Command Line Interface

ESLint requires Node.js for installation. Follow the instructions in the Getting Started Guide (https://eslint.org/docs/user-guide/getting-started) to install ESLint.

Most users use `npx` (https://docs.npmjs.com/cli/v8/commands/npx) to run ESLint on the command line like this:

```
npx eslint [options] [file|dir|glob]*
```

Such as:

```
# Run on two files
npx eslint file1.js file2.js

# Run on multiple files
npx eslint lib/**
```

Please note that when passing a glob as a parameter, it will be expanded by your shell. The results of the expansion can vary depending on your shell, and its configuration. If you want to use node `glob` syntax, you have to quote your parameter (using double quotes if you need it to run in Windows), as follows:

```
npx eslint "lib/**"
```

Note: You can also use alternative package managers such as Yarn (https://yarnpkg.com/) or pnpm (https://pnpm.io/) to run ESLint. Please refer to your package manager's documentation for the correct syntax.

Options

The command line utility has several options. You can view the options by running `npx eslint -h`.

```
eslint [options] file.js [file.js] [dir]

Basic configuration:
  --no-eslintrc                   Disable use of configuration from .eslintr
c.*
  -c, --config path::String       Use this configuration, overriding .eslintr
c.* config options if present
  --env [String]                  Specify environments
  --ext [String]                  Specify JavaScript file extensions
  --global [String]               Define global variables
  --parser String                 Specify the parser to be used
  --parser-options Object         Specify parser options
  --resolve-plugins-relative-to path::String  A folder where plugins should b
e resolved from, CWD by default

Specifying rules and plugins:
  --plugin [String]               Specify plugins
  --rule Object                   Specify rules
  --rulesdir [path::String]       Load additional rules from this directory.
Deprecated: Use rules from plugins

Fixing problems:
  --fix                           Automatically fix problems
  --fix-dry-run                   Automatically fix problems without saving t
he changes to the file system
  --fix-type Array                Specify the types of fixes to apply (direct
ive, problem, suggestion, layout)

Ignoring files:
  --ignore-path path::String      Specify path of ignore file
  --no-ignore                     Disable use of ignore files and patterns
  --ignore-pattern [String]       Pattern of files to ignore (in addition to
those in .eslintignore)

Using stdin:
  --stdin                         Lint code provided on <STDIN> - default: fa
lse
  --stdin-filename String         Specify filename to process STDIN as

Handling warnings:
  --quiet                         Report errors only - default: false
  --max-warnings Int              Number of warnings to trigger nonzero exit
```

```
code - default: -1

Output:
  -o, --output-file path::String  Specify file to write report to
  -f, --format String             Use a specific output format - default: sty
lish
  --color, --no-color             Force enabling/disabling of color

Inline configuration comments:
  --no-inline-config              Prevent comments from changing config or ru
les
  --report-unused-disable-directives  Adds reported errors for unused eslint-
disable directives

Caching:
  --cache                         Only check changed files - default: false
  --cache-file path::String       Path to the cache file. Deprecated: use --c
ache-location - default: .eslintcache
  --cache-location path::String   Path to the cache file or directory
  --cache-strategy String         Strategy to use for detecting changed files
in the cache - either: metadata or content - default: metadata

Miscellaneous:
  --init                          Run config initialization wizard - default:
false
  --env-info                      Output execution environment information -
default: false
  --no-error-on-unmatched-pattern  Prevent errors when pattern is unmatched
  --exit-on-fatal-error           Exit with exit code 2 in case of fatal erro
r - default: false
  --debug                         Output debugging information
  -h, --help                      Show help
  -v, --version                   Output the version number
  --print-config path::String     Print the configuration for the given file
```

Options that accept array values can be specified by repeating the option or with a comma-delimited list (other than `--ignore-pattern` which does not allow the second style).

Example:

```
npx eslint --ext .jsx --ext .js lib/

npx eslint --ext .jsx,.js lib/
```

Basic configuration

`--no-eslintrc`

Disables use of configuration from `.eslintrc.*` and `package.json` files.

