Node.js release process

This document describes the technical aspects of the Node.js release process. The intended audience is those who have been authorized by the Node.js Technical Steering Committee (TSC) to create, promote, and sign official release builds for Node.js, hosted on https://nodejs.org/.

Table of contents

Who can make a release?

Release authorization is given by the Node.js TSC. Once authorized, an individual must have the following:

1. Jenkins release access

There are three relevant Jenkins jobs that should be used for a release flow:

a. Test runs: node-test-pull-request is used for a final full-test run to ensure that the current *HEAD* is stable.

b. Nightly builds: (optional) **iojs+release** can be used to create a nightly release for the current *HEAD* if public test releases are required. Builds triggered with this job are published straight to https://nodejs.org/download/nightly/ and are available for public download.

c. Release builds: iojs+release does all of the work to build all required release assets. Promotion of the release files is a manual step once they are ready (see below).

The Node.js build team is able to provide this access to individuals authorized by the TSC.

2. <nodejs.org> access

The *dist* user on nodejs.org controls the assets available in https://nodejs.org/download/. https://nodejs.org/dist/ is an alias for https://nodejs.org/download/release/.

The Jenkins release build workers upload their artifacts to the web server as the *staging* user. The *dist* user has access to move these assets to public access while, for security, the *staging* user does not.

Nightly builds are promoted automatically on the server by a cron task for the *dist* user.

Release builds require manual promotion by an individual with SSH access to the server as the *dist* user. The Node.js build team is able to provide this access to individuals authorized by the TSC.

3. A publicly-listed GPG key

A `SHASUMS256.txt` file is produced for every promoted build, nightly, and releases. Additionally for releases, this file is signed by the individual responsible for that release. In order to be able to verify downloaded binaries, the public should be able to check that the `SHASUMS256.txt` file has been signed by someone who has been authorized to create a release.

The GPG keys should be fetchable from a known third-party keyserver. The SKS Keyservers at https://sks-keyservers.net are recommended. Use the submission form to submit a new GPG key. You'll need to do an ASCII-armored export of your key first:

```
$ gpg --armor --export email@server.com > ~/nodekey.asc
```

Keys should be fetchable via:

```
$ gpg --keyserver pool.sks-keyservers.net --recv-keys <FINGERPRINT>
```

The key you use may be a child/subkey of an existing key.

Additionally, full GPG key fingerprints for individuals authorized to release should be listed in the Node.js GitHub README.md file.

How to create a release

Notes:

- Dates listed below as *"YYYY-MM-DD"* should be the date of the release **as UTC**. Use `date -u +'%Y-%m-%d'` to find out what this is.
- Version strings are listed below as *"vx.y.z"* or *"x.y.z"*. Substitute for the release version.
- Examples will use the fictional release version `1.2.3`.

0. Pre-release steps

Before preparing a Node.js release, the Build Working Group must be notified at least one business day in advance of the expected release. Coordinating with Build is essential to make sure that the CI works, release files are published, and the release blog post is available on the project website.

Build can be contacted best by opening up an issue on the Build issue tracker.

When preparing a security release, contact Build at least two weekdays in advance of the expected release. To ensure that the security patch(es) can be properly tested, run a `node-test-pull-request` job against the `master` branch of the `nodejs-private/node-private` repository a day or so before the CI lockdown procedure begins. This is to confirm that Jenkins can properly access the private repository.

1. Update the staging branch

Checkout the staging branch locally.

```
$ git remote update
$ git checkout v1.x-staging
$ git reset --hard upstream/v1.x-staging
```

If the staging branch is not up to date relative to `master`, bring the appropriate PRs and commits into it.

Go through PRs with the label `vN.x`. e.g. PRs with the `v8.x` label.

For each PR:

- Run or check that there is a passing CI.
- Check approvals (you can approve yourself).
- Check that the commit metadata was not changed from the `master` commit.

- If there are merge conflicts, ask the PR author to rebase. Simple conflicts can be resolved when landing.

When landing the PR add the `Backport-PR-URL:` line to each commit. Close the backport PR with `Landed in ...`. Update the label on the original PR from `backport-requested-vN.x` to `backported-to-vN.x`.

To determine the relevant commits, use `branch-diff`. The tool is available on npm and should be installed globally or run with `npx`. It depends on our commit metadata, as well as the GitHub labels such as `semver-minor` and `semver-major`. One drawback is that when the `PR-URL` metadata is accidentally omitted from a commit, the commit will show up because it's unsure if it's a duplicate or not.

