* [Ht2: Git](#HT2_Git)
* [Ht3: Automation:](#HT3_Automation)
* [Ht4-5: Dependency, Shell Scripting](#HT4_5_Dependency)
* [Ht6: External Dependencies:](#HT6_ExternalDependencies)
* [Ht7: Lint](#HT7_Lint)
* [Ht8, 9: JavaScript Gotchas](#HT8_9)
* [Ht10: Localhost server](#HT10)
* [Ht11: Assertions](#HT11)  Chai
* [Ht12: Test frameworks](#HT12)  Mocha
* [Ht13: Cross-Browser Test Runner – Karma](#HT13)
* [Ht14: Automating](#HT14) Karma
* [Ht15: Testing Android, iOS, IE, and More](#HT15)
* [Ht16: Modular Tests:](#HT16) commonJS
* [Ht17: Modular in Production](#HT17) : **browserify**
* [**Ht18: The Build Step:**](#HT18) **browserify**
* **[Ht19: Frontend modules: modified Chai for Karma](#HT19)**

Tools:

**Test Automation components:**

1. **Assertion library - Chai Ht11**
   * + - This makes it a lot easier to test your code, so you don't have to do thousands of if statements.
       - Use **Chai** assertion to verify if function run correctly.
2. **Test framework - Mocha Ht12**
   * + - Testing frameworks are used to organize and execute tests.
       - Use “desc” , and “task” to group all the test tasks in Jakefile.js
3. **Cross-Browser Test Runner - Karma Ht13**
   * + - Testing Environments are the places where you run your tests.
       - Karma is like a test server to run and test js in multiple browsers.

**Ht2: Git**

1. **Ls –la**: list all files including the hidden files
2. **git version**: check version
3. **git init**: To initial git folder
4. **git status**: To check if there are unsubmit files.
5. **git add** . : To track unsubmit files.
6. **git reset:** To reset the tracked files.
7. **sudo git reset -**-hard to reset files to the last checked in state
8. **git commit --**am “initial commit” **a**: all files, **m**: commit message
9. **gitk –**all: See what had gone to git.
10. **git log**: to see all the log for submitting.
11. **git remote add origin git@github.com:kalarsu/javascript.git**
12. **git push -u origin master:** The push command tells Git where to put our commits. The name of our remote is origin and the default local branch name is master. The -u tells Git to remember the parameters, so that next time we can simply run git push and Git will know what to do.
13. **git push --set-upstream origin master:** To push the current branch and set the remote as upstream, use
14. git pull origin master: to pull changes to local
15. **git diff HEAD**: what is different from our last commit
16. git diff –staged:
17. git remote set-url origin <https://github.com/kalarsu/javascript.git>
18. git branch --set-upstream-to=origin/<branch> master: you wish to set tracking information for this branch you can do so with

**Ht3: Automation: To create a automated build.**

For other people to come in and to make a build from an existing system with software installed. Here is the automation to rebuild the system from scratch.

1. New javascript file build.js, **to write an automated build**.

(**function**(){  
 **"use strict"**;*//help javascript prevent errors, not allow sloppy coding* **console**.log(**"BUILD OK"**);  
}());

>node build.js , will see the result.

>git status

>git add . (. Means all files)

>git commit –am “Got a basic ‘hello world’ program running”

>git push

1. Build automation needs:
   1. Self-Documentation
   2. Command – Line Processing
   3. Dependency Resolution
   4. Code, Not Configuration
   5. Straightforward and Simple
   6. There are tools like Grunt, Gulp and Jake
2. Sudo npm install jake –g
3. Rename build.js to Jakefile.js
4. **>jake** this will run the Jakefile.js, but couldn’t find default task
5. in Jakefile.js add default task as following:

(**function**(){  
 **"use strict"**;*//help javascript prevent errors, not allow sloppy coding* **console**.log(**"\n\nBUILD OK"**);  
  
 task(**"default"**, **function**(){  
 **console**.log(**"Hello, I am the default task"**);  
 });  
}());

1. **>jake** this time, it will run good.
2. Add in a new task as following:

task(**"gooble"**, **function**(){  
 **console**.log(**"gooble task"**);  
});

>jake gooble this will run task gooble

1. Add desc(“Default Task”) before default task, This will be documentation for that task.

desc(**"default task"**);*//documentation for following task*task(**"default"**, **function**(){  
 **console**.log(**"Hello, I am the default task"**);  
});

**>jake –T or >jake** –tasks , this will sow all the tasks with description.

This is what it meant, self-documentation.

1. Clean the code as following:

(**function**(){  
 **"use strict"**;*//help javascript prevent errors, not allow sloppy coding* desc(**"default build"**);*//documentation for following task, >jake --tasks or >jake -T will show all the tasks, this is what it meant self-documentation* task(**"default"**, **function**(){  
 **console**.log(**"\n\nBUILD OK"**);  
 });  
}());

1. Check in the code

**Ht4: Dependency**

1. When moving the code from one machine to another, in order to make **nodeJS** and **jake** work, the other machine has to install all the dependencies. That was not automation.
2. There are 2 dependency management strategies:
   1. Automatically install dependencies from somewhere else. **Pro**: Prevents repository bloat.
   2. Include dependencies in the source code repository.**Pro:** Ensures correct dependency version is available.
3. So we don’t use sudo npm install jake **–g.** Because when other people try to download the source code, then have to install all the dependencies that’s not automated build.
4. Steps to re-set up environment that other person doesn’t need to install anything just **git pull** then can run on another machine:
   1. **sudo npm uninstall jake –g**
   2. **npm init :** this will generate package.json. Ref: docs.npmjs.com/files/package.json
   3. **clean the code as following**

{  
 **"name"**: **"letscodejavascript"**,  
 **"version"**: **"1.0.0"**,  
 **"private"**: **true**

}

//private: true , **this will tell npm if someone try to publish it it will cause error**

* 1. **sudo npm install jake --ignore-scripts --save-dev:**
     1. Jake is a automation build tool
     2. install locally and -ignore-scripts don’t check in binary files from jake to git.
     3. --save-dev: Jake will save the setting into package.json.
  2. **npm rebuild**: go through and run all installed script and run all installed files
  3. **sudo node\_modules/.bin/jake :** since jake wasn’t installed –g globaly, use this command to run.

|  |
| --- |
| * 1. At other machine to download the code from git and run:      1. **git pull**      2. **npm rebuild**      3. **node\_modules/.bin/jake** |

**Ht5: Shell Scripting:**

* 1. to avoid typing **sudo node\_modules/.bin/jake** all the time, adding following shell script:
     1. new file: jake.sh, type in **node\_modules/.bin/jake $\*** (**$\*** will pass in parameter in like --help --tasks or -T )
     2. because this file doesn’t have permission to execute, need to do:

**chmod +x jake.**sh (chmod : change mode, +x: adding execution permission)

* + 1. **./jake.sh** : now this will run
    2. **new jake.cmd** and type in node\_modules\.bin\jake %\*, in command line just type >jake. This is **for windows user**
  1. sudo git commit --amend, and **shift+z+z** to save the changes, this is to modify from previous commit.
  2. in jake.cmd file, add at the first line as #!/bin/sh : this is a build in command prom for Unix, it tells Unix which command to use when running shell script. It’s using #!/bin/sh to understand $\*, so the jake.sh looks like following now.

