**Grunt**

Grunt is javascript task runner.

Grunt and Grunt plugins are installed and managed via [npm](https://npmjs.org/), the [Node.js](http://nodejs.org/) package manager.

Before setting up Grunt ensure that your [npm](https://npmjs.org/) is up-to-date by running npm update -g npm.

Grunt 0.4.x requires stable Node.js versions >= 0.8.0.

**[Installing the CLI](http://gruntjs.com/getting-started" \l "installing-the-cli)**

npm install -g grunt-cli

Note that installing grunt-cli does not install the Grunt task runner! The job of the Grunt CLI is simple: run the version of Grunt which has been installed next to a Gruntfile. This allows multiple versions of Grunt to be installed on the same machine simultaneously.

## [Preparing a new Grunt project](http://gruntjs.com/getting-started" \l "preparing-a-new-grunt-project)

A typical setup will involve adding two files to your project: package.json and the Gruntfile.

**package.json**: This file is used by [npm](https://npmjs.org/) to store metadata for projects published as npm modules. You will list grunt and the Grunt plugins your project needs as [devDependencies](https://docs.npmjs.com/files/package.json" \l "devdependencies) in this file.

**Gruntfile**: Itjs is used to configure or define tasks and load Grunt plugins. 

## [package.json](http://gruntjs.com/getting-started" \l "package.json)

## The package.json file belongs in the root directory of your project, next to the Gruntfile, and should be committed with your project source.

There are a few ways to create a package.json file for your project:

* Most [grunt-init](http://gruntjs.com/project-scaffolding) templates will automatically create a project-specific package.json file.
* The [npm init](https://docs.npmjs.com/cli/init) command will create a basic package.json file.

{

{

"name": "my-project-name",

"version": "0.1.0",

"devDependencies":

"grunt": "~0.4.5",

grunt-contrib-jshint": "~0.10.0",

"grunt-contrib-nodeunit": "~0.4.1"

"grunt-contrib-uglify": "~0.5.0"

}

}

"name": "my-project-name",

"version": "0.1.0",

"devDependencies": {

"grunt": "~0.4.5",

"grunt-contrib-jshint": "~0.10.0",

"grunt-contrib-nodeunit": "~0.4.1",

"grunt-contrib-uglify": "~0.5.0"

}

}

### [Installing Grunt and gruntplugins](http://gruntjs.com/getting-started" \l "installing-grunt-and-gruntplugins)

this will install the latest version of Grunt in your project folder, adding it to your devDependencies:

npm install grunt --save-dev

The same can be done for gruntplugins and other node modules. As seen in the following example installing the JSHint task module:

npm install grunt-contrib-jshint --save-dev

**How to write grunt file:**

The first part is the "wrapper" function, which encapsulates your Grunt configuration.

module.exports = function(grunt) {

}

Within that function we can initialize our configuration object:

grunt.initConfig({

});

Next we can read in the project settings from the package.json file into the pkg property

pkg: grunt.file.readJSON('package.json')

Finally, we have to load in the Grunt plugins we need. These should have all been installed through npm.

grunt.loadNpmTasks('grunt-contrib-uglify');

And finally set up some tasks. Most important is the default task:

// this would be run by typing "grunt test" on the command line

grunt.registerTask('test', ['jshint', 'qunit']);

// the default task can be run just by typing "grunt" on the command line

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

**Available Grunt plugins**

* Many of the tasks you need are already available as Grunt Plugins, and new plugins are published every day.
* Saas
* Jshint
* Coffee script
* Handlebars
* Require.js
* Styus
* Jade
* Less

Who use grunt:

These are just a few companies and projects that are using Grunt

* Sauce labs
* Adobe
* Skype
* Bocoup
* Bitovi
* Jquery
* Fliament group
* modernizer

tasks:

**Concat**

concat: {

options: {

// define a string to put between each file in the concatenated output

separator: ';'

},

dist: {

// the files to concatenate

src: ['src/\*\*/\*.js'],

// the location of the resulting JS file

dest: 'dist/<%= pkg.name %>.js'

}

}

Note how I refer to the name property that's in the JSON file. We access this using pkg.name as earlier we defined the pkg property to be the result of loading the package.json file, which is then parsed to a JavaScript object. Grunt has simple template engine to output the values of properties in the configuration object. Here I tell the concat task to concatenate all files that exist within src/ and end in .js.

