Pull requests

- Dependencies
- Setting up your local environment
  - Step 1: Fork
  - Step 2: Branch
- The process of making changes
  - Step 3: Code
  - Step 4: Commit
    - Commit message guidelines
  - Step 5: Rebase
  - Step 6: Test
  - Step 7: Push
  - Step 8: Opening the pull request
  - Step 9: Discuss and update
    - Approval and request changes workflow
  - Step 10: Landing
- Reviewing pull requests
  - Review a bit at a time
  - Be aware of the person behind the code
  - Respect the minimum wait time for comments
  - Abandoned or stalled pull requests
  - Approving a change
  - Accept that there are different opinions about what belongs in Node.js
  - Performance is not everything
  - Continuous integration testing
- Notes
  - Commit squashing
  - Getting approvals for your pull request
  - Waiting until the pull request gets landed
  - Check out the collaborator guide
  - Appendix: subsystems

Dependencies

Node.js has several bundled dependencies in the *deps/* and the *tools/* directories that are not part of the project proper. Changes to files in those directories should be sent to their respective projects. Do not send a patch to Node.js. We cannot accept such patches.

In case of doubt, open an issue in the issue tracker or contact one of the project collaborators.

Node.js has many channels on the OpenJS Foundation Slack. Interesting channels are: #nodejs for general help, questions and discussions. #nodejs-dev for development of Node.js core specifically.

Node.js also has an unofficial IRC channel: #Node.js.

Setting up your local environment

To get started, you will need to have `git` installed locally. Depending on your operating system, there are also a number of other dependencies required. These are detailed in the Building guide.

Depending on your environment you might want to grab IDE specific settings from IDE configs.

Once you have `git` and are sure you have all of the necessary dependencies, it's time to create a fork.

Step 1: Fork

Fork the project on GitHub and clone your fork locally.

```
git clone git@github.com:username/node.git
cd node
git remote add upstream https://github.com/nodejs/node.git
git fetch upstream
```

Configure `git` so that it knows who you are:

```
git config user.name "J. Random User"
git config user.email "j.random.user@example.com"
```

You can use any name/email address you prefer here. We only use the metadata generated by `git` using this configuration for properly attributing your changes to you in the `AUTHORS` file and the changelog.

If you would like for the GitHub UI to link the commit to your account and award you the `Contributor` label after the changes have been merged, make sure this local email is also added to your GitHub email list.

Step 2: Branch

As a best practice to keep your development environment as organized as possible, create local branches to work within. These should also be created directly off of the upstream default branch.

```
git checkout -b my-branch -t upstream/HEAD
```

The process of making changes

Step 3: Code

The vast majority of pull requests opened against the `nodejs/node` repository includes changes to one or more of the following:

- the C/C++ code contained in the `src` directory
- the JavaScript code contained in the `lib` directory
- the documentation in `doc/api`
- tests within the `test` directory.

If you are modifying code, please be sure to run `make lint` (or `vcbuild.bat lint` on Windows) to ensure that the changes follow the Node.js code style guide.

Any documentation you write (including code comments and API documentation) should follow the Style Guide. Code samples included in the API docs will also be checked when running `make lint` (or `vcbuild.bat lint` on Windows). If you are adding to or deprecating an API, add or change the appropriate YAML documentation. Use `REPLACEME` for the version number in the documentation YAML:

```
### `request.method`
<!-- YAML
added: REPLACEME
-->

* {string} The request method.
```

For contributing C++ code, you may want to look at the C++ Style Guide, as well as the README of `src/` for an overview of Node.js C++ internals.

Step 4: Commit

It is a best practice to keep your changes as logically grouped as possible within individual commits. There is no limit to the number of commits any single pull request may have, and many contributors find it easier to review changes that are split across multiple commits.

```
git add my/changed/files
git commit
```

Multiple commits often get squashed when they are landed. See the notes about commit squashing.

Commit message guidelines

A good commit message should describe what changed and why.

1. The first line should:
   - contain a short description of the change (preferably 50 characters or less, and no more than 72 characters)
   - be entirely in lowercase with the exception of proper nouns, acronyms, and the words that refer to code, like function/variable names
   - be prefixed with the name of the changed subsystem and start with an imperative verb. Check the output of `git log --oneline files/you/changed` to find out what subsystems your changes touch.

