LEARN RUBY ON RAILS

BOOK ONE

DANIEL KEHOE

2

Learn Ruby on Rails: Book One

Version 3.0.0, 14 January 2016

Daniel Kehoe

ii

Contents

Chapter 1

Free Offer

I want you to have Book Two in this series for free.

You are reading Book One, which introduces basic concepts and gives you the background you need to succeed. In Book Two, you'll build a useful web application, for hands-on learning. The two books go together, which is why I want you to have both books.

I've created an online version of both books at the website learn-rails.com. You'll also find PDFs available for download. Look for the link "Free Online Edition" when you visit the site:

- learn-rails.com

Use the invitation code:

- LEANPUB300B1

I'll ask you to provide your email address when you sign up to get free access to Book Two. I work hard to keep the books up to date, incorporating improvements and fixing errors as readers report issues. I update the online edition of

the books often and I send email to notify of updates. If you bought the book from Amazon or another retailer, email is the only way to learn about updates.

I'll also let you know about the Capstone Rails Tutorials, which you'll want to read after you finish this book series. I promise there is no better way to learn Rails.

Chapter 2

Introduction

Welcome. This is a first step on your path to learn Ruby on Rails.

This book contains the background that's missing from other tutorials. Here you'll learn key concepts so you'll have a solid foundation for continued study. Whether you choose to continue with another book in this series, a video course, or a code school, everything will make sense when you start here.

You can read this book anywhere, at your leisure, on your phone or tablet. Use this book to gain background understanding when you are not at your computer. With Book Two, the next in the series, you'll need a computer at hand so you can build your first web application.

In Book Two, you'll build a working web application so you'll gain hands-on experience. Along the way, you'll practice techniques used by professional Rails developers. And I'll help you'll understand why Rails is a popular choice for web development.

You can start with Book Two before finishing this book if you're eager to get started building your first application. In fact, I recommend it, because the hands-on learning in Book Two reinforces the concepts you learn in this book.

Is It for You?

If you've built simple websites using HTML, you'll quickly progress to building websites with Rails. Or, if you have experience in a language such as PHP or Java, you'll make the jump to the Rails framework. But I promise you don't need to be a programmer to succeed with this book or the next. You'll be surprised how quickly you become familiar with the Unix command line interface and the Ruby programming language even if you've never tried programming before.

My books are ideal if you are:

- a student

- a startup founder

- making a career change

Some readers are happy to learn to communicate with developers they work with. If you are starting a business, and hiring developers, or working alongside developers as a manager or developer, this book will help you talk with developers. However, the true purpose of my book is to help you become you a Rails developer yourself. I want to help you launch a startup or begin a new career.

What To Expect

There is deep satisfaction in building an application and making it run. With this book and the next, I'll give you everything you need to build a real-world Rails application. More importantly, I'll explain everything you build, so you understand how it works.

When you've completed this tutorial, you will be ready for more advanced self-study, including the Capstone Rails Tutorials, textbook introductions to Rails,

or workshops and code camps that provide intensive training in Ruby on Rails. Other curriculums often skip the basics. With this tutorial you'll have a solid grounding in key concepts. You won't feel overwhelmed or frustrated as you continue your studies. I think you'll also have fun!

This book is good preparation for:

- textbooks such as Michael Hartl's Ruby on Rails Tutorial

- introductory workshops from RailsBridge or Rails Girls

- intensive training with immersive code camps

- Capstone Rails Tutorials from the RailsApps Project

We are blessed with many textbooks, workshops, and classroom programs that teach Ruby on Rails. I believe this book is unique in covering the basics while introducing the tools and techniques of professional Rails development.

What's in Book One

Book One is a self-help book that can change your life, though here you won't find any inspirational quotes or magical thinking.

I explain the culture and practices of the Rails community. I introduce the basic concepts you'll need to understand web application development. You'll learn how to be a successful learner and how to get help when you need it. I also provide a plan for study so you can learn more when you need it. There's so much to learn, it helps to have a map so you know where to go next.

Programming can be frustrating and Rails isn't easy for beginners. The chapter, "Rails Challenges," describes many of the problems learners encounter. It's natural to get discouraged so take a look when you begin to feel overwhelmed.

Two chapters, "Crossing the Chasm", and "Level Up", will help you after you put the book down. Many learners feel stranded if their only experience is

step-by-step tutorials. These chapters are designed to give you a strategy for building an application on your own.

What's in Book Two

You'll start coding in Book Two. It's a hands-on tutorial that will lead you through the code needed to build a real-world web application. Don't skip around in Book Two. The tutorial is designed to unfold in steps, one section leading to another, until you reach the "Testing" chapter.

You can complete Book Two in one long weekend, though it will take concentration and stamina. If you work through the book over a longer timespan, try to set aside uninterrupted blocks of two hours or more for reading and coding, as it takes time to focus and concentrate.

Feel free to start Book Two before you finish this book. Begin coding with Book Two while you get background knowledge from this book at your leisure.

A Warning About Links

My books are densely packed with links to background reading. If you click every link, you'll be a well-informed student, but you may never finish the book! It's up to you to master your curiosity. Follow the links only when you want to dive deeper.

What Comes Next

The best way to learn is by doing; when it comes to code, that means building applications. Hands-on learning with actual Rails applications is the key to absorbing and retaining knowledge.

After you read this book, you'll be able to work with the example applications from the RailsApps Project. The project provides open source example ap-

plications for Rails developers, for free. Each application is accompanied by a tutorial in the Capstone Rails Tutorials series, so there's no mystery code. Each application can be generated in a few minutes with the Rails Composer tool, which professional developers use to create starter applications.

The RailsApps Project is solely supported by sales of the books and the Capstone Rails Tutorials. If you purchase the Capstone Rails Tutorials, you'll keep the project going. And you'll have my sincere appreciation for your support.

Versions

Book One is relevant and useful for any version of Rails. Book Two requires a specific version of Rails (the newest at the time it was revised) and shows how to install the latest version of Rails.

Staying In Touch

If you obtained this book from Amazon or another retailer, take a moment to get on the mailing list for the book. I'll let you know when I release updates to the book.

- Get on the mailing list for the book

A Note to Reviewers and Teachers

This book approaches the subject differently than most introductions to Rails. It introduces concepts of product planning, project management, and website analytics to place development within a larger context of product development and marketing. In Book Two, rather than show the student how to use scaffolding, I introduce the model-view-controller design pattern by creating the components manually. Lastly, though every other Rails tutorial shows how to

use a database, Book Two doesn't, because I want the book to be a short intro-
duction and I believe the basic principles of a web application stand out more
clearly without adding a database to the application. Though this tutorial is
not a typical Rails introduction, I hope you'll agree that it does a good job in
preparing Rails beginners for continued study, whether it is a course or more
advanced books.

Using the Book in the Classroom

If you've organized a workshop, course, or code camp, and would like to assign
the book as recommended reading, contact me at daniel@danielkehoe.com to
arrange access to the book for your students. The book is available at no charge
to students enrolled in qualified workshops or classes.

Let's Get Started

In the next chapter, we'll start with basic concepts.
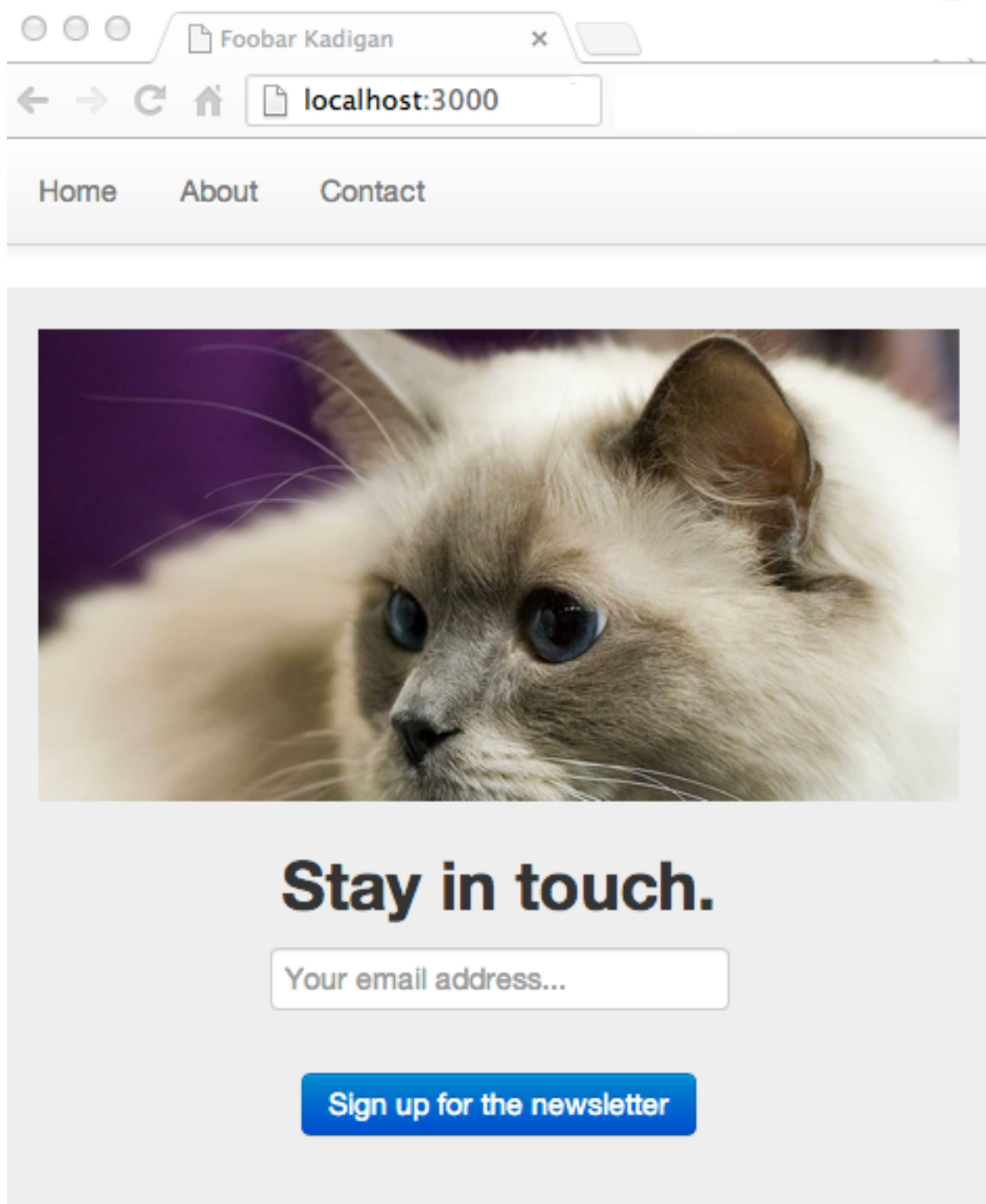
Figure 2.1: The application you will build in Book Two.

Chapter 3

Concepts

This chapter provides the background, or big picture, you will need to understand Rails.

These are the key concepts you'll need to know before you try to use Rails.

In the following two chapters, you'll gain a deeper understanding of Rails, including its history, the guiding principles of Rails, and reasons for its popularity. First, let's consider how the web works.

How the Web Works

We start with absolute basics, as promised.

When you "visit a website on the Internet" you use a *web browser* such as Safari, Chrome, Firefox, or Internet Explorer.

Web browsers are *applications* (software programs) that work by reading *files*.

Compare a *word processing program* with a *web browser*. Both word processing programs and web browsers read files. Microsoft Word reads files that are stored on your computer to display documents. A web browser retrieves files from remote computers called *servers* to display web pages. Simply put, the

World Wide Web is nothing more than files delivered to web browsers by web servers.

Web browsers make *requests* to web servers. Every web address, or *URL*, is a request to a web server. A web server *responds* by sending one or more files. We call this the *request-response cycle*.

Everything displayed by a web browser comes from four kinds of files:

- HTML - *structure* (layout) and *content* (text)

- CSS - *stylesheets* to set visual appearance

- JavaScript - *programming* to alter the page

- Multimedia - images, video, or other media files

At a minimum, a web page requires an HTML file. HTML files contain the words you see on a web page, along with *markup tags* that indicate headlines, paragraphs, and other types of text such as lists. If a web browser receives only an HTML file, it will display text, with default styles for headlines and paragraphs supplied by the browser.

Because it is the World Wide Web, HTML files also contain *hypertext links* to other web pages. Sometimes links appear in the form of a button or an image. Sometimes a web page contains a form with a button that sends information to the web server. Links are web addresses, or URLs, and (you guessed it), they return files.

If the page is always the same, every time it is displayed by the web browser, we say it is *static*. Webmasters don't need software such as Rails to deliver static documents; they just create files for delivery by an ordinary *web server* program. When you learn HTML and create simple web pages, you learn to upload files to a hosting service that provides web servers that deliver your HTML files to web browsers. In principle, you can run a web server delivering web pages from your computer at home but, in practice, most people want a

web server that runs 24 hours a day and is located in a *data center* that has fast and reliable connections to the Internet.

Static websites are ideal for particle-physics papers (which was the original use of the World Wide Web). But most sites on the web, especially those that allow a user to sign in, post comments, or order products and services, generate web pages *dynamically*. When you see a form with a button, you probably are looking at a page that makes a request to a web application.

Dynamic websites often combine web pages with information from a database. A database stores information such as a user's name, comments, advertisements, or any other repetitive, structured data. A database *query* can provide a selection of data that customizes a webpage for a particular user or changes the web page so it varies with each visit.

Dynamic websites use a programming language such as Ruby to assemble HTML, CSS, and JavaScript files on the fly from component files or a database. A software program written in Ruby and organized using the Rails *development framework* is a Rails *web application*. A web server program that runs Rails applications to generate dynamic web pages is an *application server* (but usually we just call it a web server).

Software such as Rails can access a database, combining the results of a database query with static content to be delivered to a web browser as HTML, CSS, and JavaScript files. Keep in mind that the web browser only receives ordinary HTML, CSS, and JavaScript files; the files themselves are assembled dynamically by the Rails application running on the server.

Even if you are not going to use a database, there are good reasons to generate a website using a programming language. For example, if you are creating several web pages, it often makes sense to assemble an HTML file from smaller components. For example, you might make a small file that will be included on every page to make a footer (Rails calls these "partials"). Just as importantly, if you are using Rails, you can add features to your website with code that has been developed and tested by other people so you don't have to build everything yourself.

The widespread practice of sharing code with other developers for free, and collaborating with strangers to build applications or tools, is known as *open source* software development. Rails is at the heart of a vibrant open source development community, which means you leverage the work of tens of thousands of skilled developers when you build a Rails application. When Ruby code is packaged up for others to share, the package is called a *gem*. The name is apt because shared code is valuable, like a gem.

Ruby is a programming language. *Rails* is a development framework. Rails is software code written in the Ruby language. It is a *library* or collection of gems that we add to the core Ruby language. More importantly, Rails is a set of *structures and conventions* for building a web application using the Ruby language. By using Rails, you get well-tested code that implements many of the most-needed features of a dynamic website. When you need additional features, you can add additional gems.

With Rails, you will be using shared standard practices that make it easier to collaborate with others and maintain your application. As an example, consider the code that is used to access a database. Using Ruby without the Rails framework, or using another language such as PHP, you could mix the complex programming code that accesses the database with the code that generates HTML. With the insight of years of developers' collective experience in maintaining and debugging such code, Rails provides a library of code that segregates database access from the code that displays pages, enforcing *separation of concerns*, and making more modular, maintainable programs.

In a nutshell, that's how the web works, and why Rails is useful.

For more on the history of Rails, and an explanation of why it is popular, see the next chapters. But before we dive into Rails, let's look at the increasingly complex world of web development, particularly the difference between front-end and back-end applications, and the programming languages we use, Ruby and JavaScript.

Programming Languages

JavaScript and Ruby are both general-purpose programming languages.

Developers use other popular programming languages such as C, Python, and Java. And developers like to talk about newer languages such as Elixir and Go, often comparing the popularity of programming languages. Most developers use only one or two popular languages on the job such as Ruby and JavaScript but hardcore programmers love to try new languages.

Just a note: Java and JavaScript are unrelated, except by name. Java is a general-purpose language used in large enterprises, such as banking, where large teams of developers build applications. JavaScript is a language that was developed for use in web browsers. It was named "JavaScript" to take advantage of the popularity of Java but has little in common with Java except for the name.

And a further note: HTML, the Hypertext Markup Language, is not a programming language. It is a *markup language* that uses tags to add structure and links to text. It doesn't allow *conditional execution* such as `if...  then...  else` which is key to programming. If you know HTML you can be a "coder," writing HTML code, but you are not really a programmer.

Ruby and JavaScript

Ruby is the programming language you'll use when creating web applications that run on your local computer or a remote server using the Rails web application development framework.

JavaScript is the programming language that controls every web browser. The companies that build web browsers (Google, Apple, Microsoft, Mozilla, and others) agreed to use JavaScript as the standard browser programming language. You might imagine an alternative universe in which Ruby was the browser programming language. That's not the real world; plus it would be

boring, as learning more than one language makes us smarter and better programmers.

JavaScript and JQuery

Though most of the code in Rails applications is written in Ruby, developers add JavaScript to Rails applications to implement features such as browser-based visual effects and user interaction. For simple Rails applications, you only need to learn Ruby. For more sophisticated web applications, you'll need to know both Ruby and JavaScript.

JavaScript was first used on websites to add little features to the browser. For example, JavaScript can be used to display the current date and time on a web page. Or JavaScript can be used to pop up an annoying window when you try to leave a web page. There was little consistent structure to early JavaScript programs. And because JavaScript is an older language without a built-on package manager, there were no package libraries like Ruby gems to add functionality. Instead, web developers shared scripts or snippets of code to add commonly-implemented features.

JQuery

In 2006, a group of developers released jQuery, a robust collection of scripts that are a foundation for most of the simple interactive user features found on websites today. Rails includes jQuery as part of any Rails application. You'll find jQuery on 65% of websites.

To understand jQuery, you need to know that every web browser takes an intermediate step between receiving an HTML file and displaying a web page. After a web browser receives a file from a web server, it creates code in the computer's memory that describes the web page, complete with text and formatting, which we call the *Document Object Model*, or DOM. JQuery scripts

manipulate the DOM, which is the web browser's internal representation of web page. JQuery commands make changes to the DOM. For example, you can write Javascript that replaces words on the page with different words. Or you can hide and reveal sections of the page.

Many Rails applications use jQuery or pure JavaScript to add interactive browser features. Though it is easy to add JavaScript to web pages generated by Rails, the result is often "JavaScript gravy" or "soup:" a sloppy mix of JavaScript snippets and jQuery plugins poured over Rails views. After seeing applications built with JavaScript soup, developers often get an urge to adopt a framework like Rails for JavaScript, with standard structures and conventions to organize code.

We'll look at JavaScript frameworks later in this chapter. But first, consider a universe where JavaScript is the only language you need to know. Not all developers use Rails; some use full-stack JavaScript.

Full-Stack JavaScript

In the last few years, developers have been improving the JavaScript language so it can be used for server-side development as well as development of applications that run in the browser. System administrators can install the Node.js code library to enable servers to run JavaScript. Server-side JavaScript web application frameworks are available, such as Express and Meteor, but none are as popular as Ruby on Rails. Some code schools are now teaching "full stack JavaScript" using the "MEAN Stack," which combines four popular JavaScript technologies; MongoDB, Express, AngularJS, and Node.js.

There are at least a dozen popular languages that developers use to build web applications. This will continue to be true. Full-stack JavaScript is now an option but Rails is not going away. For now, the Rails ecosystem is better known with more resources for building web applications and learning development. There will always be jobs for Rails developers. Startups will always be able to find developers who have experience with Rails. Your effort in learning Rails will continue to be worthwhile. However, it is important to learn JavaScript as

well.

Front and Back Ends

We often talk about "front-end" and "back-end" development. The architecture of the web, split between web browsers running on local computers and web application programs running on remote servers, is inherently *client-server*. When we write applications for both the front end and back end, we are *full-stack* developers.

Full-stack developers handle it all, including connections to database servers, server-side web applications, JavaScript for web browsers, and *system administration* of Unix servers. Increasingly, as web development gets more complex, there are fewer full-stack developers and more specialists. If you are building a web application for your own business, you may need to be a full-stack developer. But if you look for a job, you may choose to specialize as a front-end or back-end developer. Front-end developers have some design skill, worry a lot about user experience, and develop expertise with JavaScript. Back-end developers don't worry as much about design. They focus on the architecture of applications, database structure, and application performance using Ruby.

Rails was originally created for full-stack development. Its *model-view-controller* architecture manages both connections to databases and display of web pages, requiring nothing more than the Ruby language and the Rails software library. JavaScript was originally created for front-end development. Some developers now use JavaScript for full-stack development but it is still primarily used for adding interactive features to web pages.

