

Addison-Wesley Professional Ruby Series



RUBY ON RAILS 2.3 TUTORIAL

LEARN RAILS BY EXAMPLE

MICHAEL HARTL

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 publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Copyright © 2011 Michael Hartl

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671 3447

The source code in *Ruby on Rails 3 Tutorial* is released under the MIT License.

ISBN 13: 978-0-13-248802-0

ISBN 10: 0-13-248802-7

First release, October 2010

Editor-in-Chief

Mark Taub

Executive Acquisitions Editor

Debra Williams Cauley

Managing Editor

John Fuller

Project Editor

Elizabeth Ryan

Copy Editor

Erica Orloff

Indexer

Kathrin Unger

Proofreader

Claire Splan

Publishing Coordinator

Kim Boedigheimer

Cover Designer

Gary Adair

Compositor

Glyph International

Contents

Acknowledgments	xiii
About this Book	xv
About the Author	xvii

Chapter 1 From Zero to Deploy 1

1.1	Introduction	3
1.1.1	Comments for Various Readers	4
1.1.2	“Scaling” Rails	7
1.1.3	Conventions in This Book	7
1.2	Up and Running	9
1.2.1	Development Environments	9
1.2.2	Ruby, RubyGems, and Rails	12
1.2.3	The First Application	14
1.2.4	Model-View-Controller (MVC)	15
1.2.5	Script/Server	18
1.3	Version Control with Git	23
1.3.1	Installation and Setup	24
1.3.2	Adding and Committing	26
1.3.3	What Good Does Git Do You?	27
1.3.4	GitHub	28
1.3.5	Branch, Edit, Commit, Merge	30

- 1.4 Deploying 35
 - 1.4.1 Heroku Setup 35
 - 1.4.2 Heroku Deployment, Step One 36
 - 1.4.3 Heroku Deployment, Step Two 36
 - 1.4.4 Heroku Commands 37
- 1.5 Conclusion 39

Chapter 2 A Demo App 41

- 2.1 Planning the Application 41
 - 2.1.1 Modeling Users 43
 - 2.1.2 Modeling Microposts 43
- 2.2 The Users Resource 44
 - 2.2.1 A User Tour 45
 - 2.2.2 MVC in Action 47
 - 2.2.3 Weaknesses of This Users Resource 57
- 2.3 The Microposts Resource 57
 - 2.3.1 A Micropost Microtour 57
 - 2.3.2 Putting the *Micro* in Microposts 60
 - 2.3.3 A User `has_many` Microposts 62
 - 2.3.4 Inheritance Hierarchies 64
 - 2.3.5 Deploying the Demo App 66
- 2.4 Conclusion 67

Chapter 3 Mostly Static Pages 69

- 3.1 Static Pages 71
 - 3.1.1 Truly Static Pages 71
 - 3.1.2 Static Pages with Rails 74
- 3.2 Our First Tests 78
 - 3.2.1 Testing Tools 79
 - 3.2.2 TDD: Red, Green, Refactor 82
- 3.3 Slightly Dynamic Pages 93
 - 3.3.1 Testing a Title Change 93
 - 3.3.2 Passing Title Tests 96
 - 3.3.3 Instance Variables and Embedded Ruby 98
 - 3.3.4 Eliminating Duplication with Layouts 102
- 3.4 Conclusion 104
- 3.5 Exercises 105

Chapter 4 Rails-Flavored Ruby 107

- 4.1 Motivation 107
 - 4.1.1 A `title` Helper 107
 - 4.1.2 Cascading Style Sheets 110
- 4.2 Strings and Methods 112
 - 4.2.1 Comments 113
 - 4.2.2 Strings 113
 - 4.2.3 Objects and Message Passing 116
 - 4.2.4 Method Definitions 120
 - 4.2.5 Back to the `title` Helper 121
- 4.3 Other Data Structures 121
 - 4.3.1 Arrays and Ranges 122
 - 4.3.2 Blocks 125
 - 4.3.3 Hashes and Symbols 127
 - 4.3.4 CSS Revisited 130
- 4.4 Ruby Classes 132
 - 4.4.1 Constructors 132
 - 4.4.2 Class Inheritance 133
 - 4.4.3 Modifying Built-In Classes 137
 - 4.4.4 A Controller Class 138
 - 4.4.5 A User Class 141
- 4.5 Exercises 143

Chapter 5 Filling in the Layout 145

- 5.1 Adding Some Structure 145
 - 5.1.1 Site Navigation 146
 - 5.1.2 Custom CSS 151
 - 5.1.3 Partials 159
- 5.2 Layout Links 163
 - 5.2.1 Integration Tests 163
 - 5.2.2 Rails Routes 166
 - 5.2.3 Named Routes 169
- 5.3 User Signup: A First Step 171
 - 5.3.1 Users Controller 171
 - 5.3.2 Signup URL 175
- 5.4 Conclusion 177
- 5.5 Exercises 178

Chapter 6 Modeling and Viewing Users, Part I 181

- 6.1 User Model 182
 - 6.1.1 Database Migrations 184
 - 6.1.2 The Model File 188
 - 6.1.3 Creating User Objects 190
 - 6.1.4 Finding User Objects 194
 - 6.1.5 Updating User Objects 196
- 6.2 User Validations 197
 - 6.2.1 Validating Presence 197
 - 6.2.2 Length Validation 203
 - 6.2.3 Format Validation 205
 - 6.2.4 Uniqueness Validation 209
- 6.3 Viewing Users 213
 - 6.3.1 Debug and Rails Environments 213
 - 6.3.2 User Model, View, Controller 216
 - 6.3.3 A Users Resource 220
- 6.4 Conclusion 223
- 6.5 Exercises 223

Chapter 7 Modeling and Viewing Users, Part II 225

- 7.1 Insecure Passwords 225
 - 7.1.1 Password Validations 226
 - 7.1.2 A Password Migration 230
 - 7.1.3 An Active Record Callback 232
- 7.2 Secure Passwords 236
 - 7.2.1 A Secure Password Test 236
 - 7.2.2 Some Secure Password Theory 238
 - 7.2.3 Implementing `has_password?` 240
 - 7.2.4 An Authenticate Method 243
- 7.3 Better User Views 247
 - 7.3.1 Testing the User Show Page (With Factories) 248
 - 7.3.2 A Name and a Gravatar 252
 - 7.3.3 A User Sidebar 258
- 7.4 Conclusion 261
 - 7.4.1 Git Commit 261
 - 7.4.2 Heroku Deploy 262
- 7.5 Exercises 263

Chapter 8 Sign Up 265

- 8.1 Signup Form 265
 - 8.1.1 Using `form_for` 267
 - 8.1.2 The Form HTML 270
- 8.2 Signup Failure 273
 - 8.2.1 Testing Failure 273
 - 8.2.2 A Working Form 277
 - 8.2.3 Signup Error Messages 281
 - 8.2.4 Filtering Parameter Logging 283
- 8.3 Signup Success 285
 - 8.3.1 Testing Success 285
 - 8.3.2 The Finished Signup Form 287
 - 8.3.3 The Flash 288
 - 8.3.4 The First Signup 290
- 8.4 RSpec Integration Tests 292
 - 8.4.1 Webrat 293
 - 8.4.2 Users Signup Failure Should Not Make a New User 294
 - 8.4.3 Users Signup Success Should Make a New User 297
- 8.5 Conclusion 299
- 8.6 Exercises 300

Chapter 9 Sign In, Sign Out 303

- 9.1 Sessions 303
 - 9.1.1 Sessions Controller 304
 - 9.1.2 Signin Form 306
- 9.2 Signin Failure 310
 - 9.2.1 Reviewing Form Submission 310
 - 9.2.2 Failed Signin (Test and Code) 313
- 9.3 Signin Success 317
 - 9.3.1 The Completed `create` Action 318
 - 9.3.2 Remember Me 319
 - 9.3.3 Cookies 326
 - 9.3.4 Current User 327
- 9.4 Signing Out 334
 - 9.4.1 Destroying Sessions 334
 - 9.4.2 Signin Upon Signup 336
 - 9.4.3 Changing the Layout Links 338
 - 9.4.4 Signin/out Integration Tests 341

- 9.5 Conclusion 343
- 9.6 Exercises 343

Chapter 10 Updating, Showing, and Deleting Users 345

- 10.1 Updating Users 345
 - 10.1.1 Edit Form 346
 - 10.1.2 Enabling Edits 352
- 10.2 Protecting Pages 355
 - 10.2.1 Requiring Signed-In Users 355
 - 10.2.2 Requiring the Right User 359
 - 10.2.3 An Expectation Bonus 361
 - 10.2.4 Friendly Forwarding 362
- 10.3 Showing Users 364
 - 10.3.1 User Index 365
 - 10.3.2 Sample Users 369
 - 10.3.3 Pagination 371
 - 10.3.4 Partial Refactoring 378
- 10.4 Destroying Users 379
 - 10.4.1 Administrative Users 380
 - 10.4.2 The `destroy` Action 384
- 10.5 Conclusion 387
- 10.6 Exercises 388

Chapter 11 User Microposts 391

- 11.1 A Micropost Model 391
 - 11.1.1 The Basic Model 392
 - 11.1.2 User/Micropost Associations 395
 - 11.1.3 Micropost Refinements 399
 - 11.1.4 Micropost Validations 403
- 11.2 Showing Microposts 405
 - 11.2.1 Augmenting the User Show Page 405
 - 11.2.2 Sample Microposts 412
- 11.3 Manipulating Microposts 414
 - 11.3.1 Access Control 416
 - 11.3.2 Creating Microposts 419
 - 11.3.3 A Proto-Feed 424

- 11.3.4 Destroying Microposts 433
- 11.3.5 Testing the New Home Page 435
- 11.4 Conclusion 438
- 11.5 Exercises 438

Chapter 12 Following Users 441

- 12.1 The Relationship Model 442
 - 12.1.1 A Problem with the Data Model (and a Solution) 443
 - 12.1.2 User/Relationship Associations 449
 - 12.1.3 Validations 453
 - 12.1.4 Following 454
 - 12.1.5 Followers 459
 - 12.2 A Web Interface for Following and Followers 461
 - 12.2.1 Sample Following Data 462
 - 12.2.2 Stats and a Follow Form 463
 - 12.2.3 Following and Followers Pages 472
 - 12.2.4 A Working Follow Button the Standard Way 476
 - 12.2.5 A Working Follow Button with Ajax 480
 - 12.3 The Status Feed 485
 - 12.3.1 Motivation and Strategy 485
 - 12.3.2 A First Feed Implementation 489
 - 12.3.3 Scopes, Subselects, and a Lambda 491
 - 12.3.4 The New Status Feed 496
 - 12.4 Conclusion 497
 - 12.4.1 Extensions to the Sample Application 498
 - 12.4.2 Guide to Further Resources 500
 - 12.5 Exercises 501
- Index 503

This page intentionally left blank

CHAPTER 4

Rails-Flavored Ruby

Grounded in examples from Chapter 3, this chapter explores some elements of Ruby important for Rails. Ruby is a big language, but fortunately the subset needed to be productive as a Rails developer is relatively small. Moreover, this subset is *different* from the usual approaches to learning Ruby, which is why, if your goal is making dynamic web applications, I recommend learning Rails first, picking up bits of Ruby along the way. To be a Rails *expert*, you need to understand Ruby more deeply, and this book gives you a good foundation for developing that expertise. As noted in Section 1.1.1, after finishing *Rails Tutorial* I suggest reading a pure Ruby book such as *Beginning Ruby*, *The Well-Grounded Rubyist*, or *The Ruby Way*.

This chapter covers a lot of material, and it's okay not to get it all on the first pass. I'll refer back to it frequently in future chapters.

4.1 Motivation

As we saw in the last chapter, it's possible to develop the skeleton of a Rails application, and even start testing it, with essentially no knowledge of the underlying Ruby language. We did this by relying on the generated controller and test code and following the examples we saw there. This situation can't last forever, though, and we'll open this chapter with a couple of additions to the site that bring us face-to-face with our Ruby limitations.

4.1.1 A `title` Helper

When we last saw our new application, we had just updated our mostly static pages to use Rails layouts to eliminate duplication in our views (Listing 4.1).

Listing 4.1 The sample application site layout.
app/views/layouts/application.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Ruby on Rails Tutorial Sample App | <%= @title %></title>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

This layout works well, but there's one part that could use a little polish. Recall that the title line

```
Ruby on Rails Tutorial Sample App | <%= @title %>
```

relies on the definition of **@title** in the actions, such as

```
class PagesController < ApplicationController

  def home
    @title = "Home"
  end

  .
  .
  .
```

But what if we don't define an **@title** variable? It's a good convention to have a *base title* we use on every page, with an optional variable title if we want to be more specific. We've *almost* achieved that with our current layout, with one wrinkle: As you can see if you delete the **@title** assignment in one of the actions, in the absence of an **@title** variable the title appears as follows:

```
Ruby on Rails Tutorial Sample App |
```

In other words, there's a suitable base title, but there's also a trailing vertical bar character | at the end of the title.

One common way to handle this case is to define a *helper*, which is a function designed for use in views. Let's define a `title` helper that returns a base title, "Ruby on Rails Tutorial Sample App", if no `@title` variable is defined, and adds a vertical bar followed by the variable title if `@title` is defined (Listing 4.2).¹

Listing 4.2 Defining a `title` helper.
`app/helpers/application_helper.rb`

```
module ApplicationHelper

  # Return a title on a per-page basis.
  def title
    base_title = "Ruby on Rails Tutorial Sample App"
    if @title.nil?
      base_title
    else
      "#{base_title} | #{@title}"
    end
  end
end
```

This may look fairly simple to the eyes of an experienced Rails developer, but it's *full* of new Ruby ideas: modules, comments, local variable assignment, booleans, control flow, string interpolation, and return values. We'll cover each of these ideas in this chapter.

Now that we have a helper, we can use it to simplify our layout by replacing

```
<title>Ruby on Rails Tutorial Sample App | <%= @title %></title>
```

with

```
<title><%= title %></title>
```

1. If a helper is specific to a particular controller, you should put it in the corresponding helper file; for example, helpers for the Pages controller generally go in `app/helpers/pages_helper.rb`. In our case, we expect the `title` helper to be used on all the site's pages, and Rails has a special helper file for this case: `app/helpers/application_helper.rb`.

as seen in Listing 4.3. Note in particular the switch from the instance variable `@title` to the helper method `title` (without the `@` sign). Using Autotest or `spec spec/`, you should verify that the tests from Chapter 3 still pass.

Listing 4.3 The sample application site layout.
app/views/layouts/application.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title><%= title %></title>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

4.1.2 Cascading Style Sheets

There's a second addition to our site that seems simple but adds several new Ruby concepts: including style sheets into our site layout. Though this is a book in web development, not web design, we'll be using cascading style sheets (CSS) to give the sample application some minimal styling, and we'll use the Blueprint CSS framework as a foundation for that styling.

To get started, download the latest Blueprint CSS. (For simplicity, I'll assume you download Blueprint to a **Downloads** directory, but use whichever directory is most convenient.) Using either the command line or a graphical tool, copy the Blueprint CSS directory **blueprint** into the **public/stylesheets** directory, a special directory where Rails keeps stylesheets. On my Mac, the commands looked like this, but your details may differ:

```
$ cp -r ~/Downloads/joshuaclayton-blueprint-css-016c911/blueprint \
> public/stylesheets/
```

Here `cp` is the Unix copy command, and the `-r` flag copies recursively (needed for copying directories). (As mentioned briefly in Section 3.2.1, the tilde `~` means “home directory” in Unix.)

Once you have the stylesheets in the proper directory, Rails provides a helper for including them on our pages using Embedded Ruby (Listing 4.4).

Listing 4.4 Adding stylesheets to the sample application layout.
app/views/layouts/application.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title><%= title %></title>
    <%= stylesheet_link_tag 'blueprint/screen', :media => 'screen' %>
    <%= stylesheet_link_tag 'blueprint/print', :media => 'print' %>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

Let's focus on the new lines:

```
<%= stylesheet_link_tag 'blueprint/screen', :media => 'screen' %>
<%= stylesheet_link_tag 'blueprint/print', :media => 'print' %>
```

These use the built-in Rails helper **stylesheet_link_tag**, which you can read more about at the Rails API.² The first **stylesheet_link_tag** line includes the stylesheet **blueprint/screen.css** for screens (e.g., computer monitors), and the second includes **blueprint/print.css** for printing. (The helper automatically appends the **.css** extension to the filenames if absent, so I've left it off for brevity.) As with the **title** helper, to an experienced Rails developer these lines look simple, but there are at least four new Ruby ideas: built-in Rails methods, method invocation with missing parentheses, symbols, and hashes. In this chapter, we'll cover these new ideas as well. (We'll see the HTML produced by these stylesheets included in Listing 4.6 of Section 4.3.4.)

By the way, with the new stylesheets the site doesn't look much different from before, but it's a start (Figure 4.1). We'll build on this foundation starting in Chapter 5.³

2. I've linked my favorite Rails API source, RailsBrain. Its API version is currently a little out of date, but it doesn't usually matter much. If you think it might matter, you can use the official Rails API, but I prefer the RailsBrain interface. Incidentally, "API" stands for application programming interface.

