How to quickly and conveniently disable all console.log statements in my code?

Asked 10 years, 2 months ago Active 12 days ago Viewed 178k times

226

98

Is there any way to turn off all `console.log` statements in my JavaScript code, for testing purposes?

javascript debugging console

edited Feb 28 '17 at 8:16

Sebastian Zartner
13.5k 7 63 103

asked Jul 31 '09 at 23:50

Zack Burt
3,360 6 47 74

11 use a text editor that supports "replace all" and replace "console.log" with "//console.log" – helloandre Aug 1 '09 at 0:13

5 @helloandre - that gets a little tiresome though if you use log, info, warn debug and error – UpTheCreek Jun 3 '12 at 9:11

Hopefully we'll reach a point where browser implementations automatically bypass console statements unless the browser's debugging tool is enabled. – faintsignal Nov 5 '15 at 22:22

2 The answers below are great, but you don't have to reinvent the wheel on this one. Have a look at picolog. It has an API compatible with (NodeJS's) console, so you can use it as a drop-in replacement. It supports logging levels out of the box, works in the browser, on NodeJS and on Nashorn, can be easily configured from the querystring (browser) or environment variable `PICOLOG_LEVEL` (node) and it's *super* small. Less then 900 bytes minified and zipped. Disclaimer: I am the author. – Stijn de Witt Jan 3 '16 at 9:10

There is a simple way to override all `console` functions. Just look at stapp.space/disable-javascript-console-on-production – Piotr Stapp Jan 12 '16 at 12:25

25 Answers

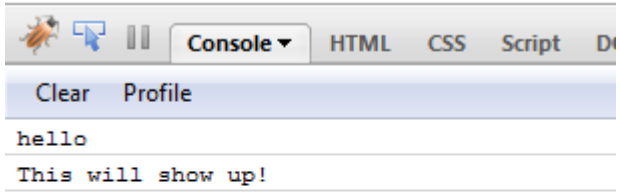
Redefine the console.log function in your script

That's it, no more messages to console.

EDIT:

Expanding on Cide's idea. A custom logger which you can use to toggle logging on/off from your code.

From my Firefox console:




```
var logger = function()
{
    var oldConsoleLog = null;
    var pub = {};

    pub.enableLogger =  function enableLogger()
                        {
                            if(oldConsoleLog == null)
                                return;

                            window['console']['log'] = oldConsoleLog;
                        };

    pub.disableLogger = function disableLogger()
                        {
                            oldConsoleLog = console.log;
                            window['console']['log'] = function() {};
                        };

    return pub;
}();
```

```
    {
        console.log('hello');

        logger.disableLogger();
        console.log('hi', 'hiya');
        console.log('this wont show up in console');

        logger.enableLogger();
        console.log('This will show up!');
    }
);
```

How to use the above 'logger'? In your ready event, call logger.disableLogger so that console messages are not logged. Add calls to logger.enableLogger and logger.disableLogger inside the method for which you want to log messages to the console.

edited Sep 13 '17 at 12:47
gunwin
1,680 4 25 44

answered Jul 31 '09 at 23:54
SolutionYogi
27.2k 10 64 77

Please provide detail as to what doesn't work? Does the above line give you an error? If yes, what's the error message? – SolutionYogi Aug 1 '09 at 0:58

1 Works for me in IE8. ;-) – Eugene Lazutkin Aug 1 '09 at 1:56

Excellent, thanks. Do you know if this works in IE7? – Zack Burt Aug 1 '09 at 2:39

The code overwrites and restores console.log function. If IE7 supports console.log method, it should work. – SolutionYogi Aug 1 '09 at 2:48

3 console.log = function() {} does not seem to work in Firefox. You still get a 'console is not defined' error. – DA. Oct 28 '09 at 16:29

69

The following is more thorough:

```
var DEBUG = false;
if(!DEBUG){
    if(!window.console) window.console = {};
    var methods = ["log", "debug", "warn", "info"];
    for(var i=0;i<methods.length;i++){
        console[methods[i]] = function(){};
    }
```

overhead. In the case of a browser like IE6 with no console, the dummy methods will be created to prevent errors. Of course there are many more functions in Firebug, like trace, profile, time, etc. They can be added to the list if you use them in your code.