Rails is popular for both the front end and the back end. The conventions of building Rails applications are widely known and a developer can leverage the work of tens of thousands of other developers by including open-source gems. A Rails back end can connect to thousands of different services and efficiently handle traffic for any average website. If your business is so successful that you must deal with problems of scale, you'll find many Rails experts able to help.

For these reasons, developers continue to use Rails to build web applications, even as full-stack JavaScript grows in popularity.

JavaScript is becoming more popular but developers will continue to build full-stack Rails applications where complex JavaScript user interfaces are not required. This could be quick "minimal viable product" (MVP) tests of business ideas or any project where it is too costly to invest in separate front-end and back-end development. If you are going to quickly build a web application, and launch a business, just build it with Rails and don't worry about implementing features in JavaScript.

If you want a complex front end that uses JavaScript extensively, you may still want to use Rails for the back end.

Rails 5

The Rails 5 beta version was released December 18, 2015. The final version will be out in early 2016.

Rails 5 recognizes the increasing use of Rails for back-end development. Rails 5 offers an option to build back-end-only applications that don't generate browser views. Instead, a stripped-down Rails 5 application returns data that can be consumed and displayed by another application. With this approach, you can have Rails on the back end and a JavaScript web browser application, or a mobile application, on the front end.

With Rails 5, you can build a back-end application that just delivers data to a front-end application. Before Rails 5, most Rails applications generated entire web pages containing the results from a database query. Now with Rails 5, Rails developers have begun to build web applications that deliver just data, in the JSON (JavaScript Object Notation) format. With this approach, the Rails application provides an API, or *application programming interface*, that provides standardized responses to requests from an application in the hands of a user, which could be an IOS or Android application on a phone, or a JavaScript application running in a web browser.

Now that you've learned about the differences between front-end and back-end applications, let's learn about popular JavaScript frameworks.

JavaScript Frameworks

I've described Rails as a web development framework that uses the Ruby language. You can also find web development frameworks that uses the JavaScript language.

JavaScript frameworks were first developed so developers could build SPAs, *single-page applications*. Early examples of SPAs, such as Apple's MobileMe or iWork in 2008, were attempts to use JavaScript to build desktop-like applications to run in the web browser. The goal was to create RIAs, rich Internet applications, with similarities to software applications installed on the Mac or Windows, using JavaScript. Of course, no one really wants desktop applications anymore. We browse web pages to get information. We install mobile apps to get access to services. Sometimes we want access to services in web browsers. But we don't really want desktop applications in a web browser.

AngularJS and Ember.js

AngularJS and Ember.js are the most popular JavaScript frameworks used to build single-page applications. They provide the structure and conventions that developers want for maintainable code. For complex, richly interactive web applications, these frameworks get rid of the "JavaScript soup" that is common in Rails applications.

Unlike Rails, which is hugely popular, none of these JavaScript frameworks have achieved hegemony or dominance among developers. For a learner, these frameworks require more programming experience than Rails. All of these frameworks break the familiar request-response cycle of the web. Just one URL delivers the entire single-page application. There are no web pages, only views

that change completely or partially in response to user actions. Drawbacks to single-page applications are a slow initial load of the application, difficulty using analytics tools such as Google Analytics which rely on new pages loading in the browser, and views that won't show up in Google search results. For all these reasons, single-page applications are best used to deliver services, not information, and require advanced skills to develop.

React

There's a newer framework that can be used to dry up JavaScript soup. It's React, a JavaScript framework developed by engineers at Facebook. It can be used for single-page applications or to organize JavaScript for interactive features in ordinary Rails applications. Unlike AngularJS or Ember.js, React only manages views, not connections to databases or routing of requests, so it is not a full-stack framework, just a framework for the view layer. React's approach to building web pages is abstract and complex. But React is a good choice for complex interactive features, if you're determined to avoid JavaScript soup in your Rails application.

Ultimately, your decision to use JavaScript, and how much, depends on the kind of web application you want to build. If you want to build a product catalog, with many simple pages populated by a database, a full-stack Rails application may be all you need, perhaps with a few jQuery plugins. If your application is like Facebook, with lots of interactive features and many different pages, React may be a good choice as part of a full-stack Rails application. If you want to deliver a service such as live stock-market charts, you may want to build a complex single-page application using AngularJS or Ember.js, plus IOS and Android apps, connected to a Rails API server.

If you're going to use JavaScript, either full-stack, or combined with Rails, use a JavaScript framework to avoid JavaScript soup. React could be a good place to start.

You'll have a better idea of the importance of Rails if you understand "What is Rails?" Let's look at that next.

Chapter 4

What is Rails?

Rails is a *library*, or collection of code, plus *structures and conventions* for building a web application.

Technically, it is a package library (a RubyGem), that is installed using the operating system command-line interface, and adds functionality to the Ruby language.

Rails provides structures and conventions for web development, so less code is required to build a web application. All Rails applications use the same structures, providing consistency among applications. Common and well-known conventions make it easy for developers to collaborate on Rails applications and share improvements and add-on code libraries with a wide community.

The structures and conventions of Rails are codified as the Rails API (the application programming interface, or directives that control the code). The Rails API is documented online and described in books, articles, and blog posts. Learning Rails means learning how to use the Rails conventions and its API.

Rails as a Community

Rails, in a larger sense, is more than a software library and an API. Rails is the central project of a vast community that produces software libraries that simplify the task of building complex websites. Members of the Rails community share many core values, often use the same tools, and support each other with an informal network that is built on volunteerism. Overlapping the informal community is an economic network that includes jobs, recruiters, consulting firms, conferences, businesses that build websites with Rails, and investors that fund startups. Rails is popular among web startups, significantly because the pool of open source software libraries (RubyGems, or "gems") makes it possible to build complex sites quickly.

This, then, is Rails: a software library and an API, plus a community of enthusiastic developers. But to learn to use Rails, you'll need to consider it from additional points of view.

Six Perspectives on Rails

To really understand Rails, and succeed in building Rails applications, we need to consider Rails from six additional perspectives.

A parable from the Indian subcontinent describes six blind men who encounter an elephant. The blind man who feels a leg says the elephant is like a pillar; the one who feels the tail says the elephant is like a rope; the one who feels the belly says the elephant is like a wall; and so on. Like six blind men encountering an elephant, it can be difficult to understand Rails unless you look at it from multiple points of view.

Here are six different ways of looking at Rails.

Web Browser Perspective

From the **perspective of the web browser**, Rails is simply a program that generates HTML, CSS, and JavaScript files. These files are generated dynamically. You can't see the files on the server side but you can view these files by using the web developer tools that are built in to every browser. Later you'll examine these files when you learn to troubleshoot a Rails application.

Programmer Perspective

From the **perspective of a programmer**, Rails is a set of files organized with a specific structure. The structure is the same for every Rails application; this makes it easy to collaborate with other Rails developers. We use text editors to edit these files to make a web application.

Software Architect Perspective

From the **perspective of a software architect**, Rails is a structure of *abstractions* that enable programmers to collaborate and organize their code. Thinking in abstractions means we group things in categories and analyze relationships. Conceptual categories and relationships can be made "real" in code. Software programs are built of "concepts made real" that are the moving parts of a software machine.

Notice that we are talking about abstractions, using language like a philosopher. Most tutorials give you step-by-step instructions, like following a recipe. You'll get that in Book Two in this series. But to really understand programming, you should grasp that we are working with "abstractions made real." The area of your brain that follows instructions is distinct from the area that thinks abstractly. To be a programmer, you'll need both parts of the brain working together. Don't worry; I'll explain the abstractions in terms anyone can understand.

To a software architect, *classes* are the basic parts of a software machine. A class can represent something in the physical world as a collection of various attributes or properties (for example, a User with a name, password, and email address). Or a class can describe another abstraction, such as a Number, with attributes such as quantity, and behavior, such as "can be added and subtracted." You'll get a better grasp of classes in Book Two when you learn "Just Enough Ruby."

To a software architect, Rails is a pre-defined set of classes that are organized into a higher level of abstraction known as an API, or *application programming interface*. The Rails API is organized to conform to certain widely known *software design patterns*. You'll become familiar with these abstractions as you build a Rails application. Later in the tutorial, we'll learn about the *model–view–controller* design pattern. As a beginner, you will see the MVC design pattern reflected in the file structure of a Rails application. That's where you will first see "abstractions made real."

Gem Hunter Perspective

We can look at Rails from the **perspective of a gem hunter**. Rails is popular because developers have written and shared many software libraries (RubyGems, or "gems") that provide useful features for building websites. We can think of a Rails application as a collection of gems that provide basic functionality, plus custom code that adds unique features for a particular website. Some gems are required by every Rails application. For example, database adaptors enable Rails to connect to databases. Other gems are used to make development easier, for example, gems for testing that help programmers find bugs. Still other gems add functionality to the website, such as gems for logging in users or processing credit cards. Knowing what gems to use, and why, is an important aspect of learning Rails. Book Two will show you how to build a web application using some of the most commonly used gems.

Time Traveler Perspective

We can also look at Rails from the **perspective of a time traveler** in order to understand the importance of *software version control*. Specifically, we use the Git revision control system to record a series of snapshots of your project's filesystem. Git makes it easy to back up and recover files; more importantly, Git lets you make exploratory changes, trying out code you may decide to discard, without disturbing work you've done earlier. You can use Git with GitHub, a popular "social coding" website, for remote backup of your projects and community collaboration. Git can keep multiple versions ("branches") of your local code in sync with a remote GitHub repository, making it possible to collaborate with others on open source or proprietary projects. Strictly speaking, Git and GitHub are not part of Rails (they are tools that can be used on any development project). And there are several other version control systems that are used in open source development. But a professional Rails developer uses Git and GitHub constantly on any real-world Rails project.

Tester Perspective

Finally, we can consider a Rails application from the **perspective of a tester**. Software testing is part of Rails culture; Rails is the first web development platform to make testing an integrated part of development. Before Rails, automated testing was rarely part of web development. A web application would be tested by users and (maybe) a QA team. If automated tests were used, the tests were often written after the web application was largely complete. Rails introduced the discipline of Test-Driven Development (TDD) to the wider web development community. With TDD, tests are often written *before* any implementation coding. It may seem odd to write tests first, but for a skilled TDD practitioner, it brings coherence to the programming process. First, the developer will give thought to what needs to be accomplished and think through alternatives and edge cases. Second, the developer will have complete test coverage for the project. With good test coverage, it is easier to *refactor*, rearranging code to be more elegant or efficient. Running a test suite after refactoring

provides assurance that nothing inadvertently broke after the changes. TDD is seen as a necessary skill of an experienced Rails developer. Book Two will introduce you to the basic concepts of test-driven development and show you how to write simple tests.

You've seen Rails from six different perspectives. You understand Rails is a software library plus an API, as well as a community of developers. Now let's dive deeper and consider how Rails fits in to a larger *technology stack* and how it can vary within the stack.

Understanding Stacks

To understand Rails from the perspective of a professional Rails developer, you'll need to grasp the idea of a *technology stack* and recognize that Rails can have more than one stack.

Full Stack

A technology stack is a set of technologies or software libraries that are used to develop an application or deliver web pages. "Stack" is a term that is used loosely and descriptively. It is a collection of technologies that fit together, interacting with each other, and providing different types of services. We say "stack" because we use different software systems or software libraries as if they were building blocks or bricks deployed in layers.

Often we consider a stack as layers added to the operating system. For a web application, we need a web server, a database, a programming language, and perhaps additional software libraries.

Earlier we learned that full-stack developers have skills to work with operating systems, web servers, databases, web applications, and JavaScript programs in a web browser. That's a formidable stack of technologies.

There is no organization that tells you what building blocks you must use. As a

technologist, your choice of stack reflects your experience, values, and personal preference, just like religion or favorite beverage.

For example, Mark Zuckerberg developed Facebook in 2004 using the LAMP application stack:

- Linux (operating system)

- Apache (web server)

- MySQL (database)

- PHP (programming language)

Zuckerberg chose an operating system, a web server, a database, and a programming language with which he was comfortable. In 2004, Rails was not well known and Zuckerberg chose to implement his ideas using PHP without any additional web application software library.

For this tutorial, your application stack will be:

- Mac OS X, Linux, or Windows

- WEBrick (web server)

- SQLite (database)

- Ruby on Rails (language and framework)

You'll use an operating system that's familiar to you, a basic web server that comes with Ruby, a database that's preinstalled with Mac or Linux operating system, the Ruby language, and the Rails web application software library.

Rails Stacks

Sometimes when we talk about a stack, we only care about part of a larger stack. For example, a Rails stack includes the gems we choose to add features to a website or make development easier. When we select the gems we'll use for a Rails application, we're choosing a stack or layers of services we need just for a web application.

Sometimes the choice of components is driven by the requirements of an application. At other times, the stack is a matter of personal preference. Just as craftsmen and aficionados debate the merits of favorite tools and techniques in any profession, Rails developers avidly dispute what's the best Rails stack for development.

The company 37signals, where the creator of Rails works, uses this Rails stack:

- ERB for view templates

- MySQL for databases

- Minitest for testing

It is not important (at this point) to know what the acronyms mean (we'll learn later).

Another stack is more popular among Rails developers:

- Haml for view templates

- PostgreSQL for databases

- RSpec for testing

We'll learn later what the terms mean. For now, just recognize that parts of the Rails framework can be swapped out, just like making substitutions when you

order from a menu at a restaurant. The Rails stack can vary, and it is part of a larger stack that includes an operating system, web server, and database.

You can learn much about Rails by following the experts' debates about the merits of a favorite stack. The debates are a source of much innovation and improvement for the Rails framework. In the end, the power of the crowd prevails; usually the best components in the Rails stack are the most popular.

The proliferation of choices for the Rails stack can make learning difficult, particularly because the components used by many leading Rails developers are not the components used in many beginner tutorials. In this tutorial, we stick to solid ground where there is no debate. In more advanced tutorials, we'll explore stack choices and choose components that are most often used by professional developers.

Chapter 5

Why Rails?

Before you start building an application with Rails, it may help to know why developers like using Rails. This chapter looks at the history of Rails, its organizing principles, and the reasons for its popularity. First, though, we'll consider Ruby, the language used for Rails.

Why Ruby?

Ruby is a programming language, created 20 years ago by Yukihiro "Matz" Matsumoto. By most measures of programming language popularity, Ruby ranks among the top ten, though usually as tenth (or so) in popularity, and largely due to the popularity of Rails. Like Java or the C language, Ruby is a general-purpose programming language, though it is best known for its use in web programming.

In a podcast from This Developer's Life and in an interview from 2005, David Heinemeier Hansson, the creator of Rails, describes building an online project management application named BaseCamp in 2004. He had been using the PHP programming language because he could get things done quickly but was frustrated because of a lack of abstraction and frequently repetitive code that made PHP "dirty." Hansson wanted to use the "clean" software enginering ab-

stractions supported in the Java programming language but found development in Java was cumbersome. He tried Ruby and was excited about the ease of use (he calls it pleasure) he found in the Ruby language.

Ruby is known among programmers for a terse, uncluttered syntax that doesn't require a lot of extra punctuation. Compared to Java, Ruby is streamlined, with less code required to create basic structures such as data fields. Ruby is a modern language that makes it easy to use high-level abstractions such as metaprogramming. In particular, metaprogramming makes it easy to develop a "domain specific language" that customizes Ruby for a particular set of uses (Rails and many gems use this "DSL" capability).

Ruby's key advantage is RubyGems, the package manager that makes it easy to create and share software libraries (gems) that extend Ruby. RubyGems provides a simple system to install gems. Anyone can upload a gem to the central RubyGems website, making the gem immediately available for installation by anyone. The RubyGems website is where you'll obtain the most recent version of Rails. And it is where you will obtain all the gems that help you build complex websites.

Ruby has several disadvantages (at least when programmers want to argue). Its processing performance is slow relative to C++ or Java. The execution speed of a language is seldom important, though, relative to the benefits gained by programmer productivity and the general level of performance required by most websites. For websites that require lots of simultaneous activity, Ruby is not well-suited to the sophisticated software engineering required to execute simultaneous activity efficiently (standard Ruby lacks "parallelism", though some versions support it). Lastly, some programmers complain that Ruby programs (and especially Rails) contain "too much magic" (that is, complex operations that are hidden behind simple directives). These concerns haven't stopped Rails from becoming a popular web development platform.

Why Rails?

Rails is popular and widely used because its conventions are pervasive and astute. Any web application has complex requirements that include basic functions such as generating HTML, processing form submissions, or accessing a database. Without a web application development framework, a programmer has a mammoth task to implement all the required infrastructure. Even with a web application development framework, a programmer can take an idiosyncratic approach, building something that no one else can easily take apart and understand. The singular virtue of Rails is that Heinemeier Hansson, and the core team that joined him, decided that there is one best way to implement much of the infrastructure required by a web application. Many of the implementation decisions appear arbitrary. In fact, though Heinemeier Hansson is often lambasted as autocratic in his approach to improving Rails, the Rails API reflects deep experience and intelligence in implementing the requirements of a web application development framework. The benefit is that every developer who learns the "Rails way" produces a web application that any other Rails developer can unravel and understand more quickly than if they encountered idiosyncratic code without as many conventions. That means collaboration is easier, development is quicker, and there's a larger pool of open source libraries to enhance Rails.

The advantage of establishing conventions might seem obvious, but when Rails was released in 2004, web development was dominated by PHP, which lent itself to idiosyncratic code produced by solo webmasters, and Java frameworks such as Struts, which were often seen as burdened by an excess of structure. Other frameworks, such as Apple's WebObjects, Adobe's ColdFusion, and Microsoft's .NET Framework, were in wide use but the frameworks were products controlled by the companies and built by small teams, which tended to restrict innovation. Today PHP, Java frameworks, and .NET remain popular, largely among solo webmasters (PHP), enterprise teams (Java), and Windows aficionados (.NET) but Rails has become very popular and has influenced development of other server-side frameworks.

The design decisions that went into the first version of Rails anchored a vir-

tuous circle that led to Rails's growth. Within the first year, Rails caught the attention of prominent software engineers, notably Martin Fowler and Dave Thomas (proponents of agile software development methodologies). Rails is well-matched to the practices of agile software development, particular in its emphasis on software testing and "convention over configuration." The interest and advocacy of opinion leaders from the agile camp led to greater visibility in the wider open source community, culminating in a keynote lecture by Heinemeier Hansson at the 2005 O'Reilly Open Source Convention. Because Rails was adopted by software engineers who are influencers and trend setters, it is often said that Rails is favored by "the cool kids." If that is so, it is largely because Rails is well-suited to software engineering practices that are promoted by thought leaders like Fowler and Thomas.

Rails Guiding Principles

The popularity of Rails is an outgrowth of the Rails "philosophy" or guiding principles. If you read blogs by Rails developers, you'll often see references to these principles. Understanding these principles will help you make sense of Rails or, at least, some of the debates on developer blogs.

Rails is Opinionated

In the mid-1990s, web applications were often written in Perl, a programming language that promised, "There's more than one way to do it." Perl is a prime example of "non-opinionated" software; there's no "right way" or "best way" to solve programming problems in Perl. Famously, Perl's documentation states, "In general, [Perl's built-in functions] do what you want, unless you want consistency."

In contrast, Rails is said to be "opinionated." There is a "Rails way" for many of the problems that must be solved by a web application developer. If you follow the Rails conventions, you'll have fewer decisions to make and you'll

find more of what you need is already built. The benefit is faster development, improved collaboration, and easier maintenance.

Rails is Omakase

Omakase is a Japanese phrase that means "I'll leave it to you." Customers at sushi restaurants can order omakase, entrusting the chef to make a pleasing selection instead of making their own à la carte choices. In a famous essay Heinemeier Hansson declared Rails is Omakase, and said, "A team of chefs picked out the ingredients, designed the APIs, and arranged the order of consumption on your behalf according to their idea of what would make for a tasty full-stack framework.... When we, or in some cases I — as the head chef of the omakase experience that is Rails — decide to include a dish, it's usually based on our distilled tastes and preferences. I've worked in this establishment for a decade. I've poured well in the excess of ten thousand hours into Rails. This doesn't make my tastes right for you, but it certainly means that they're well formed."

Understanding that Rails is omakase means accepting that many of the opinions enshrined in the Rails API are the decisions of a Benevolent Dictator for Life, informed by discussion with other developers who have made significant contributions to the Rails code base. For the most part, Heinemeier Hansson's "opinions" will serve you well.