For a list of commits that could be landed in a patch release on v1.x:

```
$ branch-diff v1.x-staging master --exclude-label=semver-major,semver-minor,dont-land-on-v1
```

Previously released commits and version bumps do not need to be cherry-picked.

Carefully review the list of commits:

- Checking for errors (incorrect `PR-URL`)
- Checking semver status - Commits labeled as `semver-minor` or `semver-major` should only be cherry-picked when appropriate for the type of release being made.
- If you think it's risky and the change should wait for a while, add the `baking-for-lts` tag.

When you are ready to cherry-pick commits, you can automate with the following command. (For semver-minor releases, make sure to remove the `semver-minor` tag from `exclude-label`.)

```
$ branch-diff v1.x-staging master --exclude-label=semver-major,semver-minor,dont-land-on-v1
```

When cherry-picking commits, if there are simple conflicts you can resolve them. Otherwise, add the `backport-requested-vN.x` label to the original PR and post a comment stating that it does not land cleanly and will require a backport PR. You can refer the owner of the PR to the "Backporting to Release Lines" guide.

If commits were cherry-picked in this step, check that the test still pass.

```
$ make test
```

Then, push to the staging branch to keep it up-to-date.

```
$ git push upstream v1.x-staging
```

2. Create a new branch for the release

Create a new branch named `vx.y.z-proposal`, off the corresponding staging branch.

```
$ git checkout -b v1.2.3-proposal upstream/v1.x-staging
```

3. Update `src/node_version.h`

Set the version for the proposed release using the following macros, which are already defined in `src/node_version.h`:

```
#define NODE_MAJOR_VERSION x
#define NODE_MINOR_VERSION y
#define NODE_PATCH_VERSION z
```

Set the `NODE_VERSION_IS_RELEASE` macro value to `1`. This causes the build to be produced with a version string that does not have a trailing pre-release tag:

```
#define NODE_VERSION_IS_RELEASE 1
```

4. Update the changelog

Step 1: Collect the formatted list of changes Collect a formatted list of commits since the last release. Use `changelog-maker` to do this:

```
$ changelog-maker --group
```

`changelog-maker` counts commits since the last tag and if the last tag in the repository was not on the current branch you may have to supply a `--start-ref` argument:

```
$ changelog-maker --group --filter-release --start-ref v1.2.2
```

`--filter-release` will remove the release commit from the previous release.

Step 2: Update the appropriate doc/changelogs/CHANGELOG_*.md file There is a separate `CHANGELOG_Vx.md` file for each major Node.js release line. These are located in the `doc/changelogs/` directory. Once the formatted list of changes is collected, it must be added to the top of the relevant changelog file in the release branch (e.g. a release for Node.js v4 would be added to the `/doc/changelogs/CHANGELOG_V4.md`).

Please do *not* add the changelog entries to the root `CHANGELOG.md` file.

The new entry should take the following form:

```
<a id="x.y.x"></a>
## YYYY-MM-DD, Version x.y.z (Release Type), @releaser

### Notable changes

* List interesting changes here
* Particularly changes that are responsible for minor or major version bumps
* Also be sure to look at any changes introduced by dependencies such as npm
* ... and include any notable items from there
```

```
### Commits

* Include the full list of commits since the last release here. Do not include "Working on
```

The release type should be either Current, LTS, or Maintenance, depending on the type of release being produced.

You can use `branch-diff` to get a list of commits with the `notable-change` label:

```
$ branch-diff upstream/v1.x v1.2.3-proposal --require-label=notable-change -format=simple
```

Be sure that the `<a>` tag, as well as the two headings, are not indented at all.

At the top of the root `CHANGELOG.md` file, there is a table indexing all releases in each major release line. A link to the new release needs to be added to it. Follow the existing examples and be sure to add the release to the *top* of the list. The most recent release for each release line is shown in **bold** in the index. When updating the index, please make sure to update the display accordingly by removing the bold styling from the previous release.

Run `make format-md` to ensure the `CHANGELOG_Vx.md` and `CHANGELOG.md` files are formatted correctly.

Step 3: Update any REPLACEME and DEP00XX tags in the docs
If this release includes new APIs then it is necessary to document that they were first added in this version. The relevant commits should already include `REPLACEME` tags as per the example in the docs README. Check for these tags with

```
grep REPLACEME doc/api/*.md
```

and substitute this node version with

```
sed -i "s/REPLACEME/$VERSION/g" doc/api/*.md
```

or

```
perl -pi -e "s/REPLACEME/$VERSION/g" doc/api/*.md
```

`$VERSION` should be prefixed with a `v`.

