**JSHint, A Static Code Analysis Tool for JavaScript**

[ [Use it online](http://jshint.com/) • [Docs](http://jshint.com/docs/) • [FAQ](http://jshint.com/docs/faq) • [Install](http://jshint.com/install/) • [Contribute](http://jshint.com/contribute/) • [Blog](http://jshint.com/blog/) • [Twitter](https://twitter.com/jshint/) ]

JSHint is a community-driven tool that detects errors and potential problems in JavaScript code. Since JSHint is so flexible, you can easily adjust it in the environment you expect your code to execute. JSHint is open source and will always stay this way.

**Our goal**

The project aims to help JavaScript developers write complex programs without worrying about typos and language gotchas.

Any code base eventually becomes huge at some point, so simple mistakes — that would not show themselves when written — can become show stoppers and add extra hours of debugging. So, static code analysis tools come into play and help developers spot such problems. JSHint scans a program written in JavaScript and reports about commonly made mistakes and potential bugs. The potential problem could be a syntax error, a bug due to an implicit type conversion, a leaking variable, or something else entirely.

Only 15% of all programs linted on [jshint.com](http://jshint.com/) pass the JSHint checks. In all other cases, JSHint finds some red flags that could've been bugs or potential problems.

Please note, that while static code analysis tools can spot many different kind of mistakes, it can't detect if your program is correct, fast or has memory leaks. You should always combine tools like JSHint with unit and functional tests as well as with code reviews.

**Reporting a bug**

To report a bug simply create a [new GitHub Issue](https://github.com/jshint/jshint/issues/new) and describe your problem or suggestion. We welcome all kinds of feedback regarding JSHint including but not limited to:

* When JSHint doesn't work as expected
* When JSHint complains about valid JavaScript code that works in all browsers
* When you simply want a new option or feature

Before reporting a bug, please look around to see if there are any open or closed tickets that discuss your issue, and remember the wisdom: pull request > bug report > tweet.

**Who uses JSHint?**

Engineers from these companies and projects use JSHint:

* [Mozilla](https://www.mozilla.org/)
* [Wikipedia](https://wikipedia.org/)
* [Facebook](https://facebook.com/)
* [Twitter](https://twitter.com/)
* [Bootstrap](http://getbootstrap.com/)
* [Disqus](https://disqus.com/)
* [Medium](https://medium.com/)
* [Yahoo!](https://yahoo.com/)
* [SmugMug](http://smugmug.com/)
* [jQuery](http://jquery.com/)
* [PDF.js](http://mozilla.github.io/pdf.js)
* [Coursera](http://coursera.com/)
* [Adobe Brackets](http://brackets.io/)
* [Apache Cordova](http://cordova.io/)
* [RedHat](http://redhat.com/)
* [SoundCloud](http://soundcloud.com/)
* [Nodejitsu](http://nodejitsu.com/)
* [Yelp](https://yelp.com/)
* [Voxer](http://voxer.com/)
* [EnyoJS](http://enyojs.com/)
* [QuickenLoans](http://quickenloans.com/)
* [Cloud9](http://c9.io/)
* [CodeClimate](https://codeclimate.com/)
* [Zendesk](http://zendesk.com/)
* [Apache CouchDB](http://couchdb.apache.org/)
* [Google](https://www.google.com/)
* [Codacy](https://www.codacy.com/) [ref](https://support.codacy.com/hc/en-us/articles/207995005-Special-Thanks)

And many more!

# Documentation

JSHint is a program that flags suspicious usage in programs written in JavaScript. The core project consists of a library itself as well as a CLI program distributed as a Node module.

More docs: [List of all JSHint options](http://jshint.com/docs/options/) · [Command-line Interface](http://jshint.com/docs/cli/) · [API](http://jshint.com/docs/api/) · [Writing your own reporter](http://jshint.com/docs/reporters/) · [FAQ](http://jshint.com/docs/faq/)

### Basic usage

First, check out [the installation instructions](http://jshint.com/install/) for details on how to install JSHint in your preferred environment. Both the command line executable and the JavaScript API offer unique ways to configure JSHint's behaviour. The most common usages are:

* [As a command-line tool](http://jshint.com/docs/cli/) (via [Node.js](https://nodejs.org/))
* [As a JavaScript module](http://jshint.com/docs/api/)

Regardless of your preferred environment, you can control JSHint's behavior through specifying any number of [linting options](http://jshint.com/docs/options/). In addition, JSHint will honor any directives declared within the input source code--see [the section on in-line directives](http://jshint.com/docs/#inline-configuration) for more information.

### Configuration

JSHint comes with a default set of warnings but it was designed to be very configurable. There are three main ways to configure your copy of JSHint: you can either specify the configuration file manually via the --config flag, use a special file .jshintrc or put your config into your projects package.json file under the jshintConfig property. In case of .jshintrc, JSHint will start looking for this file in the same directory as the file that's being linted. If not found, it will move one level up the directory tree all the way up to the filesystem root. (Note that if the input comes from stdin, JSHint doesn't attempt to find a configuration file)

This setup allows you to have different configuration files per project. Place your file into the project root directory and, as long as you run JSHint from anywhere within your project directory tree, the same configuration file will be used.

