



C o m m u n i t y   E x p e r i e n c e   D i s t i l l e d

# CoffeeScript Application Development

Write code that is easy to read, effortless to maintain, and even more powerful than JavaScript

Ian Young

**[PACKT]** open source\*  
PUBLISHING community experience distilled

# CoffeeScript Application Development

Write code that is easy to read, effortless to maintain,  
and even more powerful than JavaScript

Ian Young



BIRMINGHAM - MUMBAI

# CoffeeScript Application Development

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: August 2013

Production Reference: 1200813

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-78216-266-7

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Aniket Sawant ([aniket\\_sawant\\_photography@hotmail.com](mailto:aniket_sawant_photography@hotmail.com))

# Credits

**Author**

Ian Young

**Project Coordinator**

Kranti Berde

**Reviewers**

Becker

Adam Bronte

Enrique Vidal

**Proofreader**

Mario Cecere

**Indexer**

Tejal Soni

**Acquisition Editor**

Martin Bell

**Production Coordinator**

Prachali Bhiwandkar

**Lead Technical Editor**

Ankita Shashi

**Cover Work**

Prachali Bhiwandkar

**Technical Editors**

Dipika Gaonkar

Aparna K

Pragati Singh

Aniruddha Vanage

# About the Author

**Ian Young** wrote his very first program on a TI-89 scientific calculator — an infinite loop that printed an insulting message to one of his friends. As one might expect, things could only improve from there. Ian graduated from Grinnell College with a degree in Computer Science, and since then has been working as a web developer for small tech companies; first in Minneapolis and now in San Diego. He loves web technology, small teams, frequent iteration, testing, beautiful ideas, free speech, free beer, and any tool that reduces cognitive overhead.

# Acknowledgements

Katherine, for putting up with my stupid face.

My reviewers and editors, for finding all of my mistakes.

Photos, my favorite part of the book:

- Steve Jurvetson (<https://flickr.com/photos/jurvetson/2229899>)
- Rosalia Wilhelm (<https://commons.wikimedia.org/wiki/File:Widderkaninchen.JPG>)

Open source software, without which none of this would be possible:

- Jeremy Ashkenas, CoffeeScript (<http://coffeescript.org/>)
- Ryan Dahl, Node (<http://nodejs.org/>)
- Isaac Z. Schlueter, npm (<https://github.com/isaacs/npm>)
- Dustin Diaz, reqwest (<https://github.com/ded/reqwest>)
- Tilde, Inc., RSVP.js (<https://github.com/tildeio/rsvp.js>)
- David Heinemeier Hansson, Rails (<http://rubyonrails.org/>)
- Brunch team, Brunch (<http://brunch.io/>)
- TJ Holowaychuk, Express (<http://expressjs.com/>)
- Andrew Dunkman, connect-assets (<https://github.com/adunkman/connect-assets>)

# About the Reviewers

**Adam Bronte** is a well-versed software developer expert on web technologies. He is the co-founder and CTO of the pet services company, Furlocity. With over six years of experience in the industry, Adam has worked on all aspects of software development.

**Enrique Vidal** is a Software Engineer from Tijuana. He has worked on web development and system administration for many years, he is now focusing on Ruby and CoffeeScript development.

He has been fortunate to work with great developers such as this book's author, in different companies in the United States and México. He enjoys the challenge of coding payment systems, online invoicing, social networking applications, and so on. He is keen on helping startups at an early stage and actively supporting a few open source projects.

---

I'd like to thank Packt and the author for allowing me to be part of this book's technical reviewer team.