#!/bin/sh  
  
node\_modules/.bin/jake $\*

* 1. in jake.cmd modify as following, @echo off will not show node\_modules\.bin\jake path in the command line

@echo off  
  
node\_modules\.bin\jake %\*

|  |
| --- |
| * 1. In another machine, when need to recover back to a clean repository, do fillowing   >sudo git clean –fdx:  >sudo git reset --hard: reset files to last commit status  >git pull ,to get all the files from git and then  >npm rebuild |

* + 1. clean: delete untracked files ,
    2. -fdx: f: **f**orce clean to occur, d: remove directories too, x: remove excluded(gitignore would files too)

This will reset to clean files, when just git pull from git repository. So before running jake , need to do >npm rebuild.

* 1. New .gitignore file, and type in following

#Mac OS X  
.DS\_Store  
  
#WebStorm  
.idea/workspace.xml  
  
# npm  
node\_modules/\*\*/.bin/

This is to ignore unnecessary file to check in.

* 1. Sudo rm –rf node\_modules/.bin Delete unnecessary .bin files
     1. rm: delete files
     2. –rf: **r**ecursively delete directories, **f**orce everything to delete
  2. Add npm rebuild in the second line for both **jake.sh** and **jake.cmd**. So when git pull a clean repositories and run >sudo ./jake.sh or >jake(for windows), it will automatic build the project (automatically install all the dependencies).

@echo off  
  
echo Building npm modules:

npm rebuild  
  
node\_modules\.bin\jake $\*

* 1. Add following into jake.sh and jake.cmd

**[** ! **-f** node\_modules/.bin/jake **] &&** echo **"Building npm modules:" &&** npm rebuild :

* For ***jake.sh*** file: if node\_modules/.bin/jake doesn’t exists then print out message and npm rebuild, and npm rebuild. Complete code as following:

#!/bin/sh  
  
[ ! -f node\_modules/.bin/jake ] && echo "Building npm modules:" && npm rebuild  
  
node\_modules/.bin/jake $\*

* For **jake.cmd** (Windows) code as following:

@echo off  
  
if not exist node\_modules\.bin\jake(  
 echo building npm modules:  
 call npm rebuild  
)  
  
call node\_modules\.bin\jake %\*

|  |
| --- |
| Now , In another machine, when need to recover back to a clean repository, do following:  >sudo git clean -fdx:  >sudo git reset --hard  >git pull  >sudo ./jake.sh : Jake will automatically rebuild  >sudo ./jake.sh –T : list all the tasks |

**Ht6: External Dependencies:**

1. When rebuild, we should also check if node version matching. Adding new task(“version”) in jakefile.js as following:

(**function**(){  
 **"use strict"**;

desc(**"default build"**);task(**"default"**, [**"version"**], **function**(){ //run version before running default  
 **console**.log(**"\n\nBUILD OK"**);  
 });  
  
 desc(**"Check Node version"**);  
 task(**"version"**, **function**(){  
 **console**.log(**"Checking Node version: ."**);  
 **let** actualVersion = process.**version**; **if**(actualVersion !== EXPECTED\_NODE\_VERSION){  
 fail(**"Incorrect Node version: expedted:"** + EXPECTED\_NODE\_VERSION + **", but was: "**+actualVersion); *//fail is a Jake function* }  
 });

}());

>./jake.sh –T to see all the tasks

>./jake.sh version to run task version

1. Add “engines”: {“node”: “5.5.0”} to *package.json*, for Jakefile.js to fetch and verify node version. Package.json looks as following:

(ref: <https://docs.npmjs.com/files/package.json> for engine)

{  
 **"name"**: **"letscodejavascript"**,  
 **"version"**: **"1.0.0"**,  
 **"private"**: **true**,  
 **"engines"**:{  
 **"node"**: **"5.5.0"** },  
 **"devDependencies"**: {  
 **"jake"**: **"^8.0.12"** }  
}

1. In **Jakefile.js** change code as following. (get node version from package.json)

(**function**(){  
 **"use strict"**;*//help javascript prevent errors, not allow sloppy coding* desc(**"default build"**);*//documentation for following task, >jake --tasks or >jake -T will show all the tasks, this is what it meant self-documentation* task(**"default"**, [**"version"**], **function**(){ *//run "version" task before running default* **console**.log(**"\n\nBUILD OK"**);  
 });  
  
 desc(**"Check Node version"**);  
 task(**"version"**, **function**(){  
 **console**.log(**"Checking Node version: ."**);  
  
  
 **var** packageJson = require(**"./package.json"**); *//require was build into node* **var** expectedVersion = **"v"** + packageJson.engines.node;  
  
 **let** actualVersion = process.version; *//current node version* **if**(actualVersion !== expectedVersion){  
 fail(**"Incorrect Node version: expedted:"** + expectedVersion + **", but was: "**+actualVersion); *//fail is a Jake function* }  
 });  
  
}());

1. sudo npm install **semver** --ignore-scripts --save-dev:

- semver is a parser for node for parsing version number.

- <https://github.com/npm/node-semver>

- It will check 2 version number is equal and ignore the first letter. A leading "=" or "v" character is stripped off and ignored.

* 1. --ignore-scripts: because we would like to check into git
  2. --save-dev: save as development dependency

1. git add ., git commit –am “message”: check in semver
2. sudo npm rebuild: to see if there is dependency we need to ignore
3. git status: looks like not no file need to be ignore
4. In Jakefile.js, add var semver = require("semver"); at the top, and in if statement change to if( semver.neq(actualVersion, expectedVersion) ) //neq = not equal, this will ignore the first “v” character.

(**function**(){  
 **"use strict"**;*//help javascript prevent errors, not allow sloppy coding* **var** semver = require(**"semver"**);  
 desc(**"default build**

task(**"default"**, [**"version"**], **function**(){

**console**.log(**"\n\nBUILD OK"**);  
 });  
  
 desc(**"Check Node version"**);  
 task(**"version"**, **function**(){  
 **console**.log(**"Checking Node version: ."**);  
  
 **var** packageJson = require(**"./package.json"**); *//require was build into node* **var** expectedVersion = packageJson.engines.node;  
  
 **let** actualVersion = process.**version**; *//current node version* **if**( semver.neq(actualVersion, expectedVersion) ){*//neq : not equal* fail(**"Incorrect Node version: expedted:"** + expectedVersion + **", but was: "**+actualVersion); }  
 });  
}());

**Ht7: Lint:** Safe coding

* + - * Static code analysis
      * Link will look into source code and find error without running it.
      * <https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#JavaScript>

1. IIFE around every file : ( function(){}() ), to have a function expression and use () to execute right away. And use ( ………..) to wrap it called IIFE. So nothing run at global scope.
2. “use strict”; This tell JavaScript to check error at run time.