Convention Over Configuration

"Convention over configuration" is an example of Rails as "opinionated software." It is an extension of the concept of a default, a setting or value automatically assigned without user intervention. Some software systems, notably Java web application frameworks, need multiple configuration files, each with many settings. For example, a configuration file might specify that a database table named "sales" corresponds to a class named "Sales." The configuration

file permits flexibility (a developer can easily change the setting if the table is named "items_sold"). Instead of relying on extensive configuration files, Rails makes assumptions. By convention, if you create a model object in Rails named "User," it will save data to a database table named "users" without any configuration required. Rails will also assume the table name is plural if the class name is singular.

"Convention over configuration" means you'll be productive. You won't spend time setting up configuration files. You'll spend less time thinking about where things go and what names to assign. And, because other developers have learned the same conventions, it is easier to collaborate.

Don't Repeat Yourself

Known by the acronym DRY, "Don't Repeat Yourself" is a principle of software development formulated by Andy Hunt and Dave Thomas and widely advocated among Rails developers. In its simplest form, it is an admonition to avoid duplication. When code is duplicated, an application becomes more complex, making it more difficult to maintain and more vulnerable to unintended behavior (bugs). The DRY principle can be extended to development processes as well as code. For example, manual testing is repetitive; automated testing is DRY. Software design patterns that introduce abstraction or indirection can make code more DRY; for example, by eliminating repetitive if-then logic.

Code reuse is a fundamental technique in software development. It existed long before Andy Hunt and Dave Thomas promoted the DRY principle. Rails takes advantage of Ruby's metaprogramming features to not just reuse code but eliminate code where possible. With a knowledge of Rails conventions, it's possible to create entire simple web applications with only a few lines of code.

Where Rails Gets Complicated

It helps to understand the guiding principles of Rails. But it's even more helpful to know how (and why) Rails is complicated by departures from the guiding principles.

When Rails has No Opinion

As you gain experience with Rails, you may encounter areas where Rails doesn't state an opinion. For example, for years there was no "official" approach to queueing background jobs. (Tasks that take time, such as contacting a remote server, are best handled as "background jobs" that won't delay display of a web page.) Fortunately, by 2015, the Rails core maintainers released the ActiveJob feature which implemented queueing. Much of the lively debate that drives development of new versions of Rails is focused on thrashing out the "opinions" that eventually will be enshrined in the Rails API.

Omakase But Substitutions Are Allowed

Implicit in the notion of "Rails is omakase" is an understanding that "substitutions are allowed." Most of Heinemeier Hansson's preferences are accepted by all Rails developers. However, many experienced developers substitute items on the menu at the Rails café. This has led to the notion that Rails has Two Default Stacks, as described in an essay by Steve Klabnik. Professional developers often substitute an alternative testing framework or use a different syntax for creating page views than the "official" version chosen by Heinemeier Hansson. This complicates learning because introductory texts often focus on the omakase selections but you'll encounter alternatives in blog posts and example code.

Conventions or Magic?

One of the joys of programming is knowing that everything that happens in an application is explained by the code. If you know where to look, you'll see the source of any behavior. For a skilled programmer, "convention over configuration" adds obscurity. Without a configuration file, there is no obvious code that reveals that data from a class named "Person" is saved to a datatable named "people." As a beginner, you'll simply accept the magic and not confound yourself trying to find how it works. It's not always easy to learn the conventions. For example, you may have a User object and a "users" datatable. Rails will also expect you to create a "controller object." Should it be named "UserController" (singular) or "UsersController" (plural)? You'll only know if you let Rails generate the code or you pay close attention to tutorials and example code.

DRY to Obscurity

The risk that "convention over configuration" leads to obscurity is compounded by the "Don't Repeat Yourself" principle. To avoid repetitive code, Rails often will offer default behavior that looks like magic because the underlying implementation is hidden in the depths of the Rails code library. You can implement a simple web application with only a few lines of custom code but you may wonder where all the behavior comes from. This can be frustrating when, as a beginner, you attempt to customize your application to do more than what's shown in simple tutorials.

In the next chapter, we'll consider some of the challenges that make it difficult to learn and use Rails.

Chapter 6

Rails Challenges

Rails is popular. Rails is powerful. But Rails isn't easy to learn.

You may have heard of a psychological phenomenon called "resistance." When we struggle with something new, or must adapt to the unfamiliar, we resist. We get discouraged. We complain. Sometimes we feel we should quit.

This chapter is here to help with your resistance.

Its purpose is to acknowledge that, yes, *Rails can be difficult*.

Tens of thousands of people are successfully using Rails. I'll hazard a guess that none are significantly smarter, more motivated, or a better student than you. Perhaps some of them had more time to study or better access to mentors, but these factors simply accelerate the speed of learning Rails. If you get discouraged, or think Rails is too hard, recognize that you are encountering your own resistance, not any genuine limitation. Take a break, set aside your learning materials, and come back when your natural curiosity and eagerness has returned.

Sometimes resistance attaches to imaginary problems (like "I'm not smart enough"). Just as often, resistance attaches to real problems, but magnifies them into insurmountable obstacles ("Rails is impossible to use on Windows!"). The best way to overcome these obstacles is to acknowledge the resistance, investigate

the obstacle, and seek support from peers.

This chapter describes some of things that make Rails difficult.

These Rails challenges are obstacles, but other people overcame them. You can, too.

A List of Challenges

This list is incomplete. If you've encountered a Rails challenge that isn't listed here, email me at daniel@danielkehoe.com and I will add your suggestion to the next revision of the book.

It is difficult to install Ruby.

The installation process for Ruby on Rails is more difficult than downloading and installing any consumer software applications. You are setting up a development environment and you need system software as well as Ruby. Depending on what you've done before, you may have altered your system, introducing potentials for conflicts. Book Two provides links to good installation guides in the "Get Started" chapter. But installation instructions can't accommodate the specific configuration of your computer. Sometimes you just have to look for someone to help. You can also use a hosted development environment, like Cloud9 or Nitrous.io.

Rails is a nightmare on Windows.

Windows is very popular, so why is it difficult to develop with Rails on Windows? It seems the Rails community has a bias against Windows. It does, and there's a reason. Rails is an open source project. Most open source developers use Unix-based system tools. It is difficult and time-consuming to convert Unix-based system tools to the Microsoft Windows operating system. Open

source developers prefer to spend their time maintaining and improving their Unix-based projects. And expert Windows developers are seldom interested in porting Unix-based system tools to Windows. So system utilities such as RVM are not available for Windows. And developers who create gems are seldom interested in spending time to solve the problems that arise when code has to be adapted for the idiosyncrasies of the Windows platform. This situation is not going to change, so you have to make a choice. Stay with Windows or get comfortable with Unix-based systems.

Why do I have to learn Git? It is difficult.

Real software development requires version control and Git is the standard tool for Rails developers. If all you do is build applications as a classroom exercise, you don't need to learn Git. You can skip all the parts of the book that mention Git. But sooner or later, if you start doing real projects, you'll need Git. Simple Git commands are not difficult to learn. When you've developed your skills and confidence you can learn the more advanced Git functions, such as branching.

Why worry about versions?

For simple projects you don't need version management. My books introduce version management and prepare you to handle version conflicts. As you tackle more complex projects, and as new versions of Rails are released, you'll face version issues and version management will be helpful.

Do I really need to learn about testing?

For student projects, no, you don't need to learn about testing. But as soon as money or reputation is at stake on a project, you'll need to begin using test-driven development. Book Two introduces TDD but you'll need intermediate-level tutorials to develop proficiency. Once you've grasped the basics, testing

will become easy, and it actually is fun and satisfying.

Rails error reporting is cryptic.

Actually, Rails error reporting is quite good. Stack traces are detailed and error messages are descriptive. Beginners have a problem because the stack traces and error messages provide a technical analysis of a problem in terms that an experienced developer can understand. If error reporting was "simplified" it might not be as intimidating but it would not as accurate. It's up to you to gain enough knowledge to understand the error messages. Finally, the error reporting mechanism can point you to the line in your code that triggers a problem, but it can't know what you trying to do, or describe the error in anything but technical terms.

There is too much magic.

The Rails "convention over configuration" principle leads to obscurity. Default behavior often looks like magic because the underlying implementation is hidden in the depths of the Rails code library. If you like to know how things work, this can be frustrating. You really have only two choices when you encounter Rails magic. You can take time to dig into the source code. If you do so, you'll encounter frustration as you encounter complex and sophisticated code, but you may also improve your understanding and skill as a Ruby programmer. Or you can take on faith that "it just works." Often, you just need to use the convention several times in different projects to get comfortable with the magic and stop worrying that you don't fully understand it.

It is difficult to grasp MVC and REST.

Even if you learn that the acronyms mean *model-view-controller* and *representational state transfer*, MVC and REST are abstract concepts. If you simply

follow a tutorial, the author will show you how to build an application that uses MVC or REST, but you won't see any alternative, or understand why MVC or REST are best practices. When it is time to build your own application, if you don't understand the importance of *separation of concerns* you won't be sure how to structure your application. Understanding software architecture requires abstract reasoning, imagination, and experience, which takes time. It is difficult to grasp but approach it with curiosity, seek explanations, and you'll grasp it soon enough.

Rails contains lots of things I don't understand.

If you look at the Rails directory structure, you'll see many files and folders. If you look at the Rails API, or pick up a Ruby tutorial, you'll also see code that is unfamiliar. Book Two describes some of what you see. As you build more applications, you will gain proficiency and master more of Rails and Ruby. Yet even as you gain mastery of Rails, there will be aspects that remain unfamiliar. Don't let the sheer complexity stop you. The truth is, you don't have to know "all" of Rails or Ruby to build web applications.

There is too much to learn.

Very true. To be a full-stack web developer you need to know HTML, JavaScript, CSS, Ruby, testing, databases, and much, much more. You might think that developers who started ten years ago have an advantage because there wasn't as much to learn when they started. But today there are many more high-quality tutorials and educational programs to accelerate your learning. And resources like Google and Stack Overflow have many more answers to questions. As the knowledge domain has grown, so have the learning resources. You don't have to learn everything. Get a foundation in the basics and then dive deep as a specialist in an area that appeals to you.

It is difficult to find up-to-date advice.

Rails has been around since 2004 with major new versions released every two years. Chances are, answers to questions you find on Stack Overflow or Google were written for an older version of Rails. There is no easy way to determine if the answer is out of date. A particular aspect of Rails may have changed—or not. Even worse, the answer may work, but there may be a better way that reflects current best practices. To filter the outdated in Google, use the "Search Tools" options for specifying a timeframe. Look closely at the date of a blog posting or Stack Overflow answer. Try to find a newer answer. Usually, if there are a series of answers and things have changed, you'll see the current best answer. If you're uncertain, don't be shy about posting your question to Stack Overflow. More importantly, make it your business to keep up with the community, reading Peter Cooper's Ruby Weekly email newsletter or his daily RubyFlow site.

It is difficult to know what gems to use.

There are so many gems available for Rails. Some add useful features, like tagging or a mailing list API. Some are basic, such as gems for a database or front-end framework. Even among basic gems, Rails offers choices. Which are best? The Ruby Toolbox can help, but mostly you will find guidance from looking at example projects and noticing what other developers are using. There's wisdom in the crowd.

Rails changes too often.

If you look at the Ruby on Rails Release History you'll see there is a new major release approximately every 1.5 years. Each major release is well tested and relatively free of bugs. But new features or new approaches often require rewrites of older applications. Commercial software products often make a priority of keeping the API consistent over time. That's not Rails. Rails is an

open source project and the core team embraces innovation. The maintainers expect that you'll keep up with changes.

It is difficult to transition from tutorials to building real applications.

Copying and pasting from tutorials is a good way to begin learning Rails. But you'll only become a skilled Rails developer when you build something that is not shown in a tutorial. The first few hours (or days) when you start building a custom application can be very difficult. Focus on the basics that are described in this book. Start with user stories. Build pieces that you know how to do. Look for code samples on blogs or GitHub or Stack Overflow. Try "spikes," little experiments that test ideas for implementation. Seek advice from peers or mentors. At first it will be slow going. But you will pick up momentum. The chapter, "Crossing the Chasm," will provide specific strategies to help.

I'm not sure where the code goes.

If you follow tutorials, you'll learn "where the code goes" with the model–view–controller design pattern. With a sense of the request-response cycle, RESTful actions in the controller, and a few guidelines such as "skinny controller, fat model" you'll be able to build intermediate-level Rails applications. Front-end code, particularly JavaScript, can be difficult because not a lot has been published about Rails best practices. In particular, the Rails asset pipeline can be confusing for anyone who has done front-end development without Rails. If you don't know what you're supposed to do, do whatever works, then look for someone who can help you by providing a code review.

People like me don't go into programming.

Until recently in most countries, most Rails developers have been young men with an engineering background. For people who don't fit the stereotypical profile, it can be hard to find role models or peers who demonstrate that Rails is something everyone can learn. The challenge can be subtle, as when you have the feeling that maybe if you were different you'd find it easier to make progress. Or the challenge can be overt, when behavior of fellow students or co-workers is disturbing or hurtful (often they don't even know!). Lack of diversity, and the cluelessness that accompanies it, is unfortunate in the Rails community. But many people are working to make the community more welcoming and inclusive. Organizations such as Rails Girls and Railsbridge are creating more diversity in the community. You may find support from peers there to affirm that you, too, are entitled to knowledge and success.

These are some of the challenges you will face in learning and using Rails. Next, we will look at how to get help.

Chapter 7

Get Help When You Need It

Sometimes I'm asked, "Where's the Rails manual?" There isn't one. No single document tells you how to use Rails. Instead, there's a wealth of documentation that describes various aspects of Rails. Book Two contains all the documentation you need to build an application. But before you start, I'd like to suggest additional resources that will be helpful.

Getting Help With the Book

Let's consider what to do if you encounter problems as you build the tutorial application in Book Two.

If you are in a classroom, or studying in a group, ask a peer to look at your problem. Most problems are caused by simple typos or formatting errors. Your classmate may see what you overlooked.

The code in Book Two was tested by many people and worked flawlessly at the time this was written. The learn-rails example application on GitHub serves as a "reference implementation" if you have problems. The example application is updated more frequently than the published tutorial.

Stack Overflow provides a question-and-answer forum for readers of my books.

As the author of this book, I can't solve your individual problems or help you directly by email. If I did, I would not have time to create the tutorials that benefit so many people. However, I watch for questions on Stack Overflow. Everyone benefits when solutions are made public.

- tag your questions on Stack Overflow with **railsapps** for extra attention

If your questions aren't answered on Stack Overflow, try the Reddit forum:

- /r/learnrubyonrails

The Reddit forum has its own Slack channel for real-time messaging:

- learnrubyonrails-slack.herokuapp.com/

I sincerely hope you won't encounter obstacles as you build the tutorial application. Thousands of beginners have successfully completed the book and, unless a gem has recently changed, you should have no problem.

Now let's consider where to look for help when you are working on your own Rails projects.

Getting Help With Rails

What will you do when you get stuck?

"Google it," of course. But here's a trick to keep in mind. Google has options under "Search tools" to show only recent results from the past year. Use it to filter out stale advice that pertains only to older versions of Rails.

Stack Overflow is as important as Google for finding answers to programming problems. Stack Overflow answers are often included in Google search results, but you can go directly to Stack Overflow to search for answers to your

questions. Like Google, answers from Stack Overflow are helpful if you check carefully to make sure the answers are recent. Also be sure to compare answers to similar questions; the most popular answer is not always the correct answer to your particular problem.

Requests for advice (especially anything that provokes opinions) are often rejected on Stack Overflow. Instead, try Reddit for advice or recommendations. You'll find discussion forums ("subreddits") devoted to Rails and Ruby. You can also visit the Quora question-and-answer site for topics devoted to Rails and Ruby.

References

Here are suggestions for the most important references.

If you feel overwhelmed by all the links, remember that you can use Book Two to build the tutorial application without any additional resources. Right now, it's important to know additional help is available when you need it.

RailsGuides

The Rails Guides are Rails's official documentation, written for intermediate-level developers who already have experience writing web applications. The Rails Guides are an excellent reference if you want to check the correct syntax for Rails code. You'll be able to use the Rails Guides after completing this tutorial.

Cheatsheets

Tobias Pfeiffer has created a useful Rails Beginner Cheat Sheet that provides a good overview of Rails syntax and commands.

Even better than a cheatsheet, for Mac users, is an application named Dash that offers fingertip access to reference documentation for Ruby, Rails, HTML, CSS, JavaScript, and many other languages and frameworks.

API Documentation

The API documentation for Ruby and Rails shows every class and method. These are extremely technical documents (the only thing more technical is reading the source code itself). The documents offer very little help for beginners, as each class and method is considered in isolation, but there are times when checking the API documentation is the only way to know for certain how something works.

- Rails Documentation - official API docs

- Rails Searchable API Doc - alternative interface for the API docs

- apidock.com/rails - Rails API docs with usage notes

- apidock.com/ruby - Ruby API docs with usage notes

- Omniref - Ruby and all gem API docs with questions and answers

Omniref is the best place to ask questions about anything in the Ruby or Rails API.

I recommend Dash as a tool to look up classes, modules, and methods in Ruby and Rails. Dash is a Mac OS X app; use Zeal on Linux. Dash and Zeal run offline (they don't need an Internet connection) so you can use them anywhere.

Meetups, Hack Nights, and Workshops

I'd like to urge you to find ways you can work with others who are learning Rails. Peer support is really important when you face a challenge and want to

overcome obstacles.

Most large urban areas have meetups or user group meetings for Rails developers. Try Meetup.com or google "ruby rails (my city)". The community of Rails developers is friendly and eager to help beginners. If you are near a Rails meetup, it is really worthwhile to connect to other developers for help and support. You may find a group that meets weekly for beginners who study together.

Local user groups often sponsor hack nights or hackathons which can be evening or weekend collaborative coding sessions. You don't have to be an expert. Beginners are welcome. You can bring your own project which can be as simple as completing a tutorial. You will likely find a study partner at your level or a mentor to help you learn.

If you are a woman learning Rails, look for one of the free workshops from RailsBridge or Rails Girls. These are not exclusively for women; everyone considered a "minority" in the tech professions is encouraged to participate; and men are included when invited by a woman colleague or friend.

Pair Programming

Learning to code is challenging, especially if you do it alone. Make it social and you'll learn faster and have more fun.

There's a popular trend in the workplace for programmers to work side-by-side on the same code, sharing a keyboard and screen. It's effective, both to increase productivity and to share knowledge, and many coders love it. When programmers are not in the same office, they share a screen remotely and communicate with video chat.

Look for opportunities to pair program. It's the best way to learn to code, even if your pairing partner is only another beginner. Learn more about pair programming on the site pairprogramwith.me and find a pairing partner at codermatch.me or letspair.net.

Remote pair programming requires tools for screen sharing and video chat. Pairing sessions often use:

- Google+ Hangouts

- Screenhero

- Floobits

- Cloud9 IDE

- Nitrous.io

More tools are emerging as remote pair programming becomes popular.

Pairing With a Mentor

By far, the best way to learn is to have a mentor at your side as you undertake a project. That is an opportunity that is seldom available, unless you've been hired to be part of a team in a company that encourages pair programming.

You can try RailsMentors, a network of volunteer mentors offering free help.

If you can pay for help, find a mentor using HackHands or AirPair. Market rates are expensive for a student, obviously, but if you are learning on the job or building an application for your own business, connecting online with a mentor might be a godsend.

AirPair connects developers for real-time help using video chat and screen sharing. Experts set their own rate and the site matches you according to your budget. Expect to pay market rates for consulting ranging from USD $40 per hour to $150 per hour or more.

HackHands promises to instantly connect you with a qualified expert at a cost of one dollar per minute for mentorship using video chat and screen sharing.

Code Review

Code review is an essential part of the development process. There's always more than one way to implement a feature, and some ways are better than others, but you may not know it unless you ask someone to look at your code. When you pair with a mentor, you get the benefit of code review. But even if you don't have a mentor, you can get code review online. StackExchange, the parent of StackOverflow, has a free site for code review, and a new service promises code review as a service:

- codereview.stackexchange.com

- devinput.io

Expert code review will accelerate your learning faster than anything else.

Knowing where to go for help is important; it is just as important to stay current.

Staying Up-to-Date

Rails changes frequently and its community is very active. Changes to Rails, expert blog articles, and new gems can impact your projects, even if you don't work full-time as a Rails developer. Consequently, I urge you to stay up-to-date with news from the community.

For daily news about Rails, check Peter Cooper's RubyFlow site which lists new blog posts from Rails developers each day.

I urge you to sign up for two weekly email newsletters:

- Ruby Weekly

- Green Ruby News

Another weekly email newsletter is more technical, and focused on code arriving in the next version of Rails:

- This Week in Rails

If you like podcasts, check out Ruby Rogues and Envy Labs's Ruby5.

Finally, you can follow @rails_apps on Twitter for news about the RailsApps project.