You can also check if the debugger has those special methods or not (ie, IE) and zero out the ones it does not support:

```
if(window.console && !console.dir){
var methods = ["dir", "dirxml", "trace", "profile"]; //etc etc
    for(var i=0;i<methods.length;i++){
        console[methods[i]] = function(){};
    }
}
```

answered Aug 1 '09 at 14:52

mwilcox
3,326 19 19

25

As far as I can tell from the documentation, Firebug doesn't supply any variable to toggle debug state. Instead, wrap console.log() in a wrapper that conditionally calls it, i.e.:

```
DEBUG = true; // set to false to disable debugging
function debug_log() {
    if ( DEBUG ) {
        console.log.apply(this, arguments);
    }
}
```

To not have to change all the existing calls, you can use this instead:

```
DEBUG = true; // set to false to disable debugging
old_console_log = console.log;
console.log = function() {
    if ( DEBUG ) {
        old_console_log.apply(this, arguments);
    }
}
```

1 Thanks, although this means I need to rewrite all my console.log statements as debug.log. – Zack Burt Aug 1 '09 at 0:05

This is the right way to do it - ofcourse if you are starting from scratch. – OpenSource Aug 1 '09 at 0:13

2 It is also the right way to do it if you have a good find/replace function in your editor. – BaroqueBobcat Aug 1 '09 at 0:34

2 No need to write your own wrapper, btw, at least if you're using jQuery. The jQuery Debugging Plugin works great. As a bonus it provides emulation of console.log on browsers without it. trainofthoughts.org/blog/2007/03/16/jquery-plugin-debug – Nelson Aug 1 '09 at 0:40

The only [minor] problem, of course, being that you need to install a plugin. :) Good to know, though - thanks! – Cide Aug 1 '09 at 0:41

14

You should not!

It is not a good practice to overwrite built-in functions. There is also no guarantee that you will suppress all output, other libraries you use may revert your changes and there are other functions that may write to the console; `.dir()`, `.warning()`, `.error()`, `.debug()`, `.assert()` etc.

As some suggested, you could define a `DEBUG_MODE` variable and log conditionally. Depending on the complexity and nature of your code, it may be a good idea to write your own logger object/function that wraps around the console object and has this capability built-in. That would be the right place to do deal with **instrumentation**.

That said, for 'testing' purposes you can write **tests** instead of printing to the console. If you are not doing any testing, and those `console.log()` lines were just an aid to write your code, **simply delete them**.

answered Jan 6 '15 at 11:27

istepaniuk
2,007 2 19 41

4 `"other libraries you use may revert your changes"` : if I disable `console.log` at the very start, they can't revert to the old function. Well, they can rewrite the `console.log` source code, but why? `"it may be a good idea to write your own logger object/function that wraps around the console object"` : I've done this in past and it's a bad idea. The trace of the console output refers to the wrapper and not to the line that invokes it, making debugging more difficult. – Marco Sulla Aug 7 '15 at 10:54

3 @LucasMalor "at the very start" implies the code is coupled to that infrastructure thus limiting its reusability. It is difficult to generalized though; A game, some DOM animations are not the same as domain logic inside a complex SPA, the later shouldn't be browser-aware, let alone knowing about something called "console". In that case you should have a proper testing strategy instead of hacking some `console.log('Look ma, it reaches this`

where logging functions are disabled, it's a logic that you can apply everywhere. `"the later shouldn't be browser-aware"` : well, so you shouldn't use JS :P – Marco Sulla Aug 12 '15 at 10:15

