Содержание

Предисловие

Моя бывшая компания (CD Baby) была одной из первых, громогласно заявивших о переходе на Ruby on Rails, а затем еще громче возвестившей о возврате на PHP (Google расскажет вам об этой драме). Эту книгу, написанную Майклом Хартлом, так настоятельно рекомендовали, что я должен был прочитать ее, и именно благодаря книге «Учебник Ruby on Rails» я вернулся к Rails.

На своем пути я встречал много книг о Rails, и эта – одна из тех немногих, что, наконец, «зацепила» меня. Здесь все делается способом, типичным для Rails, который прежде казался мне крайне неестественным, но теперь, после изучения этой книги, пришло ощущение комфорта и естественности этого подхода. К тому же это единственная книга о Rails, которая соблюдает методику разработки через тестирование на всем своем протяжении; именно этот подход строго рекомендуется специалистами, но ранее я нигде не встречал такой его отчетливой демонстрации. Наконец, включив в примеры использование таких инструментов, как Git, GitHub и Heroku, автор дает почувствовать, что из себя представляет разработка проектов в реальном мире. Учебные примеры в этой книге не являются разрозненными фрагментами кода.

Линейное повествование – отличный формат. Лично я прочитал «Учебник Ruby on Rails» за три полных дня[1], выполняя все примеры и задачи в конце каждой главы. Делайте все от начала и до конца, не перескакивая от одной темы к другой, и вы получите максимальную пользу.

Наслаждайтесь!

Derek Sivers (sivers.org),
основатель CD Baby

---

[1] Это нестандартная ситуация! Обычно прохождение всего учебника занимает *гораздо* больше, чем три дня.

Благодарности

«Учебник Ruby on Rails» во многом обязан своим появлением моей предыдущей книге по Rails – «Rails Space» и, следовательно, моему соавтору Орилиусу Прочазка (Aurelius Prochazka). Я хотел бы поблагодарить Орилиуса и за работу над прошлой книгой, и за поддержку этой. Я также хочу поблагодарить Дебру Уильямс Коли (Debra Williams Cauley), редактора обеих этих книг; пока она будет брать меня с собой на бейсбол, я буду продолжать писать книги для нее.

Я хотел бы поблагодарить многих фанатов Ruby, учивших и вдохновлявших меня на протяжении многих лет. Это: Дэвид Хейнмейер Ханссон (David Heinemeier Hansson), Йехуда Кац (Yehuda Katz), Карл Лерхе (Carl Lerche), Джереми Кемпер (Jeremy Kemper), Ксавье Нория (Xavier Noria), Райан Бейтс (Ryan Bates), Джеффри Грозенбах (Geoffrey Grosenbach), Питер Купер (Peter Cooper), Мэтт Аимонетти (Matt Aimonetti), Марк Бейтс (Mark Bates), Грегг Поллак (Gregg Pollack), Вейн Е. Сегуин (Wayne E. Seguin), Ами Хой (Amy Hoy), Дэйв Челимски (Dave Chelimsky), Пэт Мэддокс (Pat Maddox), Том Престон-Вернер (Tom Preston-Werner), Крис Ванстрат (Chris Wanstrath), Чад Фаулер (Chad Fowler), Джош Сассер (Josh Susser), Оби Фернандес (Obie Fernandez), Ян Мак-Фарланд (Ian McFarland), Стивен Бристоль (Steven Bristol), Пратик Найк (Pratik Naik), Сара Мей (Sarah Mei), Сара Аллен (Sarah Allen), Вольфрам Арнольд (Wolfram Arnold), Алекс Чеффи (Alex Chaffee), Джилс Бокет (Giles Bowkett), Эван Дорн (Evan Dorn), Лонг Нгуен (Long Nguyen), Джеймс Линденбаум (James Lindenbaum), Адам Уиггинс (Adam Wiggins), Тихон Бернстам (Tikhon Bernstam), Рон Эванс (Ron Evans), Уайат Грин (Wyatt Greene), Майлз Форрест (Miles Forrest), Санди Метц (Sandi Metz), Райан Дэвис (Ryan Davis), Аарон Паттерсон (Aaron Patterson), сотрудники Pivotal Labs, команда Heroku, ребята из thoughtbot и команда GitHub. Наконец, многих, многих читателей – слишком многих, чтобы здесь перечислить их всех, – внесших большое количество предложений по улучшению и сообщивших об ошибках во время работы над этой книгой, и я с благодарностью признаю, что она получилась настолько хорошей во многом благодаря им.

Об авторе

Майкл Хартл – автор книги «Учебник Ruby on Rails», одного из лучших введений в веб-разработку, сооснователь Softcover – платформы для упрощенной публикации электронных книг. Его предыдущий опыт включает написание учебника «RailsSpace», ныне серьезно устаревшего, и разработку Insoshi, некогда популярной платформы создания социальных сетей, написанной на Ruby on Rails. В 2011 году Майкл был награжден премией «Ruby Hero Award» за вклад в развитие сообщества пользователей Ruby. Закончил Гарвардский колледж, имеет степень кандидата физических наук, присвоенную в Калифорнийском технологическом институте, и является выпускником предпринимательских курсов Y Combinator.

Авторские права и лицензия

Глава 1

От нуля к развертыванию

Добро пожаловать в «Учебник Ruby on Rails: Изучаем разработку веб-приложений на основе Rails». Цель данной книги – научить вас разрабатывать собственные веб-приложения с использованием популярного фреймворка Ruby on Rails. Если эта тема для вас в новинку, «Учебник Ruby on Rails» даст вам полное представление о разработке веб-приложений, включая основополагающие знания о Ruby, Rails, HTML и CSS, базах данных, управлении версиями, тестировании и развертывании – достаточно для начала вашей карьеры веб-разработчика или предпринимателя в сфере компьютерных технологий. С другой стороны, если вы уже знакомы с веб-разработкой, эта книга поможет вам освоить основы фреймворка Rails, включая MVC и REST, генераторы, миграции, маршрутизацию и встроенный язык Ruby. В любом случае, когда вы закончите чтение этой книги, вы будете готовы извлечь максимум пользы из более продвинутых книг, блогов и скринкастов (видеоуроков), являющихся частью обширной образовательной экосистемы[1].

В данном учебнике используется комплексный подход к веб-разработке: в процессе обучения мы напишем три примера приложений с возрастающей сложностью; в разделе 1.3 мы начнем создание простенького приложения, затем, в главе 2, перейдем к более продвинутому и, наконец, в главах с 3 по 12 полностью сосредоточимся на большом учебном примере. Как можно догадаться, примеры приложений в этом учебнике не привязаны к какой-либо категории веб-сайтов; хотя заключительный пример будет иметь значительное сходство с весьма популярной социальной сетью Twitter (которая, по совпадению, изначально была написана на Rails). Основное внимание здесь будет уделяться общим принципам разработки, поэтому полученные знания станут для вас надежным фундаментом, вне зависимости от того, какие виды приложений вы будете создавать.

Я часто слышу вопрос: «Какими знаниями нужно обладать для изучения веб-разработки с этим учебником?» Как мы увидим в разделе 1.1.1, веб-разработка – сложная тема, особенно для полных новичков. Хотя изначально учебник предназначался для читателей с некоторым опытом программирования, он нашел

---

[1] Самую свежую версию книги (на англ. языке. – *Прим. перев.*) можно найти на сайте http://www.railstutorial.org/. Если вы читаете печатную версию книги, сверьте версию своего экземпляра с онлайн-версией на http://www.railstutorial.org/book.

множество читателей среди начинающих разработчиков. Учитывая это, в данном (третьем) издании значительно снижен порог входа в разработку с Ruby on Rails (см. блок 1.1).

Блок 1.1 ❖ Снижение порога входа

С целью снижения порога входа для начинающих разработчиков в третьем издании сделано следующее:

- использована стандартная среда разработки, размещенная в облаке (раздел 1.2), что позволяет обойти множество проблем, связанных с установкой и настройкой новой системы;
- использован «предустановленный стек» Rails, включающий встроенный фреймворк тестирования MiniTest;
- уменьшено количество внешних зависимостей (RSpec, Cucumber, Capybara, Factory Girl);
- использован облегченный и более гибкий подход к тестированию;
- исключены сложные в настройке варианты (Spork, RubyTest);
- меньше внимания уделяется особенностям, свойственным конкретной версии Rails, и больше – общим принципам веб-разработки.

Я надеюсь, что эти изменения сделают третье издание учебника доступным для еще более широкой аудитории, чем предыдущие.

В этой первой главе мы начнем изучение фреймворка Ruby on Rails с установки необходимого программного обеспечения и настройки рабочего окружения (раздел 1.2). Затем создадим первое Rails-приложение с названием `hello_app`. В книге «Учебник Ruby on Rails» особое значение придается хорошим привычкам в программировании, поэтому сразу после создания нового Rails-проекта мы поместим его в систему управления версиями Git (раздел 1.4). И, верите вы в это или нет, в этой главе мы даже развернем наше первое приложение в Сети (раздел 1.5).

В главе 2 мы создадим второй проект для демонстрации основ работы Rails-приложения. Для быстроты в этом *мини-приложении* (с названием `toy_app`) мы применим прием «скаффолдинга» (scaffolding, см. блок 1.2) для генерации кода. Поскольку код получится одновременно и сложным, и уродливым, мы оставим его в стороне и сосредоточимся на взаимодействии с приложением через его идентификатор *URI* (который часто называют адресом *URL*)[1], с использованием веб-браузера.

Остальная часть учебника описывает разработку единственного большого учебного приложения (с названием `sample_app`), весь код которого будет написан с нуля. При этом мы будем использовать фиктивные объекты, приемы разработки через тестирование (Test-Driven Development, TDD) и интеграционные тесты. В главе 3 мы сначала создадим статические страницы, а затем добавим в них не-

[1] Аббревиатура URI расшифровывается как Uniform Resource Identifier (универсальный идентификатор ресурсов), а аббревиатура URL – как Uniform Resource Locator (универсальный указатель ресурсов). На практике URL обычно соответствует тому, что «находится в адресной строке браузера».

много динамического контента. В главе 4 мы совершим небольшое турне по языку Ruby, лежащему в основе Rails. Затем, в главах с 5 по 10, завершим базовую часть приложения, создав макет сайта, модель данных пользователя, а также систему регистрации и аутентификации (включая активацию учетной записи и сброс пароля). Наконец, в главах 11 и 12 мы добавим поддержку микроблоггинга и немного социальных функций, чтобы сделать рабочий пример сайта.

Блок 1.2 ❖ Скаффолдинг: быстрее, проще, заманчивее

С самого начала фреймворк Rails привлек к себе внимание знаменитым видеороликом от Дэвида Хейнмейера Ханссона (David Heinemeier Hansson) – создателя Rails, демонстрирующим создание микроблога за 15 минут. Этот и другие подобные видеоролики – отличный способ познакомиться с возможностями Rails, и я рекомендую посмотреть их. Но предупреждаю: они совершают свой удивительный пятнадцатиминутный подвиг, используя функцию под названием «скаффолдинг» (scaffolding), которая в значительной степени опирается на код, волшебным образом сгенерированный Rails-командой `generate scaffold`.

Работая над учебником по Ruby on Rails, мне приходилось бороться с искушением положиться на эту функцию, потому что это быстрее, проще, заманчивее. Но сложность и огромный объем кода, производимый генератором, могли стать непреодолимым препятствием для начинающего Rails-разработчика; вы, вероятно, сможете использовать его, но почти наверняка не сможете понять. Увлекшись скаффолдингом, вы рискуете превратиться в виртуозного генератора сценариев с весьма поверхностным знанием Rails.

В этой книге мы будем придерживаться (почти) полярно противоположного подхода: хотя в главе 2 создадим мини-приложение, сгенерировав код автоматически, основой учебника все же является приложение, которое мы начнем писать в главе 3. На каждом этапе его разработки мы будем писать небольшие куски кода, достаточно простые для понимания, но все же требующие некоторых усилий для их усвоения. Результатом станет более глубокое понимание Rails, которое, в свою очередь, даст вам хорошую базу для написания практически любых типов веб-приложений.

1.1. Введение

«Ruby on Rails» (или просто «Rails») – это фреймворк для разработки веб-приложений на языке программирования Ruby. Со времен своего дебюта в 2004 году Ruby on Rails довольно быстро стал одним из самых мощных и популярных инструментов создания динамических веб-приложений. Rails используется множеством различных компаний: Airbnb, Basecamp, Disney, GitHub, Hulu, Kickstarter, Shopify, Twitter и Yellow Pages. Помимо этого, существует множество компаний, занимающихся разработкой веб-приложений и специализирующихся на Rails, таких как ENTP, thoughtbot, Pivotal Labs, Hashrocket и HappyFunCorp, плюс бесчисленное множество независимых консультантов, преподавателей и индивидуальных разработчиков.

Что же делает Rails таким замечательным? Во-первых, Ruby on Rails – это открытый исходный код, доступный на условиях лицензии MIT, и, как следствие, его можно загружать и использовать совершенно бесплатно. Rails также обязан своим успехом изящному и компактному дизайну. Используя податливость языка Ruby, лежащего в его основе, Rails фактически определяет предметно-ориентированный язык для разработки веб-приложений. В результате множество часто встречающихся задач веб-программирования, таких как динамическое создание разметки HTML, определение моделей данных и маршрутизация URL, легко решаются в Rails, а конечный код приложений получается кратким и читаемым.

Rails также быстро адаптируется к новым тенденциям в веб-технологиях. Например, Rails одним из первых полностью реализовал архитектурный стиль REST структурирования веб-приложений (мы будем изучать его на всем протяжении учебника). И когда в других фреймворках появляются новые успешные приемы, создатель Rails, Дэвид Хейнмейер Ханссон (David Heinemeier Hansson), и рабочая группа Rails не стесняются использовать чужой опыт. Пожалуй, наиболее драматичным примером является слияние Rails с конкурирующим веб-фреймворком Merb, благодаря которому Rails получил модульный дизайн Merb, стабильный API и улучшенную производительность.

