it has the needed [approvals](#), [CI](#), [wait time](#) and no [outstanding objections](#). [Breaking changes](#) must receive [TSC review](#) in addition to other requirements. If a pull request meets all requirements except the [wait time](#), please add the [author ready](#) label.

Objections

Collaborators can object to a pull request by using the "Request Changes" GitHub feature. Dissent comments alone don't constitute an objection. Any pull request objection must include a clear reason for that objection, and the objector must remain responsive for further discussion towards consensus about the direction of the pull request. Where possible, provide a set of actionable steps alongside the objection.

If the objection is not clear to others, another collaborator can ask an objecting collaborator to explain their objection or to provide actionable steps to resolve the objection. If the objector is unresponsive for seven days after a collaborator asks for clarification, a collaborator may dismiss the objection.

Pull requests with outstanding objections must remain open until all objections are satisfied. If reaching consensus is not possible, a collaborator can escalate the issue to the TSC by pingging `@nodejs/tsc` and adding the `tsc-agenda` label to the issue.

Helpful resources

- [How to Do Code Reviews Like a Human \(Part One\)](#)
- [How to Do Code Reviews Like a Human \(Part Two\)](#)
- [Code Review Etiquette](#)

Waiting for approvals

Before landing pull requests, allow 48 hours for input from other collaborators. Certain types of pull requests can be fast-tracked and can land after a shorter delay. For example:

- Focused changes that affect only documentation and/or the test suite:
 - `code-and-learn` tasks often fall into this category.
 - `good-first-issue` pull requests might also be suitable.
- Changes that fix regressions:
 - Regressions that break the workflow (red CI or broken compilation).

- Regressions that happen right before a release, or reported soon after.

To propose fast-tracking a pull request, apply the `fast-track` label. Then a GitHub Actions workflow will add a comment that collaborators can upvote.

If someone disagrees with the fast-tracking request, remove the label. Do not fast-track the pull request in that case.

The pull request can be fast-tracked if two collaborators approve the fast-tracking request. To land, the pull request itself still needs two collaborator approvals and a passing CI.

Collaborators can request fast-tracking of pull requests they did not author. In that case only, the request itself is also one fast-track approval. Upvote the comment anyway to avoid any doubt.

Testing and CI

All fixes must have a test case which demonstrates the defect. The test should fail before the change, and pass after the change.

Do not land any pull requests without the necessary passing CI runs. A passing (green) GitHub Actions CI result is required. A passing (green or yellow) [Jenkins CI](#) is also required if the pull request contains changes that will affect the `node` binary. This is because GitHub Actions CI does not cover all the environments supported by Node.js.

► Changes that affect the `node` binary

If there are GitHub Actions CI failures unrelated to the change in the pull request, try the "🔄 Re-run all jobs" button, on the right-hand side of the "Checks" tab.

If there are Jenkins CI failures unrelated to the change in the pull request, try "Resume Build". It is in the left navigation of the relevant `node-test-pull-request` job. It will preserve all the green results from the current job but re-run everything else. Start a fresh CI if more than seven days have elapsed since the original failing CI as the compiled binaries for the Windows and ARM platforms are only kept for seven days.

Useful Jenkins CI jobs

- [node-test-pull-request](#) is the CI job to test pull requests. It runs the `build-ci` and `test-ci` targets on all supported platforms.
- [citgm-smoker](#) uses [CitGM](#) to allow you to run `npm install && npm test` on a large selection of common modules. This is useful to check whether a change will cause breakage in the ecosystem.
- [node-stress-single-test](#) can run a group of tests over and over on a specific platform. Use it to check that the tests are reliable.
- [node-test-commit-v8-linux](#) runs the standard V8 tests. Run it when updating V8 in Node.js or floating new patches on V8.
- [node-test-commit-custom-suites-freestyle](#) enables customization of test suites and parameters. It can execute test suites not used in other CI test runs (such as tests in the `internet` or `pummel` directories). It can also make sure tests pass when provided with a flag not used in other CI test runs (such as `--worker`).

Starting a Jenkins CI job

From the CI Job page, click "Build with Parameters" on the left side.

