Extracted from:

Automate with Grunt

The Build Tool for JavaScript

This PDF file contains pages extracted from *Automate with Grunt*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2014 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

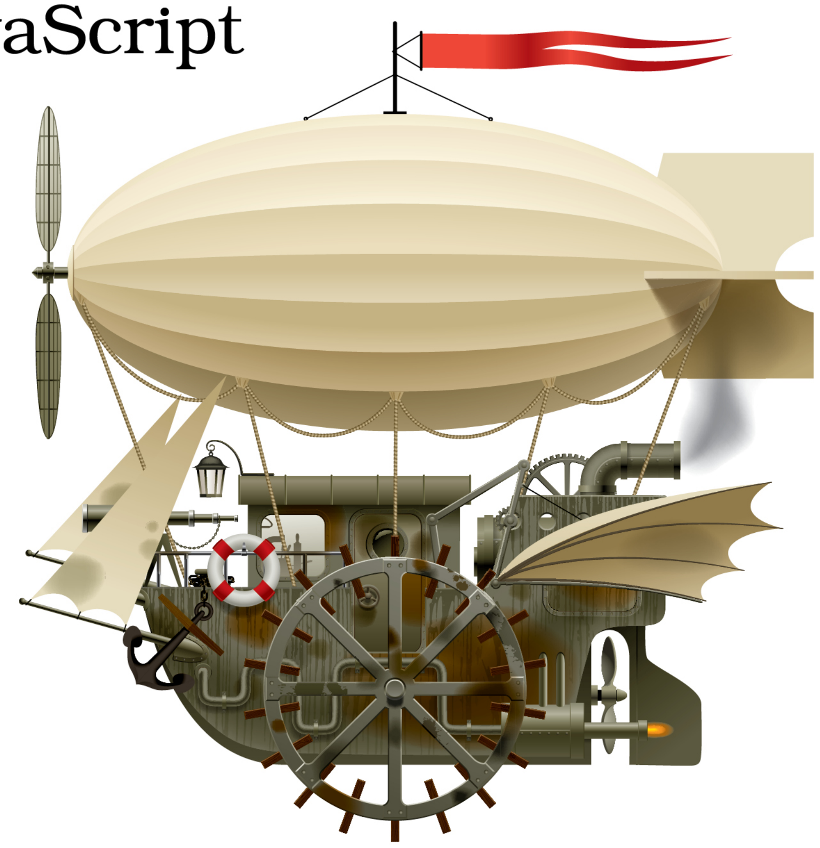
Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Automate with Grunt

The Build Tool for JavaScript



Brian P. Hogan

Edited by Susannah Davidson Pfalzer

Automate with Grunt

The Build Tool for JavaScript

Brian P. Hogan

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

For international rights, please contact *rights@pragprog.com*.

Copyright © 2014 The Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 9781941222119
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—April 2014

CHAPTER 1

The Very Basics

Grunt is a task runner, designed to be more configuration than code. And while many of the examples you'll see in the wild involve copying and pasting configuration snippets and loading plug-ins, don't be fooled into thinking that's all Grunt can do. Sure, there are some amazing plug-ins that will vastly improve your workflow, but if you know JavaScript, Grunt becomes a very powerful automation tool for many types of projects. If you have a manual repetitive task you run as part of your development or deployment process, chances are there's a way to automate that process with Grunt.

In this chapter we'll set up our first project with Grunt and cover how the basic task system works. We'll use simple JavaScript programs to highlight and explore Grunt's basic features. By the end you'll be able to create basic tasks and handle errors.

Installing Grunt and Configuring a Gruntfile

Before we can do anything with Grunt, we have to install the Grunt command-line tool. Grunt is written in Node.js, and to install it we use npm, the tool Node uses to download and manage dependencies. Open a new Terminal and type the following command:

```
$ npm install -g grunt-cli
```

This installs the Grunt command-line utility globally. On Linux or OS X, you may need to run this command using sudo if you get a "Permission Denied" error.

Grunt is broken into separate packages, each serving a specific purpose. The grunt-cli package gives us a command-line interface. But to use this interface, we have to install the grunt package as well; installing grunt-cli doesn't automatically install grunt for us.

Instead, we install Grunt into our project as a dependency. The grunt-cli tool we installed globally on our system will then work with the version of grunt within our project. Let's create a simple project folder and set everything up.

Create a new folder called kicking_tires and navigate into that folder in your Terminal:

```
$ mkdir kicking_tires
$ cd kicking_tires
```

From here, you could install Grunt with npm install grunt, but there's a better way. Node.js applications use a file called package.json to store the metadata about a project as well as track a project's dependencies. If we create this file, we can add Grunt as a dependency to our project, making it easier to set things up in the future.

Type the following command to create a new package.json file.

```
$ npm init
```

You'll be asked a series of questions about your project. For this project you can leave everything at the default settings. Your package.json file will end up looking like this when the wizard finishes:

```
{
  "name": "kicking_tires",
  "version": "0.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &amp;&amp; exit 1"
  },
  "author": "",
  "license": "BSD-2-Clause"
}
```

Now that we have this file in place, we can add Grunt as a development dependency like this:

```
$ npm install grunt --save-dev
```

Grunt will be installed into the node_modules/ subfolder of the current folder, and it'll be added to the package.json file as a development dependency. If you look at your package.json file you'll see this at the bottom now:

```
"devDependencies": {
  "grunt": "~0.4.4"
}
```

The devDependencies section lists dependencies that are used only to build an application. Grunt isn't something an application needs to run; we use Grunt only as a tool for developing an application. However, a library that lets us connect to a MySQL database would be a true dependency, not a development dependency.