3 @MarcoSulla I think he's getting at writing cleaner code. Saying "....you shouldn't use JS" is a bit heavy handed. Ideally, as a programmer, no matter what your environment is, you should modularize as much as possible; if it doesn't care about browsers, then you can deploy it in more places: that's one less dependency to worry about breaking your stuff. So, IMHO yeah, he's actually right. Keep in mind that you started by saying "If you create a common basic template..." which in and of itself introduces a dependency. This kind of thinking is what complicates software. Food for thought. – dudewad Aug 31 '15 at 21:57

Adobe SiteCatalyics throws a lot of junk in my console and makes a hassle of debugging in some cases. So being able to temporarily disable console.log when I execute a third-party call would be pretty useful for me – spinn Mar 9 '18 at 10:54

13

I realize this is an old post but it still pops up at the top of Google results, so here is a more elegant non-jQuery solution that works in the latest Chrome, FF, and IE.

```
(function (original) {
    console.enableLogging = function () {
        console.log = original;
    };
    console.disableLogging = function () {
        console.log = function () {};
    };
})(console.log);
```

answered Sep 9 '13 at 20:51

Joey Schooley
236 2 7

12

I know you asked how to disable console.log, but this might be what you're really after. This way you don't have to explicitly enable or disable the console. It simply prevents those pesky console errors for people who don't have it open or installed.

```
if(typeof(console) === 'undefined') {
    var console = {};
    console.log = console.error = console.info = console.debug = console.warn =
console.trace = console.dir = console.dirxml = console.group = console.groupEnd =
console.time = console.timeEnd = console.assert = console.profile = function() {};
```

Scott Greenfield
2,046 2 18 20

1 For IE specific logging disabling see Chris S. answer. – GuruM Feb 11 '13 at 6:50

10

Just change the flag `DEBUG` to override the console.log function. This should do the trick.

```
var DEBUG = false;
// ENABLE/DISABLE Console Logs
if(!DEBUG){
  console.log = function() {}
}
```

answered Dec 17 '14 at 17:40

Swanidhi
1,644 1 15 18

2 I'd go one step further and wrap this in a logger function/class. Something like this: `function myLog(msg) { if (debug) { console.log(msg); } }` – sleblanc Apr 14 '15 at 19:44

If using Angular, you can use it as a global configuration in your application.js file and use it as a global property to switch the logs on /off. Remember, console will not be undefined if you have the developers toolbar open in IE. – Swanidhi May 1 '15 at 21:37

9

I am surprised that of all those answers no one combines:

- No jquery
- Anonymous function to not pollute global namespace
- Handle case where window.console not defined
- Just modify the .log function of the console

I'd go for this:

```
    var debug = false

    if (debug === false) {
        if ( typeof(window.console) === 'undefined') { window.console = {}; }
        window.console.log = function () {};
    }
})()
```

edited May 8 '15 at 10:08

answered May 8 '15 at 10:02

Xavier13

596 6 15

8

If you're using IE7, console won't be defined. So a more IE friendly version would be:

```
if (typeof console == "undefined" || typeof console.log == "undefined")
{
   var console = { log: function() {} };
}
```

answered May 20 '11 at 18:40

Chris S

51.5k 47 211 234

2 +1 for handling IE eccentricities. – GuruM Feb 11 '13 at 6:48

7

After I searched for this issue aswell and tried it within my cordova app, I just want to warn every developer for windows phone to not overwrite

```
    console.log
```

because the app will crash on startup.

It won't crash if you're developing local if you're lucky, but submitting in store it will result in crashing the app.

```
window.console.log
```

if you need to.