Chapter 8

Plan Your Product

Tutorials from other authors focus only on coding. But Rails developers do more than code. Software development is a process that begins with planning and ends with analysis and review. Coding, testing, and deployment is at the core but you'll need to learn about the entire process to succeed professionally. That's why we look at product planning and project management.

For this beginning tutorial, we'll introduce concepts about product planning and project management that you will encounter as a Rails developer.

Product Owner

On your project, who is the *product owner*?

The product owner is the advocate for the customer, making sure that the team creates value for the users.

If you are a solo operator, you are the one who will decide what features and functionality will be included in your application. But if you're part of a team, either in a startup, as a consultant, or in a corporate setting, it may not be clear who has responsibility for looking at the application from the point of view of the application user. Someone must decide which features and functionality

are essential and which must be left out. We call this *managing scope* and combating *feature creep*.

It's important to assign a product owner. Without a product owner in charge, tasks remain vague and developers have difficulty making progress.

In large organizations, a product owner may be a product manager or a project manager. A product owner usually is not a management executive (though there will likely be an executive sponsor). Everyone on the team — including management, developers, and stakeholders — should agree to designate a product owner and give that person authority to define features and requirements.

User Stories

A product owner's principal tool for product planning is the *user story*.

In the past, when software engineering primarily served government or large corporations, product planning started with *requirements gathering* defined as *use cases*, and culminated in a *requirements specification*. User stories are a faster, more flexible approach to product planning that originated with an approach called Agile software development.

User stories are a way to discuss and describe the requirements for a software application. The process of writing user stories helps a product owner identify all the features that are needed for an application. Breaking down the application's functionality into discrete user stories helps organize the work and track progress toward completion.

User stories are often expressed in the following format:

```
As a
I want
In order to
```

Here is an example:

```
*Join Mailing List*
As a visitor to the website
I want to join a mailing list
In order to receive news and announcements
```

A typical application has dozens of user stories, from basic sign-in requirements to the particular functionality that makes the application unique.

You don't need special software to write user stories. Just use index cards or a Word document. In the next chapter, we'll see how you can enter user stories as tasks in a to-do list. Here's the format:
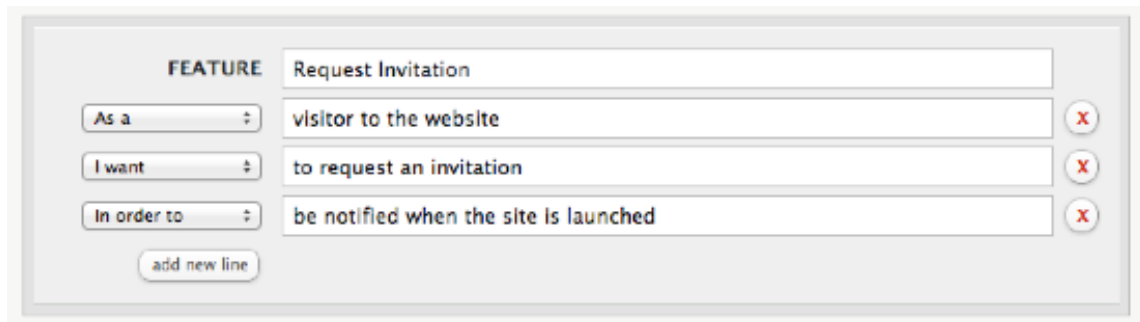
Figure 8.1: A user story.

Just like Rails provides a structure for building a web application, user stories provide a structure for organizing your product plan.

Wireframes and Mockups

Often, before writing user stories, a product owner will make rough sketches of various web pages. Sketching is a phase where you try out ideas to clarify your vision for the application. Sketching can lead to a wireframe or a mockup. These terms are often used interchangeably but there are differences in meaning.

A *wireframe* is a drawing showing all functional elements of a web page. It should not depict a proposed graphic design for a website, rather it should be a diagram of a web page, without color or graphics.

A *mockup* adds graphic design to a wireframe; including branding devices, color, and placeholder content. A mockup gives an impression of the website's "personality" as well as proposed functionality.

One of the most popular tools for creating wireframes is Balsamiq Mockups (despite the name, it produces wireframes, not mockups). There are dozens of others listed in the article Rails and Product Planning.

As a product owner, writing user stories or sketching wireframes will help you refine product requirements. Some people like a visual approach with wireframes; others prefer words and narrative. Either approach will work; both are good.

Graphic Design

Very few people have skills as both a visual designer and a programmer. The tools are different; graphic designers typically use Adobe Photoshop, though web-savvy designers often create designs directly in HTML and CSS, while developers write code.

If you're lucky, you will work with a skilled graphic designer as you build your web application. If you are very lucky, you may work with someone who is a *user experience* (UX) designer or *interaction designer* (IxD). Interaction design is a demanding, sophisticated discipline that requires the mindset of an anthropologist and the eye of a visual artist to find not just the most pleasing, but the most effective visual design for an application user interface. You can find interaction designers discussing their concerns on the IxDA website, including the differences between interaction design and UX design.

If you're working with a graphic designer you might collaborate on a *moodboard* or a *design brief* to define the look and feel of your application. If the

designer works in Photoshop, you'll face the challenge of converting design layouts from Photoshop to HTML and CSS. There are service firms that do this for a fee but obviously it's easier to work with a designer who can implement a layout directly in HTML and CSS.

Rails can be particularly challenging when it comes to integrating graphic design with code. Rails uses a hybrid of HTML markup mixed with Ruby programming code in its *view* files (depending on the stack you've selected, the view files can use ERB, Haml, or other syntaxes for mixing HTML and Ruby). Few designers are comfortable with Ruby code mixed with HTML so you may end up doing integration yourself.

If you don't have a skilled graphic designer available to help, you can use Bootstrap or other front-end frameworks such as Zurb Foundation to quickly add an attractive design to your application.

In a later chapter that covers HTML and CSS, you'll learn about templates and themes for Bootstrap that provide a beginning point for page design.

Software Development Process

Product planning is the initial phase of a larger software development process. You can approach this casually, and start coding with curiosity and ambition, finding your own best way to the end product, by trial and error. Most hobbyist and student developers need no other approach.

When money or reputation is at stake, casual approaches to software development are risky. Compared to other forms of engineering, software development is peculiarly prone to failure. As recently as 2003, IBM stated, "Most software projects fail. In fact, the Standish group reports that over 80 of projects are unsuccessful either because they are over budget, late, missing function, or a combination. Moreover, 30% of software projects are so poorly executed that they are canceled before completion."

Professional software developers, being intelligent and reflexive, and driven by

a desire to become more efficient, or wanting to avoid the wrath of bosses and clients, frequently look for ways to reduce risk and improve the software development process. In recent years they've succeeded in improving the success rate of software engineering, largely due to the adoption of *software development methodologies* that improve the *business process* of producing software.

If you're a hobbyist or casual programmer, you don't need to learn about software development methodologies.

If you are going to be held accountable for the success or failure of a project, you should learn more about software development methodologies.

If you're going to be interviewing for a job as a programmer, it pays to recognize some of the names of software development methodologies and ask whether your employer has adopted a particular approach, especially if you'd like to work for a company that prides itself on being well-organized and supportive of staff development. Hiring managers may say, "we've synthesized several methodologies," which may mean they don't have a good answer for the question, or it may mean they are prepared to thoughtfully discuss the merits of various approaches to software development. Managers who can discuss software development methodologies are more likely to be concerned about the welfare of their team.

Here are some software development methodologies you may hear about, with some notable characteristics:

- waterfall process - an old and disparaged approach

- Agile software development - an iterative and incremental approach

- Scrum - known for "sprints" and daily standup meetings

- Extreme Programming (XP) - pair programming and test-driven development

Agile, Scrum, and XP are all related, and often mixed in practice.

As you mature as a software developer, take time to think about the process of building software and learn more about software development methodologies.

Behavior-Driven Development

There is one prominent software development approach that is important for product planning. It is called Behavior-Driven Development (BDD), or sometimes, Behavior-Driven Design.

BDD takes user stories and turns them into detailed scenarios that are accompanied by tests.

Here's a screenshot from a consultant's web application that shows how a user story can be extended from a "feature" to include detailed "scenarios."

Rails developers turn these scenarios into tests using either a software tool named Cucumber or RSpec to run automated test suites. Most developers write scenarios using a simple text editor.

With automated tests, a product owner can determine if developers have succeeded in implementing the required features. This process is called *acceptance testing*. Automated tests also make it easy for developers to determine if the application still works as they add features, fix bugs, or reorganize code. This process is called *regression testing*.

On a small project like our tutorial application, you won't use BDD. It's easy enough to manually test the features before you deploy your application.

For an introductory book, BDD is an advanced topic. But on a project where money and reputation is at stake, BDD can be very important. Every time an application is deployed, there's a chance that something could be broken. Software development is plagued with "fix one thing, accidentally break another" as code is refactored or improved. Manual testing can't be expected to reveal every bug. That's why automated testing, providing coverage of every significant user-facing feature, is the only way to know if you've deployed without known bugs.

In Book Two, a "Testing" chapter will introduce you to the terminology and concepts of automated testing. You won't have to worry about testing when we build the tutorial application, but afterward you'll learn enough about testing to be prepared for more advanced tutorials.

Figure 8.2: Feature and scenario.

Chapter 9

Manage Your Project

How do you know you're making progress? Are you taking care of everything that needs to be done? These questions are at the center of project management. Whether you are working alone or as part of a team, you need to define your tasks and track progress toward your goal.

The previous chapter on product planning showed how *user stories* can be used to break down an application into discrete features. User stories can be the basis for a list of tasks.

To-Do List

You can track your tasks with a simple to-do list. Some entrepreneurs like the discipline of the GTD system (Getting Things Done) for personal productivity and time management. Our article on Rails and Project Management offers a list of popular to-do list applications, either for personal task management or team-oriented task management.

Kanban

Kanban is a method of managing projects that has been adapted from lean manufacturing for use in software development. In Japanese, "Kan" means visual, and "ban" means card or board.

Imagine putting a big whiteboard on your wall and creating columns for a series of to-do lists. The columns, called *swimlanes*, are labelled: Backlog, Ready, Coding, Testing, Done. Each swimlane contains index cards that describe a user story or other task. To plan your work and track progress, you'll move the index cards across the board from column to column. To stay focused and avoid becoming overwhelmed, you'll only pick the most important user stories or tasks from the backlog column and you'll limit the number of items in each column to what can be realistically accomplished in the time available. That's the essence of kanban as it is used for software development.

See the article on Rails and Project Management for a list of kanban web applications. Trello is particularly popular for task management.

Agile Methodologies

For a solo project or a small team, you'll do fine with a simple to-do list or (even better) a kanban web application for managing your software development process.

If you've got enough people to need to hire a project manager, you should look at project management software that supports teams using Agile software development methodologies. Pivotal Tracker is the best known tool but there are many other agile tools.

Learn more about Agile if you're going to hire developers for a startup or if you are going to work for an established company. In most successful companies, Agile processes have replaced the much-maligned waterfall process that was once the norm for software development.

Chapter 10

Mac, Linux, or Windows

This is a book for every beginner, so I'll explain how we use a text editor and terminal application for development. First, though, let's ask, "Mac OS X, Linux, or Windows?"

Your Computer

You can develop web applications with Rails on computers running Mac OS X, Linux, or Microsoft Windows operating systems. Most Rails developers use Mac OS X or Linux because the underlying Unix operating system has long been the basis for open source programming.

Later in this chapter, I'll give links to installation instructions for Mac OS X and Linux.

For Windows users, I have to say, installing Rails on Windows is frustrating and painful. Readers and workshop students often tell me that they've given up on learning Rails because installation of Ruby on Windows is difficult and introduces bugs or creates configuration issues. Even when you succeed in getting Rails to run on Windows, you will encounter gems you cannot install. For these reasons, I urge you to use Cloud9, a browser-based development

environment, on your Windows laptop.

Hosted Computing

If you are using Windows, or have difficulty installing Ruby on your computer, try using Cloud9.

Cloud9 provides a hosted development environment. That means you set up an account and then access a remote computer from your web browser. The Cloud9 service is free for ordinary use. There is no credit card required to set up an account. You'll only be charged if you add extra computer memory or disk space (which you don't need for ordinary Rails development).

The Cloud9 service gives you everything you need for Rails development, including a Unix shell with Ruby pre-installed, plus a browser-based file manager and text editor. Any device that runs a web browser will give you access to Cloud9, including a tablet or smartphone, though you need a broadband connection, a sizable screen, and a keyboard to be productive.

Installing Ruby

Your first challenge in learning Rails is installing Ruby on your computer.

Frankly, this can be the most difficult step in learning Rails because no tutorial can sort out the specific configuration of your computer. Get over this hump and everything else becomes easy.

The focus of this book is the background you need to understand Rails. In Book Two, you'll build a real-world Rails application. Before you can build the application, you'll need to install the latest versions of the Ruby language and the Rails gem. You can get started now, with the links provided below, or you can wait until you have started reading Book Two.

You'll spend at least an hour installing Ruby and Rails, so defer the task until

you have sufficient time at your computer.

Mac OS X

See this article for Mac OS X installation instructions:

Install Ruby on Rails - Mac OS X

Ubuntu Linux

See this article for Ubuntu installation instructions:

Install Ruby on Rails - Ubuntu

Hosted Computing

Cloud9 is a browser-based development environment. Cloud9 is free for small projects. If you have a fast broadband connection to the Internet, this is your best choice for developing Rails on Windows. And it is a good option if you have any trouble installing Ruby on Mac or Linux because the Cloud9 hosted environment provides everything you need, including a Unix shell with Ruby and RVM pre-installed, plus a browser-based file manager and text editor. Using a hosted development environment is unconventional but leading developers do so and it may be the wave of the future.

See this article for Cloud9 installation instructions:

Install Ruby on Rails - Cloud9

The article shows how to get started with Cloud9.

Windows

Here are your choices for Windows:

- Use the Cloud9 hosted development environment

- Install the Railsbridge Virtual Machine

- Use RubyInstaller for Windows

Cloud9 is ideal if you have a fast Internet connection. If not, download the Railsbridge Virtual Machine to create a virtual Linux computer with Ruby 2.2 and Rails 4.2 using Vagrant. Other tutorials may suggest using RailsInstaller, but it will not provide an up-to-date version of Ruby or Rails. Also, RVM does not run on Windows.

Chapter 11

Terminal Unix

You'll need to use the Terminal application and Unix commands to develop Rails applications.

Most people use a graphical user interface (GUI) to interact with their computers, tablets, or phones. As a developer, instead of using the GUI, you'll get "under the hood" and work directly with the engine that controls your computer, the operating system. This is what makes software programming look intimidating to learners. You won't use menus or buttons as you work. Instead, you'll type commands, line by line, into a window that looks like a computer interface from the 1970s. In fact, the terminal, or console, is a direct legacy of computers that were developed even earlier, in the 1960s. The terminal continues to be the fundamental tool of software development.

The Terminal

The Terminal application or console gives us access to the Unix command line, or shell.

We call the command line the *shell* because it is the outer layer of the operating system's internal mechanisms (which we call the *kernel*).

On Mac OS X, you can use the Terminal application. Experienced developers often upgrade to the more powerful iTerm2 application but you can start with the installed Terminal application.

Look for the Terminal in the following places:

- Mac OS X: *Applications > Utilities > Terminal*

- Linux: *Applications > Accessories > Terminal*

- Windows: *Taskbar Start Button > Command Prompt*

On the Mac, search for the Mac OS X Terminal application by pressing the Command-Spacebar combination (which Apple calls "Spotlight Search") and searching for "Terminal." The magnifying glass in the upper right corner of your screen will also launch "Spotlight Search." Or look in the **Applications/Utilities/** folder for the Terminal application. You'll need to click the name of the application to launch the Terminal.

For Linux or Windows, The Command Line Crash Course explains how to launch a terminal application.

If you have your computer in front of you, launch your terminal application now.

Try out the terminal application by entering a shell command.

```
$ whoami
```

Don't type the **$** character. We call it "the prompt." The **$** character is a cue that you should enter a shell command. This is a longtime convention that indicates you should enter a command in the terminal application or console.

The Unix shell command **whoami** returns your username.

```
$ whoami
danielkehoe
```

Instead, you might see:

```
command not found: $
```

which indicates you typed the **$** character by mistake.

If you are new to programming, using a text editor and the shell will seem primitive compared to the complexity and sophistication of Microsoft Word or Photoshop. Software developers edit files with simple text editors and run programs in the shell. That's all we do. We have to remember the commands we need (or consult a cheatsheet) because there are no graphical menus or toolbars. Yet with nothing more than a text editor and the command line interface, programmers have created everything that you use on your computer.

Unix Commands Explained

Unix commands seem cryptic at first. They are a shorthand that's familiar to experienced developers. If a Unix command is mysterious, you can look it up with Google. But a better approach is to use the website:

- explainshell.com

Try it out. Visit the website and enter **ls -1p**. It's a Unix command we'll use often in Book Two. The site will explain that the command "lists directory contents, one file per line, with a slash appended to directories." Now that you know about explainshell.com, here's no need to ever be mystified by a Unix command.

Getting Fancy With the Prompt

If you watch experienced developers at work, you may see their consoles are colorful, with lots of information shown in the prompt. You'll see Git status, current directory, and RVM gemset or Ruby version. Many developers replace the standard Bash shell with the Z shell and Oh-my-zsh. You don't have to install the Z shell to get a fancy prompt; the Bash-it utility is easy to install and gives you much of the functionality. A fancy prompt is helpful but requires some Unix skills to install. Don't worry about getting fancy now; you can try it down the road.

Learning Unix Commands

You can follow the tutorial in Book Two without learning any Unix commands in advance. Everything you need to know is given at each step. If time is short, and you want to get started, you can jump into the Book Two tutorial without learning Unix commands.

Eventually, you'll realize you've learned quite a few basic Unix commands without making an effort. But if you have time, you'll feel more confident if you spend some time watching a few videos or reading books that teach the Unix command line basics.

If you haven't used the computer's command line interface (CLI) before, I recommend either Learn Enough Command Line to Be Dangerous or The Command Line Crash Course to gain confidence with Unix shell commands.

If you don't have time for the books recommended above, continue reading this chapter for an introduction to command line basics.

Exit Gracefully

Before you learn about Unix commands, learn how to exit a command line software program. It is the most important Unix skill you'll need. If you don't learn it now, you'll get stuck inside Unix programs and panic when you can't return to the command line prompt.

If you're keeping a notebook for things you learn, write this down:

```
To EXIT FROM UNIX commands, type:
Control-c

If that doesn't work, try:
q
exit
Control-d
Control-z

If nothing works:
CLOSE THE TERMINAL WINDOW
```

To type `Control-c`, hold down the "Control" key while pressing the `c` key on your keyboard.

There is no universal command to exit a Unix program. Any of these exit techniques might work. None do harm to your computer, so if you get stuck, try them all.

Software developers don't usually say, "I'm running a Unix program." Usually they say, "I've launched a Unix process." As you learn more about Unix, you'll learn about the difference between `Control-c`, which "kills a process," and `Control-z`, which "suspends a process." Right now, you don't need to learn about processes or what it means to kill or suspend them.

Structure of Unix Commands

Unix commands are cryptic if you've never seen them before. But there's a common pattern you'll begin to recognize.

```
$ command -option argument
```

Notice there are spaces between each part of a Unix command.

Prompt

The **$** character is the prompt. Sometimes you'll see extra information before the prompt:

```
My-MacBook:~ danielkehoe$
```

The prompt can be customized to provide useful information, such as the name of the current folder or the current user. Some people change the **$** character to a different character, such as the **>** character.

Don't be confused by a custom prompt and don't worry about customizing your prompt right now. Just remember that when you see a **$** character on the command line, the computer is waiting for you to enter a command.

Command

```
$ command -option argument
```

Each Unix command is a tiny software program that is already installed on your computer. Each command does a few simple things, such as list the contents of

a folder or create an empty file. There are dozens of Unix commands but you only need to learn a few to develop a Rails application.

Entering a command at the prompt and pressing "Return" (or "Enter") will result in either of two things. The computer will respond "command not found" or it will run the command and return a result to the terminal window ("the console").

Sometimes a command is not found because it is not a built-in command and it is not installed on your computer. More often, you've made an error in typing the command. Unix doesn't like capital letters ("uppercase") so unless there is a capital letter in a filename, you probably will need to type lowercase characters.

Option

```
$ command –option argument
```

Most Unix commands have a default response. Many commands have options for different responses if you set an option "flag" or "switch." You indicate an option with a **–** character. Most people call that character a hyphen or dash. You may hear programmers call it the "minus" character. Sometimes an option is set with **–**, that you might describe as "dash dash" or "minus minus." It's difficult to see on the typeset page of this book, but to get the **–** character you type the "dash" character once. In this book, **–**, which appears as a slightly longer dash, is actually typed as two dashes.

