EL TUTORIAL DE RUBY ON RAILS

APRENDA DESARROLLO WEB CON RAILS

TERCERA EDICIÓN

LIBRO Y VIDEO EXPLICATIVOS POR MICHAEL HARTL

Tutorial de Ruby on Rails

Aprenda Desarrollo Web con Rails

Michael Hartl

Contenido

Prefacio

Mi anterior compañía (CD Baby) fue una de las primeras en cambiar efusivamente a Ruby on Rails, y luego aún más efusivamente, regresar a PHP (búsqueme en Google para leer acerca de esta tragedia). Este libro escrito por Michael Hartl vino tan altamente recomendado que tuve que probarlo, y el *Tutorial de Ruby on Rails* es lo que utilicé para regresar a Rails de nuevo.

Aunque he trabajado a mi modo con varios libros de Rails, éste es el que finalmente me hizo "comprenderlo". Todo se hace muy a "el modo Rails"—una forma que se sentía muy artificial para mí antes, pero ahora, luego de haber leído este libro, finalmente se siente natural. Éste es también el único libro de Rails que realiza desarrollo basado en pruebas todo el tiempo, un enfoque altamente recomendado por los expertos pero que nunca había sido demostrado claramente antes. Finalmente, al incluír Git, GitHub, y Heroku en los ejemplos del demo, el autor realmente le da una sensación de lo que es hacer un proyecto en el mundo real. Los códigos de ejemplo del tutorial no están aislados.

La narración lineal es un gran formato. Personalmente, me empapé del *Tutorial Rails* durante tres largos días,[1] realizando todos los ejemplos y retos al final de cada capítulo. Hágalo de principio a fin, sin saltar de un lado a otro, y obtendrá el beneficio primordial.

Disfrútelo!

Derek Sivers (sivers.org)
Fundador, CD Baby

[1] ¡Esto no es usual! Completar todo el libro usualmente toma *mucho* más tiempo que tres días.

Agradecimientos

El *Tutorial de Ruby on Rails* debe mucho a mi libro previo de Rails, *RailsSpace*, y por tanto a mi coautor Aurelius Prochazka. Me gustaría agradecer a Aure tanto el trabajo que hizo en aquél libro, como su apoyo para éste. También me gustaría agradecer a Debra Williams Cauley, mi editora tanto en *RailsSpace* como en el *Tutorial de Ruby on Rails*; siempre que ella me siga llevando a los juegos de béisbol, yo continuaré escribiendo libros para ella.

Me gustaría expresar mi reconocimiento a una larga lista de Rubyistas que me han enseñado e inspirado a través de los años: David Heinemeier Hansson, Yehuda Katz, Carl Lerche, Jeremy Kemper, Xavier Noria, Ryan Bates, Geoffrey Grosenbach, Peter Cooper, Matt Aimonetti, Mark Bates, Gregg Pollack, Wayne E. Seguin, Amy Hoy, Dave Chelimsky, Pat Maddox, Tom Preston-Werner, Chris Wanstrath, Chad Fowler, Josh Susser, Obie Fernandez, Ian McFarland, Steven Bristol, Pratik Naik, Sarah Mei, Sarah Allen, Wolfram Arnold, Alex Chaffee, Giles Bowkett, Evan Dorn, Long Nguyen, James Lindenbaum, Adam Wiggins, Tikhon Bernstam, Ron Evans, Wyatt Greene, Miles Forrest, la buena gente de Pivotal Labs, la pandilla Heroku, los chicos de pensamiento robot, y al equipo de GitHub. Finalmente, muchos, muchos lectores—demasiados para ser nombrados—han contribuído con un gran número de reportes de errores y sugerencias durante la escritura de este libro; les agradezco su ayuda para hacerlo lo mejor posible.

Sobre el Autor

Michael Hartl es el autor del *Ruby on Rails Tutorial*, uno de los tutoriales líderes en la introducción al desarrollo web, y es también el cofundador de Softcover, una plataforma de auto-publicación para autores. Previo a esto Michael escribió y desarrolló *RailsSpace*, un libro y tutorial de Rails en extremo obsoleto y desarrolló Insoshi, la alguna vez popular y ahora obsoleta red social en Ruby on Rails. En el 2011, Michael recibió el reconocimiento de Ruby Hero Award por sus contribuciones a la comunidad de Ruby. Es graduado del Harvard College, es doctor en Física por Caltech, y es un miembro del programa de emprendedores Y Combinator.

Derechos de autor y licencia

[2] Nota del Traductor: Dado que esta licencia la emite el MIT (Massachusetts Institute of Technology), el texto oficial sólo se encuentra disponible en inglés. La traducción que se presenta, fue tomada de la Wikipedia, donde también puede consultar la versión original.

```
/*
 * ----------------------------------------------------------------------------
 * "LA LICENCIA CERVEZA-WARE" (Revisión 43):
 * Michael Hartl escribió este código. Mientras que conserve este aviso, usted
 * puede hacer lo que quiera con este material. Si algún día nos conocemos, y si
 * usted cree que este material lo vale, puede comprarme una cerveza a cambio.
 * ----------------------------------------------------------------------------
 */
```

Capítulo 1

Desde cero hasta el despliegue

Bienvenido al *Tutorial de Ruby on Rails: Aprenda desarrollo web con Rails*. El propósito de este libro es enseñarle cómo desarrollar aplicaciones web personalizadas, y nuestra herramienta elegida es el popular marco de trabajo web Ruby on Rails. Si usted es neófito en el tema, el *Tutorial de Ruby on Rails* le proporcionará una minuciosa introducción al desarrollo de aplicaciones web, incluyendo un aprendizaje básico en Ruby, Rails, HTML & CSS, bases de datos, control de versiones, pruebas, y despliegue—suficiente para lanzarlo en una carrera como desarrollador web o emprendendor en tecnología. Por otra parte, si usted ya conoce el desarrollo web, este libro le enseñará rápidamente lo esencial del marco de trabajo de Rails, incluyendo MVC y REST, generadores, migraciones, ruteos y Ruby embebido. En cualquier caso, cuando usted termine el *Tutorial de Ruby on Rails* estará en posición de beneficiarse de libros mucho más avanzados, blogs y videos que son parte del floresciente ecosistema educativo en programación.[1]

El *Tutorial de Ruby on Rails* adopta un enfoque integrado al desarrollo web construyendo tres aplicaciones de ejemplo con creciente sofisticación, empezando con una aplicación mínima *hola* (Sección 1.3), una aplicación ligera-

[1]La versión más actualizada del *Tutorial de Ruby on Rails* puede encontrarla en el sitio web del libro http://www.railstutorial.org/. Si usted está leyendo este libro fuera de línea, asegúrese de revisar la versión en línea del Tutorial de Ruby on Rails en http://www.railstutorial.org/book para conocer las últimas actualizaciones.

mente más compleja *de juguete* (Capítulo 2), y una aplicación *de ejemplo* real (Capítulos 3 al 12). Como se deduce de sus nombres genéricos, las aplicaciones desarrolladas en el *Tutorial de Ruby on Rails* no son específicas de ningún tipo de sitio web; aunque la aplicación de ejemplo final tendrá algo más que una sutil semejanza con cierto sitio social de micromensajes muy popular (un sitio que, coincidentemente, también fue originalmente escrito en Rails), el énfasis a lo largo del tutorial está en principios generales, de forma que usted obtenga una base sólida, sin importar qué tiempo de aplicaciones web desee usted crear.

Una pregunta común es cuánto conocimiento o experiencia previa es necesaria para aprender desarrollo web mediante el *Tutorial de Ruby on Rails*. Como se comenta con mayor profundidad en la Sección 1.1.1, el desarrollo web es un reto, especialmente para los novatos. Aunque el tutorial fue originalmente diseñado para lectores con alguna experiencia previa en programación o en desarrollo web, en realidad ha encontrado una audiencia significativa entre los desarrolladores principiantes. Teniendo esto en cuenta, esta tercera edición del *Tutorial de Rails Tutorial* ha dado varios pasos importantes encaminados a reducir la barrera para empezar con Rails (Recuadro 1.1).

Recuadro 1.1. Reduciendo la barrera

Esta tercera edición del *Tutorial de Ruby on Rails* tiene por objetivo reducir la barrera para empezar con Rails de varias formas:

- El uso de un ambiente de desarrollo estándar en la nube (Sección 1.2), el cual deja a un lado muchos de los problemas asociados con la instalación y configuración de un nuevo sistema
- El uso del “default stack” de Rails, incluyendo el marco de trabajo incorporado para realizar mini-pruebas
- La eliminación de muchas dependencias externas (RSpec, Cucumber, Capybara, Factory Girl)
- Un enfoque más ligero y flexible de las pruebas

- Aplazamiento o eliminación de opciones de configuración complejas (Spork, RubyTest)
- Menos énfasis en características específicas de cierta versión de Rails, con mayor énfasis en principios generales del desarrollo web

Espero que estos cambios hagan que la tercera edición del *Tutorial de Ruby on Rails* sea accesible a una audiencia aún mayor que la de las versiones anteriores.

En este primer capítulo, empezaremos con Ruby on Rails instalando todo el software necesario y configurando nuestro ambiente de desarrollo (Sección 1.2). Luego crearemos nuestra primera aplicación Rails, llamada **`hello_app`**. El *Tutorial de Rails* enfatiza el uso de buenas prácticas para el desarrollo de software, por lo que inmediatamente después de crear nuestro nuevo proyecto de Rails, lo pondremos bajo un control de versiones con Git (Sección 1.4). Y, créalo o no, en este capítulo llegaremos a colocar nuestra primera aplicación en internet al *desplegarla* en producción (Sección 1.5).

En el Capítulo 2, crearemos un segundo proyecto, cuyo propósito es demostrar las funcionalidades básicas de una aplicación Rails. Para ponernos en marcha rápidamente, construiremos una *aplicación de juguete* (llamada **`toy_app`**) usando un programa generador de estructuras temporales (Recuadro 1.2) para generar código; como este código es tan feo como complejo, en el Capítulo 2 nos enfocaremos en interactuar con la aplicación de juguete a través de sus *URIs* (a menudo llamadas *URLs*)[2] cuando lo está utilizando.

El resto del tutorial se enfoca en desarrollar una gran *aplicación real de ejemplo* (llamada **`sample_app`**), escribiendo todo el código desde cero. Desarrollaremos tal aplicación usando una combinación de *bosquejos*, *desarrollo orientado a pruebas* (TDD por sus siglas en inglés *Test Driven Development*) y *pruebas de integración*. Empezaremos en el Capítulo 3 creando páginas estáticas y luego agregando un poco de contenido dinámico. Nos desviaremos

[2] *URI* por sus siglas en inglés *Uniform Resource Identifier*, mientras que la ligeramente menos genérica *URL* por sus siglas en inglés *Uniform Resource Locator*. En la práctica, la URL es usualmente equivalente a "lo que usted ve en la barra de direcciones de su navegador".

ligeramente en el Capítulo 4 para aprender un poco acerca del lenguaje Ruby que es la base de Rails. Luego, en los Capítulos 5 a 10, completaremos las bases de la aplicación de ejemplo creando una estructura de diseño para el sitio, un modelo de datos de usuario y un registro completo junto con su sistema de autenticación (incluyendo la activación de la cuenta y el reinicio de contraseñas). Finalmente, en los Capítulos 11 a 12 agregaremos las características sociales y de micromensajes para hacer del sitio de ejemplo, un sitio funcional.

Recuadro 1.2. Generador de estructuras temporales: Más rápido, más fácil, más seductor

Desde el principio, Rails se ha beneficiado de un palpable sentido de la emoción, empezando con el famoso video de un weblog en 15-minutos del creador de Rails, David Heinemeier Hansson. Ese video y sus sucesores son una excelente probadita del poder de Rails; le recomiendo que los vea. Pero le advierto: para realizar la asombrosa hazaña en quince minutos utilizan una característica llamada *scaffolding*, que depende en gran medida de *código generado*, mágicamente creado por el comando `generate scaffold` de Rails.

Al escribir el tutorial de Ruby on Rails, es tentador depender de la generación de código—es más rápido, más fácil, más seductor. Pero la complejidad y la cantidad de código generado de esta manera puede resultar completamente abrumador para un desarrollador principiante de Rails; usted puede ser capaz de utilizarlo, pero probablemente no lo entienda. Seguir el desarrollo usando la generación automática de código lo pone en riesgo de convertirlo en un generador virtuoso de scripts con poco (y frágil) conocimiento real de Rails.

En el *Tutorial de Ruby on Rails*, tomaremos la ruta (casi) opuesta: aunque en el Capítulo 2 desarrollaremos una pequeña aplicación de juguete usando esta técnica, la parte más importante del *Tutorial de Rails* es la aplicación de ejemplo, la cual empezaremos a escribir en el Capítulo 3. Durante el desarrollo de la aplicación de ejemplo, escribiremos *pequeños pedazos* de código—suficientemente simples de entender, pero también suficientemente nuevos como para ser desafiantes. El efecto acumulado será un conocimiento más profundo y más flexible de Rails,

proporcionándole una buena base para escribir casi cualquier tipo de aplicación web.

1.1 Introducción

Ruby on Rails (o sólo "Rails" para abreviar) es un marco de trabajo para desarrollo web escrito en el lenguaje de programación Ruby. Desde su debut en 2004, Ruby on Rails se ha convertido rápidamente en una de las herramientas más poderosas y populares para construír aplicaciones web dinámicas. Rails es utilizado en compañías tan diversas como AirBnB, Basecamp, Disney, GitHub, Hulu, Kickstarter, Shopify, Twitter y las Páginas Amarillas. Existen también muchas tiendas de desarrollo web que se especializan en Rails, tales como ENTP, Thoughtbot, Pivotal Labs, Hashrocket y HappyFunCorp, además de innumerables consultores independientes, entrenadores y contratistas.

¿Qué es lo que hace a Rails tan fantástico? Antes que nada, Ruby on Rails es 100% de código abierto, disponible bajo la permisiva Licencia MIT, y como resultado no cuesta nada descargarlo ni usarlo. Rails también debe mucho de su éxito a su diseño compacto y elegante; explotando la maleabilidad del lenguaje subyacente Ruby, Rails crea efectivamente un lenguaje de dominio específico para escribir aplicaciones web. Como resultado, muchas tareas de programación web comunes—tales como la generación de HTML, elaboración de modelos de datos y ruteo de URLs—son fáciles con Rails, y el código resultante de la aplicación es conciso y legible.

Rails también se adapta rápidamente a nuevos desarrollos en tecnología web y al diseño de marcos de trabajo. Por ejemplo, Rails fue uno de los primeros marcos de trabajo en implementar completamente el estilo de arquitectura REST para estructurar aplicaciones web (el cual estaremos aprendiendo a lo largo del tutorial). Y cuando otros marcos de trabajo desarrollan nuevas técnicas exitosas, el creador de Rails David Heinemeier Hansson y el equipo principal de Rails no dudan en incorporar esas ideas. Quizá el ejemplo más dramático es la fusión de Rails y Merb, un marco de trabajo web rival también

en Ruby, por lo que Rails ahora aprovecha el diseño modular y estable de la API de Merb y cuenta con un desempeño mejorado.

Finalmente, Rails se beneficia de una inusualmente entusiasta y diversa comunidad. Los resultados incluyen cientos de contribuidores de código abierto, congresos bien atendidos, y un gran número de gemas (soluciones auto-contenidas a problemas específicos tales como la paginación y la subida de imágenes a servidor), una rica variedad de blogs informativos, y una enorme cantidad de foros de discusión y canales IRC. El gran número de programadores Rails también hace más fácil de manejar los inevitables errores de aplicación: el algoritmo "busca en Google el mensaje de error" casi siempre conduce a una entrada relevante en un blog o a una discusión en un foro.

1.1.1 Prerequisitos

No hay prerequisitos formales para este libro—el *Tutorial de Ruby on Rails* contiene tutoriales integrados no sólo para Rails, sino también para el lenguaje subyacente Ruby, el marco de trabajo para pruebas por default (minitest), la línea de comandos de Unix, HTML, CSS, una pequeña cantidad de JavaScript y un poquito de SQL. Es una gran cantidad de material para absorber, aunque yo generalmente recomiendo tener algo de bases en HTML y en programación antes de empezar este tutorial. Dicho esto, una sorprendente cantidad de principiantes han utilizado el *Tutorial de Ruby on Rails* para aprender desarrollo web desde cero, por lo que si usted tiene experiencia limitada le sugiero intentarlo. Si se siente abrumado, siempre puede empezar con uno de los recursos listados más adelante y luego regresar. Otra estrategia recomendada por múltiples lectores es simplemente hacer el tutorial dos veces; puede sorprenderle cuánto aprendió la primera vez (y cuán fácil es la segunda).

Una pregunta común al aprender Rails es si se requiere aprender Ruby primero. La respuesta depende de su estilo de aprendizaje personal y en cuánta experiencia de programación tenga. Si usted prefiere aprender todo sistemáticamente desde el principio, o si nunca ha programado antes, entonces aprender Ruby primero puede funcionarle bien, y en tal caso le recomiento *Learn to Program* de Chris Pine y *Beginning Ruby* de Peter Cooper. Por otra parte, muchos desarrolladores principiantes en Rails están emocionados de crear aplicaciones

web, y prefieren no esperar a terminar un libro completo de Ruby antes de escribir una sola página web. En tal caso, le recomiendo seguir el pequeño tutorial interactivo de Try Ruby[3] para tener una visión general antes de empezar con el *Tutorial de Rails*. Si aún así encuentra este tutorial demasiado difícil, puede intentar comenzar con *Learn Ruby on Rails* de Daniel Kehoe o One Month Rails, dos de los cuales se enfocan en totales novatos más que el *Tutorial de Ruby on Rails*.