3. If you're impatient, feel free to check out the Blueprint CSS Quickstart tutorial.

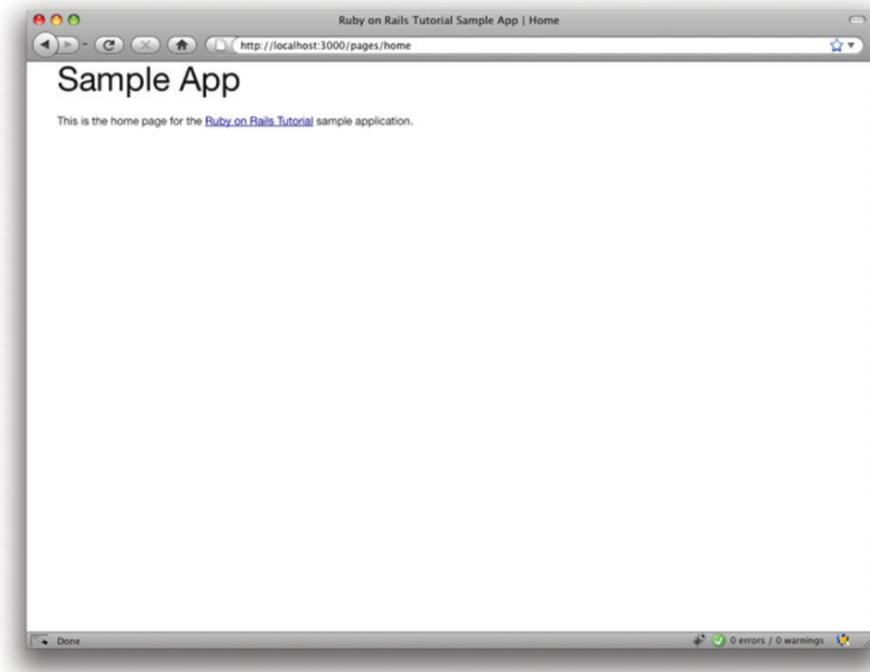


Figure 4.1 The Home page with the new Blueprint stylesheets.

4.2 Strings and Methods

Our principal tool for learning Ruby will be the *Rails console*, which is a command-line tool for interacting with Rails applications. The console itself is built on top of interactive Ruby (irb), and thus has access to the full power of Ruby. (As we'll see in Section 4.4.4, the console also has access to the Rails environment.) Start the console at the command line as follows:

```
$ script/console
Loading development environment (Rails 2.3.8)
>>
```

By default, the console starts in a *development environment*, which is one of three separate environments defined by Rails (the others are *test* and *production*). This distinction won't be important in this chapter; we'll learn more about environments in Section 6.3.1.

The console is a great learning tool, and you should feel free to explore—don't worry, you (probably) won't break anything. When using the console, type Ctrl-C if you get stuck, or Ctrl-D to exit the console altogether.

Throughout the rest of this chapter, you might find it helpful to consult the Ruby API.⁴ It's packed (perhaps even *too* packed) with information; for example, to learn more about Ruby strings you can look at the Ruby API entry for the **String** class.

4.2.1 Comments

Ruby *comments* start with the pound sign **#** and extend to the end of the line. Ruby (and hence Rails) ignores comments, but they are useful for human readers (including, often, the original author!). In the code

```
# Return a title on a per-page basis.  
def title  
.  
.  
.
```

the first line is a comment indicating the purpose of the subsequent function definition.

You don't ordinarily include comments in console sessions, but for instructional purposes I'll include some comments in what follows, like this:

```
>> 17 + 42 # Integer addition  
=> 59
```

If you follow along in this section typing or copying-and-pasting commands into your own console, you can of course omit the comments if you like; the console will ignore them in any case.

4.2.2 Strings

Strings are probably the most important data structure for web applications, since web pages ultimately consist of strings of characters sent from the server to the browser. Let's get started exploring strings with the console.

4. I've linked to RubyBrain, sister site to RailsBrain.

```
>> ""           # An empty string
=> ""
>> "foo"        # A nonempty string
=> "foo"
```

These are *string literals* (also, amusingly, called *literal strings*), created using the double quote character ". The console prints the result of evaluating each line, which in the case of a string literal is just the string itself.

We can also concatenate strings with the `+` operator:

```
>> "foo" + "bar" # String concatenation
=> "foobar"
```

Here is the result of evaluating "foo" plus "bar" in the string "foobar".⁵

Another way to build up strings is via *interpolation* using the special syntax `#{}:`⁶

```
>> first_name = "Michael" # Variable assignment
=> "Michael"
>> "#{first_name} Hartl" # Variable interpolation
=> "Michael Hartl"
```

Here we've *assigned* the value "Michael" to the variable `first_name` and then interpolated it into the string `"#{first_name} Hartl"`. We could also assign both strings a variable name:

```
>> first_name = "Michael"
=> "Michael"
>> last_name = "Hartl"
=> "Hartl"
>> first_name + " " + last_name # Concatenation, with a space in between
=> "Michael Hartl"
>> "#{first_name} #{last_name}" # The equivalent interpolation
=> "Michael Hartl"
```

5. For more on the origins of "foo" and "bar"—and, in particular, the possible *non*-relation of "foobar" to "FUBAR"—see the Jargon File entry on "foo".

6. Programmers familiar with Perl or PHP should compare this to the automatic interpolation of dollar sign variables in expressions like "foo \$bar".

Note that the final two expressions are equivalent, but I prefer the interpolated version; having to add the single space " " seems a bit awkward.

Printing

To *print* a string, the most commonly used Ruby function is **puts** (pronounced “put ess”, for “put string”):

```
>> puts "foo"      # put string
foo
=> nil
```

The **puts** method operates as a *side-effect*: the expression **puts "foo"** prints the string to the screen and then returns literally nothing: **nil** is a special Ruby value for “nothing at all”. (In what follows, I’ll sometimes suppress the **=> nil** part for simplicity.)

Using **puts** automatically appends a newline character `\n` to the output; the related **print** method does not:

```
>> print "foo"     # print string (same as puts, but without the newline)
foo=> nil
>> print "foo\n"  # Same as puts "foo"
foo
=> nil
```

Single-Quoted Strings

All the examples so far have used *double-quoted strings*, but Ruby also supports *single-quoted strings*. For many uses, the two types of strings are effectively identical:

```
>> 'foo'          # A single-quoted string
=> "foo"
>> 'foo' + 'bar'
=> "foobar"
```

There’s an important difference, though; Ruby won’t interpolate into single-quoted strings

```
>> '#{foo} bar'   # Single-quoted strings don't allow interpolation
=> "\#{foo} bar"
```

Note how the console returns values using double-quoted strings, which requires a backslash to *escape* characters like `#`.

If double-quoted strings can do everything that single-quoted strings can do, and interpolate to boot, what's the point of single-quoted strings? They are often useful because they are truly literal, and contain exactly the characters you type. For example, the “backslash” character is special on most systems, as in the literal newline `\n`. If you want a variable to contain a literal backslash, single quotes make it easier:

```
>> '\n'      # A literal backslash n
=> "\\n"
```

As with the `#` character in our previous example, Ruby needs to escape the backslash with an additional backslash; inside double-quoted strings, a literal backslash is represented with *two* backslashes.

For a small example like this, there's not much savings, but if there are lots of things to escape it can be a real help:

```
>> 'Newlines (\n) and tabs (\t) both use the backslash character \.'
```

```
=> "Newlines (\\n) and tabs (\\t) both use the backslash character \\."
```

4.2.3 Objects and Message Passing

Everything in Ruby, including strings and even `nil`, is an *object*. We'll see the technical meaning of this in Section 4.4.2, but I don't think anyone ever understood objects by reading the definition in a book; you have to build up your intuition for objects by seeing lots of examples.

It's easier to describe what objects *do*, which is respond to messages. An object like a string, for example, can respond to the message `length`, which returns the number of characters in the string:

```
>> "foobar".length      # Passing the "length" message to a string
=> 6
```

Typically, the messages that get passed to objects are *methods*, which are functions defined on those objects.⁷ Strings also respond to the `empty?` method:

```
>> "foobar".empty?
=> false
>> "".empty?
=> true
```

Note the question mark at the end of the `empty?` method. This is a Ruby convention indicating that the return value is *boolean*: `true` or `false`. Booleans are especially useful for *control flow*:

```
>> s = "foobar"
>> if s.empty?
>>   "The string is empty"
>> else
>>   "The string is nonempty"
>> end
=> "The string is nonempty"
```

Booleans can also be combined using the `&&` (“and”), `||` (“or”), and `!` (“not”) operators:

```
>> x = "foo"
=> "foo"
>> y = ""
=> ""
>> puts "Both strings are empty" if x.empty? && y.empty?
=> nil
>> puts "One of the strings is empty" if x.empty? || y.empty?
"One of the strings is empty"
=> nil
>> puts "x is not empty" if !x.empty?
=> "x is not empty"
```

7. Apologies in advance for switching haphazardly between *function* and *method* throughout this chapter; in Ruby, they’re the same thing: All methods are functions, and all functions are methods, because everything is an object.

Since everything in Ruby is an object, it follows that `nil` is an object, so it, too, can respond to methods. One example is the `to_s` method that can convert virtually any object to a string:

```
>> nil.to_s
""
```

This certainly appears to be an empty string, as we can verify by *chaining* the messages we pass to `nil`:

```
>> nil.empty?
NoMethodError: You have a nil object when you didn't expect it!
You might have expected an instance of Array.
The error occurred while evaluating nil.empty?
>> nil.to_s.empty?      # Message chaining
true
```

We see here that the `nil` object doesn't itself respond to the `empty?` method, but `nil.to_s` does.

There's a special method for testing for `nil`-ness, which you might be able to guess:

```
>> "foo".nil?
=> false
>> ".nil?
=> false
>> nil.nil?
=> true
```

If you look back at Listing 4.2, you'll see that the `title` helper tests to see if `@title` is `nil` using the `nil?` method. This is a hint that there's something special about instance variables (variables with an `@` sign), which can best be understood by contrasting them with ordinary variables. For example, suppose we enter `title` and `@title` variables at the console without defining them first:

```
>> title      # Oops! We haven't defined a title variable.
NameError: undefined local variable or method `title'
>> @title     # An instance variable in the console
```

```
=> nil
>> puts "There is no such instance variable." if @title.nil?
There is no such instance variable.
=> nil
>> "#{@title}" # Interpolating @title when it's nil
""
```

You can see from this example that Ruby complains if we try to evaluate an undefined local variable, but issues no such complaint for an instance variable; instead, instance variables are **nil** if not defined. This code also explains why the code

```
Ruby on Rails Tutorial Sample App | <%= @title %>
```

becomes

```
Ruby on Rails Tutorial Sample App |
```

when **@title** is **nil**: Embedded Ruby inserts the string corresponding to the given variable, and the string corresponding to **nil** is **""**.

The last example also shows an alternate use of the **if** keyword: Ruby allows you to write a statement that is evaluated only if the statement following **if** is true. There's a complementary **unless** keyword that works the same way:

```
>> string = "foobar"
>> puts "The string '#{string}' is nonempty." unless string.empty?
The string 'foobar' is nonempty.
=> nil
```

It's worth noting that the **nil** object is special, in that it is the *only* Ruby object that is false in a boolean context, apart from **false** itself:

```
>> if nil
>> true
>> else
>> false           # nil is false
>> end
=> false
```

In particular, all other Ruby objects are *true*, even 0:

```
>> if 0
>>   true      # 0 (and everything other than nil and false itself) is true
>> else
>>   false
>> end
=> true
```

4.2.4 Method Definitions

The console allows us to define methods the same way we did with the **home** action from Listing 3.5 or the **title** helper from Listing 4.2. (Defining methods in the console is a bit cumbersome, and ordinarily you would use a file, but it's convenient for demonstration purposes.) For example, let's define a function **string_message** that takes a single *argument* and returns a message based on whether the argument is empty or not:

```
>> def string_message(string)
>>   if string.empty?
>>     "It's an empty string!"
>>   else
>>     "The string is nonempty."
>>   end
>> end
=> nil
>> puts string_message("")
It's an empty string!
>> puts string_message("foobar")
The string is nonempty.
```

Note that Ruby functions have an *implicit return*, meaning they return the last statement evaluated—in this case, one of the two message strings, depending on whether the method's argument **string** is empty or not. Ruby also has an explicit return option; the following function is equivalent to the one above:

```
>> def string_message(string)
>>   return "It's an empty string!" if string.empty?
>>   return "The string is nonempty."
>> end
```

The alert reader might notice at this point that the second **return** here is actually unnecessary—being the last expression in the function, the string **"The string is nonempty."** will be returned regardless of the **return** keyword, but using **return** in both places has a pleasing symmetry to it.

4.2.5 Back to the **title** Helper

We are now in a position to understand the **title** helper from Listing 4.2:⁸

```

module ApplicationHelper
  # Return a title on a per-page basis.           # Documentation comment
  def title                                     # Method definition
    base_title = "Ruby on Rails Tutorial Sample App" # Variable assignment
    if @title.nil?                               # Boolean test for nil
      base_title                                  # Implicit return
    else
      "#{base_title} | #{@title}"                 # String interpolation
    end
  end
end

```

These elements—function definition, variable assignment, boolean tests, control flow, and string interpolation—come together to make a compact helper method for use in our site layout. The final element is **module ApplicationHelper**: code in Ruby modules can be *mixed in* to Ruby classes. When writing ordinary Ruby, you often write modules and include them explicitly yourself, but in this case Rails handles the inclusion automatically for us. The result is that the **title** method is automatically available in all our views.

4.3 Other Data Structures

Though web apps are ultimately about strings, actually *making* those strings requires using other data structures as well. In this section, we'll learn about some Ruby data structures important for writing Rails applications.

8. Well, there will still be *one* thing left that we don't understand, which is how Rails ties this all together: mapping URLs to actions, making the **title** helper available in views, etc. This is an interesting subject, and I encourage you to investigate it further, but knowing exactly *how* Rails works is not necessary to *using* Rails. (For a deeper understanding, I recommend *The Rails Way* by Obie Fernandez.)

4.3.1 Arrays and Ranges

An array is just a list of elements in a particular order. We haven't discussed arrays yet in *Rails Tutorial*, but understanding them gives a good foundation for understanding hashes (Section 4.3.3) and for aspects of Rails data modeling (such as the **has_many** association seen in Section 2.3.3 and covered more in Section 11.1.2).

So far we've spent a lot of time understanding strings, and there's a natural way to get from string to arrays using the **split** method:

```
>> "foo bar baz".split # Split a string into a three-element array
=> ["foo", "bar", "baz"]
```

The result of this operation is an array of three strings. By default, **split** divides a string into an array by splitting on whitespace, but you can split on nearly anything else:

```
>> "fooxbarxbaz".split('x')
=> ["foo", "bar", "baz"]
```

As is conventional in most computer languages, Ruby arrays are *zero-offset*, which means that the first element in the array has index 0, the second has index 1, and so on:

```
>> a = [42, 8, 17]
=> [42, 8, 17]
>> a[0] # Ruby uses square brackets for array access.
=> 42
>> a[1]
=> 8
>> a[2]
=> 17
>> a[-1] # Indices can even be negative!
=> 17
```

We see here that Ruby uses square brackets to access array elements. In addition to this bracket notation, Ruby offers synonyms for some commonly accessed elements:

```
>> a # Just a reminder of what 'a' is
=> [42, 8, 17]
>> a.first
```

```
=> 42
>> a.second
=> 8
>> a.last
=> 17
>> a.last == a[-1]    # Comparison using ==
=> true
```

This last line introduces the equality comparison operator `==`, which Ruby shares with many other languages, along with the associated `!=` (“not equal”), etc.:

```
>> x = a.length      # Like strings, arrays respond to the 'length' method.
=> 3
>> x == 3
=> true
>> x == 1
=> false
>> x != 1
=> true
>> x >= 1
=> true
>> x < 1
=> false
```

In addition to **length** (seen in the first line above), arrays respond to a wealth of other methods:

```
>> a.sort
=> [8, 17, 42]
>> a.reverse
=> [17, 8, 42]
>> a.shuffle
=> [17, 42, 8]
```

By the way, the **shuffle** method is available only on Ruby 1.8.7 or later; if you’re using 1.8.6, you can use

```
>> a.sort_by { rand }
```

instead.

You can also add to arrays with the “push” operator, `<<`:

```
>> a << 7                                # Pushing 7 onto an array
[42, 8, 17, 7]
>> a << "foo" << "bar"                   # Chaining array pushes
[42, 8, 17, 7, "foo", "bar"]
```

This last example shows that you can chain pushes together, and also that, unlike arrays in many other languages, Ruby arrays can contain a mixture of different types (in this case, integers and strings).