   Examples:

   - `net: add localAddress and localPort to Socket`
   - `src: fix typos in async_wrap.h`
2. Keep the second line blank.
3. Wrap all other lines at 72 columns (except for long URLs).
4. If your patch fixes an open issue, you can add a reference to it at the end of the log. Use the `Fixes:` prefix and the full issue URL. For other references use `Refs:`.

   Examples:

   - `Fixes: https://github.com/nodejs/node/issues/1337`
   - `Refs: https://eslint.org/docs/rules/space-in-parens.html`
   - `Refs: https://github.com/nodejs/node/pull/3615`

5. If your commit introduces a breaking change (`semver-major`), it should contain an explanation about the reason of the breaking change, which situation would trigger the breaking change and what is the exact change.

Sample complete commit message:

```
subsystem: explain the commit in one line

The body of the commit message should be one or more paragraphs, explaining
things in more detail. Please word-wrap to keep columns to 72 characters or
less.

Fixes: https://github.com/nodejs/node/issues/1337
Refs: https://eslint.org/docs/rules/space-in-parens.html
```

If you are new to contributing to Node.js, please try to do your best at conforming to these guidelines, but do not worry if you get something wrong. One of the existing contributors will help get things situated and the contributor landing the pull request will ensure that everything follows the project guidelines.

Step 5: Rebase

As a best practice, once you have committed your changes, it is a good idea to use `git rebase` (not `git merge`) to synchronize your work with the main repository.

```
git fetch upstream HEAD
git rebase FETCH_HEAD
```

This ensures that your working branch has the latest changes from `nodejs/node` .

Step 6: Test

Bug fixes and features should always come with tests. A guide for writing tests in Node.js has been provided to make the process easier. Looking at other tests to see how they should be structured can also help.

The `test` directory within the `nodejs/node` repository is complex and it is often not clear where a new test file should go. When in doubt, add new tests to the `test/parallel/` directory and the right location will be sorted out later.

Before submitting your changes in a pull request, always run the full Node.js test suite. To run the tests (including code linting) on Unix / macOS:

```
./configure && make -j4 test
```

We can speed up the builds by using Ninja. For more information, see Building Node.js with Ninja.

And on Windows:

```
> vcbuild test
```

For some configurations, running all tests might take a long time (an hour or more). To run a subset of the test suite, see the running tests section of the Building guide.

Step 7: Push

Once you are sure your commits are ready to go, with passing tests and linting, begin the process of opening a pull request by pushing your working branch to your fork on GitHub.

```
git push origin my-branch
```

Step 8: Opening the pull request

From within GitHub, opening a new pull request will present you with a pull request template. Please try to do your best at filling out the details, but feel free to skip parts if you're not sure what to put.

Once opened, pull requests are usually reviewed within a few days.

To get feedback on your proposed change even though it is not ready to land, use the `Convert to draft` option in the GitHub UI. Do not use the `wip` label as it might not prevent the PR from landing before you are ready.

Step 9: Discuss and update

You will probably get feedback or requests for changes to your pull request. This is a big part of the submission process so don't be discouraged! Some contributors may sign off on the pull request right away, others may have more detailed comments or feedback. This is a necessary part of the process in order to evaluate whether the changes are correct and necessary.

To make changes to an existing pull request, make the changes to your local branch, add a new commit with those changes, and push those to your fork. GitHub will automatically update the pull request.

```
git add my/changed/files
git commit
git push origin my-branch
```

If a git conflict arises, it is necessary to synchronize your branch with other changes that have landed upstream by using `git rebase`:

```
git fetch upstream HEAD
git rebase FETCH_HEAD
git push --force-with-lease origin my-branch
```

Important: The `git push --force-with-lease` command is one of the few ways to delete history in `git`. It also complicates the review process, as it won't allow reviewers to get a quick glance on what changed. Before you use it, make sure you understand the risks. If in doubt, you can always ask for guidance in the pull request.

There are a number of more advanced mechanisms for managing commits using `git rebase` that can be used, but are beyond the scope of this guide.

Feel free to post a comment in the pull request to ping reviewers if you are awaiting an answer on something. If you encounter words or acronyms that seem unfamiliar, refer to this glossary.

Approval and request changes workflow

All pull requests require "sign off" in order to land. Whenever a contributor reviews a pull request they may find specific details that they would like to see changed or fixed. These may be as simple as fixing a typo, or may involve substantive changes to the code you have written. While such requests are intended to be helpful, they may come across as abrupt or unhelpful, especially requests to change things that do not include concrete suggestions on *how* to change them.

