AUTOMATING WITH NODE.JS

By Shaun Michael Stone

© First Edition UK - 2018

Table of Contents

Introduction

"I will always choose a lazy person to do a difficult job because a lazy person will find an easy way to do it."

– *Anonymous*

Preface

Being in the technical field presents itself with some fun… and some not so fun tasks, we can all agree upon that. For the not so fun bits I try to filter out all of the repetitive administrative work with something that I'm better known for as a developer; writing code. Yes, I can be a bit lazy, but that's because I value my time. You'd be surprised at how much time goes into updating comments on Jira, zipping files and emailing them to colleagues, updating configuration files, and copying and pasting directories. Arghh, I need to stop listing these before I fall asleep.

At a previous job I found myself doing things that could've easily been automated by a script. Anything that feels repetitive should ring alarm bells for you. If it doesn't, then you should change your mindset now. Look at what you do on a daily basis, think about it, read this book, and see if it changes your perspective. Ideally it should.

On one weekend, after noticing repetitive tasks at work, I took note of the steps involved in these tasks from start to finish, and set out to build a suite of commands that would automate them. It proved to be efficient for both me and some of my teammates, and it gave me more time to concentrate on Reddit… I mean, my work.

I remember reading a funny story about a programmer who automated anything that took longer than ninety seconds. His coffee machine was connected to the local network, and he sent commands to it and timed how long it took for him to walk over to pick up his freshly brewed cup. He even programmed it to send a text message via Twilio to his wife if his machine was logged in fifteen minutes after the end of the working day, saying he was working late that night.

Being fairly accustomed to using Bash scripting in the past on a Linux virtual machine, I decided initially it was the right tool for what I wanted to achieve. I'd need access to the file system. I could make use of the powerful searching commands, store variables, read in standard input from the user, and use conditional statements to decide on how to proceed. Perfect! But then I thought, I wonder if I can achieve the same with Node JS?

I created the bash version initially, but digging further, I learned I could create the project with npm directly. So I rewrote the project and presented it to the team. The great news was that me and my team were allocated time to work on the project during work hours, and one of the technical architects was keen to integrate this tool into our workflow. Winning!

There are two ways we can implement the code you will be learning in this book. You can treat it as a global list of commands that behave in the same way as an alias on a terminal, or you can create a build tool that deploys your project, taking care of all the tedious tasks you are used to doing.

This book will help you build something along the lines of what I have, but it's obvious to point out that every company's workflow follows a different path and set of business rules. Don't worry though, section two of this book explains a good way of identifying and detailing your workflow. Once you have identified this path and the associated workflow, it should be pretty straightforward to apply the knowledge acquired from this book.

End Goal

Let's not beat around the bush. Once you've finished reading this book, you should be able to create global commands and a working bespoke Node build tool that allows you to automate the repetitive tasks you hate doing. This build tool will be shaped around your company's goals, or your own. Either way, the intention is to make your life easier. Because life is hard enough as it is, right?

Structure

The book is structured into two parts:

Part 1

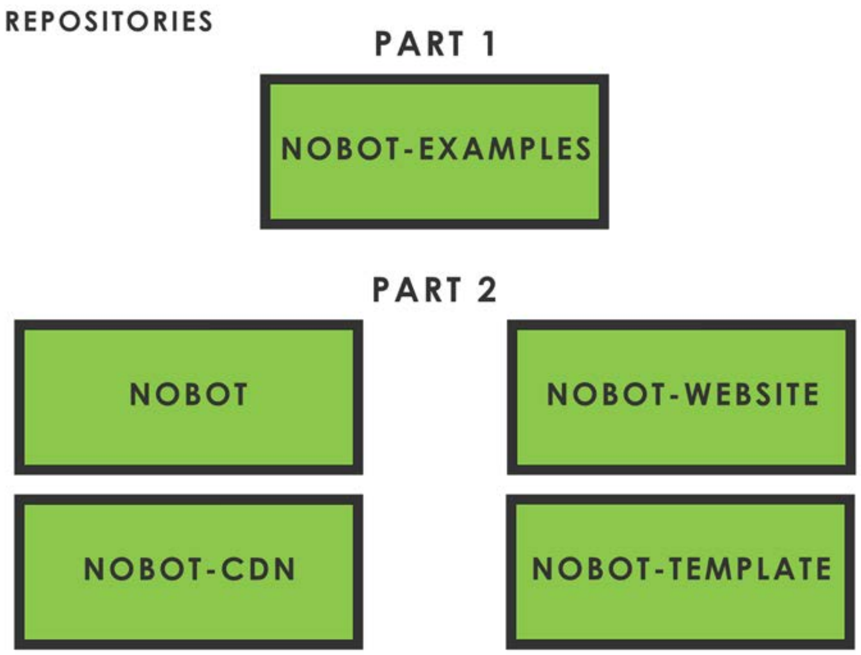
The first part is a collection of recipes, or building blocks that behave as individual global commands. These can be used as you go about your day, and can be called at any time to speed up your workflow or just for pure convenience.