Before we saw `split` convert a string to an array. We can also go the other way with the `join` method:

```
>> a
[42, 8, 17, 7, "foo", "bar"]
>> a.join                                # Join on nothing
=> "428177foobar"
>> a.join(', ')                          # Join on comma-space
=> "42, 8, 17, 7, foo, bar"
```

Closely related to arrays are *ranges*, which can probably most easily be understood by converting them to arrays using the `to_a` method:

```
>> 0..9
=> 0..9
>> 0..9.to_a                             # Oops, call to_a on 9
ArgumentError: bad value for range
>> (0..9).to_a                           # Use parentheses to call to_a on the range
=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Though `0..9` is a valid range, the second expression above shows that we need to add parentheses to call a method on it.

Ranges are useful for pulling out array elements:

```
>> a = %w[foo bar baz quux]              # Use %w to make a string array
["foo", "bar", "baz", "quux"]
>> a[0..2]
=> ["foo", "bar", "baz"]
```

Ranges also work with characters:

```
>> ('a'..'e').to_a
=> ["a", "b", "c", "d", "e"]
```

4.3.2 Blocks

Both arrays and ranges respond to a host of methods that accept *blocks*, which are simultaneously one of Ruby's most powerful and most confusing features:

```
>> (1..5).each { |i| puts 2 * i }
2
4
6
8
10
=> 1..5
```

This code calls the **each** method on the range (**1..5**) and passes it the block `{ |i| puts 2 * i }`. The vertical bars around the variable name in `|i|` are Ruby syntax for a block variable, and it's up to the method to know what to do with the block; in this case, the range's **each** method can handle a block with a single local variable, which we've called **i**, and it just executes the block for each value in the range.

Curly braces are one way to indicate a block, but there is a second way as well:

```
>> (1..5).each do |i|
?>   puts 2 * i
>> end
2
4
6
8
10
=> 1..5
```

Blocks can be more than one line, and often are. In *Rails Tutorial* we'll follow the common convention of using curly braces only for short one-line blocks and the **do..end** syntax for longer one-liners and for multi-line blocks:

```
>> (1..5).each do |number|
?>   puts 2 * number
>>   puts '--'
>> end
2
--
4
--
6
--
8
--
10
--
=> 1..5
```

Here I've used **number** in place of **i** just to emphasize that any variable name will do.

Unless you already have a substantial programming background, there is no shortcut to understanding blocks; you just have to see them a lot, and eventually you'll get used to them.⁹ Luckily, humans are quite good at making generalizations from concrete examples; here are a few more, including a couple using the **map** method:

```
>> 3.times { puts "Betelgeuse!" } # 3.times takes a block with no variables.
"Betelgeuse!"
"Betelgeuse!"
"Betelgeuse!"
=> 3
>> (1..5).map { |i| i**2 } # The ** notation is for 'power'.
=> [1, 4, 9, 16, 25]
>> %w[a b c] # Recall that %w makes string arrays.
=> ["a", "b", "c"]
>> %w[a b c].map { |char| char.upcase }
=> ["A", "B", "C"]
```

As you can see, the **map** method returns the result of applying the given block to each element in the array or range.

9. Programming experts, on the other hand, might benefit from knowing that blocks are *closures*, which are one-shot anonymous functions with data attached.

By the way, we're now in a position to understand the line of Ruby I threw into Section 1.4.4 to generate random subdomains:¹⁰

```
('a'..'z').to_a.shuffle[0..7].join
```

Let's build it up step by step:

```
>> ('a'..'z').to_a                # An alphabet array
=> ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o",
    "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"]
>> ('a'..'z').to_a.shuffle        # Shuffle it!
=> ["c", "g", "l", "k", "h", "z", "s", "i", "n", "d", "y", "u", "t", "j", "q",
    "b", "r", "o", "f", "e", "w", "v", "m", "a", "x", "p"]
>> ('a'..'z').to_a.shuffle[0..7]  # Pull out the first eight elements.
=> ["f", "w", "i", "a", "h", "p", "c", "x"]
>> ('a'..'z').to_a.shuffle[0..7].join # Join them together to make one string.
=> "mznpybuj"
```

4.3.3 Hashes and Symbols

Hashes are essentially a generalization of arrays: You can think of hashes as basically like arrays, but not limited to integer indices. (In fact, some languages, especially Perl, call hashes *associative arrays* for this reason.) Instead, hash indices, or *keys*, can be almost any object. For example, we can use strings as keys:

```
>> user = {}                        # {} is an empty hash
=> {}
>> user["first_name"] = "Michael"  # Key "first_name", value "Michael"
=> "Michael"
>> user["last_name"] = "Hartl"     # Key "last_name", value "Hartl"
=> "Hartl"
>> user["first_name"]              # Element access is like arrays
=> "Michael"
>> user                            # A literal representation of the hash
=> {"last_name"=>"Hartl", "first_name"=>"Michael"}
```

10. Recall that `shuffle` isn't available in Ruby 1.8.6; you have to use `sort_by { rand }` instead.

Hashes are indicated with curly braces containing key-value pairs; a pair of braces with no key-value pairs—i.e., `{}`—is an empty hash. It's important to note that the curly braces for hashes have nothing to do with the curly braces for blocks. (Yes, this can be confusing.) Though hashes resemble arrays, one important difference is that hashes don't generally guarantee keeping their elements in a particular order.¹¹ If order matters, use an array.

Instead of defining hashes one item at a time using square brackets, it's easy to use their literal representation:

```
>> user = { "first_name" => "Michael", "last_name" => "Hartl" }
=> {"last_name"=>"Hartl", "first_name"=>"Michael"}
```

Here I've used the usual Ruby convention of putting an extra space at the two ends of the hash—a convention ignored by the console output. (Don't ask me why the spaces are conventional; probably some early influential Ruby programmer liked the look of the extra spaces, and the convention stuck.)

So far we've used strings as hash keys, but in Rails it is much more common to use *symbols* instead. Symbols look kind of like strings, but prefixed with a colon instead of surrounded by quotes. For example, `:name` is a symbol. You can think of symbols as basically strings without all the extra baggage:¹²

```
>> "name".length
4
>> :name.length
NoMethodError: undefined method `length' for :name:Symbol
>> "foobar".reverse
=> "raboof"
>> :foobar.reverse
NoMethodError: undefined method `reverse' for :foobar:Symbol
```

Symbols are a special Ruby data type shared with very few other languages, so they may seem weird at first, but Rails uses them a lot, so you'll get used to them fast.

11. Apparently Ruby 1.9 guarantees that hashes keep their elements in the same order entered, but it would be unwise ever to count on a particular ordering.

12. As a result of having less baggage, symbols are easier to compare to each other; strings need to be compared character by character, while symbols can be compared all in one go. This makes them ideal for use as hash keys.

In terms of symbols as hash keys, we can define a **user** hash as follows:

```
>> user = { :name => "Michael Hartl", :email => "michael@example.com" }
=> {:name=>"Michael Hartl", :email=>"michael@example.com"}
>> user[:name]           # Access the value corresponding to :name.
=> "Michael Hartl"
>> user[:password]      # Access the value of an undefined key.
=> nil
```

We see here from the last example that the hash value for an undefined key is simply **nil**. Hash values can be virtually anything, even other hashes, as seen in Listing 4.5.

Listing 4.5 Nested hashes

```
>> params = {}           # Define a hash called 'params' (short for 'parameters').
=> {}
>> params[:user] = { :name => "Michael Hartl", :email => "mhartl@example.com" }
=> {:name=>"Michael Hartl", :email=>"mhartl@example.com"}
>> params
=> {:user=>{:name=>"Michael Hartl", :email=>"mhartl@example.com"}}
>> params[:user][:email]
=> "mhartl@example.com"
```

These sorts of hashes-of-hashes, or *nested hashes*, are heavily used by Rails, as we'll see starting in Section 8.2.

As with arrays and ranges, hashes respond to the **each** method. For example, consider a hash named **flash** with keys for two conditions, **:success** and **:error**:

```
>> flash = { :success => "It worked!", :error => "It failed. :-( " }
=> {:success=>"It worked!", :error=>"It failed. :-( " }
>> flash.each do |key, value|
?> puts "Key #{key.inspect} has value #{value.inspect}"
>> end
Key :success has value "It worked!"
Key :error has value "It failed. :-( "
```

Note that, while the **each** method for arrays takes a block with only one variable, **each** for hashes takes two, a *key* and a *value*. Thus, the **each** method for a hash iterates through the hash one key-value *pair* at a time.

The last example uses the useful **inspect** method, which returns a string with a literal representation of the object it's called on:

```
>> puts flash                # Put the flash hash as a string (with ugly results).
successIt worked!errorIt failed. :-(
>> puts flash.inspect        # Put the flash hash as a pretty string
{:success=>"It worked!", :error=>"It failed. :-(")
>> puts :name, :name.inspect
name
:name
>> puts "It worked!", "It worked!".inspect
It worked!
"It worked!"
```

By the way, using **inspect** to print an object is common enough that there's a shortcut for it, the **p** function:

```
>> p flash                    # Same as 'puts flash.inspect'
{:success=>"It worked!", :error=>"It failed. :-(")
```

4.3.4 CSS Revisited

It's time now to revisit the lines from Listing 4.4 used in the layout to include the cascading style sheets:

```
<%= stylesheet_link_tag 'blueprint/screen', :media => 'screen' %>
<%= stylesheet_link_tag 'blueprint/print', :media => 'print' %>
```

We are now nearly in a position to understand this. As mentioned briefly in Section 4.1.2, Rails defines a special function to include stylesheets, and

```
stylesheet_link_tag 'blueprint/screen', :media => 'screen'
```

is a call to this function. But there are two mysteries. First, where are the parentheses? In Ruby, they are optional; these two lines are equivalent:

```
# Parentheses on function calls are optional.
stylesheet_link_tag('blueprint/screen', :media => 'screen')
stylesheet_link_tag 'blueprint/screen', :media => 'screen'
```

Second, the **:media** argument sure looks like a hash, but where are the curly braces? When hashes are the *last* argument in a function call, the curly braces are optional; these two lines are equivalent:

```
# Curly braces on final hash arguments are optional.
stylesheet_link_tag 'blueprint/screen', { :media => 'screen' }
stylesheet_link_tag 'blueprint/screen', :media => 'screen'
```

So, we see now that each of the lines

```
<%= stylesheet_link_tag 'blueprint/screen', :media => 'screen' %>
<%= stylesheet_link_tag 'blueprint/print', :media => 'print' %>
```

calls the **stylesheet_link_tag** function with two arguments: a string, indicating the path to the stylesheet, and a hash, indicating the media type (**'screen'** for the computer screen and **'print'** for a printed version). Because of the `<%= %>` brackets, the results are inserted into the template by ERb, and if you view the source of the page in your browser you should see the HTML needed to include a stylesheet (Listing 4.6).¹³

Listing 4.6 The HTML source produced by the CSS includes

```
<link href="/stylesheets/blueprint/screen.css" media="screen" rel="stylesheet"
type="text/css" />
<link href="/stylesheets/blueprint/print.css" media="print" rel="stylesheet"
type="text/css" />
```

13. You may see some funky numbers, like **?1257465942**, after the CSS filenames. These are inserted by Rails to ensure that browsers reload the CSS when it changes on the server.

4.4 Ruby Classes

We've said before that everything in Ruby is an object, and in this section we'll finally get to define some of our own. Ruby, like many object-oriented languages, uses *classes* to organize methods; these classes are then *instantiated* to create objects. If you're new to object-oriented programming, this may sound like gibberish, so let's look at some concrete examples.

4.4.1 Constructors

We've seen lots of examples of using classes to instantiate objects, but we have yet to do so explicitly. For example, we instantiated a string using the double quote characters, which is a *literal constructor* for strings:

```
>> s = "foobar"           # A literal constructor for strings using double quotes
=> "foobar"
>> s.class
=> String
```

We see here that strings respond to the method `class`, and simply return the class they belong to.

Instead of using a literal constructor, we can use the equivalent *named constructor*, which involves calling the `new` method on the class name:

```
>> s = String.new("foobar") # A named constructor for a string
=> "foobar"
>> s.class
=> String
>> s == "foobar"
=> true
```

This is equivalent to the literal constructor, but it's more explicit about what we're doing.

Arrays work the same way as strings:

```
>> a = Array.new([1, 3, 2])
=> [1, 3, 2]
```

Hashes, in contrast, are different. While the array constructor **Array.new** takes an initial value for the array, **Hash.new** takes a *default* value for the hash, which is the value of the hash for a nonexistent key:

```
>> h = Hash.new
=> {}
>> h[:foo]           # Try to access the value for the nonexistent key :foo.
=> nil
>> h = Hash.new(0)   # Arrange for nonexistent keys to return 0 instead of nil.
=> {}
>> h[:foo]
=> 0
```

4.4.2 Class Inheritance

When learning about classes, it's useful to find out the *class hierarchy* using the **super-class** method:

```
>> s = String.new("foobar")
=> "foobar"
>> s.class           # Find the class of s.
=> String
>> s.class.superclass # Find the superclass of String.
=> Object
>> s.class.superclass.superclass # Find the superclass of Object (it's nil!).
=> nil
```

A diagram of this inheritance hierarchy appears in Figure 4.2. We see here that the superclass of **String** is **Object**, but **Object** has no superclass. This pattern is true of every Ruby object: Trace back the class hierarchy far enough and every class in Ruby ultimately inherits from **Object**, which has no superclass itself. This is the technical meaning of “everything in Ruby is an object.”

To understand classes a little more deeply, there's no substitute for making one of our own. Let's make a **Word** class with a **palindrome?** method that returns **true** if the word is the same spelled forward and backward:

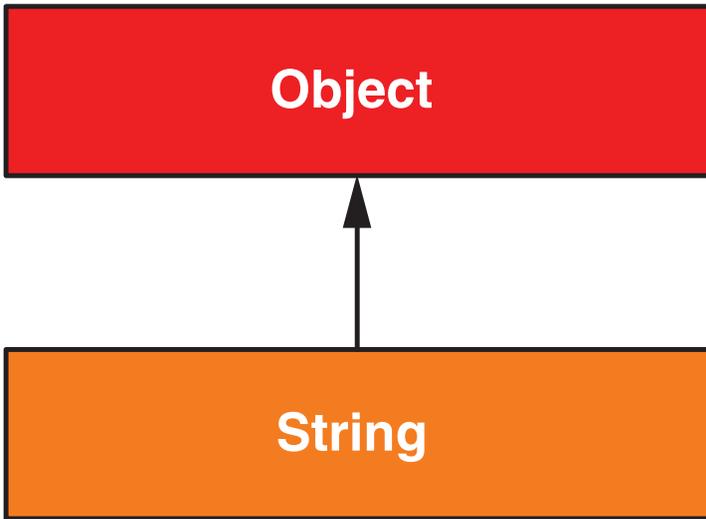


Figure 4.2 The inheritance hierarchy for the **String** class.

```
>> class Word
>>   def palindrome?(string)
>>     string == string.reverse
>>   end
>> end
=> nil
```

We can use it as follows:

```
>> w = Word.new           # Make a new Word object
=> #<Word:0x22d0b20>
>> w.palindrome?("foobar")
=> false
>> w.palindrome?("level")
=> true
```

If this example strikes you as a bit contrived, good; this is by design. It's odd to create a new class just to create a method that takes a string as an argument. Since a word *is* a string, it's more natural to have our **Word** class *inherit* from **String**, as seen in Listing 4.7. (You should exit the console and re-enter it to clear out the old definition of **Word**.)

Listing 4.7 Defining a **Word** class in **irb**

```
>> class Word < String           # Word inherits from String.
>>   # Return true if the string is its own reverse.
>>   def palindrome?
>>     self == self.reverse      # self is the string itself.
>>   end
>> end
=> nil
```

Here **Word** < **String** is the Ruby syntax for inheritance (discussed briefly in Section 3.1.2), which ensures that, in addition to the new **palindrome?** method, words also have all the same methods as strings:

```
>> s = Word.new("level")        # Make a new Word, initialized with "level".
=> "level"
>> s.palindrome?                # Words have the palindrome? method.
=> true
>> s.length                     # Words also inherit all the normal string methods.
=> 5
```

Since the **Word** class inherits from **String**, we can use the console to see the class hierarchy explicitly:

```
>> s.class
=> Word
>> s.class.superclass
=> String
>> s.class.superclass.superclass
=> Object
```

This hierarchy is illustrated in Figure 4.3.

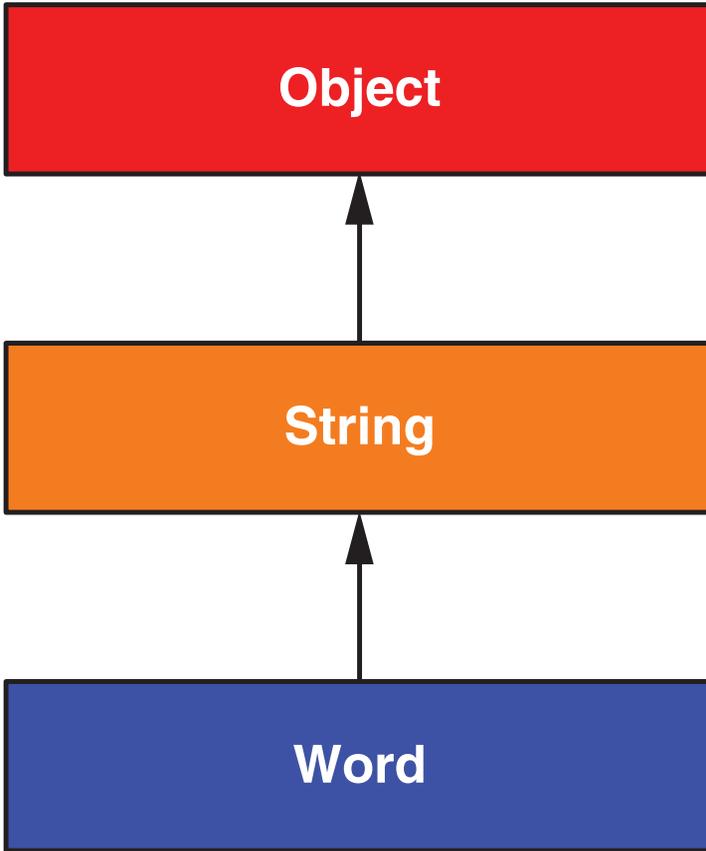


Figure 4.3 The inheritance hierarchy for the (non-built-in) `Word` class from Listing 4.7.

In Listing 4.7, note that checking that the word is its own reverse involves accessing the word inside the `word` class. Ruby allows us to do this using the `self` keyword: Inside the `word` class, `self` is the object itself, which means we can use