Try not to be discouraged. If you feel that a particular review is unfair, say so, or contact one of the other contributors in the project and seek their input. Often such comments are the result of the reviewer having only taken a short amount of time to review and are not ill-intended. Such issues can often be resolved with a bit of patience. That said, reviewers should be expected to be helpful in their feedback, and feedback that is simply vague, dismissive and unhelpful is likely safe to ignore.

Step 10: Landing

In order to land, a pull request needs to be reviewed and approved by at least two Node.js Collaborators (one collaborator approval is enough if the pull request has been open for more than 7 days) and pass a CI (Continuous Integration) test run. After that, as long as there are no objections from other contributors, the pull request can be merged. If you find your pull request waiting longer than you expect, see the notes about the waiting time.

When a collaborator lands your pull request, they will post a comment to the pull request page mentioning the commit(s) it landed as. GitHub might show the pull request as `Closed` at this point, but don't worry. If you look at the branch you raised your pull request against, you should see a commit with your name on it. Congratulations and thanks for your contribution!

Reviewing pull requests

All Node.js contributors who choose to review and provide feedback on Pull Requests have a responsibility to both the project and the individual making the contribution. Reviews and feedback must be helpful, insightful, and geared towards improving the contribution as opposed to simply blocking it. Do not expect to be able to block a pull request from advancing simply because you say "No" without giving an explanation. Be open to having your mind changed. Be open to working with the contributor to make the pull request better.

Reviews that are dismissive or disrespectful of the contributor or any other reviewers are strictly counter to the Code of Conduct.

When reviewing a pull request, the primary goals are for the codebase to improve and for the person submitting the request to succeed. Even if a pull request does not land, the submitters should come away from the experience feeling like their effort was not wasted or unappreciated. Every pull request from a new contributor is an opportunity to grow the community.

Review a bit at a time

Do not overwhelm new contributors.

It is tempting to micro-optimize and make everything about relative performance, perfect grammar, or exact style matches. Do not succumb to that temptation.

Focus first on the most significant aspects of the change:

1. Does this change make sense for Node.js?
2. Does this change make Node.js better, even if only incrementally?
3. Are there clear bugs or larger scale issues that need attending to?
4. Is the commit message readable and correct? If it contains a breaking change is it clear enough?

When changes are necessary, *request* them, do not *demand* them, and do not assume that the submitter already knows how to add a test or run a benchmark.

Specific performance optimization techniques, coding styles and conventions change over time. The first impression you give to a new contributor never does.

Nits (requests for small changes that are not essential) are fine, but try to avoid stalling the pull request. Most nits can typically be fixed by the Node.js collaborator landing the pull request but they can also be an opportunity for the contributor to learn a bit more about the project.

It is always good to clearly indicate nits when you comment: e.g. `Nit: change foo() to bar(). But this is not blocking.`

If your comments were addressed but were not folded automatically after new commits or if they proved to be mistaken, please, hide them with the appropriate reason to keep the conversation flow concise and relevant.

Be aware of the person behind the code

Be aware that *how* you communicate requests and reviews in your feedback can have a significant impact on the success of the pull request. Yes, we may land a particular change that makes Node.js better, but the individual might just not want to have anything to do with Node.js ever again. The goal is not just having good code.

Respect the minimum wait time for comments

There is a minimum waiting time which we try to respect for non-trivial changes, so that people who may have important input in such a distributed project are able to respond.

For non-trivial changes, pull requests must be left open for at least 48 hours. Sometimes changes take far longer to review, or need more specialized review from subject-matter experts. When in doubt, do not rush.

Trivial changes, typically limited to small formatting changes or fixes to documentation, may be landed within the minimum 48 hour window.

Abandoned or stalled pull requests

If a pull request appears to be abandoned or stalled, it is polite to first check with the contributor to see if they intend to continue the work before checking if they would mind if you took it over (especially if it just has nits left). When doing so, it is courteous to give the original contributor credit for the work they started (either by preserving their name and email address) in the commit log, or by using an `Author:` meta-data tag in the commit.

Approving a change

Any Node.js core collaborator (any GitHub user with commit rights in the `nodejs/node` repository) is authorized to approve any other contributor's work. Collaborators are not permitted to approve their own pull requests.

Collaborators indicate that they have reviewed and approve of the changes in a pull request either by using GitHub's Approval Workflow, which is preferred, or by leaving an `LGTM` ("Looks Good To Me") comment.