You generally need to enter only one or both of the following options in the form:

- `GIT_REMOTE_REF` : Change to the remote portion of git refspec. To specify the branch this way, `refs/heads/BRANCH` is used (e.g. for `master` -> `refs/heads/master`). For pull requests, it will look like `refs/pull/PR_NUMBER/head` (e.g. for pull request #42 -> `refs/pull/42/head`).
- `REBASE_ONTO` : Change that to `origin/master` so the pull request gets rebased onto master. This can especially be important for pull requests that have been open a while.

Look at the list of jobs on the left hand side under "Build History" and copy the link to the one you started (which will be the one on top, but click through to make sure it says something like "Started 5 seconds ago" (top right) and "Started by user ...").

Copy/paste the URL for the job into a comment in the pull request. [node-test-pull-request](#) is an exception where the GitHub bot will automatically post for you.

The [node-test-pull-request](#) CI job can be started by adding the `request-ci` label into the pull request. Once this label is added, `github-actions bot` will start the `node-test-pull-request` automatically. If the `github-actions bot` is unable to start the job, it will update the label with `request-ci-failed` .

Internal vs. public API

All functionality in the official Node.js documentation is part of the public API. Any undocumented object, property, method, argument, behavior, or event is internal. There are exceptions to this rule. Node.js users have come to rely on some undocumented behaviors. Collaborators treat many of those undocumented behaviors as public.

All undocumented functionality exposed via `process.binding(...)` is internal.

All undocumented functionality in `lib/internal/**/*.js` is internal. It is public, though, if it is re-exported by code in `lib/*.js` .

Non-exported `Symbol` properties and methods are internal.

Any undocumented object property or method that begins with `_` is internal.

Any native C/C++ APIs/ABIs requiring the `NODE_WANT_INTERNALS` flag are internal.

Sometimes, there is disagreement about whether functionality is internal or public. In those cases, the TSC makes a determination.

For undocumented APIs that are public, open a pull request documenting the API.

Breaking changes

At least two TSC members must approve backward-incompatible changes to the master branch.

Examples of breaking changes include:

- Removal or redefinition of existing API arguments.
- Changing return values.
- Removing or modifying existing properties on an options argument.
- Adding or removing errors.
- Altering expected timing of an event.
- Changing the side effects of using a particular API.

Breaking changes and deprecations

Existing stable public APIs that change in a backward-incompatible way must undergo deprecation. The exceptions to this rule are:

- Adding or removing errors thrown or reported by a public API.
- Changing error messages for errors without error code.
- Altering the timing and non-internal side effects of the public API.
- Changes to errors thrown by dependencies of Node.js, such as V8.
- One-time exceptions granted by the TSC.

For more information, see [Deprecations](#).

Breaking changes to internal elements

Breaking changes to internal elements can occur in semver-patch or semver-minor commits. Take significant care when making and reviewing such changes. Make an effort to determine the potential impact of the change in the ecosystem. Use [Canary in the Goldmine](#) to test such changes. If a change will cause ecosystem breakage, then it is semver-major. Consider providing a Public API in such cases.

Unintended breaking changes

Sometimes, a change intended to be non-breaking turns out to be a breaking change. If such a change lands on the master branch, a collaborator can revert it. As an alternative to reverting, the TSC can apply the semver-major label after-the-fact.

Reverting commits

Revert commits with `git revert <HASH>` or `git revert <FROM>..<TO>`. The generated commit message will not have a subsystem and might violate line length rules. That is OK. Append the reason for the revert and any `Refs` or `Fixes` metadata. Raise a pull request like any other change.

Introducing new modules

Treat commits that introduce new core modules with extra care.

Check if the module's name conflicts with an existing ecosystem module. If it does, choose a different name unless the module owner has agreed in writing to transfer it.

If the new module name is free, register a placeholder in the module registry as soon as possible. Link to the pull request that introduces the new core module in the placeholder's `README`.

For pull requests introducing new core modules:

- Allow at least one week for review.
- Land only after sign-off from at least two TSC members.
- Land with a [Stability Index](#) of Experimental. The module must remain Experimental until a semver-major release.

Additions to Node-API

Node-API provides an ABI-stable API guaranteed for future Node.js versions. Node-API additions call for unusual care and scrutiny. If a change adds to `node_api.h`, `js_native_api.h`, `node_api_types.h`, or `js_native_api_types.h`, consult [the relevant guide](#).