```
self == self.reverse
```

to check if the word is a palindrome.¹⁴

4.4.3 Modifying Built-In Classes

While inheritance is a powerful idea, in the case of palindromes it might be even more natural to add the `palindrome?` method to the `String` class itself, so that (among other things) we can call `palindrome?` on a string literal, which we currently can't do:

```
>> "level".palindrome?
NoMethodError: undefined method `palindrome?' for "level":String
```

Somewhat amazingly, Ruby lets you do just this; Ruby classes can be *opened* and modified, allowing ordinary mortals such as ourselves to add methods to them:¹⁵

```
>> class String
>>   # Return true if the string is its own reverse.
>>   def palindrome?
>>     self == self.reverse
>>   end
>> end
=> nil
>> "deified".palindrome?
=> true
```

(I don't know which is cooler: that Ruby lets you add methods to built-in classes, or that `"deified"` is a palindrome.)

14. For more on Ruby classes and the `self` keyword, see the RailsTips post on Class and Instance Variables in Ruby.

15. For those familiar with JavaScript, this functionality is comparable to using a built-in class prototype object to augment the class. (Thanks to reader Erik Eldridge for pointing this out.)

Modifying built-in classes is a powerful technique, but with great power comes great responsibility, and it's considered bad form to add methods to built-in classes without having a *really* good reason for doing so. Rails does have some good reasons; for example, in web applications we often want to prevent variables from being *blank*—e.g., a user's name should be something other than spaces and other whitespace—so Rails adds a **blank?** method to Ruby. Since the Rails console automatically includes the Rails extensions, we can see an example here (this won't work in plain **irb**):

```
>> "".blank?
=> true
>> "   ".empty?
=> false
>> "   ".blank?
=> true
>> nil.blank?
=> true
```

We see that a string of spaces is not *empty*, but it is *blank*. Note also that **nil** is blank; since **nil** isn't a string, this is a hint that Rails actually adds **blank?** to **String**'s base class, which (as we saw at the beginning of this section) is **Object** itself. We'll see some other examples of Rails additions to Ruby classes in Section 9.3.3.

4.4.4 A Controller Class

All this talk about classes and inheritance may have triggered a flash of recognition, because we have seen both before, in the Pages controller (Listing 3.16):

```
class PagesController < ApplicationController

  def home
    @title = "Home"
  end

  def contact
    @title = "Contact"
  end

  def about
    @title = "About"
  end

end
```

You're now in a position to appreciate, at least vaguely, what this code means: **PagesController** is a class that inherits from **ApplicationController**, and comes equipped with **home**, **contact**, and **about** methods, each of which defines the instance variable **@title**. Since each Rails console session loads the local Rails environment, we can even create a controller explicitly and examine its class hierarchy:¹⁶

```
>> controller = PagesController.new
=> #<PagesController:0x22855d0>
>> controller.class
=> PagesController
>> controller.class.superclass
=> ApplicationController
>> controller.class.superclass.superclass
=> ActionController::Base
>> controller.class.superclass.superclass.superclass
=> Object
```

A diagram of this hierarchy appears in Figure 4.4.

We can even call the controller actions inside the console, which are just methods:

```
>> controller.home
=> "Home"
```

This return value of **"Home"** comes from the assignment **@title = "Home"** in the **home** action.

But wait—actions don't have return values, at least not ones that matter. The point of the **home** action, as we saw in Chapter 3, is to render a web page. And I sure don't remember ever calling **PagesController.new** anywhere. What's going on?

What's going on is that Rails is *written in* Ruby, but Rails isn't Ruby. Some Rails classes are used like ordinary Ruby objects, but some are just grist for Rails' magic mill. Rails is *sui generis*, and should be studied and understood separately from Ruby. This is why, if your principal programming interest is writing web applications, I recommend learning Rails first, then learning Ruby, then looping back to Rails.

16. You don't have to know what each class in this hierarchy does. I don't know what they all do, and I've been programming in Ruby on Rails since 2005. This means either that (a) I'm grossly incompetent or (b) you can be a skilled Rails developer without knowing all its innards. I hope for both our sakes that it's the latter. :-)

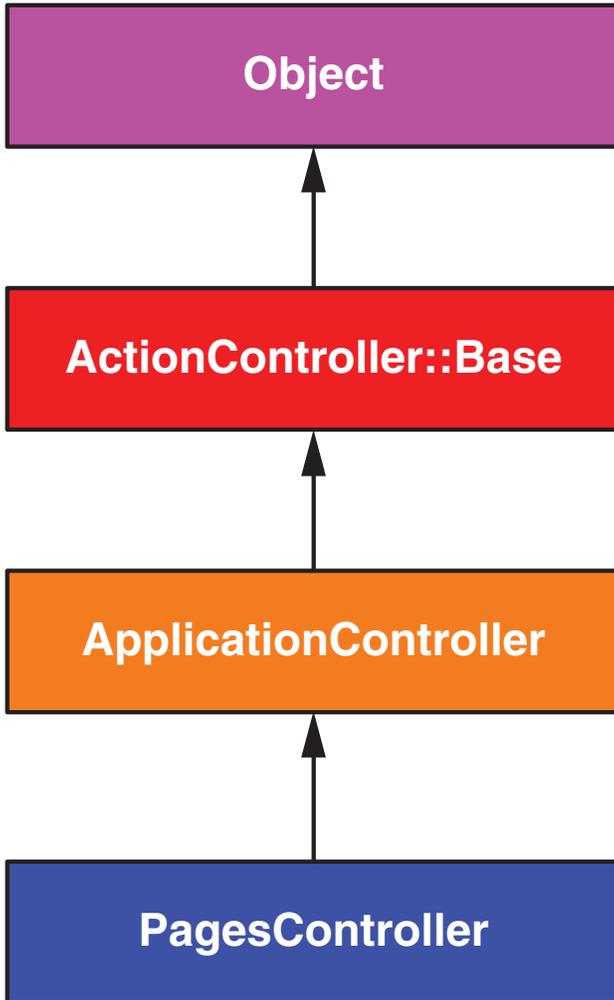


Figure 4.4 The inheritance hierarchy for the Pages controller.

4.4.5 A User Class

We end our tour of Ruby with a complete class of our own, a **User** class that anticipates the User model coming up in Chapter 6.

So far we’ve entered class definitions at the console, but this quickly becomes tiresome; instead, create the file **example_user.rb** in your Rails root directory and fill it with the contents of Listing 4.8. (Recall from Section 1.1.3 that the Rails root is the root of your *application* directory; for example, the Rails root for my sample application is `/Users/mhartl/rails_projects/sample_app`.)

Listing 4.8 Code for an example user.

example_user.rb

```
class User
  attr_accessor :name, :email

  def initialize(attributes = {})
    @name = attributes[:name]
    @email = attributes[:email]
  end

  def formatted_email
    "#{@name} <#{@email}>"
  end
end
```

There’s quite a bit going on here, so let’s take it step by step. The first line,

```
attr_accessor :name, :email
```

creates *attribute accessors* corresponding to a user’s name and email address. This creates “getter” and “setter” methods that allow us to retrieve (get) and assign (set) **@name** and **@email** instance variables.

The first method, **initialize**, is special in Ruby: It’s the method called when we execute **User.new**. This particular **initialize** takes one argument, **attributes**:

```
def initialize(attributes = {})
  @name = attributes[:name]
  @email = attributes[:email]
end
```

Here the `attributes` variable has a *default value* equal to the empty hash, so that we can define a user with no name or email address (recall from Section 4.3.3 that hashes return `nil` for nonexistent keys, so `attributes[:name]` will be `nil` if there is no `:name` key, and similarly for `attributes[:email]`).

Finally, our class defines a method called `formatted_email` that uses the values of the assigned `@name` and `@email` variables to build up a nicely formatted version of the user's email address using string interpolation (Section 4.2.2):

```
def formatted_email
  "#{@name} <#{@email}>"
end
```

Let's fire up the console, `require` the example user code, and take our `User` class out for a spin:

```
>> require 'example_user'      # This is how you load the example_user code.
=> ["User"]
>> example = User.new
=> #<User:0x224ceec @email=nil, @name=nil>
>> example.name                # nil since attributes[:name] is nil
=> nil
>> example.name = "Example User" # Assign a non-nil name
=> "Example User"
>> example.email = "user@example.com" # and a non-nil email address
=> "user@example.com"
>> example.formatted_email
=> "Example User <user@example.com>"
```

This code creates an empty example user and then fills in the name and email address by assigning directly to the corresponding attributes (assignments made possible by the `attr_accessor` line in Listing 4.8). When we write

```
example.name = "Example User"
```

Ruby is setting the `@name` variable to `"Example User"` (and similarly for the `email` attribute), which we then use in the `formatted_email` method.

Recalling from Section 4.3.4 that we can omit the curly braces for final hash arguments, we can create another user by passing a hash to the `initialize` method to create a user with pre-defined attributes:

```
>> user = User.new(:name => "Michael Hartl", :email => "mhartl@example.com")
=> #<User:0x225167c @email="mhartl@example.com", @name="Michael Hartl">
>> user.formatted_email
=> "Michael Hartl <mhartl@example.com>"
```

We will see starting in Chapter 8 that initializing objects using a hash argument is common in Rails applications.

4.5 Exercises

1. Using Listing 4.9 as a guide, combine the **split**, **shuffle**, and **join** methods to write a function that shuffles the letters in a given string.
2. Using Listing 4.10 as a guide, add a **shuffle** method to the **String** class.
3. Create three hashes called **person1**, **person2**, and **person3**, with first and last names under the keys **:first** and **:last**. Then create a **params** hash so that **params[:father]** is **person1**, **params[:mother]** is **person2**, and **params[:child]** is **person3**. Verify that, for example, **params[:father][:first]** has the right value.
4. Learn about the **Hash** method **merge**.

Listing 4.9 Skeleton for a string shuffle function

```
>> def string_shuffle(s)
>>   s.split('').??.?
>> end
=> nil
>> string_shuffle("foobar")
```

Listing 4.10 Skeleton for a **shuffle** method attached to the **String** class