Example:

```
npx eslint --no-eslintrc file.js
```

`-c` , `--config`

This option allows you to specify an additional configuration file for ESLint (see Configuring ESLint (configuring) for more).

Example:

```
npx eslint -c ~/my-eslint.json file.js
```

This example uses the configuration file at `~/my-eslint.json` .

If `.eslintrc.*` and/or `package.json` files are also used for configuration (i.e., `--no-eslintrc` was not specified), the configurations will be merged. Options from this configuration file have precedence over the options from `.eslintrc.*` and `package.json` files.

`--env`

This option enables specific environments. Details about the global variables defined by each environment are available on the Specifying Environments (configuring/language-options#specifying-environments) documentation. This option only enables environments; it does not disable environments set in other configuration files. To specify multiple environments, separate them using commas, or use the option multiple times.

Examples:

```
npx eslint --env browser,node file.js
npx eslint --env browser --env node file.js
```

`--ext`

This option allows you to specify which file extensions ESLint will use when searching for target files in the directories you specify. By default, ESLint lints `*.js` files and the files that match the `overrides` entries of your configuration.

Examples:

```
# Use only .ts extension
npx eslint . --ext .ts

# Use both .js and .ts
npx eslint . --ext .js --ext .ts

# Also use both .js and .ts
npx eslint . --ext .js,.ts
```

Note: `--ext` is only used when the arguments are directories. If you use glob patterns or file names, then `--ext` is ignored.

For example, `npx eslint lib/* --ext .js` will match all files within the `lib/` directory, regardless of extension.

`--global`

This option defines global variables so that they will not be flagged as undefined by the `no-undef` rule. Any specified global variables are assumed to be read-only by default, but appending `:true` to a variable's name ensures that `no-undef` will also allow writes. To specify multiple global variables, separate them using commas, or use the option multiple times.

Examples:

```
npx eslint --global require,exports:true file.js
npx eslint --global require --global exports:true
```

`--parser`

This option allows you to specify a parser to be used by ESLint. By default, `espree` will be used.

`--parser-options`

This option allows you to specify parser options to be used by ESLint. Note that the available parser options are determined by the parser being used.

Examples:

```
echo '3 ** 4' | npx eslint --stdin --parser-options=ecmaVersion:6 # will fail
with a parsing error
echo '3 ** 4' | npx eslint --stdin --parser-options=ecmaVersion:7 # succeeds,
yay!
```

`--resolve-plugins-relative-to`

Changes the folder where plugins are resolved from. By default, plugins are resolved from the current working directory. This option should be used when plugins were installed by someone other than the end user. It should be set to the project directory of the project that has a dependency on the necessary plugins. For example:

- When using a config file that is located outside of the current project (with the `--config` flag), if the config uses plugins which are installed locally to itself, `--resolve-plugins-relative-to` should be set to the directory containing the config file.
- If an integration has dependencies on ESLint and a set of plugins, and the tool invokes ESLint on behalf of the user with a preset configuration, the tool should set `--resolve-plugins-relative-to` to the top-level directory of the tool.

Specifying rules and plugins

`--plugin`

This option specifies a plugin to load. You can omit the prefix `eslint-plugin-` from the plugin name.

Before using the plugin, you have to install it using npm.

Examples:

```
npx eslint --plugin jquery file.js
npx eslint --plugin eslint-plugin-mocha file.js
```

`--rule`

This option specifies rules to be used. These rules will be merged with any rules specified with configuration files. (You can use `--no-eslintrc` to change that behavior.) To define multiple rules, separate them using commas, or use the option multiple times. The levn (https://github.com/gkz/levn#levn--) format is used for specifying the rules.

If the rule is defined within a plugin, you have to prefix the rule ID with the plugin name and a `/`.

Examples:

```
npx eslint --rule 'quotes: [error, double]'
npx eslint --rule 'guard-for-in: error' --rule 'brace-style: [error, 1tbs]'
npx eslint --rule 'jquery/dollar-sign: error'
```

`--rulesdir`

Deprecated: Use rules from plugins instead.

