Contributing to normalize.css

Please take a moment to review this document in order to make the contribution process easy and effective for everyone involved.

Following these guidelines helps to communicate that you respect the time of the developers managing and developing this open source project. In return, they should reciprocate that respect in addressing your issue or assessing patches and features.

Using the issue tracker

The issue tracker is the preferred channel for bug reports, features requests and submitting pull requests, but please respect the following restrictions:

- Please **do not** use the issue tracker for personal support requests.
- Please **do not** derail or troll issues. Keep the discussion on topic and respect the opinions of others.

Bug reports

A bug is a *demonstrable problem* that is caused by the code in the repository. Good bug reports are extremely helpful - thank you!

Guidelines for bug reports:

1. **Use the GitHub issue search** – check if the issue has already been reported.
2. **Check if the issue has been fixed** – try to reproduce it using the latest `master` branch in the repository.
3. **Isolate the problem** – create a live example (e.g., on Codepen) of a reduced test case.

A good bug report shouldn't leave others needing to chase you up for more information. Please try to be as detailed as possible in your report. What is your environment? What steps will reproduce the issue? What browser(s) and OS experience the problem? What would you expect to be the outcome? All these details will help people to fix any potential bugs.

Example:

> *Short and descriptive example bug report title*
>
> *A summary of the issue and the browser/OS environment in which it occurs. If suitable, include the steps required to reproduce the bug.*
>
> 1. *This is the first step*
> 2. *This is the second step*
> 3. *Further steps, etc.*
>
> `<url>` *- a link to the reduced test case*
>
> *Any other information you want to share that is relevant to the issue being reported. This might include the lines of code that you have identified as causing the bug, and potential solutions (and your opinions on their merits).*

Feature requests

Feature requests are welcome. But take a moment to find out whether your idea fits with the scope and aims of the project. It's up to *you* to make a strong case to convince the project's developers of the merits of this feature. Please provide as much detail and context as possible.

Pull requests

Good pull requests - patches, improvements, new features - are a fantastic help. They should remain focused in scope and avoid containing unrelated commits.

Please ask first before embarking on any significant work, otherwise you risk spending a lot of time working on something that the project's developers might not want to merge into the project.

Please adhere to the coding conventions used throughout a project (whitespace, accurate comments, etc.) and any other requirements (such as test coverage).

Follow this process if you'd like your work considered for inclusion in the project:

1. Fork the project, clone your fork, and configure the remotes:

   ```
   # Clone your fork of the repo into the current directory
   git clone https://github.com/<your-username>/normalize.css
   # Navigate to the newly cloned directory
   cd normalize.css
   # Assign the original repo to a remote called "upstream"
   git remote add upstream https://github.com/necolas/normalize.css
   ```

2. If you cloned a while ago, get the latest changes from upstream:

   ```
   git checkout master
   git pull upstream master
   ```

3. Never work directly on `master`. Create a new topic branch (off the latest version of `master`) to contain your feature, change, or fix:

   ```
   git checkout -b <topic-branch-name>
   ```

4. Commit your changes in logical chunks. Please adhere to these git commit message conventions or your code is unlikely be merged into the main project. Use Git's interactive rebase feature to tidy up your commits before making them public.

   Be sure to add a test to the `test.html` file if appropriate, and test your change in all supported browsers.

5. Locally rebase the upstream development branch into your topic branch:

   ```
   git pull --rebase upstream master
   ```

6. Push your topic branch up to your fork:

   ```
   git push origin <topic-branch-name>
   ```

7. Open a Pull Request with a clear title and description.

IMPORTANT: By submitting a patch, you agree to allow the project owner to license your work under the same license as that used by the project.

CSS Conventions

Keep the CSS file as readable as possible by following these guidelines:

- Comments are short and to the point.
- Comments without a number reference the entire rule.
- Comments describe the selector when the selector does not make the normalization obvious.
- Comments begin with "Correct the..." when they deal with less obvious side effects.
- Rules are sorted by cascade, specificity, and then alphabetic order.
- Selectors are sorted by specificity and then alphabetic order.
- `in browser` applies to all versions.
- `in browser v-` applies to all versions up to and including the version.
- `in browser v+` applies to all versions after and including the version.
- `in browser v-v` applies to all versions including and between the versions.

Maintainers

If you have commit access, please follow this process for merging patches and cutting new releases.

Accepting patches

1. Check that a patch is within the scope and philosophy of the project.
2. Check that a patch has any necessary tests and a proper, descriptive commit message.
3. Test the patch locally.
4. Do not use GitHub's merge button. Apply the patch to `master` locally (either via `git am` or by checking the whole branch out). Amend minor problems with the author's original commit if necessary. Then push to GitHub.

Releasing a new version

1. Include all new functional changes in the CHANGELOG.
2. Use a dedicated commit to increment the version. The version needs to be added to the CHANGELOG (inc. date), the `package.json`, and `normalize.css` files.
3. The commit message must be of `v0.0.0` format.
4. Create an annotated tag for the version: `git tag -m "v0.0.0" 0.0.0`.
5. Push the changes and tags to GitHub: `git push --tags origin master`
6. Checkout the `gh-pages` branch and follow the instructions in the README.

Semver strategy

Semver is a widely accepted method for deciding how version numbers are incremented in a project. Versions are written as MAJOR.MINOR.PATCH.

Any change to CSS rules whatsoever is considered backwards-breaking and will result in a new **major** release. No changes to CSS rules can add functionality in a backwards-compatible manner, therefore no changes are considered **minor**. Others changes with no impact on rendering are considered backwards-compatible and will result in a new **patch** release.