Deprecations

Node.js uses three [Deprecation](#) levels. For all deprecated APIs, the API documentation must state the deprecation status.

- Documentation-Only Deprecation

- A deprecation notice appears in the API documentation.
- There are no functional changes.
- By default, there will be no warnings emitted for such deprecations at runtime.
- Might cause a runtime warning with the `--pending-deprecation` flag or `NODE_PENDING_DEPRECATION` environment variable.
- Runtime Deprecation
 - Emits a warning at runtime on first use of the deprecated API.
 - If used with the `--throw-deprecation` flag, will throw a runtime error.
- End-of-Life
 - The API is no longer subject to the semantic versioning rules.
 - Backward-incompatible changes including complete removal of such APIs can occur at any time.

Apply the `notable change` label to all pull requests that introduce Documentation-Only Deprecations. Such deprecations have no impact on code execution. Thus, they are not breaking changes (`semver-major`).

Runtime Deprecations and End-of-Life APIs (internal or public) are breaking changes (`semver-major`). The TSC can make exceptions, deciding that one of these deprecations is not a breaking change.

Avoid Runtime Deprecations when an alias or a stub/no-op will suffice. An alias or stub will have lower maintenance costs for end users and Node.js core.

All deprecations receive a unique and immutable identifier. Documentation, warnings, and errors use the identifier when referring to the deprecation. The documentation for the deprecation identifier must always remain in the API documentation. This is true even if the deprecation is no longer in use (for example, due to removal of an End-of-Life deprecated API).

A *deprecation cycle* is a major release during which an API has been in one of the three Deprecation levels. Documentation-Only Deprecations can land in a minor release. They can not change to a Runtime Deprecation until the next major release.

No API can change to End-of-Life without going through a Runtime Deprecation cycle. There is no rule that deprecated code must progress to End-of-Life. Documentation-Only and Runtime Deprecations can remain in place for an unlimited duration.

Communicate pending deprecations and associated mitigations with the ecosystem as soon as possible. If possible, do it before the pull request adding the deprecation lands on the master branch.

Use the `notable-change` label on pull requests that add or change the deprecation level of an API.

Involving the TSC

Collaborators can opt to elevate pull requests or issues to the [TSC](#). Do this if a pull request or issue:

- Is labeled `semver-major`, or
- Has a significant impact on the codebase, or
- Is controversial, or
- Is at an impasse among collaborators who are participating in the discussion.

@-mention the `@nodejs/tsc` GitHub team if you want to elevate an issue to the [TSC](#). Do not use the GitHub UI on the right-hand side to assign to `@nodejs/tsc` or request a review from `@nodejs/tsc`.

The TSC serves as the final arbiter where required.

Landing pull requests

1. Avoid landing pull requests that have someone else as an assignee. Authors who wish to land their own pull requests will self-assign them. Sometimes, an author will delegate to someone else. If in doubt, ask the assignee whether it is okay to land.
2. Never use GitHub's green "[Merge pull request](#)" button. Reasons for not using the web interface button:
 - The "Create a merge commit" method will add an unnecessary merge commit.
 - The "Squash and merge" method will add metadata (the pull request #) to the commit title. If more than one author contributes to the pull request, squashing only keeps one author.
 - The "Rebase and merge" method has no way of adding metadata to the commit.
3. Make sure CI is complete and green. If the CI is not green, check for unreliable tests and infrastructure failures. If there are not corresponding issues in the [node](#) or [build](#) repositories, open new issues. Run a new CI any time someone pushes new code to the pull request.
4. Check that the commit message adheres to [commit message guidelines](#).
5. Add all necessary [metadata](#) to commit messages before landing. If you are unsure exactly how to format the commit messages, use the commit log as a reference. See [this commit](#) as an example.

For pull requests from first-time contributors, be [welcoming](#). Also, verify that their git settings are to their liking.

All commits should be self-contained, meaning every commit should pass all tests. This makes it much easier when bisecting to find a breaking change.

Using the commit queue GitHub labels

See the [commit queue guide](#).