Configuration file is a simple JSON file that specifies which JSHint options to turn on or off. For example, the following file will enable warnings about undefined and unused variables and tell JSHint about a global variable named MY\_GLOBAL.

{

"undef": true,

"unused": true,

"globals": {

"MY\_GLOBAL": true

}

}

### Inline configuration

In addition to using configuration files you can configure JSHint from within your files using special comments. These comments start with a label such as jshint or globals (complete list below) and are followed by a comma-separated list of values. For example, the following snippet will enable warnings about undefined and unused variables and tell JSHint about a global variable named MY\_GLOBAL.

/\* jshint undef: true, unused: true \*/

/\* globals MY\_GLOBAL \*/

You can use both multi- and single-line comments to configure JSHint. These comments are function scoped meaning that if you put them inside a function they will affect only this function's code.

### Directives

Here's a list of configuration directives supported by JSHint:

#### jshint

A directive for setting JSHint options.

/\* jshint strict: true \*/

#### jslint

A directive for setting JSHint-compatible JSLint options.

/\* jslint vars: true \*/

#### globals

A directive for telling JSHint about global variables that are defined elsewhere. If value is false (default), JSHint will consider that variable as read-only. Use it together with the undef option.

/\* globals MY\_LIB: false \*/

You can also blacklist certain global variables to make sure they are not used anywhere in the current file.

/\* globals -BAD\_LIB \*/

#### exported

A directive for telling JSHint about global variables that are defined in the current file but used elsewhere. Use it together with the unused option.

/\* exported EXPORTED\_LIB \*/

#### members

A directive for telling JSHint about all properties you intend to use. **This directive is deprecated.**

#### ignore

A directive for telling JSHint to ignore a block of code.

// Code here will be linted with JSHint.

/\* jshint ignore:start \*/

// Code here will be ignored by JSHint.

/\* jshint ignore:end \*/

All code in between ignore:start and ignore:end won't be passed to JSHint so you can use any language extension such as [Facebook React](http://facebook.github.io/react/). Additionally, you can ignore a single line with a trailing comment:

ignoreThis(); // jshint ignore:line

### Options

Most often, when you need to tune JSHint to your own taste, all you need to do is to find an appropriate option. Trying to figure out how JSHint options work can be confusing and frustrating (and we're working on fixing that!) so please read the following couple of paragraphs carefully.

JSHint has two types of options: enforcing and relaxing. The former are used to make JSHint more strict while the latter are used to suppress some warnings. Take the following code as an example:

function main(a, b) {

return a == null;

}

This code will produce the following warning when run with default JSHint options:

line 2, col 14, Use '===' to compare with 'null'.

Let's say that you know what you're doing and want to disable the produced warning but, in the same time, you're curious whether you have any variables that were defined but never used. What you need to do, in this case, is to enable two options: one relaxing that will suppress the === null warning and one enforcing that will enable checks for unused variables. In your case these options are unused and eqnull.

/\*jshint unused:true, eqnull:true \*/

function main(a, b) {

return a == null;

}

After that, JSHint will produce the following warning while linting this example code:

demo.js: line 2, col 14, 'main' is defined but never used.

demo.js: line 2, col 19, 'b' is defined but never used.

Sometimes JSHint doesn't have an appropriate option that disables some particular warning. In this case you can use jshint directive to disable warnings by their code. Let's say that you have a file that was created by combining multiple different files into one:

"use strict";

/\* ... \*/

// From another file

function b() {

"use strict";

/\* ... \*/

}

This code will trigger a warning about an unnecessary directive in function b. JSHint sees that there's already a global "use strict" directive and informs you that all other directives are redundant. But you don't want to strip out these directives since the file was auto-generated. The solution is to run JSHint with a flag --verbose and note the warning code (W034 in this case):

$ jshint --verbose myfile.js

myfile.js: line 6, col 3, Unnecessary directive "use strict". (W034)

Then, to hide this warning, just add the following snippet to your file:

/\* jshint -W034 \*/

A couple things to note:

1. This syntax works only with warnings (code starts with W), it doesn't work with errors (code starts with E).
2. This syntax will disable all warnings with this code. Some warnings are more generic than others so be cautious.

To re-enable a warning that has been disabled with the above snippet you can use:

/\* jshint +W034 \*/

This is especially useful when you have code which causes a warning but that you know is safe in the context. In these cases you can disable the warning as above and then re-enable the warning afterwards:

var y = Object.create(null);

// ...

/\*jshint -W089 \*/

for (var prop in y) {

// ...

}

/\*jshint +W089 \*/

[This page](http://jshint.com/docs/options/) contains a list of all options supported by JSHint.

#### Switch statements

By default JSHint warns when you omit break or return statements within switch statements:

switch (cond) {

case "one":

doSomething(); // JSHint will warn about missing 'break' here.

case "two":

doSomethingElse();

}

If you really know what you're doing you can tell JSHint that you intended the case block to fall through by adding a /\* falls through \*/ comment:

switch (cond) {

case "one":

doSomething();

/\* falls through \*/

case "two":

doSomethingElse();

}