Наконец, вокруг Rails сплотилось необычайно увлеченное и разнообразное сообщество: сотни разработчиков, представительные конференции, огромное количество гемов (пакетов, законченных решений конкретных задач, таких как постраничный вывод и выгрузка изображений), богатый набор информативных блогов и рог изобилия форумов и IRC-каналов. Большое количество Rails-программистов также облегчает работу с (неизбежными) ошибками, возникающими в процессе разработки: алгоритм «Ищи в Google сообщение об ошибке» практически всегда помогает найти подходящую статью в блоге или сообщение на форуме.

1.1.1. Необходимая квалификация

Формально эта книга не требует некоторого набора необходимых знаний. Она содержит учебные сведения не только о фреймворке Rails, но также о лежащем в его основе языке Ruby, встроенном фреймворке тестирования (MiniTest), командной строке Unix, HTML, CSS, немного JavaScript и даже чуть-чуть SQL. Это очень объемный материал, и потому для работы с учебником желательно иметь некоторый опыт использования HTML и программирования. Однако данный учебник использовало так много новичков для изучения веб-разработки с нуля, что даже если у вас мало опыта, я все же советую попробовать. Если содержимое учебника покажется сложным, вы всегда можете сделать шаг назад и начать с одного из ресурсов, перечисленных ниже. Другая возможная стратегия (рекомендованная многими читателями): прочитать учебник дважды; возможно, вы удивитесь, как много узнали в первый раз (и насколько проще читать учебник по второму кругу).

Стоит ли вначале изучить Ruby? Ответ зависит от вашего личного стиля обучения и опыта в программировании. Если вы предпочитаете изучать все систематически, с самых основ, или прежде никогда не программировали, вам, возмож-

но, стоит начать с изучения Ruby, и в этом случае я рекомендую книгу «Учись программировать» Криса Пайна (Chris Pine)[1] и «Начало Ruby» Питера Купера (Peter Cooper)[2]. С другой стороны, множество начинающих Rails-разработчиков имеет своей целью создание веб-приложений и, скорее, предпочтет не корпеть над толстенной книгой, чтобы написать одну-единственную веб-страницу. В этом случае я рекомендую поработать с коротким интерактивным учебником «Try Ruby»[3], чтобы получить общее представление о Ruby. Если и после этого учебник окажется для вас слишком сложным, попробуйте начать с «Learn Ruby on Rails» Дэниэла Кехо (Daniel Kehoe)[4] или «One Month Rails»[5] – обе книги ориентированы на начинающих, в отличие от данного учебника.

Независимо от того, где вы начнете, к концу этого учебника вы будете готовы к чтению других, более продвинутых ресурсов по Rails. Вот некоторые из тех, которые я могу рекомендовать:

- Code School[6]: хорошие интерактивные онлайн-курсы по программированию;
- Turing School of Software & Design[7]: очные, 27-недельные курсы в Денвере, Колорадо. Читатели этой книги могут получить скидку в $500, воспользовавшись промокодом RAILSTUTORIAL500;
- Tealef Academy[8]: хорошая образовательная онлайн-площадка, посвященная Rails-разработке (включая продвинутый материал);
- Thinkful[9]: онлайн-курс в паре с профессиональным инженером, работа по учебному плану, основанному на конкретном проекте;
- RailsCastsот[10] Райана Бейтса (Ryan Bates): отличные (в основном бесплатные) видеоуроки по Rails;
- RailsApps[11]: большой выбор подробных проектов и учебников, посвященных разным темам;
- Rails Guides[12]: актуальные руководства по Rails.

1.1.2. Типографские соглашения

Соглашения в этой книге главным образом очевидны. В этом разделе я упомяну лишь те, которые таковыми не являются.

---

1 http://www.shokhirev.com/mikhail/ruby/ltp/title.html. – *Прим. перев.*
2 http://www.amazon.com/gp/product/1430223634.
3 http://tryruby.org/.
4 http://learn-rails.com/learn-ruby-on-rails.html.
5 http://mbsy.co/7Zdc7.
6 https://my.getambassador.com/.
7 https://www.turing.io/friends/tutorial.
8 https://launchschool.com/.
9 https://www.thinkful.com/a/railstutorial.
10 http://railscasts.com/.
11 https://tutorials.railsapps.org/.
12 http://rusrails.ru/.

В этой книге присутствует множество примеров применения командной строки. Для простоты все они используют приглашение в стиле Unix (знак доллара):

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

Как отмечено в разделе 1.2, я рекомендую пользователям всех операционных систем (особенно Windows) использовать облачную среду разработки (раздел 1.2.1), в которой есть встроенная командная строка Unix (Linux). Это особенно полезно, потому что в Rails имеется множество команд, которые можно запустить из командной строки. Например, в разделе 1.3.2 мы запустим локальный веб-сервер командой **`rails server`**:

```
$ railsserver
```

По аналогии с приглашением командной строки, здесь используется соглашение о разделителях каталогов в стиле Unix (то есть прямого слеша /). Например, конфигурационный файл **`production.rb`** учебного приложения будет представлен как:

```
config/environments/production.rb
```

Этот путь к файлу следует интерпретировать как относительный, начинающийся в корневом каталоге приложения, абсолютный путь к которому зависит от системы; в облачной интегрированной среде разработки (IDE) (раздел 1.2.1) этот путь выглядит так:

```
/home/ubuntu/workspace/sample_app/
```

То есть полный путь к `production.rb` имеет вид:

```
/home/ubuntu/workspace/sample_app/config/environments/production.rb
```

Для простоты я часто буду опускать путь к корневому каталогу приложения и писать просто: **`config/environments/production.rb`**.

В учебнике нередко демонстрируется вывод разных программ (команд, системы управления версиями, программ на Ruby и т. д.). Из-за неисчислимого количества небольших различий между компьютерными системами вывод, который вы увидите, возможно, не всегда в точности совпадет с тем, что показан в тексте, но это не повод для беспокойства. Некоторые команды могут вызывать ошибки, связанные с особенностями вашей системы, поэтому, чтобы не решать сизифову задачу документирования всех таких погрешностей, я отсылаю вас к алгоритму «ищи сообщение об ошибке в Google», который, между прочим, является хорошей практикой для программирования. Если вы столкнетесь с проблемами в процессе обучения, я советую обратиться к ресурсам, список которых приводится в справочном разделе[1].

[1] https://www.railstutorial.org/#help.

Поскольку в учебнике (помимо всего прочего) рассматривается вопрос тестирования Rails-приложений, часто бывает полезно знать, что данный кусок кода приводит к неудаче (обозначается красным цветом) или, наоборот, к успеху тестирования (обозначается зеленым цветом). Для удобства код помечен соответственно **КРАСНЫМ** и **ЗЕЛЕНЫМ**.

К каждой главе учебника прилагаются упражнения, выполнение которых необязательно, но рекомендуется. Чтобы сделать основной текст независимым от упражнений, решения обычно не включаются в последующие листинги кода. В редких случаях, когда решение упражнения используется в дальнейшем, оно будет приведено в основном тексте.

Наконец, для удобства в книге применяются два соглашения, облегчающие понимание большого количества примеров кода. Во-первых, в некоторых листингах выделены одна или несколько строк, как показано ниже:

```
class User < ActiveRecord::Base
  validates :name, presence: true
validates :email, presence: true
end
```

Такое выделение обычно указывает на наиболее важный новый код в данном примере и часто (хотя и не всегда) отражает разницу между этим и предыдущим листингом. Во-вторых, для краткости и простоты во многих листингах в книге используются вертикальные точки, например:

```
class User < ActiveRecord::Base
  .
  .
  .
  has_secure_password
end
```

Эти точки обозначают пропущенный код, и их не нужно копировать буквально.

1.2. За работу

Установка Ruby, Rails и всего сопутствующего программного обеспечения поддержки может привести в отчаяние даже опытных Rails-разработчиков. Проблема осложняется большим разнообразием окружений: разные операционные системы, версии, предпочтения в выборе текстовых редакторов и интегрированных сред разработки (IDE) и т. д. Пользователи, уже установившие среду разработки, могут сразу переходить к настройкам, а новым пользователям (как указано в блоке 1.1) я предлагаю избежать проблем с установкой и настройкой за счет использования *облачной интегрированной среды разработки*. Облачная IDE запускается в обычном браузере и, следовательно, одинаково хорошо работает на любой платформе, что особенно полезно для операционных систем (например, Windows), в которых разработка с Rails исторически была сложным занятием. Если, несмотря на все

предполагаемые трудности, вы по-прежнему хотите создать локальную среду разработки, я рекомендую следовать инструкциям на InstallRails.com[1].

1.2.1. Среда разработки

Принимая во внимание множество своеобразных настроек и предпочтений, число вариантов окружений разработки может равняться числу Rails-программистов. Чтобы избежать этих сложностей, я адаптировал данную книгу под превосходную облачную среду разработки Cloud9. В частности, мне было очень приятно сотрудничать с Cloud9, чтобы предложить вам среду разработки, ориентированную на потребности именно этого издания учебника. Получившееся рабочее пространство Cloud9 предоставляется предварительно настроенным, со всем программным обеспечением (включая Ruby, RubyGems, Git), необходимым для профессиональной разработки с Rails. (Отдельно мы будем устанавливать только сам фреймворк Rails, но это сделано намеренно (раздел 1.2.2).) Облачная IDE также включает три основных компонента, необходимых для разработки веб-приложений: текстовый редактор, навигатор файловой системы и терминал командной строки (рис. 1.1). Кроме того, ее текстовый редактор поддерживает глобальный поиск «Найти в файлах», который я считаю необходимым для навигации по любым большим

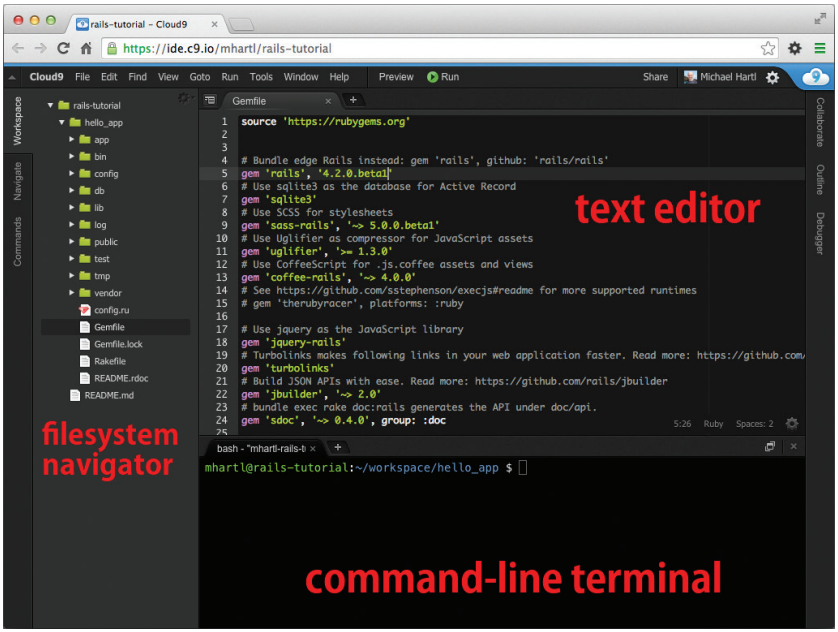

Рис. 1.1 ❖ Анатомия облачной IDE

[1] http://installrails.com/.

проектам Ruby или Rails[1]. И наконец, если вы решите использовать не только облачную IDE (я могу лишь приветствовать изучение других инструментов), на ее примере вы получите великолепное введение в основные возможности текстовых редакторов и других инструментов разработки.

Ниже приводится пошаговая инструкция, описывающая, как начать работу с облачной средой разработки:

- создайте бесплатную учетную запись в Cloud9[2];
- щелкните на ссылке **Goto your Dashboard** (Перейти к панели управления);
- щелкните на ссылке **Create New Workspace** (Создать новое рабочее пространство);
- как показано на рис. 1.2, создайте рабочее пространство `rails-tutorial` (не `rails_tutorial`), установите в настройках флажок **Private to the people I invite** (Только для приглашенных) и выберите значок **RailsTutorial** (не значок **Ruby on Rails**);

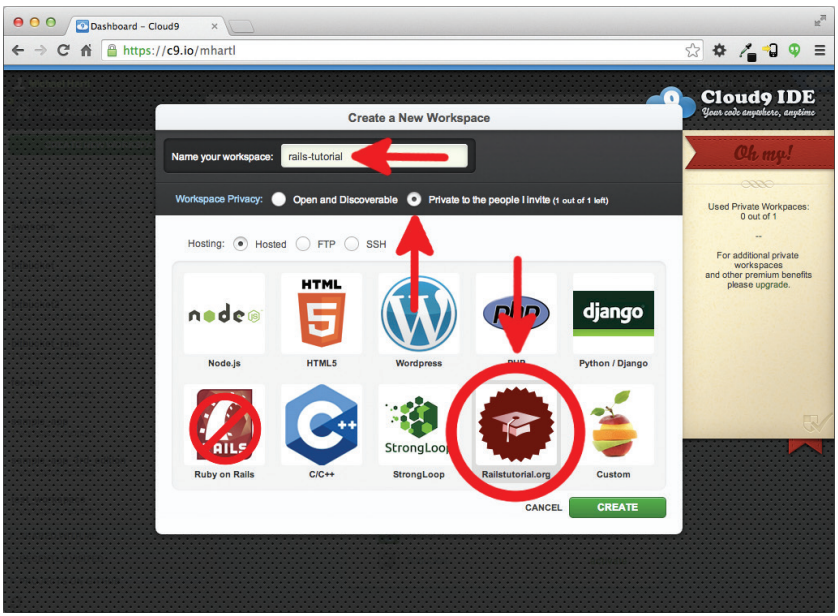

Рис. 1.2 ❖ Создание нового рабочего пространства в Cloud9

- щелкните на ссылке **Create** (Создать);
- после того как Cloud9 закончит подготовку рабочего пространства, выберите его и щелкните на ссылке **Start editing** (Начать редактирование).