---

# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

## Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

## Free Access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.





# Table of Contents

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Running a CoffeeScript Program</b>	<b>7</b>
<b>Installing Node.js</b>	<b>7</b>
Installing Node.js on OS X	8
Using the installer	8
Using Homebrew	9
Using Macports	10
Installing Node.js on Windows	10
<b>Using the installer</b>	<b>11</b>
Using the standalone executable	12
Using Chocolatey	12
Installing Node.js on Linux	13
Using a graphical package manager	13
Using the command line	14
Compiling Node.js manually	15
Skipping the Node installation step	16
<b>Testing our Node installation</b>	<b>16</b>
Testing npm	17
<b>Installing CoffeeScript</b>	<b>18</b>
<b>Our very first CoffeeScript code</b>	<b>19</b>
<b>Compiling from a CoffeeScript file</b>	<b>19</b>
<b>CoffeeScript support in the editor</b>	<b>20</b>
Support in TextMate	20
Support in Sublime Text 2	21
Support in Vim	21
Support in Emacs	21
<b>Starting our web application</b>	<b>22</b>
One more thing	23
<b>Summary</b>	<b>24</b>

---

<b>Chapter 2: Writing Your First Lines of CoffeeScript</b>	<b>25</b>
<b>Following along with the examples</b>	<b>25</b>
Seeing the compiled JavaScript	26
<b>CoffeeScript basics</b>	<b>26</b>
Statements	27
Variables	27
Comments	28
<b>Calling functions</b>	<b>29</b>
Precedence	30
<b>Control structures</b>	<b>31</b>
Using if statements	31
The else and else if statements	33
The unless statement	33
Single-line form	34
<b>Comparison operators</b>	<b>35</b>
<b>Arrays</b>	<b>37</b>
Ranges	38
Loops	39
Loop comprehensions	41
A few more array tricks	42
Checking array membership	42
<b>Simple objects</b>	<b>43</b>
Iterating over objects	45
<b>Summary</b>	<b>45</b>
<b>Chapter 3: Building a Simple Application</b>	<b>47</b>
<b>Building our application</b>	<b>48</b>
<b>String Interpolation</b>	<b>52</b>
Using string interpolation in our application	53
<b>Defining functions</b>	<b>54</b>
Function naming	55
Function return behavior	56
<b>Adding dynamic behavior to our application</b>	<b>58</b>
<b>Switch statements</b>	<b>63</b>
Using a switch statement in our application	65
<b>Summary</b>	<b>67</b>
<b>Chapter 4: Improving Our Application</b>	<b>69</b>
<b>Checking if a value exists</b>	<b>69</b>
Using the existential operator	70
Null values in chained calls	71
Assigning new values conditionally when null	72

---

Dealing with nulls in our application	73
<b>Assigning multiple values at once</b>	<b>77</b>
Using destructuring assignment in our application	79
<b>Advanced function arguments</b>	<b>81</b>
Default argument values	83
Using default arguments in our application	84
Accepting a variable number of arguments with splats	87
Invoking functions with splats	88
Using splats in our application	89
<b>Summary</b>	<b>93</b>
<b>Chapter 5: Classes in CoffeeScript</b>	<b>95</b>
<b>Defining a class in CoffeeScript</b>	<b>95</b>
Attaching methods to a class	96
How CoffeeScript builds classes in JavaScript	97
Maintaining state with object properties	98
Calling other methods on this object	98
Attaching a method outside of the class definition	100
<b>Constructors</b>	<b>101</b>
CoffeeScript constructors in JavaScript	102
<b>Calling methods statically on classes</b>	<b>103</b>
<b>Inheritance</b>	<b>105</b>
CoffeeScript's inheritance in JavaScript	107
<b>Using CoffeeScript with other class libraries</b>	<b>109</b>
Backbone classes in CoffeeScript	110
Ember classes in CoffeeScript	111
<b>Summary</b>	<b>112</b>
<b>Chapter 6: Refactoring with Classes</b>	<b>113</b>
<b>The refactoring cycle</b>	<b>113</b>
<b>Structuring our data with classes</b>	<b>114</b>
Adding business logic	116
More data modeling	117
More business logic	118
<b>Managing display logic with classes</b>	<b>120</b>
Displaying a collection	122
The top-level display logic	124
<b>A final refactoring pass</b>	<b>125</b>
<b>Using inheritance while refactoring</b>	<b>128</b>
<b>Getting the green light</b>	<b>130</b>
<b>Summary</b>	<b>130</b>