Using `git-node`

In most cases, using [the `git-node` command](#) of [node-core-utils](#) is enough to land a pull request. If you discover a problem when using this tool, please file an issue [to the issue tracker](#).

Quick example:

```
$ npm install -g node-core-utils
$ git node land $PRID
```

To use `node-core-utils`, you will need a GitHub access token. If you do not have one, `node-core-utils` will create one for you the first time you use it. To do this, it will ask for your GitHub password and two-factor authentication code. If you wish to create the token yourself in advance, see [the `node-core-utils` guide](#).

Technical HOWTO

Infrequently, it is necessary to manually perform the steps required to land a pull request rather than rely on `git-node`.

► Manual Landing Steps

Troubleshooting

Sometimes, when running `git push upstream master`, you might get an error message like this:

```
To https://github.com/nodejs/node
! [rejected]          master -> master (fetch first)
```



```
error: failed to push some refs to 'https://github.com/nodejs/node'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

That means a commit has landed since your last rebase against `upstream/master`. To fix this, pull with rebase from upstream, run the tests again, and (if the tests pass) push again:

```
git pull upstream master --rebase
make -j4 test
git push upstream master
```

I made a mistake

- Ping a TSC member.
- With `git`, there's a way to override remote trees by force pushing (`git push -f`). This is generally forbidden as it creates conflicts in other people's forks. It is permissible for simpler slip-ups such as typos in commit messages. You are only allowed to force push to any Node.js branch within 10 minutes from your original push. If someone else pushes to the branch or the 10-minute period passes, consider the commit final.
 - Use `--force-with-lease` to reduce the chance of overwriting someone else's change.

Long Term Support

What is LTS?

Long Term Support (LTS) guarantees 30-month support cycles for specific Node.js versions. You can find more information [in the full release plan](#). Once a branch enters LTS, the release plan limits the types of changes permitted in the branch.

How are LTS branches managed?

Each LTS release has a corresponding branch (v10.x, v8.x, etc.). Each also has a corresponding staging branch (v10.x-staging, v8.x-staging, etc.).

Commits that land on master are cherry-picked to each staging branch as appropriate. If a change applies only to the LTS branch, open the pull request against the *staging* branch. Commits from the staging branch land on the LTS branch only when a release is being prepared. They might land on the LTS branch in a different order than they do in staging.

Only members of @nodejs/backporters should land commits onto LTS staging branches.

How can I help?

When you send your pull request, please state if your change is breaking. Also state if you think your patch is a good candidate for backporting. For more information on backporting, please see the [backporting guide](#).

There are several LTS-related labels:

- `lts-watch-` labels are for pull requests to consider for landing in staging branches. For example, `lts-watch-v10.x` would be for a change to consider for the `v10.x-staging` branch.

- `land-on-` are for pull requests that should land in a future v*.x release. For example, `land-on-v10.x` would be for a change to land in Node.js 10.x.

Any collaborator can attach these labels to any pull request/issue. As commits land on the staging branches, the backporter removes the `lts-watch-` label. Likewise, as commits land in an LTS release, the releaser removes the `land-on-` label.

Attach the appropriate `lts-watch-` label to any pull request that might impact an LTS release.

Who to CC in the issue tracker

Subsystem	Maintainers
<code>benchmark/*</code>	@nodejs/benchmarking, @mscdex
<code>doc/*, *.md</code>	@nodejs/documentation
<code>lib/assert</code>	@nodejs/assert
<code>lib/async_hooks</code>	@nodejs/async_hooks for bugs/reviews (+ @nodejs/diagnostics for API)
<code>lib/buffer</code>	@nodejs/buffer
<code>lib/child_process</code>	@nodejs/child_process
<code>lib/cluster</code>	@nodejs/cluster
<code>lib/{crypto,tls,https}</code>	@nodejs/crypto
<code>lib/dgram</code>	@nodejs/dgram
<code>lib/domains</code>	@nodejs/domains
<code>lib/fs, src/{fs,file}</code>	@nodejs/fs
