ESLint

Since version **11.0.0**, Next.js provides an integrated ESLint experience out of the box. Add `next lint` as a script to `package.json`:

```
"scripts": {
  "lint": "next lint"
}
```

Then run `npm run lint` or `yarn lint`:

```
yarn lint
```

If you don't already have ESLint configured in your application, you will be guided through the installation and configuration process.

```
yarn lint

# You'll see a prompt like this:
#
# ? How would you like to configure ESLint?
#
# ❯   Base configuration + Core Web Vitals rule-set (recommended)
#     Base configuration
#     None
```

One of the following three options can be selected:

- **Strict**: Includes Next.js' base ESLint configuration along with a stricter Core Web Vitals rule-set. This is the recommended configuration for developers setting up ESLint for the first time.

  ```
  {
    "extends": "next/core-web-vitals"
  }
  ```

- **Base**: Includes Next.js' base ESLint configuration.

  ```
  {
    "extends": "next"
  }
  ```

- **Cancel**: Does not include any ESLint configuration. Only select this option if you plan on setting up your own custom ESLint configuration.

If either of the two configuration options are selected, Next.js will automatically install `eslint` and `eslint-config-next` as development dependencies in your application and create an `.eslintrc.json` file in the root of your project that includes your selected configuration.

You can now run `next lint` every time you want to run ESLint to catch errors. Once ESLint has been set up, it will also automatically run during every build (`next build`). Errors will fail the build, while warnings will not.

> *If you do not want ESLint to run during* `next build` *, refer to the documentation for [Ignoring ESLint](#).*

We recommend using an appropriate [integration](#) to view warnings and errors directly in your code editor during development.

ESLint Config

The default configuration (`eslint-config-next`) includes everything you need to have an optimal out-of-the-box linting experience in Next.js. If you do not have ESLint already configured in your application, we recommend using `next lint` to set up ESLint along with this configuration.

> *If you would like to use* `eslint-config-next` *along with other ESLint configurations, refer to the [Additional Configurations](#) section to learn how to do so without causing any conflicts.*

Recommended rule-sets from the following ESLint plugins are all used within `eslint-config-next` :

- [`eslint-plugin-react`](#)
- [`eslint-plugin-react-hooks`](#)
- [`eslint-plugin-next`](#)

You can see the full details of the shareable configuration in the [`eslint-config-next`](#) package.

This will take precedence over the configuration from `next.config.js` .

ESLint Plugin

Next.js provides an ESLint plugin, [`eslint-plugin-next`](#) , already bundled within the base configuration that makes it possible to catch common issues and problems in a Next.js application. The full set of rules is as follows:

	Rule	Description
✓	[next/google-font-display](#)	Enforce optional or swap font-display behavior with Google Fonts
✓	[next/google-font-preconnect](#)	Enforce preconnect usage with Google Fonts
✓	[next/link-passhref](#)	Enforce passHref prop usage with custom Link components
✓	[next/no-css-tags](#)	Prevent manual stylesheet tags
✓	[next/no-document-import-in-page](#)	Disallow importing next/document outside of pages/document.js
✓	[next/no-head-import-in-document](#)	Disallow importing next/head in pages/document.js
✓	[next/no-html-link-for-pages](#)	Prohibit HTML anchor links to pages without a Link component
✓	[next/no-img-element](#)	Prohibit usage of HTML <img> element
✓	[next/no-head-element](#)	Prohibit usage of HTML <head> element
✓	[next/no-page-custom-font](#)	Prevent page-only custom fonts

✓	next/no-sync-scripts	Forbid synchronous scripts
✓	next/no-title-in-document-head	Disallow using <title> with Head from next/document
✓	next/no-unwanted-polyfillio	Prevent duplicate polyfills from Polyfill.io
✓	next/inline-script-id	Enforce id attribute on next/script components with inline content
✓	next/no-typos	Ensure no typos were made declaring Next.js's data fetching function
✓	next/next-script-for-ga	Use the Script component to defer loading of the script until necessary.

- ✓: Enabled in the recommended configuration

If you already have ESLint configured in your application, we recommend extending from this plugin directly instead of including `eslint-config-next` unless a few conditions are met. Refer to the Recommended Plugin Ruleset to learn more.

Custom Settings

`rootDir`

If you're using `eslint-plugin-next` in a project where Next.js isn't installed in your root directory (such as a monorepo), you can tell `eslint-plugin-next` where to find your Next.js application using the `settings` property in your `.eslintrc` :

```
{
  "extends": "next",
  "settings": {
    "next": {
      "rootDir": "packages/my-app/"
    }
  }
}
```

`rootDir` can be a path (relative or absolute), a glob (i.e. `"packages/*/"`), or an array of paths and/or globs.