**uglify plugin**

which minifies our JavaScript:

uglify: {

options: {

// the banner is inserted at the top of the output

banner: '/\*! <%= pkg.name %> <%= grunt.template.today("dd-mm-yyyy") %> \*/\n'

},

dist: {

files: {

'dist/<%= pkg.name %>.min.js': ['<%= concat.dist.dest %>']

}

}

}

This tells uglify to create a file within dist/ that contains the result of minifying the JavaScript files. Here I use <%= concat.dist.dest %> so uglify will minify the file that the concat task produces.

[Task Configuration and Targets](http://gruntjs.com/configuring-tasks" \l "task-configuration-and-targets)

When a task is run, Grunt looks for its configuration under a property of the same name. Multi-tasks can have multiple configurations, defined using arbitrarily named "targets."

grunt.initConfig({

concat: {

foo: {

// concat task "foo" target options and files go here.

},

bar: {

// concat task "bar" target options and files go here.

},

},

uglify: {

bar: {

// uglify task "bar" target options and files go here.

},

},

});

[Options](http://gruntjs.com/configuring-tasks" \l "options)

an options property may be specified to override built-in defaults

grunt.initConfig({

concat: {

options: {

// Task-level options may go here, overriding task defaults.

},

foo: {

options: {

// "foo" target options may go here, overriding task-level options.

},

},

bar: {

// No options specified; this target will use task-level options.

},

},

});

[Files](http://gruntjs.com/configuring-tasks" \l "files)

Grunt has powerful abstractions for declaring on which files the task should operate. There are several ways to define **src-dest** (source-destination) file mappings,

All files formats support src and dest but the "Compact" and "Files Array" formats support a few additional properties:

* filter Either a valid [fs.Stats method name](http://nodejs.org/docs/latest/api/fs.html#fs_class_fs_stats) or a function that is passed the matched src filepath and returns true or false.
* expand Process a dynamic src-dest file mapping, see ["Building the files object dynamically"](http://gruntjs.com/configuring-tasks#building-the-files-object-dynamically) for more information.

**[Compact Format](http://gruntjs.com/configuring-tasks" \l "compact-format)**

This form allows a single **src-dest** (source-destination) file mapping per-target. It is most commonly used for read-only tasks, like [grunt-contrib-jshint](https://github.com/gruntjs/grunt-contrib-jshint), where a single src property is needed, and no dest key is relevant

grunt.initConfig({

jshint: {

foo: {

src: ['src/aa.js', 'src/aaa.js']

},

},

concat: {

bar: {

src: ['src/bb.js', 'src/bbb.js'],

dest: 'dest/b.js',

},

},

});

### [Files Object Format](http://gruntjs.com/configuring-tasks" \l "files-object-format)

This form supports multiple src-dest mappings per-target, where the property name is the destination file, and its value is the source file(s). Any number of src-dest file mappings may be specified in this way, but additional properties may not be specified per mapping.

grunt.initConfig({

concat: {

foo: {

files: {

'dest/a.js': ['src/aa.js', 'src/aaa.js'],

'dest/a1.js': ['src/aa1.js', 'src/aaa1.js'],

},

},

bar: {

files: {

'dest/b.js': ['src/bb.js', 'src/bbb.js'],

'dest/b1.js': ['src/bb1.js', 'src/bbb1.js'],

},

},

},

});

### [Files Array Format](http://gruntjs.com/configuring-tasks" \l "files-array-format)

This form supports multiple src-dest file mappings per-target, while also allowing additional properties per mapping.