1. sudo npm install **jshint** --ignore-scripts --save-dev
2. git add ., git commit –am “message”: check in jshint
3. sudo npm rebuild: to see if there is dependency we need to ignore
4. git status: looks like not no file need to be ignore
5. use sudo node\_modules/.bin/jshint to run
6. sudo node\_modules/.bin/jshint --help
7. sudo node\_modules/.bin/jshint Jakefile.js Check if there is error in Jakefile.js
8. Add task into Jakefile.js as following: so we don’t have to run line 6 long script.

**Ref:** <http://jakejs.com/docs#running_shell-commands:_`jakeexec`_and_`jake.create_exec`>

task(**"default"**, [**"version"**, **"lint"**], **function**(){ *//run "version","lint" task before running default* ***console***.log(**"\n\nBUILD OK"**);  
});

desc(“Lint Javascript Code”);

task(**"lint"**, **function**(){  
 ***console***.log(**"Linting Javascript: ."**);  
 ***jake***.exec(**"node node\_modules/jshint/bin/jshint Jakefile.js"**, {**interactive**: **true**}, complete);  
}, { **async**: **true** });

* + - * complete: *//run complete funtion when it's done*
      * async: true: *tell jake not to end the task until the complete function is called*

1. sudo ./jake.sh to see if lint task is working good to check jakefile.js file error.
2. Running jake.exec is a bit slow, so let’s use **simplebuild** plugin as following, to simplify the script.
   1. ref: https://www.npmjs.com/package/simplebuild-jshint
   2. sudo npm install **simplebuild**-jshint --ignore-scripts --save-dev
   3. document under https://www.npmjs.com/package/simplebuild-jshint
3. git add ., git commit –am “check in simplebuild”
4. sudo npm rebuild: to see if there is dependency we need to ignore
5. git status: looks like not no file need to be ignore
6. In **Jakefile.js**:
   * + - var jshint = require("simplebuild-jshint")
       - Command out jake.exec(“….”)
       - Under task (“Lint”, function(){ ……. }, update to following:

process.**stdout**.write(**"Linting Javascript: "**);//using global process instead of console.log

jshint.checkFiles({  
 **files**: **"Jakefile.js"**,  
 **options**: {  
 **bitwise**: **true**, *//to check if bitwise issue single & or | typeo, detail at http://jshint.com/docs/options/* **eqeqeq** : **true** },  
 **globals**: {}  
}, complete, fail); *//complete, fail are Jake build in*

**Ht8, 9: JavaScript Gotchas**

1. Using IFEE, to prevent creating global scope variable, and execute right away.

(function(){

}());

1. Variable scope is defined for closest function , not other {} - elastic scope like if statement.

Function test(){

var xx = true;

If(xx == true){

var xx = “false”;

}

console.log(“xx=”+xx); // result xx=false

}

In this case yy doesn’t belongs to if {} , instead, it belong to test() function scope.

1. In jakefile.js, use **bitwise: true** //*to check if bitwise issue (single & or | typo), detail at* *<http://jshint.com/docs/options/>*
   * + - “use strict” in javasscript will do the run time checking, but using Lint within jakefile.js will check before run time.
       - Now the code with bitwise looks like:

task(**"lint"**, **function**(){process.**stdout**.write(**"Linting Javascript: "**); jshint.checkFiles({  
 **files**: **"Jakefile.js"**,  
 **options**: {  
 **bitwise**: **true**  
 },  
 **globals**: {}  
 }, complete, fail);  
}, {**async**: **true**});

1. **curly: true** // always put curly braces around blocks
2. **eqeqeq** : **true** *//to see error caused by ==, instead, should use ===*

function test(){

var a = 0;

var b = [];

if(a==b) console.log(“Equal”); //this result will be Equal, so use === to avoid this

}

1. **freeze**: **true**,*//prohibits overwriting prototype of native object such as Array, and Date*
2. **latedef**: **"nofunc"** *// prohibits the use of a variable before it was defined, nofunc: not for function.* Note: Call a function before it declares is valid as following, because JavaScript hoisting function to the top.

**callme();**

function callme(){

}

However, using a variable before it declared is invalid as following. Variable will be hoisted to the top as well, but only var x will be hoisted not the value.

Console.log(x);

var x=19;

ref: http://www.w3schools.com/js/js\_hoisting.asp

1. strict: **true**;*// required to use "use strict" at the top*
2. undef: true, *// prohibits the use of explicitly undeclared variables.*

*This will start complain a lot of errs. So type in following in #10, #11*

1. node: true, browser: true // we are using node and browser
2. under globals:

**globals**: {  
 **desc**: **false**, // we sue desc, task, complete, fail, but we don’t change them  
 **task**: **false**,  
 **complete**: **false**,  
 **fail**: **false**}

//meaning we are not changing it to prevent desc, task, not defined.

However, we only use desc, task, complete, fail in jake.js file , so

Put them at the top of js file and command out as

*/\* globals desc: false, task: false, complete : false, fail: false \*/*

**Ht10: Localhost server: Automated Cross browser testing**

1. Create new folder src and new html file index.html for cross browsers testing. When hover this index.html file, top right corner will appear different browser icon and just click either of them to test this file.
2. However, we will install **http-server**, so the result will be closest to the production.
3. sudo npm install **http-server** --ignore-scripts --save-dev
4. ./node\_modules/.bin/http-server src : will activate the web server
5. according to the message show on terminal, <http://127.0.0.1:8080>, or <http://localhost:8080> : will launch the index.html page
6. sudo git add . , sudo git commit –am “message”, sudo npm rebuild
7. Adding a task “run” as following: //this will execute the command and run the web server

desc(**"Run a localhost server"**);  
task(**"run"**, **function**(){  
 ***jake***.exec(**"node node\_modules/http-server/bin/http-server src"**, {**interactive**: **true, async: true**}, complete);  
});

* + - * **interactive**: **true**: so we can see the output
      * **complete**: To run the complete function when it’s don.

Add jake:false in Jakefile.js on the top command line within globals as following:

* + - * This is to define jake as a global variable.

*/\* globals* ***jake:false****, desc:false, task:false, complete:false, fail:false \*/*

1. ./jake.sh run

**Test Automation components:**

1. **Assertion library - Chai Ht11**
   * + - This makes it a lot easier to test your code, so you don't have to do thousands of if statements.
2. **Test framework - Mocha Ht12**

Testing frameworks are used to organize and execute tests.

1. **Cross-Browser Test Runner - Karma Ht13**

Testing Environments are the places where you run your tests.

**Ht11: Assertions - Chai**

* + - 1. **Assertion : Allow you to check the result of a function**. Following assertEqual function is to check if the result is correct, but we don’t write our own assertion, we use popular plugin called “**Chai**”.

