Setting up Prettier, ESLint, Airbnb, Stylelint, Husky, and lint-staged for a consistent Node.js workflow

Язык оригинала: en

# Оригинал

Whether you’re coding solo or working in a team, one thing becomes obvious fast: style inconsistencies and formatting nitpicks waste time.  
Over the years, I’ve landed on a setup that saves mental overhead, keeps the codebase clean, and scales well across teams and projects. In this guide, I’ll walk you through how I configure Prettier, ESLint with Airbnb’s style guide, Stylelint, Husky, and lint-staged in a Node.js environment.  
Let’s start with a high-level view of what each tool brings to the table.  
📦 Overview: What Each Tool Does (and Why You Need It)  
Tool  
What It Solves  
Prettier  
Automatically formats code to a consistent style across file types.  
ESLint  
Detects syntax errors and enforces coding best practices in JS/TS code.  
Airbnb Config  
A widely adopted ESLint config that enforces readable, consistent JS code.  
Stylelint  
Linter for CSS/SCSS that catches errors and enforces style conventions.  
lint-staged  
Ensures linters only run on staged files — fast and targeted.  
Husky  
Hooks into Git to run checks before code is committed, keeping your main branch clean.  
When combined, these tools reduce the surface area for bugs, reduce friction in code reviews, and let you focus on solving real problems — not arguing about trailing commas.  
🧹 Step 1: Prettier — Your Formatting Backbone  
Prettier is an opinionated formatter. It doesn’t care how you want your code to look — it enforces a consistent, readable style. It supports many file types: .html, .json, .js, .ts, .css, .scss, .md, etc.  
🔧 Install  
npm  
install  
--save-dev  
prettier  
Enter fullscreen mode  
Exit fullscreen mode  
Create a  
.prettierrc  
file at the root of your project. You can modify these settings as per your preferences. Refer to the  
Prettier  
documentation for all available options. Here's a sample configuration I use:  
{  
"bracketSpacing"  
:  
true  
,  
"printWidth"  
:  
120  
,  
"singleQuote"  
:  
true  
,  
"tabWidth"  
:  
2  
,  
"trailingComma"  
:  
"all"  
}  
Enter fullscreen mode  
Exit fullscreen mode  
Why these settings?  
bracketSpacing  
- Improves readability with space inside object literals:  
{ key: value }  
.  
printWidth  
- I find 80 too cramped; 120 gives more breathing room, especially for modern monitors.  
singleQuote  
- JavaScript traditionally uses single quotes; this reduces diff noise and aligns with many style guides.  
tabWidth  
- Standard size, especially in JS/TS communities.  
trailingComma  
- Makes diffs cleaner and reduces bugs when adding new lines.  
Create a  
.prettierignore  
file at the root of your project to exclude files/directories you don’t want Prettier to touch — generated files, IDE configs, etc.  
node\_modules  
build  
dist  
coverage  
.vscode  
Enter fullscreen mode  
Exit fullscreen mode  
🧠 Step 2: ESLint + Airbnb — Code Quality & Best Practices  
Where Prettier handles style, ESLint enforces logic and structure. It catches things like unused variables, accidental globals and subtle syntax issues.  
The Airbnb style guide is battle-tested and opinionated. It promotes clarity, consistency and best practices out of the box.  
🔧 Install dependencies  
npm  
install  
--save-dev  
eslint eslint-plugin-node eslint-config-node  
npm  
install  
--save  
-dev  
eslint-plugin-prettier eslint-config-prettier  
npx install-peerdeps  
--dev  
eslint-config-airbnb-base  
Enter fullscreen mode  
Exit fullscreen mode  
Note: Use  
eslint-config-airbnb  
instead of  
airbnb-base  
if you’re working with React.  