```
>> class String
>>   def shuffle
>>     self.split('').??.?
>>   end
>> end
=> nil
>> "foobar".shuffle
```

This page intentionally left blank

Index

f indicates figure, t indicates table, and b indicates box.

!, 117
#{}, 114
&&, 117
@. *See* Instance variables
== (equality comparison operator),
 123
=~ (equals-tilde operator), 174
!= (not equal), 123
<<: (push operator), 124, 377
<%= ... %>, 100
[] (array elements), 122
{ } (block variable), 125
“ ” (double quotes), 114
{ } (hash), 127–128
| |, 332b–333b
 block variable, 125
 boolean combination, 117
@ (instance variables), 54–55, 99
+ operator, 114
!: operator (not), 333
(pound sign), 113
<%= @title %>, 100, 108
/ URL, 164t, 165
<%= yield %>, 102–103

A

a (anchor tag), 97
about action, 82
About page
 with embedded Ruby title, 101
 full HTML structure, 97
/about URL, 164, 164t
About view, with HTML structure removed,
 103–104
about_path definition, changing, 163, 168
about_URL, 168
Access control, 416–419
 authenticate method in Microposts
 controller actions, adding, 418
 authenticate method in Sessions helper,
 moving, 417–418
 Microposts controller, tests for, 418
Accessible attributes, 190
ActionController::Base, 65–66, 65f
ActionController::Routing::Routes.draw do
 [map], 167
Actions. *See also specific actions*
 Rails, 74
 views and, 77

- Actions, user
 - destroy, 46–47, 54f
 - edit, 46
 - new, 46, 48f
 - show, 46
 - with updated information, 52f
- action=“users,” 273
- Active Record, 54, 183
- Active record callback, 232–236
- ActiveRecord::Base, 64
- ActiveRecord::RecordNotFound, 195
- add_column method, 232, 232f
- add_index method, 212
- admin attribute
 - boolean, migration to add to users, 381–382, 382f
 - tests for, 380–381
- :admin=> :true, 386
- admin user, sample data populator
 - code with, 383
- Advantages, of Ruby on Rails, 3–4
- Ajax requests, Relationship controller
 - responses to
 - code, 483–484
 - tests, 481–482
- Ajax, working follow button using, 480–485
- :alt, 149
- alt attribute, 149–150
- Anchor tag (a), 97
- annotate command, 189–190
- annotate-models gem, 189
- API, REST, 500
- app/, 16t
- app/ director, 16
- ApplicationController, 65–66, 138–139
- ApplicationController class with
 - inheritance, 66
- application.html.erb, 102
- app/models/user.rb
 - User.authenticate method, 246–247
 - validating email format with regular expression, 207t, 208f
- Array elements ([]), 122
- Array.new, 133
- Arrays, 122–124
 - associative, 127
 - constructors for, 132–133
 - following, of users, 446–447
 - joining, 123
 - length method on, 123
 - push operator on, 124
 - ranges on, 124
 - shuffle method on, 123
 - split method on, 122
 - splitting of, 122
 - strings to, 122
- Associations. *See also specific associations*
 - micropost, 62–63, 64f
 - user/micropost, 394–399, 394f, 399t
- Associative arrays, 127
- @attr, 394
 - in initialization hash, 199–200
 - in password validation test, 227–228
- @attr hash, in test for signup failure, 275
- attr_accessible attributes
 - revisiting, 383–384
 - in user model, 383
- attr_accessible, for adding passwords and
 - password confirmation, 230
- attr_accessible method
 - for name and email access, 190
 - for name and email presence, 203
 - for name presence, 198
 - use of, 198
 - for user microposts, 393–394
- attr_accessor, 141–142
- attr_accessor keyword, 184, 228
- attr_accessor :password, 229

Attribute accessors. *See also specific types*
 for name and email, 141–142
 attributes, 141–142. *See also specific attributes*
 @attr.merge
 to make new user, 200–202
 for password validation, 227–228
 attr.merge techniques, 228
 authenticate method, 313–314, 357
 code, 246–247
 defining, alternative, 246
 exercises, 263–264
 Microposts controller actions, adding to, 418
 for secure passwords, 243–247
 Sessions helper, moving into, 417–418
 tests for, 244
 writing, 247

Authentication, 225
 adding, before filter, 357
 first tests for, 356–357
 for signed-in user, tests, 359
 of users, 237

Authlogic, 181b

Automated testing, 69

.autotest additions
 for OS X, 165
 for Ubuntu Linux, 166

Autotest installation, 80–82

autotest-fsevent gem, 81

autotest-growl gem, 81–82

Avatar, 255

B

Bates, Rain, 2

Before filter, 357

before filter :authenticate, 357

before(:each), 200

before_save callback

to create encrypted_password, 233–234
 to create encrypted_password attribute,
 233–234

before_save method, 233–234

Behavior-driven development (BDD), 78

belongs_to association, 395, 395f, 398,
 452

adding to Relationship model, 453
 user/relationships, testing, 452

be_valid, 202

blank? method, 138

for encrypted password attribute, 233
 for name, 199
 use of, 199

Blocks, 125–127

Blueprint CSS, 110–111, 112f

downloading, 110
 fixing IE idiosyncrasies with, 148
 home page with, 111, 112f

blueprint/print.css, 111

blueprint/screen.css, 111

Body, custom.css for, 152–155, 154f

Body section, 73

Books, resource, 501

Booleans, 117

Bracket notation, 122

Branches, Git, 30–31, 30–32

creating new, 74
 master, 30

Brand new user model, code for, 189

Browsers, 11, 11f

build method, 404

Built-in classes, modifying, 137–138

Button

follow (*See Follow button*)
 linking to signup page, 176, 177f
 signup, 156–157, 157f

C

Callback, active record, 232–236

callback method, 232–233

CanCan, 181b

Capybara, 293

- Cascading style sheets (CSS), 110–112
 - Blueprint (*See* Blueprint CSS)
 - custom (custom.css), 151–159 (*See also* Custom cascading style sheets (custom.css))
 - embedded Ruby helper for, 111
 - revisited, 130–131
 - stylesheet_link_tag, 111, 130
 - for styling user show page, including sidebar, 260, 261f
- :case_sensitive, 210
- change method, 297
- Characters, ranges on, 125
- checkout command, 28
- Chrome, 11
- :class, 149
- class << self, 246
- Class hierarchy, 133
- Class inheritance, 133–137, 134f, 136f
- class method, 132, 244–247
- Class, user, 141–143
 - example, code, 141
 - with inheritance, 64
- Class, with inheritance, 49
- Classes, 132–143. *See also specific classes*
 - built-in classes, modifying, 137–138
 - class inheritance, 133–137, 134f, 136f
 - constructors, 132–133
 - controller, 138–139, 140f
 - defining, 76–77
 - HTML elements, assigning to, 149
 - vs.* ids, 149
 - instantiation of, 132
 - use of, 76
 - user, 141–143
- Classes, Ruby, 132–143
 - built-in classes, modifying, 137–138
 - class inheritance, 133–137, 134f, 136f
 - constructors, 132–133
 - controller, 138–139, 140f
 - user, 141–143
- Clearance, 181b–182b
- cleartext, 238
- click_link method, 437
- click_link :Sign out”, 342
- Closures, 126
- Code. *See also specific topics*
 - generated, 2
- :collection, 429, 464
- Colors, custom, on home page, 152–155, 154f
- Columns, 184. *See also specific types*
 - add_column method, 232, 232f
 - created_at magic column, 186, 393
 - create_table magic column, 186
 - magic, 186
 - remove_column method, 232
 - updated_at magic column, 186, 393
- Command lines, 9–11, 10f
- Commands. *See specific applications and commands*
- Commenting out validating, to ensure failing test, 199
- Comments, Ruby, 113
- commit command, Git, 26–27
- Commit, Git, 32–33
- Concatenation, of strings, 114
- :conditions, 428
- config/, 16t
- Config directory, 8, 166–167, 166f
- config.gem, 249
- config.gems method, 372
- config/routes.rb for URL mapping, 166
- Confirm passwords, 226
- Console, 10, 63
- Console, Rails, 112–113
- Constant, 206

- Constructors, 132–133
 - literal, 132
 - named, 132
- contact action, 77
- Contact page
 - with embedded Ruby title, 101
 - with full HTML structure, 97
 - generated view for, 77
- /contact URL, 164t
- Contact view, with HTML structure removed, 103
- Container, custom.css for, 152–155
- container div, 149
- content, 44f
- content attribute, 392
 - accessible, 393–394
- content field, 43
- Control flow, booleans in, 117
- Control, generating user, 172
- Controller, 16, 17f. *See also specific controllers*
 - definition of, 74
 - inheritance hierarchies for, 65–66, 65f
 - making, script for, 75
 - Rails actions in, 74
- Controller action, 48, 55f
- Controller class, 138–139, 140f
- Controller, users, 171–175, 218–220, 219f
 - following and followers actions, adding, 464
 - generating, 171–172
 - with show action, 218
 - signup page, testing, 172–173
 - signup page title, testing, 173–175
 - title for new user page, setting custom, 175
- controller variable, for signing user in, 328–329
- Conventions, 7–9
- Cookies, 326–327, 326f
 - in sessions, 304, 320b–321b
 - signin, 326–327
- correct_user method, 360
- count method, 296, 409
- cp, 110
- create action, 51. *See also specific topics*
 - for Ajax request responses, 482
 - completed Sessions controller, 318–319
 - in failed user signup, test for, 275–276
 - for form submission, 274–275
 - params hash in, 310–313
 - RESTful route, 56t, 222t
 - Sessions controller, 310f
 - Sessions controller, preliminary version, 310–311
 - Sessions, with friendly forwarding, 364
 - signin form, 304
 - for signup failure (but not success), 278
 - in signup success, test for, 286
 - user, with save and redirect, 287
- create action, Microposts controller
 - adding, 418–419
 - adding to Home page, 422–423
 - code, 422
 - tests, 420–421
 - vs.* user analogue, 421
 - using, 419–424, 419f
- Create following relationship, Ruby JavaScript (RJS), 484
- create method, 209
- create! method, 209
- created_at magic column, 186, 393
- create_table magic column, 186
- create_table method, 186
- Creating app repository at GitHub, 69, 70f
- Creating microposts, 419–424, 419f. *See also*
 - create action, Microposts controller
 - form partial for, 423
 - micropost instance variable in home action, adding, 424
 - partial for user info sidebar, 423–424
- Creating user objects, 190–194, 191f

- Cross-site scripting attack, 254
- CSS. *See* Cascading style sheets (CSS)
- curl, 86b
- Current user, 327–334
- current users controller spec code, 247–248
- current-user
 - code, 330
 - defining assignment to, 330
 - definition, tempting but less useful, 330–331
 - finding, with `remember_token`, 331
- `current_user`, 328
- `current-user=` method, 330
- `current_user?` method, 360
- `current_user_helper` method, 350
- Custom cascading style sheets (`custom.css`), 148, 151–159
 - for container, body, and links, 152–155, 154f
 - for error message styling, 282–283, 282f
 - for filling in layouts, 154f, 156f–158f
 - for footer, 161–163
 - for links, 152–155, 154f
 - for navigation, 155–156, 156f
 - for round corners, 158–159, 158f
 - for signup button, 156–157
 - for signup button, big, green & clickable, 156–157, 157f
 - for signup form, 269
 - for site navigation link, 148
 - for user index, 368
- Custom title, setting, for new user page, 175
- Cygwin, 10, 11
- D**
- Data model, 42–43, 43f
- Data model, micropost, 393f
- Data populator code, with `adminuser`, 383
- Data, sample, adding microposts to, 413
- Database. *See also specific databases*
 - migrating up, 186–187
 - SQLite, user row in, 217, 217f
 - in user model, 183
- Database indices, 213b
- Database migrations, 21, 23, 45, 184–188
 - generating user model in, 184–185
 - migration in, 185–186
 - rake `db:migrate` in, 186–187, 187f
 - `self.down` method in, 187
 - `self.up` method in, 186–187
 - tables in, 184–185
- `db/`, 16t
- `db/development.sqlite3`, 187, 188f
- `db:populate`, 370
- `db:push:`, 67
- Debug and rails environments, 213–216, 214f
- Debug information, adding to site layout, 214, 214f
- `debug` method, 213–214
- Declaration, 72, 73
- `def` keyword, 76
- `:default => false`, 382
- Default Rails files, 15
- `default_scope` model, 400–401
- DELETE
 - HTTP, 85b–86b
 - RESTful route, 56t
- Delete links, user, viewable only by admins, 384
- Deleting users. *See* Destroying users
- Demo app, 41–68
 - microposts resource, 57–67 (*See also* Microposts resource)
 - strengths of, 67
 - users resource, 44–57 (*See also* Resource, users)
 - weaknesses of, 67–68

- Demo app, planning, 41–43, 42f
 - data model, 42–43
 - modeling microposts, 43, 44f
 - modeling users, 43, 43f
- deny_access method
 - for friendly forwarding, 363
 - for user authentication, 358
- Dependent: destroy, 402–403
- Deploying Rails, 35–38
 - Heroku commands, 37–38, 38f
 - Heroku setup, 35–36
 - Heroku step one, 36
 - Heroku step two, 36–37, 37f
 - value of, 35
- describe, 200
- destroy action, 51, 194, 384–387, 385f
 - for Ajax request responses, 482
 - to delete sessions, 334–336
 - method chaining in, 387
 - Microposts, adding, 418–419
 - Microposts controller, code, 435
 - Microposts controller, tests, 434–435
 - putting filter before restricting to admins, 386–387
 - RESTful route, 56t, 222t
 - signin form, 304
- Destroy following relationship, Ruby JavaScript (RJS), 484
- destroy user, 46–47, 54f
- Destroyed microposts
 - with user destroyed, code, 403
 - with user destroyed, testing, 402–403
- Destroying microposts, 433–435, 433f
 - Microposts controller destroy action, code, 435
 - Microposts controller destroy action, tests, 434–435
 - mockup of proto-feed with delete links, 433f
 - partial for sowing single micropost, 433–434
- Destroying user objects, 194
- Destroying users, 379–387
 - administrative users, 380–384, 382f
 - destroy action, 384–387, 385f
 - mockup for, 379, 380f
 - tests for, 385–386
- Development environment, 9–12, 112
 - debug, 215–216
 - IDEs, 9
 - text editors and command lines, 9–11, 10f
- Development log, 191
 - filter password and password_confirmation from log, code, 284
 - with passwords filtered, 284
 - with passwords visible, 283–284, 284
 - tailing, 191, 191f
- Devise, 181b
- Directory, Rails projects, 14
- Directory structure
 - default Rails, 16t
 - new Rails project, 14–15, 15f
- div tag, 149
 - error explanation, 295–296, 295f
- do keyword, 268
- doc/, 16t
- Doctype (document type), 73
 - declaration, 72
- do..end, 125–126
- Domain logic, 16
- Domain-specific language (DSL), 79, 84
- Double-quoted strings, 114–115
- draw method, 167
- Duplication, eliminating with layouts, 102–104
- Dynamic pages, slightly, 93–104
 - eliminating duplication with layouts, 102–104
 - features, 93
 - instance variables and embedded Ruby, 98–101

Dynamic pages, slightly (*continued*)
 title change, testing, 93–95, 93t
 title test, passing, 92f, 96–98, 98f

E

E Text Editor, 10

each method, 125, 129, 205

edit action, 46t

Git, 32

RESTful route, 56t, 222t

edit action, user

incomplete, 348

tests for, 347

user edit view, 348–349

edit form, 346–351

edit action, incomplete user, 348

edit action, user, tests for, 347

hidden input field, 350

HTML, 350

mockup, 346, 346f

settings, editing, 350, 351f

settings link, adding, 349–350, 349f

user edit view, 348–349, 349f

Edit page, 46

correct_user before filter to protect, 360

user, 51f

edit user, 46, 46t, 51f, 52f

Editors, text, 9–11, 10f

Edits, enabling, 352–355, 352f

edit_user_path route, 350

edit_user_path(@user), 259t

edit_user_url(@user), 259t

Emacs, 10

email, 43f

Email addresses

duplicate, test for rejection of, 209

uniqueness of, migration for enforcing, 212

uniqueness of, validating, 210

uniqueness of, validating, ignoring
 case, 211

email attribute, 43, 43f

making accessible, 190

presence of, test for, 203

presence of, validating, 203

Email format validation

with regular expression, 206–208, 207t,
 208f

tests for, 205–206

Email regex, 206–208, 207t, 208f

Embedded Ruby (*ERb*), 99–101

empty? method, 117, 233, 408

en, 96

Enabling edits, 352–355, 352f

encrypt method, 234, 241, 242

Encrypted password, 228, 230

before_save callback in creation of,
 233–234

existence of, testing for, 230–231

migration to add column to users table,
 212

users table, adding to, 232f

encrypted_password attribute, 230

before_save callback for creation of,
 233–234

nonempty, testing that, 233

encrypted_password column, adding to users
 table, 231–232, 232f

encrypt_password method, 233–236,
 323–324

Engine Yard, 35

Engine Yard Cloud, 35

Environment. *See specific type*

environment.rb, 167

Equality comparison operator (==), 123

Equals-tilde operator (=), 174

ERb, 99–101

Error explanation div, 295–296, 295f

Error messages

500 Internal Server Error, 18–19, 20f, 21f
 autotest-fsevent, 0.1.3, 81

- CSS styling, 282–283, 282f
 - display in signup form, adding, 281
 - error installing sqlite3-ruby, 20
 - failed micropost creation, 62, 62f
 - home page with form errors, 426f
 - signup, 281–283, 282f
 - unrecognized option ‘-v,’ 14
 - warning: CRLF will be replaced by LF in
 - .gitignore, 26
 - errorExplanation, 295–296, 295f
 - errors.full_messages, 281
 - errors.messages method, 282–283, 282f
 - errors.messages object, 281–282
 - Example user, code, 141
 - example_user.rb, 141
 - Exception, 195
 - Expectation error, 277
 - Expectation, message, 276–277, 277b
 - expire 20.years.from.now, 321, 326
 - Explicit return, 120
 - Extensions, 498–500
 - follower notifications, 499
 - messaging, 499
 - password reminders, 499
 - replies, 498–499
 - REST API, 500
 - RSS feed, 499
 - search, 500
 - signup confirmation, 499
- F**
- f flag, 191
 - f (form), 268
 - factories
 - to simulate user model objects, 249–250
 - use of, 248
 - for user show page, testing, 248–252
 - Factory Girl (gem), 249
 - to build new users, 276
 - defining sequence in, 375
 - including in test environment file, 249
 - simulating uses with, 374
 - Failed micropost creation, 62, 62f
 - Failed signin (test and code), 313–316
