

# Running PhantomJS for Unit Testing with the auxiliary of NodeJS and Grunt

Installation and testing

July 20, 2018

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Installation</b>	<b>3</b>
2.1	Creating a Project folder . . . . .	3
2.2	Installing NodeJS . . . . .	3
2.3	Installing Grunt . . . . .	3
2.4	Changing Package.json content . . . . .	4
2.5	Installing Node Modules . . . . .	4
2.6	Setting up Gruntfile.JS for unit testing . . . . .	4
<b>3</b>	<b>Testing examples</b>	<b>5</b>
3.1	First test - Calculator . . . . .	5
3.2	Second Test - Concatenating strings . . . . .	9
<b>4</b>	<b>Gerrit and Jenkins (Graphical interface)</b>	<b>13</b>

# 1 Introduction

This report is designated to demonstrate the process of installation of **NodeJS**, **Grunt**(**PhantomJS pre-built**) with the purpose of unit testing with **Qunit**.

Ideas and thoughts of implementing the result of this set of programs in **Jenkins Graphical Interface** will be presented at the end of the report.

## 2 Installation

### 2.1 Creating a Project folder

- Create a **Project** folder with the following configuration:
  - **Project** - Main Project Folder
    - \* **src** - Plugin Source, Project Files, Etc
    - \* **tests** - Folder which contains the unit tests
      - **lib** - Library for plugins. Inside this folder are just examples of plugins that can be used. None have to be installed.
      - **index.html** - The **Qunit** test we'll be using. It can be 1 file or several, in case of doing separate tests. It must be indicated in the **Gruntfile.js** in order to run.
      - **tests.js** - The functions which are to be tested.
    - \* **Gruntfile.js** - Grunt runner which indicates tests to run.
    - \* **package.json** - To specify our project dependencies.

```
└─ project
   ├── src
   ├── tests
   │   ├── lib
   │   │   ├── jquery-1.x.x.min.js
   │   │   ├── qunit-1.10.0.js
   │   │   └── qunit-1.10.0.css
   │   ├── index.html
   │   └── tests.js
   ├── Gruntfile.js
   ├── package.json
   └── ...
```

Figure 1: Structure of Project Folder.

### 2.2 Installing NodeJS

- Install NodeJS - <https://nodejs.org/en/>. Install version 8.11.3 LTS (Recommended for most users).

### 2.3 Installing Grunt

- Open Command prompt and go to the project folder directory.
- Write "**npm install -g grunt-cli**".

## 2.4 Changing Package.json content

- In `package.json`, change the content to:

```
{
  "name": "projectName",
  "version": "1.0.0",
  "devDependencies": {
    "grunt": "~0.4.1",
    "grunt-contrib-qunit": ">=0.2.1",
    "grunt-contrib-watch": ">=0.3.1"
  }
}
```

## 2.5 Installing Node Modules

- While in the project directory write "**npm install**" in Command Prompt.

## 2.6 Setting up Gruntfile.JS for unit testing

- The **Gruntfile.js** should have the following configuration for qunit:

```
module.exports = function(grunt) {
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'), // the package file to use

    qunit: {
      all: ['tests/*.html']
    }
  });
  // load up your plugins

  // register one or more task lists (you should ALWAYS have a "default" task list)
  grunt.loadNpmTasks('grunt-contrib-qunit');
  // ...

  grunt.registerTask('default', ['qunit']);
  //grunt.registerTask('taskName', ['taskToRun', 'anotherTask']);
};
```

- In this specific setup, it is set to run every test in the tests folder. In order to run specific tests, the qunit parameter must be changed:

```
...
qunit:{
  all:['tests/test1.html','tests/test2.html','tests/test3.html', ...etc]
}
...
```

## 3 Testing examples

### 3.1 First test - Calculator

- Using a simple adding function in javascript, which consists of adding any given numbers:

```
function calc(number1, number2)
{
    var result= number1 + number2;
    return result;
}
```

The name of this file is **Calc.js**, located in **project/tests**.

- In the **index.html** file, the content is:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<link rel="stylesheet" href="https://code.jquery.com/qunit/qunit-2.6.1.css">
<script src="https://code.jquery.com/qunit/qunit-2.6.1.js"></script>
<script src="Calc.js" ></script> //Source of the function to test.