Create an  
.eslintrc.json  
file at the root of your project. Here's a sample configuration that uses the Airbnb base rules (which exclude React-specific rules). It also adds Prettier and node config as a rule. Check out  
ESLint  
docs for full sets of available rules:  
{  
"env"  
:  
{  
"browser"  
:  
true  
,  
"commonjs"  
:  
true  
,  
"es2021"  
:  
true  
},  
"extends"  
:  
[  
"airbnb-base"  
,  
"prettier"  
,  
"plugin:node/recommended"  
],  
"plugins"  
:  
[  
"prettier"  
],  
"rules"  
:  
{  
"prettier/prettier"  
:  
"warn"  
,  
"no-console"  
:  
"warn"  
,  
"no-unused-vars"  
:  
"warn"  
,  
"no-shadow"  
:  
"off"  
,  
"no-underscore-dangle"  
:  
"off"  
}  
}  
Enter fullscreen mode  
Exit fullscreen mode  
Key rules explained:  
prettier/prettier  
- ESLint surfaces Prettier issues, but only as warnings.  
no-console  
- Logs are useful during development, but shouldn’t ship to prod.  
no-unused-vars  
- Helps avoid bloated, misleading code.  
no-underscore-dangle  
- I often use \_id from MongoDB, so this rule gets turned off.  
🎨 Step 3: Stylelint — Keep Your Stylesheets Clean  
Stylelint works just like ESLint, but for styles. It ensures your SCSS/CSS is consistent, avoids deprecated syntax and promotes clean layout logic.  
🔧 Install  
npm  
install  
--save-dev  
sass stylelint stylelint-config-standard-scss stylelint-config-prettier  
Enter fullscreen mode  
Exit fullscreen mode  
Create  
.stylelintrc.json  
in the project root. Here's a sample configuration:  
{  
"extends"  
:  
[  
"stylelint-config-standard-scss"  
,  
"stylelint-config-prettier"  
],  
"rules"  
:  
{  
"color-function-notation"  
:  
"legacy"  
,  
"selector-class-pattern"  
:  
null  
}  
}  
Enter fullscreen mode  
Exit fullscreen mode  
What this does:  
standard-scss  
gives us a solid base config for SCSS.  
tylelint-config-prettier  
disables rules that conflict with Prettier.  
We relax strict class naming rules  
selector-class-pattern  
to suit different naming conventions (BEM, utility-first, etc.).  
⚡ Step 4: lint-staged — Lint Only What Changed  
Without lint-staged, Husky would run ESLint/Prettier on the whole codebase for every commit. That’s slow and unnecessary. lint-staged is a package that can be used to run formatting and linting commands on staged files in a Git repo.  
🔧 Install  
npm  
install  
--save-dev  
lint-staged  
Enter fullscreen mode  
Exit fullscreen mode  
lint-staged will run specific commands on staged files before committing them. Add the following configuration to your  
package.json  
. The below configures lint-staged to run Prettier, Stylelint and ESLint. Only add the Stylelint script for front-end projects:  
"lint-staged"  
:  
{  
"\*.{js,ts,jsx,tsx}"  
:  
[  
"prettier --write"  
,  
"eslint"  
],  
"\*.{css,scss,json,md}"  
:  
[  
"prettier --write"  
,  
"stylelint --allow-empty-input"  
]  
}  
Enter fullscreen mode  
Exit fullscreen mode  
This setup makes sure only staged files are linted/formatted, keeping pre-commit hooks fast and focused.  
🔐 Step 5: Husky — Git Gatekeeper  
Husky  
allows us to run scripts at key Git lifecycle moments. Here, we’ll use it to run lint-staged before any commit. Thus code only ever gets into the repo after it has been consistently formatted and verified to be free of linting errors. This is a particularly big advantage in a team setting.  
🔧 Install and set up  
npm  
install  
--save-dev  
husky  
npx husky init  
Enter fullscreen mode  
Exit fullscreen mode  
The  
init  
command simplifies setting up Husky in a project. It creates a  
pre-commit  
script in  
.husky/  
and updates the  
prepare  
script in  
package.json  
. Modifications can be made later to suit your workflow.  
Update the  
.husky/pre-commit  
file:  
npx lint-staged  
Enter fullscreen mode  
Exit fullscreen mode  
This ensures your staged files pass all checks before being committed.  
Create  
.husky/install.mjs  
file:  
// Skip Husky install in production and CI  
if  
(  
process  
.  
env  
.  
NODE\_ENV  
===  
'  
production  
'  
||  
process  
.  
env  
.  