 - Failed user signup, tests, 275–276
 - Failing test. *See also specific topics*
 - commenting out a validating to ensure, 199
 - Faker gem, 369, 413
 - False, Ruby objects, 119
 - feed method, 424–426
 - @feed_items instance variable, 431–432
 - fill_in :user_name, 298–299
 - Filter(ing)
 - before, 357
 - adding authenticate before, 357
 - parameter logging, 283–284
 - find, 194–195
 - find_by_email, 195
 - find_by_remember_token method, 323, 331–332
 - Finding user objects, 194–195
 - Firebug, 11, 11f
 - Firebug Lite, 11
 - Firefox, 11, 11f
 - Firefox HTML Validator, 11
 - First application, rails command for, 14–15
 - First signup, success, 290–291, 291f, 292f
 - First-test development. *See* TDD (test-driven development)
 - First-time repository setup, 24–25
 - First-time system setup, 24
 - first_user.microposts, 63
 - Fixnum, 327
 - fixtures, 248
 - flash (flash message), 288–290
 - adding contents of, to site layout, 288–289
 - adding to user signup, 290
 - failed signin attempt, 315, 316b, 316f
 - vs.* flash.now, 316b
 - on successful user signup, test, 289–290

- Flash dat bow, 316b
- flash hash, 129, 288–290
- flash[:error], 289
 - message, 315, 316f
- flash.now, 289, 315, 316b
- flash[:success], 289–290
- Follow button, user profile with, 444f, 471f
- Follow button, working, standard way,
 - 476–480, 477f, 479f
 - Relationships controller action, tests,
 - 477–479
 - Relationships controller, code, 479–480
- Follow button, working, using Ajax, 480–485
 - advantages, 480
 - form for following user, 481
 - form for unfollowing user, 481
 - implementation, 480
 - including Prototype JavaScript Library in site layout, 482–483
 - Relationship controller responses to Ajax requests, code, 483–484
 - Relationship controller responses to Ajax requests, tests, 481–482
 - Ruby JavaScript (RJS) to create following relationship, 484
 - Ruby JavaScript (RJS) to destroy following relationship, 484
- Follow form
 - adding to user profile page, 470–471, 471f, 472f
 - web interface, 468–472
- follow! utility method, 456–457, 477–479
- followed_id, 444, 446, 447f
 - adding indices on columns for, 447–448
 - relationship, making accessible, 449
- Follower notifications, 499
- Follower stats
 - adding to Home page, 468
 - adding to user profile page, 470–471, 471f, 472f
 - partial for displaying, 466–468, 469f
- follower_id, 444, 447–448
- followers action
 - adding to Users controller, 464
 - code, 475–476
 - RESTful route, 465t
 - test, 473–475
- Followers, Relationship model, 459–461, 459f
 - reverse relationships, implement
 - user.followers using, 459–460
 - reverse relationships table, 459, 459f
 - reverse relationships, testing, 459–460
- Followers, show_follow used view to render, 476
- Following
 - origin of, 444
 - show_follow used view to render, 476
- following action
 - adding to Users controller, 464
 - code, 475–476
 - RESTful route, 465t
 - test, 473–475
- Following and followers pages, 472–476
 - followings and followers actions, code, 475
 - followings and followers actions, test, 473–475
 - mockup, followers, 472, 474f
 - mockup, following, 472, 473f
 - show_follow view used to render following and followers, 476
- following? boolean method, 455–456
- following? method, 456
- Following relationship
 - Ruby JavaScript (RJS) to create, 484
 - Ruby JavaScript (RJS) to destroy, 484
- Following, Relationship model, 454–458
 - following? and follow! utility methods, 456–457
 - following utility methods, tests, 456
 - unfollow! method, code, 458
 - unfollow! method, test, 457–458

- User model following association in
 - has_many :through, adding, 455
 - user.following attribute, test, 454–455
- Following table, 444, 447f
- Following user form, using Ajax, 481
- Following users, 441–497. *See also specific topics exercises*, 501–502
 - Home page with follow stats, 469f
 - Home page with working status feed, 496f
 - merging, 497
 - Relationship model, 442–461
 - showing current user’s followers, 479f
 - showing users being followed by current user, 477f
 - user profile with follow button, 471f
 - user profile with unfollow button, 472f
 - web interface for following and followers, 461–485 (*See also* Web interface for following and followers)
- Following users, mockups
 - finding user to follow, 443f
 - followers page, 474f
 - Home page with status feed and incremented following counter, 446f
 - profile of another user, with follow button, 444f
 - profile with unfollow button and incremented followers count, 445f
 - starting page, 442f
 - stats partial, 464f
 - user following page, 473f
 - user’s Home page with status feed, 486f
- Following users, status feed, 485–497
 - first feed implementation, 489–491
 - lambda, 495–496
 - motivation and strategy, 485–489, 486f
 - new status feed, 496–497, 496f
 - scopes, 491–494
 - subselects, 494–495
- following! utility method, 455–456
- following? utility method, 456–457
- following utility methods, tests for, 456
- Following/follower relationship, adding to sample data, 462–463
- Following/follower statistics on Home page, testing, 465–466
- Follow/unfollow form, partial for, 468
- Footer, site
 - add CSS for, 161–163
 - partial, 160–161
 - partial, layout with, 161, 162f
 - partial, with links, 170, 171f
 - _footer.html.erb, 160–161
- Foreign key, 451
- Forgery, thwarting, 273
- Form, 267–268
 - create action for submission of, 274–275
 - edit, 346–351 (*See also* edit form)
 - follow/unfollow, partial for, 468
- Form, follow
 - adding to user profile page, 470–471, 471f, 472f
 - web interface, 468–472
- Form, following user
 - standard, 470
 - using Ajax, 481
- Form, signin, 306–309
 - after trying to access protected page, 358–359, 358f
 - code, 308
 - create action for, 304
 - example of, 309f
 - HTML, 308–309
 - mockup, 306, 307f
- Form, signup, 265–273
 - constructing, 265–273 (*See also* Signup form)
 - current state of, 265–266, 266f
 - error message display, adding, 281, 282f
 - finished, 287
 - form_for helper method for, 267–269
 - HTML, 270–273, 270f, 272f

- Form, signup (*continued*)
 - mockup of, 267f
 - new users page, tests for, 266–267
 - text *vs.* password fields in, HTML, 272, 272f
 - user form_for, 270f
- Form tag, 273
- Form, unfollowing user, Ajax, 481
- Format validation, 205–208
- formatted_email method, 142
- form_for helper, 307–308
- form_for method, 267–269, 470
- form_for, @user attribute field fill-in, 280
- form_for(@user), 348, 351
- Friendly forwarding, 362–366
 - code to implement, 363
 - integration test for, 362–363
 - Sessions create action with, 364
- from_users_followed_by, 495–496
- Functions, 76
- G**
- gedit, 10
- gems, 12
- gems manifest, 372
- generate, 2b
- generate command, 185
- generate Micropost, 57
- generate rspec_controller script, 82–83
- generate rspec_model, 392
- generate script, 75
- Generated code, 2
- GET
 - HTTP, 85b–86b
 - RESTful, 56t
- get function, in integration tests, 164
- Git, 23–35
 - adding and committing, 26–27
 - branch, 30–32, 74
 - commit, 32–33
 - edit, 32
 - GitHub, 28–30, 29f–31f
 - merge, 33–34
 - push, 34, 35f
 - value of, 23, 27–28
- git add ., 26
- git branch, 30–32
- git branch command, 31
- git checkout -b modify-README, 31, 31f
- git checkout command, 28
- git checkout, with -b flag, 30
- git commit, 26–27, 262
- git config, 24
- git init, 24
- Git, installation and setup, 24–25
 - first-time repository, 24–25
 - first-time system, 24
 - ignore log files, 25
- git log, 27
- git push, 27
- git push heroku, 262
- git status, 26
- GitHub, 28–30, 29f–31f
- .gitignore, 25, 69
- GMate plugins, 10
- Gravatar
 - exercises on, 263–264
 - in user views, 252–258, 256f, 257f
- gravatar method, 257
- gravatar_for method, 255–258
 - definition of, from gravatar plugin source code, 258
 - use of, 255–256, 256f
 - workings of, 257
- Green, 90–92
 - adding action to page controller in, 90–92, 91f, 92f
 - definition of, 82
- Growl, 81
- gVim, 10

H

- h method, 254–255
- h2 tag, 252
- Haml, 100
- Hansson, David Heinemeier, 2, 3
- Hash, user, 129
- Hashes, 127–128. *See also specific hashes*
 - constructors for, 133
 - curly braces on, 131
 - initialization, 192
 - nested, 129, 312
 - nil (undefined key), 129
- Hash.new, 133
- has_many, 62–63
- has_many association (relationship), 395f
 - functions, 442–443
 - user/micropost associations, 395, 395f, 398
 - user/relationships, implementing, 451–452
- has_many method, 403
- has_many :relationships, 446, 447f
- has_many :through, 443, 447, 455
- has_password? method, 237
 - implementing, 240–243
 - tests for, 237–238
 - for users, 237
 - to write authenticate method, 243
- have_tag method, 93–94, 174, 252–253, 292
- have_tag(“div.pagination”), 378
- Head section, 72, 73
- Header, site partial, 160
 - layout with, 159–160
 - with links, 170
- /help URL, 164t
- Heroku, 35–38
 - commands, 37–38, 38f
 - interface, 37, 38f
 - setup, 35–36
 - step one, 36
 - step two, 36–37, 37f
 - subdomains, 38
- heroku rename, 37–38, 37f
- History, 3
- home action, 77, 139
 - feed instance variable, adding, 429
 - inside pages controller, 75
 - micropost instance variable, adding, 424
 - with paginated feed, 497
- Home page
 - Blueprint CSS, 111, 112f
 - with custom colors, 152–155, 154f
 - with embedded Ruby title, 100
 - with follow stats, 469f
 - follower stats, adding, 468
 - following/follower statistics, testing, 465–466
 - with form errors, error messages, 426f
 - with form for creating microposts, 419f
 - with full HTML structure, 96, 98f
 - generated view for, 77
 - with link to signup page, 151, 152, 152f
 - with logo image, but no custom CSS, 151, 152f
 - microposts, adding, 422–423
 - with navigation styling, 155–156, 156f
 - new, testing, 435–437, 436f
 - proto-feed, micropost (*See* Proto-feed, micropost)
 - raw view of, 75, 76f
 - route mapping for, 168
 - status feed, adding, 430–431
 - with status feed, and incremented following counter, 446f
 - with status feed, mockup, 485, 486f
 - with status feed, user’s, 486f
 - with status feed, working, 496f
- Home view, with HTML structure removed, 103
- home.html.erb, 77, 103
- href (hypertext reference), 97
- HTML-escaping, 254–255

HTML file with friendly greeting, static page,
72–73, 73f

html_escape method, 254–255

HTTP basic operations, 85b–86b

HTTP request method, 51

http://localhost:3000
default page, 18–20, 19f, 20f
default page with SQLite gem, 21–22,
22f

http://localhost:3000/index.html, 71

I

id, 43f
assigning to HTML elements, 149
vs. classes, 149

id attribute, 43, 43f

id column, migration and, 187

id field, 43

IDEs (integrated development environments), 9

if keyword, alternate use of, 119

if-else, in working signup form, 277–278

image_tag helper, 149

img, 149

Implicit return, 120

include? method, 426–427

Index
database, 213
user, 365–369, 369f
users table, add to, 212

:index, 367

index action, 46t, 364
example of, 47–48
Microposts, 418–419
paginating users in, 374
RESTful route, 56t, 222t
simplified user, 53–54
user, 367

Index page, 46, 47f
user, tests, 365–367

Index, user, 365–369, 369f
action, 367
code, 367–368
custom style sheet, 368
fully functional, 369, 369f
fully refactored, 379
layout link, 368
mockup, 364, 365f
with pagination, 373, 375f, 376f
tests, 365–367
view, 56, 367–368

Index view, refactoring
completion, 379
first attempt, 378

Index view, user
code, 367–368
MVC, 56

index.html file, 71, 72f

Information, user, stub view for
showing, 218

Inheritance
class, 133–137, 134f, 136f (*See also* Class inheritance)
class with, 49
functionality, 54
Word < String for, 135

Inheritance hierarchies, 64–66, 65f
for controllers, 65–66, 65f
for models, 64–65, 65f
for (non-built-in) word class, 135,
136f
for Pages controller, 139, 140f
for string class, 133, 134f

Initial user spec, 199–200

Initialization hash, 192

initialize method, 141–142

Insecure passwords. *See* Password(s), insecure

Inspect element, 11

inspect method, 130

Installation

- of Autotest, 80–82
 - of Git, 24–25
 - of Rails, 13–14
 - of Rspec, 79–80
 - of Ruby, 12
 - of RubyGems, 12–13
 - of sqlite3 and libsqlite3-dev, 20
- Instance variable nilness, 118–119
- Instance variables, 54–56, 99
- Instantiated classes, 132
- Integer identifier, 43, 43f
- integrate_views method, 95, 173, 248, 292
- Integration tests, 163–166, 164t
- in Autotest, 165–166
 - .autotest additions for OS X, 165
 - .autotest additions for Ubuntu Linux, 166
 - for routes, 163–165
 - for signing in and out, 341–342
 - for signup link, 175–176
- Integration tests, for sign up
- RSp, 292–299
 - users signup failure should not make a new user, 294–297
 - users signup success should make a new user, 297–299
 - Webrat, 293–294
- Interface, public, 236
- Internet Explorer (IE), fixing idiosyncrasies in, 148
- Interpolation, 114
- iTerm, 10, 10f
- J**
- JavaScript Library, 481
- JavaScript Library, Prototype
- including in site layout, 482–483
 - use of, 481
- javascript.include_tag :defaults, 482–483
- join method, 124

K

- Kate, 10
- Katz, Yehuda, 3
- Komodo Edit Sublime Text editor, 10

L

- Lambda
- for status feed, 495–496
 - testing for signup failure with, 296–297, 299
- Layout, 145–179. *See also specific topics*
- with added structure, 147–148, 147f
 - cascading style sheets in, 130–131
 - commit and merge, 177–178
 - debug information in layout, adding, 214, 214f
 - for duplication elimination, 102–104
 - exercises on, 178–179
 - with footer partial, 161, 162f
 - Prototype JavaScript Library in, adding, 482–483
 - sample, 110
 - sample application, 102–103, 108
 - with stylesheet and header partials, 159–160
 - stylesheets in sample application, adding, 111
- Layout, adding structure, 145–163
- custom CSS, 151–159, 154f, 156f–158f
 - new branch, making, 146
 - overview, 145
 - partials, 159–163, 162f
 - site navigation, 146–151, 147f, 148f
- Layout links, 163–171
- about_path definition, changing, 163
 - custom.css for, 152–156, 156f
 - integration tests, 163–166, 164t (*See also* Integration tests)
 - named routes, 163, 164t, 169–171, 171f
 - rails routes, 166–169, 166f

- Layout links (*continued*)
 - signed-in users, changing, 339
 - signin/signout links on, tests for, 338–339
 - signup status, changing, 338–341, 341f
 - user index, 368
- Layout, partial
 - for site footer, 160–161
 - for site header, 160
- Layout, user signup, 171–177
 - signup URL, 175–177, 177f
 - users controller, 171–175
- Length
 - of microposts, constraining, 61
 - of name attribute, 204
 - validation of, 61–62, 62f, 203–205
- Length method, on arrays, 123
- li (list item tag), 150
- lib/, 16t
- libsqlite3-dev, installing, 20
- :limit option, 412–413
- Links
 - custom.css for, 152–155, 154f
 - layout (*See* Layout links)
 - partial footer with, 170, 171f
 - partial header with, 170
 - URL mapping for, 163, 164t
- Links, delete
 - mockup of proto-feed with, 433f
 - user, viewable only by admins, 384
- link_to function, filling in second arguments, 169
- link_to helper, 150–151
- link_to method, for site footer partials, 160–161
- link_to named routes, 169–171
 - about page at /about, 170, 171f
 - footer partial with links, 170
 - header partial with links, 169–170
- List item (li), 150
- List, user, 46, 46t, 47f
- Literal constructor, 132
- Literal strings, 114
- localhost, 18
- log, 27
- log/, 16t
- log/*.log, 25
- Logo
 - downloading and installing, 151
 - helper for, 340
- Lorem.sentence method, 413
- ls command, Unix, 28
- M**
- m flag, 27
- MacVim, 10
- macvim, 10
- Magic columns, 186
- Make utility, 23
- make_relationships method, 462–463
- Manipulating microposts, 414–437
 - access control, 416–419
 - controllers, 415
 - creating, 419–424, 419f (*See also* create action, Microposts controller; Creating microposts)
 - destroying, 433–435, 433f (*See also* Destroying microposts)
 - home page, testing new, 435–437, 436f
 - Microposts resource, 414–415
 - Microposts resource routes, 416
 - proto-feed, 424–432 (*See also* Proto-feed, micropost)
 - RESTful routes, 415, 417t
 - status feed, 415
- map method, 126
- map.about, 167
- map.resources method, 305–306
- map.resources :microposts routing rule, 58, 58t
- map.resources :users, added to routes.rb, 273–274

- map.signup rule, 176
- maps.resources, 464
- Mass assignment, 190
- Master branch, 30
- :maximum, 204, 229
- :media argument, 130–131
- :member, 464
- Merb, 3
- merge, 200–202
- Merge, Git, 33–34
- Message expectation, 276–277, 277b
- Message passing, 116–120
- Messaging, 499
- method chaining, 387
- :method=> :delete, 384
- method=“post,” 273
- Methods, 76. *See also specific methods*
 - defining, 120–121
 - definition of, 117
 - objects and message passing, 116–120
 - Rails console, 112–113
 - title helper revisited, 121
- Microblog, 41–68. *See also Demo app*
- Micropost class with inheritance, 64
- Micropost migration, 392
- Micropost model, 391–405
 - attr_accessible method, 393–394
 - basic model, 392–394, 393f
 - data model, 393f
 - micropost refinements, 399–403 (*See also* Micropost refinements)
 - micropost validations, 403–405
 - user/micropost associations, 394–399, 394f, 399t
 - validations, code, 405
 - validations, test, 403–404
- Micropost refinements, 399–403
 - default scope, 400–401
 - dependent: destroy, 402–403
 - Micropost spec, initial (lightly edited), 394
 - Micropost validations, 403–405
 - micropost.association method, 399–400
 - Microposts. *See also* Demo app; Micropost model; *specific topics*