Al final de este tutorial, no importa dónde haya empezado, usted debería estar listo para los muchos recursos de nivel intermedio a avanzado de Rails que hay allá afuera. Aquí hay algunos que yo particularmente recomiendo:

- Code School: Buenos cursos de programación interactivos en línea.
- The Turing School of Software & Design: un programa de entrenamiento de tiempo completo, cuya duración es de 27 semanas en Denver, Colorado, con un descuento de $500 USD para los lectores de este tutorial usando el código RAILSTUTORIAL500.
- Bloc: Un campamento de entrenamiento en línea con un programa estructurado, asesoría personalizada y un enfoque en aprendizaje a través de proyectos concretos. Use el cupón BLOCLOVESHARTL para obtener un descuento de $500 USD en la inscripción (matrícula).
- Tealeaf Academy: Un buen campo de entrenamiento en desarrollo Rails en línea (incluye material avanzado).
- Thinkful: Una clase en línea que lo contacta con un ingeniero profesional conforme trabaja en un programa basado en proyectos.
- Pragmatic Studio: Cursos de Ruby y Rails en línea, de Mike y Nicole Clark. Junto con el autor de *Programming Ruby*, Dave Thomas, Mike impartió el primer curso de Rails que tomé, allá en el año 2006.
- RailsCasts de Ryan Bates: Excelentes videos de Rails (la mayoría son gratuitos).

[3] http://tryruby.org/

- RailsApps: Una gran variedad de proyectos de Rails detallados en un tema específico y tutoriales.
- Rails Guides: Referencias temáticas y actualizadas de Rails.

1.1.2 Acuerdos utilizados en este libro

Los acuerdos de este libro son mayormente auto-explicativos. En esta sección, mencionaré algunos que podrían no serlo.

Muchos ejemplos del libro utilizan comandos en la línea de comandos. Por simplicidad, todos los ejemplos de línea de comandos utilizan un cursor del estilo Unix (signo de dólar), como sigue:

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

Como mencionamos en la Sección 1.2, le recomiendo a los usuarios de todos los sistemas operativos (especialmente Windows) que utilicen un ambiente de desarrollo en la nube (Sección 1.2.1), el cual incluye una línea de comandos Unix (Linux). Esto es particularmente útil porque Rails tiene muchos comandos que pueden ejecutarse en la línea de comandos. Por ejemplo, en la Sección 1.3.2 ejecutaremos un servidor de desarrollo web local con el comando **`rails server`**:

```
$ rails server
```

Como con el cursor de la línea de comandos, el *Tutorial de Rails* utiliza la convención de Unix para los delimitadores de directorios (es decir, una diagonal **`/`**). Por ejemplo, el archivo de configuración de la aplicación de ejemplo **`production.rb`** se muestra como sigue:

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

Esta ruta al archivo debe entenderse como relativa al directorio raíz de la aplicación, la cual variará dependiendo del sistema; en el IDE en la nube (Sección 1.2.1), sería como ésta:

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

Por lo que, la ruta absoluta al archivo **`production.rb`** sería

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

Por brevedad, típicamente omitiré la ruta de la aplicación y escribiré sólo **`config/environments/production.rb`**.

El *Tutorial de Rails* a menudo muestra la salida de varios programas (comandos de terminal, status del control de versiones, programas Ruby, etc.). Debido a innumerables pequeñas diferencias entre los sistemas de cómputo, la salida que usted visualiza no siempre coincidirá exactamente con lo que se muestra en el texto, pero esto no debe preocuparle. Adicionalmente, algunos comandos muestran errores dependiendo de su sistema; en vez de intentar la colosal tarea de documentar todos esos errores en este tutorial, se lo delego al algoritmo "busque en Google el mensaje de error", el cual entre otras cosas es una buena práctica en el desarrollo de software en la vida real. Si usted encuentra algún problema mientras sigue el tutorial, le sugiero consultar los recursos listados en la sección Ayuda del Tutorial de Rails.[4]

Como el *Tutorial de Rails* cubre las pruebas de las aplicaciones de Rails, a menudo es útil saber si un pedazo particular de código provoca que el conjunto de pruebas completo falle (indicado por el color rojo) o pase (indicado por el color verde). Por convención, el código que resulta en una prueba fallida es etiquetado como **ROJO**, mientras que el código que resulta en una prueba exitosa es etiquetado como **VERDE**.

Cada capítulo del tutorial incluye ejercicios; realizarlos es opcional pero recomendado. Con la finalidad de mantener el tema principal independiente de los ejercicios, las soluciones generalmente no son incorporadas en listados

[4] http://www.railstutorial.org/#help

subsecuentes. En el raro caso de que la solución a un ejercicio se utilice posteriormente, éste será solucionado explícitamente en el texto principal.

Finalmente, por conveniencia, el *Tutorial de Ruby on Rails* adopta dos convenciones diseñadas para hacer que los códigos de ejemplo sean más fáciles de entender. La primera consiste en que algunos listados de código incluyen una o más líneas resaltadas, como se muestra a continuación:

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

Tales líneas usualmente indican el código nuevo más importante del ejemplo dado, y a menudo (aunque no sucede siempre) representan la diferencia entre el nuevo listado de código y los anteriores. La segunda consiste en que, por claridad y simplicidad, muchos de los listados de código incluyen puntos suspensivos verticales, como los siguientes:

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

Estos puntos representan código omitido que no debe copiarse literalmente.

1.2 En funcionamiento

Aún para desarrolladores Rails con experiencia, instalar Ruby, Rails, y todo el software asociado puede ser un ejercicio frustrante. Las causas del problema son los múltiples escenarios: diferentes sistemas operativos, números de versión, preferencias en el editor de texto y en el ambiente de desarrollo integrado (IDE, por sus siglas en inglés *Integrated Development Environment*), etc. Los usuarios que ya tienen instalado un IDE en su equipo local pueden utilizar sus

configuraciones preferidas, pero (como se menciona en el Recuadro 1.1) se les sugiere a los nuevos usuarios dejar a un lado esta instalación y los detalles de configuración y se les invita a que utilicen un *ambiente de desarrollo integrado en la nube*. Este ambiente en la nube se ejecuta dentro de un navegador web ordinario y por tanto funciona de la misma forma en las diferentes plataformas, lo cual es especialmente útil para los sistemas operativos (tales como Windows) en los que el desarrollo con Rails ha sido históricamente difícil. Si aún a pesar de los retos que involucra, usted prefiere completar el *Tutorial de Ruby on Rails* usando un ambiente de desarrollo local, le recomiendo seguir las instrucciones que están en InstallRails.com.[5]

1.2.1 Ambiente de desarrollo

Considerando las diferentes personalizaciones idiosincrásicas, existen probablemente tantos ambientes de desarrollo como programadores Rails. Para evitar esta complejidad, el *Tutorial de Ruby on Rails* se estandariza en el excelente ambiente de desarrollo en la nube Cloud9. En particular, para esta tercera edición me complace tener una alianza con Cloud9 para ofrecer un ambiente de desarrollo específicamente adaptado a las necesidades de este tutorial. El espacio de trabajo resultante viene pre-configurado con la mayoría del software necesario para desarrolladores profesionales de Rails, incluyendo Ruby, RubyGems y Git. (De hecho, el único software que instalaremos por separado es Rails mismo, y esto es intencional (Sección 1.2.2).) El IDE en la nube incluye los tres componentes esenciales que se requieren para desarrollar aplicaciones web: un editor de texto, un navegador de archivos y una terminal de línea de comandos (Figura 1.1). Entre otras características, el editor de texto del IDE en la nube soporta la búsqueda global "Buscar dentro de archivos" que considero esencial para navegar dentro de un proyecto grande de Ruby o de Rails.[6] Finalmente, aún si usted decide no utilizar el IDE en la nube de forma exclusiva en la vida real (y ciertamente le recomiendo aprender a utilizar otras

[5] Aún así, a los usuarios de Windows se les advierte que el instalador de Rails recomendado por InstallRails a menudo está desactualizado, y es muy probable que sea incompatible con el presente tutorial.

[6] Por ejemplo, para buscar la definición de una función llamada `foo`, usted puede realizar una búsqueda global de "def foo".

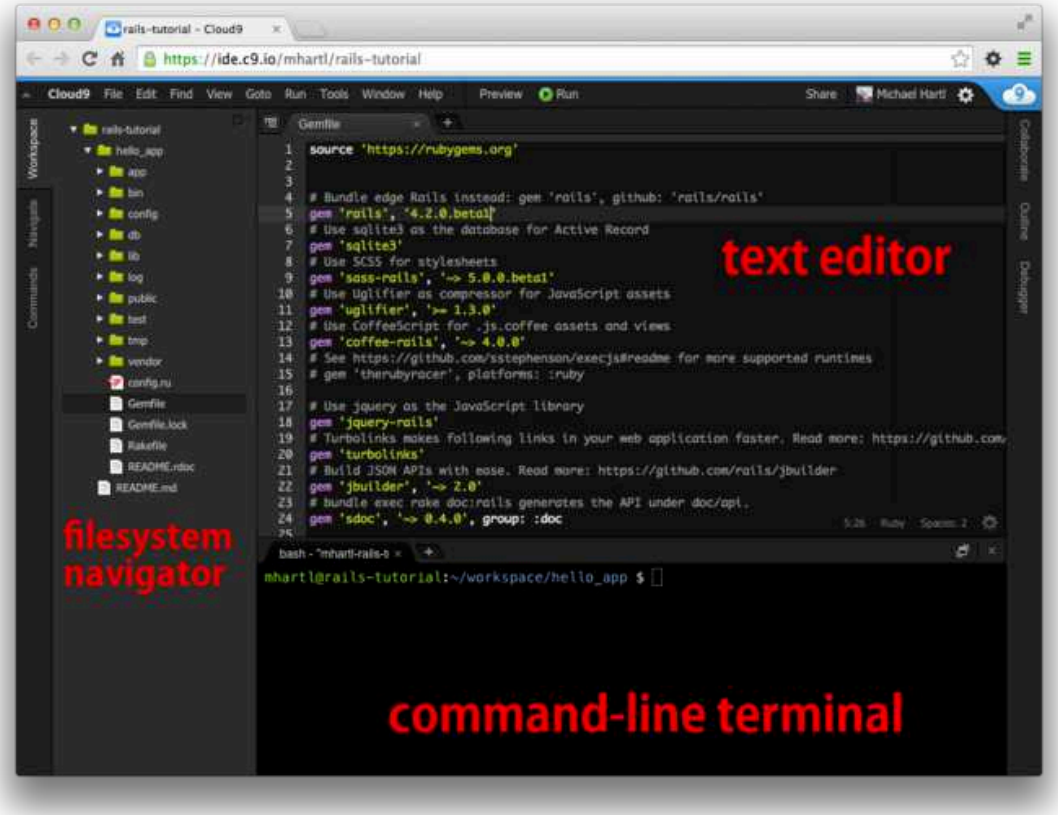

Figura 1.1: La anatomía de el IDE en la nube.

herramientas también), proporciona una excelente introducción a las capacidades generales de los editores de texto y otras herramientas de desarrollo.

Estos son los pasos para empezar a utilizar el ambiente de desarrollo en la nube:

1. Regístrese para obtener una cuenta gratuita en Cloud9[7]

2. Presione el botón “Ir al panel de control”

3. Seleccionar “Crear un nuevo espacio de trabajo”

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

4. Como se muestra en la Figura 1.2, cree un espacio de trabajo llamado "rails-tutorial" (*no* "rails_tutorial"), seleccione la opción "Privado para la gente que yo invite", y seleccione el ícono del Tutorial de Rails (*no* el ícono de Ruby on Rails)

5. Presione el botón "Crear"

6. Luego de que Cloud9 haya terminado de preparar su espacio de trabajo, selecciónelo y presione el botón "Empezar a editar"

Como el uso de dos espacios para indentar es una convención casi-universal en Ruby, le recomiendo configurar el editor para que utilice estos dos espacios en vez de los cuatro que usa por default. Como se muestra en la Figura 1.3, usted puede hacer esto presionando el ícono de engrane que se encuentra en la esquina superior derecha y luego seleccione "Editor de Código (Ace)" para editar la configuración "Soft Tabs". (Observe que esto toma efecto inmediatamente; por lo que no necesita presionar el botón "Guardar".)

1.2.2 Instalando Rails

El ambiente de desarrollo de la Sección 1.2.1 incluye todo el software necesario para empezar excepto por el mismo Rails.[8] Para instalar Rails, utilizaremos el comando `gem` proporcionado por el administrador de paquetes *RubyGems*, lo cual implica escribir el comando que se muestra en el Listado 1.1 en su terminal de línea de comandos. (Si usted está desarrollando en su sistema local, esto significa que utilice una terminal regular; si está utilizando el IDE en la nube, esto significa que utilice el área de línea de comandos que se muestra en la Figura 1.1.)

Listado 1.1: Instalando Rails con un número específico de versión.

```
$ gem install rails -v 4.2.2
```

[8] Actualmente, Cloud9 incluye una versión vieja de Rails que es incompatible con el presente tutorial, razón por la cual es importante que lo instalemos nosotros mismos.

Figura 1.2: Creando un nuevo espacio de trabajo en Cloud9.

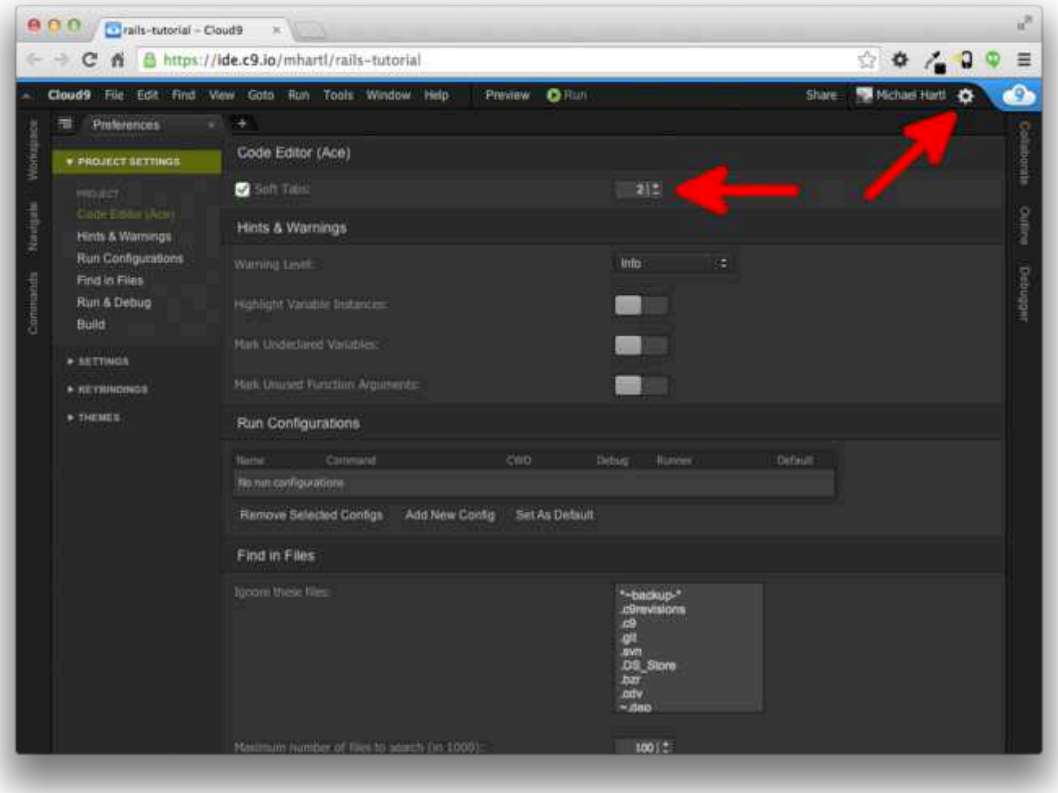

Figura 1.3: Configurando Cloud9 para que utilice dos espacios en la indentación.

Aquí la opción `-v` se encarga de que la versión de Rails especificada sea instalada, lo cual es importante para obtener resultados consistentes con este tutorial.

1.3 La primera aplicación

Siguiendo una larga tradición en la programación de computadoras, nuestro objetivo para la primera aplicación es escribir un programa "hola mundo". En particular, crearemos una aplicación simple que muestre la cadena "hello, world!" en una página web, tanto en nuestro ambiente de desarrollo (Sección 1.3.4) como en internet (Sección 1.5).

Virtualmente todas las aplicaciones Rails empiezan de la misma forma, ejecutando el comando `rails new`. Este útil comando crea un esqueleto de la aplicación Rails en un directorio que usted elija. Para empezar, los usuarios que *no* están utilizando el IDE en la nube que recomendamos en la Sección 1.2.1, deberían crear un directorio `workspace` para sus proyectos de Rails, si es que aún no tienen uno (Listado 1.2) y luego posicionarse en ese directorio. (El Listado 1.2 utiliza los comandos de Unix `cd` y `mkdir`; vea el Recuadro 1.3 si usted no está familiarizado con estos comandos.)

Listado 1.2: Creando un directorio `workspace` para los proyectos de Rails (no es necesario en la nube).

```
$ cd                 # Change to the home directory.
$ mkdir workspace    # Make a workspace directory.
$ cd workspace/      # Change into the workspace directory.
```

Recuadro 1.3. Un breve curso acerca de la línea de comandos de Unix

Para los lectores que utilizan Windows o (en menor grado pero aún significativo) Macintosh OS X, la línea de comandos de Unix puede que no les resulte

Descripción	Comando	Ejemplo
enlista contenidos	`ls`	`$ ls -l`
crea directorio	`mkdir <dirname>`	`$ mkdir workspace`
cambia de directorio	`cd <dirname>`	`$ cd workspace/`
cambia al directorio padre		`$ cd ..`
cambia al directorio *home*		`$ cd ~` o sólo `$ cd`
cambia al subdirectorio de *home* indicado		`$ cd ~/workspace/`
renombra un archivo	`mv <source> <target>`	`$ mv README.rdoc README.md`
copia un archivo	`cp <source> <target>`	`$ cp README.rdoc README.md`
borra un archivo	`rm <file>`	`$ rm README.rdoc`
borra un directorio vacío	`rmdir <directory>`	`$ rmdir workspace/`
borra un directorio no vacío	`rm -rf <directory>`	`$ rm -rf tmp/`
muestra el contenido de un archivo	`cat <file>`	`$ cat ~/.ssh/id_rsa.pub`

Tabla 1.1: Algunos comandos Unix comunes.

familiar. Afortunadamente, si usted está utilizando el ambiente en la nube recomendado, automáticamente tiene acceso a la línea de comandos de Unix (Linux) ejecutando una terminal de línea de comandos estándar conocida como Bash.

