

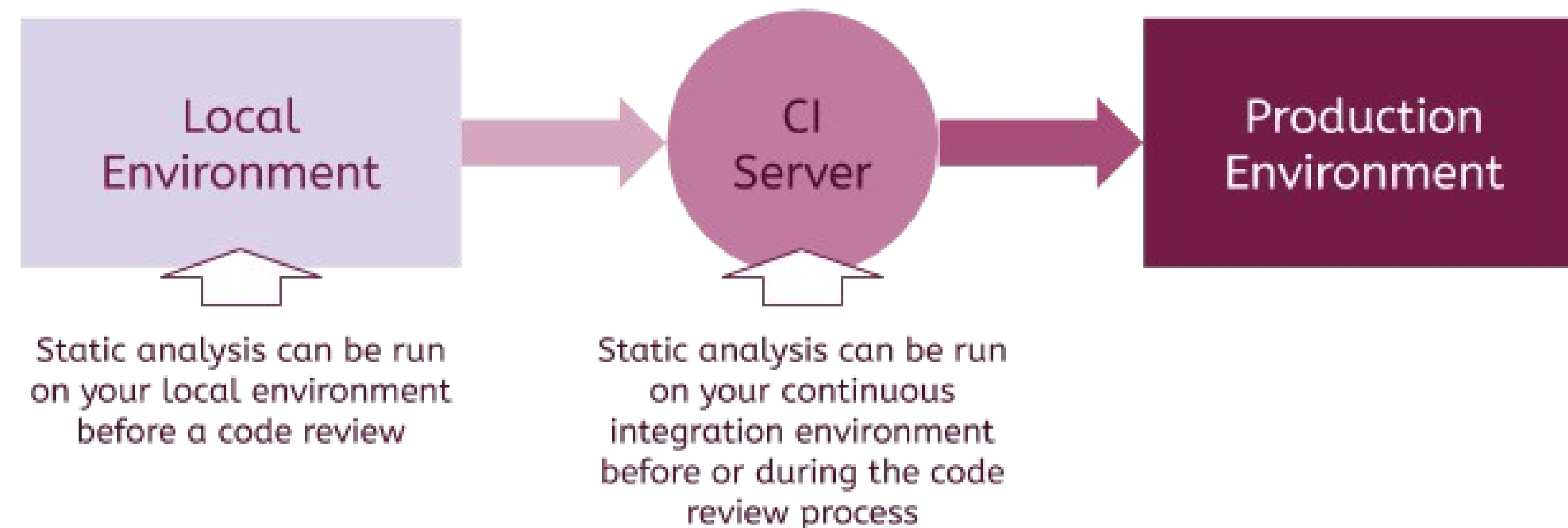
No-config JavaScript/TypeScript style enforcement with StandardJS and Prettier

What is Static Analysis?

Static analysis, also called **Static Code Analysis**, is the practice of analyzing source code before it is running.

Static analysis can be fully automated, therefore it's one of the best ways to improve the quality of your JavaScript code without investing developer time.

Programs which automate static code analysis are called **Linters**. Except for reporting problems, modern linters can also automatically fix some of the issues if you request them to do so.



Analyzing JavaScript with ESLint

ESLint is a configurable JavaScript linter that helps you find and fix problems in your JavaScript code. Problems can be anything from potential runtime bugs, to not following best practices, to styling issues.

Rules are the core building block of ESLint. A rule validates if your code meets a certain expectation, and what to do if it does not meet that expectation.



```
9  module.exports = {
10    parser: '@babel/eslint-parser',
11    extends: [
12      'airbnb',
13      'plugin:react-hooks/recommended',
14      'plugin:react-redux/recommended',
15      'plugin:security/recommended',
16    ],
17    > env: { ...
22    },
23    plugins: [
24      '@babel',
25      'react',
26      'jsx-a11y',
27      'react-intl',
28      'flowtype',
29      'eslint-plugin-formatjs',
30      'react-redux',
31    ],
32    > parserOptions: { ...
41    },
42    rules: {
43      'no-underscore-dangle': 'off',
44      'no-multiple-empty-lines': 'off',
45      'no-tabs': 'error',
46      'no-restricted-exports': 'warn',
47      'default-case-last': 'error',
48      'no-unsafe-optional-chaining': 'error',
49      'function-call-argument-newline': ['error', 'consistent'],
50      'no-promise-executor-return': 'error',
51      'prefer-exponentiation-operator': 'off',
52      'prefer-regex-literals': 'off',
53      'arrow-parens': [
54        'error',
55        'always',
56      ],
57      'arrow-body-style': [
58        'error',
59        'as-needed',
60      ],
61    },
62  }
```

