LEARN WEB DEVELOPMENT BY EXAMPLE

THE RUBY ON RAILS TUTORIAL

version 3.2

BOOK AND SCREENCASTS BY

Michael Hartl

Ruby on Rails Tutorial

Learn Rails by Example

Michael Hartl

Contents

1 From zero to deploy **1**

1.1 Introduction 4
1.1.1 Comments for various readers 5
1.1.2 "Scaling" Rails 8
1.1.3 Conventions in this book 9
1.2 Up and running 11
1.2.1 Development environments 12
1.2.2 Ruby, RubyGems, Rails, and Git 15
1.2.3 The first application 21
1.2.4 Bundler 22
1.2.5 `rails server` 29
1.2.6 Model-view-controller (MVC) 32
1.3 Version control with Git 32
1.3.1 Installation and setup 34
1.3.2 Adding and committing 37
1.3.3 What good does Git do you? 38
1.3.4 GitHub 40
1.3.5 Branch, edit, commit, merge 41
1.4 Deploying 47
1.4.1 Heroku setup 48
1.4.2 Heroku deployment, step one 50
1.4.3 Heroku deployment, step two 51
1.4.4 Heroku commands 51
1.5 Conclusion 54

2 A demo app **55**
2.1 Planning the application . . . 56
2.1.1 Modeling demo users . . . 57
2.1.2 Modeling demo microposts . . . 59
2.2 The Users resource . . . 59
2.2.1 A user tour . . . 62
2.2.2 MVC in action . . . 70
2.2.3 Weaknesses of this Users resource . . . 76
2.3 The Microposts resource . . . 77
2.3.1 A micropost microtour . . . 77
2.3.2 Putting the *micro* in microposts . . . 80
2.3.3 A user `has_many` microposts . . . 83
2.3.4 Inheritance hierarchies . . . 86
2.3.5 Deploying the demo app . . . 90
2.4 Conclusion . . . 90

3 Mostly static pages **93**
3.1 Static pages . . . 99
3.1.1 Truly static pages . . . 99
3.1.2 Static pages with Rails . . . 103
3.2 Our first tests . . . 112
3.2.1 Test-driven development . . . 113
3.2.2 Adding a page . . . 119
3.3 Slightly dynamic pages . . . 124
3.3.1 Testing a title change . . . 124
3.3.2 Passing title tests . . . 128
3.3.3 Embedded Ruby . . . 129
3.3.4 Eliminating duplication with layouts . . . 132
3.4 Conclusion . . . 135
3.5 Exercises . . . 136
3.6 Advanced setup . . . 139
3.6.1 Eliminating `bundle exec` . . . 140
3.6.2 Automated tests with Guard . . . 142
3.6.3 Speeding up tests with Spork . . . 146

3.6.4 Tests inside Sublime Text 151

4 Rails-flavored Ruby **153**
4.1 Motivation 153
4.2 Strings and methods 158
4.2.1 Comments 159
4.2.2 Strings 160
4.2.3 Objects and message passing 163
4.2.4 Method definitions 165
4.2.5 Back to the title helper 166
4.3 Other data structures 167
4.3.1 Arrays and ranges 167
4.3.2 Blocks 171
4.3.3 Hashes and symbols 174
4.3.4 CSS revisited 177
4.4 Ruby classes 179
4.4.1 Constructors 179
4.4.2 Class inheritance 180
4.4.3 Modifying built-in classes 183
4.4.4 A controller class 186
4.4.5 A user class 187
4.5 Conclusion 191
4.6 Exercises 191

5 Filling in the layout **193**
5.1 Adding some structure 193
5.1.1 Site navigation 194
5.1.2 Bootstrap and custom CSS 201
5.1.3 Partials 212
5.2 Sass and the asset pipeline 216
5.2.1 The asset pipeline 217
5.2.2 Syntactically awesome stylesheets 220
5.3 Layout links 227
5.3.1 Route tests 230

5.3.2 Rails routes . . . 232
5.3.3 Named routes . . . 236
5.3.4 Pretty RSpec . . . 237
5.4 User signup: A first step . . . 243
5.4.1 Users controller . . . 243
5.4.2 Signup URI . . . 244
5.5 Conclusion . . . 247
5.6 Exercises . . . 249