<code>lib/{_}http{*}</code>	@nodejs/http
<code>lib/inspector.js, src/inspector_*</code>	@nodejs/v8-inspector
<code>lib/internal/bootstrap/*</code>	@nodejs/process
<code>lib/internal/url, src/node_url</code>	@nodejs/url
<code>lib/net</code>	@bnoordhuis, @indutny, @nodejs/streams
<code>lib/repl</code>	@nodejs/repl
<code>lib/{_}stream{*}</code>	@nodejs/streams
<code>lib/timers</code>	@nodejs/timers
<code>lib/util</code>	@nodejs/util
<code>lib/zlib</code>	@nodejs/zlib
<code>src/async_wrap.*</code>	@nodejs/async_hooks

src/node_api.*	@nodejs/node-api
src/node_crypto.*	@nodejs/crypto
test/*	@nodejs/testing
tools/node_modules/eslint, .eslintrc	@nodejs/linting
build	@nodejs/build
src/module_wrap.*, lib/internal/modules/*, lib/internal/vm/module.js	@nodejs/modules
GYP	@nodejs/gyp
performance	@nodejs/performance
platform specific	@nodejs/platform- {aix,arm,freebsd,macos,ppc,smartos,s390,windows}
python code	@nodejs/python
upgrading c-ares	@rvagg
upgrading http-parser	@nodejs/http, @nodejs/http2
upgrading libuv	@nodejs/libuv
upgrading npm	@nodejs/npm
upgrading V8	@nodejs/V8, @nodejs/post-mortem
Embedded use or delivery of Node.js	@nodejs/delivery-channels

When things need extra attention, are controversial, or `semver-major` : @nodejs/tsc

If you cannot find who to cc for a file, `git shortlog -n -s <file>` can help.

Labels

General labels

- `confirmed-bug` : Bugs you have verified
- `discuss` : Things that need larger discussion
- `feature request` : Any issue that requests a new feature
- `good first issue` : Issues suitable for newcomers to fix
- `meta` : Governance, policies, procedures, etc.
- `tsc-agenda` : Open issues and pull requests with this label will be added to the Technical Steering Committee meeting agenda

-
- `author-ready` - A pull request is *author ready* when:
 - There is a CI run in progress or completed.
 - There is at least one collaborator approval (or two TSC approvals for semver-major pull requests).
 - There are no outstanding review comments.

Please always add the `author ready` label to pull requests that qualify. Please always remove it again as soon as the conditions are not met anymore, such as if the CI run fails or a new outstanding review comment is posted.

- `semver-{minor,major}`
 - be conservative – that is, if a change has the remote *chance* of breaking something, go for semver-major
 - when adding a semver label, add a comment explaining why you're adding it
 - minor vs. patch: roughly: "does it add a new method / does it add a new section to the docs"
 - major vs. everything else: run last versions tests against this version, if they pass, **probably** minor or patch

LTS/version labels

We use labels to keep track of which branches a commit should land on:

- `dont-land-on-v?.x`
 - For changes that do not apply to a certain release line
 - Also used when the work of backporting a change outweighs the benefits
- `land-on-v?.x`
 - Used by releasers to mark a pull request as scheduled for inclusion in an LTS release
 - Applied to the original pull request for clean cherry-picks, to the backport pull request otherwise
- `backport-requested-v?.x`
 - Used to indicate that a pull request needs a manual backport to a branch in order to land the changes on that branch
 - Typically applied by a releaser when the pull request does not apply cleanly or it breaks the tests after applying
 - Will be replaced by either `dont-land-on-v?.x` or `backported-to-v?.x`
- `backported-to-v?.x`
 - Applied to pull requests for which a backport pull request has been merged
- `lts-watch-v?.x`
 - Applied to pull requests which the Release working group should consider including in an LTS release
 - Does not indicate that any specific action will be taken, but can be effective as messaging to non-collaborators
- `release-agenda`
 - For things that need discussion by the Release working group
 - (for example semver-minor changes that need or should go into an LTS release)
- `v?.x`
 - Automatically applied to changes that do not target `master` but rather the `v?.x-staging` branch

Once a release line enters maintenance mode, the corresponding labels do not need to be attached anymore, as only important bugfixes will be included.

Other labels

- Operating system labels
 - `macos`, `windows`, `smartos`, `aix`
 - No `linux` label because it is the implied default

- Architecture labels

- `arm`, `mips`, `s390`, `ppc`
- No `x86(_64)` label because it is the implied default