StandardJS: JavaScript Standard Style

No configuration. The easiest way to enforce code quality in your project. No decisions to make. No .eslintrc files to manage. It just works.

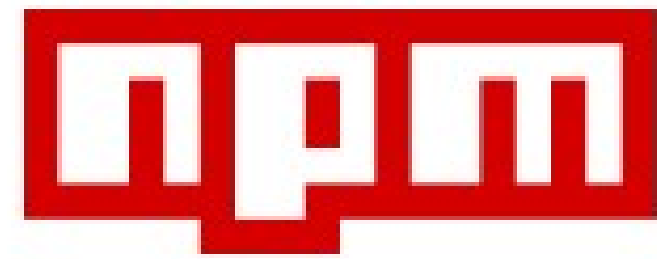
Automatically format code. Just run `standard --fix` and say goodbye to messy or inconsistent code.

Catch style issues & programmer errors early. Save precious code review time by eliminating back-and-forth between reviewer & contributor.

StandardJS uses ESLint / ESLint rules under the hood.



Who uses JavaScript Standard Style



GitHub



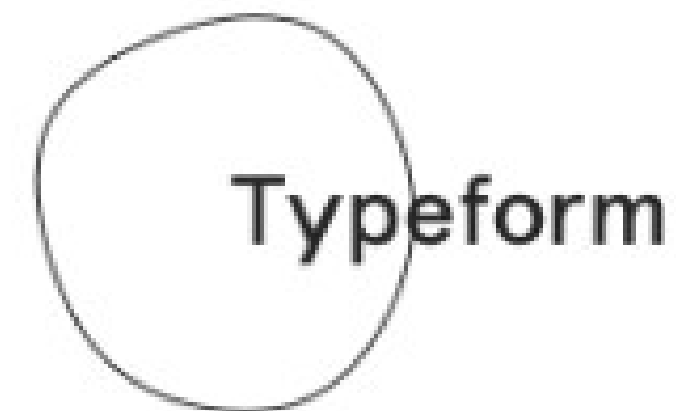
express



NUXTJS



zendesk



ts-standard

TypeScript Style Guide, with linter and automatic code fixer based on StandardJS.

Install globally with `sudo npm i -g ts-standard` and just run:

OR:

Integrate with your existing `eslint` setup:

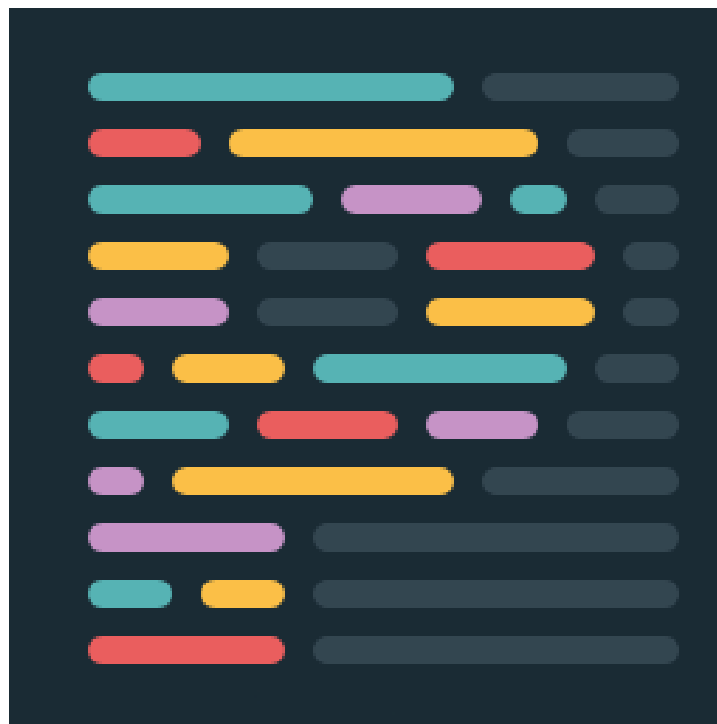
```
ts-standard  
or  
ts-standard --fix
```

```
yarn add -D \  
  eslint-config-standard-with-typescript \  
  @typescript-eslint/parser \  
  @typescript-eslint/eslint-plugin \  
  eslint-plugin-import \  
  eslint-plugin-n \  
  eslint-plugin-promise  
  
# .eslintrc.json  
{  
  "extends": ["next/core-web-vitals", "standard-with-typescript"],  
  "parserOptions": {  
    "project": "./tsconfig.json"  
  }  
}
```