<script>
QUnit.test("Calculador test", function( assert ){ //Run Qunit
var num1 = 2; //Variable 1 of our Qunit test
var num2 = 3; //Variable 2 of our Qunit test

assert.equal(calc(num1,num2),5); //First test

assert.equal(calc(num1,num2),50); //Second test

assert.equal(calc(num2,num2),6); //Third test

assert.equal(calc(num1,num1),4) //Fourth test

});

</script>
</head>
<body>
<div id="qunit"></div>
</body>
</html>
}
```

It is not needed to specify to run the index.html test, since the **Gruntfile.js** is set to run every test inside the tests folder.

In order to test the function **calc**, a series of tests was set, 1 of which is programmed to fail.

In the first test it is added 2 and 3 and is expected a result of 5, which returns true, as expected.

In the second test it is expected 50. This expected result is intentionally inducing an error, returning false.

In the third test it is added 3 and 3 and is expected 6, which returns true.

In the third test it is added 2 and 2 and is expected 4, which returns true.

In conclusion, the overall expected result of running this series of tests is 1 failure. In order to run the test, set the directory to the project folder. Afterwards, write "grunt" and it will run the Qunit test.



```
Command Prompt
C:\Users\ricardo.a.rodriques\Desktop\project>grunt
Running "qunit:all" (qunit) task
Testing tests/index.html F
>> Calculator Test
>> Message: null
>> Actual: 5
>> Expected: 50
>> file:///C:/Users/ricardo.a.rodriques/Desktop/project/tests/index.html:16:16

Warning: 1 tests completed with 1 failed, 0 skipped, and 0 todo.
4 assertions (in 24ms), passed: 3, failed: 1 Use --force to continue.

Aborted due to warnings.
C:\Users\ricardo.a.rodriques\Desktop\project>
```

Figure 2: Result after first series of tests

It can be observed that it occurred an error in the index.html file. To check which error was in specific we open the index.html file.

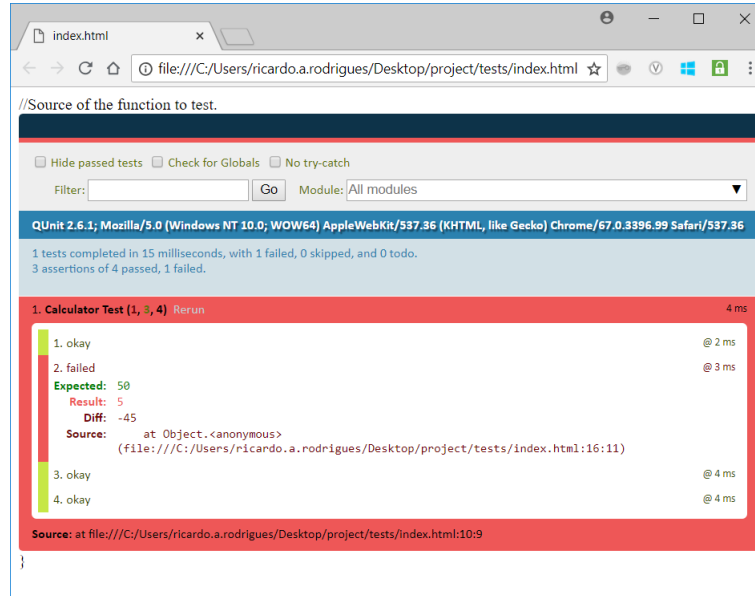


Figure 3: Result of the index.html file

The second test failed as expected, meaning that the function executed as planned. The **index.html** file was edited to expect a result of 5. In the Command Prompt, write "**grunt**" again, after editing the **index.html** file.

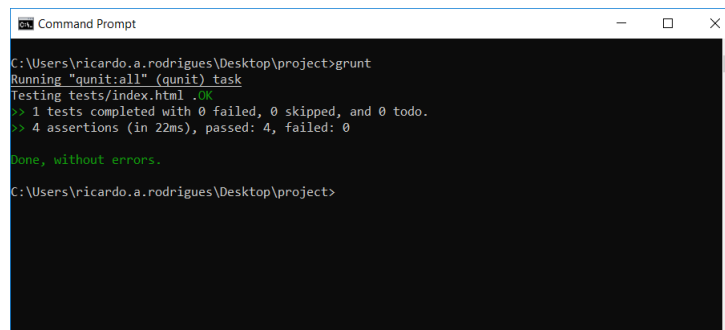


Figure 4: Result after second series of tests



Every assertion returned true, meaning that the function is executing as planned. When an error was induced, one of the assertions returned false and when corrected, it returned true. We can also observe the same result in the index.html file.

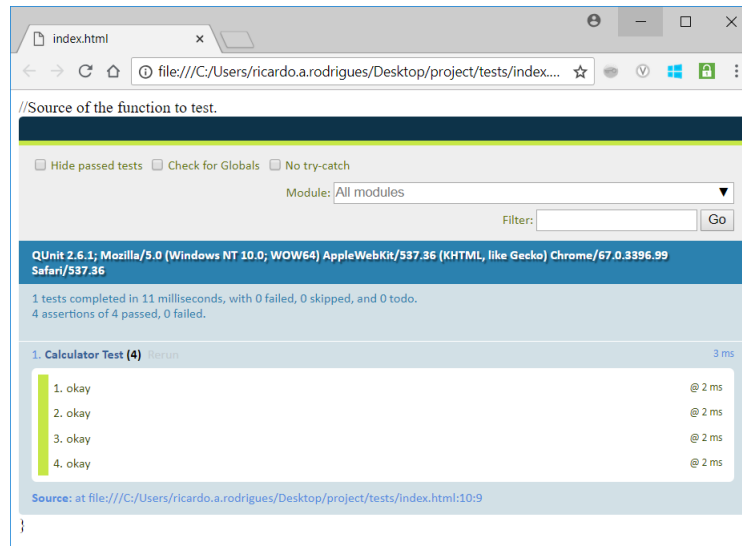


Figure 5: Result of the index.html file after correcting it

It yields the same results as in Command Prompt.

### 3.2 Second Test - Concatenating strings

The following function (**funcstring**) concatenates two different strings into one:

```
function string_test(string1, string2)
{
    var result= string1 + string2;
    return result;
}
```

In order to test **funcstring**, it was created a new HTML file (**stringtest.html**) with the following content:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<link rel="stylesheet" href="https://code.jquery.com/qunit/qunit-2.6.1.css">
<script src="https://code.jquery.com/qunit/qunit-2.6.1.js"></script>
<script src="funcstring.js" ></script>

<script>
QUnit.test("String test", function( assert ){
    var string_example1 = "kapa";
    var string_example2 = "kopo";
    var string_example3 = "kepe"
    assert.equal(string_test(string_example1,string_example2),"kapakopo");

    assert.equal(string_test(string_example2,string_example1),"kopokapa");

    assert.equal(string_test(string_example1,string_example3),"kapakepe");
});

</script>
</head>
<body>

<div id="qunit"></div>

</body>
</html>

}
```

For this test, the **Gruntfile.js** was edited to only run this test:

```
    module.exports = function(grunt) {
    grunt.initConfig({
      pkg: grunt.file.readJSON('package.json'), // the package file to use

      qunit: {

        all: ['tests/stringtest.html'],

      },

      watch: {
        files: ['tests/*.js', 'tests/*.html', 'src/*.js'],
        tasks: ['qunit']
      }
    });

    // load up your plugins

    // register one or more task lists (ALWAYS have a "default" task list)

    grunt.loadNpmTasks('grunt-contrib-qunit');
    grunt.loadNpmTasks('grunt-contrib-watch');

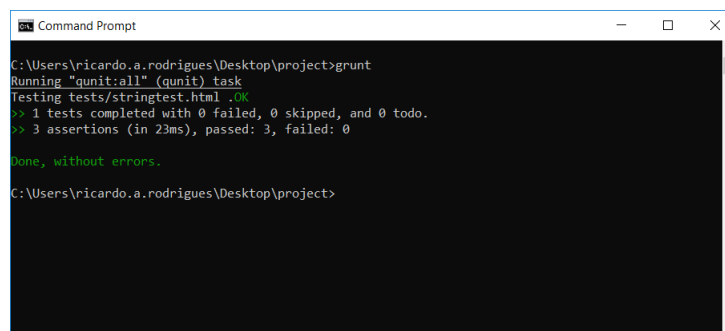
    // ...

    grunt.registerTask('default', ['qunit']);

  };