This option allows you to specify another directory from which to load rules files. This allows you to dynamically load new rules at run time. This is useful when you have custom rules that aren't suitable for being bundled with ESLint.

Example:

```
npx eslint --rulesdir my-rules/ file.js
```

The rules in your custom rules directory must follow the same format as bundled rules to work properly. You can also specify multiple locations for custom rules by including multiple `--rulesdir` options:

```
npx eslint --rulesdir my-rules/ --rulesdir my-other-rules/ file.js
```

Note that, as with core rules and plugin rules, you still need to enable the rules in configuration or via the `--rule` CLI option in order to actually run those rules during linting. Specifying a rules directory with `--rulesdir` does not automatically enable the rules within that directory.

Fixing problems

`--fix`

This option instructs ESLint to try to fix as many issues as possible. The fixes are made to the actual files themselves and only the remaining unfixed issues are output. Not all problems are fixable using this option, and the option does not work in these situations:

1. This option throws an error when code is piped to ESLint.
2. This option has no effect on code that uses a processor, unless the processor opts into allowing autofixes.

If you want to fix code from `stdin` or otherwise want to get the fixes without actually writing them to the file, use the `--fix-dry-run` option.

`--fix-dry-run`

This option has the same effect as `--fix` with one difference: the fixes are not saved to the file system. This makes it possible to fix code from `stdin` (when used with the `--stdin` flag).

Because the default formatter does not output the fixed code, you'll have to use another one (e.g. `json`) to get the fixes. Here's an example of this pattern:

```
getSomeText | npx eslint --stdin --fix-dry-run --format=json
```

This flag can be useful for integrations (e.g. editor plugins) which need to autofix text from the command line without saving it to the filesystem.

`--fix-type`

This option allows you to specify the type of fixes to apply when using either `--fix` or `--fix-dry-run`. The four types of fixes are:

1. `problem` - fix potential errors in the code
2. `suggestion` - apply fixes to the code that improve it
3. `layout` - apply fixes that do not change the program structure (AST)
4. `directive` - apply fixes to inline directives such as `// eslint-disable`

You can specify one or more fix type on the command line. Here are some examples:

```
npx eslint --fix --fix-type suggestion .
npx eslint --fix --fix-type suggestion --fix-type problem .
npx eslint --fix --fix-type suggestion,layout .
```

This option is helpful if you are using another program to format your code but you would still like ESLint to apply other types of fixes.

Ignoring files

`--ignore-path`

This option allows you to specify the file to use as your `.eslintignore`. By default, ESLint looks in the current working directory for `.eslintignore`. You can override this behavior by providing a path to a different file.

Example:

```
npx eslint --ignore-path tmp/.eslintignore file.js
npx eslint --ignore-path .gitignore file.js
```

`--no-ignore`

Disables excluding of files from `.eslintignore`, `--ignore-path`, `--ignore-pattern`, and `ignorePatterns` property in config files.

Example:

```
npx eslint --no-ignore file.js
```

`--ignore-pattern`

This option allows you to specify patterns of files to ignore (in addition to those in `.eslintignore`). You can repeat the option to provide multiple patterns. The supported syntax is the same as for `.eslintignore` files (configuring/ignoring-code#the-eslintignore-file), which use the same patterns as the `.gitignore` specification (https://git-scm.com/docs/gitignore). You should quote your patterns in order to avoid shell interpretation of glob patterns.

Example:

```
npx eslint --ignore-pattern '/lib/' --ignore-pattern '/src/vendor/*' .
```

Using stdin

`--stdin`

This option tells ESLint to read and lint source code from STDIN instead of from files. You can use this to pipe code to ESLint.

Example:

```
cat myfile.js | npx eslint --stdin
```

`--stdin-filename`

This option allows you to specify a filename to process STDIN as. This is useful when processing files from STDIN and you have rules which depend on the filename.