If this release includes any new deprecations it is necessary to ensure that those are assigned a proper static deprecation code. These are listed in the docs (see `doc/api/deprecations.md`) and in the source as `DEP00XX`. The code must be assigned a number (e.g. `DEP0012`). This assignment should occur when the PR is landed, but a check will be made when the release build is run.

5. Create release commit

The `CHANGELOG.md`, `doc/changelogs/CHANGELOG_Vx.md`, `src/node_version.h`, and `REPLACEME` changes should be the final commit that will be tagged for the release. When committing these to git, use the following message format:

```
YYYY-MM-DD, Version x.y.z (Release Type)

Notable changes:

* Copy the notable changes list here, reformatted for plain-text
```

For security releases, begin the commit message with the phrase `This is a security release.` to allow the distribution indexer to identify it as such:

```
YYYY-MM-DD, Version x.y.z (Release Type)

This is a security release.

Notable changes:

* Copy the notable changes list here, reformatted for plain-text
```

6. Propose release on GitHub

Push the release branch to `nodejs/node`, not to your own fork. This allows release branches to more easily be passed between members of the release team if necessary.

Create a pull request targeting the correct release line. For example, a `v5.3.0-proposal` PR should target `v5.x`, not master. Paste the CHANGELOG modifications into the body of the PR so that collaborators can see what is changing. These PRs should be left open for at least 24 hours, and can be updated as new commits land. If the CHANGELOG pasted into the pull request is long enough that it slows down the GitHub UI, consider pasting the commits into `<details>` tags or in follow up comments.

If using the `<details>` tag, use the following format:

```
<details>
<summary>Commits</summary>

* Full list of commits...
</details>
```

If you need any additional information about any of the commits, this PR is a good place to @-mention the relevant contributors.

After opening the PR, update the release commit to include `PR-URL` metadata and force-push the proposal.

7. Ensure that the release branch is stable

Run a `node-test-pull-request` test run to ensure that the build is stable and the HEAD commit is ready for release.

Also run a `node-test-commit-v8-linux` test run if the release contains changes to `deps/v8`.

Perform some smoke-testing. There is the `citgm-smoker` CI job for this purpose. Run it once with the base `vx.x` branch as a reference and with the proposal branch to check if new regressions could be introduced in the ecosystem.

8. Produce a nightly build *(optional)*

If there is a reason to produce a test release for the purpose of having others try out installers or specifics of builds, produce a nightly build using **iojs+release** and wait for it to drop in https://nodejs.org/download/nightly/. Follow the directions and enter a proper length commit SHA, enter a date string, and select "nightly" for "disttype".

This is particularly recommended if there has been recent work relating to the macOS or Windows installers as they are not tested in any way by CI.

9. Produce release builds

Use **iojs+release** to produce release artifacts. Enter the commit that you want to build from and select "release" for "disttype".

Artifacts from each worker are uploaded to Jenkins and are available if further testing is required. Use this opportunity particularly to test macOS and Windows installers if there are any concerns. Click through to the individual workers for a run to find the artifacts.

All release workers should achieve "SUCCESS" (and be green, not red). A release with failures should not be promoted as there are likely problems to be investigated.

You can rebuild the release as many times as you need prior to promoting them if you encounter problems.

If you have an error on Windows and need to start again, be aware that you'll get immediate failure unless you wait up to 2 minutes for the linker to stop from previous jobs. i.e. if a build fails after having started compiling, that worker will still have a linker process that's running for another couple of minutes which will prevent Jenkins from clearing the workspace to start a new one. This isn't a big deal, it's just a hassle because it'll result in another failed build if you start again!

10. Test the build

Jenkins collects the artifacts from the builds, allowing you to download and install the new build. Make sure that the build appears correct. Check the version numbers, and perform some basic checks to confirm that all is well with the build before moving forward.

11. Tag and sign the release commit

Once you have produced builds that you're happy with, create a new tag. By waiting until this stage to create tags, you can discard a proposed release if something goes wrong or additional commits are required. Once you have created a tag and pushed it to GitHub, you ***must not*** delete and re-tag. If you make a mistake after tagging then you'll have to version-bump and start again and count that tag/version as lost.

Tag summaries have a predictable format. Look at a recent tag to see:

```
git tag -v v6.0.0
```

The message should look something like `2016-04-26 Node.js v6.0.0 (Current) Release`.