---

<b>Chapter 7: Advanced CoffeeScript Usage</b>	<b>131</b>
<b>Getting our context right</b>	<b>131</b>
Using fat arrows in our project	134
<b>Saving our work with memoization</b>	<b>135</b>
Using memoization in our application	136
<b>A new idiom: options objects</b>	<b>138</b>
Using options objects in our application	139
<b>Summary</b>	<b>144</b>
<b>Chapter 8: Going Asynchronous</b>	<b>145</b>
<b>Understanding asynchronous operations</b>	<b>145</b>
<b>Getting to know our remote API</b>	<b>147</b>
<b>Making an asynchronous request</b>	<b>148</b>
<b>Using a third-party library</b>	<b>151</b>
Refactoring	152
<b>Wrangling multiple asynchronous calls</b>	<b>153</b>
Requests in a loop	154
Determining when we're finished	157
<b>Alternatives for managing asynchronous calls</b>	<b>158</b>
Promises	158
Using Promises in our application	159
An async helper library	162
Using Async.js in our application	162
IcedCoffeeScript	164
Using IcedCoffeeScript in our application	165
<b>Summary</b>	<b>167</b>
<b>Chapter 9: Debugging</b>	<b>169</b>
<b>Discovering a problem</b>	<b>169</b>
<b>Working with source maps</b>	<b>170</b>
Source maps in the Firefox developer tools	171
Inspecting our application state	172
Using the debugger	174
Source maps in the Chrome developer tools	178
Inspecting our application state	178
Using the debugger	180
<b>Fixing the problem</b>	<b>184</b>
<b>Summary</b>	<b>185</b>
<b>Chapter 10: Using CoffeeScript in More Places</b>	<b>187</b>
<b>CoffeeScript directly in the browser</b>	<b>188</b>
<b>CoffeeScript in the browser console</b>	<b>189</b>
A CoffeeScript console in Firefox	189
A CoffeeScript console in Chrome	192

---

<b>Using CoffeeScript with Rails</b>	<b>194</b>
Setting up the asset pipeline	195
Creating a new Rails application	195
Rails 3.0	195
Rails 3.1 and 3.2	196
Rails 4	197
Setting up our application	198
Adding some CoffeeScript	199
Precompiling assets	202
<b>Using CoffeeScript with Brunch</b>	<b>203</b>
Creating a Brunch project	203
Filling out our application	204
Precompiling assets	207
<b>Using CoffeeScript with Node.js</b>	<b>208</b>
Creating our project	208
Keeping the server up-to-date	209
Adding CoffeeScript compilation	210
Finishing our application	211
Cleaning up our script dependencies	213
<b>Summary</b>	<b>215</b>
<b>Chapter 11: CoffeeScript on the Server</b>	<b>217</b>
<hr/>	
<b>Running a server with CoffeeScript</b>	<b>217</b>
Running our application	219
<b>Adding an endpoint for data</b>	<b>220</b>
<b>Using a database</b>	<b>223</b>
Handling errors	225
<b>Using a Cakefile</b>	<b>226</b>
Writing a Cake task	227
More Cake tasks	227
<b>Making our application interactive</b>	<b>228</b>
Seeing the results	231
<b>Summary</b>	<b>233</b>
<b>Index</b>	<b>235</b>

---



# Preface

If you do web development, chances are you've at least *heard* of CoffeeScript. Though it's less than five years old, this little language has received a lot of attention, and it's getting harder to ignore. Maybe you've already worked with it a little bit, or maybe you're just wondering what the fuss is all about. Good news! CoffeeScript is a delightful language that can help you write better code and have fun doing it. In this book, we will explore the language itself, and find out first-hand how it can help us build beautiful web applications.