---

[1] Например, чтобы найти определение функции `foo`, можно запустить глобальный поиск по фразе `def foo`.

[2] https://c9.io/web/sign-up/free.

Так как использование двух пробелов для отступов считается практически универсальным соглашением в Ruby, я также рекомендую изменить настройки редактора (по умолчанию используются четыре пробела). Как показано на рис. 1.3, щелкните на значке с изображением шестеренки в правом верхнем углу, выберите пункт **Code Editor (Ace)** (Редактор кода (Ace)) и измените параметр **Soft Tabs** (Мягкая табуляция). (Обратите внимание, что настройки применяются мгновенно и нет необходимости нажимать кнопку **Save** (Сохранить).)

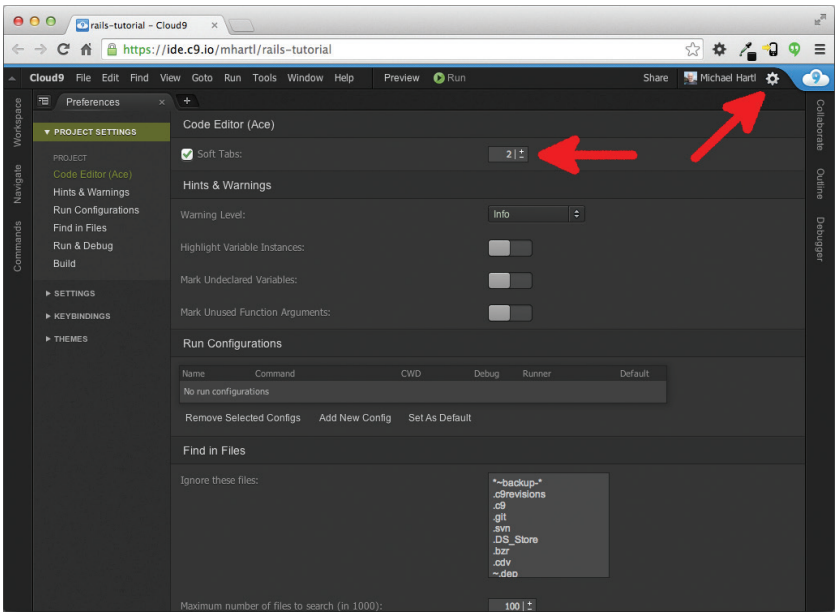

Рис. 1.3 ❖ Настройка использования двух пробелов для отступа в Cloud9

1.2.2. Установка Rails

Среда разработки, представленная в разделе 1.2.1, включает все ПО, необходимое для начала, кроме самого фреймворка Rails[1]. Для установки воспользуемся командой `gem` диспетчера пакетов *Ruby Gems*. Введите в командной строке терминала команду, как показано в листинге 1.1. (Если вы работаете в локальной системе, необходимо использовать обычное окно терминала; если в облачной IDE – область командной строки, как на рис. 1.1.)

Листинг 1.1 ❖ Установка Rails с определенным номером версии

```
$ gem install rails -v 4.2.0
```

[1] На данный момент в Cloud9 используется более старая версия Rails, и она несовместима с настоящим учебником, поэтому так важно установить его самостоятельно.

Параметр -v гарантирует установку определенной версии Rails. Это очень важно для получения результатов, совместимых с этим учебником.

1.3. Первое приложение

Следуя давним традициям в программировании, нашей первой целью будет создание программы, которая выводит фразу «hello, world» («Привет, мир!»). В частности, мы создадим простое приложение, которое отобразит строку «hello, world!» на веб-странице: в среде разработки на компьютере (раздел 1.3.4) и в Интернете (раздел 1.5).

Практически все приложения Rails начинаются одинаково – с команды `rails new`. Эта удобная команда создает скелет приложения Rails в выбранном каталоге. Тем, кто не стал использовать Cloud9 IDE, рекомендованную в разделе 1.2.1, сначала необходимо создать каталог `workspace` для проекта, если он еще не создан (листинг 1.2), и перейти в него. (В листинге 1.2 показаны команды `cd` и `mkdir`; если вы с ними пока не знакомы, прочитайте блок 1.3.)

Листинг 1.2 ❖ Создание каталога workspace для проекта (не нужно в облачной IDE)

```
$ cd              # Перейти в домашний каталог.
$ mkdir workspace # Создать каталог workspace.
$ cdworkspace/    # Перейти в каталог workspace.
```

Блок 1.3 ❖ Краткий курс командной строки Unix

Для читателей, пришедших из Windows или (в меньшей, но все еще значительной степени) из Macintosh OS X, командная строка Unix может выглядеть непривычной. К счастью, если вы используете рекомендованную облачную среду разработки, вы автоматически получаете доступ к командной строке Unix (Linux) в стандартной оболочке интерфейса, известной как Bash[1].

Основная идея командной строки проста: вводя короткие команды, пользователь может выполнить множество операций, таких как создание каталогов (`mkdir`), перемещение и копирование файлов (`mv` и `cp`), навигация в файловой системе за счет смены каталогов (`cd`). Командная строка может показаться примитивной тем, кто знаком в основном с графическим интерфейсом, внешность обманчива: это один из наиболее мощных инструментов в арсенале разработчика. Действительно, крайне редко удается увидеть рабочий стол опытного разработчика, на котором не было бы открыто несколько окон терминала с командной оболочкой.

Вообще говоря, это очень глубокая тема, но для следования за примерами в этом учебнике понадобится лишь несколько наиболее часто используемых команд, перечисленных в табл. 1.1. Для более глубокого изучения командной строки Unix читайте «Conquering the Command Line»[2] (Покоряя командную строку) Марка Бейтса (доступны бесплатная онлайн-версия и электронная книга с видеоуроками[3]).

---

1 https://ru.wikipedia.org/wiki/Bash.

2 http://conqueringthecommandline.com/book.

3 http://conqueringthecommandline.com/#pricing.

Следующий шаг в локальной и облачной системе – создание первого приложения, как показано в листинге 1.3. Обратите внимание, что в нем явно обозначен номер версии Rails (`_4.2.0_`) в виде части команды. Это гарантирует использование той же версии Rails для создания файловой структуры первого приложения, что была установлена в листинге 1.1. (Если команда из листинга 1.3 вернет ошибку, например `Could not find 'railties'` (Не могу найти 'railties'), значит, установлена неверная версия Rails, и необходимо перепроверить, была ли команда из листинга 1.1 выполнена в точности, как написано.)

Таблица 1.1 ❖ Некоторые часто используемые команды Unix

Описание	Команда	Пример
Список содержимого	`ls`	`$ ls -l`
Создать каталог	`mkdir <имя>`	`$ mkdir workspace`
Перейти в каталог	`cd <имя>`	`$ cd workspace/`
Перейти на один каталог вверх		`$ cd ..`
Перейти в домашний каталог		`$ cd ~ или просто $ cd`
Перейти в каталог внутри домашнего каталога		`$ cd ~/workspace/`
Переместить (переименовать) файл	`mv <источник><место назначения>`	`$ mv README.rdoc README.md`
Скопировать файл	`cp <источник><место назначения>`	`$ cp README.rdoc README.md`
Удалить файл	`rm <файл>`	`$ rm README.rdoc`
Удалить пустой каталог	`rmdir <каталог>`	`$ rmdir workspace/`
Удалить непустой каталог	`rm -rf <каталог>`	`$ rm -rf tmp/`
Объединить и показать содержимое файла	`cat <файл>`	`$ cat ~/.ssh/id_rsa.pub`

Листинг 1.3 ❖ Запуск railsnew (с указанием номера версии).

```
$ cd ~/workspace
$ rails _4.2.0_ new hello_app
      create
      create README.rdoc
      create Rakefile
      create config.ru
      create .gitignore
      create Gemfile
      create app
      create app/assets/javascripts/application.js
      create app/assets/stylesheets/application.css
      create app/controllers/application_controller.rb
      .
      .
      .
      create test/test_helper.rb
      create tmp/cache
```

```
      create tmp/cache/assets
      create vendor/assets/javascripts
      create vendor/assets/javascripts/.keep
      create vendor/assets/stylesheets
      create vendor/assets/stylesheets/.keep
         run bundle install
Fetching gem metadata from https://rubygems.org/..........
Fetching additional metadata from https://rubygems.org/..
Resolving dependencies...
Using rake 10.3.2
Using i18n 0.6.11
.
.
.
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is installed.
         run bundle exec spring binstub --all
* bin/rake: spring inserted
* bin/rails: spring inserted
```

В конце листинга 1.3 видно, что реализация `rails new` автоматически запускает команду `bundle install` по завершении создания файлов. В разделе 1.3.1 мы детально обсудим, что это значит.

Таблица 1.2 ❖ Обзор начальной структуры Rails-каталогов

Файл/Каталог	Назначение
app/	Основной код приложения, включает модели, представления, контроллеры и вспомогательные функции
app/assets	Ресурсы приложения, такие как каскадные таблицы стилей (CSS), JavaScript-файлы и изображения
bin/	Двоичные выполняемые файлы
config/	Конфигурация приложения
db/	Файлы базы данных
doc/	Документация для приложения
lib/	Модули библиотек
lib/assets	Библиотека ресурсов приложения, таких как каскадные таблицы стилей (CSS), JavaScript-файлы и изображения
log/	Файлы журналов приложения
public/	Общедоступные данные (например, через браузер), такие как страницы с описанием ошибок
bin/rails	Программа для автоматического создания программного кода, открытия консольных сеансов или запуска локального веб-сервера
test/	Тесты приложения
tmp/	Временные файлы
vendor/	Код сторонних разработчиков, например расширения и гемы

[1] http://rack.github.io/.

***Таблица 1.2** (окончание)*

Файл/Каталог	Назначение
vendor/assets	Сторонние ресурсы, такие как каскадные таблицы стилей (CSS), JavaScript-файлы и изображения
README.rdoc	Краткое описание приложения
Rakefile	Служебные задачи для выполнения командой `rake`
Gemfile	Гемы, необходимые приложению
Gemfile.lock	Список гемов, обеспечивающий использование одинаковых версий гемов всеми копиями приложения
config.ru	Файл конфигурации для Rack[1]
.gitignore	Шаблоны файлов, которые должны игнорироваться Git

Отметьте, как много файлов и каталогов создает команда `rails`. Эта стандартная структура файлов и каталогов (рис. 1.4) является одним из многих преимуществ Rails – немедленный переход от нуля к (минимально) действующему приложению. Кроме того, так как эта структура является общей для всех приложений Rails, вы легко будете ориентироваться в чужом коде. Обзор типовых Rails-файлов представлен в табл. 1.2; с большинством из них мы познакомимся в остальной части книги. В частности, в разделе 5.2.1 мы обсудим каталог `app/assets`, часть конвейера ресурсов, который значительно упрощает организацию и развертывание активно используемых файлов (ресурсов), таких как каскадные таблицы стилей и JavaScript-файлы.

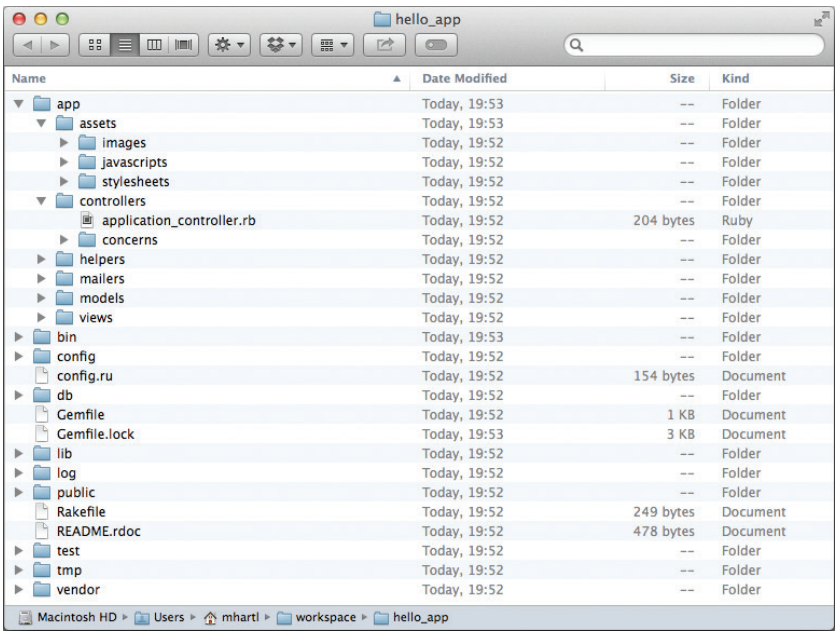

Рис. 1.4 ❖ Структура каталогов вновь созданного Rails-приложения

1.3.1. Компоновщик

Следующий этап после создания нового Rails-приложения – запуск *компоновщика* (bundler) для установки и включения необходимых гемов (пакетов). Как отмечалось в разделе 1.3, компоновщик автоматически запускается командой `rails` (как `bundle install`), но в этом разделе мы немного изменим комплект гемов приложения и вновь запустим компоновщик. Для этого нужно открыть файл `Gemfile` в текстовом редакторе. (В облачной IDE распахните дерево каталогов приложения в панели навигатора и дважды щелкните на файле `Gemfile`.) Некоторые детали и номера версий могут немного отличаться, но в целом результат должен выглядеть как на рис. 1.5 и в листинге 1.4. (Этот файл содержит программный код на языке Ruby, но не обращайте пока внимания на синтаксис; мы подробно поговорим о Ruby в главе 4.) Если файлы и каталоги выглядят иначе, чем на рис. 1.5, щелкните на значке шестеренки в навигаторе файлов и выберите пункт **Refresh File Tree** (Обновить дерево файлов). (Как правило, это необходимо делать всякий раз, когда дерево файлов выглядит не так, как ожидается.)