Install `git-secure-tag` npm module:

```
$ npm install -g git-secure-tag
```

Create a tag using the following command:

```
$ git secure-tag <vx.y.z> <commit-sha> -sm "YYYY-MM-DD Node.js vx.y.z (<release-type>) Relea
```

`release-type` is either "Current" or "LTS". For LTS releases, you should also include the release code name.

```
2019-10-22 Node.js v10.17.0 'Dubnium' (LTS) Release
```

The tag **must** be signed using the GPG key that's listed for you on the project README.

12. Set up for the next release

On release proposal branch, edit `src/node_version.h` again and:

- Increment `NODE_PATCH_VERSION` by one
- Change `NODE_VERSION_IS_RELEASE` back to `0`

Commit this change with the following commit message format:

```
Working on vx.y.z # where 'z' is the incremented patch number

PR-URL: <full URL to your release proposal PR>
```

This sets up the branch so that nightly builds are produced with the next version number *and* a pre-release tag.

Merge your release proposal branch into the stable branch that you are releasing from and rebase the corresponding staging branch on top of that.

```
$ git checkout v1.x
$ git merge --ff-only v1.2.3-proposal
$ git push upstream v1.x
$ git checkout v1.x-staging
$ git rebase v1.x
$ git push upstream v1.x-staging
```

13. Cherry-pick the release commit to `master`

```
$ git checkout master
$ git cherry-pick v1.x^
```

Git should stop to let you fix conflicts. Revert all changes that were made to `src/node_version.h`:

```
$ git checkout --ours HEAD -- src/node_version.h
```

If there are conflicts in `doc` due to updated `REPLACEME` placeholders (that happens when a change previously landed on another release branch), keep both version numbers. Convert the YAML field to an array if it is not already one.

Then finish cherry-picking and push the commit upstream:

```
$ git add src/node_version.h doc
$ git cherry-pick --continue
$ make lint
$ git push upstream master
```

Do not cherry-pick the "Working on vx.y.z" commit to `master`.

14. Push the release tag

Push the tag to the repository before you promote the builds. If you haven't pushed your tag first, then build promotion won't work properly. Push the tag using the following command:

```
$ git push <remote> <vx.y.z>
```

Note: Please do not push the tag unless you are ready to complete the remainder of the release steps.

15. Promote and sign the release builds

The same individual who signed the release tag must be the one to promote the builds as the `SHASUMS256.txt` file needs to be signed with the same GPG key!

Use `tools/release.sh` to promote and sign the build. Before doing this, you'll need to ensure you've loaded the correct ssh key, or you'll see the following:

```
# Checking for releases ...
Enter passphrase for key '/Users/<user>/.ssh/id_rsa':
dist@direct.nodejs.org's password:
```

The key can be loaded either with `ssh-add`:

```
# Substitute node_id_rsa with whatever you've named the key
$ ssh-add ~/.ssh/node_id_rsa
```

or at runtime with:

```
# Substitute node_id_rsa with whatever you've named the key
$ ./tools/release.sh -i ~/.ssh/node_id_rsa
```

`tools/release.sh` will perform the following actions when run:

a. Select a GPG key from your private keys. It will use a command similar to: `gpg --list-secret-keys` to list your keys. If you don't have any keys, it will bail. If you have only one key, it will use that. If you have more than one key it will ask you to select one from the list. Be sure to use the same key that you signed your git tag with.

b. Log in to the server via SSH and check for releases that can be promoted, along with the list of artifacts. It will use the `dist-promotable` command on the server to find these. You will be asked, for each promotable release, whether you want to proceed. If there is more than one release to promote (there shouldn't be), be sure to only promote the release you are responsible for.

c. Log in to the server via SSH and run the promote script for the given release. The command on the server will be similar to: `dist-promote vx.y.z`. After this step, the release artifacts will be available for download and a `SHASUMS256.txt` file will be present. The release will still be unsigned, however.

d. Use `scp` to download `SHASUMS256.txt` to a temporary directory on your computer.

e. Sign the `SHASUMS256.txt` file using a command similar to: `gpg --default-key YOURKEY --digest-algo SHA256 --clearsign /path/to/SHASUMS256.txt`. You will be prompted by GPG for your password. The signed file will be named SHASUMS256.txt.asc.

f. Output an ASCII armored version of your public GPG key using a command similar to: `gpg --default-key YOURKEY --digest-algo SHA256 --detach-sign /path/to/SHASUMS256.txt`. You will be prompted by GPG for your password. The signed file will be named SHASUMS256.txt.sig.

g. Upload the `SHASUMS256.txt` files back to the server into the release directory.