grunt.initConfig({

concat: {

foo: {

files: [

{src: ['src/aa.js', 'src/aaa.js'], dest: 'dest/a.js'},

{src: ['src/aa1.js', 'src/aaa1.js'], dest: 'dest/a1.js'},

],

},

bar: {

files: [

{src: ['src/bb.js', 'src/bbb.js'], dest: 'dest/b/', nonull: true},

{src: ['src/bb1.js', 'src/bbb1.js'], dest: 'dest/b1/', filter: 'isFile'},

],

},

},

});

### [Custom Filter Function](http://gruntjs.com/configuring-tasks" \l "custom-filter-function)

The filter property can help you target files with a greater level of detail. Simply use a valid [fs.Stats method name](http://nodejs.org/docs/latest/api/fs.html#fs_class_fs_stats). The following will clean only if the pattern matches an actual file:

grunt.initConfig({

clean: {

foo: {

src: ['tmp/\*\*/\*'],

filter: 'isFile',

},

},

});

Or create your own filter function and return true or false whether the file should be matched. For example the following will only clean folders that are empty:

grunt.initConfig({

clean: {

foo: {

src: ['tmp/\*\*/\*'],

filter: function(filepath) {

return (grunt.file.isDir(filepath) && require('fs').readdirSync(filepath).length === 0);

},

},

},

});

[Globbing patterns](http://gruntjs.com/configuring-tasks" \l "globbing-patterns)

Grunt supports filename expansion (also know as globbing) via the built-in [node-glob](https://github.com/isaacs/node-glob) and [minimatch](https://github.com/isaacs/minimatch) libraries.

matches any number of characters, but not /

* ? matches a single character, but not /
* \*\* matches any number of characters, including /, as long as it's the only thing in a path part
* {} allows for a comma-separated list of "or" expressions
* ! at the beginning of a pattern will negate the match

**[Why doesn't my asynchronous task complete?](http://gruntjs.com/frequently-asked-questions" \l "why-doesn-t-my-asynchronous-task-complete)**

Chances are this is happening because you have forgotten to call the [this.async](http://gruntjs.com/grunt.task#wiki-this-async) method to tell Grunt that your task is asynchronous. For simplicity's sake, Grunt uses a synchronous coding style, which can be switched to asynchronous by calling this.async() within the task body.

Note that passing false to the done() function tells Grunt that the task has failed.

For example:

grunt.registerTask('asyncme', 'My asynchronous task.', function() {

var done = this.async();

doSomethingAsync(done);

});

Plugin or Task

Concat:

concat: {

options: {

// define a string to put between each file in the concatenated output

separator: ';'

},

dist: {

// the files to concatenate

src: ['src/\*\*/\*.js'],

// the location of the resulting JS file

dest: 'dist/<%= pkg.name %>.js'

}

}

Here I tell the concat task to concatenate all files that exist within src/ and end in .js.

**Uglify:**

the uglify plugin, which minifies our JavaScript:

uglify: {

options: {

// the banner is inserted at the top of the output

banner: '/\*! <%= pkg.name %> <%= grunt.template.today("dd-mm-yyyy") %> \*/\n'

},

dist: {

files: {

'dist/<%= pkg.name %>.min.js': ['<%= concat.dist.dest %>']

}

}

}

**JSHint :**

Validate files with JSHint

jshint: {

// define the files to lint

files: ['gruntfile.js', 'src/\*\*/\*.js', 'test/\*\*/\*.js'],

// configure JSHint (documented at http://www.jshint.com/docs/)

options: {

// more options here if you want to override JSHint defaults

globals: {

jQuery: true,

console: true,

module: true

}

}

}

JSHint simply takes an array of files and then an object of options

**watch plugin:**

Run tasks whenever watched files change

watch: {

files: ['<%= jshint.files %>'],

tasks: ['jshint', 'qunit']

}