La idea básica de la línea de comandos es simple: al emitir comandos cortos, los usuarios realizan una gran cantidad de operaciones, tales como crear directorios (`mkdir`), mover y copiar archivos (`mv` y `cp`) y navegar en el sistema de archivos al cambiar de directorio (`cd`). Aunque la línea de comandos puede parecer primitiva para los usuarios que están familiarizados principalmente con interfaces gráficas (GUIs), las apariencias engañan: la línea de comandos es una de las herramientas más poderosas disponibles para un desarrollador. De hecho, rara vez usted verá el escritorio de un desarrollador experimentado sin varias ventanas de terminales abiertas ejecutando programas de línea de comandos.

El tema en general es profundo, pero para los propósitos de este tutorial sólo necesitamos algunos de los comandos de Unix más comunes, como se resume en la Tabla 1.1. Para conocer más detalladamente la línea de comandos de Unix, vea *Conquering the Command Line* de Mark Bates (disponible en versión en línea gratuita, libros y videos).

El siguiente paso tanto en sistemas locales como en el IDE en la nube es

crear la primera aplicación usando el comando del Listado 1.3. Observe que el Listado 1.3 explícitamente incluye el número de versión de Rails (`_4.2.2_`) como parte del comando. Esto asegura que la misma versión de Rails que instalamos en el Listado 1.1 es utilizada para crear la estructura de archivos de la primera aplicación. (Si el comando del Listado 1.3 regresa un error como "Could not find 'railties'", significa que usted no tiene instalada la versión correcta de Rails y debería verificar nuevamente que ha ejecutado el comando del Listado 1.1 exactamente como está escrito.)

Listado 1.3: Ejecutando `rails new` (con un número de versión específico).

```
$ cd ~/workspace
$ rails _4.2.2_ 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
```

Como puede observar al final del Listado 1.3, ejecutar `rails new` automáticamente invoca el comando `bundle install` luego de haber terminado la creación de archivos. Revisaremos con mayor detalle qué significa esto al empezar la Sección 1.3.1.

Observe cuántos archivos y directorios ha creado el comando `rails`. Esta estructura estándar de directorios y archivos (Figura 1.4) es una de las muchas ventajas de Rails; le lleva a usted de cero a una aplicación funcional (mínima). Más aún, como la estructura es común a todas las aplicaciones de Rails, usted puede orientarse inmediatamente cuando revise el código de alguien más. Un resumen de los archivos Rails creados por default, se muestra en la Tabla 1.2; aprenderemos más acerca de estos archivos y directorios a lo largo del libro. En particular, empezando la Sección 5.2.1 revisaremos el directorio `app/assets`, como parte de la *cadena de procesos conectados* que facilita más que nunca la organización y despliegue de recursos tales como hojas de estilo en cascada y archivos JavaScript.

1.3.1 Bundler

Luego de crear una aplicación nueva de Rails, el siguiente paso es utilizar *Bundler* para instalar e incluír las gemas necesarias para la aplicación. Como mencionamos brevemente en la Sección 1.3, Bundler es ejecutado automáticamente (mediante `bundle install`) por el comando `rails`, pero en esta sección haremos algunos cambios a las gemas de la aplicación indicadas por default y ejecutaremos Bundler de nuevo. Esto implica abrir el archivo `Gemfile` en un editor de texto. (Con el IDE en la nube, esto significa dar click en la flecha del navegador de archivos para abrir el directorio de la aplicación de ejemplo y dar doble click en el ícono `Gemfile`.) Aunque los números de versión exacta y los detalles pueden variar ligeramente, los resultados deben verse similares a los de la Figura 1.5 y los del Listado 1.4. (El código de este archivo es Ruby, pero en este momento no se preocupe de la sintaxis; en el Capítulo 4 revisaremos Ruby con mayor detalle.) Si los archivos y los directorios no aparecen como los de la Figura 1.5, presione el ícono de engrane en el navegador de archivos y seleccione “Refrescar árbol de archivos”. (Como regla general, deber refrescar el árbol de archivos cada vez que los directorios

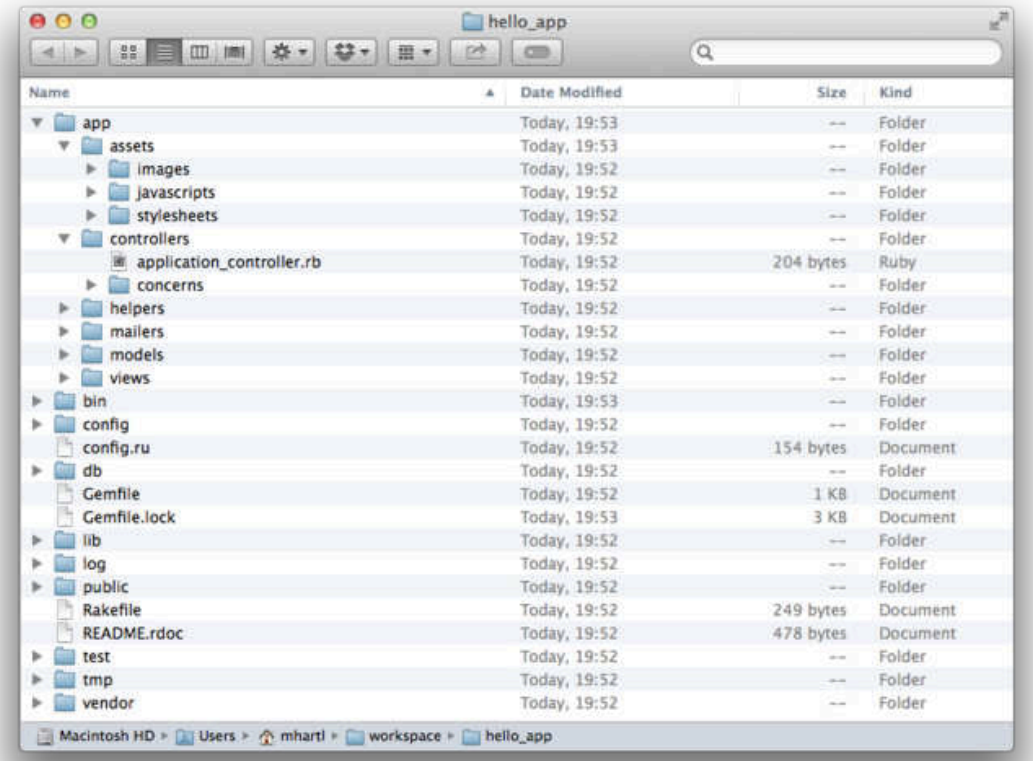

Figura 1.4: La estructura de directorios de una aplicación Rails recién creada.

Archivo/Directorio	Propósito
`app/`	Código principal de la aplicación (app), incluye modelos, vistas, controladores y auxiliares
`app/assets`	Recursos de la aplicación tales como hojas de estilo en cascada (CSS), archivos JavaScript e imágenes
`bin/`	Archivos binarios ejecutables
`config/`	Configuración de la aplicación
`db/`	Archivos de base de datos
`doc/`	Documentación de la aplicación
`lib/`	Biblioteca de módulos
`lib/assets`	Biblioteca de recursos tales como hojas de estilo en cascada (CSS), archivos JavaScript e imágenes
`log/`	Archivos de bitácora de la aplicación
`public/`	Datos accesibles al público (por ejemplo, vía navegadores web), tales como páginas de error
`bin/rails`	Un programa para generar código, abrir sesiones de consola o iniciar un servidor local
`test/`	Pruebas de la aplicación
`tmp/`	Archivos temporales
`vendor/`	Código de terceros tales como complementos (plugins) y gemas
`vendor/assets`	Recursos de terceros tales como hojas de estilo en cascada (CSS), archivos JavaScript e imágenes
`README.rdoc`	Una breve descripción de la aplicación
`Rakefile`	Tareas de utilería disponibles mediante el comando `rake`
`Gemfile`	Gemas requeridas para esta aplicación
`Gemfile.lock`	Una lista de gemas utilizadas para asegurar que todas las copias de la aplicación utilicen las mismasversion
`config.ru`	Un archivo de configuración para el software intermediario Rack
`.gitignore`	Patrones de nombres de archivos que deben ser ignorados por Git

Tabla 1.2: Un resumen de la estructura de directorios por default de Rails.

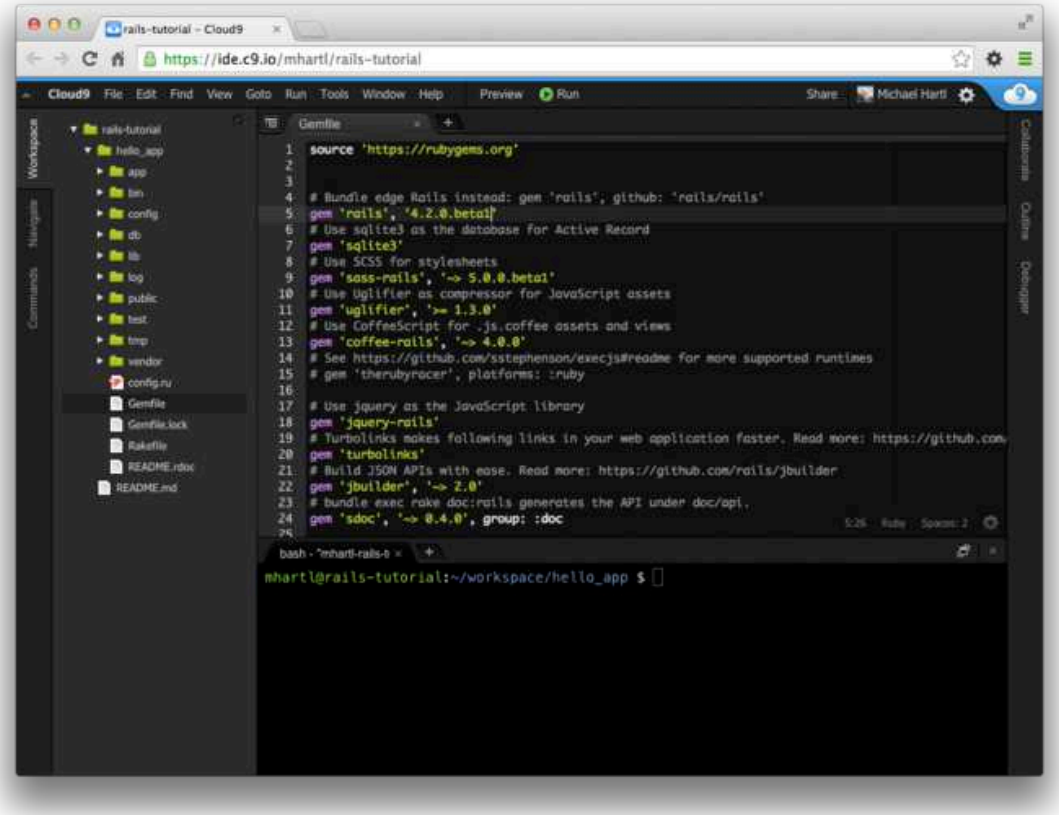

Figura 1.5: El archivo **Gemfile** por default abierto en un editor de texto.

o archivos no aparezcan como se espera.)

Listado 1.4: El archivo **Gemfile** por default en el directorio **hello_app**.

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


# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.2'
# Use sqlite3 as the database for Active Record
gem 'sqlite3'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
```

```
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'
# See https://github.com/sstephenson/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby

# Use jquery as the JavaScript library
gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read more:
# https://github.com/rails/turbolinks
gem 'turbolinks'
# Build JSON APIs with ease. Read more: 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

# Use ActiveModel has_secure_password
# gem 'bcrypt', '~> 3.1.7'

# Use Unicorn as the app server
# gem 'unicorn'

# Use Capistrano for deployment
# gem 'capistrano-rails', group: :development

group :development, :test do
  # Call 'debugger' anywhere in the code to stop execution and get a
  # debugger console
  gem 'byebug'

  # Access an IRB console on exceptions page and /console in development
  gem 'web-console', '~> 2.0.0.beta2'

  # Spring speeds up development by keeping your application running in the
  # background. Read more: https://github.com/rails/spring
  gem 'spring'
end
```

Muchas de estas líneas están comentadas mediante el símbolo de número **#**; están ahí para mostrarle algunas de las gemas comúnmente utilizadas y para dar ejemplos de la sintaxis de Bundler. Por ahora, no necesitamos más que las gemas por default.

A menos que usted especifique un número de versión al comando **gem**, Bundler automáticamente instalará la versión más reciente de la gema requerida. Un ejemplo de este caso es:

```
gem 'sqlite3'
```

Existen también otras dos formas de especificar un rango de versiones para una gema, lo cual nos permite tener control, hasta cierto punto, de la versión utilizada por Rails. La primera se indica como sigue:

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

Esto instala la versión más reciente de la gema **uglifier** (que se encarga de comprimir los archivos de la cadena de procesos conectados) siempre que ésta sea mayor o igual a la versión **1.3.0**—aún si ésta es, digamos, la versión **7.2**. El segundo método se muestra a continuación:

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

Esto instala la versión más reciente de la gema **coffee-rails** siempre que ésta sea mayor que la versión **4.0.0** y sea menor que la versión **4.1**. En otras palabras, la notación >= siempre instala la gema más reciente, mientras que la notación `~> 4.0.0` sólo instala gemas actualizadas en liberaciones menores (es decir, de **4.0.0** a **4.0.1**), pero no liberaciones mayores (por ejemplo, de **4.0** a **4.1**). Desafortunadamente, la experiencia muestra que aún liberaciones menores pueden causar problemas, por lo que en el *Tutorial de Ruby on Rails* tomaremos precauciones extremas al incluír el número exacto de versión para todas las gemas. Usted puede utilizar la versión más reciente de cualquier gema, incluyéndola mediante la construcción ~> en el archivo **Gemfile** (lo cual recomiendo para los usuarios más avanzados), pero queda advertido que esto puede causar que los ejemplos y ejercicios del tutorial se comporten de forma impredecible.

Convertir el archivo **Gemfile** del Listado 1.4 para que utilice las versiones de gema exactas produce el código que se muestra en el Listado 1.5. Observe que hemos aprovechado la oportunidad para que la gema `sqlite3` sea incluída sólo en ambientes de desarrollo o pruebas (Sección 7.1.1), lo cual evita conflictos potenciales con la base de datos utilizada por Heroku (Sección 1.5).

Listado 1.5: Un archivo `Gemfile` con una versión explícita para cada gema de Ruby.

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

gem 'rails',                '4.2.2'
gem 'sass-rails',           '5.0.2'
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
```

Una vez que ha introducido el contenido del Listado 1.5 en el archivo `Gemfile` de la aplicación, instale las gemas utilizando `bundle install`:[9]

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

El comando `bundle install` puede tomar unos momentos en ejecutarse, pero cuando termina, nuestra aplicación estará lista para correr.

1.3.2 `rails server`

Gracias a la ejecución de `rails new` en la Sección 1.3 y a `bundle install` en la Sección 1.3.1, tenemos ya una aplicación que podemos ejecutar—pero ¿cómo? Afortunadamente, Rails viene con un programa de línea de comandos

[9] Como observamos en la Tabla 3.1, usted puede omitir `install`, puesto que el comando `bundle` mismo, es un alias de `bundle install`.

o *script*, que ejecuta un servidor web *local* para auxiliarnos en el desarrollo de nuestra aplicación. El comando exacto depende del ambiente en el que usted lo esté utilizando: en un sistema local, ejecute solamente `rails server` (Listado 1.6), mientras que en Cloud9 necesita proporcionar una *dirección IP* adicional y un *número de puerto* para indicarle al servidor Rails la dirección que puede utilizar para hacer visible la aplicación al mundo exterior (Listado 1.7).[10] (Cloud9 utiliza las *variables de ambiente* especiales: `$IP` y `$PORT` para asignar la dirección IP y el número de puerto dinámicamente. Si usted desea ver los valores de estas variables, escriba `echo $IP` o `echo $PORT` en la línea de comandos.) Si su sistema se queja de la falta de un motor para ejecutar JavaScript, visite la página execjs en GitHub para obtener una lista de candidatos. En particular le recomiendo instalar Node.js.

Listado 1.6: Ejecutando el servidor Rails en una máquina local.

```
$ 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
```

Listado 1.7: Ejecutando el servidor Rails en el IDE en la nube.

```
$ 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
```

Sin importar la opción que usted elija, le recomiendo que ejecute el comando `rails server` en una segunda pestaña de la terminal de forma que pueda seguir ejecutando comandos en la primera pestaña, como se muestra en

[10] Normalmente, los sitios web corren sobre el puerto 80, pero usualmente esto requiere privilegios especiales, por lo que es una convención utilizar para el servidor de desarrollo, un número de puerto mayor que tenga menos restricciones.