It is possible to only sign a release by running `./tools/release.sh -s vX.Y.Z`.

16. Check the release

Your release should be available at `https://nodejs.org/dist/vx.y.z/` and https://nodejs.org/dist/latest/. Check that the appropriate files are in place. You may want to check that the binaries are working as appropriate and have the right internal version strings. Check that the API docs are available at https://nodejs.org/api/. Check that the release catalog files are correct at https://nodejs.org/dist/index.tab and https://nodejs.org/dist/index.json.

17. Create a blog post

There is an automatic build that is kicked off when you promote new builds, so within a few minutes nodejs.org will be listing your new version as the latest release. However, the blog post is not yet fully automatic.

Create a new blog post by running the nodejs.org release-post.js script. This script will use the promoted builds and changelog to generate the post. Run `npm run serve` to preview the post locally before pushing to the nodejs.org repository.

- You can add a short blurb just under the main heading if you want to say something important, otherwise the text should be publication ready.
- The links to the download files won't be complete unless you waited for the ARMv6 builds. Any downloads that are missing will have `*Coming soon*` next to them. It's your responsibility to manually update these later when you have the outstanding builds.
- The `SHASUMS256.txt.asc` content is at the bottom of the post. When you update the list of tarballs you'll need to copy/paste the new contents of this file to reflect those changes.
- Always use pull-requests on the nodejs.org repository. Be respectful of the website team, but you do not have to wait for PR sign-off. Please use the following commit message format:

  ```
  Blog: vX.Y.Z release post

  Refs: <full URL to your release proposal PR>
  ```

- Changes to the base branch, `main`, on the nodejs.org repository will trigger a new build of nodejs.org so your changes should appear a few minutes after pushing.

18. Create the release on GitHub

- Go to the New release page.
- Select the tag version you pushed earlier.
- For release title, copy the title from the changelog.
- For the description, copy the rest of the changelog entry.

- Click on the "Publish release" button.

19. Cleanup

Close your release proposal PR and delete the proposal branch.

20. Announce

The nodejs.org website will automatically rebuild and include the new version. To announce the build on Twitter through the official @nodejs account, email pr@nodejs.org with a message such as:

> v5.8.0 of @nodejs is out: https://nodejs.org/en/blog/release/v5.8.0/
> ... something here about notable changes

To ensure communication goes out with the timing of the blog post, please allow 24 hour prior notice. If known, please include the date and time the release will be shared with the community in the email to coordinate these announcements.

Ping the IRC ops and the other Partner Communities liaisons.

21. Celebrate

In whatever form you do this...

LTS Releases

Marking a release line as LTS

To mark a release line as LTS, the following changes must be made to `src/node_version.h`:

- The `NODE_MINOR_VERSION` macro must be incremented by one
- The `NODE_PATCH_VERSION` macro must be set to `0`
- The `NODE_VERSION_IS_LTS` macro must be set to `1`
- The `NODE_VERSION_LTS_CODENAME` macro must be set to the code name selected for the LTS release.

For example:

```
-#define NODE_MINOR_VERSION 12
-#define NODE_PATCH_VERSION 1
+#define NODE_MINOR_VERSION 13
+#define NODE_PATCH_VERSION 0

-#define NODE_VERSION_IS_LTS 0
-#define NODE_VERSION_LTS_CODENAME ""
+#define NODE_VERSION_IS_LTS 1
+#define NODE_VERSION_LTS_CODENAME "Erbium"
```

```
-#define NODE_VERSION_IS_RELEASE 0
+#define NODE_VERSION_IS_RELEASE 1
```

The changes must be made as part of a new semver-minor release.

Update release labels

The `lts-watch-vN.x` issue label must be created, with the same color as other existing labels for that release line, such as `vN.x`.

If the release is transitioning from Active LTS to Maintenance, the `backport-requested-vN.x` label must be deleted.

Major releases

The process for cutting a new Node.js major release has a number of differences from cutting a minor or patch release.

Schedule

New Node.js Major releases happen twice per year:

- Even-numbered releases are cut in April.
- Odd-numbered releases are cut in October.

Major releases should be targeted for the third Tuesday of the release month.

A major release must not slip beyond the release month. In other words, major releases must not slip into May or November.

The release date for the next major release should be announced immediately following the current release (e.g. the release date for 13.0.0 should be announced immediately following the release of 12.0.0).

Release branch

Approximately two months before a major release, new `vN.x` and `vN.x-staging` branches (where `N` indicates the major release) should be created as forks of the `master` branch. Up until one week before the release date, these must be kept in sync with `master`.