CI  
===  
'  
true  
'  
)  
{  
process  
.  
exit  
(  
0  
);  
}  
const  
husky  
=  
(  
await  
import  
(  
'  
husky  
'  
)).  
default  
;  
console  
.  
log  
(  
husky  
());  
Enter fullscreen mode  
Exit fullscreen mode  
This ensures husky is only installed in dev environment.  
🛠️ Step 6: Helpful NPM Scripts  
Add these to your  
package.json  
for manual use or CI pipelines, adjust according to your project setup:  
"scripts"  
:  
{  
"lint"  
:  
"eslint './\*\*/\*.@(js|jsx|ts|tsx)'"  
,  
"lint:fix"  
:  
"eslint './\*\*/\*.@(js|jsx|ts|tsx)' --fix"  
,  
"prettier:check"  
:  
"prettier --check './\*\*/\*.{js,ts,json,md,css,scss}'"  
,  
"prettier:write"  
:  
"prettier --write './\*\*/\*.{js,ts,json,md,css,scss}'"  
,  
"style:check"  
:  
"stylelint '\*\*/\*.{css,scss}' --allow-empty-input"  
,  
"prepare"  
:  
"node .husky/install.mjs"  
}  
Enter fullscreen mode  
Exit fullscreen mode  
lint  
- This script will run ESLint on all JavaScript, TypeScript, JSX and TSX files in your project.  
lint:fix  
- This will do the same as the lint script but will also attempt to automatically fix any issues that ESLint can handle.  
prettier:write  
- This script will format all JS, JSX, TS, TSX, JSON, CSS, SCSS and Markdown files using Prettier. This ensures consistency in both code and documentation.  
prettier:check  
- This will check if all the specified files adhere to the formatting rules specified by Prettier. This is especially useful as a pre-commit or CI/CD step to ensure that no non-formatted files get committed or deployed.  
style:check  
- This will check if all the specified CSS, SCSS files adhere to the formatting rules specified by Stylelint. Only add for front-end projects.  
prepare  
- This will only run Husky in a non-production environment.  
You can adjust the patterns like  
./\*\*/\*.@(js|jsx|ts|tsx|json|css|scss|md)  
to fit the specific needs of your project, and which directories or file types you want to include/exclude.  
⚙️ Using in your Workflow  
Whenever you're about to commit, you can run:  
git add  
.  
git commit  
-m  
"Set up linting and formatting"  
# lint-staged script will run every time you commit  
Enter fullscreen mode  
Exit fullscreen mode  
This will ensure that your code is both linted and formatted. If anything had not been set up correctly, you would likely get errors during commit.  
🧩 Bonus: VS Code Setup for Auto-Formatting  
To make all of this seamless, configure VS Code to format on save and use the following extensions:  
ESLint  
Stylelint  
Prettier - Code formatter  
Put the following in  
.vscode/settings.json  
in the project (you can also put these settings in your User Preferences file):  
{  
"[javascript]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"typescript.tsdk"  
:  
"node\_modules/typescript/lib"  
,  
"[typescript]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"[javascriptreact]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"[typescriptreact]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"[scss]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"scss.lint.emptyRules"  
:  
"ignore"  
,  
"stylelint.validate"  
:  
[  
"css"  
,  
"scss"  
],  
"editor.formatOnSave"  
:  
true  
}  
Enter fullscreen mode  
Exit fullscreen mode  
The above set up, will allow the extensions to lint and format on Save.  
✅ Final Thoughts  
This setup may feel like a lot up front, but once it’s in place, it quietly handles all the repetitive formatting, catches subtle issues and keeps your Git history clean.  
It’s not about obsessing over code style. It’s about freeing up mental bandwidth so you can focus on solving real problems, not arguing over trailing commas or indentation.  
🤔 What Do You Use?  
I’d love to hear how others approach this:  
What’s your go-to setup?  
Any must-disable rules in your projects?  
Do you skip this entirely for small experiments?  
Let me know your take. Always curious to hear how other devs keep their code clean.  
🛠️ Written by  
Sakib Miyahn  
- Full-Stack Software Engineer - Sydney, AU.