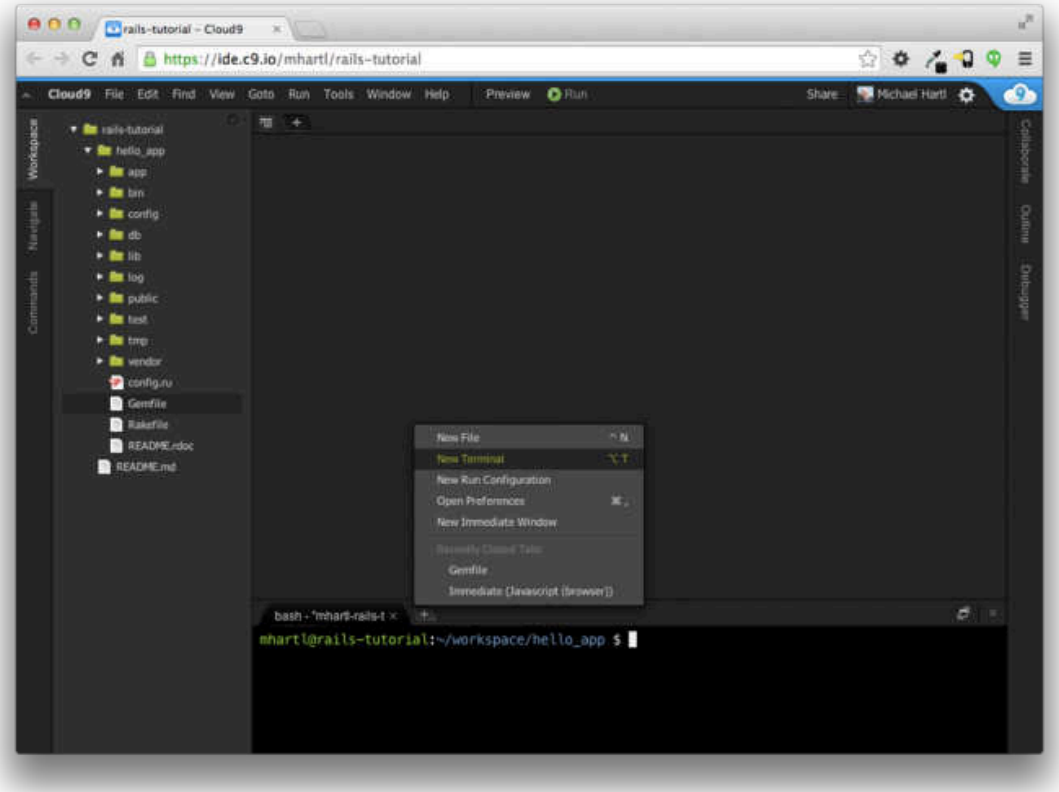

Figura 1.6: Abriendo una nueva pestaña en la terminal.

las Figuras 1.6 y 1.7. (Si usted ya inició el servidor en su primera pestaña, presione Ctrl-C para deternerlo.)[11] En un servidor local, escriba en su navegador la dirección http://localhost:3000/; en el IDE en la nube, diríjase a Compartir y dé click en la dirección de la aplicación para abrirla (Figura 1.8). En cualquier caso, el resultado debe verse similar al de la Figura 1.9.

Para ver información sobre la primera aplicación, ingrese en el enlace "Acerca del ambiente de su aplicación". Aunque los números de versión pueden variar, el resultado debe verse como en la Figura 1.10. Por supuesto, a la larga, no necesitamos la página de Rails por default, pero es bueno verla funcionar

[11] Aquí "C" se refiere a la tecla del teclado, no a la letra mayúscula, por lo que no es necesario utilizar la tecla de mayúsculas.

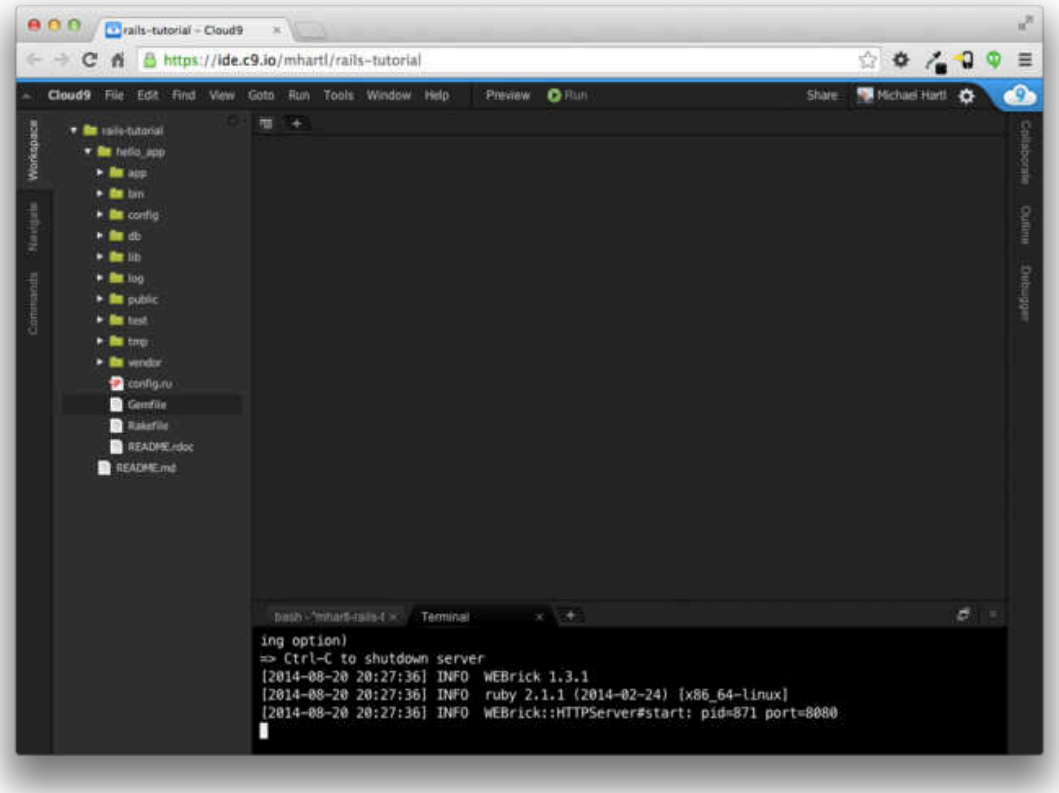

Figura 1.7: Ejecutando el servidor Rails en una pestaña separada.

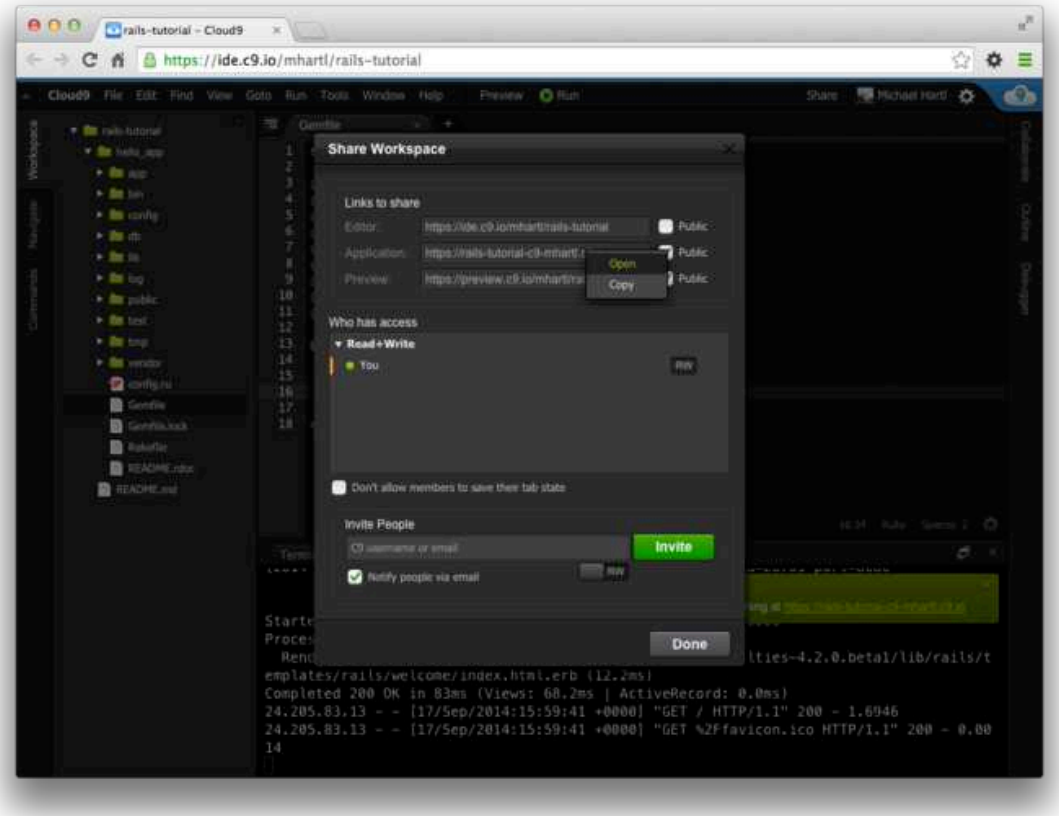

Figura 1.8: Compartiendo el servidor local que está ejecutándose en el espacio de trabajo de la nube.

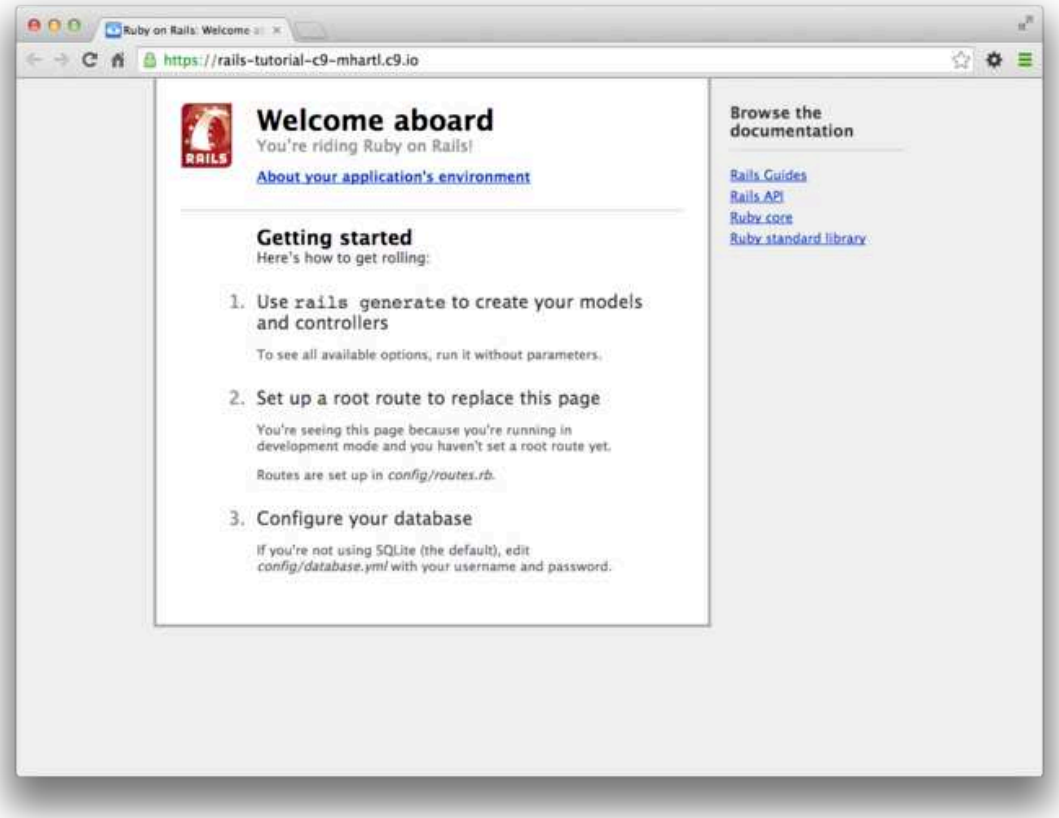

Figura 1.9: La página por default de Rails despachada por **`rails server`**.

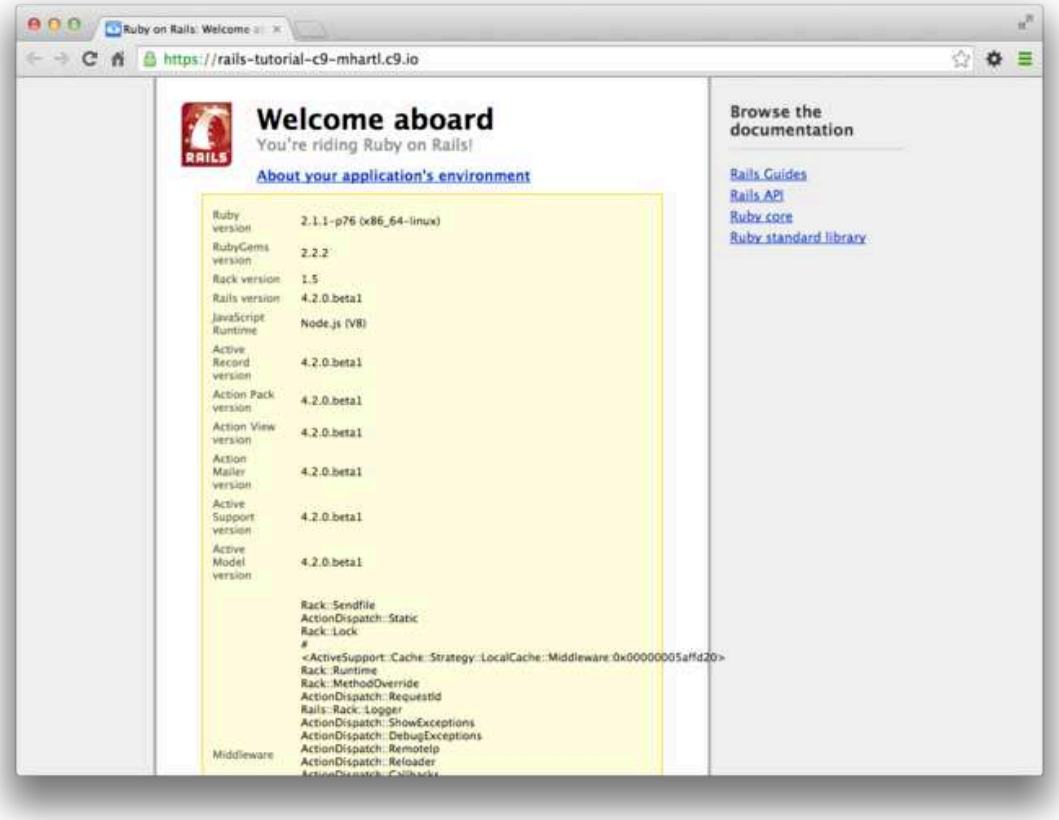

Figura 1.10: La página por default con el ambiente de la aplicación.

por ahora. Eliminaremos esta página (y la reemplazaremos por una página principal personalizada) en la Sección 1.3.4.

1.3.3 Modelo Vista-Controlador (MVC)

Aún en esta etapa temprana, es útil tener una vista general de cómo funcionan la aplicaciones de Rails (Figura 1.11). Puede que usted haya notado que la aplicación de Rails tiene una estructura estándar (Figura 1.4): un directorio de aplicación llamado `app/` con tres subdirectorios: `models`, `views`, y `controllers`. Esto es una sugerencia de que Rails sigue el patrón de arquitec-

tura denominado modelo vista-controlador (MVC), el cual separa la "lógica del dominio" (también llamada "lógica de negocio") de la lógica de presentación y entrada asociada con una interfaz de usuario gráfica (GUI). En el caso de aplicaciones web, la "lógica de dominio" típicamente consiste de modelos de datos para cosas como usuarios, artículos y productos, y la GUI es sólo una página en un navegador web.

Cuando interactuamos con una aplicación Rails, el navegador envía una *petición*, que es recibida por un servidor web y es pasada al *controlador* de Rails, el cual se encarga de realizar lo que sigue. En algunos casos, el controlador inmediatamente mostrará una *vista*, la cual es una plantilla que se convierte en HTML y es enviada de regreso al navegador. Comúnmente en los sitios dinámicos, el controlador interactúa con un *modelo*, que es un objeto Ruby que representa un elemento del sitio (tal como un usuario) y que está a cargo de comunicarse con la base de datos. Luego de invocar el modelo, el controlador genera la vista y regresa la página web completa al navegador como HTML.

Si esta discusión le parece un poco abstracta ahora, no se preocupe; nos referiremos en el futuro a esta sección frecuentemente. La Sección 1.3.4 muestra una primera aplicación tentativa de MVC, mientras que la Sección 2.2.2 incluye una discusión más detallada de MVC en el contexto de la aplicación de juguete. Finalmente, la aplicación de ejemplo utilizará todos los aspectos de MVC; revisaremos los controladores y las vistas empezando la Sección 3.2 y los modelos empezando la Sección 6.1, y veremos los tres trabajar juntos en la Sección 7.1.2.

1.3.4 ¡Hola mundo!

Como primera aplicación del marco de trabajo MVC, realizaremos un cambio mínimo a la primera aplicación agregando una *acción al controlador* para desplegar la cadena "hello, world!". (Aprenderemos más sobre acciones de controladores al empezar la Sección 2.2.2.) El resultado será reemplazar la página de Rails por default de la Figura 1.9 con la página "hello, world!" que es el objetivo de esta sección.