Two options are common among Unix commands: **–help** and **–version** (with double dashes). There are also abbreviated versions: **–h** and **–v** (with a single dash).

If you have a terminal window open on your computer, try typing:

```
$ man -v
man, version 1.6c
```

If I was coaching a beginner, I would say, "Type man space minus v. Don't forget the space and be sure to type 'Enter.'"

The computer returns the version number of the current installed **man** Unix command.

The Unix **man** command provides online documentation for every Unix command. Try:

```
$ man man
.
.
.
```

You'll see the "man pages" that show exactly what to do with the **man** command.

You can type **man** followed by the name of a command to see the documentation for the command.

It is nice to know that every Unix command comes with complete instructions for use. But try reading some man pages and you'll understand why most developers never look at man pages. Instead they use Google to search online for instructions about how to use Unix commands. Man pages suffer from the worst features of technical documentation. They are complete to the point of obscurity, providing massive detail without highlighting the most common use cases. You're better off googling for examples of how to use Unix commands.

You'll be stuck inside the man pages after you enter **man man**. Refer to your notes about how to exit a Unix program. To get out of the **man** program, type **q** to quit.

Argument

```
$ command -option argument
```

The term "argument" is borrowed from mathematics. Many Unix commands like a good argument. It is information that will be processed by a Unix command. Often it is a filename, if the command will operate on a file or output to a file.

For example, we can create a file with the Unix **touch** command or remove a file with the Unix **rm** command:

```
$ touch myfile
$ rm myfile
```

In both cases, we must supply a filename as an argument.

Quick Guide to Unix Commands

As you learn Unix commands, it is a good idea to write notes for yourself or prepare a personal cheatsheet. The act of taking notes will help you remember the commands. Ultimately, the commands will become second nature through sheer repetition as you develop Rails applications.

Here are the Unix commands you will use most often.

cd

Computers use file systems to control how data is stored and retrieved using s storage system such as a hard drive. By separating the data into individual pieces, and giving each piece a name, the information is easily separated and

identified. In offices in the 20th century, documents were grouped together as files and kept in file folders in filing cabinets. Computers don't really need to organize information as files but early computer users apparently thought of computer storage as electronic file cabinets.

In Unix, you are always expected to be somewhere in the filing cabinet. The folders, or directories, are hierarchical, so a root folder contains multiple folders, which each contains many more folders and files. One set of nested folders contains the programs and utilities which make up the operating system and system utilities. Another set of nested folders contain folders for every user who can sign in to use the computer.

Unix systems can have more than one user account. Each user is given a home directory that contains the files needed by any software programs he or she will use.

Unix expects you to always be somewhere in your computer's file system. If you get confused or lost, you can always return to your home directory. Unix programmers like cryptic shortcuts that use unique keys from the keyboard. To get to your home directory, you can enter the directory name, which is your user name, or just type the "tilde" or "squiggle" character. On most keyboards, it is the uppercase (shift key) character to the left of the numeral 1. Look at the typeset character **~** in this book. It may look like a smudged dash, but if you look closely, you'll see the squiggle.

Use the **cd** (change directory) command to go to your home directory:

```
$ cd ~
```

The computer will not return a response to the terminal window, but you'll be positioned in your home directory. That means Unix commands will look for files in your home directory or save files in your home directory unless you specify another location.

pwd

Discover where you are by asking for the "present working directory."

```
$ pwd
/Users/danielkehoe
```

The Unix **pwd** command will show the file path of your current directory.

It is easy to get confused and not be sure where you are. When you feel lost, use the **pwd** command.

ls

Use the **pwd** command to figure out where you are. Then look around with the **ls** command.

The **ls** command lists the files and folders that are contained in your present working directory.

```
$ cd ~
$ ls
Applications Documents    Movies      Pictures
Downloads    Music       Public
Desktop      Library
```

The response shows a list of files and folders. Your list will be different.

You can use options for many different lists of files and folders.

For example, you can see a list of files and folders in a single column with **ls -1**:

```
$ ls -1
Applications
Desktop
Documents
Downloads
Library
Movies
Music
Pictures
Public
```

Using **ls -1p**, you can see a single column list with slash characters marking the folders:

```
$ ls -1p
Applications/
Desktop/
Documents/
Downloads/
Library/
Movies/
Music/
Pictures/
Public/
```

It seems there are no files in my home directory. But is that true?

Hidden Files and Folders

The Unix operating system allows filenames and folder names that start with a dot. These files are hidden unless you use an **a** option to view them.

```
$ ls -la
total 400
drwxr-xr-x+ 53 danielkehoe  staff   1802 Sep 27 15:48 .
drwxr-xr-x   6 root         admin    204 Dec  8  2014 ..
-rw-r--r--@  1 danielkehoe  staff  16388 Dec  8 05:52 .DS_Store
drwx------   4 danielkehoe  staff    136 Dec  8 13:13 .Trash
-rw-------   1 danielkehoe  staff  98661 Dec 12 09:00 .bash_history
```

```
-rw-r--r--@  1 danielkehoe   staff    4926 Nov 20 22:10 .bash_profile
-rw-r--r--@  1 danielkehoe   staff     792 Sep  2  2013 .bashrc
drwx------    6 danielkehoe   staff     204 Jun 26 20:24 Applications
drwx------+  5 danielkehoe   staff     170 Dec  7 08:33 Desktop
drwx------+ 12 danielkehoe   staff     408 Dec  7 08:06 Documents
drwx------+  9 danielkehoe   staff     306 Dec 12 10:57 Downloads
drwx------@ 57 danielkehoe   staff    1938 Aug 30 20:27 Library
drwx------+  3 danielkehoe   staff     102 Jul 28  2013 Movies
drwx------+  5 danielkehoe   staff     170 Sep 21  2014 Music
drwx------+ 22 danielkehoe   staff     748 Nov 26 20:57 Pictures
drwxr-xr-x+  5 danielkehoe   staff     170 Aug 11  2013 Public
```

Now the response is very detailed, showing hidden files as well as information about file permissions, owner name, groups that have access permissions, file size, and creation date.

Hidden files are often files that contain configuration settings or user preferences. In the example above, the **.DS_Store** file is created by the Mac operating system to store the screen position of a Finder window. The **.bashrc** file contains configuration settings for the Unix shell program named bash.

Box 11.1. Hidden Files in Cloud9

If you're using Cloud9, you must change preferences to see hidden files. In the window that contains the file list, there is a gear icon (dark in color and difficult to see). Clicking the gear option will give you options:

- Show Root File System

- Show Home in Favorites

- Show Hidden Files

You must select all three options to see the hidden files.
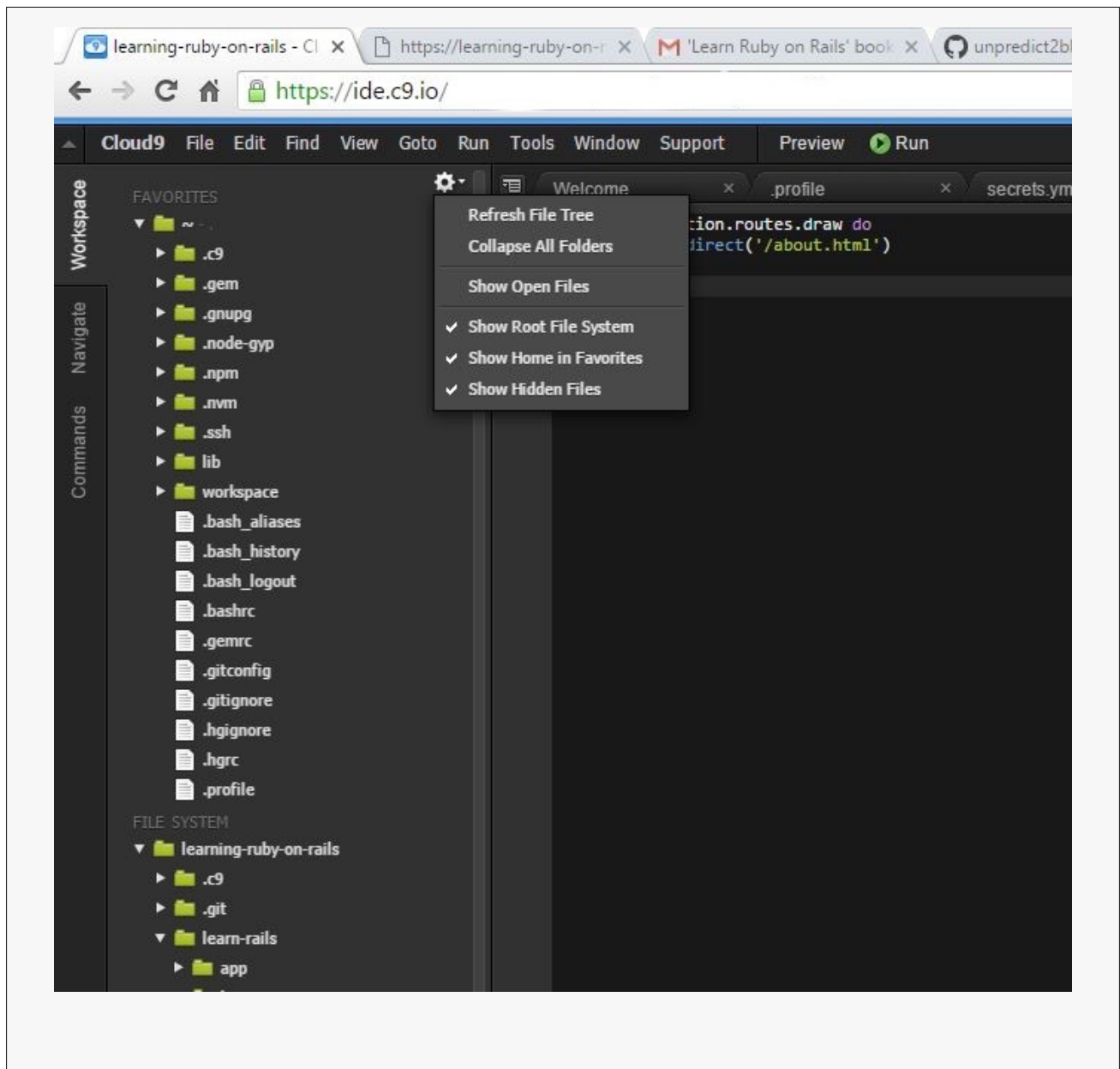
Dots

You might notice that the response lists two files that have dots for names:

```
$ ls -la
drwxr-xr-x+ 53 danielkehoe  staff   1802 Sep 27 15:48 .
drwxr-xr-x   6 root         admin    204 Dec  8  2014 ..
```

These are not really files. These are Unix shortcuts for navigating the file system.

The single dot "file" refers to the present working directory.

The double dot "file" is a shortcut for navigating up one level in the file system hierarchy.

Dot files make Unix commands even more mysterious but they are convenient.

open

On a Mac, you can enter a command in the terminal that opens a Mac Finder file browser window. The command and argument **open .** (open space dot) opens the present working directory. Try it:

```
$ open .
```

You should see files and folders listed in the Finder window that are the same as those displayed in the terminal window. Whether you use the Mac Finder graphical user interface or the terminal, you are looking at the same file system.

On the Mac, you can point and click to move around the file system. If you want to move or delete a file, and you don't remember the appropriate Unix command, you can use **open** to open a Finder window and make changes the Macintosh way.

mkdir

You can create a new folder with the Unix **mkdir** command.

Let's be sure you are in your home directory by using the **pwd** command. Then create a **workspace** folder.

```
$ cd ~
$ pwd
/Users/danielkehoe
$ mkdir workspace
$ cd workspace
$ pwd
/Users/danielkehoe/workspace
```

After creating the **workspace** folder, we **cd** into the folder and check where we are with the **pwd** command.

touch

You can create a new file with the Unix **touch** command.

Often it is easier to use a text editor to create and save a new file. But we'll use the **touch** command here to create a file we'll remove later. It is called the **touch** command because its intended purpose is update the timestamp on files or folders by "touching" the file or folder. But the command is also useful for creating new files.

If you haven't done anything since entering the previous commands, you'll still be in your **workspace** folder. But let's enter a command to move you to the **workspace** folder, just in case you are elsewhere.

First we **cd** using a filepath that contains the **~** "tilde" shortcut for the home directory. In essence, we are saying, "move to the workspace folder contained in the home directory."

```
$ cd ~/workspace
$ pwd
/Users/danielkehoe/workspace
$ touch myfile.txt
$ ls -la
```

```
total 8
drwxr-xr-x   7 danielkehoe  staff   238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff  1802 Sep 27 15:48 ..
-rw-r--r--   1 danielkehoe  staff     0 Dec 12 15:11 myfile.txt
```

You've created a file **myfile.txt** inside the **workspace** folder. Then we list the contents of the folder so we can see the new file.

mv

Unix provides the `mv` command to rename files and folders. It's an abbreviation for "move" and was originally intended to move a file from one directory to another. You can use it to move files and folders. And you can use it to "move" the name of the file as well.

Let's rename **myfile.txt**:

```
$ mv myfile.txt my_file.txt
$ ls -la
total 8
drwxr-xr-x   7 danielkehoe  staff   238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff  1802 Sep 27 15:48 ..
-rw-r--r--   1 danielkehoe  staff     0 Dec 12 15:11 my_file.txt
```

We've changed the name of the file by adding an underscore character. The underscore character is commonly used as a substitute for a space between words in a filename. The Mac operating system can accommodate spaces in filenames but it is bad practice to use spaces in filenames when you work on the command line. You can work with filenames that contain spaces by surrounding the filename with single quote characters (like 'my file.txt') but it is inconvenient and most developers simply avoid spaces in filenames.

cp

You can copy a file with `cp` command. You must enter two filenames as arguments: the name of the original file and the name you want for the copied file.

```
$ cp my_file.txt my_file_2.txt
$ ls -la
total 8
drwxr-xr-x   7 danielkehoe  staff   238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff  1802 Sep 27 15:48 ..
-rw-r--r--   1 danielkehoe  staff     0 Dec 12 15:11 my_file.txt
-rw-r--r--   1 danielkehoe  staff     0 Dec 12 15:16 my_file_2.txt
```

The `ls` response shows we have two files.

The `cp` command requires flags if we want to copy a folder.

We'll use the `mkdir` command to create a folder:

```
$ mkdir myfolder
$ ls -la
total 8
drwxr-xr-x   7 danielkehoe  staff   238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff  1802 Sep 27 15:48 ..
drwxr-xr-x   2 danielkehoe  staff    68 Dec 13 05:13 myfolder
```

Let's try to copy it:

```
$ cp myfolder myfolder2
cp: myfolder is a directory (not copied).
```

The error message indicates we cannot copy a directory.

Let's try it with the `-r` flag to recursively copy contents from one folder to another:

```
$ cp -r myfolder myfolder2
$ ls -la
total 8
drwxr-xr-x    7 danielkehoe  staff    238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff   1802 Sep 27 15:48 ..
drwxr-xr-x    2 danielkehoe  staff     68 Dec 13 05:13 myfolder
drwxr-xr-x    2 danielkehoe  staff     68 Dec 13 05:21 myfolder2
```

With the added flag, we are able to copy a folder.

rm

Let's remove the files we just created. Unix provides **rm**, the "remove" command.

First, let's check that you are still in the **workspace** folder.

```
$ pwd
/Users/danielkehoe/workspace
$ ls -la
total 8
drwxr-xr-x    7 danielkehoe  staff    238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff   1802 Sep 27 15:48 ..
-rw-r--r--    1 danielkehoe  staff      0 Dec 12 15:11 my_file.txt
-rw-r--r--    1 danielkehoe  staff      0 Dec 12 15:16 my_file_2.txt
```

Then enter the **rm** command, providing the filename as an argument:

```
$ rm my_file.txt
$ rm my_file_2.txt
$ ls -la
total 8
drwxr-xr-x    7 danielkehoe  staff    238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff   1802 Sep 27 15:48 ..
```

The **ls** command shows the files are gone.

When you delete a file using the Mac Finder file browser, the file is moved to a Trash folder. When you use the Unix **rm** command, the file is gone forever. There is no Trash folder for recovering a file when you use the **rm** command.

Removing a Folder

Let's remove a folder. We'll assume you are in the **workspace** folder and the folders you created earlier are still there.

```
$ ls -la
total 8
drwxr-xr-x    7 danielkehoe  staff   238 Dec 12 15:21 .
drwxr-xr-x+ 53 danielkehoe  staff  1802 Sep 27 15:48 ..
drwxr-xr-x    2 danielkehoe  staff    68 Dec 13 05:13 myfolder
drwxr-xr-x    2 danielkehoe  staff    68 Dec 13 05:21 myfolder2
```

The **ls** command shows we have folders **myfolder** and **myfolder2**.

We can't remove a folder with the ordinary **rm** command:

```
$ rm myfolder
rm: myfolder: is a directory
$ ls -la
total 8
drwxr-xr-x    7 danielkehoe  staff   238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff  1802 Sep 27 15:48 ..
drwxr-xr-x    2 danielkehoe  staff    68 Dec 13 05:13 myfolder
drwxr-xr-x    2 danielkehoe  staff    68 Dec 13 05:21 myfolder2
```

We get an error message and the **ls** command shows the file is still there. This can be frustrating for someone who is not skilled with Unix.

You'll need to "set a flag" (use an option) to remove a folder using the **rm** command.

```
$ rm -rf myfolder
$ rm -rf myfolder2
$ ls -la
total 8
drwxr-xr-x   7 danielkehoe  staff   238 Dec 12 15:11 .
drwxr-xr-x+ 53 danielkehoe  staff  1802 Sep 27 15:48 ..
```

Now the `ls` command shows the folders are gone.

We use the `r` option to remove the contents of the folder recursively. And the `f` option to force the removal without asking for confirmation.

The Mouse and the Command Line

The mouse belongs to the graphical user interface. That's why it doesn't work as expected on the command line. Try entering a command at the prompt:

```
$ rm -rf notmyfolder
```

If you want to edit the argument and change "notmyfolder" to "myfolder," you can try clicking with your mouse at the point where you wish to change the text. But your mouse click will be ignored. It seems you must use the delete key to back up from end of the line to retype the argument.

But there's a trick that works on most computers. If you press "Option" and click with the mouse, you can edit the command and argument as you would expect.

You can also move around on the command line with Ctrl-a to go to the beginning of a line, Ctrl-e to move to the end of the line, and Ctrl-u to delete everything on the command line. If you can't remember these shortcuts, don't worry, you can just use the delete key to move backward and retype when you need to make changes to commands.

Arrow Keys

The most useful trick for typing text on the command line is the "up arrow" key, which scrolls through a list of previous commands. The "down arrow" key scrolls the history of commands in the opposite direction (forward if you've moved backward).

With Unix, you only need to type a command once and then you can scroll back to it with the "up arrow" key. It is a real timesaver.

Tab Completion

There's another trick developers use to save time when entering commands and arguments in the terminal. If you are entering a filename, and the file already exists in the present working directory, you can press the "tab" key to autocomplete the filename after typing a few unique letters of the filename.

If there is more than one filename with the same initial letters, the Unix shell will balk and beep. Pressing tab again will list all the matching filenames. Continue typing a few more letters until the Unix shell can identify one unique filename and autocomplete.

Why Abbreviations?

Our quick introduction to Unix commands taught you about `ls`, `cp`, `rm` and other common Unix commands. If the commands were spelled out as "list," "copy," or "remove" they might be easier to remember. Some old-timers say Unix was designed to be efficient on slow teletype terminals. That may be true, but I believe the abbreviations persist because programmers are lazy and want to type as few characters as possible. Unix commands may seem obscure but with repetition they will become familiar. Until then, keep a personal cheat-sheet as your reference.

As you learn Rails and develop applications, you'll gain experience with Unix commands. What you've learned so far is enough to get started.

Chapter 12

Text Editor

You'll need a text editor for writing code and editing files.

Word processing programs, such as Microsoft Word, will not work because they introduce hidden formatting codes into text files.

Programmers' text editors provide *syntax highlighting*, making software code more readable and programmers more productive. Simple text editors such as TextEdit for Mac OS X, or WordPad for Microsoft Windows, provide no syntax highlighting and should be avoided.

You Don't Need an IDE

Programmers who come to Rails from other platforms, such as Java or C++, often ask for recommendations for an IDE, or an integrated development environment. These are software applications that combine a text editor with built-in tools such as a debugger. Some Rails developers use JetBrains RubyMine, Aptana Studio, or Komodo but most Rails developers use only a text editor and terminal application. You don't need an IDE unless you're in the habit of using one. For a beginner, they are cumbersome and add little additional value.

97

Which Text Editor

Old-timers and hardcore technologists use text editors that run in the terminal window:

- vim

- Emacs

There is a long-time rivalry between fans of Emacs and vim (and its predecessor vi). Between the two, vim is more popular.