# Перевод на русский

Whether you’re coding solo or working in a team, one thing becomes obvious fast: style inconsistencies and formatting nitpicks waste time.  
Over the years, I’ve landed on a setup that saves mental overhead, keeps the codebase clean, and scales well across teams and projects. In this guide, I’ll walk you through how I configure Prettier, ESLint with Airbnb’s style guide, Stylelint, Husky, and lint-staged in a Node.js environment.  
Let’s start with a high-level view of what each tool brings to the table.  
📦 Overview: What Each Tool Does (and Why You Need It)  
Tool  
What It Solves  
Prettier  
Automatically formats code to a consistent style across file types.  
ESLint  
Detects syntax errors and enforces coding best practices in JS/TS code.  
Airbnb Config  
A widely adopted ESLint config that enforces readable, consistent JS code.  
Stylelint  
Linter for CSS/SCSS that catches errors and enforces style conventions.  
lint-staged  
Ensures linters only run on staged files — fast and targeted.  
Husky  
Hooks into Git to run checks before code is committed, keeping your main branch clean.  
When combined, these tools reduce the surface area for bugs, reduce friction in code reviews, and let you focus on solving real problems — not arguing about trailing commas.  
🧹 Step 1: Prettier — Your Formatting Backbone  
Prettier is an opinionated formatter. It doesn’t care how you want your code to look — it enforces a consistent, readable style. It supports many file types: .html, .json, .js, .ts, .css, .scss, .md, etc.  
🔧 Install  
npm  
install  
--save-dev  
prettier  
Enter fullscreen mode  
Exit fullscreen mode  
Create a  
.prettierrc  
file at the root of your project. You can modify these settings as per your preferences. Refer to the  
Prettier  
documentation for all available options. Here's a sample configuration I use:  
{  
"bracketSpacing"  
:  
true  
,  
"printWidth"  
:  
120  
,  
"singleQuote"  
:  
true  
,  
"tabWidth"  
:  
2  
,  
"trailingComma"  
:  
"all"  
}  
Enter fullscreen mode  
Exit fullscreen mode  
Why these settings?  
bracketSpacing  
- Improves readability with space inside object literals:  
{ key: value }  
.  
printWidth  
- I find 80 too cramped; 120 gives more breathing room, especially for modern monitors.  
singleQuote  
- JavaScript traditionally uses single quotes; this reduces diff noise and aligns with many style guides.  
tabWidth  
- Standard size, especially in JS/TS communities.  
trailingComma  
- Makes diffs cleaner and reduces bugs when adding new lines.  
Create a  
.prettierignore  
file at the root of your project to exclude files/directories you don’t want Prettier to touch — generated files, IDE configs, etc.  
node\_modules  
build  
dist  
coverage  
.vscode  
Enter fullscreen mode  
Exit fullscreen mode  
🧠 Step 2: ESLint + Airbnb — Code Quality & Best Practices  
Where Prettier handles style, ESLint enforces logic and structure. It catches things like unused variables, accidental globals and subtle syntax issues.  
The Airbnb style guide is battle-tested and opinionated. It promotes clarity, consistency and best practices out of the box.  
🔧 Install dependencies  
npm  
install  
--save-dev  
eslint eslint-plugin-node eslint-config-node  
npm  
install  
--save  
-dev  
eslint-plugin-prettier eslint-config-prettier  
npx install-peerdeps  
--dev  
eslint-config-airbnb-base  
Enter fullscreen mode  
Exit fullscreen mode  
Note: Use  
eslint-config-airbnb  
instead of  
airbnb-base  
if you’re working with React.  