Como se deduce de su nombre, las acciones del controlador están definidas

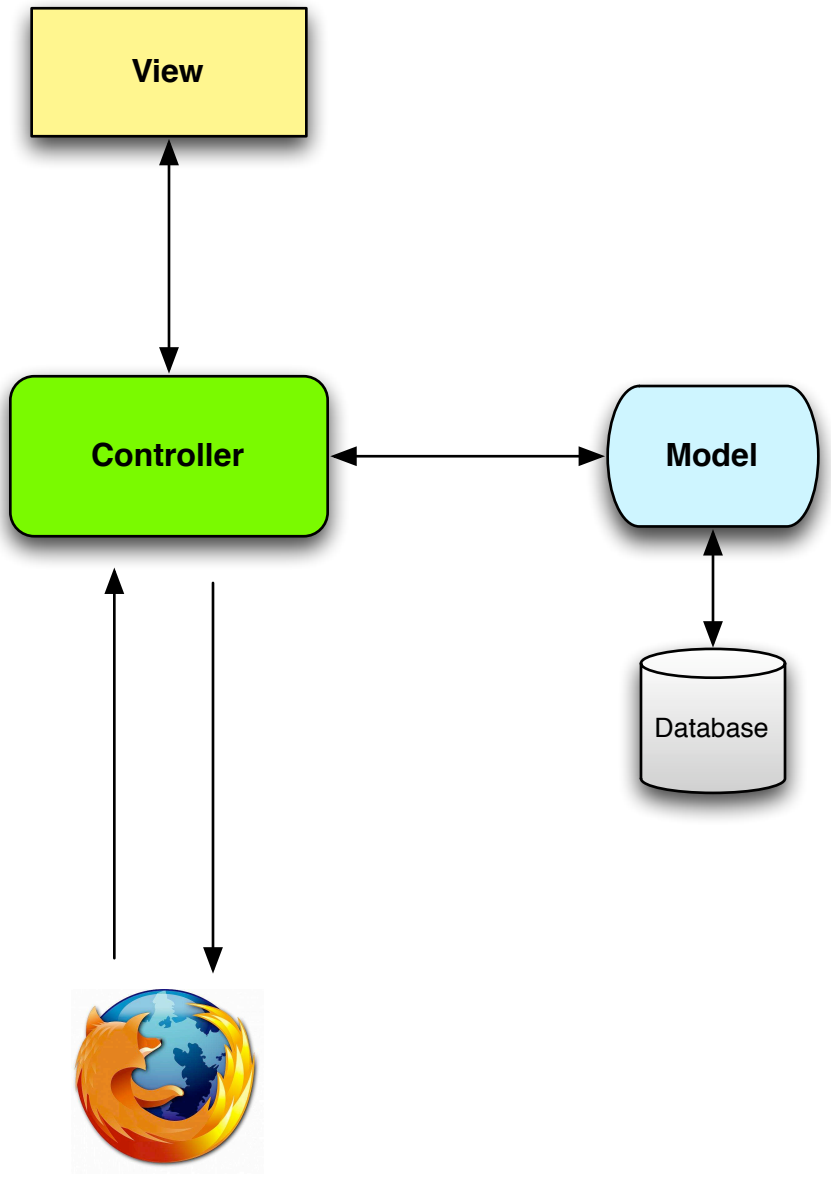

Figura 1.11: Una representación esquemática de la arquitectura del modelo vista-controlador (MVC).

dentro de un controlador. Nombraremos nuestra acción `hello` y la colocaremos en el controlador de la aplicación. De hecho, en este momento el controlador de la aplicación es el único controlador que tenemos, lo cual puede verificar ejecutando

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

para ver los controladores que tenemos en este momento. (Empezaremos a crear nuestros propios controladores en el Capítulo 2.) El Listado 1.8 muestra la definición resultante de `hello`, la cual utiliza la función `render` para regresar el texto "hello, world!". (No se preocupe por la sintaxis de Ruby en este momento; revisaremos este tema con mayor profundidad en el Capítulo 4.)

Listado 1.8: Agregando una acción `hello` al controlador de la aplicación.
app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

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

Habiendo definido una acción que regresa la cadena deseada, necesitamos decirle a Rails que utilice esta acción en vez de la página por default que vimos en la Figura 1.10. Para hacer esto, editaremos el *enrutador* de Rails, que se encarga de determinar a dónde enviaremos las peticiones que vienen del navegador. (He omitido el enrutador de la Figura 1.11 por simplicidad, pero lo revisaremos con mayor detalle a partir de la Sección 2.2.2.) En particular, queremos cambiar la página por default, por lo que cambiaremos la *ruta raíz* que es la que determina la página que es mostrada en la *URL raíz*. Como esta es la URL para una dirección como http://www.example.com/ (donde no viene nada luego de la primera diagonal que sigue al dominio), a menudo se hace referencia a la URL raíz como / ("diagonal") por brevedad.

Como se observa en el Listado 1.9, el archivo de rutas Rails (**config/routes.rb**) incluye una línea comentada que muestra cómo estructurar la ruta raíz. Aquí "welcome" es el nombre del controlador e "index" es la acción dentro de ese controlador. Para activar la ruta raíz, descomente esta línea removiendo el caracter de signo de número y luego reemplácelo con el código del Listado 1.10, el cual le indica a Rails que envíe la ruta raíz a la acción **hello** en el controlador de la aplicación. (Como observamos en la Sección 1.1.2, los puntos suspensivos verticales indican código que ha sido omitido y que no debe copiarse literalmente.)

Listado 1.9: La ruta raíz por default (comentada).
config/routes.rb

```
Rails.application.routes.draw do
  .
  .
  .
  # You can have the root of your site routed with "root"
  # root 'welcome#index'
  .
  .
  .
end
```

Listado 1.10: Estableciendo la ruta raíz.
config/routes.rb

```
Rails.application.routes.draw do
  .
  .
  .
  # You can have the root of your site routed with "root"
  root 'application#hello'
  .
  .
  .
end
```

Con el código de los Listados 1.8 y 1.10, la ruta raíz regresa "hello, world!" como vemos en la Figura 1.12.

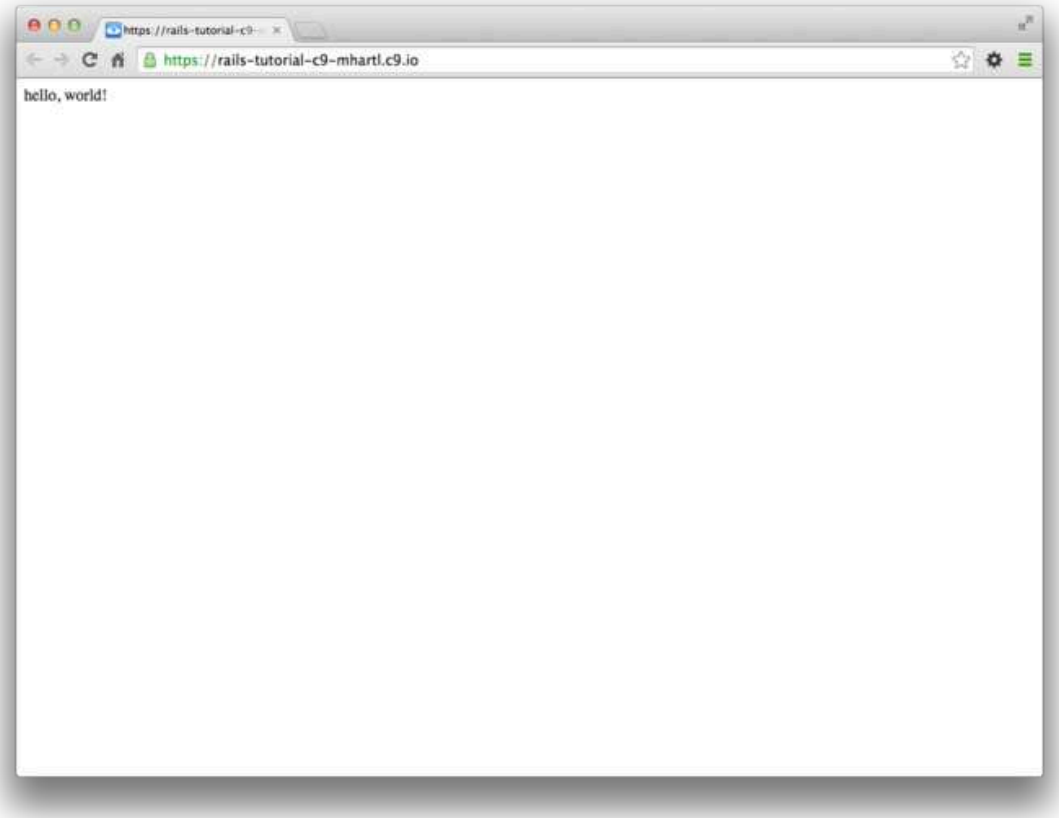

Figura 1.12: Visualizando “hello, world!” en el navegador.

1.4 Control de versiones con Git

Ahora que tenemos una aplicación de Rails nueva y funcionando, dedicaremos un momento para dar un paso que, técnicamente es opcional, pero que bajo el punto de vista de los desarrolladores de software experimentados es prácticamente esencial: poner el código fuente de nuestra aplicación bajo un *control de versiones*. Los sistemas de control de versiones nos permiten dar seguimiento a los cambios que se realicen al código de nuestro proyecto, colaborar más fácilmente, y dar marcha atrás a cualquier error inadvertido (como por ejemplo borrar archivos de forma accidental). Saber cómo utilizar un sistema de control de versiones es una habilidad requerida para cualquier desarrollador de software profesional.

Existen muchas opciones para el control de versiones, pero la comunidad Rails se ha estandarizado ampliamente en Git, un sistema de control de versiones distribuído originalmente desarrollado por Linus Torvalds para hospedar el kernel de Linux. Git es un tema amplio, y sólo estaremos rasgando la superficie en este libro, pero hay muchos recursos en línea buenos y gratuitos; especialmente le recomiendo Empezando con Bitbucket para un breve resumen y *Pro Git* de Scott Chacon como libro introductorio. Poner su código fuente bajo control de versiones con Git es *ampliamente* recomendado, no sólo porque es casi una práctica universal en el mundo Rails, sino porque le permitirá respaldar y compartir su código más fácilmente (Sección 1.4.3) así como desplegar su aplicación aquí mismo en el primer capítulo (Sección 1.5).

1.4.1 Instalación y preparación

El IDE en la nube recomendado en la Sección 1.2.1 incluye Git por default, por lo que no es necesaria ninguna instalación. En cualquier otro caso, Install-Rails.com (Sección 1.2) incluye instrucciones para instalar Git en su sistema.

Preparación inicial del sistema

Antes de utilizar Git, debe realizar una serie de pasos de iniciales. Estas son configuraciones de *sistema*, lo que significa que sólo tiene que realizarlas una

vez por computadora:

```
$ 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
```

Observe que el nombre y la dirección de correo electrónico que usted utilice en su configuración de Git serán visibles desde cualquier repositorio que usted haga público. (Del listado anterior, únicamente las primeras dos líneas son estrictamente necesarias. La tercer línea se incluye para asegurar la compatibilidad con versiones futuras de Git. La cuarta línea, también opcional, se incluye para que pueda utilizar la abreviatura `co` en vez del comando `checkout` completo. Para mayor compatibilidad con sistemas que no tienen `co` configurado, este tutorial utilizará el comando completo `checkout` (pero en la vida real, casi siempre utilizo `git co`.)

Configuración inicial del repositorio

Ahora realizaremos algunos pasos que es necesario ejecutar cada vez que usted cree un nuevo *repositorio* (algunas veces llamado *repo* por brevedad). Para empezar, navegue al directorio raíz de la aplicación e inicialice el repositorio:

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

A continuación agregue todos los archivos del proyecto al repositorio utilizando `git add -A`:

```
$ git add -A
```

Este comando agrega todos los archivos del directorio actual excepto aquellos cuyos nombres cumplan con los patrones del archivo especial `.gitignore`. El

comando **rails new** automáticamente genera un archivo **.gitignore** apropiado para un proyecto Rails, pero usted puede agregar patrones también.[12]

Los archivos agregados son inicialmente colocados en un *área de preparación*, la cual contiene cambios pendientes en su proyecto. Usted puede ver qué archivos están en esta área utilizando el comando **status**:

```
$ 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
  .
  .
  .
```

(El resultado es grande, por lo que he utilizado los puntos suspensivos verticales para indicar la salida omitida.)

Para indicarle a Git que usted desea guardar los cambios, utilice el comando **commit**:

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

La opción **-m** le permite agregar un mensaje a la confirmación; si usted omite la opción **-m**, Git abrirá el editor de texto por default del sistema y le solicitará que escriba un mensaje ahí. (Todos los ejemplos de este libro utilizan la opción **-m**.)

[12] Aunque en el tutorial principal no necesitaremos editarlo, un ejemplo para agregar una regla al archivo **.gitignore** se muestra en la Sección 3.7.3, la cual forma parte de la configuración de pruebas avanzadas de la Sección 3.7.

Es importante observar que las confirmaciones de Git son *locales*, guardadas únicamente en la máquina en la que se ejecuta el `commit`. Veremos cómo empujar los cambios hacia un repositorio remoto (mediante `git push`) en la Sección 1.4.4.

Por cierto, puede ver una lista de sus mensajes de confirmación usando el comando `git log`:

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

    Initialize repository
```

Dependiendo de qué tan larga es la historia de confirmaciones de su repositorio, puede ser necesario que usted presione la tecla `q` para terminar la ejecución del comando.

1.4.2 ¿Qué beneficios le proporciona a usted Git?

Si usted nunca ha utilizado un sistema de control de versiones, puede ser que en este momento no quede claro el beneficio que le proporciona, por lo que le daré un ejemplo. Suponga que usted ha realizado algunos cambios accidentales, tales como (¡ah!) borrar el directorio `app/controllers/` que es indispensable en el proyecto.

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

Aquí estamos utilizando el comando `ls` de Unix para enlistar el contenido del directorio `app/controllers/` y el comando `rm` para eliminarlo (Tabla 1.1). La opción `-rf` significa “forzar recursividad”, lo cual elimina de forma recursiva todos los archivos, directorios, subdirectorios y demás, sin preguntar explícitamente por cada borrado.

Revisemos el status para ver qué ha cambiado:

```
$ 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")
```

Aquí vemos que un archivo ha sido borrado, pero los cambios están únicamente en el "árbol de trabajo"; no han sido confirmados aún. Esto significa que podemos deshacer estos cambios mediante el comando **`checkout`** con la opción **`-f`** para forzar la sobreescritura de los cambios actuales:

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

Los directorios y archivos faltantes están de regreso. ¡Qué alivio!

1.4.3 Bitbucket

Ahora que hemos puesto nuestro proyecto bajo el control de versiones usando Git, es hora de subir nuestro código a Bitbucket, un sitio optimizado para hospedar y compartir repositorios Git. (Ediciones previas de este tutorial utilizaban GitHub; vea el Recuadro 1.4 para conocer las razones del cambio.) Poner una copia de su repositorio Git en Bitbucket tiene dos propósitos: es un respaldo completo de su código (incluyendo el historial completo de confirmaciones) y permitir cualquier colaboración futura con mucha más facilidad.

Recuadro 1.4. GitHub y Bitbucket

Por mucho los dos sitios más populares para alojar repositorios Git son GitHub y Bitbucket. Los dos servicios comparten varias similitudes: ambos permiten hospedar repositorios Git y colaborar en ellos, así como ofrecer formas convenientes de navegar y buscar en los repositorios. Las diferencias importantes (bajo la perspectiva de este tutorial) son que GitHub ofrece un número ilimitado de repositorios gratuitos (con colaboración) para proyectos "open-source", en tanto que realiza cargos para repositorios privados, mientras que Bitbucket permite un número ilimitado de repositorios privados gratuitos, en tanto que realiza cargos si el número de colaboradores excede de cierta cantidad. Qué servicio utilice usted para un repositorio en particular, depende de sus necesidades específicas.

Ediciones previas de este libro utilizaban GitHub por su énfasis en el soporte de código open-source, pero las crecientes preocupaciones sobre seguridad hacen que recomiende que *todos* los repositorios de aplicaciones web sean privados por default. El punto es que los repositorios de aplicaciones web pueden contener información sensible, tal como llaves criptográficas y contraseñas, lo cual puede utilizarse para comprometer la seguridad de un sitio que ejecute ese código. Es posible, por supuesto, encargarnos de que esta información sea manejada de forma segura (haciendo que Git la ignore, por ejemplo), pero esto tiende a errores y requiere mayor pericia.

Sucede que, la aplicación de ejemplo creada en este tutorial es segura de exponerse en la red, pero es peligroso confiar en este hecho en general. Por lo que, para estar lo más seguros posible, seremos precavidos y utilizaremos repositorios privados por default. Puesto que GitHub realiza cargos por repositorios privados mientras que Bitbucket ofrece un número ilimitado de forma gratuita, para nuestros propósitos Bitbucket es una mejor opción que GitHub.

Empezar a utilizar Bitbucket es simple:

1. Regístrese para obtener una cuenta de Bitbucket si es que aún no tiene una.

2. Copie su *llave pública* al portapapeles. Como se indica en el Listado 1.11, los usuarios del IDE en la nube pueden ver su llave pública usando el comando `cat`, el cual pueden seleccionar y copiar. Si usted está utilizando su propio sistema y no visualiza ningún resultado cuando ejecuta el comando del Listado 1.11, siga las instrucciones de cómo instalar una llave pública en su cuenta de Bitbucket.

3. Agregue su llave pública a Bitbucket dando click en la imagen del avatar en la esquina superior derecha y seleccione "Administrar cuenta" y luego "llaves SSH" (Figura 1.13).

Listado 1.11: Visualizando la llave pública mediante `cat`.

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

Una vez que usted ha agregado su llave pública, presione "Crear" para crear un nuevo repositorio, como se muestra en la Figura 1.14. Al completar el formulario del proyecto, recuerde habilitar la opción "Este es un repositorio privado."Luego de presionar "Crear repositorio", siga las instrucciones debajo de "Línea de comandos > Tengo un proyecto existente", que debe verse similar al Listado 1.12. (Si no se parece al Listado 1.12, puede ser porque la llave pública no fue agregada correctamente, en cuyo caso le sugiero que realice nuevamente ese paso.) Cuando suba al repositorio, responda sí en caso de que se le presente la pregunta "¿Está usted seguro de que quiere continuar con la conexión (sí/no)?"

Listado 1.12: Agregando Bitbucket y subiendo al repositorio.