## What is CoffeeScript?

CoffeeScript is a programming language. Like most programming languages, it offers control structures to describe the logic of our application, simple data types to store and manipulate information, and functions to encapsulate sections of program execution.

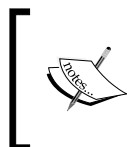
What makes CoffeeScript special is the way it is compiled. When most languages are compiled, they are translated into **machine code**—low-level instructions to the computer's processor. CoffeeScript is different: when compiled, it is instead translated *into JavaScript*. We write CoffeeScript code, give it to the CoffeeScript compiler and receive JavaScript code as output. This output can then be passed to anything that consumes JavaScript, such as a browser, or a standalone JavaScript interpreter.

This technique, dubbed **transcompilation**, allows us to use an alternative language on platforms that only directly support JavaScript. Client-side web development is the most prominent example, since JavaScript is the only supported general-purpose scripting solution on most web browsers. Other platforms such as Node.js and Rhino also offer useful features, but expect JavaScript input. JavaScript is nothing if not prolific, and CoffeeScript allows us to make use of all that existing tooling, but to write our code in a different language.



## Why CoffeeScript?

CoffeeScript was certainly not the first (or last) language to target JavaScript platforms. Many established languages, such as Ruby, Python, C, and Java have one or more projects focused on compiling that language to JavaScript. And other languages have been developed specifically to target JavaScript—notably Dart, TypeScript, and Coco.



The CoffeeScript wiki itself maintains an extensive list of other languages that compile to JavaScript. You can find it at <https://github.com/jashkenas/coffee-script/wiki/List-of-languages-that-compile-to-JS>.

While it's not alone in its approach, CoffeeScript has seen the most success of any language that compiles to JavaScript. It is the tenth most popular language on GitHub, it ships by default with Ruby on Rails, and it has large followings in both client-side and server-side developer communities.

So what makes CoffeeScript special? Just like Goldilocks and her pilfered porridge, CoffeeScript derives its strength from being *just right*. It is a marked improvement over JavaScript; we'll spend much of this book learning how CoffeeScript can help us write code that is more concise, easier to read, and less prone to bugs. However, CoffeeScript does not overreach on features. CoffeeScript has little to no runtime of its own—there is no extra metadata to track, no extra memory management, no non-standard data structures. Instead, CoffeeScript compiles directly to ordinary-looking JavaScript, much like what an experienced JavaScript developer might write. In fact, CoffeeScript is less a new language than it is a shorthand for easily expressing the best practices of JavaScript.

CoffeeScript is an eminently pragmatic language, and this is the secret to its success. It's easy for JavaScript developers to learn, and most expertise carries over. It doesn't incur performance penalties over plain JavaScript. CoffeeScript and JavaScript can coexist peacefully, so it's easy to introduce CoffeeScript into existing JavaScript projects. Perhaps most importantly, CoffeeScript avoids the "magic" that is so often a source of bugs when the developer's assumptions don't match the language designer's assumptions. With CoffeeScript, it's very easy to understand what the resulting JavaScript will do and how it will behave.

It might also help that CoffeeScript is *fun*.

## What this book covers

*Chapter 1, Running a CoffeeScript Program*, will cover installing the CoffeeScript tools and running a simple CoffeeScript program in both the console and a web browser.

*Chapter 2, Writing Your First Lines of CoffeeScript*, will explore the syntax of CoffeeScript and how it compiles to JavaScript.

In *Chapter 3, Building a Simple Application*, we will build an interactive web application and learn a few more CoffeeScript features along the way.

*Chapter 4, Improving Our Application*, will add more features to our web application, and explore more powerful CoffeeScript syntax.

*Chapter 5, Using Classes*, will teach us all about classes in CoffeeScript. It will also cover how to use them, how they work, and how to integrate with popular JavaScript frameworks.

In *Chapter 6, Refactoring with Classes*, we will use the new skills from previous chapter to refactor our web application using class-based structures.