(**function** () {  
 **"use strict"**;  
  
 *assertEqual*(*add*(3,4), 7);  
   
 **function** *add*(a,b){  
 **return** a+b;  
 }  
   
 **function** *assertEqual*(actual, expected){  
 **if**(actual !== expected) **throw new** Error(**"Expected="** + expected + **", but got="**+actual);  
 }  
}());

1. modify jakefile.js, under lint, files: [ “Jakefile.js”, “src/\*\*/\*.js” ]
   * + - src/\*\*/\*.js: means all file under “src” folder and all the sub directory .js files, will be verify by lint
       - So, whatever js file modified and saved under src/ will run lint
       - sudo ./jake.sh to test
       - sudo node src/test.js to run javascript
2. Created an ***assertion*** function (test.js) to check if there are error:

*assertEqual*(*add*(3,4), 7)  
  
**function** *add*(a,b){  
 **return** a + b;  
}  
  
**function** *assertEqual*(actual, expected){  
 **if** (actual !== expected) **throw new** Error(**"Expected: "**+expected + **", but got: "** + actual);

*//throw new Error will stop the program if there is no try,catch*   
}

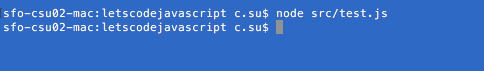
* 1. Sudo node src/test.js to test it

1. A popular third part assertion library: **Chai** <http://chaijs.com/>
   1. Sudo npm install **chai** --save-dev --ignore-scripts
   2. sudo git add . , sudo git commit –am “message”, sudo npm rebuild
   3. **var** assert = require(**"chai"**).assert; *//to include chai library*
   4. assert.equal(*add*(3,4),7); //chai assertion function, so we don’t have to write our own
   5. node src/test**.**js to test

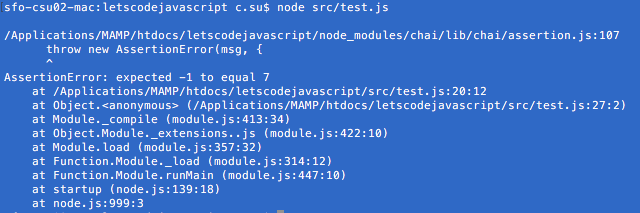
(**function** () {  
 **"use strict"**;

//basic addition  
 **var** assert = require(**"chai"**).**assert**;  
  
 assert.equal(*add*(3,4), 7); *//simeple one line chai replace following write your own assertEqual function  
  
 //assertEqual(add(3,4), 7);* **function** *add*(a,b){  
 **return** a+b;  
 }  
  
 *// function assertEqual(actual, expected){  
 // if(actual !== expected) throw new Error("Expected=" + expected + ", but got="+actual);  
 // }*}());

* + - * **When there is no error:**

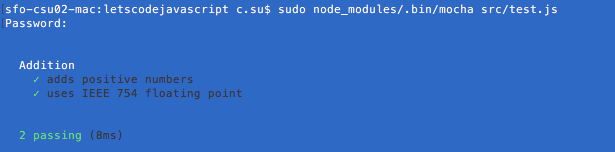


* + - * **When there is error. (change to a -b)**

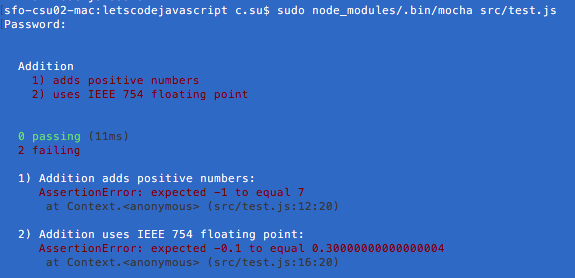
****

**Ht12: Test frameworks - Mocha**

1. A popular JavaScript test framework Mocha <https://mochajs.org/>
   * sudo npm install **mocha** --ignore-scripts --save-dev
   * sudo git add . , sudo git commit –am “message”, sudo npm rebuild
   * Instead of using
     + sudo node src/test.js,
     + use sudo node\_modules/.bin/mocha src/test.js
   * Group all the test cases within **describe**
     + describe(**"Addition"**, **function**(){ *//use "describe" to group all the test cases* it(**"adds positive numbers"**, **function**(){ *//use "it" for test case, and write the comment in the code* assert.equal(*add*(3,4),7);*// Chai* });  
        it(**"IEEE 754 floating point"**, **function**(){  
        assert.equal(*add*(0.1,0.2), 0.30000000000000004); *//Chai* });  
       });
   * sudo node\_modules/.bin/mocha src/test.js
     + - When there is no error



* + - * When there are errors (function add() ,change to a - b)



1. **Mocha** is **TDD** (Test-Driven development) framework. Another popular framework is **Jasmine** (BBD: Behavior-Driven development) <http://jasmine.github.io/>**.** TDD is more focus on programmer, BDD is more focus on communication for none programmer. **It’s all about script differences, but they all doing the same thing.**

**Ht13: Cross-Browser Test Runner -Karma**

1. Install Karma <http://karma-runner.github.io/0.13/index.html>
   * **sudo npm install karma --save-dev --ignore-scripts**
   * sudo git add . , sudo git commit –am “message”, sudo npm rebuild to see if there is any binary
   * git status //there are a lot of untrack binary files, copy them and paste them into .gitignore file:

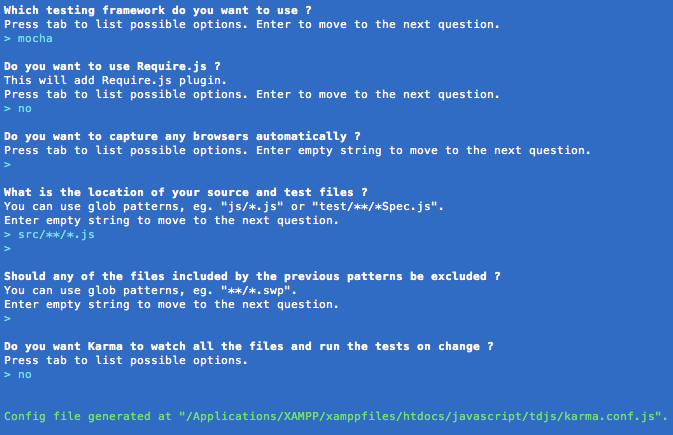
# Karma binaries  
node\_modules/karma/node\_modules/chokidar/node\_modules/fsevents/build/  
node\_modules/karma/node\_modules/socket.io/node\_modules/engine.io/node\_modules/ws/node\_modules/bufferutil/build/  
node\_modules/karma/node\_modules/socket.io/node\_modules/engine.io/node\_modules/ws/node\_modules/utf-8-validate/build/  
node\_modules/karma/node\_modules/socket.io/node\_modules/socket.io-client/node\_modules/engine.io-client/node\_modules/ws/node\_modules/bufferutil/build  
node\_modules/karma/node\_modules/socket.io/node\_modules/socket.io-client/node\_modules/engine.io-client/node\_modules/ws/node\_modules/utf-8-validate/build/