Emacs and vi are popular among skilled programmers because there's no need to leave the terminal window to edit a file. Programmers keep their fingers on the keyboard and don't need to reach for a mouse to use the text editor. That makes programmers more productive. In the long run, learning vim or Emacs will make you more productive as well as impressing your colleagues with your technical skill. Both vim and Emacs are difficult to learn. If you're a beginner, don't attempt to learn vim or Emacs if you're short on time.

Most beginners will use a text editor with a graphical user interface (GUI). You're likely to encounter one of these:

- Atom

- Sublime Text

- TextMate

Atom and Sublime Text are available for Mac OS X, Windows, or Linux.

Atom is an open-source text editor, developed by a team at GitHub. It is the newest of the text editors. If you have not yet installed a text editor, I recommend getting Atom. It can be downloaded and used for free.

Sublime Text has been popular since 2008, particularly among developers who began learning Rails in the last seven years. Developers are expected to pay $70 USD for use of Sublime Text, which removes a nagging popup that reminds users to pay for Sublime Text.

Textmate has been around since 2004 and remains popular among a small group of veteran Rails developers who have never bothered to learn vim or switch to Sublime Text or Atom. It is a commercial product priced at $56 USD.

You can find tutorials for these text editors on YouTube. Or skim the Atom documentation or Sublime Text documentation.

For a book with tips and tricks about using a text editor efficiently, see Learn Enough Text Editor to Be Dangerous.

Editor Shell Command

Carefully follow the instructions for installing your text editor. Installation is like any other desktop application; however, there is an unusual and important final step. Developers want to launch a text editor and open a file for editing by typing a command in the terminal window.

If you've installed Atom, click the "Install shell commands" item under the "Atom" menu to enable the **atom** command. If you do so, you can open a file in Atom from the command line:

```
$ atom myfile.txt
```

Sublime Text has a similar configuration option for the Mac, though it takes more fiddling to set up. The Sublime Text documentation explains how. If you need more help, see Olivier Lacan's blog post, Launch Sublime Text 3 from the Command Line. After you've completed the configuration, you can open a file in Sublime Text from the command line:

```
$ subl myfile.txt
```

Opening a file from the command line is a big win for productivity.

It's even more useful to open an entire folder from the command line. In the "Terminal Unix" chapter, you learned that Unix has a "dot file" that represents the present working directory. To open an entire folder in Atom, try:

```
$ atom .
```

The text editor window will display a "tree view" of files and folders in a window pane. Click on any file to open it. It is very convenient to see an entire project at a glance in your text editor.

Chapter 13

Learn Ruby

Experienced Rails developers debate whether beginners should study Ruby before learning Rails.

Most experienced Rails developers recommend that you study Ruby before attempting to learn Rails. Most code camps teach a week or two of Ruby before introducing web development with Rails. Given that Rails is based on the Ruby programming language, it seems logical to teach Ruby as a prerequisite to learning Rails.

There are several flaws with this approach. Every "learn Ruby" book, video, or online course teaches Ruby with a series of classroom exercises. A concept is introduced, followed by examples, and then a programming puzzle or quiz, much like the way arithmetic was taught to children in the 1950s. If you are academically gifted, you can learn the basics of Ruby with this approach. But many people become frustrated and don't learn well this way.

Though experienced Rails developers think a beginner should learn Ruby this way, many actually knew other programming languages already, and learned Ruby by skimming a language reference to notice differences, and googled for help when they needed to figure out how to write something in Ruby. They improved their skills by reading blogs or watching screencasts such as Avdi Grimm's Ruby Tapas. But mostly they learned Ruby by working on real-world

problems that required skill with Ruby.

The fact is, though you can get oriented to the basics of Ruby in a week, it takes a year or more of regular use of the language to gain proficiency. Rubyists develop their skill with Ruby over years, not weeks. That's true of most general purpose programming languages. Learning Ruby is a lifelong education program, not a short course.

Does that mean you'll never learn Rails? On the contrary. Despite what experienced Rails developers will say, you can begin building Rails applications without developing Ruby proficiency. Rails is largely a DSL, or *domain specific language*, with its own keywords and directives built using Ruby, and following the Ruby language syntax. Many developers started learning Rails by following tutorials to build Rails applications without first studying Ruby. You will pick it up as you go along.

Before my colleagues lambast me for leading newbies astray, let me say that trying to learn Rails without making an effort to learn Ruby is inefficient and counter productive. As you follow a Rails tutorial, make a parallel effort to learn Ruby. One will support the other. By building real applications with Ruby and Rails, and making an effort to learn more about Ruby at the same time, you'll be motivated to learn Ruby and you'll retain more knowledge of Ruby. And you'll be a better Rails developer.

Ruby Language Literacy

In this book series, I've taken a realistic approach. In Book Two, you will get started building a real-world Rails application. You will gain familiarity with the syntax of the Ruby language by studying the code that is needed to build the application. After you've been exposed to real-world Rails code, you'll read a short chapter to learn about the basic features of the Ruby language. The goal of the chapter is to develop Ruby language literacy. With a grounding in real-world Rails code, and a short introduction to the Ruby language, you'll have a solid basis to develop your Ruby knowledge plus motivation to do so.

What you need, more than anything, when you start working with Rails, is reading knowledge of Ruby. With a reading knowledge of Ruby you'll avoid feeling overwhelmed or lost when you encounter code examples or work through a tutorial. Later, as you tackle complex projects and write original code, you'll need to know enough of the Ruby language to implement the features you need. But as a student, you'll be following tutorials that give you all the Ruby you need. Your job is to recognize the language keywords and use the correct syntax when you type Ruby code in your text editor.

Your hardest challenge will be to learn the names of the structures you see in code examples. This is why it is helpful to work your way through a short introduction to Ruby. You'll need to be able to recognize when you are looking at an array or a hash. You should recognize when you are looking at an iterator or the Ruby block syntax. Eventually, you'll recognize more exotic Ruby formulations such as the lambda. It is okay if you can't write a lambda function or even know when to use one; many Rails developers start work before learning Ruby thoroughly.

By all means, if you love the precision and order of programming languages, dive into the study of Ruby from the beginning. It's good advice, just not for everyone. If you've got the time and inclination, get started with these recommended books and videos before you read Book Two. If you can, study them at the same time you work on the project in Book Two. At the very least, look at these resources after you finish Book Two and before you start another tutorial. You'll be glad you did.

Resources for Learning Ruby

Collaborative Learning

The best way to learn Ruby is to actually use it. That's the concept behind this site:

- Exercism.io

With Exercism, you'll work though code exercises and get feedback from other learners.

Online Tutorials

- TryRuby.org - free browser-based interactive tutorial from Code School

- Codecademy Ruby Track - free browser-based interactive tutorials from Codecademy

- Ruby Monk - free browser-based interactive tutorial from C42 Engineering

- Ruby Koans - free browser-based interactive exercises from Jim Weirich and Joe O'Brien

- Ruby in 100 Minutes - free tutorial from JumpstartLab

- Code Like This - free tutorials by Alex Chaffee

- RailsBridge Ruby - basic introduction to Ruby

- CodeSchool Ruby Track - instructional videos with in-browser coding exercises

Books

- Learn To Program - free ebook by Chris Pine

- Learn To Program - expanded $18.50 ebook by Chris Pine

- Learn Code the Hard Way - free from Zed Shaw and Rob Sobers

- Beginning Ruby - by Peter Cooper

- Programming Ruby - by Dave Thomas, Andy Hunt, and Chad Fowler

- Eloquent Ruby - by Russ Olsen

- Books by Avdi Grimm, including *Confident Ruby* and *Objects on Rails*.

Newsletters

- Practicing Ruby - $8/month for access to over 90 helpful articles on Ruby

- RubySteps - weekly lessons by email from Pat Maddox

Screencasts

- RubyTapas - $9/month for access to over 100 screencasts on Ruby

Chapter 14

Crossing the Chasm

In my books, you learn how programming is actually done in practice. In this book, you learn about the culture of Rails beyond the code. In Book Two, you'll follow step-by-step instructions to build and deploy a real Rails application.

This chapter is here to help you surmount the problems that come after you finish my books. You can wait and read it later, after you've built applications using tutorials, or you can read it now and learn to overcome the challenges you'll face when you are done reading tutorials.

What comes next partly depends on your goals. You may be eager to launch a startup, you may want a job as a developer, or you may have a project to tackle at work or in your spare time. Whatever you choose to do, you'll face the challenge of building an application without instructions. Here I'll give you ideas about overcoming that challenge.

Facing the Gap

There's a name for the obstacle that lies in wait for beginners who teach themselves to code by following a tutorial. I call it the "tutorial gap." It's the yawning chasm you face when it is time to build a custom application of your own.

Even though you've just built and deployed a working application, the moment you are without instructions, the chasm will open wide. You'll feel it most acutely when you realize you don't know where to start. Should you search for a gem? Should you start by building a view, a model, or a controller? Should you do test-first development and write a test? What should you test if you don't know where to start?

A similar problem lies in wait for beginners the first day on the job. You'll hear it called the "junior gap." The term refers to the chasm between the time a developer is hired and becomes a "junior dev." A "junior dev" is a team member who is more-or-less self-sufficient, learning new skills without being a burden for other developers, and able to contribute to a company at a level that increases team productivity. Whether self-taught, hired after graduation from a university, or from a 9 or 12 week code camp, new staff members seldom have the skills to be fully productive members of a team. Your success at a new job depends on how quickly you overcome the junior gap.

The "tutorial gap" and the "junior gap" are versions of the same chasm. You'll cross the chasm each time you build an application. The more applications you build, the narrower the chasm becomes, and the less help you need, until you've crossed the chasm so many times that panic is replaced by delight when someone asks you to implement something you've never encountered before. That's the point when you've become a self-sufficient and productive developer.

If you assume that becoming a successful developer depends on acquiring technical skills, the chasm may thwart you. The chasm cannot be crossed with technical knowledge, no matter how much you learn about Ruby, Rails, JavaScript or any other technology. Crossing the chasm requires "soft skills," including cultivating your own problem-solving abilities, and realizing that software development is fundamentally a social activity.

In this chapter, I'm going to describe two ways to cross the chasm. First, I'll give you a technique to jump start work on a custom application. It's a strategy that will enhance your problem-solving abilities. After that I'll suggest how to get help from a mentor, focusing on the social practice that is at the heart of software development.

Bridging the Gap With a Strategy

When you start work on a custom application, you're like a writer who has to write an essay and faces a blank screen. If you've been taught to write an essay, you probably learned to make an outline and start with a topic sentence or introductory statement. Not every writer starts that way, but it's a strategy to get started. You can use a similar strategy to get started with custom application development. Here's a process you can use:

- Ask: Why will someone want to use your application? Write a product description.

- Ask: What will a visitor first see and do? Write a user story for the home page.

- If you like to think visually, create wireframes for some of the important features.

- Create user stories for some of the important features.

- Generate a starter application with **rails new** or Rails Composer.

- Pick any user story. Make a static page using HTML to mock up what the user will see.

- Write a feature test to verify the content on your static page. This is your first code.

- Create a model. Use the model to set a variable containing some string from the static page.

- Create a controller. Give it the same name as the model. Create a route for the controller.

- Replace the static page with a dynamic page. Use the controller to set an instance variable and render a view.

- Does your feature test still pass? Modify your feature test if necessary.

- Ask: How will data displayed on the page get into the model? From a third-party API? A user-submitted form?

- Replace the hard-coded string in the model with dynamic data obtained from a user or an API.

- Check your feature test. Does it still pass? Do you need to change the test to use the model?

- Ask: Have you implemented the user story? Do you need to revise the user story?

- Commit your work to Git. Then throw it away if you're not satisfied. You can always "do over" later.

- Ask: Is there another user story you can implement? Get started again.

You may have to repeat this process many times before you have something you can keep. Each time, you will discover what you don't know. Perhaps you haven't expressed a user story well and you need to rethink your feature. Or you've reached the limit of your technical knowledge and you need to do some research and further study. Each time you repeat this process you are practicing crossing the chasm.

Not every writer starts an essay the same way, and not every developer will use the strategy detailed above. If you've ever faced writers' block, you may have heard this advice from teachers: Write something, anything, just to get started. It's the same with software code: Just begin, anywhere. If you follow the strategy described above, you'll have a place to get started. It is a process you can repeat when you start any application. If you need more help, see the book Practicing Rails by Justin Weiss, which provides advice and exercises to overcome the tutorial gap.

Bridging the Gap With Social Practice

Let's consider the social aspect of software development.

If you work on your own, trying to master the art of software development, it will take you a very long time, you'll need extraordinary patience and tenacity, and you won't become a very good software developer. To accelerate the process, and improve your skill, you must reach out to others. If you're shy and introverted, this will be hard; if you're bossy or reluctant to reveal your shortcomings, it may also be difficult. However, you must make the effort if you want to cross the chasm.

Software development often looks like a solo activity but it is not. Developers talk to users to improve the user interface and product features. Open source libraries, whether gems or full frameworks like Rails, are developed collaboratively. Code reviews are an opportunity to ask others to help you improve your implementation. Pairing is an intense and effective way to write better code and share knowledge. You'll learn more, and faster, from both experienced developers and inexperienced peers, when you work with others.

Making an Effort

It is important to have a strategy to get started on your own, as described above. You must make an initial effort, even if picking yourself up by your bootstraps doesn't get you very far. No one, particularly software developers, wants to help someone who can't show they've made a best effort on their own. You may surprise yourself when you make an initial effort; you'll find out what you know and identify what you need to learn. Start a list of topics you need to research or questions you need to answer. Do research. If no clear answer emerges, list the possibilities and show them to someone else. Even if you think you've found the answer on Stack Overflow or a blog, show someone your initial problem and the solution you've found. Find out if your colleague agrees or has another perspective. Showing someone your research shows you've made an effort and it will make it easier to ask for help when research doesn't provide a solution.

(Refer to the "Get Help When You Need It" chapter if you're not sure where to do research.)

Conversation Starters

You can ask people to answer questions online, on Stack Overflow, Reddit, Quora, mailing lists, forums, IRC channels, or even by directly sending email to developers. This counts as social activity but it is inherently limited. Open-ended interaction is better, either in person or through video chat. Prime the pump with to-the-point questions, as you would on Stack Overflow, but allow the conversation to meander. Make sure your conversation includes conversation helpers like these:

- What do you think?

- How would you do this?

- Is there a better way?

- What do you think I should look at next?

Ask the kind of questions that elicit opinions that you can't ask on Stack Overflow. Never forget to acknowledge the gift you're receiving of time and knowledge. Express your thanks, state clearly how the conversation has benefited you, and offer to report back on how the collaboration has helped your progress.

Pay It Forward

Don't be shy about asking for help. If you've made a best effort to solve a problem on your own, and you're willing to help others in the future, you'll get all the help you need, and more.

There's an unwritten rule of the open source movement. It applies whenever you ask a stranger for help, whether opening an issue on GitHub or asking for help with a project. You *are not entitled to anything*, whether its software code, a bug fix, or just help, *if you're only consuming*. Open source is free for the taking but you're not welcome if you're a mooch. Luckily for all of us, *causality rules do not apply* (or at least there's no temporal causality). That means, if you contribute something in the future, help will be forthcoming, sometimes more profusely than you've asked for. You must contribute in kind, of course. If you offer to pay money to have a bug fixed, you're violating the spirit of open source, and you can expect either grumpiness, a hefty consulting fee, or both. If you indicate you're willing to help with documentation or code, you'll be welcomed even if you're currently incapable of contributing. This applies in an interesting way to beginners. If you show you've tried to solve a problem on your own, and ask for help, many developers will help without any compensation other than the conviction that someday you'll help someone in a similar situation. Software development relies on a booming pay it forward economy.

Finding a Mentor

Now that you've given some thought to the social aspect of software development, let's consider where to go for help.

When career advisors talk about closing the junior gap, they'll often point to the importance of mentorship as the key element in becoming a skilled software developer. It's just as important when you're developing software for a startup or a personal project. Mentorship will help you cross the chasm.

You may ask, how can I find a mentor? In your mind's eye, you may be imagining the mentor that will help you become a skilled Rails developer. He or she is a few years older than yourself, has a great job leading a team at a successful startup, probably contributes to several well-known open source gems, and takes a break every Saturday for a few hours to teach you how to code. Sorry,

that is not likely to happen.

The mentors who will help you will not fit the picture of a wise sage or crone, no more than Prince Charming or Princess Buttercup will ever be anyone's spouse. In most situations, the mentorship relationship will be unacknowledged. The seeds of mentorship lie in any interaction where you ask for help and receive guidance.

Mentorship is a relationship you must cultivate, much like friendship. And, like friendship, you will seldom ever ask someone to be your mentor. Most people would balk at the awkwardness, either because they don't see themselves as qualified, or because it suggests an open-ended commitment and responsibility that is intrusive. Like making new friends, it is your job to seek out mentoring moments, ensure the experience is mutually rewarding, and suggest the possibility of repeating the experience. Mentorship is a relationship built from a series of successful mentoring interactions. Repeat the interactions and you have mentorship.

Creating Mentorship Moments

In the chapter "Get Help When You Need It," you learned where to look for help. Let's consider where you can look for opportunities to experience mentorship moments.

Online

When you ask for help on Stack Overflow, Reddit, or Quora, the answers will be most useful when you focus narrowly on a specific question. Mentorship comes from open-ended interaction, when a conversation can move in unanticipated directions, so most online interactions seldom turn into mentorship moments.

Online interactions may help you find people who can coach you. Clicking on

a user's name on a site such as Stack Overflow, Reddit, or Quora will show you a user's profile. When someone is helpful or knowledgeable, check if they show their geographic location in their profile. If they don't, perhaps they've provided a link to their website or Twitter account. If not, you may be able to send a private message. Reach out and ask where they are located, if they have time to meet to answer more questions, or can suggest anyone in your city who might be helpful. It is worth checking to see if someone is local even though most people online are not nearby.

GitHub

GitHub is a special case. Interactions on GitHub are at the core of the Rails community. It's where open source software gets built. Collaboration on GitHub leads to mentorship, friendships, and business partnerships. That's why it is so important to build a credible GitHub profile by uploading the projects you build while learning, to show that you are working steadily at becoming a better programmer.

When you look at a repository on GitHub, look at the account of the person who maintains the project. Look at the commits and the issues. Click through to the profiles of the people who've made the commits or commented on the issues. Experienced developers often show their location in their profile and provide their email address. If you find someone in your city, make contact. Of course, don't ask a stranger point-blank to be a mentor! Tell them about your experience and what interests you. Ask where you can meet developers locally. You may learn about a meetup or user group meeting. If your contact is helpful, you may have an opportunity to meet for coffee.

Meetups

Meetups are a prime place to cultivate in-person mentorship moments. All large metro areas have active meetups for web developers, Rails developers,

programmers, or startup entrepreneurs. If you're in a rural area, make the drive once a month to the big city to connect with the community. To find the meetups, search Meetup.com or google "ruby rails (my city)". If you're near a university campus, check for activity on the campus. If you're in a tech hub such as San Francisco, there's a meetup almost every night of the week. Smaller cities have relevant meetups monthly. If you can't find a meetup, start one!

When you visit a meetup, remember that mentorship can be hidden in anyone. Like any public event, you'll meet people who are flakey, bigoted, garrulous, prone to halitosis or innumerable personality quirks, as well as a minority who are fascinating and obviously knowledgeable. Don't dismiss anyone. You'll find mentorship moments where you least expect them.

Workshops and Classes

You may be surprised that workshops and classes are not the ideal place to find a mentor. Obviously, given a good instructor and a relevant curriculum, a workshop or class is a great place to learn. However, it is very unlikely that a teacher will become an ongoing mentor. The instructor's goal is to share knowledge with a group of people, so any focus on you as an individual has to be limited. Furthermore, the instructor is probably not available outside of the class or on an ongoing basis. Don't sign up for a class hoping the instructor will become your mentor.

However, a workshop or class is an ideal place to connect with peers. There is no other place where you'll easily find other people who want to learn the same things as you. When I teach, I'm surprised how often people come to a class expecting their education will end as soon as the class is over. If you take a class, seize the opportunity to make one new friend who will be your study partner after the class ends. Better yet, organize a study group to continue after the class is over. If you have only one new study partner, he or she can flake out. Instead, get together with three or four other learners once a week. Support each other, share your excitement, and invite mentors to come speak to your study group.

Peer learning has much in common with mentorship. The leading code camps recognize that collaborative problem-solving skills are as important as technical skills. When students team up to work through exercises or build applications, there's a natural give-and-take as each takes turns making discoveries and sharing knowledge. You don't have to enroll in code camp to be part of a peer learning environment; you can create it yourself in a study group. This isn't a relationship of mentorship, but it is an opportunity to experience mentorship moments.

