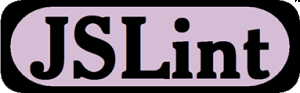
Code Quality (JSHint, JSLint etc)

[JSLint](http://www.jslint.com/), [JSHint](http://jshint.com/), [JSCS](http://jscs.info/)  [ESLint](http://eslint.org/).

Все они могут быть установлены с помощью npm, содержат набор правил на который они будут проверять код

Могут использоваться с командной строки, передавая им путь к файлу, есть плагины для таск ранеров, добавлены в редакторы

JSLint



JSLint is the oldest of the four. Douglas Crockford created it in 2002 to enforce what, in his experience, are the good parts of JavaScript. If you agree with the good parts, JSLint can be a good tool—you install it and it’s ready to go.

The downsides are that JSLint is not configurable or extensible. You can’t disable many features at all, and some of them lack documentation. The official website is not very helpful, for example it lacks any information on how to integrate it with your editor.

Pros

* Comes configured and ready to go (if you agree with the rules it enforces)

Cons

* JSLint doesn’t have a configuration file, which can be problematic if you need to change the settings
* Limited number of configuration options, many rules cannot be disabled
* You can’t add custom rules
* Undocumented features
* Difficult to know which rule is causing which error

JSHint



JSHint was created as a more configurable version of JSLint (of which it is a fork). You can configure every rule, and put them into a configuration file, which makes JSHint easy to use in bigger projects. JSHint also has good documentation for each of the rules, so you know exactly what they do. Integrating it into editors is also simple.

A small downside to JSHint is that it comes with a relaxed default configuration. This means you need to do some setup to make it useful. When comparing it with ESLint, it’s also more difficult to know which rules you need to change in order to enable or disable certain error messages.

Pros

* Most settings can be configured
* Supports a configuration file, making it easier to use in larger projects
* Has support for many libraries out of the box, like jQuery, QUnit, NodeJS, Mocha, etc.
* Basic ES6 support

Cons

* Difficult to know which rule is causing an error
* Has two types of option: enforcing and relaxing (which can be used to make JSHint stricter, or to suppress its warnings). This can make configuration slightly confusing
* No custom rule support

JSCS



JSCS is different from the others in that it doesn’t do anything unless you give it a configuration file or tell it to use a preset. You can download configurations from their website, so it’s not a big problem, and it has a number of presets, such as the jQuery coding style preset and the Google preset.

It has over 90 different rules, and you can create custom ones with plugins. JSCS also supports custom reporters, which makes it easier to integrate with tools that need their input in a specific format.

JSCS is a code style checker. This means it only catches issues related to code formatting, and not potential bugs or errors. As a result, it’s less flexible than the others, but if you need to enforce a specific coding style, it’s a job JSCS does well.

Pros

* Supports custom reporters, which can make it easier to integrate with other tools
* Presets and ready-made configuration files can make it easy to set up if you follow one of the available coding styles
* Has a flag to include rule names in reports, so it’s easy to figure out which rule is causing which error
* Can be extended with custom plugins

Cons

* Only detects coding style violations. JSCS doesn’t detect potential bugs such as unused variables, or accidental globals, etc.
* Slowest of the four, but this is not a problem in typical use

ESLint



ESLint is the most recent out of the four. It was designed to be easily extensible, comes with a large number of custom rules, and it’s easy to install more in the form of plugins. It gives concise output, but includes the rule name by default so you always know which rules are causing the error messages.

ESLint documentation can be a bit hit or miss. The rules list is easy to follow and is grouped into logical categories, but the configuration instructions are a little confusing in places. It does, however, offer links to editor integration, plugins and examples all in a single location.

Pros

* Flexible: any rule can be toggled, and many rules have extra settings that can be tweaked
* Very extensible and has many plugins available
* Easy to understand output
* Includes many rules not available in other linters, making ESLint more useful for detecting problems
* Best ES6 support, and also the only tool to support [JSX](https://jsx.github.io/)
* Supports custom reporters

Cons

* Some configuration required
* Slow, but not a hindrance

My Recommendations

My choice of these four is ESLint. JSLint is strict and not configurable, whereas JSHint is lacking the extension mechanism. JSCS is a good choice if you only want to check coding style, but ESLint does that*and* it checks your code for bugs and other problems as well.

ESLint is also the obvious choice if you want to use ES6 (or ES2015, as they seem to be calling it now). Of all the tools mentioned, it has the widest support for ES6 features.

If you want to try ESLint, I’ve made it really easy for you by creating a 5-step quick start guide. You can[download the ESLint 5-step quickstart guide from my website](http://codeutopia.net/blog/h/get-eslint-guide/).

JSHint is strong second choice. If you don’t need the advanced features of ESLint, JSHint catches a good number of issues once properly configured. JSCS, with its huge number of available rules, is a top pick if you don’t need anything other than coding style checks (indentation, braces, etc.).

I’m hesitant to recommend JSLint at all. The other tools do the same things, but don’t force any specific rules on the user. The only exception here is if you happen to agree with all the rules it enforces, in which case it might well be worth looking into.

A linting tool is a good step towards catching problems, but it only sees as many errors as its rules permit. For a more foolproof automated bug-catcher, I recommend using unit tests. Code reviews can also be useful for this purpose.