6 Modeling users **253**
6.1 User model . . . 254
6.1.1 Database migrations . . . 256
6.1.2 The model file . . . 261
6.1.3 Creating user objects . . . 264
6.1.4 Finding user objects . . . 267
6.1.5 Updating user objects . . . 269
6.2 User validations . . . 270
6.2.1 Initial user tests . . . 271
6.2.2 Validating presence . . . 274
6.2.3 Length validation . . . 278
6.2.4 Format validation . . . 280
6.2.5 Uniqueness validation . . . 283
6.3 Adding a secure password . . . 291
6.3.1 An encrypted password . . . 291
6.3.2 Password and confirmation . . . 294
6.3.3 User authentication . . . 298
6.3.4 User has secure password . . . 302
6.3.5 Creating a user . . . 305
6.4 Conclusion . . . 307
6.5 Exercises . . . 308

7 Sign up **311**
7.1 Showing users . . . 312
7.1.1 Debug and Rails environments . . . 312

7.1.2 A Users resource . . . 319
7.1.3 Testing the user show page (with factories) . . . 325
7.1.4 A Gravatar image and a sidebar . . . 330
7.2 Signup form . . . 336
7.2.1 Tests for user signup . . . 340
7.2.2 Using `form_for` . . . 344
7.2.3 The form HTML . . . 347
7.3 Signup failure . . . 351
7.3.1 A working form . . . 351
7.3.2 Signup error messages . . . 357
7.4 Signup success . . . 364
7.4.1 The finished signup form . . . 364
7.4.2 The flash . . . 366
7.4.3 The first signup . . . 369
7.4.4 Deploying to production with SSL . . . 371
7.5 Conclusion . . . 373
7.6 Exercises . . . 373

8 Sign in, sign out **379**
8.1 Sessions and signin failure . . . 380
8.1.1 Sessions controller . . . 380
8.1.2 Signin tests . . . 385
8.1.3 Signin form . . . 390
8.1.4 Reviewing form submission . . . 393
8.1.5 Rendering with a flash message . . . 397
8.2 Signin success . . . 401
8.2.1 Remember me . . . 402
8.2.2 A working `sign_in` method . . . 409
8.2.3 Current user . . . 412
8.2.4 Changing the layout links . . . 417
8.2.5 Signin upon signup . . . 421
8.2.6 Signing out . . . 425
8.3 Introduction to Cucumber (optional) . . . 427
8.3.1 Installation and setup . . . 428

8.3.2 Features and steps 428
8.3.3 Counterpoint: RSpec custom matchers 432
8.4 Conclusion 436
8.5 Exercises 436

9 Updating, showing, and deleting users **439**
9.1 Updating users 439
9.1.1 Edit form 440
9.1.2 Unsuccessful edits 448
9.1.3 Successful edits 450
9.2 Authorization 453
9.2.1 Requiring signed-in users 453
9.2.2 Requiring the right user 458
9.2.3 Friendly forwarding 460
9.3 Showing all users 464
9.3.1 User index 466
9.3.2 Sample users 471
9.3.3 Pagination 474
9.3.4 Partial refactoring 483
9.4 Deleting users 484
9.4.1 Administrative users 486
9.4.2 The `destroy` action 490
9.5 Conclusion 496
9.6 Exercises 498

10 User microposts **503**
10.1 A Micropost model 503
10.1.1 The basic model 504
10.1.2 Accessible attributes and the first validation 506
10.1.3 User/Micropost associations 508
10.1.4 Micropost refinements 514
10.1.5 Content validations 522
10.2 Showing microposts 524
10.2.1 Augmenting the user show page 524

10.2.2 Sample microposts 530
10.3 Manipulating microposts 537
10.3.1 Access control 538
10.3.2 Creating microposts 541
10.3.3 A proto-feed 551
10.3.4 Destroying microposts 560
10.4 Conclusion 566
10.5 Exercises 566

