Prettier stable

Search

Playground Docs Blog Donate GitHub

› Configuring Prettier

Options

EDIT

Prettier ships with a handful of format options.

To learn more about Prettier’s stance on options – see the Option Philosophy.

If you change any options, it’s recommended to do it via a configuration file. This way the Prettier CLI, editor integrations and other tooling knows what options you use.

Experimental Ternaries

Try prettier's new ternary formatting before it becomes the default behavior.

Valid options:

- `true` - Use curious ternaries, with the question mark after the condition.
- `false` - Retain the default behavior of ternaries; keep question marks on the same line as the consequent.

Default	CLI Override	API Override
`false`	`--experimental-ternaries`	`experimentalTernaries: <bool>`

Print Width

Specify the line length that the printer will wrap on.

> **For readability we recommend against using more than 80 characters:**
>
> In code styleguides, maximum line length rules are often set to 100 or 120. However, when humans write code, they don’t strive to reach the maximum number of columns on every

> line length limit. It is a way to say to Prettier roughly how long you'd like lines to be. Prettier will make both shorter and longer lines, but generally strive to meet the specified printWidth.
>
> Remember, computers are dumb. You need to explicitly tell them what to do, while humans can make their own (implicit) judgements, for example on when to break a line.
>
> In other words, don't try to use printWidth as if it was ESLint's max-len – they're not the same. max-len just says what the maximum allowed line length is, but not what the generally preferred length is – which is what printWidth specifies.

Default	CLI Override	API Override
`80`	`--print-width <int>`	`printWidth: <int>`

Setting `max_line_length` in an `.editorconfig` file will configure Prettier's print width, unless overridden.

(If you don't want line wrapping when formatting Markdown, you can set the Prose Wrap option to disable it.)

Tab Width

Specify the number of spaces per indentation-level.

Default	CLI Override	API Override
`2`	`--tab-width <int>`	`tabWidth: <int>`

Setting `indent_size` or `tab_width` in an `.editorconfig` file will configure Prettier's tab width, unless overridden.

Tabs

Indent lines with tabs instead of spaces.

Setting `indent_style` in an `.editorconfig` file will configure Prettier's tab usage, unless overridden.

(Tabs will be used for *indentation* but Prettier uses spaces to *align* things, such as in ternaries. This behavior is known as SmartTabs.)

Semicolons

Print semicolons at the ends of statements.

Valid options:

- `true` - Add a semicolon at the end of every statement.
- `false` - Only add semicolons at the beginning of lines that may introduce ASI failures.

Default	CLI Override	API Override
`true`	`--no-semi`	`semi: <bool>`

Quotes

Use single quotes instead of double quotes.

Notes:

- JSX quotes ignore this option – see jsx-single-quote.
- If the number of quotes outweighs the other quote, the quote which is less used will be used to format the string - Example: `"I'm double quoted"` results in `"I'm double quoted"` and `"This \"example\" is single quoted"` results in `'This "example" is single quoted'` .

See the strings rationale for more information.

Default	CLI Override	API Override
`false`	`--single-quote`	`singleQuote: <bool>`

Quote Props

- `"consistent"` - If at least one property in an object requires quotes, quote all properties.
- `"preserve"` - Respect the input use of quotes in object properties.

<table>
<tr><th>Default</th><th>CLI Override</th><th>API Override</th></tr>
<tr><td><code>"as-needed"</code></td><td><code>--quote-props <as-needed|consistent|preserve&gt;</code></td><td><code>quoteProps: "<as-needed|consistent|preserve&gt;"</code></td></tr>
</table>

Note that Prettier never unquotes numeric property names in Angular expressions, TypeScript, and Flow because the distinction between string and numeric keys is significant in these languages. See: Angular, TypeScript, Flow. Also Prettier doesn’t unquote numeric properties for Vue (see the issue about that).

JSX Quotes

Use single quotes instead of double quotes in JSX.