 - belongs to user, 63
 - CSS for, adding, 410–412, 412f
 - in demo app, 41 (*See also* Demo app)
 - length of, constraining, 61
 - manipulating, 414–437 (*See also* Manipulating microposts)
 - sample data, adding to, 413
 - showing, 405–414 (*See also* Showing microposts)
 - user has many, 63
 - Microposts controller, 59–60
 - Microposts resource, 57–67
 - associations in, 64f
 - console in, 63
 - demo app in, deploying, 66–67
 - inheritance hierarchies, 64–66, 65f
 - size of, limiting, 60–62, 62f
 - user has many microposts, 62–63, 64f
 - Microposts resource microtour, 57–60
 - create microposts, 58
 - microposts controller, 59–60
 - new microposts, 60, 60f
 - rails routes, with new rule for microposts, 58
 - Microposts, user, 391–440. *See also specific topics*
 - exercises on, 438–440
 - manipulating, 414–437
 - Micropost model, 391–405
 - showing, 405–414
 - MicropostsController class with inheritance, 65f, 66
 - /microposts/new, 60, 60f
 - Migrating up, 186–187

- Migration, 183, 185
 - database, 184–188 (*See also* Database migrations)
 - password, 230–232, 232f
 - rolling back, 187–188
 - for user model to create users table, 185–186
 - Migration generator, 212
 - Mockingbird, 145
 - Mockups, 145
 - Model(s), 16, 17f, 183. *See also* Micropost model; *specific models*
 - data, 42–43
 - inheritance hierarchies for, 64–65, 65f
 - micropost, 391–405 (*See also* Micropost model)
 - Relationship, 442–461 (*See also* Relationship model)
 - user, 182–197, 216–220 (*See also* User model)
 - Model annotation, 189–190
 - Model file, 188–190, 188f
 - Model objects, user, factory to simulate, 249–250
 - Model, view, controller (MVC), user, 216–220, 217f, 219f
 - Modeling and viewing users, 181–223. *See also specific topics*
 - data model creation, 181
 - exercises, 223
 - roll your own authentication system, 181b–182b
 - topic branch for, making, 182
 - user model, 182–197
 - user show page, 182, 183f
 - user validations, 197–213
 - Model-view-controller (MVC), 15–18, 17f, 47–56
 - controller actions, 51, 56t
 - controller actions *vs.* pages, 51
 - index action, simplified user, 53–54
 - instance variables, 54–56
 - rails routes, 48–49
 - REST architecture in, 51–52, 56t
 - steps in, 47–48, 55f
 - user index view, 56
 - user model for demo app, 54
 - users controller in schematic form, 49–50
 - Users controller–User model relationship in, 53–54
 - Modifying built-in classes, 137–138
 - module ApplicationHelper: code, 121
 - Mongrel web server, 18
 - Mostly static pages. *See* Static pages, mostly
 - Motivation, in following users, status feed, 485–489, 486f
 - Motivation, in Rails-flavored Ruby, 107–112
 - cascading style sheets, 110–111, 112f
 - title helper, 107–110
- ## N
- :name, 128
 - name attribute, 43, 43f
 - length validation for, adding, 204
 - making accessible, 190
 - presence of, validating, 198, 203
 - in user form inputs, 272–273
 - in user views, 252–255, 256f
 - validation of, failing test for, 201
 - Name length validation test, 204
 - Named constructor, 132
 - Named routes, 163, 164t, 169–171, 171f, 176. *See also specific routes*
 - Namespaces, 370
 - Navigation, site, 146–151, 148f
 - alt and class attributes, 149–150
 - container div, 149
 - custom.css for, 155–156, 156f

- image_tag helper, 149
- link for custom.css, 148
- link_to helper, 150–151
- site layout with added structure, 147–148, 147f
- stylesheets for Internet Explorer, 146–148
- ul and li tags, 150
- NERD tree project drawer, 11
- Nested hash, 129, 312
- NetBeans, 9
- new action, 46t
 - adding @user to, 269
 - RESTful route, 56t, 222t
 - signin form, 304–305
- New application, rails command for, 14–15
- New page, 46, 48f
- New Rails project, creating, 69
- new sessions action and view, tests for, 305
- New user, 46, 46t, 48f
- new user? boolean method, 351
- New user (signup) page
 - action for, 173
 - custom title, setting, 175
 - test, 266–267
- New users, form to sign up, 268–269
- Newline, 94
- new_user_path, 259t
- new_user_url, 259t
- next method, 375
- nil, 115
 - boolean context, 119
 - chaining messages passed to, 118
 - hash value for undefined key, 129
 - methods on, 118
 - test for, 118
- nil? method, 118
- NoMethodError, 235

- Not equal (!=), 123
- Notifications, follower, 499

O

- Object-oriented programming (OOP), 64
- Objects, 116–120
 - definition, 116
 - functions, 116
 - nil, 118
- Objects, user
 - creating, 190–194, 191f
 - destroying, 194
 - finding, 194–195
 - updating, 196–197
- One-element options hash, 204
- Options hash, one-element, 204
- Order of user's microposts, testing, 400–401

P

- p (paragraph) tag, 73, 77
- Page controller spec
 - adding action to, 90–92, 91f, 92f
 - writing failing, 87–89, 90f
- /page/home, 76f
- Pages
 - following and followers, 472–476, 473f, 474f (*See also* Following and followers pages)
 - new users, test, 266–267
 - static, 69–106 (*See also* Static pages, mostly)
- Pages controller
 - with added about action, 90–92
 - adding file to repository, 78
 - generating, 75–77
 - home action in, 75
 - home view in, raw, 75, 76f
 - inheritance hierarchy for, 139, 140f
 - with per-page titles, 99
 - in text editor, 75–77

- Pages controller spec
 - code, 84
 - with failing test for About page, 88–89, 89f
 - with title test, 94–95
- Pages, protecting, 355–364
 - expectation bonus, 361–362
 - friendly forwarding, 362–364
 - requiring right user, 359–361
 - requiring signed-in users, 355–359, 356f, 358f
- PagesController, 76–77, 76f, 138–139
- pages_controller.rb, 76
- /pages/home, 75
- Page–URL correspondence, 46, 46f
- paginate method, 373–374, 410, 496–497
- Paginated feed, home action with, 497
- Pagination, 371–373
 - testing, 373–378, 375f, 376f
 - will_paginate method installation for, 371–373
 - will_paginate method use for, 373–374, 375f, 376f
- palindrome? method
 - checking, 137
 - in String class, 137
 - word class from, 133–135
- Paperclip, 255
- Paragraph tag (p), 73, 77
- Parameter logging, filtering, 283–284
- params hash
 - for debugging site layout, 213–215
 - nested, 312
 - signin failure, 311–312
 - for signin form, 309
 - in signup failure, 278–280, 279f
- params[:id], 218–219, 278
- params[:session][:email], 309
- params[:session][:password], 309
- params[:user], 279–280
- Partial refactoring, 378–379
- Partials, 159–163, 162f. *See also specific partials*
 - for displaying follower stats, 466–468, 469f
 - for stylesheets and headers, 159–160
- Password(s)
 - confirmation of, user, 226
 - in development log, filtering, 284
 - in development log, visible, 283–284
 - in log, filtering from, 284
 - presence of, tests for, 227–228
- password attribute, 225, 227
 - validations for, 229
- Password(s), encrypted, 228, 230
 - adding to users table, 232f
 - before_save callback in creation of, 233–234
 - migration to add column to users table, 212
- Password(s), insecure, 225–236
 - active record callback, 232–236
 - password migration, 230–232, 232f
 - password validation, 226–230
- Password migration, 230–232, 232f
- Password reminders, 499
- Password(s), secure, 236–247
 - authenticate method, 243–247
 - has_password? method, implementing, 240–243
 - secure password test, 236–238
 - theory, 238–240
- Password validation, 226–230
 - as accessible attribute, 230
 - confirm password, 226–227
 - of password attribute, 228–229
 - test for, 227–228
- password_confirmation attribute, 227

- password_confirmation attribute, virtual, 228–229
- password_confirmation, filtering from log, 284
- PasswordReminders resource, 499
- PeepCode, 501
- Pending spec, 200
- Pending User spec, autotest with, 201f
- Persistence, 184
- Phusion Passenger, 35
- Planning demo app, 41–43, 42f
 - data model, 42–43
 - modeling microposts, 43, 44f
 - modeling users, 43, 43f
- Pluralization convention, default, 443–444
- POST
 - HTTP, 85b–86b
 - RESTful, 56t
- post method
 - to create action, 274–275
 - to create new object, 273
- Preparation, 4–5
 - experienced programmers, new to web development, 6
 - experienced Rails programmers, 7
 - experienced Ruby programmers, 6
 - experienced web developers new to Rails, 6
 - inexperienced programmers, designers, 5–6
 - inexperienced programmers, non-designers, 5
 - inexperienced Rails programmers, 6–7
- Presence, validating, 197–203. *See also* validates_presence_of; *specific attributes and topics*
- Printing strings, 115
- private keyword, 234–235
- Profile link
 - adding, 340–341, 341f
 - test, 340
- Profile page, user
 - adding follow form and follower stats to, 470–471, 471f, 472f
 - after successful signin, 317, 317f
- protected keyword, 235
- Protecting pages, 355–364. *See also* Pages, protecting
- Proto-feed, micropost, 424–432
 - (empty) @feed_items instance variable in create action, adding, 431–432
 - feed instance variable in home action, adding, 429
 - home page after creating new micropost, 432f
 - home page with form errors, 426f
 - home page with new micropost form, 425f
 - home page with proto-feed, 431f
 - home page with proto-feed, mockup, 427f
 - partial for single feed item, 430
 - preliminary implementation, 428
 - (proto-)status feed, tests for, 425–427
 - status feed in Home page, adding, 430–431, 431f
 - status feed partial, 429
- Prototype JavaScript Library
 - including in site layout, 482–483
 - use of, 481
- public/, 16t
- Public directory, 71, 72f
- Public interface, 236
- public.index.html, 72f
- public/index.html, 71, 72f
- Push, Git, 34, 35f
- Push operator (<<:), 124, 377
- PUT
 - HTTP, 85b–86b
 - RESTful, 56t
- puts method, 115

Q

q, 27

R

-r, 110

RadRails, 9

rails command, 14–15, 41–42

Rails console, 112–113

Rails Machine, 35

Rails root, 8

Rails router, 48, 55f

Rails routes, 48–49, 166–169, 166f

config/routes.rb, 166–167, 166f

for mapping root route, 168–169

with new rule for microposts resources, 58

with rule for users resource, 49

for static pages, 167–168

URL helpers in, 168–169

RailsBrain, 111

Railscasts, 498, 500

Rails.env.development?, 215

Rails-flavored Ruby, 107–143. *See also specific*

topics

arrays, 122–124

blocks, 125–127

cascading style sheets revisited, 130–131

exercises, 143

hashes, 127–128

method definitions, 120–121

motivation, 107–112 (*See also* Motivation, in
Rails-flavored Ruby)

objects and message passing, 116–120

other data structures, 121–131

Rails console, 112–113

ranges, 124–125

Ruby classes, 132–143 (*See also* Classes,
Ruby)

strings, 113–116 (*See also* Strings)

symbols, 128–130

title helper revisited, 121

rails.vim enhancements, 11

Rainbow attack, 239

Rake, 21, 23b

for populating database with sample users,
370–371, 371f

rake db:migrate, 21, 45, 188

in production, 216

in user model, 187f

rake db:migrate command, 23b

rake db:reset, 252

rake db:rollback, 187

rake spec, 87

Rakefile, 16t

Random subdomain generation, 127

Ranges, 124–125

README, 16t

GitHub, 31f

GitHub, initial, 30

Red, 87–90

definition of, 82

writing failing page controller spec in,
87–89, 89f, 90f

Red, green, refactor cycle, 82–92

Autotest, with Growl, 87

Autotest, with Growl notification, 88f

generating pages controller, 82–84

GET home, running test, 86–87

GET home, writing test, 84–85,
85b–86b

green, 90–92 (*See also* Green)

overview, 82

red, 87–90 (*See also* Red)

refactor in, 92

redirect, in user create action, 287

redirect_back_or method, 364

redirect_to @user, 287

Refactor, 82, 92

Refactoring

of code, 69

partial, 378–379

- Refinements, micropost, 399–403
 - default scope, 400–401
 - dependent: destroy, 402–403
- regex, 206
- Regular expression, 174, 206
- Relationship controller responses to Ajax requests
 - code, 483–484
 - tests, 481–482
- Relationship creation, testing with save!
 - method, 450
- Relationship model, 442–461
 - constructing, overview, 442–443
 - followers, 459–461, 459f
 - following, 454–458 (*See also* Following)
 - problem with data model (and solution), 443–449, 447f–449f
 - Relationship data model, 449, 449f
 - user following, naïve implementation of, 444–445, 447f
 - user following, through intermediate Relationship model, 447, 448f
 - user/relationship associations, 449–453 (*See also* User/relationship associations)
 - validations, 453–454
- Relationships controller
 - code, 479–480
 - tests, 477–479
- reload, 196
- Remember me, signin, 319–325
- remember_me! method, 321–322
 - method, 322–323
 - tests, 322
- remember_token attribute, 321b, 324
 - finding current user by, 331
 - migration for, 324–325
- :remember_token key, 326
- remove_column method, 232
- rename, in Heroku, 37–38, 37f
- render call
 - for partial refactoring, 378–379
 - in show_followers, 476, 477f, 479f
- render helper, for site layout with partials, 159–160
- render method, in controller actions, 277–278
- render_template function, 165
- replace_html method, 484
- Replies, 498–499
- Repository
 - adding RSpec files to, 80
 - first-time setup, 24–25
 - GitHub, 31f
- Repository, creating at GitHub
 - demo app, 42f
 - static pages app, 69, 70f
- Representational state transfer, 220
- Request, 16
- request object, 363
- require, 200
- require ‘digest,’ 240
- Resource, users, 44–57, 220–222, 222f, 222t
 - adding to routes file, 220–221, 222f
 - MVC in action, 47–56 (*See also* Model-view-controller (MVC))
 - named routes in, 259t
 - purpose of, 44
 - scaffold command, 44–45
 - user tour, 45–47, 46t
 - weaknesses of, 57
- Resources, 500–501
 - books, 501
 - PeepCode, 501
 - Railscasts, 500
 - Scaling Rails, 501
 - screencasts, *Ruby on Rails Tutorial*, 500
- respond to? method, 230
- respond_to blocks, 500
- response.should be_success, 86b

- REST, 3, 220
 - REST API, 500
 - REST architecture, 41, 51–52, 56t
 - REST resources, created/destroyed, 444–445
 - RESTful actions, standard, adding resource to
 - get for Sessions, 305–306, 306t
 - RESTful resource, sessions as, 304
 - RESTful routes, 51, 56t
 - custom rules in, 465t
 - in Micropost resource, 415, 417t
 - in users resource, 222, 222t
 - return keyword, 120–121
 - Reverse relationships
 - implementing user.followers using, 459–461
 - table for, 459, 459f
 - testing, 459–460
 - Reverse relationships table, 459, 459f
 - Root directory, 8
 - Root route mapping, 168–169
 - Round corners
 - custom.css for, 158f
 - stylesheet rules for, 158–159, 158f
 - Route(s)
 - integration test, 165
 - named, 163, 164t, 169–171, 171f (*See also* Named routes)
 - rails, 48–49, 166–169, 166f (*See also* Rails routes)
 - RESTful, 51, 56t
 - RESTful, with new rule for microposts, 58
 - for signup page, 176
 - for static pages, 167
 - Route mapping, for home page, 168
 - Router, Rails, 48, 55f
 - Routes file, adding users resource to, 220–221, 222f
 - routes.rb, map.resources :users added to, 273–274
 - Rows, 184
 - Rspec
 - installation of, 79–80
 - using Webrat with, 294
 - RSpec integration tests, for sign up, 292–299
 - users signup failure should not make a new user, 294–297
 - users signup success should make a new user, 297–299
 - Webrat, 293–294
 - RSpec, using Webrat with, 294
 - RSS feed, 499
 - Rubular regular expression editor, 206–207, 207f
 - Ruby. *See also specific topics*
 - learning, before Ruby on Rails, 4–5
 - online tutorial on, 5
 - ruby, 8
 - Ruby JavaScript (RJS)
 - to create following relationship, 484
 - to destroy following relationship, 484
 - ruby script/generate, 75
 - ruby script/server, 18
 - RubyMine, 9
- S**
- Safari, 11
 - Salt, 239
 - Salt column migration, to add to users table, 241, 241f
 - Sample data, adding microposts to, 413
 - Sample users, 369–371, 371f
 - Sandbox, starting console in, 190–191
 - Sass, 100
 - save method, 192
 - in building new users, 276
 - in user create action, 287
 - save! method, testing relationship creation
 - with, 450
 - save_without_validation method, 323

- Scaffold code generation, for microposts, 57–58
- scaffold command, 44–45
- Scaffold generator, 44–45
 - app generation via, 41
 - users resource (*See* Resource, users)
- Scaffolding, 1, 2b–3b
- Scaling Rails, 7, 501
- schema_migrations, 21
- Scope, 491–494
- Screencasts, *Ruby on Rails Tutorial*, 500
- Script, 18
- script/, 16t
- script/console, 63
- script/generate, 185
- script/generate scaffold, 57–58
- Script/server, 18–23
 - 500 Internal Server Error, 18–19, 20f, 21f
 - application environment, 18–19, 19f
 - http://localhost:3000, 18–20, 19f, 20f
 - Rake database setup, 21
 - restart, 21–22, 22f
 - SQLite, 19–20
 - sqlite3 and sqlite3-dev installation, 20
- Search, 500
- secure one-way hasfing, 236
- Secure passwords. *See* Password(s), secure
- secure_hash function, 238–240
- self keyword, 137
 - authentication, 245–246
 - encrypted passwords, 235–236
- self.down method, 187
- self.encrypted_password, 235
- self.up method, 186–187
- Server, default Rails, 18, 19f
- session facility, 363
- Sessions, 303–309
 - adding resource to get standard RESTful actions for, 305–306, 306t
 - new session action and view, tests for, 305
 - sessions controller, 304–306, 306t
 - signin form, 306–309, 307f, 309f
- /sessions, 306t
- Sessions controller, 304–306, 306t
- SessionsHelper module, 319–320
- /sessions/new, 305, 310, 311f
- Settings link, adding, 349–350, 349f
- SHA2, 239
- Short-circuit evaluation, 333b
- should_not be_valid, 202
- should_not method, 202
- show action, 46t, 218–219, 219f
 - RESTful route, 56t, 222t
 - user, 248
 - users controller with, 218
- Show page, 46, 49f
- Show page, user
 - CSS for styling, including sidebar, 260, 261f
