Jul 31, 2017 · 4 min read

Your last ESLint config

There is a moment in each team's life in which it needs to decide what code style it's going to use. After many hours of flame war, countless code examples discussed, Dan Abramov's snippets cited, the team achieves coveted consensus. We, techy people know that it is crucial to note such important conclusion, so we take battle-tested tool, let's say ESLint, and compose the holy `.eslintrc` file.

And from now on the life of all team members is much better, every indent, breakline, spacing has it's rule and can be linted in no time using one command. Except not really…

ESLint is a great tool but it can't fix all the problems related to code formatting/linting. Just look at this table. Rows marked with wrench icon represent rules which can be fixed by ESLint automatically. Not so many, aye? In my opinion the flashpoint is *line width*. Poor ESLint doesn't know how to break your reaaaaaaaaly long line, because it could be a long string, an inlined object, an array or even a function declaration.

There are also some rules related to JSX which are not covered by ESLint fully, so there is some place for some our beloved space-vs-tabs flame war.

Fortunately *there is an app for that* (tool, to be precise). I mean Prettier. This cool piece of tech reads the code, breaks it into symbols and then rewrites your code in its own way. It will break your too long inlined object in it's own fasion and will not complain at all. Cool, isn't it? Yea, I know that some of the results are not exactly the same your team agreed upon, but hey, there is a machine which will do it for you! Automatically. No more flame wars, no more discussions about indentation!

So, now we have pretty, formatted and linted code. What could go wrong next? Just listen: sooner or later you will need to share this setup between two projects. Or maybe you'd like to use the setup in your off-time pet projects? Read on.

ESLint meets Prettier

We will begin with introducing Prettier to your ESLint. It would be perfect to setup it in such a way that not interested team members will not even know about it's existence. Fortunately we have plugin for that: eslint-plugin-prettier. It will add Prettier's rules to ESLint configuration and allow ESLint to use Prettier as a code formatter with `--fix` command. There is also the second piece of this puzzle: eslint-config-prettier. This config will disable all the ESLint rules that are in conflict with Prettier opinion, so there sould be no conflicts between them.

Let's install all of the tools *(I'll be using yarn in all examples)*:

```
yarn add eslint eslint-plugin-prettier eslint-config-
prettier prettier -D -E
```

And then create `.eslintrc.js` (I like my configs being JS):

```
module.exports = {
  extends: ['eslint:recommended', 'prettier'], // extending
  plugins: ['prettier'], // activating esling-plugin-pretti
  rules: {
    'prettier/prettier': [ // customizing prettier rules (u
      'error',
      {
        singleQuote: true,
        trailingComma: 'all',
      },
```

Example ESLint config using Prettier in the pipeline.

Now you're ready to lint the code:

`yarn eslint .`

And cast Prettier spell:

`yarn eslint --fix .`

Boom! There you have it. Fully working ESLint + Prettier setup. Now, let's talk about sharing the setup.

Sharing our ESLint config

Now, I'm sure that you (and your team) are happy with the config and code formatting. It would be a great waste of time if you would compose the config files for every other project over and over. Or maybe your config is better than Airbnb's one? I think it's time to share it.

We start with creating a single JS file which stores our ESLint config. The filename is determined by `main` field in `package.json` : eg.

```
"main": "index.js"
```

In this case we are putting the config we like to share into `index.js` . We can also reuse `.eslintrc.js` if we like to use it during defining our rules:

```
const eslintrc = require('./.eslintrc');

module.exports = eslintrc;
```

Then, remember to move your dev dependencies to peer dependencies:

```
yarn add eslint eslint-plugin-prettier eslint-config-
prettier prettier -P
```

And loosen them a bit (that way it will be easer to reuse the config):

```
[...]
"eslint": ">= 4",
"eslint-config-prettier": ">= 2",
"eslint-plugin-prettier": ">= 2",
"prettier": ">= 1.5"
[...]
```

Now you're ready to publish your config to npm or use it directly from repo/directory.

You use your new config by extending project's `.eslintrc` with config name:

```
[...]
extends: ['my-config-name'],
[...]
```

Here you can read more about sharing ESLint config.