11 Following users **571**
11.1 The Relationship model 572
11.1.1 A problem with the data model (and a solution) 572
11.1.2 User/relationship associations 582
11.1.3 Validations 586
11.1.4 Followed users 587
11.1.5 Followers 592
11.2 A web interface for following users 595
11.2.1 Sample following data 596
11.2.2 Stats and a follow form 597
11.2.3 Following and followers pages 607
11.2.4 A working follow button the standard way 617
11.2.5 A working follow button with Ajax 620
11.3 The status feed 627
11.3.1 Motivation and strategy 627
11.3.2 A first feed implementation 630
11.3.3 Subselects 634
11.3.4 The new status feed 637
11.4 Conclusion 637
11.4.1 Extensions to the sample application 639
11.4.2 Guide to further resources 642
11.5 Exercises 643

12 Rails 4.0 supplement **645**
12.1 Upgrading from Rails 3.2 to 4.0 646

12.1.1 Rails 4.0 setup 646
12.1.2 Getting to green 651
12.1.3 Some specific issues 652
12.1.4 Finishing up 657
12.1.5 Additional resources 658
12.2 Strong parameters 658
12.3 Security updates 661
12.3.1 Secret key 661
12.3.2 Encrypted remember tokens 662

Foreword

My former company (CD Baby) was one of the first to loudly switch to Ruby on Rails, and then even more loudly switch back to PHP (Google me to read about the drama). This book by Michael Hartl came so highly recommended that I had to try it, and the *Ruby on Rails Tutorial* is what I used to switch back to Rails again.

Though I've worked my way through many Rails books, this is the one that finally made me "get" it. Everything is done very much "the Rails way"—a way that felt very unnatural to me before, but now after doing this book finally feels natural. This is also the only Rails book that does test-driven development the entire time, an approach highly recommended by the experts but which has never been so clearly demonstrated before. Finally, by including Git, GitHub, and Heroku in the demo examples, the author really gives you a feel for what it's like to do a real-world project. The tutorial's code examples are not in isolation.

The linear narrative is such a great format. Personally, I powered through the *Rails Tutorial* in three long days, doing all the examples and challenges at the end of each chapter. Do it from start to finish, without jumping around, and you'll get the ultimate benefit.

Enjoy!

Derek Sivers (sivers.org)
Founder, CD Baby

Acknowledgments

The *Ruby on Rails Tutorial* owes a lot to my previous Rails book, *RailsSpace*, and hence to my coauthor Aurelius Prochazka. I'd like to thank Aure both for the work he did on that book and for his support of this one. I'd also like to thank Debra Williams Cauley, my editor on both *RailsSpace* and the *Ruby on Rails Tutorial*; as long as she keeps taking me to baseball games, I'll keep writing books for her.

I'd like to acknowledge a long list of Rubyists who have taught and inspired me over the years: David Heinemeier Hansson, Yehuda Katz, Carl Lerche, Jeremy Kemper, Xavier Noria, Ryan Bates, Geoffrey Grosenbach, Peter Cooper, Matt Aimonetti, Gregg Pollack, Wayne E. Seguin, Amy Hoy, Dave Chelimsky, Pat Maddox, Tom Preston-Werner, Chris Wanstrath, Chad Fowler, Josh Susser, Obie Fernandez, Ian McFarland, Steven Bristol, Pratik Naik, Sarah Mei, Sarah Allen, Wolfram Arnold, Alex Chaffee, Giles Bowkett, Evan Dorn, Long Nguyen, James Lindenbaum, Adam Wiggins, Tikhon Bernstam, Ron Evans, Wyatt Greene, Miles Forrest, the good people at Pivotal Labs, the Heroku gang, the thoughtbot guys, and the GitHub crew. Finally, many, many readers—far too many to list—have contributed a huge number of bug reports and suggestions during the writing of this book, and I gratefully acknowledge their help in making it as good as it can be.

About the author

Michael Hartl is the author of the *Ruby on Rails Tutorial*, the leading introduction to web development with Ruby on Rails. His prior experience includes writing and developing *RailsSpace*, an extremely obsolete Rails tutorial book, and developing Insoshi, a once-popular and now-obsolete social networking platform in Ruby on Rails. In 2011, Michael received a Ruby Hero Award for his contributions to the Ruby community. He is a graduate of Harvard College, has a Ph.D. in Physics from Caltech, and is an alumnus of the Y Combinator entrepreneur program.