```

Open Command Prompt and write "**grunt**".



```

C:\Users\ricardo.a.rodriques\Desktop\project>grunt
Running "qunit:all" (qunit) task
Testing tests/stringtest.html .OK
>> 1 tests completed with 0 failed, 0 skipped, and 0 todo.
>> 3 assertions (in 23ms), passed: 3, failed: 0

Done, without errors.
C:\Users\ricardo.a.rodriques\Desktop\project>

```

Figure 6: First series of tests in stringtest

All three assertions returned true, meaning the functions executes as planned. It can also be verified through the **stringtest.html** file. All three assertions pass the **Qunit** tests.

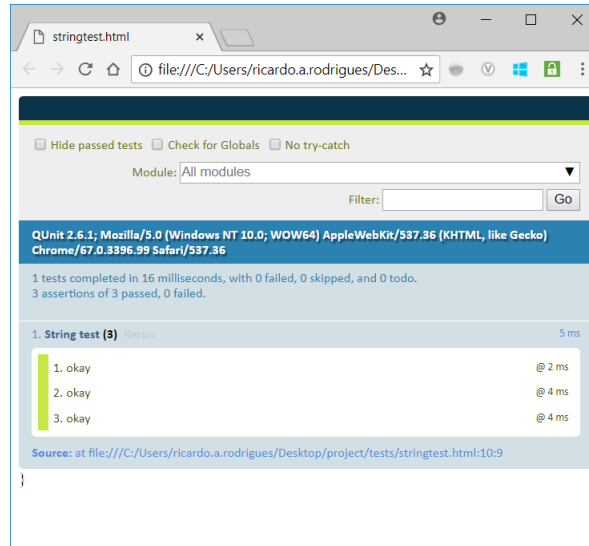


Figure 7: First series of tests in **stringtest.html** file

A new assertion will be added to the **stringtest.html** file to induce an error, just like the previous test:

```
...
assert.equal(string_test(string_example3,string_example2),"not_expected");
//Added test
...
```

Execute "**grunt**" in Command Prompt:



Figure 8: Second series of tests in **stringtest**

As supposed, an error originated. Observing the HTML file:

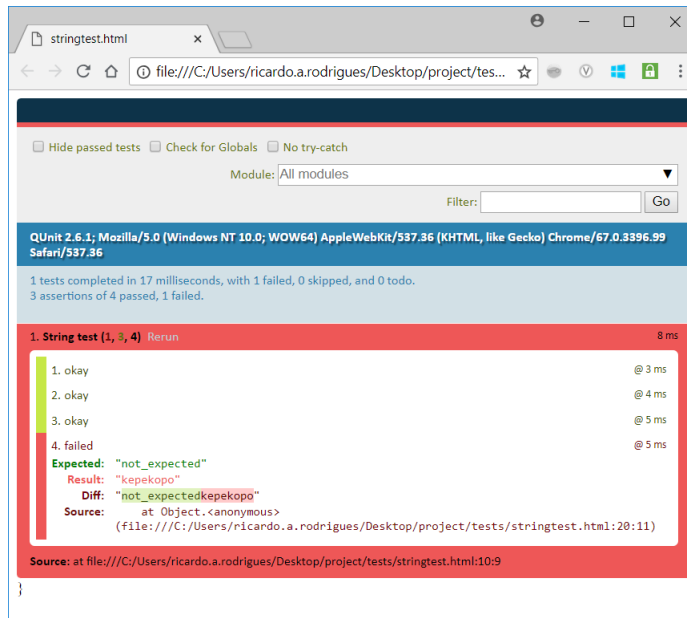


Figure 9: Second series of tests in **stringtest.html**

## 4 Gerrit and Jenkins (Graphical interface)

This set of programs is in development for implementing in **Gerrit** and **Jenkins** (**Graphical interface**). It is in pause at the moment due to a permission error; **CHMOD** in linux does not make it possible to change permissions of the files in need, nor CACLS in windows. In the near future, a **PUSH** through a **linux virtual machine** is viewed as possibility, but due to the large files of linux (Linux ISO, between 1-2 GB), it has been delayed.