Default	CLI Override	API Override
`false`	`--jsx-single-quote`	`jsxSingleQuote: <bool>`

Trailing Commas

Default value changed from `es5` to `all` in v3.0.0

Print trailing commas wherever possible in multi-line comma-separated syntactic structures. (A single-line array, for example, never gets trailing commas.)

Valid options:

- `"all"` - Trailing commas wherever possible (including function parameters and calls). To run, JavaScript code formatted this way needs an engine that supports ES2017 (Node.js 8+ or a modern browser) or downlevel compilation. This also enables trailing commas in type parameters in TypeScript (supported since TypeScript 2.7 released in January 2018).
- `"es5"` - Trailing commas where valid in ES5 (objects, arrays, etc.). Trailing commas in type parameters in TypeScript and Flow.

`"all"`	`--trailing-comma <all|es5|none>`	`trailingComma: "<all|es5|none>"`

Bracket Spacing

Print spaces between brackets in object literals.

Valid options:

- `true` - Example: `{ foo: bar }`.
- `false` - Example: `{foo: bar}`.

Default	CLI Override	API Override
`true`	`--no-bracket-spacing`	`bracketSpacing: <bool>`

Bracket Line

Put the `>` of a multi-line HTML (HTML, JSX, Vue, Angular) element at the end of the last line instead of being alone on the next line (does not apply to self closing elements).

Valid options:

- `true` - Example:

```
<button
  className="prettier-class"
  id="prettier-id"
  onClick={this.handleClick}>
  Click Here
</button>
```

- `false` - Example:

```
<button
  className="prettier-class"
  id="prettier-id"
  onClick={this.handleClick}
>
```

`false`	`--bracket-same-line`	`bracketSameLine: <bool>`

[Deprecated] JSX Brackets

This option has been deprecated in v2.4.0, use --bracket-same-line instead

Put the `>` of a multi-line JSX element at the end of the last line instead of being alone on the next line (does not apply to self closing elements).

Valid options:

- `true` - Example:

```
<button
  className="prettier-class"
  id="prettier-id"
  onClick={this.handleClick}>
  Click Here
</button>
```

- `false` - Example:

```
<button
  className="prettier-class"
  id="prettier-id"
  onClick={this.handleClick}
>
  Click Here
</button>
```

Default	CLI Override	API Override
`false`	`--jsx-bracket-same-line`	`jsxBracketSameLine: <bool>`

Arrow Function Parentheses

.

- `"always"` - Always include parens. Example: `(x) => x`
- `"avoid"` - Omit parens when possible. Example: `x => x`

Default	CLI Override	API Override
`"always"`	`--arrow-parens <always|avoid>`	`arrowParens: "<always|avoid>"`

At first glance, avoiding parentheses may look like a better choice because of less visual noise. However, when Prettier removes parentheses, it becomes harder to add type annotations, extra arguments or default values as well as making other changes. Consistent use of parentheses provides a better developer experience when editing real codebases, which justifies the default value for the option.

Range

Format only a segment of a file.

These two options can be used to format code starting and ending at a given character offset (inclusive and exclusive, respectively). The range will extend:

- Backwards to the start of the first line containing the selected statement.
- Forwards to the end of the selected statement.

Default	CLI Override	API Override
`0`	`--range-start <int>`	`rangeStart: <int>`
`Infinity`	`--range-end <int>`	`rangeEnd: <int>`

Parser

Specify which parser to use.

Prettier automatically infers the parser from the input file path, so you shouldn’t have to change this setting.

when it comes to invalid code and less battle-tested than the `typescript` parser.

Valid options:

- `"babel"` (via @babel/parser) *Named `"babylon"` until v1.16.0*
- `"babel-flow"` (same as `"babel"` but enables Flow parsing explicitly to avoid ambiguity) *First available in v1.16.0*
- `"babel-ts"` (similar to `"typescript"` but uses Babel and its TypeScript plugin) *First available in v2.0.0*
- `"flow"` (via flow-parser)
- `"typescript"` (via @typescript-eslint/typescript-estree) *First available in v1.4.0*
- `"espree"` (via espree) *First available in v2.2.0*
- `"meriyah"` (via meriyah) *First available in v2.2.0*
- `"acorn"` (via acorn) *First available in v2.6.0*
- `"css"` (via postcss) *First available in v1.7.1*
- `"scss"` (via postcss-scss) *First available in v1.7.1*
- `"less"` (via postcss-less) *First available in v1.7.1*
- `"json"` (via @babel/parser parseExpression) *First available in v1.5.0*
- `"json5"` (same parser as `"json"`, but outputs as json5) *First available in v1.13.0*
- `"jsonc"` (same parser as `"json"`, but outputs as "JSON with Comments") *First available in v3.2.0*
- `"json-stringify"` (same parser as `"json"`, but outputs like `JSON.stringify`) *First available in v1.13.0*
- `"graphql"` (via graphql/language) *First available in v1.5.0*
- `"markdown"` (via remark-parse) *First available in v1.8.0*
- `"mdx"` (via remark-parse and @mdx-js/mdx) *First available in v1.15.0*
- `"html"` (via angular-html-parser) *First available in 1.15.0*
- `"vue"` (same parser as `"html"`, but also formats vue-specific syntax) *First available in 1.10.0*
- `"angular"` (same parser as `"html"`, but also formats angular-specific syntax via angular-estree-parser) *First available in 1.15.0*
- `"lwc"` (same parser as `"html"`, but also formats LWC-specific syntax for unquoted template attributes) *First available in 1.17.0*
- `"yaml"` (via yaml and yaml-unist-parser) *First available in 1.14.0*

Note: the default value was `"babylon"` until v1.13.0.

Note: the Custom parser API has been removed in v3.0.0. Use plugins instead (how to migrate).

File Path

Specify the file name to use to infer which parser to use.

For example, the following will use the CSS parser:

```
cat foo | prettier --stdin-filepath foo.css
```

This option is only useful in the CLI and API. It doesn’t make sense to use it in a configuration file.

Default	CLI Override	API Override
None	`--stdin-filepath <string>`	`filepath: "<string>"`

Require Pragma

First available in v1.7.0

Prettier can restrict itself to only format files that contain a special comment, called a pragma, at the top of the file. This is very useful when gradually transitioning large, unformatted codebases to Prettier.

A file with the following as its first comment will be formatted when `--require-pragma` is supplied:

```
/**
 * @prettier
 */
```

or

```
/**
 * @format
```

`false`	`--require-pragma`	`requirePragma: <bool>`

Insert Pragma

First available in v1.8.0

Prettier can insert a special `@format` marker at the top of files specifying that the file has been formatted with Prettier. This works well when used in tandem with the `--require-pragma` option. If there is already a docblock at the top of the file then this option will add a newline to it with the `@format` marker.

Note that “in tandem” doesn’t mean “at the same time”. When the two options are used simultaneously, `--require-pragma` has priority, so `--insert-pragma` is ignored. The idea is that during an incremental adoption of Prettier in a big codebase, the developers participating in the transition process use `--insert-pragma` whereas `--require-pragma` is used by the rest of the team and automated tooling to process only files already transitioned. The feature has been inspired by Facebook’s adoption strategy.

Default	CLI Override	API Override
`false`	`--insert-pragma`	`insertPragma: <bool>`

Prose Wrap

First available in v1.8.2

By default, Prettier will not change wrapping in markdown text since some services use a linebreak-sensitive renderer, e.g. GitHub comments and BitBucket. To have Prettier wrap prose to the print width, change this option to "always". If you want Prettier to force all prose blocks to be on a single line and rely on editor/viewer soft wrapping instead, you can use `"never"` .

Valid options:

- `"always"` - Wrap prose if it exceeds the print width.
- `"never"` - Un-wrap each block of prose into one line.
- `"preserve"` - Do nothing, leave prose as-is. *First available in v1.9.0*

HTML Whitespace Sensitivity

First available in v1.15.0. First available for Handlebars in 2.3.0

Specify the global whitespace sensitivity for HTML, Vue, Angular, and Handlebars. See whitespace-sensitive formatting for more info.