Листинг 1.4 ❖ Начальный Gemfile в каталоге hello_app

```
source 'https://rubygems.org'

# Включить новую версию Rails: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.0'
# Использовать sqlite3 как базу данных для Active Record
gem 'sqlite3'
# Использовать SCSS для таблиц стилей
gem 'sass-rails', '~> 5.0.1'
# Использовать Uglifier для сжатия JavaScript
gem 'uglifier', '>= 1.3.0'
# Использовать CoffeeScript для ресурсов .js.coffee и представлений
gem 'coffee-rails', '~> 4.0.0'
# Дополнительную информацию о других средах выполнения ищите
# по адресу: https://github.com/sstephenson/execjs#readme

# Использовать jquery как библиотеку JavaScript
gem 'jquery-rails'
# Turbolinks ускоряет следование по ссылкам. Дополнительную информацию
# ищите по адресу: https://github.com/rails/turbolinks
gem 'turbolinks'
# Упрощает сборку JSONAPI. Дополнительную информацию
# ищите по адресу: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc

# Использовать Active Model has_secure_password
# gem 'bcrypt', '~> 3.1.7'

# Использовать Unicorn как сервер приложений
# gem 'unicorn'
```

```
# Использовать Capistrano для развертывания
# gem 'capistrano-rails', group: :development

group :development, :testdo
  # Позволяет вызвать 'debugger' в любом месте в коде, чтобы остановить
  # выполнение и открыть консоль отладчика
  gem 'byebug'

  # Доступ к консоли IRB на странице exceptions и /console в разработке
  gem 'web-console', '~> 2.0.0.beta2'

  # Увеличивает скорость разработки, сохраняя приложение запущенным в фоне.
  # Подробности по адресу: https://github.com/rails/spring
  gem 'spring'
end
```

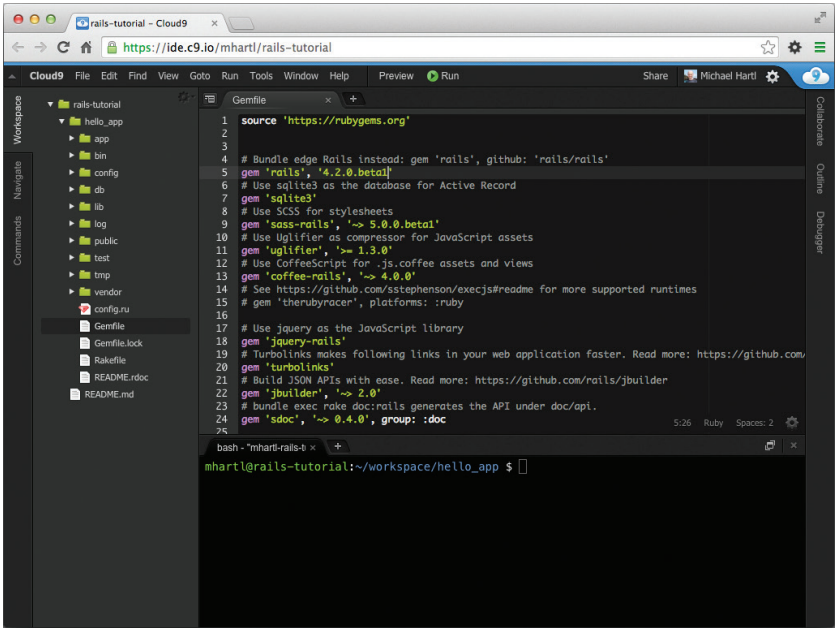

Рис. 1.5 ❖ Начальный Gemfile в текстовом редакторе

Многие строки в файле закомментированы символом #; их цель – показать некоторые часто используемые гемы, а также примеры синтаксиса компоновщика. Сейчас нам не понадобятся никакие новые гемы, кроме используемых по умолчанию.

Если не указать номер версии в команде `gem`, компоновщик установит самую последнюю, например:

```
gem 'sqlite3'
```

Есть два основных способа определения диапазона версий гема, позволяющих явно указать, что именно будет использовать Rails. Первый:

```
gem 'uglifier', '>= 1.3.0'
```

В этом случае будет установлена самая последняя версия гема `uglifier` (осуществляет сжатие файлов ресурсов) не ниже версии 1.3.0, даже если это будет версия 7.2. Второй метод:

```
gem 'coffee-rails', '~> 4.0.0'
```

Эта команда установит гем `coffee-rails` не ниже версии `4.0.0` и *не* выше `4.1`. Другими словами, запись `>=` всегда устанавливает самый последний гем, тогда как `~> 4.0.0` установит только незначительное обновление (например, с `4.0.0` до `4.0.1`), но не позволит установить значительное обновление (например, с `4.0` до `4.1`). К сожалению, практика показывает, что даже незначительные обновления гемов могут нарушить совместимость, поэтому в данной книге мы будем действовать осторожно, явно прописывая конкретные версии для всех гемов. Вы можете использовать самую последнюю версию любого гема, использовав конструкцию `~>` в файле `Gemfile` (однако я рекомендую этот шаг только опытным пользователям), но имейте в виду, что это может привести к совершенно непредсказуемому поведению примеров из этой книги.

Прописав в `Gemfile` из листинга 1.4 точные версии гемов, мы получим файл, показанный в листинге 1.5. Обратите внимание, что здесь гем `sqlite3` подключается только к окружению разработки и тестирования (раздел 7.1.1) – это предотвращает потенциальные конфликты с базой данных, которую использует фреймворк Heroku (раздел 1.5).

Листинг 1.5 ❖ Gemfile с явно указанными номерами версий гемов Ruby

```
source 'https://rubygems.org'

gem 'rails',                '4.2.0'
gem 'sass-rails',           '5.0.1'
gem 'uglifier',             '2.5.3'
gem 'coffee-rails',         '4.1.0'
gem 'jquery-rails',         '4.0.3'
gem 'turbolinks',           '2.3.0'
gem 'jbuilder',             '2.2.3'
gem 'sdoc',                 '0.4.0', group: :doc

group :development, :test do
  gem 'sqlite3',     '1.3.9'
  gem 'byebug',      '3.4.0'
  gem 'web-console', '2.0.0.beta3'
  gem 'spring',      '1.1.3'
end
```

Скорректировав содержимое Gemfile-приложения, как показано в листинге 1.5, установите геммы командой `bundle install`[1]:

[1] Как отмечено в табл. 3.1, слово `install` можно опустить, так как команда `bundle` – это псевдоним для `bundle install`.

```
$ cd hello_app/
$ bundle install
Fetching source index for https://rubygems.org/
.
.
.
```

Команде `bundle install` может потребоваться некоторое время на выполнение, но по ее завершении приложение будет готово к работе.

1.3.2. Сервер Rails

Выполнив команды `rails new` в разделе 1.3 и `bundle install` в разделе 1.3.1, мы получили приложение, готовое к запуску, но как его запустить? Фреймворк Rails поставляется с программой командной строки, или сценарием, который запускает локальный веб-сервер, помогающий в разработке приложений. Точная команда зависит от используемого окружения: на локальной машине можно просто выполнить `rails server` (листинг 1.6), но в Cloud9 требуется дополнительно указать IP-адрес для приема соединений и номер порта, чтобы сервер Rails знал, какой адрес использовать, чтобы сделать приложение видимым для внешнего мира (листинг 1.7)[1]. (В Cloud9 для этого используются специальные переменные окружения `$IP` и `$PORT`. Узнать значения этих переменных можно командой `echo $IP` или `echo $PORT`.)

Листинг 1.6 ❖ Запуск Rails-сервера на локальной машине

```
$ cd ~/workspace/hello_app/
$ rails server
=> Booting WEBrick
=> Rails application starting on http://localhost:3000
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
```

Листинг 1.7. Запуск Rails-сервера в cloudIDE

```
$ cd ~/workspace/hello_app/
$ rails server -b $IP -p $PORT
=> Booting WEBrick
=> Rails application starting on http://0.0.0.0:8080
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
```

Независимо от варианта я рекомендую запускать команду `rails server` во второй вкладке терминала, чтобы в первой по-прежнему можно было выполнять команды, как показано на рис. 1.6 и 1.7. (Если вы уже запустили сервер в первой вкладке, нажмите **Ctrl-C**, чтобы остановить его[2].) Если вы используете локальный

---

[1] Обычно веб-сайты запускаются на 80-м порте, но для этого часто требуются специальные привилегии, поэтому для сервера разработки нередко используют порты с более высокими номерами.

[2] «C» обозначает клавишу на клавиатуре, а не заглавную букву, поэтому не нужно нажимать еще и **Shift**.

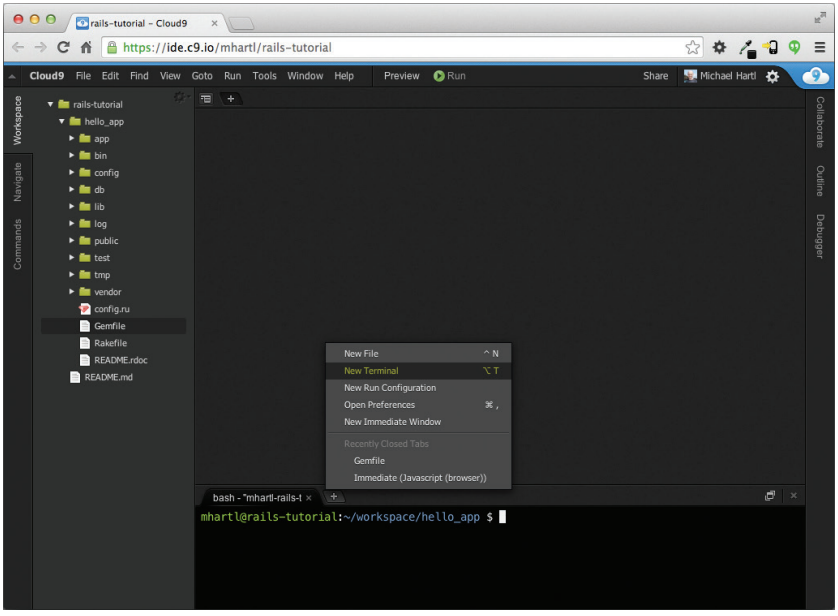

Рис. 1.6 ❖ Открытие новой вкладки терминала

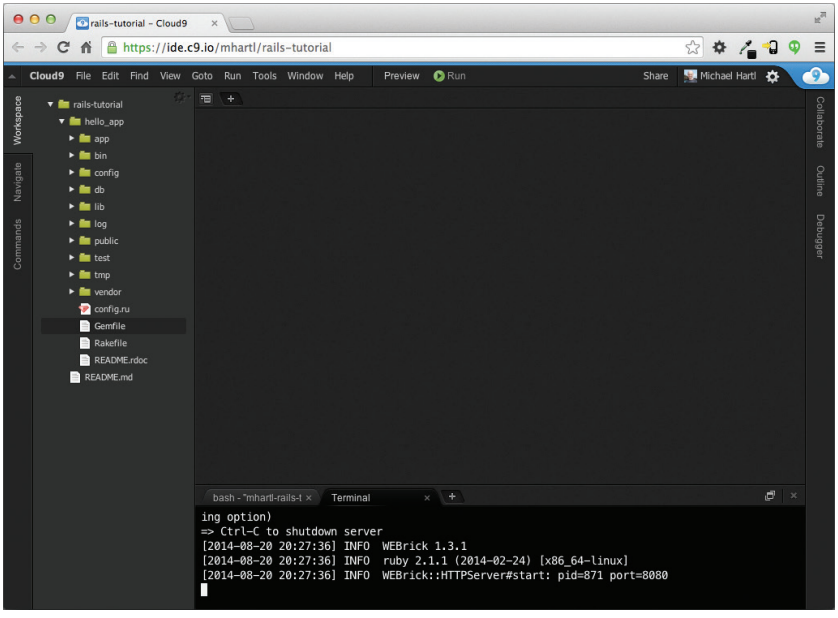

Рис. 1.7 ❖ Запуск Rails-сервера в отдельной вкладке

сервер, наберите в браузере адрес http://localhost:3000/; в облачной IDE щелкните на ссылке **Share** (Пригласить) и затем в открывшемся диалоге щелкните в поле **Application address** (Адрес приложения, см. рис. 1.8). В любом случае, результат должен быть похожим на рис. 1.9.