Prettier: performance at stake

Formatting rules:

- max-len
- no-mixed-spaces-and-tabs
- keyword-spacing
- comma-style
- ...



Code-quality rules:

- no-unused-vars
- no-extra-bind
- no-implicit-globals
- prefer-promise-reject-errors
- ...



Enabling Prettier with standard styles

Necessary packages:

- prettier – the tool itself
- prettier-config-standard – configures Prettier to match Standard style
- eslint-config-prettier – disables eslint styling rules (must come last)

Configure Prettier for Standard style:

Disable styling rules in `eslint`:

```
yarn add -D prettier \
  prettier-config-standard
  eslint-config-prettier
```

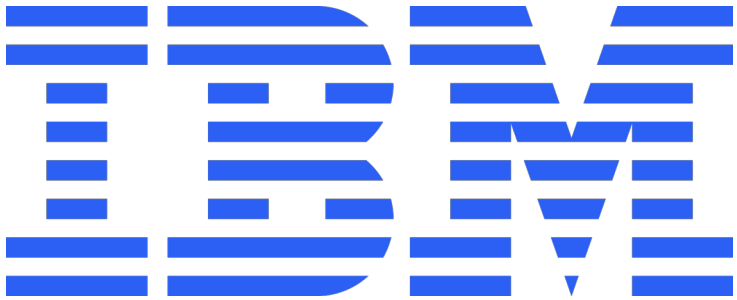
```
# package.json
{
  "scripts": {
    ...
    "format": "prettier --write '**/*.*(js|ts|tsx|json)'"
  }
  ...
  "prettier": "prettier-config-standard"
}
```

```
# .eslintrc.json
{
  "extends": ["next/core-web-vitals", "standard-with-typescript", "prettier"],
  "parserOptions": {
    "project": "./tsconfig.json"
  }
}
```


References

- ESLint: <https://eslint.org/>
- StandardJS: <https://standardjs.com/>
- Prettier: <https://prettier.io/>
- Going Beyond ESLint: An Overview of Static Analysis in JavaScript:
<https://www.telerik.com/blogs/going-beyond-eslint-overview-static-analysis-javascript>
- Feross Aboukhadijeh: Write Perfect Code With Standard And ESLint - JSConf.Asia 2018:
<https://www.youtube.com/watch?v=kuHfMw8j4xk>
- eslint-config-standard-with-typescript: <https://www.npmjs.com/package/eslint-config-standard-with-typescript>
- prettier-config-standard: <https://www.npmjs.com/package/prettier-config-standard>

Thank you.



© 2023 International Business Machines Corporation

IBM and the IBM logo are trademarks of IBM Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on [ibm.com/trademark](https://www.ibm.com/trademark).

THIS DOCUMENT IS DISTRIBUTED “AS IS” WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IN NO EVENT, SHALL IBM BE LIABLE FOR ANY DAMAGE ARISING FROM THE USE OF THIS INFORMATION, INCLUDING BUT NOT LIMITED TO, LOSS OF DATA, BUSINESS INTERRUPTION, LOSS OF PROFIT OR LOSS OF OPPORTUNITY.

Client examples are presented as illustrations of how those clients have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

Not all offerings are available in every country in which IBM operates.

Any statements regarding IBM’s future direction, intent or product plans are subject to change or withdrawal without notice.