The easiest way

If you're in a hurry, or don't want to spend time on setting up linter in your new pet project use this config: eslint-config-last. This config contains all the rules described above plus some cool features required by current standards such as: ES 2017 support, class properties and so on. This package will be totally opinionated and will contain all the rules I believe are required to develop modern, consistent JavaScript codebase.

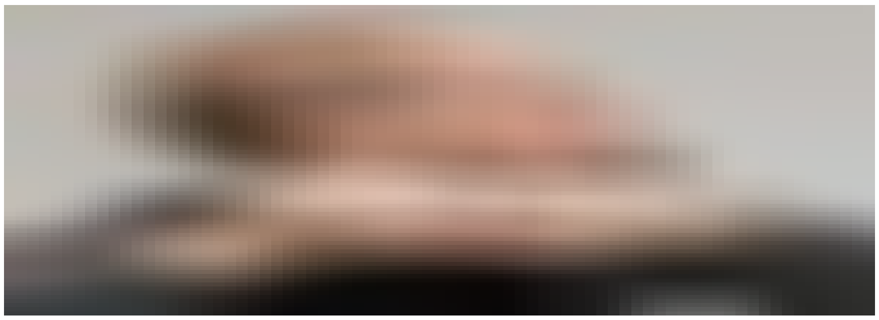
If you'd like to use it—let's say with React—you can extend the setup using:

```
yarn add eslint-config-last eslint-plugin-react -D -E
```

Installing peer dependencies:

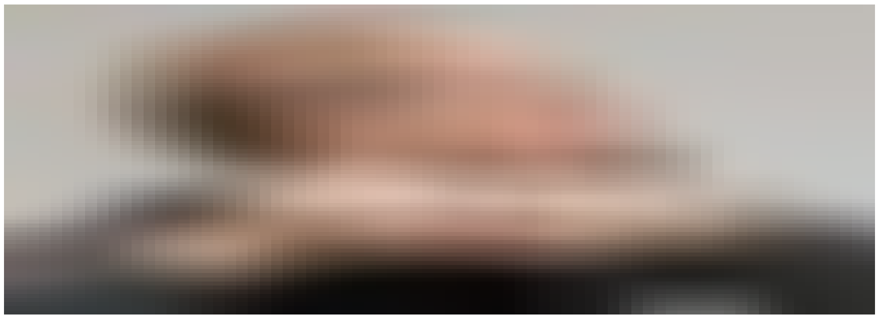
```
yarn add eslint babel-eslint prettier eslint-plugin-prettier
eslint-config-prettier -D -E
```

And then configure ESLint:

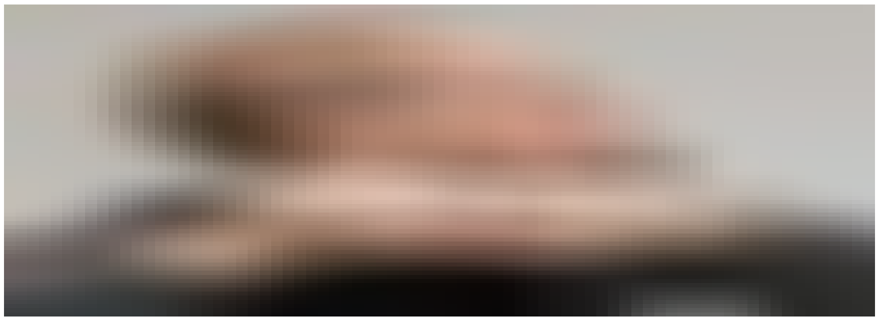


ESLint configuration extending Last config and React rules.

Now you're ready to fix your modern React code! Here is a small example of this config results:



React component code before the lint and formatting.



The code above after lint and formatting. Much better, isn't it?

Summary

And that's all for now regarding linting your JS code. We have learned what's Prettier and how to add it to ESLint pipeline. We also covered sharing your new config with others or using it in pet projects. For a start you can use a predefined config. The combination of ESLint and

Prietter allows us to focus more on the code and solutions rather then code style.

Happy coding!