# Karma's binaries don't build on vanilla Windows because they require Python.  
# They're optional, so we've removed them from the repository and ignored them:  
node\_modules/karma/node\_modules/chokidar/node\_modules/fsevents/  
node\_modules/karma/node\_modules/socket.io/node\_modules/engine.io/node\_modules/ws/node\_modules/bufferutil/  
node\_modules/karma/node\_modules/socket.io/node\_modules/engine.io/node\_modules/ws/node\_modules/utf-8-validate/  
node\_modules/karma/node\_modules/socket.io/node\_modules/socket.io-client/node\_odules/engine.io-client/node\_modules/ws/node\_modules/bufferutil/  
node\_modules/karma/node\_modules/socket.io/node\_modules/socket.io-client/node\_odules/engine.io-client/node\_modules/ws/node\_modules/utf-8-validate/

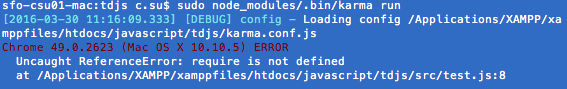
* + Delete following files, so windows system won’t fail. Karma’s binarys don’t build on vanilla windows because they require Python. They are optional, so we’ve removed them from the repository and ignored them:
  + **rm -rf** : Means remove directory and all the sub directory without asking question.
    - sudo rm -rf node\_modules/karma/node\_modules/chokidar/node\_modules/fsevents/
    - rm -rf node\_modules/karma/node\_modules/socket.io/node\_modules/engine.io/node\_modules/ws/node\_modules/bufferutil/
    - sudo rm -rf node\_modules/karma/node\_modules/socket.io/node\_modules/engine.io/node\_modules/ws/node\_modules/utf-8-validate/
    - sudo rm -rf node\_modules/karma/node\_modules/socket.io/node\_modules/socket.io-client/node\_odules/engine.io-client/node\_modules/ws/node\_modules/bufferutil/
    - sudo rm -rf node\_modules/karma/node\_modules/socket.io/node\_modules/socket.io-client/node\_odules/engine.io-client/node\_modules/ws/node\_modules/utf-8-validate/

1. To initialize Karma: sudo node\_modules/.bin/karma init.

* Options showing in following screen shot. (use tab to swap options)
* Karma.conf.js will be created.



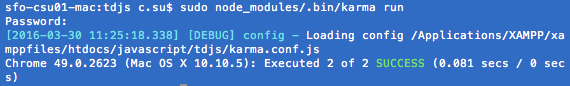
1. To start Karma server: sudo node\_modules/.bin/karma start
   * Copy the local host url: <http://localhost:9876/> to browser, that will capture the browser. When Karma run the test, it will automatically send, run and test your javascript in the browser, and report the result back to command line.
2. Open another Terminal and sudo node\_moduels/.bin/karma run



* This error was found from test.js : **var** assert = require(**"chai"**).**assert**; //this is only good when running under node, so use assertEqual function instead as following:

(**function** () {  
 **"use strict"**;  
  
 *//var assert = require("chai").assert;  
  
 //Mocha--------------------------------------* describe(**"Addition"**, **function**(){ *//use "describe" to group all the test cases* it(**"adds positive numbers"**, **function**(){ *//use "it" for test case, and write the comment in the code  
 //assert.equal(add(3,4), 7); //Chai  
 assertEqual*(*add*(3,4), 7);  
 });  
  
 it(**"uses IEEE 754 floating point"**, **function**(){  
 *//assert.equal(add(0.1, 0.2), 0.30000000000000004);//Chai  
 assertEqual*(*add*(0.1,0.2), 0.30000000000000004);  
 });  
  
 **function** *assertEqual*(actual, expected){  
 **if** (actual !== expected) **throw new** Error(**"expected"**+ expected + **", but was "** + actual);  
 }  
 });  
  
  
 **function** *add*(a,b){  
 **return** a + b;  
 }  
  
}());

* After modification, run sudo node\_moduels/.bin/karma run



* When size down the width of terminal and run above again, there is a little bug and not running correctly, so change **karma.conf.js**, change to reporters: ['dots’].
* Go back to Karma server and control+ C and start again sudo node\_modules/.bin/karma start
* Go back to another terminal and run sudo node\_moduels/.bin/karma run

**Ht14: Automating Karma**

1. We were using Karma or Mocha to run test.js, but eventually we just want to use one commend which is jake.sh to automat everything.

* Run ./jake.sh
* There are err because “describe”, and “it” were global variable by Mocha, so we need to use Karma to run it, otherwise Jake file can not recognize them. So we have to define them in **Jakefile.js** under global to prevent Mocha global from triggering lint errors.

**globals**: {

*//for Mocha* **describe**: **false**, //false means we will never change it   
 **it**: **false**,  
 **before**: **false**,  
 **after**: **false**,  
 **beforeEach**: **false**,  
 **afterEach**: **false**

}

* Run ./jake.sh again, and it works without errors. Check in the code.
* Refactor Jakefile.js code so it looks cleaner. Cut data under lint **options,** and **globals** into a new lintOptions(), and lintGlobal() function:

**options**: *lintOptions*(),

**globals**: *lintGlobal*()

**function** *lintOptions*(){  
 **return** {  
 **bitwise**: **true**, **eqeqeq**: **true**,  
 **forin**: **true**,  
 **freeze**: **true**,  
 **futurehostile**: **true**,  
 **latedef**: **"nofunc"**,  
 **noarg**: **true**,  
 **nocomma**: **true**,  
 **nonbsp**: **true**,  
 **nonew**: **true**,  
 **strict**: **true**, **undef**: **true**,  
  
 **node**: **true**,  
 **browser**: **true** };  
}

**function** *lintGlobal*(){  
 **return**{  
 *//Mocha* **describe**: **false**, *// false means , we will never change it* **it** : **false**,  
 **before**: **false**,  
 **after**: **false**,  
 **beforeEach**: **false**,  
 **afterEach**: **false** };  
}

1. So now get Karma automated in Jakefile

* Rather using node\_modules/.bin/karma start, we should use ./jake.sh karma
* Create a new task “**karma**” to start up the Karma Server, before **default** task, and check in the code.

desc(**"Start the Karma server (run this first)"**);  
task(**"karma"**, **function**(){  
 ***console***.log(**"Starting Karma server:"**);  
});

* sudo npm install simplebuild-karma --ignore-scripts --save-dev , so we don’t have to use jake.exec(“node node\_modules/.bin/karma”) …
* sudo git add . , sudo git commit –am “message”, sudo npm rebuild to see if there is any binary
* Add following into **Jakefile.js,** so we don’t have to runsudo node\_modules/.bin/karma start , to start Karma server:

**var** karma = require(**"simplebuild-karma"**);

desc(**"Start the Karma server (run this first)"**);  
task(**"karma"**, **function**(){  
 ***console***.log(**"Starting Karma Server:"**);  
 karma.start({  
 **configFile**: **"karma.conf.js"** }, complete, fail);  
}, { **async**: **true** });

|  |
| --- |
| Run ./jake.sh karma to start up Karma Server |

* Add task “test” as following, **,** so we don’t have to runsudo node\_modules/.bin/karma run , to run the test in Karma.

desc(**"Run tests for javascript in Karma"**);  
task(**"test"**, **function**(){  
 ***console***.log(**"Testing JavaScript"**);  
 karma.run({  
 **configFile**: **"karma.conf.js"** }, complete, fail);  
}, {**async**: **true**});

* Add task “test” to default task as :

task("default", ["version", "lint", **"test"**], ……

* Another terminal run: sudo ./jake.sh , this will automatically run tasks that we set up in default task, including “test” which to test JavaScript if there is any error.
* Karma.conf.js is written twice in Jakefile.js, so we use variable to contain it to refactor the code a better way.