Example

```
cat myfile.js | npx eslint --stdin --stdin-filename=myfile.js
```

Handling warnings

`--quiet`

This option allows you to disable reporting on warnings. If you enable this option, only errors are reported by ESLint.

Example:

```
npx eslint --quiet file.js
```

`--max-warnings`

This option allows you to specify a warning threshold, which can be used to force ESLint to exit with an error status if there are too many warning-level rule violations in your project.

Normally, if ESLint runs and finds no errors (only warnings), it will exit with a success exit status. However, if `--max-warnings` is specified and the total warning count is greater than the specified threshold, ESLint will exit with an error status. Specifying a threshold of `-1` or omitting this option will prevent this behavior.

Example:

```
npx eslint --max-warnings 10 file.js
```

Output

`-o`, `--output-file`

Enable report to be written to a file.

Example:

```
npx eslint -o ./test/test.html
```

When specified, the given format is output into the provided file name.

`-f`, `--format`

This option specifies the output format for the console. Possible formats are:

- checkstyle (formatters/#checkstyle)
- compact (formatters/#compact)
- html (formatters/#html)
- jslint-xml (formatters/#jslint-xml)
- json (formatters/#json)

- junit (formatters/#junit)
- stylish (formatters/#stylish) (the default)
- tap (formatters/#tap)
- unix (formatters/#unix)
- visualstudio (formatters/#visualstudio)

Example:

```
npx eslint -f compact file.js
```

You can also use a custom formatter from the command line by specifying a path to the custom formatter file.

Example:

```
npx eslint -f ./customformat.js file.js
```

An npm-installed formatter is resolved with or without `eslint-formatter-` prefix.

Example:

```
npm install eslint-formatter-pretty

npx eslint -f pretty file.js

// equivalent:
npx eslint -f eslint-formatter-pretty file.js
```

When specified, the given format is output to the console. If you'd like to save that output into a file, you can do so on the command line like so:

```
npx eslint -f compact file.js > results.txt
```

This saves the output into the `results.txt` file.

`--color`, `--no-color`

This option forces the enabling/disabling of colorized output. You can use this to override the default behavior, which is to enable colorized output unless no TTY is detected, such as when piping `eslint` through `cat` or `less`.

Examples:

```
npx eslint --color file.js | cat
npx eslint --no-color file.js
```

Inline configuration comments

`--no-inline-config`

This option prevents inline comments like `/*eslint-disable*/` or `/*global foo*/` from having any effect. This allows you to set an ESLint config without files modifying it. All inline config comments are ignored, e.g.:

- `/*eslint-disable*/`
- `/*eslint-enable*/`
- `/*global*/`
- `/*eslint*/`
- `/*eslint-env*/`
- `// eslint-disable-line`
- `// eslint-disable-next-line`

Example:

```
npx eslint --no-inline-config file.js
```

`--report-unused-disable-directives`

This option causes ESLint to report directive comments like `// eslint-disable-line` when no errors would have been reported on that line anyway. This can be useful to prevent future errors from unexpectedly being suppressed, by cleaning up old `eslint-disable` comments which are no longer applicable.

Warning: When using this option, it is possible that new errors will start being reported whenever ESLint or custom rules are upgraded. For example, suppose a rule has a bug that causes it to report a false positive, and an `eslint-disable` comment is added to suppress the incorrect report. If the bug is then fixed in a patch release of ESLint, the `eslint-disable` comment will become unused since ESLint is no longer generating an incorrect report. This will result in a new reported error for the unused directive if the `report-unused-disable-directives` option is used.

Example:

```
npx eslint --report-unused-disable-directives file.js
```

Caching

`--cache`

Store the info about processed files in order to only operate on the changed ones. The cache is stored in `.eslintcache` by default. Enabling this option can dramatically improve ESLint's running time by ensuring that only changed files are linted.

Note: If you run ESLint with `--cache` and then run ESLint without `--cache`, the `.eslintcache` file will be deleted. This is necessary because the results of the lint might change and make `.eslintcache` invalid. If you want to control when the cache file is deleted, then use `--cache-location` to specify an alternate location for the cache file.