```
$ git remote add origin git@bitbucket.org:<username>/hello_app.git
$ git push -u origin --all # pushes up the repo and its refs for the first time
```

Los comandos del Listado 1.12 primero le indican a Git que usted quiere agregar Bitbucket como el *origen* de su repositorio, y luego sube su repositorio al origen remoto. (No se preocupe acerca de lo que la opción `-u` hace; si

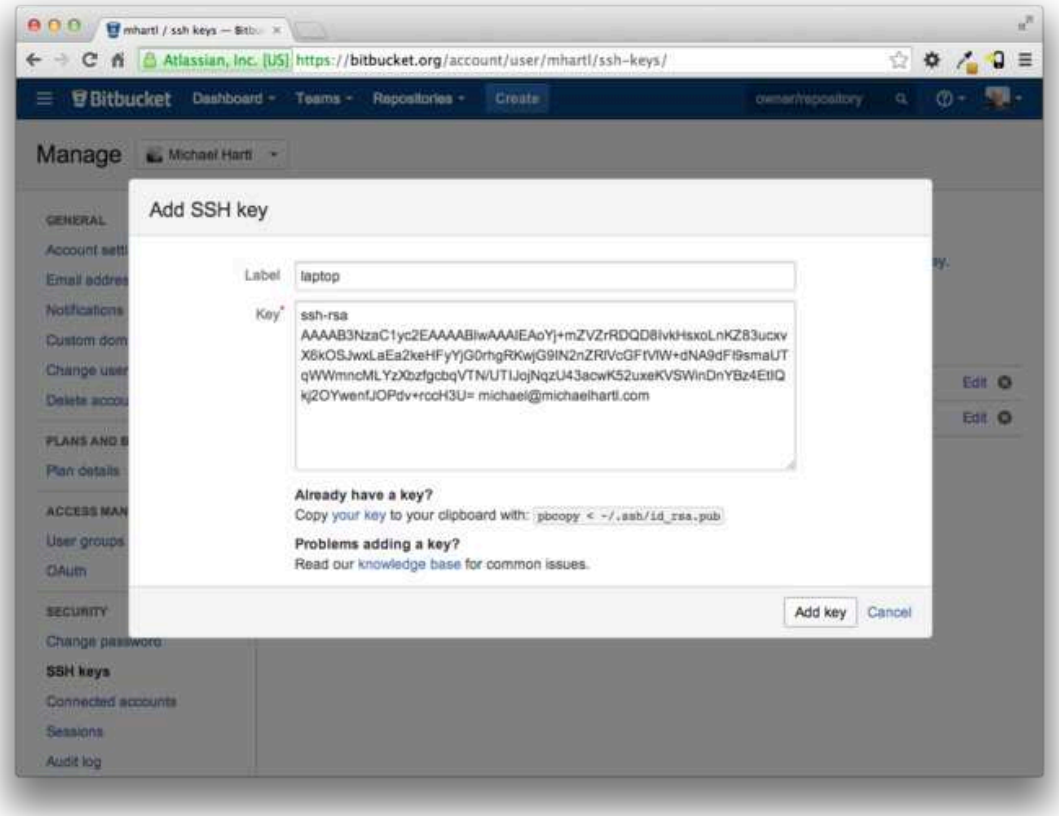

Figura 1.13: Agregando la llave pública SSH.

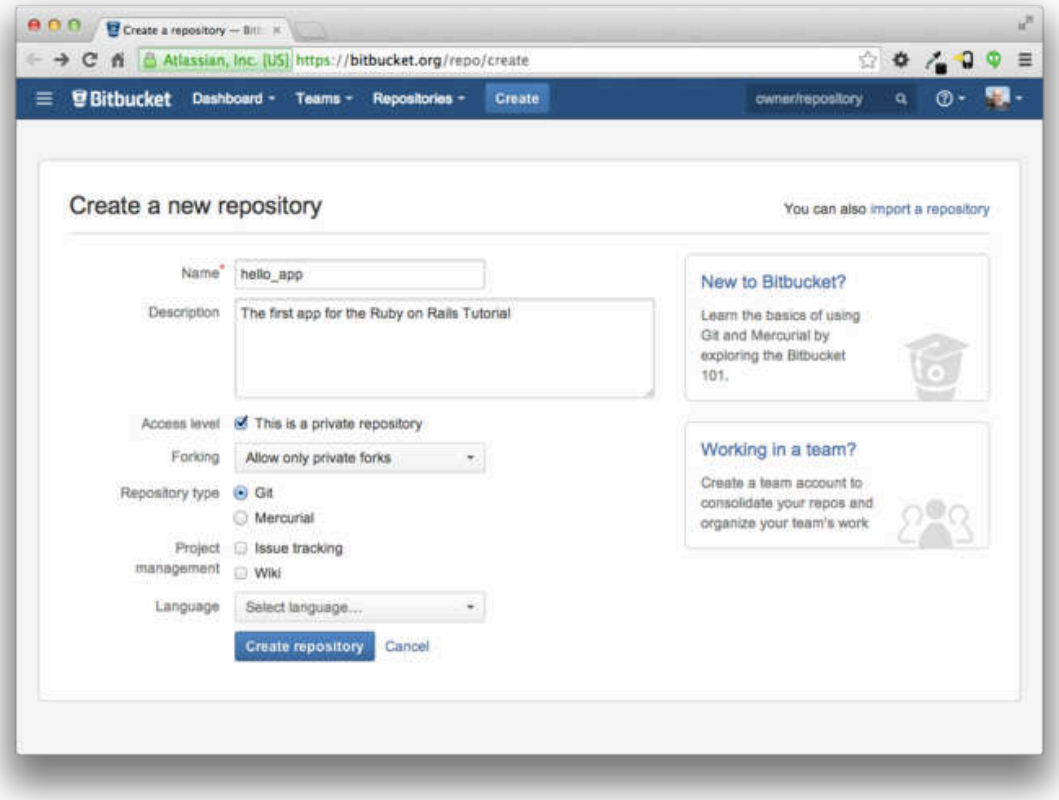

Figura 1.14: Creando el primer repositorio para una aplicación en Bitbucket.

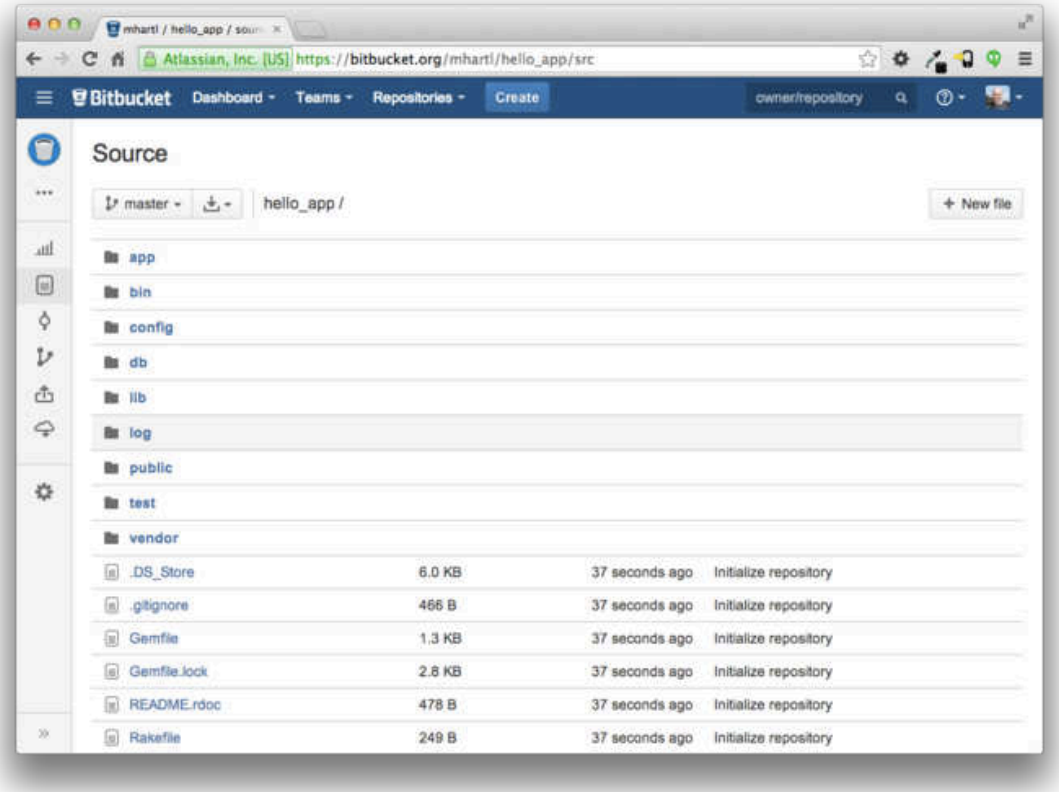

Figura 1.15: Una página de repositorio en Bitbucket.

tiene curiosidad, realice una búsqueda en la web de "git set upstream".) Por supuesto, debe reemplazar `<username>` con su nombre de usuario real. Por ejemplo, el comando que yo ejecuté fue

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

El resultado es una página en Bitbucket para el repositorio hello_app, con navegación de archivos, historial completo de confirmaciones y muchas otras bondades (Figura 1.15).

1.4.4 Ramas, edición, confirmación e integración

Si usted ha seguido los pasos indicados en la Sección 1.4.3, puede haberse percatado de que Bitbucket no detecta automáticamente el archivo `README.rdoc` de nuestro repositorio, y por tanto, se queja de que la página principal de nuestro repositorio no tiene un archivo README presente (Figura 1.16). Esto es una indicación de que el formato `rdoc` no es suficientemente común como para que Bitbucket lo entienda de forma automática, y de hecho prácticamente cualquier otro desarrollador y yo preferimos utilizar *Markdown*. En esta sección cambiaremos el archivo `README.rdoc` por `README.md`, mientras que aprovechamos la oportunidad de agregarle algo de contenido específico del Tutorial de Rails. En el proceso, veremos un primer ejemplo de los flujos de ramas, edición, confirmación e integración que recomiendo utilizar con Git.[13]

Rama

Git es increíblemente bueno creando *ramas*, las cuales son copias de un repositorio donde podemos realizar (quizá de forma experimental) cambios sin modificar los archivos originales. En la mayoría de los casos, el repositorio original es la rama *master*, y podemos crear una nueva rama para un tópico usando `checkout` junto con la opción `-b`:

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

Aquí el segundo comando, `git branch`, sólo enlista todas las ramas locales y el asterisco `*` indica en qué rama nos encontramos actualmente. Observe que `git checkout -b modify-README` no sólo crea una nueva rama sino que nos lleva a ella, como nos indica el asterisco junto a `modify-README`. (Si usted configuró el alias `co` en la Sección 1.4, puede utilizar el comando `git co -b modify-README` en lugar de la versión más larga.)

[13] Eche un vistazo a Atlassian's SourceTree app si desea una herramienta gráfica para visualizar los repositorios Git.

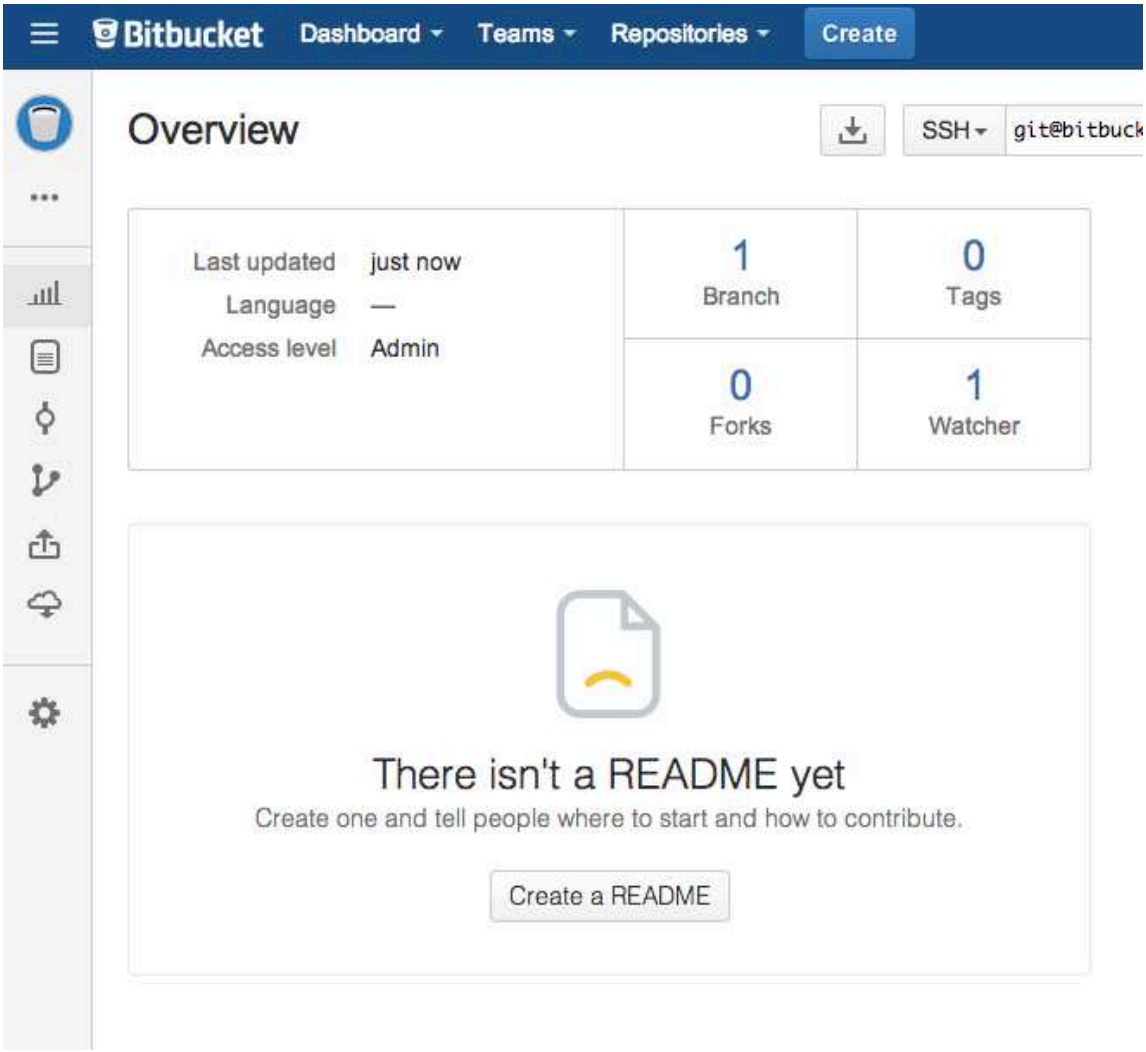

Figura 1.16: El mensaje de Bitbucket cuando falta el archivo README.

La verdadera importancia de crear ramas se hace clara cuando trabajamos en un proyecto con múltiples desarrolladores,[14] pero las ramas son útiles aún para seguir un tutorial como éste, con un solo desarrollador. En particular, la rama principal está aislada de cualquier cambio que realicemos en la rama del tópico, por lo que aún si descomponemos *realmente* las cosas siempre podemos abandonar los cambios cambiándonos a la rama principal y borrando la rama del tópico. Veremos cómo realizar esto al final de la sección.

Por cierto, para un cambio tan pequeño como éste normalmente no me tomaría la molestia de crear una nueva rama, pero en el contexto actual, representa nuestra primera oportunidad para empezar a fomentar buenos hábitos.

Edición

Luego de crear la rama del tópico, la editaremos para hacerla un poco más descriptiva. En lo personal, prefiero utilizar para este propósito el lenguaje de marcas Markdown que el lenguaje por default RDoc, y si usted utiliza la extensión `.md` entonces Bitbucket automáticamente le dará un formato agradable. Por tanto, utilizaremos la versión de Git del comando Unix `mv` (move) para renombrar el archivo:

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

Luego editaremos el archivo `README.md` agregándole el contenido del Listado 1.13.

Listado 1.13: El nuevo archivo `README.md`.

```
# Ruby on Rails Tutorial: "hello, world!"

This is the first application for the
[*Ruby on Rails Tutorial*](http://www.railstutorial.org/)
by [Michael Hartl](http://www.michaelhartl.com/).
```

[14] Vea el capítulo Ramificaciones en Git de *Pro Git* para mayor detalle.

Confirmación

Con los cambios realizados, podemos echar un vistazo al status de nuestra rama:

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

En este momento, podemos utilizar **`git add -A`** como en la Sección 1.4.1, pero **`git commit`** proporciona la opción **`-a`** como abreviatura para el (muy común) caso de confirmar todas las modificaciones existentes en los archivos (o archivos creados mediante **`git mv`**, que no cuentan como archivos nuevos para 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
```

Tenga cuidado de utilizar adecuadamente la opción **`-a`**; si usted ha agregado archivos nuevos al proyecto después de la última confirmación, es necesario notificarle a Git de su existencia primero, mediante el comando **`git add -A`**.

Observe que escribimos el mensaje de la confirmación en tiempo *presente* (y, técnicamente hablando, en modo imperativo). Git modela las confirmaciones como una serie de parches, y en este contexto tiene sentido describir lo que cada confirmación *hace*, en vez de lo que hizo. Más aún, este uso concuerda con los mensajes de confirmación generados por los mismos comandos de Git. Revise el artículo "Nuevos estilos de confirmación" para mayor información.

Integración

Ahora que hemos terminado de realizar nuestros cambios, estamos listos para *integrar* los resultados con nuestra rama principal:

```
$ 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
```

Observe que la salida de Git a menudo incluye cosas como **`34f06b7`**, las cuales están relacionadas con la representación interna que Git hace de los repositorios. Los resultados exactos que usted obtenga diferirán de éstos en detalle, pero en esencia coinciden con la salida que se mostró anteriormente.

Luego de que haya usted integrado sus cambios, puede depurar sus ramas borrando la rama de tópico que ya no utiliza mediante **`git branch -d`**:

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

Este paso es opcional, y de hecho es muy común dejar las ramas de tópico intactas. De esta forma usted puede ir y venir entre ellas y la rama principal, integrando cambios cada vez que alcance un punto en el que sea natural detenerse.

Como mencionamos antes, también es posible abandonar los cambios de su rama, en este caso con **`git branch -D`**:

```
# For illustration only; don't do this unless you mess up a branch
$ git checkout -b topic-branch
$ <really screw up the branch>
$ git add -A
$ git commit -a -m "Major screw up"
$ git checkout master
$ git branch -D topic-branch
```

A diferencia de la opción `-d`, la opción `-D` borrará la rama aún si no hemos integrado nuestros cambios.