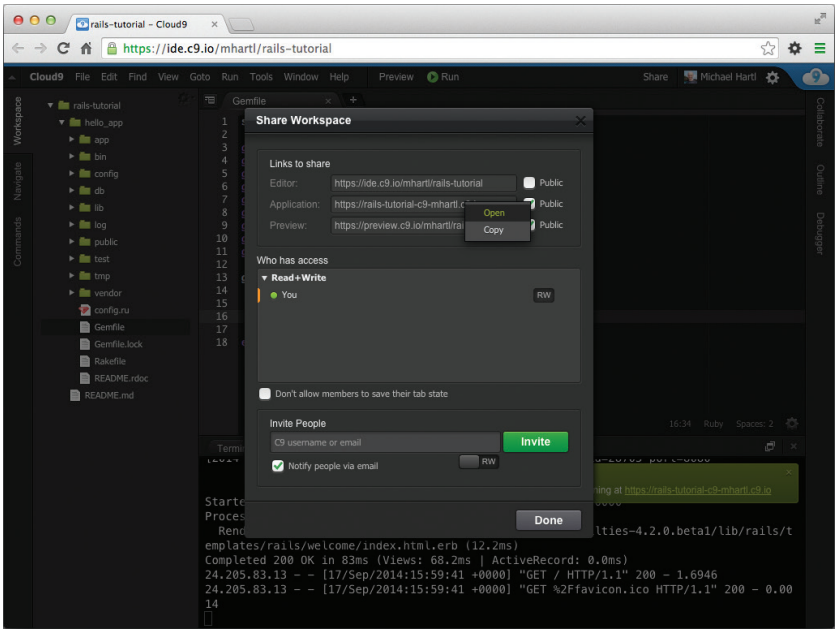

Рис. 1.8 ❖ Вызов страницы приложения в облачном рабочем пространстве

Чтобы увидеть информацию о нашем первом приложении, щелкните на ссылке **About your application's environment** (Об окружении вашего приложения). Хотя конкретные номера версий могут отличаться, в целом результат должен напоминать изображение на рис. 1.10. Очевидно, что в дальнейшем начальная страница Rails нам не понадобится, но сейчас приятно видеть ее работающей. Мы удалим ее (и заменим ее своей) в разделе 1.3.4.

1.3.3. Модель-представление-контроллер (MVC)

Даже на этой ранней стадии полезно получить общее представление о том, как работают Rails-приложения (рис. 1.11). Возможно, вы заметили в стандартной структуре Rails-приложения (рис. 1.4) каталог `app/` с тремя подкаталогами: `models`, `views` и `controllers`. Это намек, что Rails следует архитектурной схеме *модель-представление-контроллер* (Model-Miew-Controller, MVC), которая отделяет «логику предметной области» (или «бизнес-логику») от логики ввода и логики представления, связанной с графическим интерфейсом пользователя (GUI). В случае

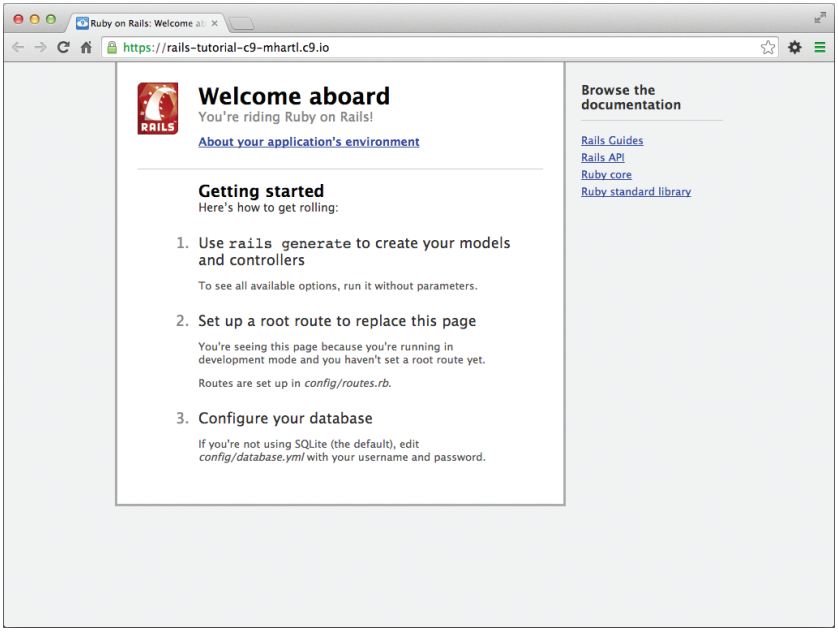

Рис. 1.9 ❖ Начальная Rails-страница по умолчанию

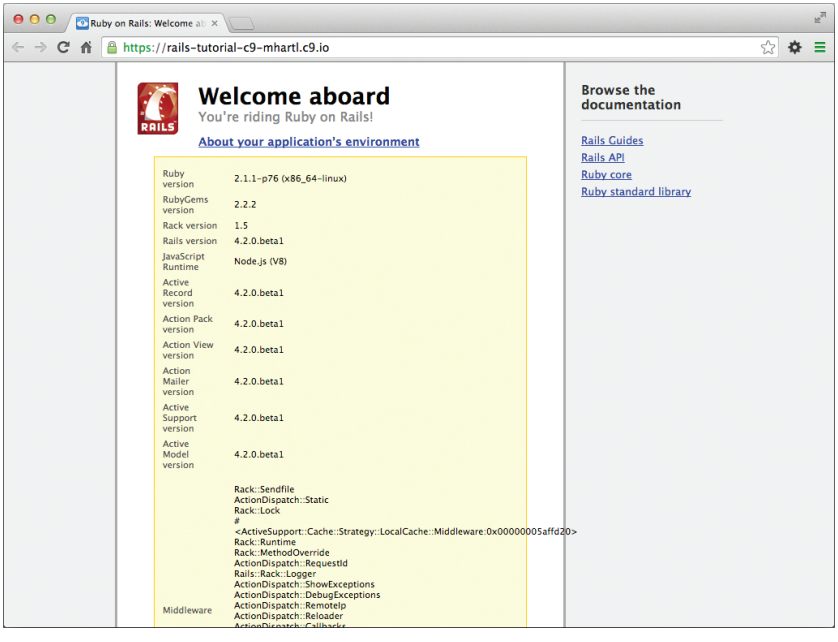

Рис. 1.10 ❖ Начальная страница с информацией об окружении приложения

веб-приложений «логика предметной области» обычно состоит из модели данных, представляющих пользователей, статьи, продукты, а GUI – это просто веб-страница в браузере.

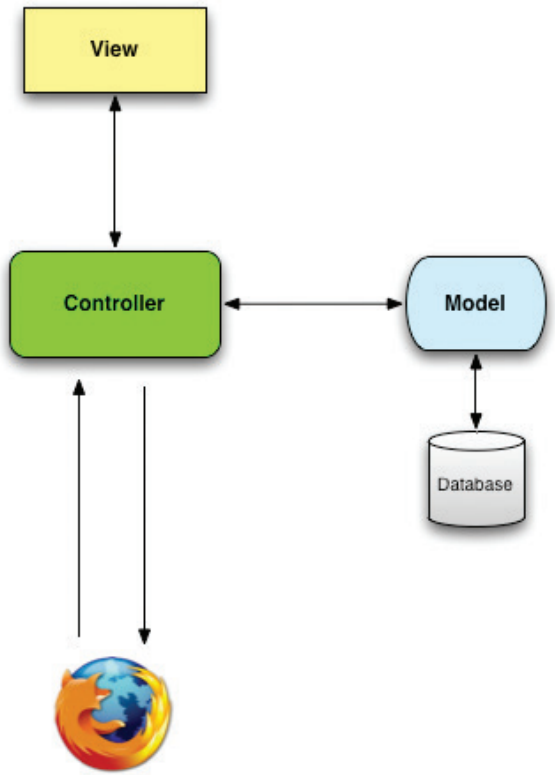

Рис. 1.11 ❖ Схематичное изображение архитектуры модель-представление-контроллер (MVC)

Взаимодействуя с приложением Rails, браузер посылает запрос, веб-сервер принимает его и передает контроллеру Rails, отвечающему за дальнейшую обработку. Иногда контроллер сразу отображает представление – шаблон, который преобразуется в разметку HTML и возвращается браузеру. В динамических сайтах гораздо чаще контроллер взаимодействует с моделью – объектом Ruby, который представляет элемент сайта (например, пользователя) и отвечает за связь с базой данных. После вызова модели контроллер отображает представление и возвращает браузеру готовую веб-страницу с разметкой HTML.

Если это обсуждение сейчас кажется вам немного абстрактным, не беспокойтесь; мы еще не раз будем возвращаться к этому разделу. В разделе 1.3.4 мы сделаем первую попытку применить MVC, но уже в разделе 2.2.2 обсудим MVC более детально в контексте мини-приложения. Наконец, в процессе создания учебного

приложения будут использоваться все аспекты MVC; мы охватим контроллеры и представления в разделе 3.2, модели – в разделе 6.1 и увидим совместную работу всей этой троицы в разделе 7.1.2.

1.3.4. Hello, world!

В качестве первого применения структуры MVC внесем тончайшие изменения в первое приложение, добавив метод контроллера, возвращающего строку «hello, world!». (Больше о методах действий контроллеров рассказывается в разделе 2.2.2.) В результате мы заменим начальную страницу Rails, изображенную на рис. 1.9, страницей «hello, world» – это и есть цель данного раздела.

Как можно догадаться, методы действий контроллеров определяются внутри контроллеров. Наш первый метод мы назовем `hello` и поместим в контроллер `ApplicationController` (контроллер приложения). В действительности сейчас это единственный контроллер, который есть в наличии, и в этом можно убедиться, выполнив команду:

```
$ ls app/controllers/*_controller.rb
```

(Мы начнем создавать собственные контроллеры в главе 2.) В листинге 1.8 показано определение метода hello, который использует функцию render, чтобы вернуть текст «hello, world!». (Не беспокойтесь сейчас о синтаксисе языка Ruby, мы разберемся с ним в главе 4.)

Листинг 1.8 ❖ Добавление метода действия hello в контроллер приложения (app/controllers/application_controller.rb)

```
class ApplicationController < ActionController::Base
  # Предотвратить атаки вида "подделка межсайтовых запросов" (CSRF),
  # возбудив исключение. Для библиотек, возможно, предпочтительнее будет
  # использовать :null_session instead.
  protect_from_forgery with: :exception

  def hello
    render text: "hello, world!"
  end
end
```

Определив метод, возвращающий нужную строку, необходимо сказать Rails, что именно он должен использоваться вместо вывода начальной страницы, изображенной на рис. 1.9. Для этого настроим *маршрутизатор* Rails, находящийся перед контроллером на рис. 1.11, и определим, куда посылать запросы, поступающие от браузера. (Для простоты я опустил маршрутизатор на рис. 1.11, но мы еще вернемся к вопросу маршрутизации в разделе 2.2.2.) В частности, нужно изменить начальную страницу, *корневой маршрут*, именно он определяет страницу, возвращаемую в ответ на обращение к *корневому URL*. Так как корневые URL совпадают с такими адресами, как http://www.example.com/ (когда за слешем ничего не следует), их часто для краткости называют / («слеш»).

Как видно в листинге 1.9, файл маршрутов (config/routes.rb) содержит закомментированную строку, в которой описывается, как построить корневой маршрут. Здесь «welcome» – это имя контроллера,«index» – метод внутри этого контроллера. Чтобы активировать корневой маршрут, раскомментируйте строку, удалив символ решетки, и замените кодом из листинга 1.10, который требует от Rails при обращении к корневому маршруту вызвать метод hello контроллера ApplicationController. (Как было отмечено в разделе 1.1.2, вертикальные точки обозначают пропущенный код, их не нужно копировать буквально.)

Листинг 1.9 ❖ Начальный (закомментированный) корневой маршрут (config/routes.rb)

```
Rails.application.routes.draw do
  .
  .
  .
  # Определить корневой маршрут к сайту можно как "root"
  # root 'welcome#index'
  .
  .
  .
end
```

Листинг 1.10 ❖ Установка корневого маршрута (config/routes.rb)

```
Rails.application.routes.draw do
  .
  .
  .
  # Определить корневой маршрут к сайту можно как "root"
  root 'application#hello'
  .
  .
  .
end
```

После изменений, показанных в листингах 1.8 и 1.10, корневой маршрут будет возвращать «hello, world!», как и требовалось (рис. 1.12).

1.4. Управление версиями с Git

Теперь, после создания нового и работающего приложения, воспользуемся моментом и сделаем то, что видится многим разработчикам Rails как практически необходимый шаг (хотя технически это необязательно), – поместим исходный код приложения в *систему управления версиями*. Системы управления версиями позволяют отслеживать изменения в коде проекта, облегчают совместную работу, а также могут откатывать любые непреднамеренные погрешности (такие как случайное удаление файлов). Грамотное использование системы управления вер-

сиями – необходимый навык для каждого профессионального разработчика программного обеспечения.

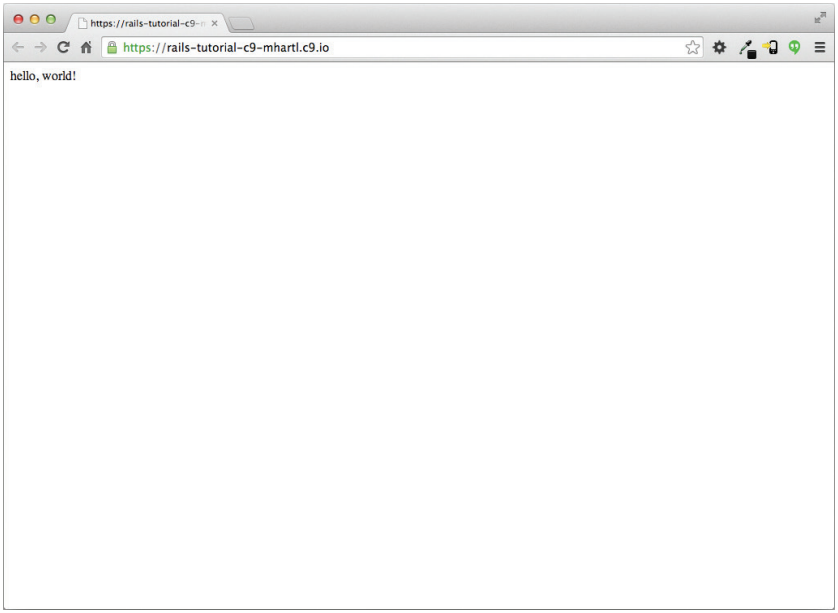

Рис. 1.12 ❖ Отображение «hello, world!» в браузере

Существует множество инструментов управления версиями, но сообщество Rails в значительной степени ориентировано на Git[1], распределенную систему управления версиями, первоначально разработанную Линусом Торвальдсом (Linus Torvalds) для хранения исходных текстов ядра Linux. Git – обширная тема, и мы лишь слегка коснемся ее в этой книге, но в Интернете можно найти много хороших бесплатных ресурсов; я особенно рекомендую краткий обзор «Bitbucket 101»[2] и книгу «Pro Git»[3] Скотта Чакона (Scott Chacon) для более подробного изучения. Помещение исходного кода в систему управления версиями Git *строго* рекомендуется не только потому, что это почти повсеместная практика в мире Rails, но также и потому, что это позволит легко создать резервную копию или открыть доступ к вашему коду (раздел 1.4.3), а еще развернуть приложение прямо здесь, в первой главе (раздел 1.5).