Linting Custom Directories and Files

By default, Next.js will run ESLint for all files in the `pages/` , `components/` , and `lib/` directories. However, you can specify which directories using the `dirs` option in the `eslint` config in `next.config.js` for production builds:

```
module.exports = {
  eslint: {
    dirs: ['pages', 'utils'], // Only run ESLint on the 'pages' and 'utils'
directories during production builds (next build)
  },
}
```

Similarly, the `--dir` and `--file` flags can be used for `next lint` to lint specific directories and files:

```
next lint --dir pages --dir utils --file bar.js
```

Caching

To improve performance, information of files processed by ESLint are cached by default. This is stored in `.next/cache` or in your defined build directory. If you include any ESLint rules that depend on more than the contents of a single source file and need to disable the cache, use the `--no-cache` flag with `next lint`.

```
next lint --no-cache
```

Disabling Rules

If you would like to modify or disable any rules provided by the supported plugins (`react` , `react-hooks` , `next`), you can directly change them using the `rules` property in your `.eslintrc` :

```
{
  "extends": "next",
  "rules": {
    "react/no-unescaped-entities": "off",
    "@next/next/no-page-custom-font": "off"
  }
}
```

Core Web Vitals

The `next/core-web-vitals` rule set is enabled when `next lint` is run for the first time and the **strict** option is selected.

```
{
  "extends": "next/core-web-vitals"
}
```

`next/core-web-vitals` updates `eslint-plugin-next` to error on a number of rules that are warnings by default if they affect Core Web Vitals.

> *The `next/core-web-vitals` entry point is automatically included for new applications built with Create Next App.*

Usage With Other Tools

Prettier

ESLint also contains code formatting rules, which can conflict with your existing Prettier setup. We recommend including eslint-config-prettier in your ESLint config to make ESLint and Prettier work together.

First, install the dependency:

```
npm install --save-dev eslint-config-prettier
# or
yarn add --dev eslint-config-prettier
```

Then, add `prettier` to your existing ESLint config:

```
{
  "extends": ["next", "prettier"]
}
```

lint-staged

If you would like to use `next lint` with lint-staged to run the linter on staged git files, you'll have to add the following to the `.lintstagedrc.js` file in the root of your project in order to specify usage of the `--file` flag.

```
const path = require('path')

const buildEslintCommand = (filenames) =>
  `next lint --fix --file ${filenames
    .map((f) => path.relative(process.cwd(), f))
    .join(' --file ')}`

module.exports = {
  '*.{js,jsx,ts,tsx}': [buildEslintCommand],
}
```

Migrating Existing Config

Recommended Plugin Ruleset

If you already have ESLint configured in your application and any of the following conditions are true:

- You have one or more of the following plugins already installed (either separately or through a different config such as `airbnb` or `react-app`):
  - `react`
  - `react-hooks`
  - `jsx-a11y`
  - `import`
- You've defined specific `parserOptions` that are different from how Babel is configured within Next.js (this is not recommended unless you have customized your Babel configuration)
- You have `eslint-plugin-import` installed with Node.js and/or TypeScript resolvers defined to handle imports

Then we recommend either removing these settings if you prefer how these properties have been configured within `eslint-config-next` or extending directly from the Next.js ESLint plugin instead:

```
module.exports = {
  extends: [
    //...
    'plugin:@next/next/recommended',
  ],
}
```

The plugin can be installed normally in your project without needing to run `next lint`:

```
npm install --save-dev @next/eslint-plugin-next
# or
yarn add --dev @next/eslint-plugin-next
```

This eliminates the risk of collisions or errors that can occur due to importing the same plugin or parser across multiple configurations.

Additional Configurations

If you already use a separate ESLint configuration and want to include `eslint-config-next`, ensure that it is extended last after other configurations. For example:

```
{
  "extends": ["eslint:recommended", "next"]
}
```

The `next` configuration already handles setting default values for the `parser`, `plugins` and `settings` properties. There is no need to manually re-declare any of these properties unless you need a different configuration for your use case. If you include any other shareable configurations, **you will need to make sure that these properties are not overwritten or modified**. Otherwise, we recommend removing any configurations that share behavior with the `next` configuration or extending directly from the Next.js ESLint plugin as mentioned above.