It begins with simple examples so you can get to know more about Node's standard library, then moves into more practical implementations. Each recipe corresponds with the **'examples'** directory found in the repository. All of the examples can be found here: https://github.com/smks/nobot-examples

Part 2

The second part is a walkthrough of creating a cross-platform build tool from the ground up. Each script that achieves a certain task will be its own command, with a main umbrella command – usually the name of your project – encapsulating them all.

Instead of using Gulp or Grunt, we will be using npm directly. The plan is to keep this tool as lightweight as possible. I will be calling the project **Nobot**, because I love naming projects, it’s an obsession. The implementation can be found here: https://github.com/smks/nobot



Above shows a high level overview of the repositories we will make use of in part 1 and part 2 of this book.

Book Coding Style

This book uses examples when working on a Mac and sometimes Windows. You may occasionally see different output.

Some of the code examples may wrap onto the next line due to spacing limitations.

The coding style follows AirBnb coding standards with ESLint. A few rules have been overridden.

Code snippets

The book will have a lot of code snippets, as you'd expect.

Below is how I would demonstrate a code example. It begins with the name of the script, followed by code snippets, and sections of content in between to explain what is happening.

my-script.js

This is where I introduce you to what on earth is going on.

```
// start of script
console.log('this is part 1 of my-script.js');
```

Above is the first bit of code. This is where I bore you of the details of what's going on, or what will happen next.

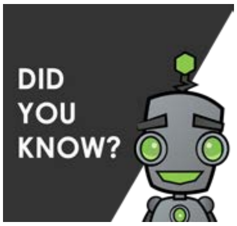
```
console.log('this is part 2 of my-script.js');
// end of script
```

Below is the output of the script '**my-script.js**'

```
$ node my-script.js
this is part 1 of my-script.js
this is part 2 of my-script.js
```

Did you know

When I'm feeling a bit generous, I provide some explanations to relevant areas associated with the code that we write.



Immutability in the context of programming - an immutable object is an object whose state cannot be changed once created. This can be useful because when you pass references of that object around, you can be rest assured other procedures will not be cheeky and modify it.

Coding time

When you see this pencil icon, get ready, because it's time to roll up your sleeves and get coding!

Running a terminal command

When I need to use the CLI, it may show as a single line.

```
node index.js
```

For multi-line, I will prefix the first line with a dollar sign.

```
$ npm install fs-extra
fetching fs-extra...
```

Prerequisites

1. A Laptop or Desktop.
2. Internet access.
3. A GitHub account with SSH set up correctly.
4. Ensure you are using the latest version of git to avoid legacy issues.
5. Make sure you have Node installed. This can be downloaded here for your Mac or Windows machine: https://nodejs.org/en. This book uses a minimum version of: **6.9.1**. At the time of writing, it should be fine to use any version above this.
6. Motivation. Please stick with it. The time you invest now will pay off in the long run.

Assumptions

It's assumed you have a simple understanding of JavaScript and GitHub. A basic idea of the CLI, and minimal - or no - experience of Node JS. All third party implementations are correct at the time of writing. Node throughout the book may be referenced as: Node, Node JS or Node.js but all references refer to the same technology.

Suggestions

Please feel free to suggest or contribute on GitHub (Raise a pull request) to the code examples as you see fit, or any possible typos in this book. You can also contact me via any of the social networks.

- GitHub - https://github.com/smks
- Twitter - https://twitter.com/shaunmstone
- Facebook - https://www.facebook.com/automatingwithnodejs
- YouTube - http://www.youtube.com/c/OpenCanvas

Or connect with me on LinkedIn for business-related requests.

LinkedIn - https://www.linkedin.com/in/shaunmstone

Technical Overview

Just to make sure we're all on the same page, here are some of the terms in this book that you should understand before proceeding. Feel free to skip past them if they're already familiar to you.