This works in my app:

```
try {
    if (typeof(window.console) != "undefined") {
        window.console = {};
        window.console.log = function () {
        };
        window.console.info = function () {
        };
        window.console.warn = function () {
        };
        window.console.error = function () {
        };
    }

    if (typeof(alert) !== "undefined") {
        alert = function ()
        {

        }
    }

} catch (ex) {

}
```

answered Jul 21 '16 at 14:43

graphefruit
199 3 13

best answer, thank you! – xims Mar 21 '17 at 13:16

This a hybrid of answers from *SolutionYogi* and *Chris S.* It maintains the console.log line numbers and file name. Example jsFiddle.

```
(function(MYAPP, $, undefined) {
    // Prevent errors in browsers without console.log
    if (!window.console) window.console = {};
    if (!window.console.log) window.console.log = function(){};

    //Private var
    var console_log = console.log;

    //Public methods
    MYAPP.enableLog = function enableLogger() { console.log = console_log; };
    MYAPP.disableLog = function disableLogger() { console.log = function() {}; };

}(window.MYAPP = window.MYAPP || {}, jQuery));


// Example Usage:
$(function() {
    MYAPP.disableLog();
    console.log('this should not show');

    MYAPP.enableLog();
    console.log('This will show');
});
```

edited Mar 19 '14 at 18:06

answered Jan 8 '13 at 21:02

Justin
17.9k 11 85 110

3

If you use Grunt you can add a task in order to remove/comment the console.log statements. Therefore the console.log are no longer called.

https://www.npmjs.org/package/grunt-remove-logging-calls

answered Oct 14 '14 at 20:34

jdborowy
51 2

I have used winston logger earlier

1. Set the environment variable from cmd/ command line (on Windows):

```
cmd
setx LOG_LEVEL info
```

Or, you could have a variable in your code if you like, but above is better.

2. Restart cmd/ command line, or, IDE/ editor like Netbeans
3. Have below like code:

```
console.debug = console.log;   // define debug function
console.silly = console.log;   // define silly function

switch (process.env.LOG_LEVEL) {
    case 'debug':
    case 'silly':
        // print everything
        break;

    case 'dir':
    case 'log':
        console.debug = function () {};
        console.silly = function () {};
        break;

    case 'info':
        console.debug = function () {};
        console.silly = function () {};
        console.dir = function () {};
        console.log = function () {};
        break;

    case 'trace':   // similar to error, both may print stack trace/ frames
    case 'warn':    // since warn() function is an alias for error()
    case 'error':
        console.debug = function () {};
        console.silly = function () {};
        console.dir = function () {};
        console.log = function () {};
        console.info = function () {};
        break;
```

4. Now use all console." as below:

```
console.error(' this is a error message '); // will print
console.warn(' this is a warn message '); // will print
console.trace(' this is a trace message '); // will print
console.info(' this is a info message '); // will print, LOG_LEVEL is set to this

console.log(' this is a log message '); // will NOT print
console.dir(' this is a dir message '); // will NOT print
console.silly(' this is a silly message '); // will NOT print
console.debug(' this is a debug message '); // will NOT print
```

Now, based on your LOG_LEVEL settings made in the point 1 (like, `setx LOG_LEVEL log` and restart command line), some of the above will print, others won't print

Hope that helped.

edited Dec 24 '18 at 5:52

answered Sep 13 '17 at 5:51

Manohar Reddy Poreddy

7,754 1 63 64

2

Warning: Shameless plug!

You could also use something like my JsTrace object to have modular tracing with module-level "switching" capability to only turn on what you want to see at the time.

http://jstrace.codeplex.com

(Also has a NuGet package, for those who care)

All levels default to "error", though you can shut them "off". Though, I can't think of why you would NOT want to see errors

You can change them like this:

```
Trace.traceLevel('ModuleName1', Trace.Levels.log);
Trace.traceLevel('ModuleName2', Trace.Levels.info);
```

answered Mar 2 '12 at 15:15

Tom McKearney
295 2 11

2

I found a little more advanced piece of code in this url JavaScript Tip: Bust and Disable console.log:

```
var DEBUG_MODE = true; // Set this value to false for production

if(typeof(console) === 'undefined') {
   console = {}
}

if(!DEBUG_MODE || typeof(console.log) === 'undefined') {
   // FYI: Firebug might get cranky...
   console.log = console.error = console.info = console.debug = console.warn =
console.trace = console.dir = console.dirxml = console.group = console.groupEnd =
console.time =    console.timeEnd = console.assert = console.profile = function() {};
}
```

answered Feb 14 '13 at 14:49

kikeenrique
1,591 2 18 37

2

I developed a library for this usecase: https://github.com/sunnykgupta/jsLogger

Features:

1. It safely overrides the console.log.
2. Takes care if the console is not available (oh yes, you need to factor that too.)
3. Stores all logs (even if they are suppressed) for later retrieval.
4. Handles major console functions like `log` , `warn` , `error` , `info` .

Sunny R Gupta
3,984 1 28 38

2

This should override all methods of window.console. You can put it on the very top of your scripts section, and if you are on a PHP framework you can only print this code when your app environment is production, or if some kind of debug flag is disabled. Then you would have all your logs in your code working on development environments or in debug mode.

```
window.console = (function(originalConsole){
    var api = {};
    var props = Object.keys(originalConsole);
    for (var i=0; i<props.length; i++) {
        api[props[i]] = function(){};
    }
    return api;
})(window.console);
```

answered Apr 25 '17 at 22:11

agm-dev
21 2

2

Ive been using the following to deal with he problem:-

```
var debug = 1;
var logger = function(a,b){ if ( debug == 1 ) console.log(a, b || "");};
```

Set debug to 1 to enable debugging. Then use the logger function when outputting debug text. It's also set up to accept two parameters.

So, instead of

```
console.log("my","log");
```

use

answered May 8 '17 at 15:42
Michae Pavlos Michael
31 1 6

1

I wrote this:

```
//Make a copy of the old console.
var oldConsole = Object.assign({}, console);

//This function redefine the caller with the original one. (well, at least i expect this
to work in chrome, not tested in others)
function setEnabled(bool) {
    if (bool) {
        //Rewrites the disable function with the original one.
        console[this.name] = oldConsole[this.name];
        //Make sure the setEnable will be callable from original one.
        console[this.name].setEnabled = setEnabled;
    } else {
        //Rewrites the original.
        var fn = function () {/*function disabled, to enable call
console.fn.setEnabled(true)*/};
        //Defines the name, to remember.
        Object.defineProperty(fn, "name", {value: this.name});
        //replace the original with the empty one.
        console[this.name] = fn;
        //set the enable function
        console[this.name].setEnabled = setEnabled

    }
}
```

Unfortunately it doesn't work on use strict mode.

So using `console.fn.setEnabled = setEnabled` and then `console.fn.setEnabled(false)` where `fn` could be almost any console function. For your case would be:

```
console.log.setEnabled = setEnabled;
console.log.setEnabled(false);
```

```
var FLAGS = {};
    FLAGS.DEBUG = true;

    FLAGS.INFO = false;
    FLAGS.LOG = false;
    //Adding dir, table, or other would put the setEnabled on the respective console
functions.

function makeThemSwitchable(opt) {
    var keysArr = Object.keys(opt);
    //its better use this type of for.
    for (var x = 0; x < keysArr.length; x++) {
        var key = keysArr[x];
        var lowerKey = key.toLowerCase();
        //Only if the key exists
        if (console[lowerKey]) {
            //define the function
            console[lowerKey].setEnabled = setEnabled;
            //Make it enabled/disabled by key.
            console[lowerKey].setEnabled(opt[key]);
        }
    }
}
//Put the set enabled function on the original console using the defined flags and set
them.
makeThemSwitchable(FLAGS);
```

so then you just need to put in the `FLAGS` the default value (before execute the above code), like `FLAGS.LOG = false` and the log function would be disabled by default, and still you could enabled it calling `console.log.setEnabled(true)`

answered Nov 12 '16 at 18:17

Gabriel De Oliveira Rohden

469 4 12

do you think that this could be used to enable the console.log in production environment on the fly? like open Chrome console, run `console.log.setEnabled(true)` and start seeing logs – Rodrigo Assis Sep 15 '17 at 18:28