The `vN.x` and `vN.x-staging` branches must be kept in sync with one another up until the date of the release.

The TSC should be informed of any `SEMVER-MAJOR` commits that land within one month of the release.

Create release labels

The following issue labels must be created:

- `vN.x`
- `backport-blocked-vN.x`
- `backport-open-vN.x`
- `backport-requested-vN.x`
- `backported-to-vN.x`
- `dont-land-on-vN.x`

The label description can be copied from existing labels of previous releases. The label color must be the same for all new labels, but different from the labels of previous releases.

Release proposal

A draft release proposal should be created 6 weeks before the release. A separate `vN.x-proposal` branch should be created that tracks the `vN.x` branch. This branch will contain the draft release commit (with the draft changelog).

Notify the `@nodejs/npm` team in the release proposal PR to inform them of the upcoming release. `npm` maintains a list of supported versions that will need updating to include the new major release.

Update `NODE_MODULE_VERSION`

This macro in `src/node_version.h` is used to signal an ABI version for native addons. It currently has two common uses in the community:

- Determining what API to work against for compiling native addons, e.g. NAN uses it to form a compatibility-layer for much of what it wraps.
- Determining the ABI for downloading pre-built binaries of native addons, e.g. node-pre-gyp uses this value as exposed via `process.versions.modules` to help determine the appropriate binary to download at install-time.

The general rule is to bump this version when there are *breaking ABI* changes and also if there are non-trivial API changes. The rules are not yet strictly defined, so if in doubt, please confer with someone that will have a more informed perspective, such as a member of the NAN team.

A registry of currently used `NODE_MODULE_VERSION` values is maintained at https://github.com/nodejs/node/blob/HEAD/doc/abi_version_registry.json. When bumping `NODE_MODULE_VERSION`, you should choose a new value not listed in the registry. Also include a change to the registry in your commit to reflect the newly used value. Ensure that the release commit removes the `-pre` suffix for the major version being prepared.

It is current TSC policy to bump major version when ABI changes. If you see a need to bump `NODE_MODULE_VERSION` outside of a major release then you should consult the TSC. Commits may need to be reverted or a major version bump may need to happen.

Test releases and release candidates

Test builds should be generated from the `vN.x-proposal` branch starting at about 6 weeks before the release.

Release Candidates should be generated from the `vN.x-proposal` branch starting at about 4 weeks before the release, with a target of one release candidate per week.

Always run test releases and release candidates through the Canary in the Goldmine tool for additional testing.

Changelogs

Generating major release changelogs is a bit more involved than minor and patch changelogs.

Create the changelog file In the `doc/changelogs` directory, create a new `CHANGELOG_V{N}.md` file where `{N}` is the major version of the release. Follow the structure of the existing `CHANGELOG_V*.md` files.

The navigation headers in all of the `CHANGELOG_V*.md` files must be updated to account for the new `CHANGELOG_V{N}.md` file.

Once the file is created, the root `CHANGELOG.md` file must be updated to reference the newly-created major release `CHANGELOG_V{N}.md`.

Generate the changelog To generate a proper major release changelog, use the `branch-diff` tool to compare the `vN.x` branch against the `vN-1.x` branch (e.g. for Node.js 12.0, we compare the `v12.x` branch against the up to date `v11.x` branch). Make sure that the local copy of the downlevel branch is up to date.

The commits in the generated changelog must then be organized:

- Remove all release commits from the list
- Remove all reverted commits and their reverts
- Separate all SEMVER-MAJOR, SEMVER-MINOR, and SEMVER-PATCH commits into lists

Generate the notable changes For a major release, all SEMVER-MAJOR commits that are not strictly internal, test, or doc-related are to be listed as notable changes. Some SEMVER-MINOR commits may be listed as notable changes on a case-by-case basis. Use your judgment there.

Update the expected assets

The promotion script does a basic check that the expected files are present. Open a pull request in the Build repository to add the list of expected files for the new release line as a new file, `v{N}.x` (where `{N}` is the major version of

the release), in the expected assets folder. The change will need to be deployed onto the web server by a member of the build-infra team before the release is promoted.

Snap

The Node.js Snap package has a "default" for installs where the user hasn't specified a release line ("track" in Snap terminology). This should be updated to point to the most recently activated LTS. A member of the Node.js Build Infrastructure team is able to perform the switch of the default. An issue should be opened on the Node.js Snap management repository requesting this take place once a new LTS line has been released.