On the Job

You are most likely to find a commitment to mentorship on the job. If you've been hired to work as a Rails developer, at a company with Rails developers on the team, you're in an ideal environment to learn. Not everyone has the skills to be a good mentor and you may struggle if you are stuck with a senior developer who is a know-it-all or assumes you know more than you do. Still, you have immediate access to developers with knowledge and the company has every reason to encourage you to learn. Unless the company has an explicit program to assign coaches to new hires, you will probably not call someone your mentor. Instead, recognize that you can cultivate mentorship moments by asking for guidance beyond the immediate assignment. In a stressful environment where your team is delivering code against deadlines, not everyone may be able to devote time to teaching. You should seek mentorship moments where you can.

Some companies are committed to building a culture of mentorship. When you are looking for a job, make it your priority to seek a job offer from companies where you will find mentors. For a first job as a developer, the opportunity to learn on the job is far more valuable than any other benefits. When you interview, ask if the company encourages pair programming. Ask if you will have time to work on a pet project to learn new skills, and if it will be acceptable to ask your teammates to answer questions or provide a code review for your pet project. Ask if the company encourages team presentations about new technologies. Ask if anyone from the company volunteers to teach at workshops

or gives presentations at meetups or conferences. Some hiring managers will be proud to describe the company's commitment to developer education. If they're not, it's a red flag that the company may not be a good fit for you.

Small startups are not a good place to look for mentorship, if the runway is short and the founders are trying to launch before funding runs out. Companies that have closed a Series A round (the first release of stock to venture capital firms and other private equity investors) will still be tightly focused on getting a product to market, with no time to coach new hires. As a company takes additional rounds of investment, beyond the Series B round, the company will have grown beyond an initial two or three engineers and may recognize the value of hiring and coaching inexperienced developers. To some companies, mentorship is part of a strategy to develop their technical depth. These are the companies that are ideal for new developers.

What's Next

If your goal is to start a career as a Rails developer, your objective should be to find a job at company that is committed to mentorship. You'll need to learn more than what is offered in this book and the next, but you can continue learning while you "go social" to cultivate mentorship moments and meet people who can introduce you to companies that are hiring Rails developers. Going to meetups, collaborating on code, and participating in a study group will help you find mentors and help you find a job.

If you are eager to launch a startup, or plan to work on a personal project, your next step will be different. Let's consider what you should do after finishing this book if you're an entrepreneur, developing a lifestyle business, or working on a personal "side project."

Entrepreneurs

If you want to launch a startup, stop and ask yourself what your priorities should be. Startup success depends on asking yourself every day, "What is the most important task I need to accomplish today?" Chances are, it is not learning to code. Working through my books, you'll learn enough to work with a skilled developer, whether a cofounder, an employee, or contractors. Your most important task is to determine the product/market fit for your business idea. You must develop a Minimum Viable Product (MVP) and start the process of acquiring customers who can tell you if your product has value. Anything else defers the day of judgment when real customers will tell you whether they are willing to spend money on your product.

If you are pursuing your own business vision, you'll only delay judgment day if your priority is to learn Rails. If you haven't already, start looking for a technical cofounder or people you can hire (if you have your own funds). Angel investors and venture capitalists are reluctant to fund a solo founder, even when an entrepreneur is highly skilled technically. Investors place more importance on the ability and track record of the team than on a business idea; obviously, a skilled team is a better investment risk than a solo founder who just started learning Rails. In today's investment climate, you won't be seeking investment if you don't yet have an MVP and customer traction. But you should start recruiting your team. The good news is that you've learned enough about web development to have credibility when approaching a potential technical cofounder. Among developers, there is no one more ridiculed than the non-technical founder who makes no effort to learn to code and expects someone else to do all the technical work (see Whartonite Seeks Code Monkey). You're not that guy or girl. You *could* build your web application yourself, given enough time. But in the best interests of your business, you should look for a partner who will be your technical mentor, guide, and helpmate as you become a better coder and build out your MVP.

You can seek a technical cofounder in the same way you seek mentorship moments. Your agenda will be larger; anyone who is a mentor or peer may be a potential business partner. Look for help to improve your technical proficiency.

As you build a personal relationship with a mentor or peer, you may have an opportunity to introduce someone to your business vision. It is very unlikely you can build a business on your own, so start looking for a partner while you continue to learn to code and develop your MVP.

You won't be having meetings with potential partners every day. On days when you are not looking for a partner, work on the user stories that will define the requirements for your MVP. You don't need anything more than you've learned in this book to develop your user stories and plan your product. Work on wireframes and show your ideas to anyone who will listen. Take a break from product planning to work on your coding skills. Try tackling one or two user stories and see if you can implement a basic feature you need for your MVP. There's no better way to learn to code than building the product that you need for your business, especially if you get help and feedback along the way.

Lifestyle Businesses and Personal Projects

Don't let anyone discourage you if you have an idea for a web application that will supplement your earnings from a job, or even let you quit your job to enjoy a "lifestyle." The investment community disdains lifestyle businesses that have limited "upside" (the revenue growth that rewards investors). Personally, I think lifestyle businesses are great. Without the pressure from investors, and with income from an existing job, you can take your time to learn to code, enjoying the process of learning application development with your goal in mind. It is still worthwhile to seek out a mentor, but you can continue to pursue learning on your own with all the resources we list in the next chapter.

Personal projects can become lifestyle businesses when they begin to generate revenue. Of course, there are many personal projects that are not intended to be moneymakers, for example, a web application for a faith group or a charity, or just a side project that helps you learn to code. Again, seek mentorship moments, or work in a group that learns together, and you'll develop your skills faster. Play around with user stories and wireframes to see if it helps you organize your project. Try writing feature tests. You may not need to write

tests for a personal project but you may discover a feeling of competence and confidence that goes with testing. With a personal project, the journey is the reward. Indulge in the luxury of learning for its own sake and focus on the satisfaction of seeing applications run that you've built with technologies that are new to you.

Build Applications

If you want to become a skilled Rails developer, nothing is more important than building applications. As you'll see in the next chapter, there is so much to learn about web application development that you can (and likely will) continue to learn until the web goes dark. Don't try to learn it all. Start building applications now.

Building applications puts everything you learn in practical context. The features you want to build will set the priorities for what you learn next. If you want users to sign in to an application, you'll learn about authentication. If you want want to show stock prices or sports scores, you'll learn about JavaScript and charts. There's no better way to learn than by building.

Build simple applications at first, with only one small feature. Take them all the way from user stories to deployment. Use Rails Composer if you want to get started fast so you can focus on your custom features. Commit your projects to your GitHub account, no matter how trivial or broken. Putting your projects on GitHub will show that are working hard and gaining experience. If you're going on job interviews, employers probably won't have time to look at your GitHub account, but you'll gain points in a job interview if you can point to a GitHub repository to show something you've built, or when you answer a question like, "What is the hardest problem you've had to solve?"

Some beginners set a goal, such as building one new application every week. That's a great plan. When your applications get more complex, they will make take more than a week to build, but keep on building. If you don't have any ideas for what to build, ask for ideas on Reddit or Quora and the indefatigable

commenters will gladly answer. At a minimum, you should build (and thoroughly understand) each of the starter applications you can build with Rails Composer.

With every application you build, the chasm of the "tutorial gap" will narrow and you will broaden your ability to tackle unfamiliar problems and challenges.

In the next chapter, we'll consider specific technical skills and I'll make recommendations for books and tutorials that will increase your technical proficiency,

Chapter 15

Level Up

With this book, you're on the way to becoming a successful Rails developer. You've learned about basic concepts and discovered where to go for help. But there's much more to learn about Rails and web application development. This chapter will suggest the next steps on your path to learning Rails.

What to Learn Next

In Book Two, you'll build a simple web application. It will cover the basics:

- Rails directory structure

- using Git

- installing gems

- configuring an application

- the request-response cycle

- model-view-controller architecture

- application layout and views

- front-end frameworks

- forms

- sending mail

- connecting to external services

- deploying an application

- analytics for traffic and usage

In addition, you'll get an introduction to the Ruby language and the basics of testing.

Here are topics you should study after Book Two:

- Databases

- Testing

- Sessions and Authentication

- Authorization

- JavaScript

I'll explain each topic and suggest where to learn more.

Databases

Book Two will explain how to create a model, a software object that represents data in a database.

When you create a model, you create an object that handles data only for the brief life of the request-response cycle, when it is active in a computer's working memory. You'll want data to persist beyond the brief request-response cycle, after objects disappear from working memory. Rails does not include a built-in database. It offers the flexibility of using several different industrial-grade database systems. Relational database management systems such as SQLite, PostgreSQL, MySQL, and Oracle all use Structured Query Language (SQL) as an interface to store and retrieve data. These databases run separately from a web application, requiring their own database servers.

Rails provides a component, named Active Record, that connects to these database servers. Active Record is a framework for Object-Relational Mapping (ORM), connecting application models to database tables in a relational database management system (DBMS). Active Record makes it possible to save model data as a record in an external database, preserving complex relationships among the data.

Application development would be easy if we could just use spreadsheets to save our data. However, some data, such as a document, is too large to fit in the columns and rows of a spreadsheet. More significantly, database management systems are designed to accommodate relationships among the data. That's why they are called relational database management systems. For example, an ecommerce application might have a Customer model and an Order model. Active Record allows developers to use the Rails API to describe associations among models and interact with a database. For example, you can make sure that an order is not created unless it is associated with a customer. Additionally, Active Record provides a query interface. You can use the query interface to find all orders associated with a particular customer.

Where to Learn

For a focused, fast introduction to Rails and databases, you should read the book:

- Easy Active Record for Rails Developers by Jason Gilmore ($29 USD)

It is my recommended follow-on to learn more about databases.

Testing

You learned about the basic concepts and terminology of testing in Book Two. If you're working on a personal project or your own startup, you can learn more about Minitest to gain the skill you need to write robust tests. If you expect to work with other Rails developers professionally, you'll need to learn to use RSpec for testing.

Where to Learn

Every intermediate-level Rails book talks about testing, without any introduction to the terminology and concepts of testing. Review the "Testing" chapter in Book Two, then learn to set up and use RSpec with a tutorial I've written, The RSpec Tutorial, which is part of the Capstone Rails Tutorials series.

To learn more about RSpec, read these two excellent books:

- Everyday Rails Testing with RSpec by Aaron Sumner ($19 USD)

- Rails 4 Test Prescriptions by Noel Rappin ($25 USD)

You don't have to read these books immediately, but be sure to add them to your reading list.

Authentication and Sessions

Most web applications need a way for users to sign in, permitting access to some features of the application only for signed-in users. The popular gem Devise is used to add authentication for users who register with an email address

and password. OmniAuth is a gem for authentication using a service such as Facebook, Twitter, or GitHub. Most Rails developers will use these gems because they are robust and well-tested.

To understand how authentication works, you'll need to learn about *sessions* in a web application. The web, as originally built, was *stateless*. A server simply responded to a request by delivering a file. To enable ecommerce applications, cookies were adopted in 1997 as a way to preserve state. Each time the browser makes a request, it will send a cookie. A web application will check the value of the cookie and, if the value remains the same, the application will recognize the requests as a sequence of actions or a *session*. A session begins with the first request from a browser to a web application and continues until the browser is closed. Cookie-based sessions give us a way to manage data through multiple browser requests. Rails does all the work of setting up an encrypted, tamper-proof session. The data we most often want to persist throughout a session is an object that represents the user.

Where to Learn

To get started quickly with either Devise or OmniAuth, I've written two tutorials which are part of the Capstone Rails Tutorials series:

- Devise Quickstart Guide

- OmniAuth Tutorial

As a learning exercise, it is worthwhile to build authentication from scratch without Devise or OmniAuth. Michael Hartl's popular book shows how to build authentication from scratch:

- Ruby on Rails Tutorial by Michael Hartl (free online)

Authorization

We use authentication to verify a user's registered identity, so we know the person signing in is the same person who signed up earlier. We use *authorization* to limit access to pages of a web application. Authorization is typically restricted by role, so users are assigned roles with differing access privileges. In the simplest implementation, we check if a user has a specific role (such as administrator) and either allow access or redirect with an "Access Denied" message. Roles are attributes associated with a user account, and often implemented in a User model.

There are no standard conventions for implementing roles and authorization in Rails. Developers often implement roles from scratch and use gems such as Pundit or CanCanCan to implement authorization. For most web applications, you'll need to learn how to implement roles and authorization.

Where to Learn

To learn about authorization, start with a free article I've written on Rails Authorization. I've also written two tutorials which are part of the Capstone Rails Tutorials series:

- Role-Based Authorization

- Pundit Quickstart Guide

These tutorials explain the code from the rails-devise-roles and rails-devise-pundit example applications, which you can generate with Rails Composer as starter applications.

JavaScript

JavaScript is a general-purpose programming language (like Ruby). It is the language used to manipulate web pages within a browser. Every web devel-

oper needs to know JavaScript. For a Rails application, you might develop application features such as auto-complete search forms using a combination of jQuery and AJAX techniques. For more sophisticated web applications, such as a single-page application (SPA) that loads in the browser as a single web page and offers a fluid user experience similar to a desktop application, you'll need to learn to use a JavaScript framework such as Ember.js, AngularJS, React, or Backbone.js. If you intend to specialize as a front-end developer, focusing on user interaction and the browser interface, you'll need to become an expert in JavaScript.

Where to Learn

There are many resources for JavaScript, more than for learning Ruby or Rails. Here are good curriculum guides:

- Learn JavaScript by Mozilla

- The Odin Project: JavaScript and jQuery

The JavaScript course from Codecademy is universally recommended, as is the book Eloquent JavaScript.

Other Topics

There is much more to learn before you gain full proficiency as a Rails developer. Here's an illustration:

The graphic above is from a blog post, This is Why Learning Rails is Hard, by Brook Riggio of the Code Fellows code camp in Seattle. You can see the graphic in detail. The first time I saw it, I felt despair. It is a mind map of all the topics a skilled developer should know. The branches on the right side are topics that are specific to Rails, as well as general skills required of a Rails

developer, such as Git, the Ruby language, and software engineering competencies. The branches on the left are general areas of knowledge that would be understood by any web developer, such as the Unix command line, web fundamentals, deployment, testing, and SQL. Brook Riggio says, "If this looks intimidating to you, you're not alone. . . . Learning Rails is hard because there are many independent concepts to learn." I recommend you spend half an hour each day googling each topic listed on Brook Riggio's map. In three months, you'll have a "big picture" of the knowledge areas that are important to a Rails developer. You can't learn everything at once, so dive further into the topics that interest you. As you tackle new projects, you'll learn more, and you'll add depth to your understanding of each topic on the map.

Curriculum Guides

Brook Riggio's map gives you a list of topics for learning. But it is helpful to tackle the topics in a sequence that makes sense. You'll also need recommendations for the best learning materials. For a curriculum that organizes the topics you need to become a web developer, I recommend:

- The Odin Project

It is a unique community-driven curriculum, organized by Erik Trautman, the founder of Viking Code School, that gives you a roadmap of what to learn, and where to learn it.

Other code camps have published curriculum guides, to be used as "pre-work" for enrolled students:

- Flatiron School Prework

- MakerSquare Prework

These curriculum guides point you to important topics as well as recommended books and courses.

Places to Learn

What's your preferred learning mode? Books, classrooms, videos, online courses? I guess you're comfortable with books, so I'll recommend the best ones for further study. But first let's consider other modes of learning.

Code Camps

Starting in New York City with General Assembly in 2011, and Dev Bootcamp in San Francisco in 2012, the market for developer education has been booming. Right now, in San Francisco, there are a dozen organizations offering immersive code camps, and dozens more in large cities worldwide. These organizations offer eight- to twelve-week courses, priced around $10,000 to $12,000 USD. For a list of code camps, see these websites:

- Course Report

- Hack_app

- Switch

- Techendo Reviews

Code camps are a great way to become a developer, if you can afford the cost, and intend to recover the cost by finding a high-paying job as a developer. The best code camps create an environment of peer-based learning, where you pair with other students to solve technical problems and build applications, just as you would in the workplace. Code camps also provide relentless pressure to learn, from teachers and peers, but primarily from yourself. The quality and depth of technical knowledge you'll acquire varies greatly depending on the code camp curriculum and the individual instructors they've hired. All code camps provide the motivation and social context for accelerated learning,

delivering self-confidence that comes from the encouragement and feedback of teachers and peers.

If you don't have money to pay for code camp, all is not lost. You can teach yourself Rails with books and, with effort, you'll be good enough to start a web-based business or look for a job. I recommend that you build your self-confidence by developing applications on your own. And certainly, find other learners and study together. Most Rails developers are self-taught, at least within the domain of web application development, so code camps or university programs are wonderful, if you can afford the cost, but not essential.

Other Classrooms

It's unusual to find a university or community college that offers classes in web development with Rails. In the US, university computer science programs focus on analytical reasoning and the conceptual underpinnings required for advanced research in the field. Some community colleges teach web development but it is difficult for the schools to find experienced Rails instructors, especially given the disparity in salaries between teaching and software engineering. However, universities or community colleges are good places to meet other students and form a study group, to provide social support for learning.

Classes taught in the community, often free or low-cost, are surprisingly good places to learn. Teachers and organizers are highly motivated and may be experienced Rails developers giving back to the community. Community-based classes or workshops are often poorly publicized, so you'll have to search hard for these courses.

Women have a good chance of finding peer organizations that teach programming and web development with Rails. Start by looking at the course schedules for these organizations:

- Rails Girls
- RailsBridge

- Girl Develop It

- Women Who Code

These organizations only offer introductory classes, so you'll have to study on your own for deeper knowledge. Short courses such as weekend workshops are valuable because you can find other students who want to start a study group.

Online Courses

It's no longer necessary to go to a classroom to go to school. Online courses range from online code camps that provide videos and one-on-one coaching, to websites that offer a selection of pre-recorded videos packaged as a course, to MOOCs (massive open online courses) offered by consortiums of universities.

Chasing the runaway success of classroom-based code camps, you'll find a number of companies that offer code schools delivered online. Here are a few that combine videos with personal coaching:

- Bloc.io

- CareerFoundry

- Code Union

- Tealeaf Academy

- The Firehose Project

- Thinkful

- Thoughtbot Upcase

- Viking Code School

The online code schools provide some of the benefits, specifically curriculum and coaching, of the classroom-based code camps, at a fraction of the cost. Videos, homework projects, and online mentors can't reproduce the intense peer-based learning of the classroom code camps. But you don't have to quit your job or move to a another city.

MOOCs provide university-level education online. You can search a database of MOOCs at the Class Central website. I recommend the edX course:

- CS169.1x Engineering Software as a Service

It is a nine week class, it is free, and it is taught four times a year. It is based on a software engineering class taught at the University of California, Berkeley. The professors have written their own textbook to accompany the class, Engineering Software as a Service: An Agile Approach Using Cloud Computing. The class is very worthwhile, if you have the time and it fits your schedule.

Videos

Online code camps and MOOCs provide supervised learning, combining videos with access to coaches or instructors. If you want self-paced study, without access to a coach, you'll find hundreds of videos online, varying greatly in quality. There's one big problem with videos: The ones that are easiest to find are often outdated. It is very difficult for producers to revise videos and, as you know, Rails changes often.

Michael Hartl, author of the Ruby on Rails Tutorial, a book I recommend, offers screencasts to accompany the book. The cost is $149 USD. At this time, a 3rd edition is in production, covering the newest version of Rails. You can find Hartl's series of free videos on YouTube but they are out of date.

Go Rails offers several dozen intermediate and advanced screencasts. Chris Oliver started producing the videos in mid-2014, so they are newer than most Rails screencasts you'll find on the web. These are task-focused videos, good

for supplementing a book or course. Some are free, some are available with a $9 USD subscription.

RailsCasts seems to always be recommended by Rails developers, and these screencasts were once among the best ways to learn about Rails. RailsCasts creator Ryan Bates left the community in mid-2013 and, unfortunately, many of the screencasts are no longer current or relevant. Still, they are worth a look.

Lynda.com offers a Ruby on Rails 4 Essential Training 12 hour video series at a cost of $25 USD for a monthly subscription. It covers Rails 4.0 and is an adequate introduction at a very low cost.

Thoughtbot Upcase, formerly known as ThoughtBot Learn Prime, is a program from the respected ThoughtBot consulting firm that provides videos for $49 USD monthly, with coding exercises and a personal coach at a higher price. You can see the Upcase curriculum for an overview. The program is well-regarded.

Code School is famous for its Rails for Zombies course, and the company offers a dozen additional courses that cover both Ruby and Rails. The courses combine instructional videos with in-browser coding exercises at a cost of $29 USD per month. The quality is high, the videos are current, and the company tries to make the topics entertaining.

