# WebRTC GitHub repo developer's guide



The repo is at github.com/webrtc and the samples can be viewed live at webrtc.github.io/samples.

**Before contributing code, please check the CONTRIBUTING file.**

## Validation

HTML, CSS and JavaScript should follow the Google style guide.

All JavaScript must pass ESLint validation and follow the options in samples/.eslintrc. ESLint plugins are available for Sublime and other editors, and ESLint validation can be run as a Grunt task. If necessary (rarely), include additional ESLint directives within individual JavaScript files.

All JavaScript files must have `'use strict';` at the top, to invoke global strict mode.

All HTML must pass HTMLHint checking. Likewise, CSSLint must not find errors (though some warnings are acceptable).

The W3C validators for CSS and HTML are also useful.

## Project structure and style

For each new code sample, create a new directory in the appropriate subdirectory of src/content.

Put JavaScript in a separate file. Each demo should have a directory structure like this:

index.html
/js/main.js

If your CSS has more than about five rules, put it in a separate file:

/css/main.css

Declare all global variables and element variables at the top of main.js.

Put all conditional statements in a block, even if only one line.

By default use === and !== for testing equality. This makes equality testing explicit, without coercion.

Put a semicolon at the end of every statement. This is easy to forget with handlers, for example:

```
stream.onFoo = function(){...}; // this is a statement
```

Prefer [] to new Array(), for example:

```
const myArray = [];
```

Prefer dot notation. For example use this:

```
pcConfig.iceServers[i].url
```

...instead of this:

```
pcConfig.iceServers[i]['url'];
```

Double quote string values in HTML, single quote in JavaScript (as per Google style guide).

Variables are camelCase. In general, don't use hyphens in names.

Indent with two spaces. When function calls or tests run onto multiple lines, indent with four spaces:

```
const iceServers = createIceServers(turnServer.uris,
    turnServer.username, turnServer.password);

alert('Failed to create data channel. ' +
    'You need Chrome 25 or later with --enable-data-channels flag');
```

```
if (sctpSelect.checked &&
    (detectedBrowser === 'chrome' && detectedVersion >= 31) ||
    detectedBrowser === 'firefox') {

...

}
```

## JavaScript quirks

Watch out not to miss the `const` or `let` keyword in variable declarations: variables used in a function without `var` are in global scope.

To put a variable in global scope, make it explicit by qualifying the name with `window`, for example:

```
const video = window.video = document.querySelector('video');

function successCallback(stream) {
  window.stream = stream; // stream available to console
  ...
}
```

Likewise, be explicit when referring to objects belonging to window, for example `window.performance`.

## Working with GitHub

For minor code changes, fork the repo and make a pull request from your GitHub account.

For major/ongoing changes, create a branch and issue pull requests from that. Please delete the branch once you've finished working on it.

Once a pull request has been generated the assigned reviewer should merge it to master once all the comments have been addressed.

Git log not clear enough to interpret? Try `git log --graph --decorate --oneline`

Squash commits? Here is how.

This page has a useful flowchart for getting out of Git tangles. Here is another page.

All events for the repo can be viewed at api.github.com/repos/GoogleChrome/webrtc/events. This includes items that may not show up in the commits log.

### Pull request won't merge automatically?

1. git pull master
2. Check out branch
3. git merge master
4. Edit the merge conflict
5. git add <changed files>
6. git commit -am 'merge master'
7. git push

### GitHub Pages

When you make a pull request, please include links to updated samples running on GitHub Pages on your repo. For example: samdutton.github.io/webrtc/src/content/getusermedia/gum. This enables reviewers to check functionality without needing to pull your branch and run the changes locally.

## Unwanted commits in pull request?

1. Checkout the branch in question
2. git rebase -i "SHA1 of the commit you do not want to see"
    a. squash all commits except the topmost one (assuming it's the commit from step 2)
3. Fix all the conflicts if any
    a. fix conflict
    b. git add filename
    c. git rebase --continue
    d. Rinse and repeat until done
    e. Note: Make sure to just leave the commit message you want in the end f
4. git push --force origin <branch>

Note: You effectively lose commit history by doing this as you change the reference commit.

## Bower

```
sudo npm install bower
```

Dependency version handing: https://github.com/npm/node-semver

Run `bower update` in the root folder of the project or sub project (e.g. testrtc), usually if a folder contains a bower.json file it's considered root.

## Validation with Grunt plugins and Travis

This project has been set up to enable automated testing with Grunt plugins and Travis.

### Grunt

Grunt uses Node.js to automate tasks written in JavaScript. Two files in a project's top level code directory provide information for Grunt to be used with that project:

- package.json describes dependencies and other project data.
- Gruntfile.js defines options for the tasks which can be run with Grunt for the project.

  For example, the Gruntfile for the WebRTC sample repo gives options for three Grunt plugins: CSSLint, HTMLHint, ESLint. These are predefined tasks used to validate code and check coding style. It's also possible to write your own custom tasks using JavaScript – for unit testing, for example.

  The `grunt` command can be called with or without parameters, depending on settings defined in the project's Gruntfile. For example, with the WebRTC sample repo, `grunt htmlhint` will run the HTMLHint Grunt plugin. Calling `grunt` on its own runs all the tasks defined by the `grunt.registerTask()` method in the Gruntfile.js:

```
grunt.registerTask('default',
  ['csslint', 'htmlhint', 'eslint']);
```

The grunt-eslint documentation has simple instructions on how to set up Grunt with ESlint: other plugins are installed in the same way. The Grunt plugins page has more information about popular plugins. Options for Grunt plugins can generally be defined in a project's Gruntfile as well as in config files (such as .eslintrc and .csslintrc).

### Travis

Once a GitHub project has been configured on the Travis website, the Travis server can hook into the GitHub API to respond to events for that project. For example, the public project travis-ci.org/GoogleChrome/webrtc corresponds to the GitHub project github.com/GoogleChrome/webrtc. (Travis can also be used with private projects.)

Once Travis has been given access to the GitHub API for a project, GitHub push or pull events trigger testing (and potentially other build steps) to be done by the Travis server. Whenever the WebRTC samples project is pushed or pulled, the Travis server runs the following steps (this build log gives more detail):

1. Install and test Node dependencies as defined by package.json in the project's top level directory.

2. Call the command defined by the scripts.test property in package.json, which for this project is as follows:

    ```
    "scripts": {
      "test": "grunt --verbose"
    }
    ```

    As described above, calling the `grunt` command on its own for this project will run three tasks as defined in Gruntfile.js: CSSLint, HTMLHint and ESLint.

3. Once all tests have been completed, Travis reports success or failure. Travis can be configured to provide various types of notification. Currently this project is set to send an email to the project owner (dutton@google.com). Travis also updates a project status icon which can be included in documentation on GitHub:

    ![Travis](https://travis-ci.org/webrtc/samples.svg?branch=master)

    
    

The travis.yml file at the project's top level directory defines the language environment, which in this case is Node.js, and can be used for other settings.

Travis can also be used to automate build tasks, though this is not currently used by the WebRTC samples project.