---

1 https://git-scm.com/.

2 https://confluence.atlassian.com/display/BITBUCKET/Clone+your+Git+repository+and+add+source+files.

3 http://git-scm.com/book/ru/v1.

1.4.1. Установка и настройка

Облачная IDE, рекомендованная в разделе 1.2.1, по умолчанию уже включает поддержку Git, то есть в этом случае нет необходимости устанавливать ее. В противном случае обращайтесь на сайт InstallRails.com (раздел 1.2), где приводятся инструкции по установке Git.

Первоначальная настройка системы

Прежде чем использовать Git, следует выполнить ряд первоначальных настроек. Это *системные* настройки, а значит, их необходимо сделать лишь единожды:

```
$ git config --global user.name "Your Name"
$ git config --global user.email your.email@example.com
$ git config --global push.default matching
$ git config --global alias.co checkout
```

Имейте в виду, что ваши имя и адрес электронной почты, указанные в настройках Git, будут доступны в любом вашем репозитории, который вы сделаете общедоступным. (Строго говоря, необходимы только первые две строки. Третья строка обеспечивает совместимость с будущими версиями Git. Четвертая – позволяет использовать команду `co` вместо более длинной `checkout`. Для максимальной совместимости с системами, где команда `co` не настроена, в учебнике будет использоваться полная команда `checkout`, но в жизни я почти всегда пишу `git co`.)

Первоначальная настройка репозитория

Сейчас мы разберем этапы создания каждого нового *репозитория* (иногда его называют *репо* для краткости). Сначала перейдите в корневой каталог первого приложения и инициализируйте новый репозиторий:

```
$ git init
Initialized empty Git repository in /home/ubuntu/workspace/hello_app/.git/
```

Затем добавьте все файлы проекта в репозиторий командой `git add -A`:

```
$ git add -A
```

Эта команда добавит все файлы из текущего каталога, кроме тех, что соответствуют шаблонам (правилам) в специальном файле `.gitignore`. Команда `rails new` автоматически создает файл `.gitignore`, соответствующий Rails-проекту, но в него также можно добавить дополнительные правила[1].

Добавленные файлы сначала помещаются в *промежуточную область*, где находятся незавершенные изменения в проекте. Посмотреть, какие файлы находятся там, можно с помощью команды `status`:

---

[1] В этом учебнике нам не понадобится редактировать его, тем не менее пример добавления правила в файл `.gitignore` будет показан в разделе 3.7.3 как часть дополнительных настроек тестирования.

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

  new file:    .gitignore
  new file:    Gemfile
  new file:    Gemfile.lock
  new file:    README.rdoc
  new file:    Rakefile
  .
  .
  .
```

(Результат очень длинный, поэтому здесь вместо большей части текста показаны вертикальные точки.)

Чтобы сообщить Git о необходимости сохранения изменений, используйте команду `commit`:

```
$ git commit -m "Initialize repository"
[master (root-commit) df0a62f] Initialize repository
.
.
.
```

Параметр `-m` позволяет добавить сообщение для фиксации; если опустить его, Git откроет редактор, установленный в системе по умолчанию, и предложит ввести сообщение в нем. (Во всех примерах в книге будет использован флаг `-m`.)

Важно отметить, что фиксации Git *локальны*, они записываются только на машине, на которой сделаны. В разделе 1.4.4 мы разберем, как отправить изменения в удаленный репозиторий (командой `git push`).

Кстати, список фиксаций можно вызвать командой `log`:

```
$ git log
commit df0a62f3f091e53ffa799309b3e32c27b0b38eb4
Author: Michael Hartl <michael@michaelhartl.com>
Date:   WedAugust 20 19:44:43 2014 +0000

    Initializerepository
```

В зависимости от длины истории репозитория может понадобиться ввести `q` для выхода.

1.4.2. Что дает использование репозитория Git?

Если раньше вы никогда не использовали систему управления версиями, вам может быть не очень понятно, зачем это нужно, поэтому позвольте мне привести всего один пример. Предположим, вы произвели некоторые случайные изменения, например (О, нет!) удалили крайне необходимый каталог `app/controllers/`.

```
$ ls app/controllers/
application_controller.rb concerns/
$ rm -rf app/controllers/
$ ls app/controllers/
ls: app/controllers/: No such file or directory
```

Здесь для вывода содержимого каталога `app/controllers/` использована Unix-команда `ls`, а для его удаления – команда `rm` (табл. 1.1). Параметр `-rf` означает «recursive force» («рекурсивно и принудительно») и обеспечивает рекурсивное удаление всех файлов, каталогов, подкаталогов и т. д., без дополнительного подтверждения.

Давайте проверим статус и посмотрим, что изменилось:

```
$ git status
On branch master
Changed but not updated:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

      deleted:    app/controllers/application_controller.rb

no changes added to commit (use "git add" and/or "git commit -a")
```

Как видите, файл был удален, но изменилось только «рабочее дерево»; изменения еще не зафиксированы. Это означает, что мы можем легко отменить изменения командой `checkout` с параметром `-f` для принудительной отмены текущих изменений:

```
$ git checkout -f
$ git status
# On branch master
nothing to commit (working directory clean)
$ ls app/controllers/
application_controller.rb concerns/
```

Пропавшие каталоги файла вернулись. Какое облегчение!

1.4.3. Bitbucket

Теперь, когда проект помещен в систему управления версиями Git, пришло время отправить его на сайт Bitbucket[1], оптимизированный для хостинга и совместного использования Git-репозиториев. (В предыдущем издании учебника вместо него использовался GitHub[2]; причины замены разъясняются в блоке 1.4.) Отправка копии Git-репозитория на сайт Bitbucket служит двум целям: полному резервному копированию программного кода (включая полную историю фиксаций) и простоте совместной разработки в будущем.

---

1 https://bitbucket.org/.

2 https://github.com/.

Блок 1.4 ❖ GitHub и Bitbucket

GitHub и Bitbucket – наиболее популярные сайты для хостинга Git-репозиториев. Они во многом похожи: оба позволяют размещать репозитории Git и использовать их для коллективной разработки, а также предоставляют удобные средства просмотра и поиска в репозиториях. Самое главное отличие (с точки зрения этого учебника) в том, что GitHub бесплатно предлагает неограниченное количество репозиториев с открытым исходным кодом (с возможностью совместной работы), но берет плату за закрытые (частные) репозитории, в то время как Bitbucket позволяет бесплатно завести неограниченное количество закрытых репозиториев, но при этом ограничивает число совместно работающих людей (увеличить его можно за плату). Таким образом, выбор службы для конкретного репозитория зависит от ваших потребностей.
В предыдущем издании этого учебника использовался GitHub из-за акцента на поддержку открытого исходного кода, но все возрастающая обеспокоенность по поводу безопасности привела меня к тому, что я рекомендую *все* репозитории веб-приложений делать закрытыми по умолчанию. Дело в том, что репозитории для веб-приложений могут содержать конфиденциальную информацию, такую как криптографические ключи или пароли, которая может быть использована для компрометации сайтов, выполняющих этот код. Конечно, можно организовать безопасную обработку информации (путем настроек игнорирования Git, например), но эти методы сложны и требуют значительного опыта.
На самом деле учебное приложение из этого учебника вполне безопасно, чтобы выложить его в сеть, но не стоит всегда полагаться на этот факт. То есть для максимальной безопасности мы будем действовать с предельной осмотрительностью и по умолчанию использовать закрытые репозитории. Поскольку GitHub берет плату за закрытость, а Bitbucket предлагает неограниченное число закрытых репозиториев бесплатно, для наших целей Bitbucket подходит лучше, чем GitHub.

Начать работу с Bitbucket очень просто:

1. Создайте учетную запись на Bitbucket[1], если у вас ее еще нет.
2. Скопируйте свой *открытый ключ*[2] в буфер обмена. Пользователи облачной IDE могут увидеть его, выполнив команду `cat`, как показано в листинге 1.11, после чего его можно выделить и скопировать. Если вы используете другую систему и запуск команды из листинга 1.11 не дает никакого результата, тогда следуйте инструкции, описывающей установку открытого ключа в учетную запись Bitbucket[3].
3. Добавьте свой открытый ключ в Bitbucket, щелкнув на аватаре в правом верхнем углу и выбрав **Manage account** (Управление учетной записью), а затем **SSH keys** (SSH-ключи, см. рис. 1.13).

Листинг 1.11 ❖ Вывод открытого ключа командой cat

```
$ cat ~/.ssh/id_rsa.pub
```

[1] https://bitbucket.org/account/signup/.

[2] https://ru.wikipedia.org/wiki/Криптосистема_с_открытым_ключом.

[3] Инструкция на русском языке: https://bitbucket.org/rtnm/gittertutorial/src.

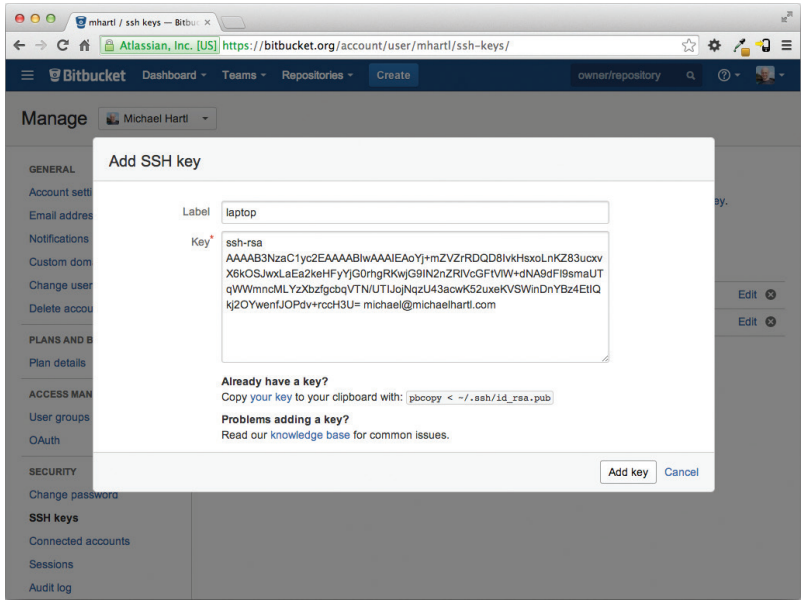

Рис. 1.13 ❖ Добавление открытого SSH-ключа

После добавления ключа щелкните на ссылке **Create** (Создать), чтобы создать новый репозиторий, как показано на рис. 1.14. Заполнив информацию о проекте, не забудьте установить флажок **This is a private repository** (Закрытый репозиторий). После щелчка на ссылке **Create repository** (Создать репозиторий) следуйте инструкциям в разделе **Command line ⇨ I have an existing project** (У меня уже есть проект), они должны выглядеть примерно как листинг 1.12. (Если они выглядят иначе, скорее всего, открытый ключ не был добавлен, и стоит попробовать добавить его еще раз.) При отправке репозитория ответьте **yes** на вопрос **Are you sure you want to continue connecting (yes/no)?** (Вы уверены, что хотите продолжить подключение?).

Листинг 1.12 ❖ Добавление Bitbucket и отправка репозитория

```
$ git remote add origin git@bitbucket.org:<username>/hello_app.git
$ git push -u origin --all        # отправляет репозиторий в первый раз
```

Команды в листинге 1.12 сообщают Git, что вы хотите добавить Bitbucket в качестве *источника* (origin) для вашего репозитория, а затем отправляют репозиторий в удаленный источник. (Не беспокойтесь о значении флага `-u`; если вам любопытно, поищите в сети по фразе: «git set upstream».) Конечно, следует заменить `<username>` фактическим именем пользователя. Например, команда, которую запустил я:

```
$ git remote add origin git@bitbucket.org:mhartl/hello_app.git
```

В результате действий, описанных выше, на сайте Bitbucket появится страница для репозитория `hello_app` с браузером файлов, полной историей фиксаций и большим количеством прочих приятных мелочей (рис. 1.15).

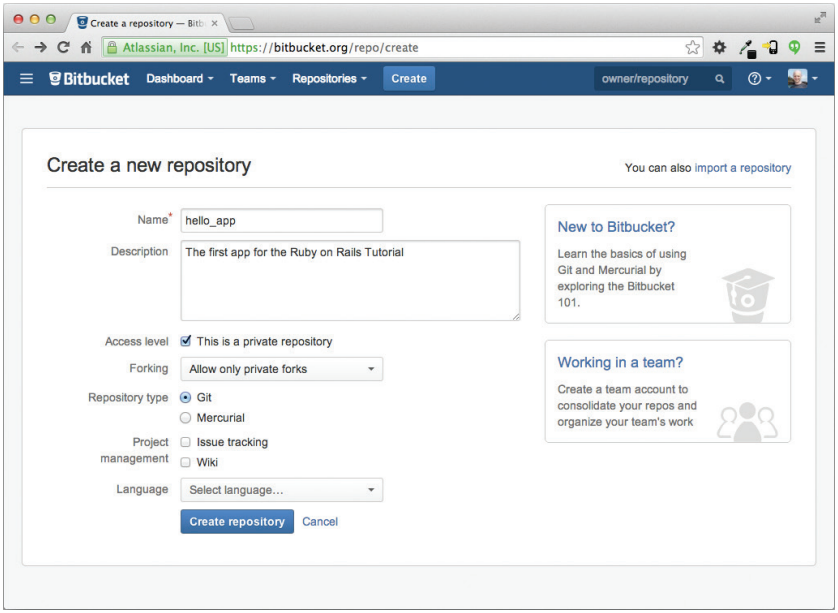

Рис. 1.14 ❖ Создание первого репозитория на сайте Bitbucket

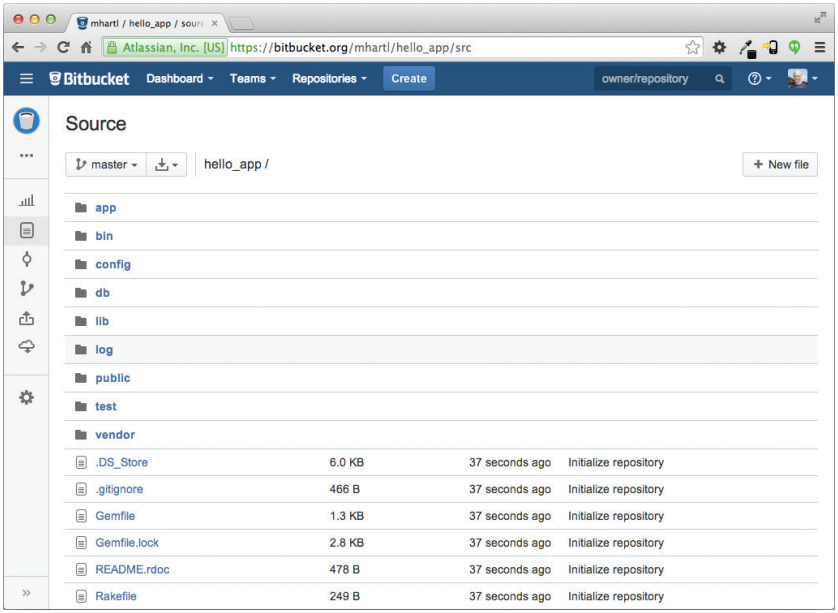

Рис. 1.15 ❖ Страница репозитория на сайте Bitbucket

1.4.4. Ветвление, редактирование, фиксация, слияние

Прошедшие все этапы в разделе 1.4.3 могли заметить, что Bitbucket не создает автоматически файл `README.rdoc` в репозитории, вместо этого он выводит на главной странице сообщение о его отсутствии (рис. 1.16). Это указывает на недостаточно широкую распространенность формата `rdoc`, чтобы Bitbucket поддерживал его автоматически, и на самом деле я и практически все известные мне разработчики предпочитают вместо него использовать формат *Markdown*. В этом разделе мы заменим файл `README.rdoc` на `README.md`, а также добавим в него описание своего проекта. В процессе мы увидим первый пример ветвления, редактирования, фиксации и слияния – именно такой рабочий процесс я рекомендую использовать в работе с Git[1].

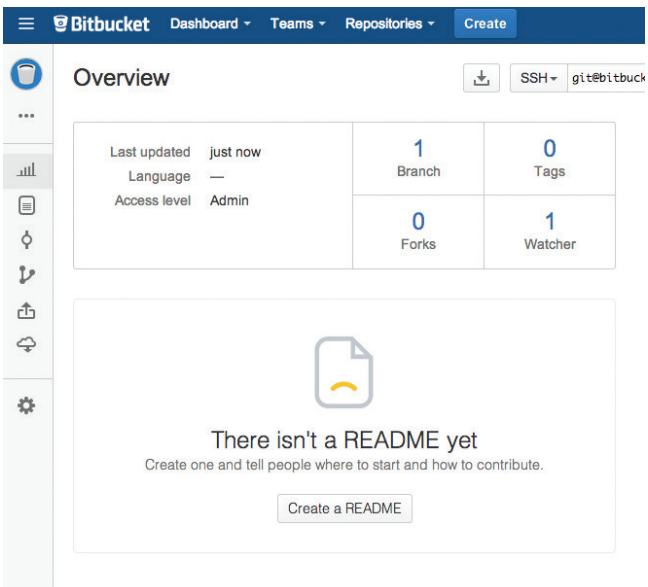

Рис. 1.16 ❖ Сообщение Bitbucket об отсутствии файла README

Ветвление

Git невероятно хорош в создании *ветвей*, которые фактически являются копиями репозитория, где можно производить изменения (возможно, экспериментальные), не модифицируя родительских файлов. В большинстве случаев родительский репозиторий – это *главная ветвь* (master), но можно создать новую рабочую, или тематическую, ветвь, выполнив команду `checkout` с флагом `-b`:

[1] Для удобства работы с Git-репозиториями я рекомендую использовать приложение Source Tree от Atlassian (описание на русском языке можно найти по адресу: http://www.teamlead.ru/pages/viewpage.action?pageId=112263556).

```
$ git checkout -b modify-README
Switched to a new branch 'modify-README'
$ gitbranch
  master
* modify-README
```

Вторая команда, `git branch`, просто перечисляет локальные ветви, а звездочкой * отмечена текущая ветвь. Обратите внимание, `git checkout -b modify-README` одновременно создает новую ветвь и делает ее текущей, на что указывает звездочка перед `modify-README`. (Если вы настроили команду `co` в разделе 1.4, вместо этого можете писать `git co -b modify-README`.)

Полностью оценить достоинства ветвления можно, только работая над проектом в составе коллектива[1], но ветви полезны даже для единственного разработчика, как в нашем случае. В частности, главная ветвь (master) изолируется от любых изменений в рабочих ветвях, поэтому, даже если мы *действительно* наворотим лишнего, всегда можно отказаться от изменений, вернувшись в главную ветвь и удалив рабочую. Мы увидим, как это делается, в конце раздела.

Между прочим, для столь малых изменений я бы не стал озадачиваться новой ветвью, но никогда не бывает слишком рано, чтобы начать практиковать хорошие привычки.

Редактирование

После создания локальной ветви отредактируем файл с описанием. Я предпочитаю использовать *язык разметки Markdown*[2] для этих целей, и если вы используете расширение файла `.md`, Bitbucket будет автоматически форматировать его. Итак, сначала запустим Git-версию Unix-команды `mv`, чтобы изменить имя файла:

```
$ git mv README.rdoc README.md
```

Затем заполним `README.md` содержимым из листинга 1.13.

Листинг 1.13. Новый файл README, README.md

```
# Учебник Ruby on Rails: "hello, world!"
Это первое приложение для
[*RubyonRailsTutorial*](http://www.railstutorial.org/)
[Майкл Хартл](http://www.michaelhartl.com/).
```

Фиксация

После внесения изменений можно взглянуть на статус ветки:

```
$ git status
On branch modify-README
Changes to be committed:
```

---

[1] Более подробно в главе «Ветвление в Git» – в книге «Pro Git» (https://git-scm.com/book/ru/v1/Ветвление_в_Git).

[2] https://ru.wikipedia.org/wiki/Markdown.

```
  (use "git reset HEAD <file>..." to unstage)

  renamed: README.rdoc -> README.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

modified: README.md
```

Сейчас мы могли бы использовать `git add -A`, как в разделе 1.4.1, но `git commit` предусматривает флаг `-a` для (очень частого) случая фиксации всех изменений существующих файлов (или файлов, созданных с использованием `git mv`, которые не считаются новыми для Git):

```
$ git commit -a -m "Improve the README file"
2 files changed, 5 insertions(+), 243 deletions(-)
delete mode 100644 README.rdoc
create mode 100644 README.md
```

Будьте осторожны, используя флаг `-a`; добавив новые файлы в проект после последней фиксации, вы должны сообщить Git о них, выполнив сначала `git add -A`.

Обратите внимание, что мы написали сообщение в настоящем времени (и, технически говоря, повелительном наклонении). Git моделирует фиксации как серии правок существующего кода, и в этом контексте имеет смысл описать, что каждая фиксация делает, а не что делала. Кроме того, такое использование соответствует сообщениям о фиксациях, генерируемым самой командой Git. Более подробно об этом можно почитать в статье «Блестящий новый стиль фиксаций»[1].

Слияние

Теперь, закончив с изменениями, надо слить рабочую ветвь с главной:

```
$ git checkout master
Switched to branch 'master'
$ git merge modify-README
Updating 34f06b7..2c92bef
Fast forward
README.rdoc    |  243 ----------------------------------------------
README.md      |    5 +
2 files changed, 5 insertions(+), 243 deletions(-)
delete mode 100644 README.rdoc
create mode 100644 README.md
```

Вывод Git часто включает такие строки, как `34f06b7`, которые связаны с внутренним представлением репозиториев Git. Ваши конкретные результаты будут отличаться в этих деталях, но в остальном они должны соответствовать выводу, показанному выше.

---

[1] https://github.com/blog/926-shiny-new-commit-styles.

После слияния изменений можно почистить ветви, удалив рабочую ветвь командой `git branch -d`, если она больше не нужна:

```
$ git branch -d modify-README
Deleted branch modify-README (was 2c92bef).
```

Этот шаг не является обязательным, и на практике рабочие ветви часто оставляют нетронутыми. Это позволяет переключаться между рабочей и главной ветвями, сливая их всякий раз, когда достигается естественный конечный пункт.

Как упоминалось выше, можно вообще отказаться от изменений в рабочей ветви командой `git branch -D`:

```
# Исключительно для иллюстрации; пользуйтесь,
# только если совсем все испортили в рабочей ветке
$ gitcheckout -btopic-branch
$ <действительно все испортили>
$ git add -A
$ git commit -a -m "Major screw up"
$ git checkout master
$ git branch -D topic-branch
```

В отличие от флага `-d`, флаг `-D` удалит ветку, даже если не было выполнено слияние.

Отправка

После обновления `README` можно отправить изменения в Bitbucket, чтобы увидеть результат. Так как одна отправка уже была сделана (раздел 1.4.3), в большинстве систем после этого можно опустить `origin master` и просто выполнить `git push`:

```
$ git push
```

Как было обещано в разделе 1.4.4, Bitbucket отформатировал новый файл с разметкой Markdown (рис. 1.17).

1.5. Развертывание

На этой ранней стадии мы уже готовы развернуть наше (все еще пустое) Rails-приложение. Этот шаг не является обязательным, но раннее и частое развертывание позволяет раньше обнаруживать проблемы в цикле разработки. Альтернативный вариант – развертывание только после напряженных усилий, в изолированном окружении разработки – часто приводит к неприятностям, когда наступает время запуска[1].

---

[1] Вообще, это не должно иметь значения для учебных приложений из этой книги, но если вы волнуетесь, что случайно и/или преждевременно опубликуете свое приложение, есть несколько способов, которые могут помочь избежать этого; один из них описан в разделе 1.5.4.

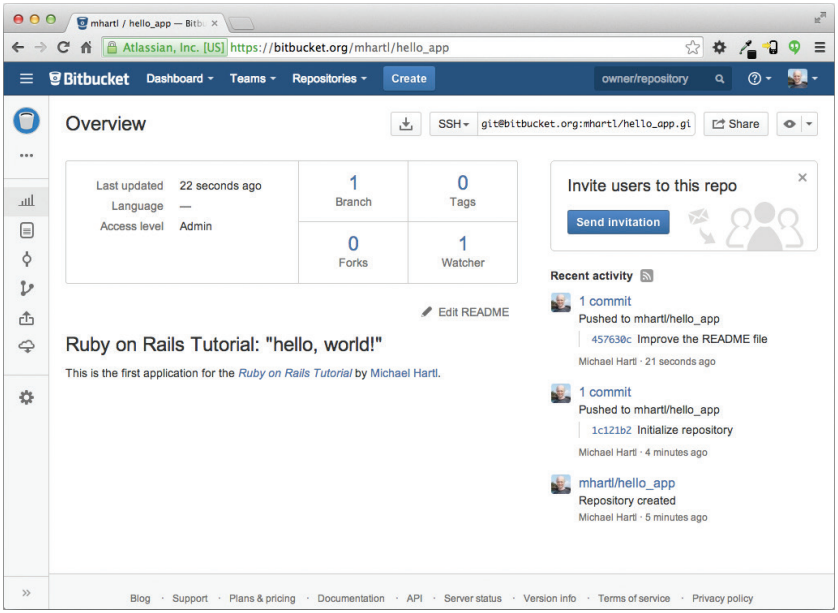

Рис. 1.17 ❖ Улучшенный файл README с разметкой Markdown

Раньше развертывание Rails-приложений было довольно сложной процедурой, но экосистема развертывания в Rails стремительно развивалась в течение нескольких последних лет, и теперь в ней есть несколько замечательных инструментов. Среди них общедоступные или виртуальные частные серверы на основе Phusion Passenger[1] (модуль для веб-серверов Apache и Nginx[2]), компании, предоставляющие полный комплекс услуг развертывания, такие как Engine Yard[3] и Rails Machine[4], и облачные службы развертывания, такие как Engine Yard Cloud[5], Ninefold[6] и Heroku[7].

Я отдаю предпочтение Heroku, хостинговой платформе, созданной специально для развертывания веб-приложений на Rails и других платформах. Heroku делает развертывание Rails-приложений смехотворно простым, если их исходный код хранится в системе управления версиями Git. (Это еще одна причина выполнить шаги установки Git из раздела 1.4, если вы до сих пор этого не сделали.) Кроме того, бесплатной службы Heroku более чем достаточно для выполнения очень

---

1 https://www.phusionpassenger.com/.

2 Произносится как «ин-джин-икс».

3 https://engineyard.com/.

4 https://railsmachine.com/.

5 https://cloud.engineyard.com/.

6 https://ninefold.com/.

7 https://www.heroku.com/.

многих задач, в том числе и для нужд этого учебника. Я не заплатил ни цента за хостинг на платформе Heroku первых двух изданий, обработавшей для меня несколько миллионов запросов. Остальная часть раздела посвящена развертыванию нашего первого приложения на Heroku. Некоторые понятия довольно сложны, поэтому не переживайте, если не поймете всех деталей сразу; важно, чтобы в конце концов мы развернули приложение непосредственно в сети.

1.5.1. Установка Heroku

Платформа Heroku использует базу данных PostgreSQL[1] (произносится «пост-грескю-эль», для краткости ее часто называют «Postgres»), а это означает, что в окружение необходимо добавить гем `pg`, чтобы позволить Rails общаться с Postgres[2]:

```
group :production do
  gem 'pg',             '0.17.1'
  gem 'rails_12factor', '0.0.2'
end
```

Обратите внимание на дополнительный гем `rails_12factor`, который Heroku использует для работы со статическими ресурсами, такими как изображения и таблицы стилей.

Получившийся файл `Gemfile` показан в листинге 1.14.

Листинг 1.14 ❖ Gemfile с дополнительными гемами

```
source 'https://rubygems.org'

gem 'rails',        '4.2.0'
gem 'sass-rails',   '5.0.1'
gem 'uglifier',     '2.5.3'
gem 'coffee-rails', '4.1.0'
gem 'jquery-rails', '4.0.3'
gem 'turbolinks',   '2.3.0'
gem 'jbuilder',     '2.2.3'
gem 'sdoc',         '0.4.0', group: :doc

group :development, :test do
  gem 'sqlite3',     '1.3.9'
  gem 'byebug',      '3.4.0'
  gem 'web-console', '2.0.0.beta3'
  gem 'spring',      '1.1.3'
end

group :production do
  gem 'pg',             '0.17.1'
```

[1] https://ru.wikipedia.org/wiki/PostgreSQL.

[2] Вообще говоря, желательно, чтобы окружение разработки максимально соответствовало эксплуатационному окружению. Это касается и баз данных, но для целей данного учебника мы всегда будем использовать SQLite локально и PostgreSQL – для эксплуатации. Дополнительная информация приводится в разделе 3.1.

```
  gem 'rails_12factor', '0.0.2'
end
```

Чтобы подготовить систему к развертыванию в эксплуатационном окружении, нужно выполнить `bundle install` со специальным флагом, предотвращающим локальную установку любых гемов из раздела `production` (в данном случае `pg` и `rails_12factor`):

```
$ bundle install --without production
```

Так как в листинге 1.14 были добавлены только гемы в раздел `production`, прямо сейчас эта команда не установит никаких дополнительных локальных гемов, но она добавит в `Gemfile.lock` гемы `pg` и `rails_12factor`. Зафиксируем полученные изменения:

```
$ gitcommit -a -m "UpdateGemfile.lockforHeroku"
```

Далее необходимо создать и настроить новую учетную запись Heroku. Для начала следует зарегистрироваться на сайте Heroku[1]. Затем – проверить, установлен ли клиент, командной строкой Heroku:

```
$ heroku version
```

Пользующиеся облачной IDE должны увидеть номер версии Heroku, это говорит о доступности клиента `heroku`, на других системах может потребоваться его установка через Heroku Toolbelt[2].

После установки интерфейса командной строки Heroku нужно выполнить команду `heroku login`, чтобы войти в систему и добавить SSH-ключ:

```
$ heroku login
$ heroku keys:add
```

Наконец, командой `heroku create` нужно создать место на серверах Heroku для учебного приложения (листинг 1.15).

Листинг 1.15 ❖ Создание нового приложения на Heroku

```
$ heroku create
Creating damp-fortress-5769... done, stack is cedar
http://damp-fortress-5769.herokuapp.com/ | git@heroku.com:damp-fortress-5769.git
Git remote heroku added
```

Команда `heroku` создаст новый поддомен для приложения, который тут же станет доступным для просмотра. Однако там пока ничего нет, так что давайте займемся развертыванием.

1.5.2. Развертывание на Heroku, шаг первый

Первым шагом для развертывания является отправка главной ветви приложения в Heroku с помощью Git:

---

[1] http://api.heroku.com/signup.

[2] https://toolbelt.heroku.com/.

```
$ git push heroku master
```

(Может появиться несколько предупреждающих сообщений, которые сейчас можно игнорировать. Мы обсудим их ниже, в разделе 7.5.)

1.5.3. Развертывание на Heroku, шаг второй

Нет никакого шага под номером два! Все готово! Чтобы увидеть только что развернутое приложение, перейдите по адресу, который вы видели при выполнении `heroku create` (то есть в листинге 1.15). (Работающие на локальной машине вместо облачной IDE могут выполнить `heroku open`.) Результат показан на рис. 1.18. Страница идентична изображению на рис. 1.12, но теперь она запущена в эксплуатационном окружении – непосредственно в Интернете.

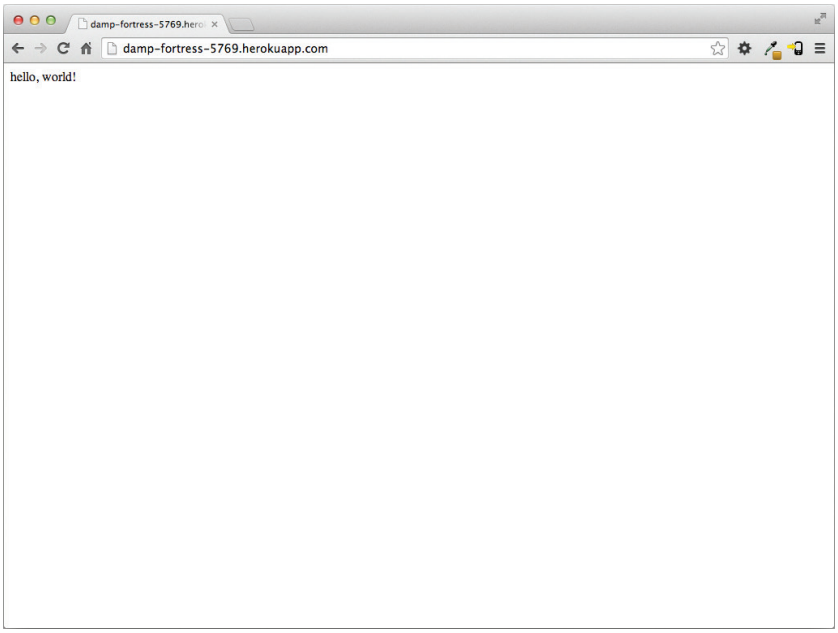

Рис. 1.18 ❖ Первое приложение из данного учебника, запущенное в Heroku

1.5.4. Команды Heroku

Существует великое множество команд Heroku[1], и мы только слегка коснемся их в этой книге. Уделю минуту, только чтобы показать одну из них – команду переименования:

```
$ heroku rename rails-tutorial-hello
```

[1] https://devcenter.heroku.com/articles/heroku-command.

Не используйте сами это имя; я его уже занял! На самом деле вам вряд ли стоит делать это прямо сейчас; вполне достаточно адреса, автоматически предоставленного Heroku. Но если вы действительно хотите переименовать свое приложение, вы сможете сделать это, а заодно защититься от непрошеных визитеров, используя случайный или невнятный поддомен, например такой:

```
hwpcbmze.herokuapp.com
seyjhflo.herokuapp.com
jhyicevg.herokuapp.com
```

С таким случайным поддоменом никто не сможет посетить ваш сайт, если только вы сами не дадите адрес. (Между прочим, в качестве анонса удивительной компактности Ruby ниже приводится код, который я использовал для генерации случайных поддоменов:

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

Очень даже неплохо.)

В дополнение к поддоменам Heroku также поддерживает пользовательские домены. (Фактически сайт[1] этой книги живет на Heroku; если вы читаете электронную версию книги в сети, значит, прямо сейчас смотрите на сайт, размещенный на Heroku!) Обращайтесь к документации Heroku за дополнительной информацией о пользовательских доменах и других возможностях Heroku.

1.6. Заключение

Мы проделали длинный путь в этой главе: установка, настройка среды разработки, управление версиями и развертывание. В следующей главе, опираясь на полученные знания, мы построим *мини-приложение*, связанное с базой данных, чтобы просто оценить возможности Rails.

Если вы хотите поделиться с общественностью своим прогрессом, не стесняйтесь твитнуть или изменить свой статус в Facebook на что-то вроде этого:

Я изучаю Ruby on Rails с @railstutorial!
http://www.railstutorial.org/

Также я рекомендую зарегистрироваться в списке рассылки Rails Tutorial[2], который гарантирует вам получение очередных обновлений (и эксклюзивных купонов), касающихся учебника.

1.6.1. Что мы узнали в этой главе

- Ruby on Rails – это фреймворк для разработки веб-приложений, написанный на языке программирования Ruby.

---

1 https://www.railstutorial.org/.

2 http://www.railstutorial.org/#email.

- Установка Rails, создание приложения, редактирование файлов – все это очень легко при использовании предварительно настроенной облачной среды разработки.
- В командную строку Rails встроена команда `rails`, которая может генерировать новые приложения (`rails new`) и запускать локальный сервер (`rails server`).
- Мы добавили метод контроллера и изменили корневой маршрут, чтобы создать приложение «hello, world».
- Поместив код приложения в систему управления версиями Git и отправив его в закрытый репозиторий Bitbucket, мы защитили себя от потери данных и получили возможность совместной работы над проектом.
- Мы развернули приложение в эксплуатационном окружении с помощью Heroku.

1.7. Упражнения

Примечание. *Руководство по решению упражнений бесплатно прилагается к любой покупке на* www.railstutorial.org.

1. Измените метод `hello` в листинге 1.8 так, чтобы он выводил «hola, mundo!» вместо «hello, world!». *Дополнительно:* убедитесь, что Rails поддерживает символы, не принадлежащие набору ASCII, использовав перевернутый восклицательный знак «¡Hola, mundo!» (рис. 1.19)[1].

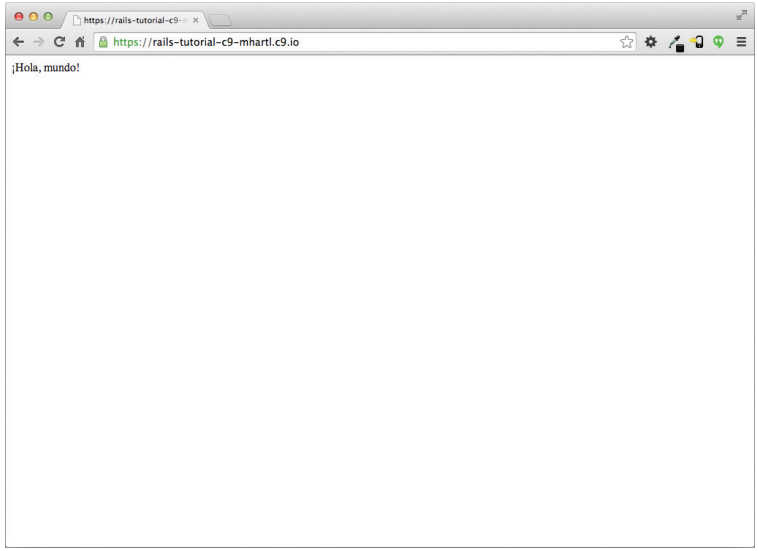

Рис. 1.19 ❖ Изменение действия контроллера для отображения «¡Hola, mundo!»

[1] Редактор может выдать сообщение, такое как: «invalid multibyte character» («неверный многобайтный символ»), но это не повод для беспокойства. Можете погуглить это сообщение, если вам интересно, как его устранить.

2. Следуя примеру метода действия `hello` из листинга 1.8, добавьте второй метод `goodbye`, который будет отображать текст «goodbye, world!». Отредактируйте файл маршрутов из листинга 1.10, чтобы корневой маршрут вел к методу `goodbye` вместо `hello` (рис. 1.20).

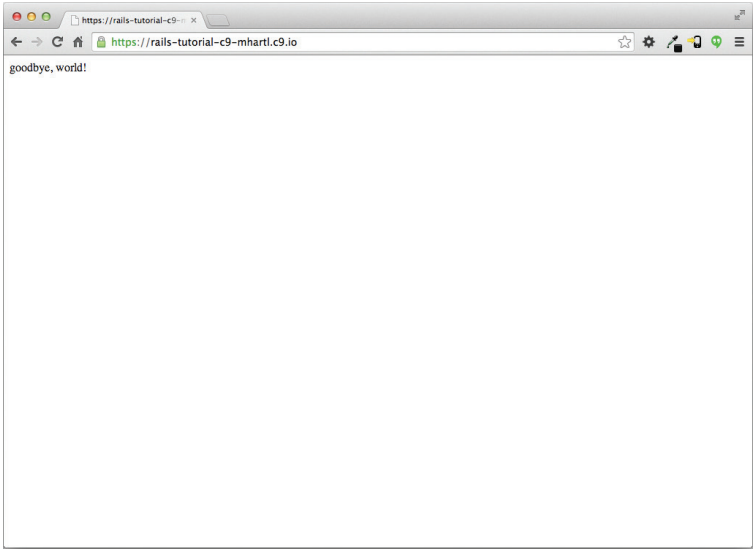

Рис. 1.20 ❖ Результат изменения корневого маршрута для вывода «goodbye, world!»