Pragmatic Studio, publishers of many high-quality Rails books, offers a series of 26 videos for $179 USD. The course is a solid introduction to Rails, though it only covers Rails 4.0.

RubyOnRailsTutor.com is a free series of short screencasts introducing Rails to beginners.

Baserails.com is a video series, available with a $25 USD monthly subscription, that shows you how to build an application. It's good if you want practice before building something on your own.

One Month Rails is an 8 hour video series aimed at beginners. It's priced at $99 (though you can find discount codes if you search). If you've read Book Two, you should skip One Month Rails and focus on intermediate-level books

and courses.

Treehouse is a subscription site ($25 USD per month) with a big budget and many course offerings. The courses on Ruby and Rails don't provide enough depth to take you beyond a beginner level.

Tuts+ offers a 3 hour course Riding Ruby on Rails. The course is out of date.

Codecademy offers a series of courses that combine videos with interactive quizzes on Ruby and JavaScript. The course on JavaScript is often recommended. However, the format is very much classroom-oriented without practical context. There is no course on Rails.

Udemy offers a range of video courses on Ruby and Rails. This is crowd-sourced content (like YouTube) and the quality varies greatly. If you've found a great course on Udemy, email me and I'll list it here in the next version of the book.

Skillshare is another source of crowd-sourced video courses. The site offers an inexpensive video series on Ruby on Rails in 30 Days that covers Rails 4. Other offerings on Ruby and Rails are out of date.

As you can see from the long list, videos are plentiful. Avoid the old ones.

Books

At the beginning of this chapter, I recommended Easy Active Record for Rails Developers by Jason Gilmore because it is focused on databases and Rails, the next thing you need to study. Other books deserve mention.

One book stands out among all:

- Ruby on Rails Tutorial by Michael Hartl (free online)

More Rails developers read, and recommend, Michael Hartl's book than any other. For you, after reading this book and Easy Active Record for Rails Devel-

opers, Michael Hartl's book will be a review of what you've learned. I hope you will breeze through it, given the fundamental concepts you've already learned.

Three other books on Rails are notable:

- Agile Web Development with Rails by Sam Ruby, Dave Thomas, and David Heinemeier Hansson

- The Rails 4 Way by Obie Fernandez, Kevin Faustino, and Vitaly Kushner

- Rails 4 in Action by Ryan Bigg and Yehuda Katz

All three are dense, comprehensive, and authoritative. In my opinion, you should start building Rails applications before digging into these books. As you begin building applications, dip into any of these books for further explanation and insight.

I recommend the book by Justin Weiss, Practicing Rails: Learn Rails Without Being Overwhelmed. The book provides useful technical tips and tricks, such as techniques for debugging, but the focus of the book is overcoming challenges that new developers commonly face. You'll find advice about keeping up with the Rails community and managing time and motivation when learning Rails, as well as overcoming "the tutorial gap" to begin building your own applications.

Several other books should be on your reading list to improve your skill:

Rebuilding Rails by Noah Gibbs. If you like to discover how things work, you'll gain a deep insight into Rails from Noah Gibbs's book, as he shows you how to build a framework like Rails from scratch.

Practical Object-Oriented Design in Ruby by Sandi Metz. A must-read that teaches the techniques of master programmers.

In addition to the books listed here, I recommended several books to help you learn the Ruby programming language at the the end of the chapter, "Just Enough Ruby."

A Final Word

Keep in mind the reason you're here. You're learning Rails so that you can build applications. I've given you a book that is dense with links and recommendations for further resources. I've met many students who are overwhelmed with all these resources. Some people postpone building anything because there is so much more to learn. Don't be that person. Skip everything I've recommended in this chapter and just get started building. When you need to learn more, you can come back and dig deeper.

Good luck with building the application in Book Two. I hope you like my approach and writing style so that you'll continue with the Capstone Rails Tutorials.
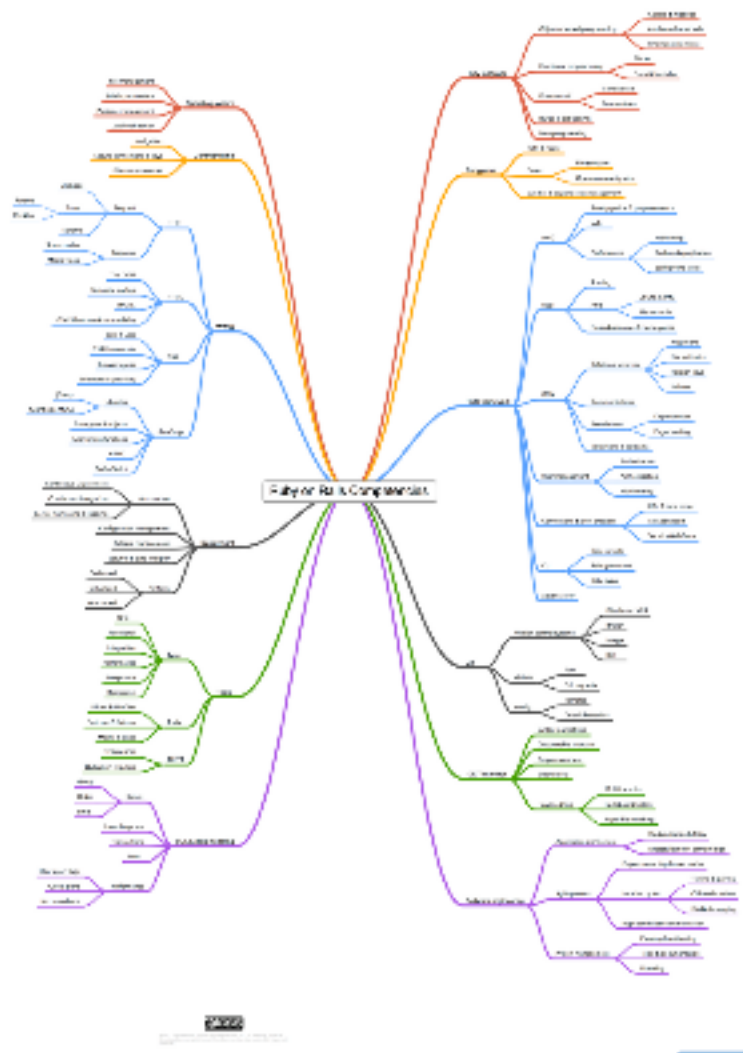
Figure 15.1: Rails Competencies.

Chapter 16

Version Notes

If you've purchased this book as part of the Capstone Rails Tutorials, you have access at any time to the most recent version of the book. If you've gotten your copy of the book elsewhere, you may have an older version that doesn't have the newest updates.

You'll find the version number and release date on the first page of this book (under the book title). Check the learn-rails GitHub repository to find out if you have the newest version of the book. The README page on the GitHub repo always shows the most recent version number for the book and the tutorial application.

Here are the changes I've made.

Version 3.0.0

Version 3.0.0 was released January 14, 2016

Extensive revision throughout the book, and the length of the book increased, so the book is now two books. Book One contains the introductory and self-help chapters and can be read without access to a computer. Book Two contains the step-by-step tutorial and requires use of a computer.

Version 2.2.2

Version 2.2.2 was released October 30, 2015

In the "Front-End Framework" chapter, updated filename to `1st_load_framework.css.` from `framework_and_overrides.css.scss` to reflect a change in the rails_layout gem.

Version 2.2.1

Version 2.2.1 was released September 19, 2015

Updated references to Ruby from version 2.2.0 to 2.2.3.

Updated references to Rails 4.2.0 to Rails 4.2.4.

Updated Visitor model `subscribe` method for the new Gibbon 2.0 API.

Recommending Cloud9 instead of Nitrous.io because Nitrous.io is no longer free.

Version 2.2.0

Version 2.2.0 was released June 6, 2015

For Amazon customers, added an offer to access the online version or download a PDF at learn-rails.com.

Google now requires use of OAuth 2.0 for application access to Google Drive. The implementation is considerably more complex than the previous implementation using a Gmail address and password. I've dropped the "Spreadsheet Connection" chapter.

Minor clarification in the "Layout and Views" chapter.

Version 2.1.6

Version 2.1.6 was released March 17, 2015

Remove references to the Thin web server in the "Deploy" chapter.

Correct version number for **`gem 'sass-rails'`** in various Gemfile listings. Fixes issue 49 and an error "Sass::SyntaxError - Invalid CSS" when the Foundation front-end framework is used.

In the "Testing" chapter, the file *test/integration/home_page_test.rb* was missing **`require 'test_helper'`**.

Updated "Rails Composer" chapter to describe new options.

Minor improvements and corrections of typos.

Version 2.1.5

Version 2.1.5 was released March 4, 2015

Use the Ruby 1.9 hash syntax in the **`validates_format_of :email`** statement.

Minor improvements and corrections of typos.

Version 2.1.4

Version 2.1.4 was released January 3, 2015

Updated references to Ruby from version 2.1.5 to 2.2.0.

Specify the "v0" version of the google_drive gem in the "Spreadsheet Connection" chapter.

Version 2.1.3

Version 2.1.3 was released December 25, 2014

Updated references to Rails 4.1.8 to Rails 4.2.0.

Version 2.1.2

Version 2.1.2 was released December 4, 2014

Released for sale as a Kindle book on Amazon, with new cover art (same cat, though).

RailsApps Tutorials now named the Capstone Rails Tutorials.

Updated references to Ruby from version 2.1.3 to 2.1.5.

Updated references to Rails 4.1.6 to Rails 4.1.8 (minor releases with bug and security fixes).

Removed link to the (now defunct?) Lowdown web application in the "Plan Your Product" chapter.

Changes to the "Asynchronous Mailing" section of "Send Mail" chapter to describe Active Job in Rails 4.2.

Minor improvements to the "Dynamic Home Page," "Deploy," "Configure," "Troubleshoot," and "Create the Application" chapters.

Version 2.1.1

Version 2.1.1 was released October 22, 2014

Minor rewriting for clarity.

Updated "Precompile Assets" section of the "Deploy" chapter.

Mentioned explainshell.com in the "Get Started" chapter.

Mentioned Zeal as a Linux alternative to Dash.

Recommended book Practicing Rails by Justin Weiss.

Version 2.1.0

Version 2.1.0 was released October 12, 2014

Updated references to Ruby from version 2.1.1 to 2.1.3.

Updated references to Rails 4.1.1 to Rails 4.1.6 (minor releases with bug and security fixes).

Four new chapters:

- "Testing"

- "Rails Composer"

- "Crossing the Chasm"

- "Level Up"

Use ActiveModel instead of the activerecord-tableless gem.

In the "Configuration" chapter, add a note to use spaces (not tabs) in the **config/secrets.yml** file.

Updated "Gems" chapter to add a troubleshooting note to the "Install the Gems" section (about errors with the Nokogiri gem).

Added a section on "Multiple Terminal Windows" to the "Create the Application" chapter.

In the "Get Help When You Need It" chapter, updated the list of recommended newsletters, replaced rubypair.com with codermatch.me, and added a section on code review. Removed reference to defunct Rails Development Directory.

Version 2.0.2

Version 2.0.2 was released May 6, 2014

Updated references to Rails 4.1.0 to Rails 4.1.1 (a minor release with a security fix).

For Nitrous.io users, clarify that "http://localhost:3000/" means the Preview browser window.

Update "Gems" chapter, section "Where Do Gems Live?" to add more explanation.

Minor change to code in the "Mailing List" chapter, setting 'mailchimp_api_key' explicitly when instantiating Gibbon, for easier troubleshooting.

Version 2.0.1

Version 2.0.1 was released April 16, 2014

Minor updates for Rails 4.1.0. Mostly small changes to the "Configure" and "Front-End Framework" chapters.

Added an explanation that, in the **config/secrets.yml** file, `domain_name` doesn't have to be kept secret and set as a Unix environment variable.

Added a hint about passwords that use punctuation marks (plus a completely irrelevant note about profanitype).

Replaced `Rails.application.secrets.gmail_username` with `Rails.applicatic` Also replaced `gmail_password` with `email_provider_password`. Just trying to make things a little more generic in case Gmail is not used as a provider.

Added a section explaining the horrid details of the `config.assets.precompile` configuration setting in the **config/application.rb** file. Please convey my displeasure to those responsible for subjecting beginners to this travesty.

In the "Deploy" chapter, restored `RAILS_ENV=production rake assets:precompile` because Rails 4.1.0 no longer barfs on this.

Added resources to the "Get Help When You Need It" chapter.

Minor rewriting of the introduction.

Version 2.0.0

Version 2.0.0 was released April 8, 2014

Updated references to Ruby from version 2.1.0 to 2.1.1.

Updated the book to Rails 4.1. The application name is no longer used in the **config/routes.rb** file.

Rails 4.1 changes the **app/assets/stylesheets/application.css.scss** file. Updated the "Front-End Framework" chapter. Also expanded the explanation of the Foundation grid.

In Rails 4.1, configuration variables are set in the **config/secrets.yml** file. The Figaro gem is dropped, along with the **config/application.yml** file. Updated the "Configure" chapter and references to configuration variables throughout the book.

In the "Deploy" chapter, changed `RAILS_ENV=production rake assets:precompile` to `rake assets:precompile` to avoid the error "database configuration does not specify adapter."

Updated "The Parking Structure" chapter with comments about "Folders of Future Importance" that experienced developers often use: **test/**, **spec/**, **features/**, **policies/**, and **services/**. Updated the "Spreadsheet Connection" chapter to mention service-oriented architectures (SOA).

Extended the section on "Limitations of Metaphors" in the "Just Enough Ruby" chapter to include the example of gender when modeling a person.

Minor rewriting for clarity throughout.

Version 1.19

Version 1.19 was released February 1, 2014

Updated the book to use Foundation 5.0. Foundation 5.0.3 was released January 15, 2014 (earlier versions 5.0.1 and 5.0.2 were incompatible with Rails Turbolinks and the Rails asset pipeline). Changed the Gemfile to remove **gem 'compass-rails'** and replace **gem 'zurb-foundation'** with **gem 'foundation-rails'**. Updated a line in the "Front-End Framework" chapter for Foundation 5.0:

```
$ rails generate layout foundation5 --force
```

The files **navigation.html.erb** and **application.html.erb** are changed for Foundation 5.0. The Bootstrap front-end framework is now independent of Twitter, so I call it "Bootstrap" not "Twitter Bootstrap." Revised the chapter "Just Enough Ruby" to incorporate suggestions from technical editor Pat Shaughnessy. Revised the chapter "Request and Response" to incorporate suggestions from technical editor Kirsten Jones. Minor rewriting for clarity throughout.

Version 1.18

Version 1.18 was released January 10, 2014

Updated references to Ruby from version 2.0.0 to 2.1.0. Changed one line in the "Front-End Framework" chapter to accommodate a change in the rails_layout gem version 1.0.1. The command was:

```
$ rails generate layout foundation4 --force
```

Changed to:

```
$ rails generate layout:install foundation4 --force
```

Updated the "Configure" chapter to add ActionMailer configuration values to the file **config/environments/development.rb**.

Version 1.17

Version 1.17 was released December 21, 2013

Updated Rails version from 4.0.1 to 4.0.2 .

Changed Gemfile to remove **gem 'compass-rails', '> 2.0.alpha.0'** and replace it with **gem 'compass-rails', '> 1.1.2'**. The 2.0.alpha.0 version was yanked from the RubyGems server. The compass-rails gem is needed for Foundation 4.3. It will not be needed for Foundation 5.0.

Changed Gemfile to replace **gem 'zurb-foundation'** with **gem 'zurb-foundation' '> 4.3.2'**. Foundation 5.0 will require **gem 'foundation-rails'** but we can't use it until an incompatibility with Turbolinks is resolved. So we will stick with Foundation 4.3.2 for now.

Revised code in the "Analytics" chapter. Using **ready page:change** instead of **page:load** to accommodate Turbolinks. Updated the **segmentio.js** file to use a new tracking script from Segment.io. Updated instructions for setting up Google Analytics tracking on Segment.io. Added concluding paragraphs "Making Mr. Kadigan Happy" to the "Analytics" chapter.

Minor clarification in the "Front-End Framework" chapter to explain that the navigation bar won't show a dropdown menu until the next chapter, when we add navigation links.

Minor clarification in the "Spreadsheet Connection" chapter to explain that Google may block access if you attempt access from a new and different computer (including Nitrous.io).

Added cat names in the "Credits and Comments" chapter.

Revised "Getting Help" chapter and added "Version Notes" chapter.

Minor clarifications, plus fixes for various typos and insignificant errors.

Chapter 17

Credits and Comments

Was the book useful to you? Follow [@rails_apps](#) on Twitter and tweet some praise. I'd love to know you were helped out by the tutorial.

You can find me on [Facebook](#) or [Google+](#). I'm happy to connect if you want to stay in touch.

If you'd like to recommend the book to others, the landing page for the book is here:

- http://learn-rails.com/learn-ruby-on-rails.html

I'd love it if you mention the book online, whether it is a blog post, Twitter, Facebook, or online forums. Recommending the book with a link makes it easier for people to find the book.

Credits

The book was created with the encouragement, financial support, and editorial assistance of hundreds of people in the Rails community.

Daniel Kehoe wrote the book and implemented the application.

Financial Backers

The following individuals provided financial contributions of over $50 to help launch the book. Please join me in thanking them for their encouragement and support.

Al Zimmerman, Alan W. Smith, Alberto A. Colón Viera, Andrew Terry, Avi Flombaum, Brian Hays, Charles Treece, Dave Doolin, Denzil Villarico, Derek Rockwell, Eito Katagiri, Evan Sparkman, Frank Castle, Fred Dixon, Fred Schoeneman, Gant Laborde, Gardner Monks, Gerard de Brieder, GoodWorksOnEarth.org, Hanspeter Leupin, Harald Lazardzig, Harsh Patel, James Bond, Jared Koumentis, Jason Landry, Jeff Whitmire, Jesse House, Joe Wilmoth Jr., John Shannon, Joost Baaij, Juan Cristobal Pazos, Kathleen Sidenblad, Laird Hayward, Logan Hasson, Ludovic Kuty, Mark Gilbert, Matt Esterly, Mike Gilbert, Niko Roberts, Norman Cohen, Paul Philippov, Robert Nadar, Rogier Hof, Ross Kinney, Ruben Calzadilla, Stephane Moreau, Susan Wilson, Sven Fuchs, Thomas Nitsche, Tom Michel, Youn Shin Kang, Yuen Lock

Editors and Proofreaders

Dozens of volunteers offered corrections and made suggestions, from fixing typos to advice about organizing the chapters.

Alberto Dubois Ribó, Alex Finnarn, Alex Zielonko, Alexandru Muntean, Alexey Dotokin, Alexey Ershov, André Arko, Andreas Basurto, Ben Swee, Brandon Schabel, Cam Skene, Daniella Zimmermann, Dapo Babatunde, Dave Levine, Dave Mox, David Kim, Duany Dreyton Bezerra Sousa, Erik Trautman, Erin Nedza, Flavio Bordoni, Fritz Rodriguez Jr, Hendri Firmana, Ishan Shah, James Hamilton, Jasna Vukovic, Jeremy Schneider, Joanne Daudier, Joel Dezenzio, Jonah Ruiz, Jonathan Lai, Jonathan Miller, Jordan Stone, Joreal Whitfield, Josh Morrow, Joyce Hsu, Julia Mokus, Julie Hamwood, Jutta Frieden, Laura Pierson Wadden, Marc Ignacio, Mark D. Blackwell, Mark Everhart, Michael Wong, Miguel Herrera, Mike Janicki, Miran Omanovic, Neha Jain, Norman Cohen, Oana Sipos, Peter Rangelov, Richard Afolabi, Robin Paul, Roderick Silva,

Sakib Ash, Sebastian Lobato Genco, Silvia Obajdin, Stas Sucov, Stefan Streichsbier, Sven Fuchs, Tam Eastley, Tim Goshinski, Timothy Jones, Tom Connolly, Tom Michel, Tomas Olivares, Verena Brodbeck, Will Schive, William Yorgan, Zachary Davy

Photos

Images provided by the lorempixel.com service are used under the Creative Commons license. Visit the Flickr accounts of the photographers to learn more about their work:

- photo of a white cat by Tomi Tapio

- photo of a cat by Steve Garner

- photo of a cat by Ian Barbour

The photo of a fluffy white cat by Tomi Tapio is used in the application.

Comments

I regularly update the book. Your comments and suggestions for improvements are welcome.

Feel free to email me directly at daniel@danielkehoe.com.

Are you stuck with code that won't work? Stack Overflow provides a question-and-answer forum for readers of this book. Use the tag "learn-ruby-on-rails" or "railsapps" when you post your question.

Found a bug in the tutorial application? Please create an issue on GitHub.