 - getting, with factory and stub, test, 250–251
 - testing, with factories, 248–252
 - tests for, 252–253
 - title for, 254
 - /users/1 with sidebar, 258–259
- Show page, user, augmenting, 405–412
 - adding CSS for, 410–412, 412f
 - adding @microposts instance variable to show action, 410
 - adding microposts to, 407–409
 - mockup, 405, 406f
 - partial for showing single micropost, 410
 - showing microposts, test, 405–407
- show user, 46, 46t, 49f

- Show view, user
 - with name and Gravatar, 256–257, 256f, 257f
 - with user's name, 255, 256f
- show_follow view, to render following and followers, 476
- Showing microposts, 405–414
 - augmenting user show page, 405–412, 412f (*See also* Show page, user, augmenting)
 - sample microposts, 412–414, 414f–416f
- Showing users, 364–379
 - mockup of user index, 364, 365f
 - pagination, 371–373
 - pagination, testing, 373–378, 375f, 376f
 - partial refactoring, 378–379
 - sample users, 369–371, 371f
 - stub view for information on, 218
 - user index, 365–369, 369f (*See also* Index, user)
- shuffle, 127
- shuffle method, 123
- Sidebar
 - user, 258–260, 261f
 - in user show page /users/1, 258–259
- Side-effect, puts method as, 115
- Sign in, 303–316
 - exercises on, 343–344
 - pending tests for, 318
 - sessions, 303–309
 - signin failure, 310–316 (*See also* Signin failure)
 - signin success, 317–334 (*See also* Signin success)
 - signing user in, filling in, test, 328–329
- Sign up, 265–301. *See also specific topics*
 - exercises, 300–301, 301f
 - failure, 273–284
 - form, 265–273
 - RSpec integration tests, 292–299
 - success, 285–292
 - signed_in? method, 329–330, 333–334, 339
 - Signed-up users, also signed in, testing, 336–337
 - /signin, 306t
 - Signin failure, 310–316
 - failed signin, test and code, 313–316
 - reviewing form submission, 310–313, 310f
 - Signin form, 306–309
 - after trying to access protected page, 358–359, 358f
 - code, 308
 - example, 309f
 - HTML, 308–309
 - mockup, 306, 307f
 - sign_in function, 326, 327–328
 - complete (but not-yet-working), 321
 - to stimulate user signin inside tests, 334
 - tests, 318–319
 - sign_in method, @user, 337
 - Signin page, adding title for, 306
 - Signin success, 317–334
 - completed create action, 318–319
 - cookies, 326–327, 326f
 - current user, 327–334
 - profile page mockup, 317, 317f
 - remember me, 319–325
 - /signin URL, 164t
 - Signing out, 334–342
 - changing layout links, 338–341, 341f
 - destroying sessions, 334–336
 - exercises on, 343–344
 - signin upon signup, 336–337
 - signin/out integration tests, 341–342

- /signout, 306t
- sign-out function, 335–336
- sign-out method, 336
- Signup
 - failed user, tests for, 275–276
 - signing in user upon, 337
- ‘/signup,’ 176
- Signup button
 - custom.css for, 156–157
 - style sheet rules for, big, green, and clickable, 156–157, 157f
- Signup confirmation, 499
- Signup error messages, 281–283, 282f
- Signup failure, 273–284
 - arranging, 294–295, 295f
 - filtering parameter logging, 283–284
 - mockup, 273
 - params hash in, 278–279, 279f
 - should not make a new user, RSpec integration test of, 294–297
 - signup error messages, 281–283, 282f
 - testing, 296
 - testing failure, 273–277, 294–296, 295f
 - testing with lambda, 297
 - in working form, 277–280, 279f
- Signup form, 265–273
 - adding error message display to, 281, 282f
 - current state of, 265–266, 266f
 - finished, 287
 - form HTML, 272f
 - form_for helper method for, 267–269
 - HTML for, 270–273, 270f
 - mockup of, 267f
 - new users page, tests for, 266–267
 - text *vs.* password fields in, HTML, 272, 272f
 - user form_for, 270f
- Signup link
 - integration test, 175–176
 - user, simple integration test for, 175–176
- Signup page
 - action for, 173
 - home page with link to, 151, 152
 - linking button to, 176, 177f
 - route for, 176
 - testing, 171–173
 - title, testing, 173–175
- Signup success, 285–292
 - finished signup form, 287
 - first signup, 290–291, 291f, 292f
 - flash, 288–290
 - mockup, 285, 285f
 - should make a new user, RSpec integration test of, 297–299
 - testing, 285–287, 298–299
- Signup URL, 175–177, 177f
- /signup URL, 164t
- Signup, user, 171–177
 - adding flash message to, 290
 - failed, tests for, 275–276
 - signup URL, 175–177, 177f
 - successful, test for flash message on, 289–290
 - users controller, 171–175
- signup_path, 176
- Single user, partial to render, 379
- Single-quoted strings, 115–116
- Site layout. *See* Layout
- Site navigation, 146–151, 147f, 148f. *See also* Navigation, site
- Slightly dynamic pages. *See* Dynamic pages, slightly
- sort_by , 127
- spec
 - initial user, 199–200
 - in name, omitting files with, 5
- spec program, 79
- spec script?, at command line, 86–87
- spec spec/, running test with, 87–88

- `_.spec.rb`, 164
- Spike, 82
- `split` method, 122
- SQLite database, 187, 188f, 217, 217f
- SQLite gem for Ruby, installing, 19–20
- SQLite, installing, 19–22
- `sqlite3`, installing, 20–21
- Staging area, 26
- Static pages, mostly, 69–106
 - committing and merging, 105
 - committing changes, 102
 - create new Rails project, 69–70
 - exercises, 105–106
 - merging changes into master branch, 102
 - pushing code to remote repository, 102
 - routes for, 167
 - slightly dynamic pages, 93–104 (*See also* Dynamic pages, slightly)
- Static pages, mostly, first tests, 78–92
 - need for, 79
 - TDD: red, green, refactor, 82–92 (*See also* TDD (test-driven development))
 - tools for, 79–82 (*See also* Testing tools)
 - writing of, differences in, 78
- Static pages, mostly, making static web pages, 71–78
 - static pages with Rails, 74–78, 76f
 - truly static pages, 71–74, 72f, 73f
- Stats, web interface, 463–472
 - follow form and follower stats in user profile page, adding, 470–471, 471f, 472f
 - follower stats in Home page, adding, 468
 - following and followers actions in Users controller, adding, 464
 - following/follower statistics on Home page, testing, 465–466
 - form for following user, 470
 - form for unfollowing user, 470
 - mockup, stats partial, 463–464, 464f
 - partial for displaying follower stats, 466–468, 469f
 - partial for follow/unfollow form, 468
 - RESTful routes, 464
- status command, Git, 26
- Status feed, 415, 485–497. *See also* Proto-feed, micropost
 - final implementation of
 - from `_users_followed_by`, 495–496
 - first feed implementation, 489–491
 - lambda, 495–496
 - mockup, final feed, 485, 486f
 - motivation and strategy, 485–489, 486f
 - new status feed, 496–497, 496f
 - scope, 491–494
 - subselect, 494–495
- `store.location` method, 363
- string, 43, 43f
- String class
 - `blank?` added to, 138
 - inheritance hierarchy for, 133, 134f
 - `palindrome?` added to, 137
- String literals, 114
- String multiplication, 204
- `string_message` method, 120
- Strings, 113–116
 - to arrays, 122
 - concatenation of, 114
 - constructors for, 132
 - double-quoted, 114–115
 - empty method on, 117
 - literals, 114
 - single-quoted, 115–116
 - variable names in, 114–115
- Strings, interpolation in, 144
- printing, 115
- single-quoted, 115–116
- Stub view
 - about page, 91–92, 91f
 - showing user information, 218

stylesheet.link_tag, 111
 stylesheet.link_tag method, 130–131
 Stylesheets. *See also* Custom cascading style sheets (custom.css)
 adding to layouts, 111
 adding to sample application layout, 111
 Internet Explorer, 146–148
 site layout with partials for, 159–160
 _stylesheets.html.erb, 159–160
 Subdomains, generating random, 127
 Subselects, 494–495
 :success, 289
 sudo, 13
 superclass method, 133, 134f
 Symbols, 128–130
 System setup, first-time, 24

T

t object, 186
 table tag, 258
 Table, users
 added (encrypted) password attribute, 232, 232f
 adding encrypted_password column to, 231–232, 232f
 adding encrypted_password to, 232f
 adding index to, 212
 adding name and email to, 185–186
 adding salt column to, 240–241, 241, 241f
 creation of, 184–185, 188f
 dropping from database, 187
 Tables, 184
 tail, 191
 Tailing, development log, 191, 191f
 taps gem, 67
 td (two table data cells), 258
 TDD (test-driven development), 2, 5, 78
 best applications, 82
 failing test, writing, 82–83
 interfaces, 236
 pages controller spec, generating, 83–84
 red, green, refactor cycle, 82–92 (*See also* Red, green, refactor cycle)
 Terminal, 10
 test/, 16t
 Testing tools, 79–82. *See also specific tools and topics*
 installing Autotest, 80–82
 installing Rspec, 79–80
 Tests (testing). *See also specific tests and topics*
 automated, 69
 integration, 163–166, 164t
 test_sign_in function, to stimulate user sign-in
 inside tests, 334
 Test::Unit, 79–80
 Text editors, 9–11, 10f
 TextMate, 94
 Textmate, 10, 10f
 Thinking Sphinx, 500
 3rd Rail, 9
 time_ago_in_words helper method, 410, 414, 414f
 Timestamp, 186
 Title
 base, 108
 change in, testing, 93–95
 setting custom, for new user page, 175
 signin page, adding, 306
 signup page, testing, 173–174
 for user show page, 254
 @title, 108, 175
 Title helper, 254
 defining, 109
 development of, 109
 need for, 107–109
 revisited, 121
 simplifying layout with, 109–110
 using h to escape the HTML, 254–255

Title tag

- creating, 72, 73, 74

- removing, 74

Title test, 93

- passing, 92f, 96–98, 98f

- testing title change in, 93–95, 93t

<title><%= @title %></title>, 108

@title variable, 99–100

tmp/, 16t

to_a method, 124

toggle! method, 381

Tools. *See also specific topics and tools*

- learning, 12

Topic branch, in Git, 182

to_s method, 118

Total Validator, 11

Tour, user, 45–47, 46t

tr (one table row), 258

True, Ruby objects as, 119–120

U

ul (unordered list tag), 150

_footer.html.erb, 160–161

_spec.rb, 164

_stylesheets.html.erb, 159–160

Unfollow button, user profile with, 445f,

472f

unfollow! method

- code, 458

- test, 457–458

unfollow! utility method, 477–479

Unfollowing user

- form for, 470

- form for, using Ajax, 481

Uniqueness caveat, 211–213

Uniqueness validation, 209–213

Unix, 8

unless keyword, 119

uppercase method, 210

update action, 51

- RESTful route, 56t, 222t

- user, code, 354–355

- user, tests, 352–354

update page, correct_user before filter to

- protect, 360

update_attributes method

- assigning email address, with Gravatar,

- 256–257, 257f

- updating user objects, 196–197

updated_at magic column, 186, 393

Updating user objects, 196–197

Updating users, 345–355. *See also specific*

topics

- edit form, 346–351

- enabling edits, 352–355

URL mapping, for site links, 163, 164t

@user

- adding to new action, 269

- to create form tag, 273

- creation of, 248

- superfluous assignments, removing,

- 361–362

- in test forgetting user show page, with

- factory and stub, 250–251

User actions. *See also specific actions*

- destroy, 46–47, 54f

- edit, 46

- new, 46, 48f

- show, 46

- with updated information, 52f

User association, micropost, tests for,

396–397

User class, 141–143

- example, code, 141

- with inheritance, 64

User edit page, 51f

User hash, 129

User information, stub view for showing, 218

- User list, 46, 46t, 47f
- User microposts. *See* Microposts, user
- User model, 182–197, 216–217, 217f
 - added (encrypted) password attribute, 232, 232f
 - added remember token, 325f
 - adding following association, with `has_many :through`, 455
 - annotated, code, 189–190
 - brand new, code, 189
 - creating data structure for users, 182–183
 - creating user objects, 190–194, 191f
 - database, 183
 - demo app, 54
 - destroying user objects, 194
 - finding user objects, 194–195
 - functionality of, testing, 200
 - generation of, 185
 - model file in, 188–190, 188f
 - model in, 183
 - updating user objects, 196–197
- User model, database migrations, 184–188
 - generating user model in, 184–185
 - migration in, 185–186
 - migration in, to create users table, 185–186
 - `rake db:migrate` in, 186–187, 187f
 - `self.down` method in, 187
 - `self.up` method in, 186–187
 - tables in, 184–185
- User model objects, factory to simulate, 249–250
- User model, view, controller (MVC), 216–220, 217f, 219f
- User objects
 - creating, 190–194, 191f
 - destroying, 194
 - finding, 194–195
 - updating, 196–197
- User show action, 248
- User show page. *See* Show page, user
- User show view
 - with name and Gravatar, 256–257, 256f, 257f
 - with user’s name, 255, 256f
- User signup link, simple integration test for, 175–176
- User spec, initial, 199–200
- User tour, 45–47, 46t
- User validations, 197–213
- `@user` variable, adding to new action, 269
- `User.all`, 47, 195
- `user.authenticate` method, 244, 246–247
 - code, 246–247
 - failed signin attempt, code, 315
 - signin failure, test and code, 313–314
 - tests for, 244
- `User.create`, 193
- `user.create!` method, 200, 370
- `User.find`, 194–195
- `User.first`, 63, 195
- `user.followers`, 444, 460–461
- `user.following`, 444
- `user.following` attribute, testing, 454–455
- `user_from_remember_token` method, 332
- `user_id` attribute, 43, 44f, 392
- `User/micropost` associations, 394f, 395–399, 399t
 - `belongs_to`, 395, 395f, 398
 - `has_many`, 395, 395f, 398
 - methods summary, 399t
 - `micropost belong_to` user, 398
 - `micropost’s` user association, tests for, 396–397
 - `user has_many` microposts, code, 398
 - `user’s` microposts attribute, tests for, 397–398

User.new, 191–192
 factory user from, 276
 User.paginate, 374
 user_patch, 259
 user_patch(@user), 259
 user_path@user, 287
 User/relationship associations, 449–453
 belongs_to, 449
 belongs_to associations in Relationship
 model, adding, 453
 foreign key, 451
 save!, testing relationship creation with,
 450
 user.relationships attribute, testing for,
 450
 user/relationships belongs_to association,
 testing, 452
 user/relationships has_many association,
 implementing, 451–452
 user.relationships attribute, testing
 for, 450
 user/relationships belongs_to association,
 testing, 452
 user/relationships has_many association,
 implementing, 451–452
 Users. *See also specific topics*
 in demo app, 41
 destroying (deleting), 379–387 (*See also*
 Destroying users)
 modeling and viewing (*See* Modeling and
 viewing users)
 new, form to sign up, 268–269
 sample, 369–371, 371f
 showing, 364–379 (*See also* Showing
 users)
 updating, 345–355 (*See also* Updating
 users)
 viewing, 213–222
 users, 43, 44f

/users, 46
 original, 46t, 47f
 RESTful route, 56t, 222t
 with second user, 53f
 :users, 48
 @users, 54
 @users = User.all, 54
 /users/1, 46, 46t
 after adding Users resource, 220, 222f
 initial, 220, 221f
 page to show, 49f
 RESTful route, 56t, 222t
 /users/1/edit, 46, 46t, 51f, 52f
 RESTful route, 56t, 222t
 @user.save, 277, 277b, 280
 UsersController class, with inheritance,
 65–66, 65f
 users/new, 46, 46t, 48f, 295
 RESTful route, 56t, 222t
 users_path, 259t
 users_path(@user), 259t
 users_spec.rb, 294
 users_url, 259t
 user_url(@user), 259t

V

valid? method, 199, 202
 validates_confirmation_of method, for
 passwords, 229
 validates_length_of method
 for names, 204
 for passwords, 229
 validates_presence_of method
 for name, 198, 199
 for name and email, 203, 204, 206
 for passwords, 229
 validates_uniqueness_of method, 209
 for email addresses, 210
 for email addresses, ignoring case, 211

- Validating presence, 197–203
 - Validations, 60–61
 - commenting out, to ensure failing test, 199
 - email address uniqueness, 210
 - email address uniqueness, ignoring case, 211
 - email format with regular expression, 206–208, 207t, 208f
 - length, 61–62, 62f
 - micropost, 403–405
 - name attribute, failing test, 201
 - name attribute presence, 198
 - password, 226–230
 - password attribute, 229
 - password, tests for, 227–228
 - Relationship model, 453–454
 - Validations, user, 197–213
 - format, 205–208
 - length, 203–205
 - presence, 197–203
 - uniqueness, 209–213
 - Variables, instance, 54–56, 99
 - nilness of, 118–119
 - vendor/, 16t
 - Version control systems, 23
 - Version control, with Git, 23–35.
 - See also* Git
 - Version, of Ruby, 12
 - vi editor, 10
 - View, 16, 17f. *See also specific types*
 - actions and, 77
 - Rails, static HTML in, 77–78
 - user, 217–219, 219f
 - for user index, 56
 - view action, 54
 - Viewing users, 213–222
 - debug and rails environments, 213–216, 214f
 - user model, view, controller, 216–220, 217f, 219f
 - users resource, 220–222, 221f, 222f, 222t
 - Views, user, better, 247–261
 - name and Gravatar, 252–258, 256f, 257f
 - testing user show page (with factories), 248–252
 - user sidebar, 258–260, 261f
 - Vim, 10, 11
 - Vim for Windows, 10
 - Virtual attribute, 228
- ## W
- Web interface for following and followers, 461–485. *See also specific topics*
 - follow form, 468–472
 - following and followers pages, 472–476, 473f, 474f
 - sample following data, 462–463
 - stats, 463–468
 - working follow button, Ajax, 480–485
 - working follow button, standard way, 476–480, 477f, 479f
 - Web servers, 18
 - Webrat, 293–294
 - Webrat, with Rspec, 294
 - WEBrick web server, 18
 - Wideframes, 145
 - will_paginate gem, 371–373
 - will_paginate method, 408
 - installation of, 371–373
 - use of, 373–374, 375f, 376f
 - Windows
 - “ruby script” for, 75
 - running Rails in, 8
 - with method, 314
 - :within option, 229

Word < String, 135

Word class

defining, in `irb`, 135

inheritance from string of, 135,

136f

inheritance hierarchy for, 135, 136f

in `irb`, 135

`palindrome?` method for, 133–135,

137

X

`xhr` method, 481

`xmlns`, 96

Y

YAML, 217

Z

Zero-offset arrays, 122