**var** KARMA\_CONFIG = **"karma.conf.js"**;

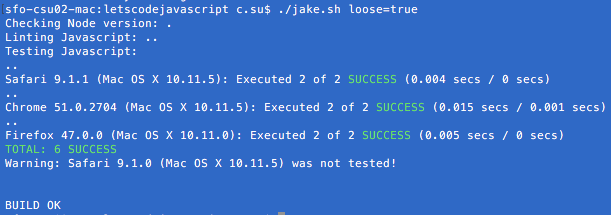
**configFile**: KARMA\_CONFIG

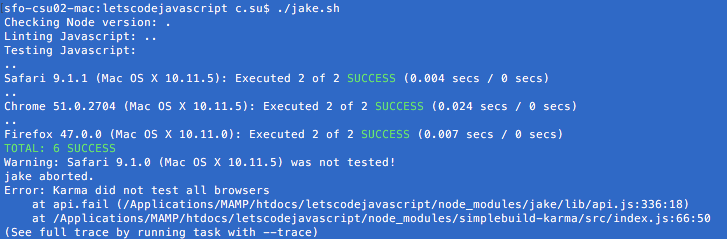
* To make sure JavaScript been tested in different browser with correct version, add following with ”**test”** task:

**expectedBrowsers**: [ *//this will check if JavaScript been test in both browsers* **"Chrome 49.0.2623 (Mac OS X 10.10.5)"**,  
 **"Firefox 45.0.0 (Mac OS X 10.10.0)"**]

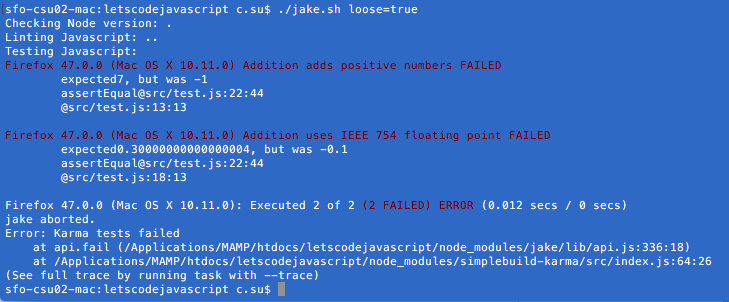
* Open both browser : <http://localhost:9876/>, and run: ./jake.sh again. In console, it will show JavaScript were being tested in both browsers.
* **strict**: **false** *//so when browser version is wrong, jake won't abort*
* **strict**: !**process**.**env**.**loose** , so even browser type is not match Jake won’t fail. If just run sudo ./jake.sh and browser type is not match jake will fail.

|  |
| --- |
| Following using Jake to automate Javascript test tasks, cross testing for all Javascript that are under src/\*\*/\*.js which define under karma.conf.js.  Run ./jake.sh loose=true |





When change function add(a,b) {return a-b}



**Ht15:** **Testing Android, iOS, IE, and More**

1. <http://dev.modern.ie/tools/vms/mac/> to download Virtual machine for IE11 on Win7 , platform: VirtualBox.
2. Download XCode for iPhone simulator
3. <http://developer.android.com/sdk/index.html> Download Android studio.
   * Configure 🡪 SDK Manager 🡪 Install packages.
   * New project 🡪 accept default setting 🡪 click on AVD Manager icon
   * Create Virtual Device 🡪 choose the default device and install necessary plugin🡪 click on play (green arrow) button.
   * In console dialog: /Users/c.su/Library/Android/sdk/tools/emulator -netdelay none -netspeed full -avd Nexus\_5\_API\_N Copy and paste to launch simulator next time.

**Ht16: Modularity: Modular Tests - CommonJS**

1. **Modular** is multiple JavaScript files, each file called modular. So far 3/2016 JavaScript doesn’t support modularity. However, there are 3 different ways to convert multiple files (modular) into single JavaScript file.
   * **Common JS**: ex: the one we use in test.js **require(“semver”); 🡨 Use this one**
   * **AMD (Asynchronous module definition)**
   * forthcoming **AMPscript**
2. Let’s break test.js into 2 files (moudlar), by creating \_*addition\_test.js (test code start with underscore \_) which contain test cases and addition.js which only contain add() function*
   * Under **\_*addition\_*test.js** file, use following to require addition.js. **require** paired up with **exports.**

**var** addition = require(**"./addition.js"**); //.means current directory

* + Under **addition.js** file, insert exports.add = add under add function; or exports.add = function add(a,b){….}. It will **exports** add() function for **require** to use.

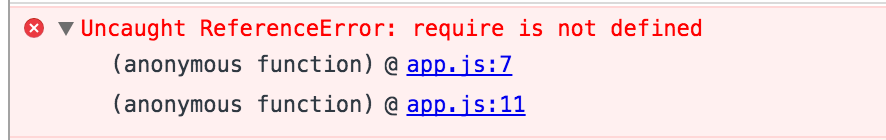
1. When running **./jake.sh** , **require** is not recognized by Karma. So we need to install **karma-commonjs**.
   * sudo npm install karma-commonjs --ignore-scripts --save-dev
   * sudo git add . , sudo git commit –am “message”, sudo npm rebuild
2. In **karma.conf.js**: add following:
   * **frameworks**: [**'mocha'**,**'commonjs'**], //tell Karma to load framework commjs
   * preprocessors: {  
      **'src/\*\*/\*.js'** : [**'commonjs'**] //tell Karma to process all js files under src to get processed by commonjs plugin  
     },
3. restart Karma server : **./jake.sh karma**
4. Another console to run **./jake.sh** again, Karma can recognized **require**, no more error message.