Note: Autofixed files are not placed in the cache. Subsequent linting that does not trigger an autofix will place it in the cache.

`--cache-file`

Path to the cache file. If none specified `.eslintcache` will be used. The file will be created in the directory where the `eslint` command is executed. **Deprecated**: Use `--cache-location` instead.

`--cache-location`

Path to the cache location. Can be a file or a directory. If no location is specified, `.eslintcache` will be used. In that case, the file will be created in the directory where the `eslint` command is executed.

If a directory is specified, a cache file will be created inside the specified folder. The name of the file will be based on the hash of the current working directory (CWD). e.g.: `.cache_hashOfCWD`

Important note: If the directory for the cache does not exist make sure you add a trailing `/` on *nix systems or `\` in windows. Otherwise the path will be assumed to be a file.

Example:

```
npx eslint "src/**/*.js" --cache --cache-location "/Users/user/.eslintcache/"
```

`--cache-strategy`

Strategy for the cache to use for detecting changed files. Can be either `metadata` or `content`. If no strategy is specified, `metadata` will be used.

The `content` strategy can be useful in cases where the modification time of your files change even if their contents have not. For example, this can happen during git operations like git clone because git does not track file modification time.

Example:

```
npx eslint "src/**/*.js" --cache --cache-strategy content
```

Miscellaneous

`--init`

This option will run `npm init @eslint/config` to start config initialization wizard. It's designed to help new users quickly create .eslintrc file by answering a few questions, choosing a popular style guide.

The resulting configuration file will be created in the current directory.

`--env-info`

This option outputs information about the execution environment, including the version of Node, npm, and local and global installations of ESLint. The ESLint team may ask for this information to help solve bugs.

`--no-error-on-unmatched-pattern`

This option prevents errors when a quoted glob pattern or `--ext` is unmatched. This will not prevent errors when your shell can't match a glob.

`--exit-on-fatal-error`

This option causes ESLint to exit with exit code 2 if one or more fatal parsing errors occur. Without this option, fatal parsing errors are reported as rule violations.

`--debug`

This option outputs debugging information to the console. This information is useful when you're seeing a problem and having a hard time pinpointing it. The ESLint team may ask for this debugging information to help solve bugs. Add this flag to an ESLint command line invocation in order to get extra debug information as the command is run (e.g. `npx eslint --debug test.js` and `npx eslint test.js --debug` are equivalent)

`-h`, `--help`

This option outputs the help menu, displaying all of the available options. All other options are ignored when this is present.

`-v`, `--version`

This option outputs the current ESLint version onto the console. All other options are ignored when this is present.

`--print-config`

This option outputs the configuration to be used for the file passed. When present, no linting is performed and only config-related options are valid.

Example:

```
npx eslint --print-config file.js
```

Ignoring files from linting

ESLint supports `.eslintignore` files to exclude files from the linting process when ESLint operates on a directory. Files given as individual CLI arguments will be exempt from exclusion. The `.eslintignore` file is a plain text file containing one pattern per line. It can be located in any of the target directory's ancestors; it will affect files in its containing directory as well as all sub-directories. Here's a simple example of a `.eslintignore` file:

```
temp.js
**/vendor/*.js
```

A more detailed breakdown of supported patterns and directories ESLint ignores by default can be found in Ignoring Code (configuring/ignoring-code).

Exit codes

When linting files, ESLint will exit with one of the following exit codes:

- `0`: Linting was successful and there are no linting errors. If the `--max-warnings` flag is set to `n`, the number of linting warnings is at most `n`.
- `1`: Linting was successful and there is at least one linting error, or there are more linting warnings than allowed by the `--max-warnings` option.
- `2`: Linting was unsuccessful due to a configuration problem or an internal error.

Group (https://groups.google.com/group/eslint) GitHub (https://github.com/eslint/eslint)

Twitter (https://twitter.com/geteslint) Discord (/chat)

Edit this page (https://github.com/eslint/eslint/edit/main/docs/src/user-guide/command-line-interface.md)