When explicitly using the "Changes requested" component of the GitHub Approval Workflow, show empathy. That is, do not be rude or abrupt with your feedback and offer concrete suggestions for improvement, if possible. If you're not sure *how* a particular change can be improved, say so.

Most importantly, after leaving such requests, it is courteous to make yourself available later to check whether your comments have been addressed.

If you see that requested changes have been made, you can clear another collaborator's `Changes requested` review.

Change requests that are vague, dismissive, or unconstructive may also be dismissed if requests for greater clarification go unanswered within a reasonable period of time.

Use `Changes requested` to block a pull request from landing. When doing so, explain why you believe the pull request should not land along with an explanation of what may be an acceptable alternative course, if any.

Accept that there are different opinions about what belongs in Node.js

Opinions on this vary, even among the members of the Technical Steering Committee.

One general rule of thumb is that if Node.js itself needs it (due to historic or functional reasons), then it belongs in Node.js. For instance, `url` parsing is in Node.js because of HTTP protocol support.

Also, functionality that either cannot be implemented outside of core in any reasonable way, or only with significant pain.

It is not uncommon for contributors to suggest new features they feel would make Node.js better. These may or may not make sense to add, but as with all changes, be courteous in how you communicate your stance on these. Comments that make the contributor feel like they should have "known better" or ridiculed for even trying run counter to the Code of Conduct.

Performance is not everything

Node.js has always optimized for speed of execution. If a particular change can be shown to make some part of Node.js faster, it's quite likely to be accepted. Claims that a particular pull request will make things faster will almost always be met by requests for performance benchmark results that demonstrate the improvement.

That said, performance is not the only factor to consider. Node.js also optimizes in favor of not breaking existing code in the ecosystem, and not changing working functional code just for the sake of changing.

If a particular pull request introduces a performance or functional regression, rather than simply rejecting the pull request, take the time to work *with* the contributor on improving the change. Offer feedback and advice on what would make the pull request acceptable, and do not assume that the contributor should already know how to do that. Be explicit in your feedback.

Continuous integration testing

All pull requests that contain changes to code must be run through continuous integration (CI) testing at https://ci.nodejs.org/.

Only Node.js core collaborators and triagers can start a CI testing run. The specific details of how to do this are included in the new collaborator Onboarding guide. Usually, a collaborator or triager will start a CI test run for you as approvals for the pull request come in. If not, you can ask a collaborator or triager to start a CI run.

Ideally, the code change will pass ("be green") on all platform configurations supported by Node.js. This means that all tests pass and there are no linting errors. In reality, however, it is not uncommon for the CI infrastructure itself to fail on specific platforms or for so-called "flaky" tests to fail ("be red"). It is vital to visually inspect the results of all failed ("red") tests to determine whether the failure was caused by the changes in the pull request.

Notes

Commit squashing

In most cases, do not squash commits that you add to your pull request during the review process. When the commits in your pull request land, they may be squashed into one commit per logical change. Metadata will be added to the commit message (including links to the pull request, links to relevant issues, and the names of the reviewers). The commit history of your pull request, however, will stay intact on the pull request page.

For the size of "one logical change", 0b5191f can be a good example. It touches the implementation, the documentation, and the tests, but is still one logical change. All tests should always pass when each individual commit lands on one of the `nodejs/node` branches.

Getting approvals for your pull request

A pull request is approved either by saying LGTM, which stands for "Looks Good To Me", or by using GitHub's Approve button. GitHub's pull request review feature can be used during the process. For more information, check out the video tutorial or the official documentation.

After you push new changes to your branch, you need to get approval for these new changes again, even if GitHub shows "Approved" because the reviewers have hit the buttons before.

Waiting until the pull request gets landed

A pull request needs to stay open for at least 48 hours from when it is submitted, even after it gets approved and passes the CI. This is to make sure that everyone has a chance to weigh in. If the changes are trivial, collaborators may decide it doesn't need to wait. A pull request may well take longer to be merged in. All these precautions are important because Node.js is widely used, so don't be discouraged!

Check out the collaborator guide

If you want to know more about the code review and the landing process, see the collaborator guide.

Appendix: subsystems

- `lib/*.js` (`assert`, `buffer`, etc.)
- `build`
- `doc`
- `lib / src`
- `test`
- `tools`

You can find the full list of supported subsystems in the nodejs/core-validate-commit repository. More than one subsystem may be valid for any particular issue or pull request.