**Ht17: Modular in Production browserify**

1. When running on browser, **require** in JavaScript still not been recognized by browser so we need to use following plugin. Browserify analysis all required files from all JavaScript files (modules) into one single bundled JavaScript file.
   1. browserify: <http://browserify.org> 🡨 Use this
   2. webpack: <http://webpack.github.io>
2. sudo npm install browserify --ignore-scripts --save-dev
3. sudo git add . , sudo git commit –am “message”, sudo npm rebuild
4. sudo node\_modules/.bin/browserify to check out browserify ***usage information***.
5. Create **app.js** under src/ , and type following. Under index.html, add <**script** src=**"app.js"**></**script**>

(**function** () {  
 **"use strict"**;  
  
 **var** addition = require(**"./addition.js"**);  
  
 ***console***.log(**"Hello!"**);  
 ***console***.log(**"42 + 13 ="** + addition.add(42,13));  
}());

1. Sudo ./jake.sh run, and http://localhost: 8080. Browser console will show require is not defined.



1. node\_modules/.bin/browserify src/app.js -o src/bundle.js. browserify will analysis app.js all the required files and bundle them into src/bundle.js.
   1. **--outfile, -o**: Write the browserify bundle to this file. If unspecified, browserify prints to stdout.
2. Change <script src = “bundle.js”></script> in index.html
3. Sudo ./jake.sh run, now <http://localhost:8080> will console out the info without err message.

**Ht18: The Build Step: browserify**

1. Because bunle.js file is generated for machine to read, so when >./jake.sh , there will be lots error cause by lint. Therefore, we need to copy index.html and move bundle.js under new folder /generated/dist. To separated source and generated code, and tell jake file not to lint generated code.
2. node\_modules/.bin/browserify src/app.js -o generated/dist/bundle.js to generated bundle.js again since the path had changed.
3. node\_modules/.bin/http-server generated/dist then run <http://localhost:8080>
4. We don’t check in generated files, so under .gitignore file, type in following:

#generated files  
generated/

1. Instead of running broswerify and server manually, add a task “build” in Jakefile.js

desc(**"Build distribution directory"**);  
task(**"build"**, **function**(){  
 ***console***.log(**"Building distribution directory:"**);  
});

1. In Jakefile.js change task “run” update to:

desc(**"Run a localhost server"**);  
task(**"run"**, [**"build"**], **function**(){ //run task “build” before run the server, so it will generate generated/dist folder first before running the server  
 ***jake***.exec(**"node node\_modules/http-server/bin/http-server generated/dist"**, {**interactive**: **true**}, complete);  
 *//console.log("Run http-server here");*});

Sudo ./jake.sh run, now <http://localhost:8080> will console out the info without err message.

1. Delete folder “generated”
2. Under “build” task, modify as following:

desc(**"Build distribution directory"**);  
task(**"build"**, [**"generated/dist"**], **function**(){ *// ["generated/dist"] as a task which will run before “build” task , see following directory("generated/dist")* ***console***.log(**"Building distribution directory:"**);  
});  
  
directory(**"generated/dist"**);*// this will create generated/dist folder*

1. To prevent repeat typing “generated/dist” , we create constant variable on the top as following and replace wherever contain this path with this variable.

**var** DIST\_DIR = **"generated/dist"**;

1. In Jakefile.js: Under //General-purpose tasks -------- section; create a new task to erase all the generated files as following:

desc(**"Erase all generated files"**);  
task(**"clean"**, **function**(){  
 ***console***.log(**"Erasing generated files"**);  
});

1. In order to run simple shell command line in Jakefile.js, we need to install following plugin:
   * sudo npm install shelljs --ignore-scripts --save-dev
   * sudo git add . , sudo git commit –am “message”, sudo npm rebuild
2. At the top of Jakefile.js, add

**var** shell = require(**"shelljs"**);

1. Under “clean” task, add following:

shell.rm(**"-rf"**, **"generated"**); *//force to remove all the file under generated without any question*

1. ./jake.sh clean : this will delete all the files under /generated folder. Check in files.
2. Under “build” task add following, and check in files:

shell.rm(**"-rf"**, DIST\_DIR + **"/\*"**); *//delete all the file under generated/dist, so it won't complain index.html already exist*

shell.cp(**"src/index.html"**, DIST\_DIR); *// copy index.html file into generated/dist*

*//run browserify to bundle the javascript****jake***.exec(  
 **"node node\_modules/browserify/bin/cmd.js src/app.js -o "** + DIST\_DIR + **"/bundle.js"** ,  
 {**interactive**: **true**, **async**: **true**},  
 *complete*);

1. Now we run the server
   * ./jake.sh run , and
   * <http://localhost:8080>
   * ./jake.sh clean , this will clean out all the file under /generated
   * ./jake.sh run again , this will work again
2. Refactor directory as following:
   * Create /content under src , and move index.html under it.
   * Under **build** task, Change to shell.cp(**"src/content/\*"**, DIST\_DIR); //copy everything under content
   * Create /javascript under src, and move all js file under it.
   * Under **build** task modify as following:

***jake***.exec(  
 **"node node\_modules/browserify/bin/cmd.js src/javascript/app.js -o "** + DIST\_DIR + **"/bundle.js"** ,  
 {**interactive**: **true**, **async**: **true**},  
 *complete*);

* + ./jake.sh clean then ./jake.sh run
  + Under **lint** task, change following:

**files**: [**"Jakefile.js"**, **"src/javascript/\*\*/\*.js"**],

* + On very top of Jakefile.js add directory:false as global for Jake to recognize directory variable.

1. Under all the task,

complete, *fail*); *//after finish 'karma' task will run complete or fail function (async function)*}, {**async**: **true**}); *//async: true, after finish calling complete or fail, then*

*it will move on to next thing*

1. Add , {**async**: **true**} to ‘build’, and ‘run’ task.

task(**"run"**, [**"build"**], **function**(){*//run task "build" to generate generated/dist folder before running the server* ***jake***.exec(**"node node\_modules/http-server/bin/http-server "** + DIST\_DIR , {**interactive**: **true**, **async**: **true**}, *complete*);  
 *//interactive:true , so we can see the output.  
 //complete: to run complete function when it's done.*}, {**async**: **true**});

|  |
| --- |
| * + If we don’t add async: true, it will cost ***race condition*** which means before finish running **build** task, **run** task will start executing. |

**Ht19: Frontend Modules**

1. In previous JavaScript file we disable using Chai because Karma can’t recognize Chai.
   * Download <http://chaijs.com/chai.js>
   * New folder “vendor” under /src.
   * Rename chai file with version number as chai-3.5.0.js, and move to /src/vendor folder.
   * In *addition*test.js file, on the top insert following:

**var** assert = require(**"../vendor/chai-3.5.0.js"**).assert;

*//.assert is to include Chai.assert library*

* + In Karma.conf.js, change following:

**files**: [  
 **'src/javascript/\*\*/\*.js'**, *//tells Karma where to load js files* **'src/vendor/chai-3.5.0.js'** *//tells Karma to load plugin Chai*],

**preprocessors**: {  
 **'src/javascript/\*\*/\*.js'**: [**'commonjs'**], *//tell Karma to process all the js files under /src/javascript to get processed by commonjs plugin* **'src/vendor/chai-3.5.0.js'**: [**'commonjs'**] *//tell Karma to process chai plugin to get process by commonjs, so require("chai...") will works*},