Create an  
.eslintrc.json  
file at the root of your project. Here's a sample configuration that uses the Airbnb base rules (which exclude React-specific rules). It also adds Prettier and node config as a rule. Check out  
ESLint  
docs for full sets of available rules:  
{  
"env"  
:  
{  
"browser"  
:  
true  
,  
"commonjs"  
:  
true  
,  
"es2021"  
:  
true  
},  
"extends"  
:  
[  
"airbnb-base"  
,  
"prettier"  
,  
"plugin:node/recommended"  
],  
"plugins"  
:  
[  
"prettier"  
],  
"rules"  
:  
{  
"prettier/prettier"  
:  
"warn"  
,  
"no-console"  
:  
"warn"  
,  
"no-unused-vars"  
:  
"warn"  
,  
"no-shadow"  
:  
"off"  
,  
"no-underscore-dangle"  
:  
"off"  
}  
}  
Enter fullscreen mode  
Exit fullscreen mode  
Key rules explained:  
prettier/prettier  
- ESLint surfaces Prettier issues, but only as warnings.  
no-console  
- Logs are useful during development, but shouldn’t ship to prod.  
no-unused-vars  
- Helps avoid bloated, misleading code.  
no-underscore-dangle  
- I often use \_id from MongoDB, so this rule gets turned off.  
🎨 Step 3: Stylelint — Keep Your Stylesheets Clean  
Stylelint works just like ESLint, but for styles. It ensures your SCSS/CSS is consistent, avoids deprecated syntax and promotes clean layout logic.  
🔧 Install  
npm  
install  
--save-dev  
sass stylelint stylelint-config-standard-scss stylelint-config-prettier  
Enter fullscreen mode  
Exit fullscreen mode  
Create  
.stylelintrc.json  
in the project root. Here's a sample configuration:  
{  
"extends"  
:  
[  
"stylelint-config-standard-scss"  
,  
"stylelint-config-prettier"  
],  
"rules"  
:  
{  
"color-function-notation"  
:  
"legacy"  
,  
"selector-class-pattern"  
:  
null  
}  
}  
Enter fullscreen mode  
Exit fullscreen mode  
What this does:  
standard-scss  
gives us a solid base config for SCSS.  
tylelint-config-prettier  
disables rules that conflict with Prettier.  
We relax strict class naming rules  
selector-class-pattern  
to suit different naming conventions (BEM, utility-first, etc.).  
⚡ Step 4: lint-staged — Lint Only What Changed  
Without lint-staged, Husky would run ESLint/Prettier on the whole codebase for every commit. That’s slow and unnecessary. lint-staged is a package that can be used to run formatting and linting commands on staged files in a Git repo.  
🔧 Install  
npm  
install  
--save-dev  
lint-staged  
Enter fullscreen mode  
Exit fullscreen mode  
lint-staged will run specific commands on staged files before committing them. Add the following configuration to your  
package.json  
. The below configures lint-staged to run Prettier, Stylelint and ESLint. Only add the Stylelint script for front-end projects:  
"lint-staged"  
:  
{  
"\*.{js,ts,jsx,tsx}"  
:  
[  
"prettier --write"  
,  
"eslint"  
],  
"\*.{css,scss,json,md}"  
:  
[  
"prettier --write"  
,  
"stylelint --allow-empty-input"  
]  
}  
Enter fullscreen mode  
Exit fullscreen mode  
This setup makes sure only staged files are linted/formatted, keeping pre-commit hooks fast and focused.  
🔐 Step 5: Husky — Git Gatekeeper  
Husky  
allows us to run scripts at key Git lifecycle moments. Here, we’ll use it to run lint-staged before any commit. Thus code only ever gets into the repo after it has been consistently formatted and verified to be free of linting errors. This is a particularly big advantage in a team setting.  
🔧 Install and set up  
npm  
install  
--save-dev  
husky  
npx husky init  
Enter fullscreen mode  
Exit fullscreen mode  
The  
init  
command simplifies setting up Husky in a project. It creates a  
pre-commit  
script in  
.husky/  
and updates the  
prepare  
script in  
package.json  
. Modifications can be made later to suit your workflow.  
Update the  
.husky/pre-commit  
file:  
npx lint-staged  
Enter fullscreen mode  
Exit fullscreen mode  
This ensures your staged files pass all checks before being committed.  
Create  
.husky/install.mjs  
file:  
// Skip Husky install in production and CI  
if  
(  
process  
.  
env  
.  
NODE\_ENV  
===  
'  
production  
'  
||  
process  
.  
env  
.  