Copyright and license


```
The MIT License

Copyright (c) 2013 Michael Hartl

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

```
/*
 * ----------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * Michael Hartl wrote this code. As long as you retain this notice you
 * can do whatever you want with this stuff. If we meet some day, and you think
 * this stuff is worth it, you can buy me a beer in return.
 * ----------------------------------------------------------------------------
 */
```

Chapter 1

From zero to deploy

Welcome to the *Ruby on Rails Tutorial*. The goal of this book is to be the best answer to the question, "If I want to learn web development with Ruby on Rails, where should I start?" By the time you finish the *Ruby on Rails Tutorial*, you will have all the skills you need to develop and deploy your own custom web applications with Rails. You will also be ready to benefit from the many more advanced books, blogs, and screencasts that are part of the thriving Rails educational ecosystem. Finally, since the *Ruby on Rails Tutorial* uses Rails 3, the knowledge you gain here represents the state of the art in web development (Box 1.1). (The most up-to-date version of the *Ruby on Rails Tutorial* can be found on the book's website at http://railstutorial.org/; if you are reading this book offline, be sure to check the online version of the Rails Tutorial book at http://railstutorial.org/book?version=3.2 for the latest updates.)

Box 1.1. Rails 3 or Rails 4?

As of this writing, both Rails 3.2 and Rails 4.0 are officially supported versions of Rails, so the question naturally arises whether it might be better to start with the latest version (Rails 4.0) rather than the one used in this tutorial (Rails 3.2). The answer depends on your background, but if you have never used Rails before I suggest using Rails 3.2. This is partially because the Rails 4.0 ecosystem is still somewhat unstable; in addition, the differences between the two versions are

slight, with only the introduction of so-called "strong parameters" representing a significant change from the perspective of an introductory tutorial.

In order to ease the transition to Rails 4.0, both this book and the Rails Tutorial screencasts contain supplementary material on upgrading to Rails 4.0, as well an additional discussion of strong parameters and important security updates (Chapter 12). My recommendation is to learn web development first with Rails 3.2 (using the book and optionally the screencasts), and then use the supplement to learn the small number of things needed to bring your knowledge up to date with Rails 4.0.

On the other hand, if you already have a background in Rails and want to live on the bleeding edge, you are welcome to use the Rails 4.0 version of the *Rails Tutorial*, which uses Rails 4.0 throughout. Box 1.1 of the 4.0 version also contains a nearly comprehensive (but still quite short) list of the differences between the Rails 3.2 and Rails 4.0 versions of the book.

Note that the goal of this book is *not* merely to teach Rails, but rather to teach *web development with Rails*, which means acquiring (or expanding) the skills needed to develop software for the World Wide Web. In addition to Ruby on Rails, this skillset includes HTML & CSS, databases, version control, testing, and deployment. To accomplish this goal, the *Ruby on Rails Tutorial* takes an integrated approach: you will learn Rails by example by building a substantial sample application from scratch. As Derek Sivers notes in the foreword, this book is structured as a linear narrative, designed to be read from start to finish. If you are used to skipping around in technical books, taking this linear approach might require some adjustment, but I suggest giving it a try. You can think of the *Ruby on Rails Tutorial* as a video game where you are the main character, and where you level up as a Rails developer in each chapter. (The exercises are the minibosses.)

In this first chapter, we'll get started with Ruby on Rails by installing all the necessary software and by setting up our development environment (Section 1.2). We'll then create our first Rails application, called (appropriately enough) `first_app`. The *Rails Tutorial* emphasizes good software develop-

ment practices, so immediately after creating our fresh new Rails project we'll put it under version control with Git (Section 1.3). And, believe it or not, in this chapter we'll even put our first app on the wider web by *deploying* it to production (Section 1.4).

In Chapter 2, we'll make a second project, whose purpose is to demonstrate the basic workings of a Rails application. To get up and running quickly, we'll build this *demo app* (called `demo_app`) using scaffolding (Box 1.2) to generate code; since this code is both ugly and complex, Chapter 2 will focus on interacting with the demo app through its *URIs* (sometimes called *URLs*)[1] using a web browser.

The rest of the tutorial focuses on developing a single large *sample application* (called `sample_app`), writing all the code from scratch. We'll develop the sample app using *test-driven development* (TDD), getting started in Chapter 3 by creating static pages and then adding a little dynamic content. We'll take a quick detour in Chapter 4 to learn a little about the Ruby language underlying Rails. Then, in Chapter 5 through Chapter 9, we'll complete the foundation for the sample application by making a site layout, a user data model, and a full registration and authentication system. Finally, in Chapter 10 and Chapter 11 we'll add microblogging and social features to make a working example site.

The final sample application will bear more than a passing resemblance to a certain popular social microblogging site—a site which, coincidentally, was also originally written in Rails. Though of necessity our efforts will focus on this specific sample application, the emphasis throughout the *Rails Tutorial* will be on general principles, so that you will have a solid foundation no matter what kinds of web applications you want to build.