Subir cambios

Ahora que hemos actualizado el archivo `README`, podemos subir nuestros cambios a Bitbucket para ver los resultados. Puesto que ya hemos realizado esta tarea por primera vez (Sección 1.4.3), en la mayoría de los sistemas podemos omitir `origin master`, y simplemente ejecutar `git push`:

```
$ git push
```

Como prometimos en la Sección 1.4.4, Bitbucket le da formato al nuevo archivo que usa Markdown (Figura 1.17).

1.5 Desplegando

Aún en esta etapa temprana, vamos a realizar nuestro despliegue de la aplicación de Rails (casi vacía) en producción. Este paso es opcional, pero hacerlo tempranamente y a menudo nos permite detectar problemas de despliegue a tiempo en nuestro ciclo de desarrollo. La otra opción es desplegar luego de un esfuerzo laborioso realizado únicamente en el ambiente de desarrollo, lo cual a menudo conlleva terribles dolores de cabeza para integrar cuando llega la hora de lanzar la aplicación.[15]

Desplegar aplicaciones Rails solía ser doloroso, pero el ecosistema de desarrollo de Rails ha madurado rápidamente en los últimos años, y ahora hay varias opciones muy buenas. Éstas incluyen hospedaje compartido o servidores virtuales privados ejecutando Phusion Passenger (un módulo para servidores Apache y Nginx[16]), compañías que brindan servicio completo de despliegue

[15] Aunque esto no debería importar para las aplicaciones de ejemplo del *Tutorial de Rails*, si a usted le preocupa que accidentalmente se haga pública su aplicación cuando es demasiado pronto, existen varias opciones; revise la Sección 1.5.4 para ver una de ellas.

[16] Pronunciado “Engine X”.

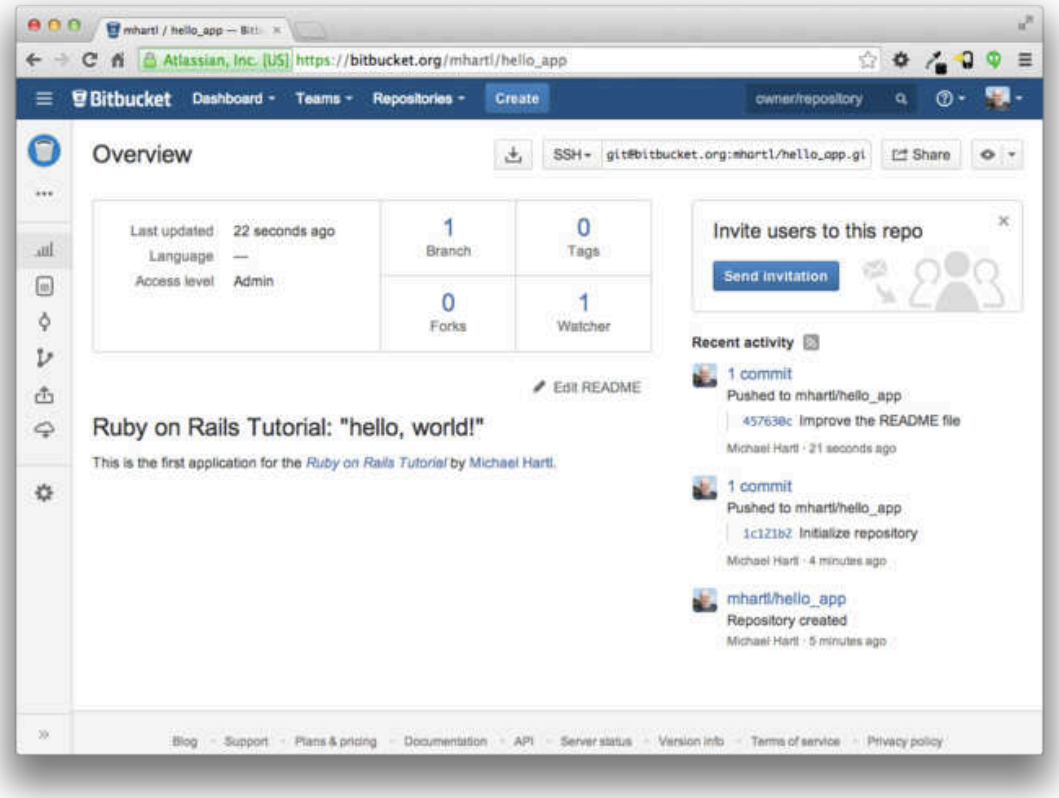

Figura 1.17: El archivo **README** mejorado con Markdown.

tales como Engine Yard y Rails Machine, y servicios de desarrollo en la nube tales como Engine Yard Cloud, Ninefold, y Heroku.

Mi opción de despliegue favorita para Rails es Heroku, que es una plataforma de hospedaje construída específicamente para desplegar Rails y otras aplicaciones web. Heroku hace que desplegar aplicaciones Rails sea extremadamente sencillo—siempre y cuando el código fuente esté bajo control de versiones con Git. (Esta es otra razón para seguir los pasos de configuración de Git indicados en la Sección 1.4 si es que aún no los realiza.) Adicionalmente, la versión gratuita del servicio de Heroku es más que suficiente para realizar los ejercicios de este libro. De hecho, las primeras dos ediciones de este tutorial estuvieron hospedadas gratuitamente en Heroku, quienes atendieron varios millones de peticiones sin cobrarme un centavo.

El resto de esta sección está dedicado a desplegar nuestra primera aplicación en Heroku. Algunos conceptos son un poco avanzados, por lo que no se preocupe por entender todos los detalles; lo importante es que al final del proceso tengamos nuestra aplicación desplegada en internet.

1.5.1 Configuración de Heroku

Heroku utiliza la base de datos PostgreSQL (a menudo llamada "Postgres" como diminutivo), lo que significa que necesitamos agregar la gema `pg` al ambiente de producción para permitir a Rails comunicarse con Postgres:[17]

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

Observe que también agregamos la gema `rails_12factor`, la cual es utilizada por Heroku para despachar recursos estáticos tales como imágenes y hojas de estilo. Finalmente, asegúrese de incorporar los cambios realizados en

[17] En general, es buena idea que los ambientes de desarrollo y producción sean lo más similares posible, lo que incluye utilizar la misma base de datos, pero para los propósitos de este tutorial siempre utilizamos SQLite localmente y PostgreSQL en producción. Revise la Sección 3.1 para mayor información.

el Listado 1.5 evitando que la gema `sqlite3` sea incluída en un ambiente de producción, puesto que SQLite no tiene soporte en Heroku:

```
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
```

El archivo **`Gemfile`** resultante se muestra en el Listado 1.14.

Listado 1.14: Un archivo **`Gemfile`** con las gemas añadidas.

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

gem 'rails',         '4.2.2'
gem 'sass-rails',    '5.0.2'
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'
  gem 'rails_12factor', '0.0.2'
end
```

Para preparar el sistema para despliegue en producción, ejecutamos **`bundle install`** con una opción especial para evitar la instalación local de las gemas de producción (que en este caso son `pg` y `rails_12factor`):

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

Como las únicas gemas agregadas en el Listado 1.14 están restringidas al ambiente de producción, en este momento este comando no instala realmente ninguna gema adicional en el entorno local, pero se requiere actualizar el archivo **`Gemfile.lock`** con las nuevas gemas `pg` y `rails_12factor`. Podemos confirmar el cambio resultante como sigue:

```
$ git commit -a -m "Update Gemfile.lock for Heroku"
```

A continuación debemos crear y configurar una cuenta en Heroku. El primer paso es registrarse en Heroku. Luego revise si su sistema ya tiene el cliente de línea de comandos de Heroku instalado:

```
$ heroku version
```

Aquellos que estén utilizando el IDE en la nube deben revisar el número de versión de Heroku, lo cual indica que la línea de comandos (CLI, por sus siglas en inglés *Command Line Interface*) de **`heroku`** está disponible, pero en otros sistemas puede ser necesario instalarlo mediante el conjunto de Herramientas de Heroku.[18]

Una vez que ha verificado que la línea de comandos de Heroku está instalada, utilice el comando **`heroku`** para iniciar sesión y agregar su llave SSH:

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

Finalmente, utilice el comando **`heroku create`** para crear un espacio en los servidores de Heroku para colocar ahí la aplicación de ejemplo (Listado 1.15).

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

Listado 1.15: Creando una nueva aplicación en 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
```

El comando `heroku create` crea un nuevo subdominio sólo para nuestra aplicación, disponible para ser utilizado de inmediato. Aún no hay nada ahí, así que despleguemos.

1.5.2 Despliegue en Heroku, paso uno

Para desplegar la aplicación, el primer paso es utilizar Git para subir la rama principal a Heroku:

```
$ git push heroku master
```

(Usted puede ver algunos mensajes de advertencia, los cuales puede ignorar por ahora. Hablaremos más de ellos en la Sección 7.5.)

1.5.3 Despliegue en Heroku, paso dos

¡No hay paso dos! Ya terminamos. Para ver nuestra recién desplegada aplicación, visite la dirección que se indicó en la salida del comando `heroku create` (es decir, el Listado 1.15). (Si usted está trabajando en su equipo local en vez del IDE en la nube, puede utilizar también `heroku open`.) El resultado se muestra en la Figura 1.18. La página es idéntica a la de la Figura 1.12, sólo que ahora está ejecutándose en un ambiente de producción en internet.

1.5.4 Comandos de Heroku

Hay muchos comandos Heroku, y apenas veremos algunos en este libro. Dediquemos un momento a revisar uno de ellos para renombrar la aplicación como sigue:

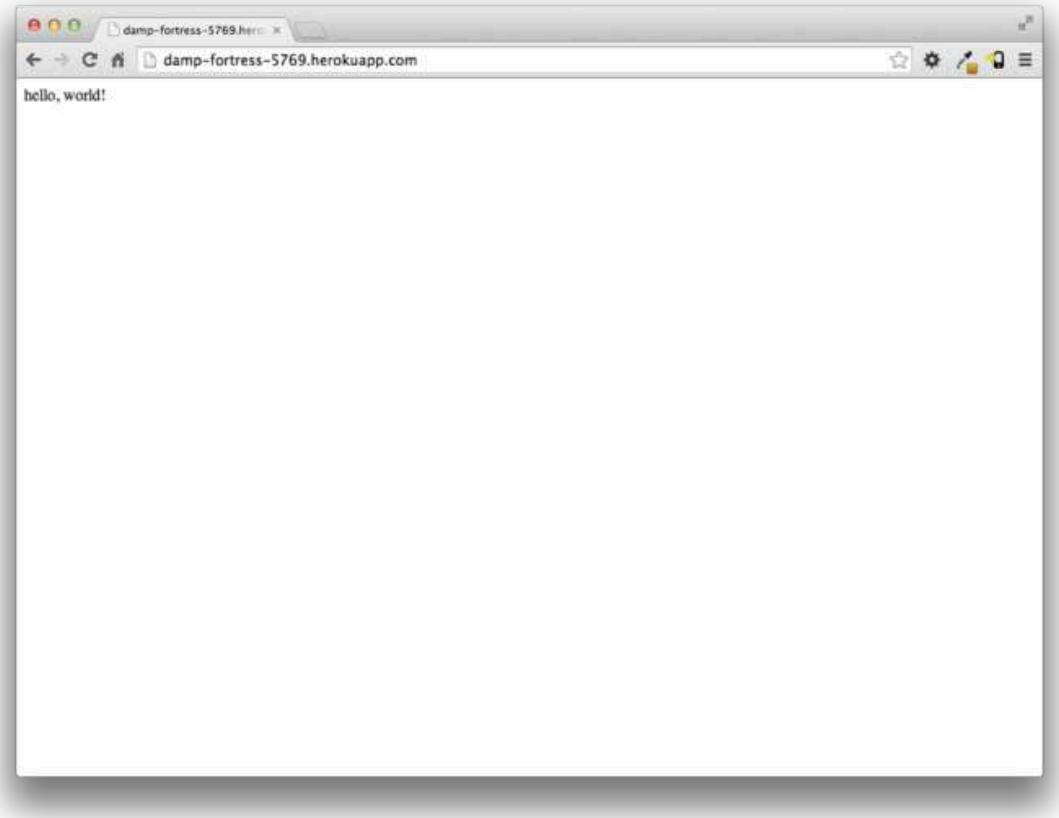

Figura 1.18: La primera aplicación del Tutorial de Rails corriendo en Heroku.

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

No utilice este nombre usted; ¡ya está ocupado por mí! De hecho, no es necesario que se tome la molestia de realizar este paso ahora; usar la dirección por default que le asignó Heroku está bien. Pero si usted desea renombrar su aplicación, puede encargarse de que sea razonablemente segura utilizando un subdominio aleatorio o poco claro, como el siguiente:

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

Con un dominio aleatorio como éste, alguien puede visitar su sitio sólo si usted le ha dado la dirección. (Por cierto, como un adelanto de lo grandiosa que es la compacidad de Ruby, aquí está el código que utilicé para generar los subdominios aleatorios:

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

Grandioso.)

Además de soportar subdominios, Heroku también soporta dominios personalizados. Revise la documentación de Heroku para mayor acerca de éste y otros temas de Heroku.

1.6 Conclusión

Hemos recorrido un largo camino en este capítulo: la instalación y configuración del ambiente de desarrollo, el control de versiones y el despliegue. En el próximo capítulo, construiremos sobre las bases del Capítulo 1 para crear una *aplicación de juguete* que tenga respaldo de base de datos, lo cual nos dará una probadita de lo que Rails puede hacer.

Si desea compartir su progreso hasta este momento, siéntase libre de enviar un tuit o actualizar su status en Facebook con algo como esto:

Estoy aprendiendo Ruby on Rails con el @railstutorial!
http://www.railstutorial.org/

También le recomiendo registrarse en la lista de correos del Tutorial de Rails[19], lo cual le asegura que recibirá actualizaciones prioritarias (y cupones exclusivos) relacionados con el *Tutorial de Ruby on Rails*.

1.6.1 Qué aprendimos en este capítulo

- Ruby on Rails es un marco de trabajo para desarrollo web escrito en el lenguaje de programación Ruby.
- Instalar Rails, generar una aplicación y editar los archivos resultantes es fácil utilizando un ambiente en la nube pre-configurado.
- Rails viene con una línea de comandos llamada `rails` que puede generar aplicaciones nuevas (`rails new`) y correr servidores locales (`rails server`).
- Agregamos una acción a un controlador y modificamos la ruta raíz para crear una aplicación "hola mundo".
- Nos protegimos contra la pérdida de datos al mismo tiempo que habilitamos la colaboración al poner el código fuente de nuestra aplicación bajo un control de versiones con Git y subiendo el código resultante a un repositorio privado en Bitbucket.
- Desplegamos nuestra aplicación en un ambiente de producción usando Heroku.

1.7 Ejercicios

Nota: El *Manual de Soluciones para los Ejercicios*, con soluciones para cada ejercicio del libro *Tutorial de Ruby on Rails*, se incluye de forma gratuita en

[19] http://www.railstutorial.org/#email

cada compra realizada en www.railstutorial.org.

1. Modifique el contenido de la acción `hello` del Listado 1.8 para leer "¡Hola, mundo!" en vez de "hello, world!". *Punto extra*: Muestre que Rails soporta caracteres no-ASCII incluyendo el signo para abrir una admiración, como en "¡Hola, mundo!" (Figura 1.19).[20]

2. Siguiendo el ejemplo de la acción `hello` del Listado 1.8, agregue una segunda acción llamada `goodbye` que muestre el texto "¡Adiós, mundo!". Edite el archivo de rutas del Listado 1.10 de forma que la ruta raíz se dirija a `goodbye` en vez de a `hello` (Figura 1.20).

[20] Su editor de texto puede mostrar un mensaje como "caracter multibyte inválido", pero esto no debe preocuparle. Usted puede buscar en Google el mensaje de error si quiere aprender cómo desaparecerlo.

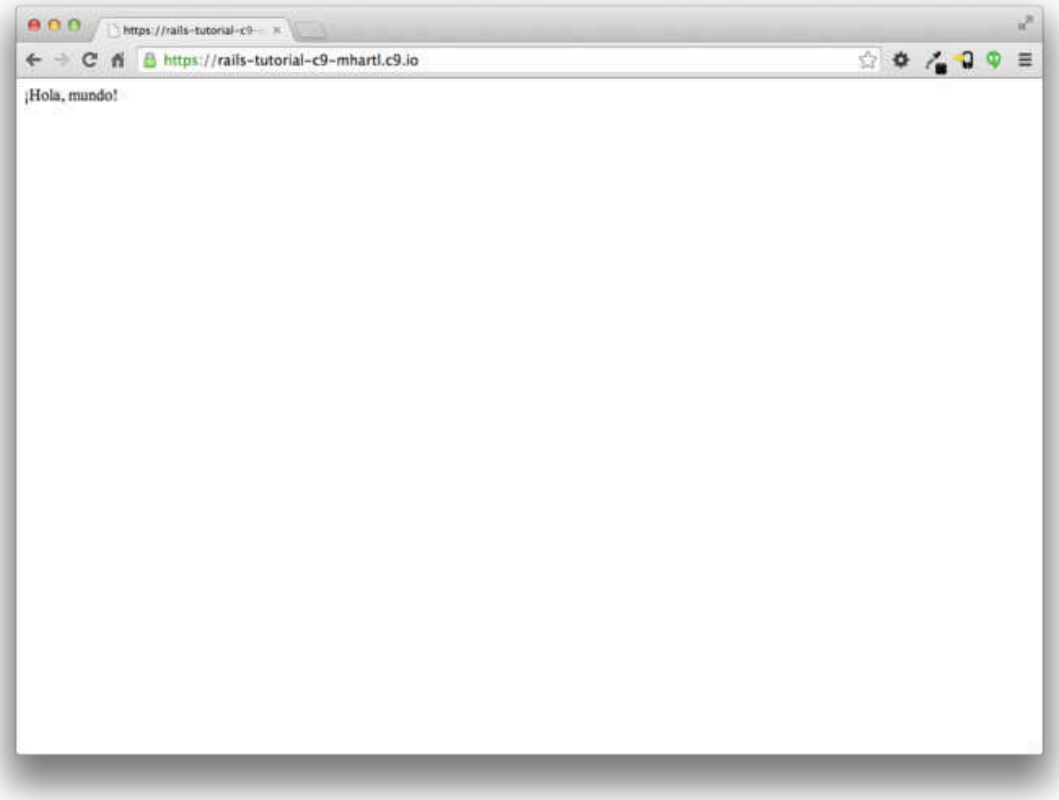

Figura 1.19: Cambiando la ruta raíz para regresar “¡Hola, mundo!”.