CI  
===  
'  
true  
'  
)  
{  
process  
.  
exit  
(  
0  
);  
}  
const  
husky  
=  
(  
await  
import  
(  
'  
husky  
'  
)).  
default  
;  
console  
.  
log  
(  
husky  
());  
Enter fullscreen mode  
Exit fullscreen mode  
This ensures husky is only installed in dev environment.  
🛠️ Step 6: Helpful NPM Scripts  
Add these to your  
package.json  
for manual use or CI pipelines, adjust according to your project setup:  
"scripts"  
:  
{  
"lint"  
:  
"eslint './\*\*/\*.@(js|jsx|ts|tsx)'"  
,  
"lint:fix"  
:  
"eslint './\*\*/\*.@(js|jsx|ts|tsx)' --fix"  
,  
"prettier:check"  
:  
"prettier --check './\*\*/\*.{js,ts,json,md,css,scss}'"  
,  
"prettier:write"  
:  
"prettier --write './\*\*/\*.{js,ts,json,md,css,scss}'"  
,  
"style:check"  
:  
"stylelint '\*\*/\*.{css,scss}' --allow-empty-input"  
,  
"prepare"  
:  
"node .husky/install.mjs"  
}  
Enter fullscreen mode  
Exit fullscreen mode  
lint  
- This script will run ESLint on all JavaScript, TypeScript, JSX and TSX files in your project.  
lint:fix  
- This will do the same as the lint script but will also attempt to automatically fix any issues that ESLint can handle.  
prettier:write  
- This script will format all JS, JSX, TS, TSX, JSON, CSS, SCSS and Markdown files using Prettier. This ensures consistency in both code and documentation.  
prettier:check  
- This will check if all the specified files adhere to the formatting rules specified by Prettier. This is especially useful as a pre-commit or CI/CD step to ensure that no non-formatted files get committed or deployed.  
style:check  
- This will check if all the specified CSS, SCSS files adhere to the formatting rules specified by Stylelint. Only add for front-end projects.  
prepare  
- This will only run Husky in a non-production environment.  
You can adjust the patterns like  
./\*\*/\*.@(js|jsx|ts|tsx|json|css|scss|md)  
to fit the specific needs of your project, and which directories or file types you want to include/exclude.  
⚙️ Using in your Workflow  
Whenever you're about to commit, you can run:  
git add  
.  
git commit  
-m  
"Set up linting and formatting"  
# lint-staged script will run every time you commit  
Enter fullscreen mode  
Exit fullscreen mode  
This will ensure that your code is both linted and formatted. If anything had not been set up correctly, you would likely get errors during commit.  
🧩 Bonus: VS Code Setup for Auto-Formatting  
To make all of this seamless, configure VS Code to format on save and use the following extensions:  
ESLint  
Stylelint  
Prettier - Code formatter  
Put the following in  
.vscode/settings.json  
in the project (you can also put these settings in your User Preferences file):  
{  
"[javascript]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"typescript.tsdk"  
:  
"node\_modules/typescript/lib"  
,  
"[typescript]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"[javascriptreact]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"[typescriptreact]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"[scss]"  
:  
{  
"editor.defaultFormatter"  
:  
"esbenp.prettier-vscode"  
},  
"scss.lint.emptyRules"  
:  
"ignore"  
,  
"stylelint.validate"  
:  
[  
"css"  
,  
"scss"  
],  
"editor.formatOnSave"  
:  
true  
}  
Enter fullscreen mode  
Exit fullscreen mode  
The above set up, will allow the extensions to lint and format on Save.  
✅ Final Thoughts  
This setup may feel like a lot up front, but once it’s in place, it quietly handles all the repetitive formatting, catches subtle issues and keeps your Git history clean.  
It’s not about obsessing over code style. It’s about freeing up mental bandwidth so you can focus on solving real problems, not arguing over trailing commas or indentation.  
🤔 What Do You Use?  
I’d love to hear how others approach this:  
What’s your go-to setup?  
Any must-disable rules in your projects?  
Do you skip this entirely for small experiments?  
Let me know your take. Always curious to hear how other devs keep their code clean.  
🛠️ Written by  
Sakib Miyahn  
- Full-Stack Software Engineer - Sydney, AU.