In *Chapter 7, Advanced CoffeeScript Features*, we will learn advanced CoffeeScript features and idioms that reduce errors and make our code easier to understand. We will use them to add more features to our web application.

*Chapter 8, Going Asynchronous*, will show how CoffeeScript can help us deal with asynchronous operations, and integrate a third-party JavaScript library into our CoffeeScript application.

In *Chapter 9, Debugging*, we will learn how to use source maps to track problems in our application all the way back to the CoffeeScript source.

*Chapter 10, Using CoffeeScript in More Places*, will cover how to integrate CoffeeScript compilation into several popular web application frameworks.

In *Chapter 11, CoffeeScript on the Server*, we will run CoffeeScript on the server with Node.js, and learn how to integrate it with standard JavaScript Node modules.

## What you need for this book

All you need for this book is a text editor and a working CoffeeScript compiler, and don't worry about the compiler — we'll cover installation and use of that tool in the first chapter! We provide instructions for using the tools on Windows, Mac OS X, and Linux. We'll be spending a lot of the book working on a client-side web application, so if you have any favorite development tools, feel free to bring those along. You'll also need a modern browser. The most recent version of Firefox or Chrome is ideal, but any other up-to-date browser such as Safari, Opera, or a *recent* Internet Explorer will also work fine.

## Who this book is for

Some familiarity with the JavaScript language will help — CoffeeScript is a close relative, so it's useful to understand what the compiler's output is doing. It's also helpful, though not necessary, to have some experience with client-side web development. We'll be building a web application with a lot of CoffeeScript, plus a little HTML and CSS.

No experience with CoffeeScript is necessary. We'll cater to everyone from the total newbie to the person who has hacked together some CoffeeScript already but wants a better grasp of what's going on and how to best utilize the language.

## Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "We can pull in another module by using the `require` function."

A block of code is set as follows:

```
fibonacci = (n) ->
  if n is 0 or n is 1
    n
  else
    fibonacci(n-1) + fibonacci(n-2)
```


When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:


```
fibonacci = (n) ->
  if n is 0 or n is 1
    n
  else
    fibonacci(n-1) + fibonacci(n-2)
```