Box 1.2. Scaffolding: Quicker, easier, more seductive

From the beginning, Rails has benefited from a palpable sense of excitement, starting with the famous 15-minute weblog video by Rails creator David Heinemeier Hansson. That video and its successors are a great way to get a taste of

[1] *URI* stands for Uniform Resource Identifier, while the slightly less general *URL* stands for Uniform Resource Locator. In practice, the URI is usually equivalent to "the thing you see in the address bar of your browser".

Rails' power, and I recommend watching them. But be warned: they accomplish their amazing fifteen-minute feat using a feature called *scaffolding*, which relies heavily on *generated code*, magically created by the Rails `generate` command.

When writing a Ruby on Rails tutorial, it is tempting to rely on the scaffolding approach—it's quicker, easier, more seductive. But the complexity and sheer amount of code in the scaffolding can be utterly overwhelming to a beginning Rails developer; you may be able to use it, but you probably won't understand it. Following the scaffolding approach risks turning you into a virtuoso script generator with little (and brittle) actual knowledge of Rails.

In the *Ruby on Rails Tutorial*, we'll take the (nearly) polar opposite approach: although Chapter 2 will develop a small demo app using scaffolding, the core of the *Rails Tutorial* is the sample app, which we'll start writing in Chapter 3. At each stage of developing the sample application, we will write *small, bite-sized* pieces of code—simple enough to understand, yet novel enough to be challenging. The cumulative effect will be a deeper, more flexible knowledge of Rails, giving you a good background for writing nearly any type of web application.

1.1 Introduction

Since its debut in 2004, Ruby on Rails has rapidly become one of the most powerful and popular frameworks for building dynamic web applications. Everyone from scrappy startups to huge companies have used Rails: 37signals, GitHub, Shopify, Scribd, Twitter, LivingSocial, Groupon, Hulu, the Yellow Pages—the list of sites using Rails goes on and on. There are also many web development shops that specialize in Rails, such as ENTP, thoughtbot, Pivotal Labs, and Hashrocket, plus innumerable independent consultants, trainers, and contractors.

What makes Rails so great? First of all, Ruby on Rails is 100% open-source, available under the permissive MIT License, and as a result it also costs nothing to download or use. Rails also owes much of its success to its elegant and compact design; by exploiting the malleability of the underlying

Ruby language, Rails effectively creates a domain-specific language for writing web applications. As a result, many common web programming tasks—such as generating HTML, making data models, and routing URIs—are easy with Rails, and the resulting application code is concise and readable.

Rails also adapts rapidly to new developments in web technology and framework design. For example, Rails was one of the first frameworks to fully digest and implement the REST architectural style for structuring web applications (which we'll be learning about throughout this tutorial). And when other frameworks develop successful new techniques, Rails creator David Heinemeier Hansson and the Rails core team don't hesitate to incorporate their ideas. Perhaps the most dramatic example is the merger of Rails and Merb, a rival Ruby web framework, so that Rails now benefits from Merb's modular design, stable API, and improved performance.

Finally, Rails benefits from an unusually enthusiastic and diverse community. The results include hundreds of open-source contributors, well-attended conferences, a huge number of plugins and gems (self-contained solutions to specific problems such as pagination and image upload), a rich variety of informative blogs, and a cornucopia of discussion forums and IRC channels. The large number of Rails programmers also makes it easier to handle the inevitable application errors: the "Google the error message" algorithm nearly always produces a relevant blog post or discussion-forum thread.

1.1.1 Comments for various readers

The *Rails Tutorial* contains integrated tutorials not only for Rails, but also for the underlying Ruby language, the RSpec testing framework, HTML, CSS, a small amount of JavaScript, and even a little SQL. This means that, no matter where you currently are in your knowledge of web development, by the time you finish this tutorial you will be ready for more advanced Rails resources, as well as for the more systematic treatments of the other subjects mentioned. It also means that there's a *lot* of material to cover; if you don't already have much experience programming computers, you might find it overwhelming. The comments below contain some suggestions for approaching the *Rails Tutorial* depending on your background.

All readers: One common question when learning Rails is whether to learn Ruby first. The answer depends on your personal learning style and how much programming experience you already have. If you prefer to learn everything systematically from the ground up, or if you have never programmed before, then learning Ruby first might work well for you, and in this case I recommend *Beginning Ruby* by Peter Cooper. On the other hand, many beginning Rails developers are excited about making *web* applications, and would rather not slog through a 500-page book on pure Ruby before ever writing a single web page. In this case, I recommend following the short interactive tutorial at Try Ruby,[2] and then optionally do the free tutorial at Rails for Zombies[3] to get a taste of what Rails can do.