The --save-dev command also saves the version number into the package.json file, and it uses the tilde in front of the version number to signify that any version 0.4.4 or higher is OK for us to use. Version 0.4.7, for example, would be valid, but 0.5.0 would not. This helps us stay current within minor version numbers, but prevents us from accidentally installing a version that's too new and incompatible. Of course, we can change this version number by hand if we like.

The node_modules folder contains all of the libraries our project depends on. This means you have a copy of the Grunt library itself in the node_modules folder.

Adding things as development dependencies allows new people who want to work on our project to easily download all of the dependencies we specify by issuing the npm install command in the folder that contains the package.json file. In fact, let's try this now. Remove the node_modules folder and then run npm install. You'll see that npm fetches Grunt again, creating a new node_modules folder.

With Grunt installed, we can test things out by running it from the command line:

```
$ grunt
```

This fires off the grunt-cli library we installed globally, which then uses the grunt library we installed in our project's node_modules folder. This lets us easily use different versions of Grunt on different projects.

But when we look at our output, we see this message:

```
A valid Gruntfile could not be found. Please see the getting started guide for
more information on how to configure grunt: http://gruntjs.com/getting-started
Fatal error: Unable to find Gruntfile.
```

Grunt is telling us that we need something called a Gruntfile in our project. A Gruntfile is a JavaScript file that specifies and configures the tasks you want to be able to run for your project. It's like a Makefile. Grunt is specifically looking for a file called Gruntfile.js in the current working directory and it can't find one, so it doesn't know what we want it to do. Let's create a Gruntfile.

Our First Task

Let's kick the tires. We'll create the default task, which is the one that runs when we type the grunt command.

Every Gruntfile starts out with some boilerplate code. Create a new file called Gruntfile.js and add this:

basics/kicking_tires/Gruntfile.js

```
module.exports = function(grunt){
  // Your tasks go here
}
```

If you're familiar with Node.js and its module system, you'll understand what's going on here. If you're not, it's not a huge deal; just know that this is what Grunt needs to interpret your tasks. You're defining a Node.js module that receives a grunt object. You'll use that object and its methods throughout your configuration files. The tasks you define and configure are then made available to Grunt so that they can be executed.

Now, within the curly braces, define the following task, which prints some text to the screen:

basics/kicking_tires/Gruntfile.js

```
grunt.registerTask('default', function(){
  console.log('Hello from Grunt.');
});
```

We use grunt.registerTask() to create a new Grunt task. We pass in a task name followed by an associated callback function. Whatever we put in the callback function is executed when we invoke the task.

To see it in action, run this new task from the Terminal:

```
$ grunt
```

You'll see the following output:

```
Running "default" task
Hello from Grunt.

Done, without errors.
```

In this task we've used Node's console.log function, but we really should use Grunt's grunt.log() object instead. It'll give us some flexibility because it supports error logging, warnings, and other handy features.

So, change the following:

basics/kicking_tires/Gruntfile.js
```
console.log('Hello from Grunt.');
```

to

basics/kicking_tires/Gruntfile.js
```
grunt.log.writeln('Hello from Grunt.');
```

and rerun the task with

```
$ grunt
```

You shouldn't see anything different. This task is not fancy by any means, but it illustrates that Grunt works, and that we can create a simple task. Let's move on.

Handling Parameters

Grunt task definitions can take in simple arguments. Let's demonstrate how this works by creating a simple "greeting" task. As before, we'll use grunt.registerTask() to create a task, but this time we'll define a parameter in the callback function.

basics/kicking_tires/Gruntfile.js
```
grunt.registerTask('greet', function(name){
  grunt.log.writeln('Hi there, ' + name);
});
```

In the body of the callback, we reference the variable just like we would in any plain JavaScript code.

Run this task with

```
$ grunt greet
```

and you'll see this output:

```
Running "greet" task
Hi there, undefined

Done, without errors.
```

We didn't actually pass an argument to the task, and so the `name` parameter's value is undefined. Grunt didn't throw an error message at us.

To supply the parameter, we use a colon followed by the value, like this:

```
$ grunt greet:Brian
```

And now we see what we're looking for.

```
Running "greet:Brian" (greet) task
Hi there, Brian

Done, without errors.
```

We don't have to stop at one argument, though. We can define tasks with multiple arguments. Let's create another rather silly task that adds some numbers together.

In the Gruntfile, add this task:

basics/kicking_tires/Gruntfile.js

```
grunt.registerTask('addNumbers', function(first, second){
  var answer = Number(first) + Number(second);
  grunt.log.writeln(first + ' + ' + second + ' is ' + answer);
});
```

To run this task, we have to supply both numbers as arguments to the task, and we do that using colons, like this:

```
$ grunt addNumbers:1:2
```

And when we do that, we see this result:

```
Running "addNumbers:1:2" (addNumbers) task
1 + 2 is 3

Done, without errors.
```

Pretty easy so far, isn't it? But what if the user didn't enter appropriate values? We need a way to handle that gracefully.