Any command-line input or output is written as follows:

```
coffee --compile --watch *.coffee
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "clicking the **Next** button moves you to the next screen".

[  Warnings or important notes appear in a box like this. ]

[  Tips and tricks appear like this. ]

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

## Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## Questions

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.

# 1

## Running a CoffeeScript Program

The very first thing we need to do in order to start using CoffeeScript is to install CoffeeScript itself. This will give us access to the CoffeeScript compiler, which we'll use to compile our beautiful CoffeeScript code into JavaScript that can be run in a browser (or other JavaScript environment). By the end of this chapter we'll be completely set up and ready to work.

There are a couple of steps involved in installing CoffeeScript. I know you're impatient to dive right into this great new language— who can blame you? But we'll have to stick it out through a little bit of system configuration. If we do so, we'll be rewarded with a stable CoffeeScript setup that works flawlessly and doesn't take any more of our attention.

In this chapter we will:

- Install the software that you need to run CoffeeScript code
- Learn how to use the software to run CoffeeScript, both from the command line and in a browser
- Use our new abilities to write a simple web application using CoffeeScript

### Installing Node.js

To run CoffeeScript, first you'll need to install Node.js. Don't worry! If you don't want to learn Node.js, you won't need to. We just need to have the platform installed because the CoffeeScript compiler uses it.



If you get stuck at any point while installing or using Node.js, the IRC channel is a great place to look for help. You can use your IRC client of choice to connect to the #node.js room in irc.freenode.net, or you can connect through a web browser by visiting <http://webchat.freenode.net/?channels=node.js>.

**Node.js** (or simply **Node**) is a platform for running JavaScript at a low level, using the powerful and fast V8 engine. It's primarily used for web development, allowing developers to write the server side components of web applications in JavaScript. Node's most notable innovation is that it's highly non-blocking. Any system call that needs to wait for a result (such as network requests and disk reads) uses a callback, so Node can service another request while it waits for an operation to finish. This way of thinking meshes nicely with web applications which do a lot of network interaction, and it provides a lot of bang for your hardware buck. While we'll be using CoffeeScript to build a client side application, it works great with Node as well. We'll show you more about that in *Chapter 11, CoffeeScript on the Server*. The CoffeeScript compiler is written entirely in CoffeeScript and runs on Node. If you're curious, you can find the annotated CoffeeScript source on <http://coffeescript.org/>.

## Installing Node.js on OS X

The Node project provides several options for installation on Mac OS X. The simplest method is the universal installer. If you don't already use a package management system for your development tools, you should use the installer. If you use **Homebrew** or **MacPorts** to manage your system and would like to install Node through those, follow the instructions for your package manager instead.



If for some reason none of these methods work for you, see the *Compiling Node.js manually* section. However, this is more difficult and not recommended unless you really need it.

## Using the installer

The Node project provides a universal installer for Mac OS X. Visit <http://nodejs.org/download/>, and look for **Macintosh Installer**.



Download that file and double-click on it. Follow the prompts to install Node on your system.

## Using Homebrew

**Homebrew** is a popular package management system for OS X. It maintains installed packages in a completely separate directory from the OS X system files, and offers easy package management from the command line. Homebrew offers an easy-to-use **formula** system to create new package definitions, and as a result offers a very large collection of user-contributed recipes.

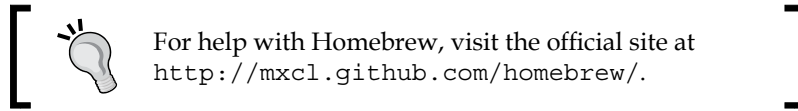


Early versions of the Node package on Homebrew suffered from numerous bugs. Recent versions have received far fewer complaints and should be acceptable for our needs. Still, if you encounter serious problems using Node from Homebrew, consider uninstalling it and using the universal installer instead.



To install Node using Homebrew, simply use the command-line installer as follows:

```
brew install node
```



## Using Macports

**MacPorts** is another package management system for OS X. Like Homebrew, it maintains installed packages separately from the OS X system files. MacPorts is an older project, and is modeled on the BSD **ports** system. While it has been waning in popularity in recent years, it still has a large user base.

To install Node using MacPorts, simply use the command-line installer as follows:

```
sudo port install nodejs
```



## Installing Node.js on Windows

There are several convenient installation options for Node on Windows. The method recommended for most people is to use the installer. If you cannot install software on your machine, or for other reasons wish to isolate Node, you can use the standalone executable instead. Finally, if you are already using the third-party package manager Chocolatey to manage packages on your machine, you may install Node through that system.

## Using the installer

The Node project provides an install file for Windows systems. Visit <http://nodejs.org/download/>, and look for **Windows Installer**:



The screenshot shows the Node.js download page. At the top is the Node.js logo. Below it, text says "Download the Node.js source code or a pre-built installer for your platform, and start developing today." and "Current version: v0.8.15". There are three main download options: "Windows Installer" (highlighted with a red dashed box), "Macintosh Installer", and "Source Code". Below these are links to download specific files: "node-v0.8.15-x86.msi" for Windows, "node-v0.8.15.pkg" for Mac, and "node-v0.8.15.tar.gz" for Source Code. A table below lists various download options for different operating systems and architectures.

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.exe)	32-bit	64-bit
Mac OS X Installer (.pkg)	Universal	
Mac OS X Binaries (.tar.gz)	32-bit	64-bit
Linux Binaries (.tar.gz)	32-bit	64-bit
SunOS Binaries (.tar.gz)	32-bit	64-bit
Source Code	node-v0.8.15.tar.gz	

Download that file and double-click on it. Follow the prompts to install Node on your system.