Technical Terms

CLI

Command Line Interface - is a textual interface for interacting with your computer. It is essentially a text-based application which takes in text input, processes it, and returns an output. When interacting with the examples in this book, you will need to open up a CLI and type in commands to make things happen, rather than clicking buttons and tabs with a mouse. If you are on Windows, this will be the Command Prompt (CMD) or PowerShell. If on Mac or Unix like systems, it will be the Terminal.

Bash

Bash is a shell command processor that runs in a CLI. You can write Bash scripts, and run them to execute a sequence of commands. You might first clone a repository, create a branch, add a text file with content, stage the file, commit it, and then push back to the remote repository all in one go. This would mean you wouldn't have to type out each command separately and is handy for automation. The reason this book does not use Bash is because – at the time of this writing – Windows does not fully support it, and we want our project to be cross platform. So we will be writing JavaScript with Node so our scripts will run on Windows as well.

Here is an example of a Bash script.

new-branch.sh

```
#!/bin/bash
# 0.0.1

git checkout master
git pull origin master
git checkout -b $1
```

Node.js

When you open up the CLI and type `node`, you are interacting with the node executable installed on your machine. When you pass a JavaScript file to it, the node executable executes the file. Node is an Event-driven I/O server-side JavaScript environment based on Google's V8 engine. It was designed to build scalable network applications. It processes incoming requests in a loop, known as the Event Loop, and operates on a single thread, using non-blocking I/O calls. This allows it to support a high volume of concurrent connections.

Node has two versions available on their website to download:

LTS

It stands for Long Term Support, and is the version of Node offering support and maintenance for at least 18 months. If you have a complex Node app and want stability, this would be the choice for you. Support and maintenance is correct at the time of writing.

Stable

Will have support for approximately 8 months, with more up-to-date features that are released more often. Use this version if you don't mind having to keep updating your application so you can keep in line with ‘on the edge’ technology.

I have opted to use the LTS version so that companies who are tied down with their version of Node will more likely be able to run the code examples and implement the build tool demonstrated in this book.

npm

When you download Node, it optionally gets bundled with a package manager called npm. It stands for Node Package Manager, and is the de facto for managing your external dependencies. If you wanted to use a library such as React or Angular, all you need to do is run `npm install [package name]`, npm would then download/install the package into your project’s `node_modules` directory, so it’s ready to be used in your app.

But this is not the only thing npm does after running this command. It also adds a record of this package to your project’s dependencies list in your `package.json`. This is very handy, as it means that your project keeps track of all its dependencies. But it gets much better.

Please note: As of npm 5.0.0, installed modules are added as a dependency to your `package.json` file by default. Before this version, you would have to add the option `--save` to do this.

Any developer wanting to use your app (including yourself from another machine) can install all dependencies with just one command: `npm install`. When running this command, npm goes through your dependency list in your project’s `package.json` file, and downloads them one by one into the `node_modules` directory.

To be able to use this dependency management goodness in a freshly created project, all you need to do is run `npm init`. This command will take you through a series of questions, and create an initial `package.json` file for you. This file, aside from keeping track of your project's dependencies, also has other information about your project, such as: project name, project version, repository details, author name, license, etc.

npm dependency

```
{organisation}/{package}
# examples
facebook/react
apache/cordova-cli
expressjs/express
```

Each dependency in the npm ecosystem has to have a unique identifier on the public registry, otherwise this would cause conflicts. Think of it like checking into a hotel, if you wanted room number seven because it's lucky, but someone else is already in there eating bread and olives, it means you'll have to settle for a different room. Same applies to package names. Anyone can create their own package and publish it to the registry, just make sure the package name you decide to use is available.

When I try to install the 'express' package, it will use the one created by the Express organisation. I can't publish a package called 'express' anymore as this is already taken.

Node Modules

When we want to break bits of code into separate files, we treat them as ‘modules’. These modules can be imported into other modules. In this example, I want to use code from the file `b.js` in my current file called `a.js`. Both files sit in the same directory for the following example.

a.js

```
const b = require('./b.js');

console.log('From a.js: running code in the file b.js');

b();
```

So above, we are importing the code from the file below:

b.js

```
const arsenalFanChant = () => {
  console.log('We love you Arsenal, we do!');
}
module.exports = arsenalFanChant;
```

Above shows `module.exports`. Whatever is assigned to this object from your JavaScript file, can be retrieved by doing a `require` from another JavaScript file.

Now, when we run script `a.js`.