Another common question is whether to use tests from the start. As noted in the introduction, the *Rails Tutorial* uses test-driven development (also called test-first development), which in my view is the best way to develop Rails applications, but it does introduce a substantial amount of overhead and complexity. If you find yourself getting bogged down by the tests, I suggest either skipping them on a first reading or (even better) using them as a tool to verify your code's correctness without worrying about how they work. This latter strategy involves creating the necessary test files (called *specs*) and filling them with the test code *exactly* as it appears in the book. You can then run the test suite (as described in Chapter 5) to watch it fail, then write the application code as described in the tutorial, and finally re-run the test suite to watch it pass.

Inexperienced programmers: The *Rails Tutorial* is not aimed principally at beginning programmers, and web applications, even relatively simple ones, are by their nature fairly complex. If you are completely new to web programming and find the *Rails Tutorial* too difficult, I suggest learning the basics of HTML and CSS and then giving the *Rails Tutorial* another go. (Unfortunately, I don't have a personal recommendation here, but *Head First HTML* looks promising, and one reader recommends *CSS: The Missing Manual* by David Sawyer McFarland.) You might also consider reading the first few chapters of

[2] http://tryruby.org/

[3] http://railsforzombies.org/

Beginning Ruby by Peter Cooper, which starts with sample applications much smaller than a full-blown web app. That said, a surprising number of beginners have used this tutorial to learn web development, so I suggest giving it a try, and I especially recommend the *Rails Tutorial* screencast series[4] to give you an "over-the-shoulder" look at Rails software development.

Experienced programmers new to web development: Your previous experience means you probably already understand ideas like classes, methods, data structures, etc., which is a big advantage. Be warned that if your background is in C/C++ or Java, you may find Ruby a bit of an odd duck, and it might take time to get used to it; just stick with it and eventually you'll be fine. (Ruby even lets you put semicolons at the ends of lines if you miss them too much.) The *Rails Tutorial* covers all the web-specific ideas you'll need, so don't worry if you don't currently know a `PUT` from a `POST`.

Experienced web developers new to Rails: You have a great head start, especially if you have used a dynamic language such as PHP or (even better) Python. The basics of what we cover will likely be familiar, but test-driven development may be new to you, as may be the structured REST style favored by Rails. Ruby has its own idiosyncrasies, so those will likely be new, too.

Experienced Ruby programmers: The set of Ruby programmers who don't know Rails is a small one nowadays, but if you are a member of this elite group you can fly through this book and then move on to *The Rails 3 Way* by Obie Fernandez.

Inexperienced Rails programmers: You've perhaps read some other tutorials and made a few small Rails apps yourself. Based on reader feedback, I'm confident that you can still get a lot out of this book. Among other things, the techniques here may be more up-to-date than the ones you picked up when you originally learned Rails.

[4] http://railstutorial.org/screencasts

Experienced Rails programmers: This book is unnecessary for you, but many experienced Rails developers have expressed surprise at how much they learned from this book, and you might enjoy seeing Rails from a different perspective.

After finishing the *Ruby on Rails Tutorial*, I recommend that experienced programmers read *The Well-Grounded Rubyist* by David A. Black, which is an excellent in-depth discussion of Ruby from the ground up, or *The Ruby Way* by Hal Fulton, which is also fairly advanced but takes a more topical approach. Then move on to *The Rails 3 Way* to deepen your Rails expertise.

At the end of this process, no matter where you started, you should be ready for the many more intermediate-to-advanced Rails resources out there. Here are some I particularly recommend:

- RailsCasts by Ryan Bates: Excellent (mostly) free Rails screencasts
- PeepCode: Excellent commercial screencasts
- Code School: Interactive programming courses
- Rails Guides: Good topical and up-to-date Rails references
- RailsCasts by Ryan Bates: Did I already mention RailsCasts? Seriously: *RailsCasts*.

1.1.2 "Scaling" Rails

Before moving on with the rest of the introduction, I'd like to take a moment to address the one issue that dogged the Rails framework the most in its early days: the supposed inability of Rails to "scale"—i.e., to handle large amounts of traffic. Part of this issue relied on a misconception; you scale a *site*, not a framework, and Rails, as awesome as it is, is only a framework. So the real question should have been, "Can a site built with Rails scale?" In any case, the question has now been definitively answered in the affirmative: some of the

most heavily trafficked sites in the world use Rails. Actually *doing* the scaling is beyond the scope of just Rails, but rest assured that if *your* application ever needs to handle the load of Hulu or the Yellow Pages, Rails won't stop you from taking over the world.

1.1.3 Conventions in this book

The conventions in this book are mostly self-explanatory. In this section, I'll mention some that may not be.

Both the HTML and PDF editions of this book are full of links, both to internal sections (such as Section 1.2) and to external sites (such as the main Ruby on Rails download page).[5]

Many examples in this book use command-line commands. For simplicity, all command line examples use a Unix-style command line prompt (a dollar sign), as follows:

```
$ echo "hello, world"
hello, world
```

Windows users should understand that their systems will use the analogous angle prompt `>`:

```
C:\Sites> echo "hello, world"
hello, world
```

On Unix systems, some commands should be executed with **sudo**, which stands for "substitute user do".[6] By default, a command executed with **sudo**

[5] When reading the *Rails Tutorial*, you may find it convenient to follow an internal section link to look at the reference and then immediately go back to where you were before. This is easy when reading the book as a web page, since you can just use the Back button of your browser, but both Adobe Reader and OS X's Preview allow you to do this with the PDF as well. In Reader, you can right-click on the document and select "Previous View" to go back. In Preview, use the Go menu: `Go > Back`.

[6] Many people erroneously believe that **sudo** stands for "superuser do" because it runs commands as the superuser (root) by default. In fact, **sudo** is a concatenation of the **su** command and the English word "do", and **su** stands for "substitute user", as you can verify by typing **man su** in your shell.

is run as an administrative user, which has access to files and directories that normal users can't touch, such as in this example from Section 1.2.2:

```
$ sudo ruby setup.rb
```

Most Unix/Linux/OS X systems require `sudo` by default, unless you are using Ruby Version Manager as suggested in Section 1.2.2; in this case, you would type this instead:

```
$ ruby setup.rb
```

Rails comes with lots of commands that can be run at the command line. For example, in Section 1.2.5 we'll run a local development web server as follows:

```
$ rails server
```

As with the command-line prompt, the *Rails Tutorial* uses the Unix convention for directory separators (i.e., a forward slash `/`). My Rails Tutorial sample application, for instance, lives in

```
/Users/mhartl/rails_projects/sample_app
```

On Windows, the analogous directory would be

```
C:\Sites\sample_app
```

The root directory for any given app is known as the *Rails root*, but this terminology is confusing and many people mistakenly believe that the "Rails root" is the root directory for Rails itself. For clarity, the *Rails Tutorial* will refer to the Rails root as the *application root*, and henceforth all directories will be relative to this directory. For example, the `config` directory of my sample application is