Valid options:

- `"css"` - Respect the default value of CSS `display` property. For Handlebars treated same as `strict` .
- `"strict"` - Whitespace (or the lack of it) around all tags is considered significant.
- `"ignore"` - Whitespace (or the lack of it) around all tags is considered insignificant.

Default	CLI Override	API Override
`"css"`	`--html-whitespace-sensitivity <css|strict|ignore>`	`htmlWhitespaceSensitivity: "<css|strict|ignore>"`

Vue files script and style tags indentation

First available in v1.19.0

Whether or not to indent the code inside `<script>` and `<style>` tags in Vue files.

Valid options:

- `false` - Do not indent script and style tags in Vue files.
- `true` - Indent script and style tags in Vue files.

Default	CLI Override	API Override
`false`	`--vue-indent-script-and-style`	`vueIndentScriptAndStyle: <bool>`

End of Line

Linux and macOS, while the latter is prevalent on Windows. Some details explaining why it is so can be found on Wikipedia.

When people collaborate on a project from different operating systems, it becomes easy to end up with mixed line endings in a shared git repository. It is also possible for Windows users to accidentally change line endings in a previously committed file from `LF` to `CRLF`. Doing so produces a large `git diff` and thus makes the line-by-line history for a file (`git blame`) harder to explore.

If you want to make sure that your entire git repository only contains Linux-style line endings in files covered by Prettier:

1. Ensure Prettier's `endOfLine` option is set to `lf` (this is a default value since v2.0.0)
2. Configure a pre-commit hook that will run Prettier
3. Configure Prettier to run in your CI pipeline using `--check` flag. If you use Travis CI, set the `autocrlf` option to `input` in `.travis.yml`.
4. Add `* text=auto eol=lf` to the repo's `.gitattributes` file. You may need to ask Windows users to re-clone your repo after this change to ensure git has not converted `LF` to `CRLF` on checkout.

All modern text editors in all operating systems are able to correctly display line endings when `\n` (`LF`) is used. However, old versions of Notepad for Windows will visually squash such lines into one as they can only deal with `\r\n` (`CRLF`).

Valid options:

- `"lf"` – Line Feed only (`\n`), common on Linux and macOS as well as inside git repos
- `"crlf"` - Carriage Return + Line Feed characters (`\r\n`), common on Windows
- `"cr"` - Carriage Return character only (`\r`), used very rarely
- `"auto"` - Maintain existing line endings (mixed values within one file are normalised by looking at what's used after the first line)

Default	CLI Override	API Override
`"lf"`	`--end-of-line <lf|crlf|cr|auto>`	`endOfLine: "<lf|crlf|cr|auto>"`

Setting `end_of_line` in an `.editorconfig` file will configure Prettier's end of line usage, unless overridden.

Control whether Prettier formats quoted code embedded in the file.

When Prettier identifies cases where it looks like you've placed some code it knows how to format within a string in another file, like in a tagged template in JavaScript with a tag named `html` or in code blocks in Markdown, it will by default try to format that code.

Sometimes this behavior is undesirable, particularly in cases where you might not have intended the string to be interpreted as code. This option allows you to switch between the default behavior (`auto`) and disabling this feature entirely (`off`).

Valid options:

- `"auto"` – Format embedded code if Prettier can automatically identify it.
- `"off"` - Never automatically format embedded code.

Default	CLI Override	API Override
`"auto"`	`--embedded-language-formatting=<off|auto>`	`embeddedLanguageFormatting: "<off|auto>"`

Single Attribute Per Line

First available in v2.6.0

Enforce single attribute per line in HTML, Vue and JSX.

Valid options:

- `false` - Do not enforce single attribute per line.
- `true` - Enforce single attribute per line.

Default	CLI Override	API Override
`false`	`--single-attribute-per-line`	`singleAttributePerLine: <bool>`

← BROWSER

CONFIGURATION FILE →

Prettier stable

Playground Docs Blog Donate GitHub

Usage

Stack Overflow

@PrettierCode on Twitter

Follow Prettier

GitHub

Issues

Star 48,911