* + Restart Karma , ./jake.sh karma , refresh <http://localhost:9876/> browser , ./jake.sh loose=true
  + Fix up code in *addition*test.js as following

describe(**"Addition"**, **function**(){ *//use "describe" to group all the test cases* it(**"adds positive numbers"**, **function**(){ *//use "it" for test case, and write the comment in the code* assert.equal(addition.add(3,4), 7); *//use this Chai  
 ~~//assertEqual(addition.add(3,4), 7);~~*  *~~🡪 take this out~~* });  
  
 it(**"uses IEEE 754 floating point"**, **function**(){  
 assert.equal(addition.add(0.1, 0.2), 0.30000000000000004);*//use this Chai  
 ~~//assertEqual(addition.add(0.1,0.2), 0.30000000000000004); 🡪 take this out~~* });  
  
 *~~// function assertEqual(actual, expected){  
 // if (actual !== expected) throw new Error("expected"+ expected + ", but was " + actual);  
 // }~~*});

* + > ./jake.sh loose=true, Chai works fine
  + > sudo npm uninstall chai --save-dev, sudo git add. , sudo git commit -am “npm version of chai is not longer needed”

1. Because we require Chai file with version number, but when Chai version changed, and we don’t want to change all the files that include Chai. So we create a assert.js file under /javascript as following. When Chai version changed, we just update assert.js file.

(**function** () {  
 **"use strict"**;  
 **var** assert = require(**"../vendor/chai-3.5.0.js"**).assert; *// .assert is to include Chai.assert library* module.exports = assert;  
  
}());

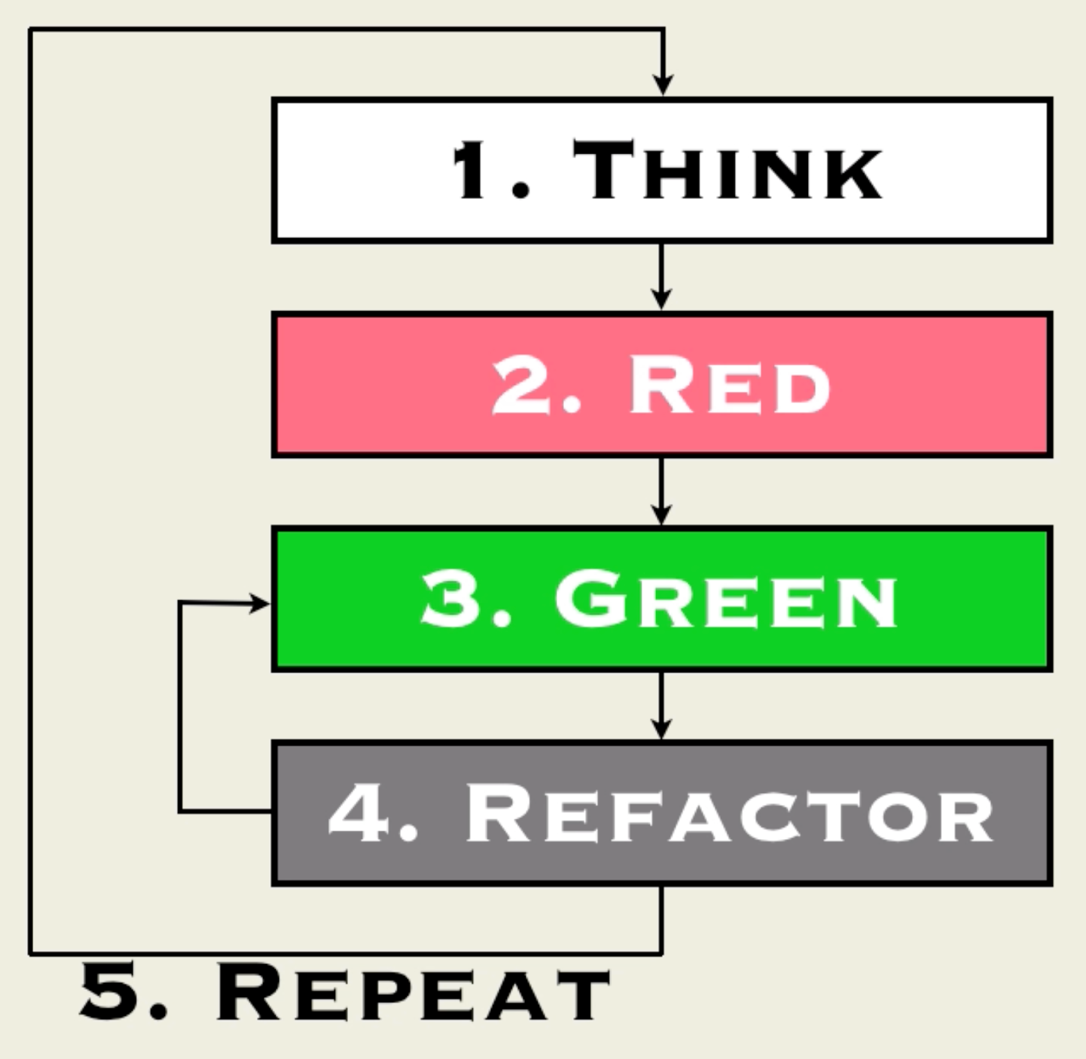
* + In \_*addition\_*test.js file

**var** assert = require(**"./assert.js"**);

**~~var~~** ~~assert = require(~~**~~"../vendor/chai-3.5.0.js"~~**~~).assert;~~

**Ht20: Test driven development (TDD)**

1. Important thing about TDD is **to use tests to remain control of the code, and use test to drive development. It contains 5 steps.** 
   * **1. Think:** of possible tests will fail your task/code.
   * **2. Red:** Write the test and it will fail, because there is no code to test yet.
   * **3. Green:** Write the code to make the test pass.
   * **4. Refactor:** the code that you written so far.
   * **5. Repeat:** the 3 and 4 steps , and repeat 1-4 steps.



1. Following is the baby steps to TDD the code process:
   * + 1. **Think**: Write the test code in \_*addition\_*test.js:

describe(**"Subtraction"**, **function**(){

it(**"subtracts positive numbers"**, **function**(){

addition.subtract(10,3);

});

});

>./jake.sh karma,

>./jake.sh loose=true,

* + - 1. **Red**: When running server and test from step 1, there will be error message says no subtract function.
      2. **Green**: Start writing function in addition.js

***exports***.subtract = **function** subtract(a, b){

**return** a - b;

};

>./jake.sh loose=true, this will pass.

* + - 1. **Refactor**: edit *addition*test.js:

describe(**"Subtraction"**, **function**(){  
 it(**"subtracts positive numbers"**, **function**(){  
 assert.equal(addition.subtract(10,3), 7);  
 });  
});

* + - 1. **Repeat**: Think through the whole process again and make a better code.

**Ht21:**