1 @RodrigoAssis yes, it will work. I've created this just to do not lose the caller line and enable anywhere, but this isn't the best way to do it. The best way for logs is use the short circuit way like: `var debug = false; debug && console.log(1/3)` because you don't need to evaluate the log content if it isn't enabled (in this case `1/3` will not be evaluated), don't lose the caller line and can enable it easily too (if don't vars as consts). – Gabriel De Oliveira Rohden Sep 18 '17 at 17:06

My comprehensive solution to disable/override all `console.*` functions is here.

1

Of course, please make sure you are including it after checking necessary context. For example, only including in production release, it's not bombing any other crucial components etc.

Quoting it here:

```
"use strict";
(() => {
  var console = (window.console = window.console || {});
  [
    "assert", "clear", "count", "debug", "dir", "dirxml",
    "error", "exception", "group", "groupCollapsed", "groupEnd",
    "info", "log", "markTimeline", "profile", "profileEnd", "table",
    "time", "timeEnd", "timeStamp", "trace", "warn"
  ].forEach(method => {
    console[method] = () => {};
  });
  console.log("This message shouldn't be visible in console log");
})();
```

Run code snippet Expand snippet

edited May 29 '18 at 17:14

answered May 29 '18 at 16:52

kmonsoor

4,105 4 30 46

If you're using gulp, then you can use this plugin:

1

> Install this plugin with the command:
>
> ```
> npm install gulp-remove-logging
> ```
>
> Next, add this line to your gulpfile:

Lastly, add the configuration settings (see below) to your gulpfile.

Task Configuration

```
gulp.task("remove_logging", function() {
     return gulp.src("src/javascripts/**/*.js")
    .pipe(
      gulp_remove_logging()
    )
    .pipe(
      gulp.dest(
        "build/javascripts/"
      )
    ); });
```

answered Aug 18 '18 at 7:23

Andrew Nessin
490 9 14

1

A simplification of https://stackoverflow.com/a/46189791/871166

```
switch (process.env.LOG_LEVEL) {
  case 'ERROR':
    console.warn = function() {};
  case 'WARN':
    console.info = function() {};
  case 'INFO':
    console.log = function() {};
  case 'LOG':
    console.debug = function() {};
    console.dir = function() {};
}
```

answered Aug 1 at 22:50

7ynk3r
585 7 16

invocation if some global variable is set to false.

0

You could even do an ajax call (now and then) so you can change the log enabled/disabled behavior on the server which can be very interesting to enable debugging when facing an issue in a staging environment or such.

answered May 31 '12 at 14:35

Stijn Geukens
10.9k 7 56 95

A working example might help. – GuruM Feb 11 '13 at 6:47

I have not implemented such a solution, not did I see it. It's theoretical so far. – Stijn Geukens Feb 11 '13 at 8:34

0

You can use logeek, It allows you to control your log messages visibility. Here is how you do that:

```
<script src="bower_components/dist/logeek.js"></script>

logeek.show('security');

logeek('some message').at('copy');       //this won't be logged
logeek('other message').at('secturity'); //this would be logged
```

You can also `logeek.show('nothing')` to totally disable every log message.

answered Jun 12 '16 at 20:44

Iman Mohamadi
3,568 25 30

0

After doing some research and development for this problem, I came across this solution which will hide warnings/Errors/Logs as per your choice.

```
(function () {
```

```
        window['console']['warn'] = function () { };
        this.addEventListener('load', function () {
            console.warn('Something bad happened.');
            window['console']['warn'] = function () { };
        });
    };
})();
```

Add this code before JQuery plugin (e.g /../jquery.min.js) even as this is JavaScript code that does not require JQuery. Because some warnings are in JQuery itself.

Thanks!!

answered Sep 20 at 10:20

Sunny_Sid
182 1 11