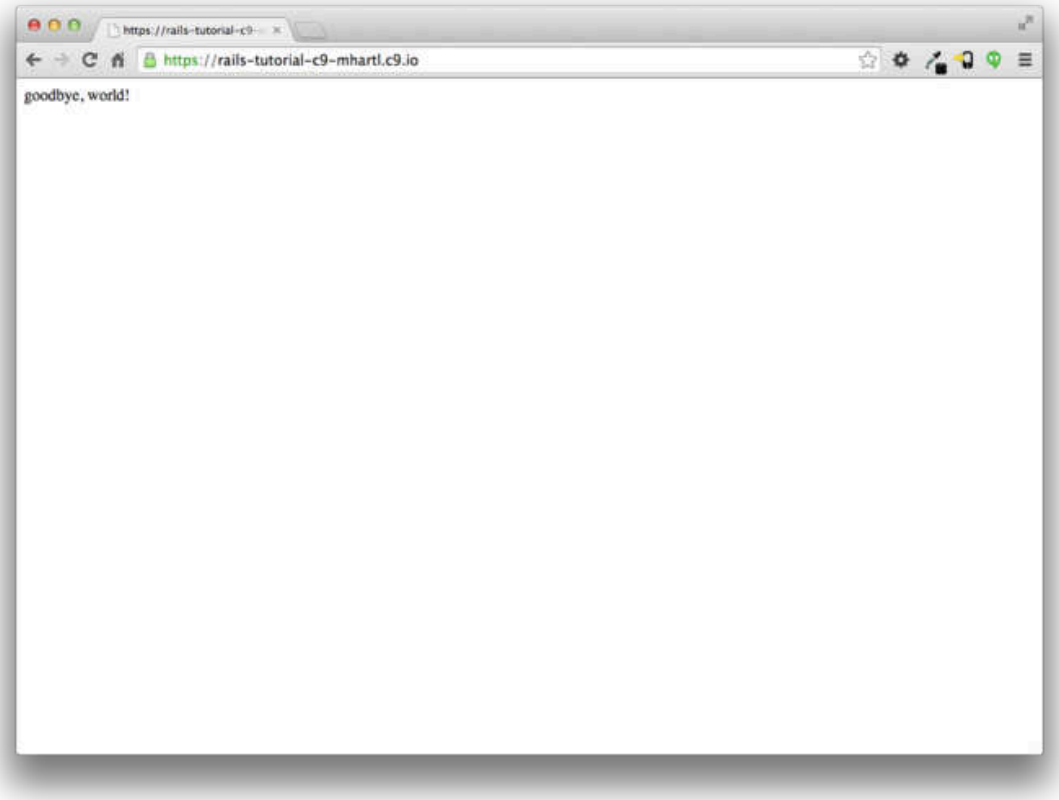

Figura 1.20: Cambiando la ruta raíz para regresar “¡Adiós, mundo!”.

Capítulo 2

Una aplicación de juguete

En este capítulo, desarrollaremos una aplicación de juguete para demostrar algunos de los poderes de Rails. El propósito es obtener una visión general de la programación con Ruby on Rails (y del desarrollo en general) al generar rápidamente una aplicación usando *generadores de estructuras*, los cuales crean una gran cantidad de funcionalidad automáticamente. Como se comentó en el recuadro 1.2, el resto del libro adoptará el enfoque opuesto, desarrollando una aplicación de ejemplo completa de forma incremental y explicando cada concepto nuevo conforme se presente, pero para una rápida visión general (y una pequeña gratificación instantánea) no hay como crear una estructura temporal. La aplicación de juguete resultante nos permitirá interactuar con ella a través de sus URLs, dándonos una idea de la estructura de una aplicación Rails, y al mismo tiempo constituye un primer ejemplo de la *arquitectura REST* favorecida por Rails.

Al igual que con la aplicación de ejemplo, la aplicación de juguete consistirá de *usuarios* y sus *microposts* (generando entonces una aplicación minimalista estilo Twitter). La funcionalidad estará totalmente subdesarrollada, y muchos de los pasos parecerán mágicos, pero no se preocupe: la aplicación de ejemplo completa desarrollará una aplicación similar desde cero empezando en el Capítulo 3, y le proporcionaré abundantes referencias al material en general. Mientras tanto, tenga paciencia y un poco de fé—el gran propósito de este tutorial es llevarlo *más allá* de lo superficial; del enfoque basado en la creación de estructuras temporales para lograr un entendimiento más profundo de Rails.

2.1 Planificando la aplicación

En esta sección, bosquejaremos nuestros planes para la aplicación de juguete. Como en la Sección 1.3, empezaremos generando un esqueleto de la aplicación utilizando el comando `rails new` con un número de versión específico de Rails:

```
$ cd ~/workspace
$ rails _4.2.2_ new toy_app
$ cd toy_app/
```

Si el comando anterior arroja un error como "Could not find 'railties'", significa que usted no tiene instalada la versión correcta de Rails, y debería verificar que ha ejecutado el comando indicado en el Listado 1.1 exactamente como está escrito. (Si está utilizando el IDE en la nube como se recomendó en la Sección 1.2.1, observe que esta segunda aplicación puede ser creada en el mismo espacio de trabajo que la primera. No es necesario crear un nuevo espacio de trabajo. Con la finalidad de visualizar los archivos que se generan, es posible que necesite dar click en el ícono de engrane ubicado en el área del navegador de archivos, y seleccionar "Refresh File Tree".)

A continuación, utilizaremos un editor de texto para actualizar el archivo `Gemfile` necesario para el Bundler, con los contenidos del Listado 2.1.

Listado 2.1: Un archivo `Gemfile` para la aplicación de juguete.

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

gem 'rails',        '4.2.2'
gem 'sass-rails',   '5.0.2'
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'
  gem 'rails_12factor', '0.0.2'
end
```

Observe que el Listado 2.1 es idéntico al Listado 1.14.

Al igual que en la Sección 1.5.1, instalaremos las gemas locales mientras evitamos la instalación de las gemas de producción utilizando la opción `--without production`:

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

Finalmente, pondremos la aplicación de juguete bajo el control de versiones con Git:

```
$ git init
$ git add -A
$ git commit -m "Initialize repository"
```

Usted debería también crear un nuevo repositorio dando click en el botón "Create" en Bitbucket (Figura 2.1), y luego subir sus archivos al repositorio remoto:

```
$ git remote add origin git@bitbucket.org:<username>/toy_app.git
$ git push -u origin --all # pushes up the repo and its refs for the first time
```

Finalmente, nunca es demasiado pronto para desplegar, sugiero que lo haga siguiendo los mismos pasos que para el "hello, world!" indicados en los Listados 1.8 y 1.9.[1] Luego haga **`commit`** de los cambios y súbalos a Heroku:

[1] La razón principal para hacer esto es que la página que se crea por defecto en Rails, usualmente genera error en Heroku, lo que hace difícil decir si el despliegue fue exitoso o no.

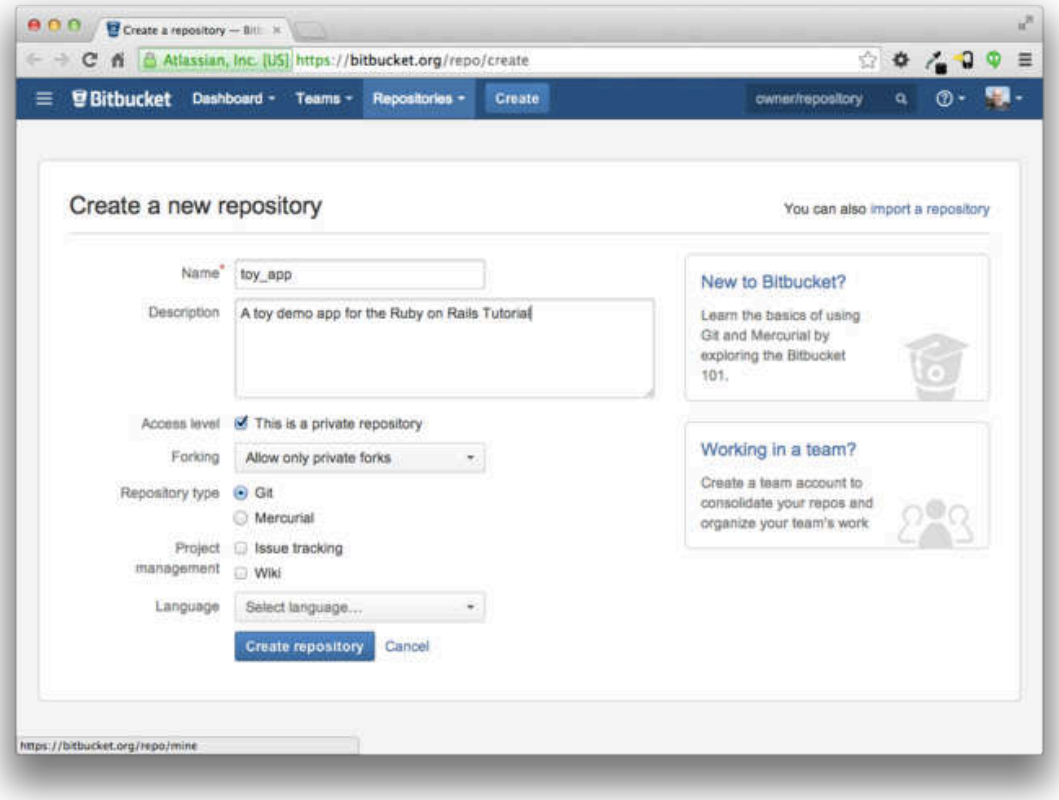

Figura 2.1: Creando el repositorio para la aplicación de juguete en Bitbucket.

```
$ git commit -am "Add hello"
$ heroku create
$ git push heroku master
```

(De igual forma que en la Sección 1.5, es probable que vea algunos mensajes de advertencia, que puede ignorar por ahora; los eliminaremos en la Sección 7.5.) Excepto por la dirección de la aplicación en Heroku, el resultado debería ser el mismo que el de la Figura 1.18.

Ahora estamos listos para iniciar el desarrollo de la aplicación. El primer paso típico cuando se desarrolla una aplicación web, es crear un *modelo de datos*, que es una representación de las estructuras necesarias para nuestra aplicación. En nuestro caso, la aplicación de juguete será un microblog, con sólo usuarios y microposts. Por lo tanto, empezaremos con un modelo para los *usuarios* de la aplicación (Sección 2.1.1), y luego añadiremos un modelo para los *microposts* (Sección 2.1.2).

2.1.1 Un modelo de juguete para usuarios

Existen tantas opciones para un modelo de datos de usuario, como diferentes formas de registro en la web; nosotros vamos a adoptar un enfoque totalmente minimalista. Los usuarios de nuestra aplicación de juguete tendrán como identificador único un número **`entero`** llamado **`id`**, un **`nombre`** visible públicamente (de tipo **`string`**), y una dirección **`email`** (también de tipo **`string`**) que además funcionará como username. Un resumen del modelo de datos para los usuarios aparece en la Figura 2.2.

Como veremos al iniciar la Sección 6.1.1, la etiqueta **`users`** de la Figura 2.2 corresponde a una *tabla* en la base de datos, y los atributos **`id`**, **`email`**, y **`name`** son *columnas* de esa tabla.

2.1.2 Un modelo de juguete para microposts

La estructura del modelo de datos para los microposts es aún más simple que la de los usuarios: un micropost sólo tiene un **`id`** y un campo **`contenido`** para

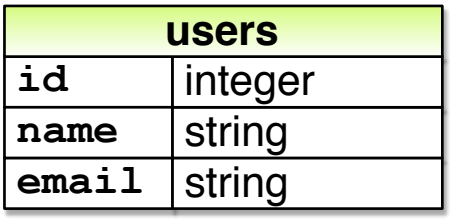

Figura 2.2: El modelo de datos para usuarios.

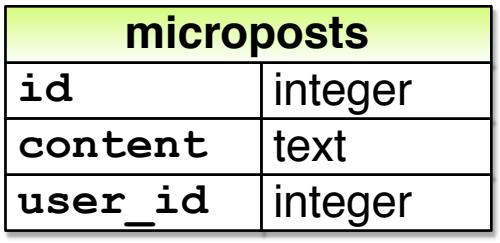

Figura 2.3: El modelo de datos para microposts.

el texto del micropost (de tipo `text`).[2] Sin embargo, existe una consideración adicional: queremos *asociar* cada micropost con un usuario en particular. Conseguiremos esto registrando el `user_id` del dueño del mensaje. Los resultados se muestran en la Figura 2.3.

Veremos en la Sección 2.3.3 (y de forma más completa en el Capítulo 11) cómo este atributo `user_id` nos permite expresar de forma concisa, la noción de que un usuario potencialmente tiene asociados muchos microposts.

[2]Debido a que los microposts son pequeños por diseño, el tipo `string` es suficientemente grande para contenerlos, pero al usar `text` se expresa mejor nuestra intención, al mismo tiempo de que nos da mayor flexibilidad por si en algún momento deseamos modificar la restricción de longitud.

2.2 El recurso Users

En esta sección, implementaremos el modelo de datos para los usuarios de la Sección 2.1.1, junto con una interfaz web para ese modelo. La combinación constituirá un *recurso Users*, que nos permitirá considerar a los usuarios como objetos que pueden ser creados, leídos, actualizados y borrados vía web mediante el protocolo HTTP. Como prometimos en la introducción, nuestro recurso Users será creado por un programa generador de estructuras temporales, que viene como estándar en cada proyecto Rails. Le exhorto a no mirar muy de cerca el código generado; en este punto sólo le causaría confusión.

La creación de estructuras temporales en Rails es generada al pasar el comando **`scaffold`** al script **`rails generate`**. El argumento del comando **`scaffold`** es el nombre del recurso en singular (en este caso, **`User`**), junto con los parámetros opcionales para los atributos del modelo de datos:[3]

```
$ rails generate scaffold User name:string email:string
      invoke  active_record
      create    db/migrate/20140821011110_create_users.rb
      create    app/models/user.rb
      invoke    test_unit
      create      test/models/user_test.rb
      create      test/fixtures/users.yml
      invoke  resource_route
       route    resources :users
      invoke  scaffold_controller
      create    app/controllers/users_controller.rb
      invoke    erb
      create      app/views/users
      create      app/views/users/index.html.erb
      create      app/views/users/edit.html.erb
      create      app/views/users/show.html.erb
      create      app/views/users/new.html.erb
      create      app/views/users/_form.html.erb
      invoke    test_unit
      create      test/controllers/users_controller_test.rb
      invoke    helper
      create      app/helpers/users_helper.rb
      invoke      test_unit
      create        test/helpers/users_helper_test.rb
      invoke    jbuilder
```

[3] El nombre de la estructura temporal seguida de la convención de *modelos*, que van en singular, a diferencia de los recursos y controladores, que van en plural. De esta forma, tenemos **`User`** en vez de **`Users`**.

```
      create      app/views/users/index.json.jbuilder
      create      app/views/users/show.json.jbuilder
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/users.js.coffee
      invoke    scss
      create      app/assets/stylesheets/users.css.scss
      invoke  scss
      create    app/assets/stylesheets/scaffolds.css.scss
```

Al incluir **`name:string`** y **`email:string`**, hemos organizado el modelo User para que tenga la forma mostrada en la Figura 2.2. (Observe que no hay necesidad de incluír un parámetro para el **`id`**; éste es creado automáticamente por Rails para emplearlo como *llave primaria* en la base de datos.)

Para continuar con la aplicación de juguete, primero necesitamos *migrar* la base de datos usando *Rake* (Recuadro 2.1):

```
$ bundle exec rake db:migrate
==  CreateUsers: migrating ====================================================
-- create_table(:users)
   -> 0.0017s
==  CreateUsers: migrated (0.0018s) ===========================================
```

Esto simplemente actualiza la base de datos con nuestro nuevo modelo de datos **`user`**. (Aprenderemos más sobre migraciones de base de datos al iniciar la Sección 6.1.1.) Observe que, con la finalidad de asegurar que el comando utiliza la versión de Rake correspondiente a nuestro **`Gemfile`**, necesitamos ejecutar **`rake`** utilizando **`bundle exec`**. En varios sistemas, incluyendo el IDE en la nube, puede omitir **`bundle exec`**, pero en otros sistemas es necesario, por lo que lo incluiré por completez.

Con esto, podemos ejecutar el servidor web local en una pestaña distinta (Figura 1.7):[4]

```
$ rails server -b $IP -p $PORT    # Use only `rails server` if running locally
```

[4]El script **`rails`** está diseñado de tal forma que usted no necesite utilizar **`bundle exec`**.

Ahora la aplicación de juguete debería estar disponible en el servidor local, como se describe en la Sección 1.3.2. (Si usted está utlizando el IDE en la nube, asegúrese de abrir el servidor de desarrollo resultante en una nueva pestaña del *navegador*, no dentro del mismo IDE.)

Recuadro 2.1. Rake

Según la tradición Unix, la utilería *make* ha jugado un rol importante en la construcción de programas ejecutables a partir de código fuente; muchos hackers de computadoras se han comprometido a ejercitar la memoria con la línea

```
$ ./configure && make && sudo make install
```

comúnmente empleada para compilar código en los sistemas Unix (incluyendo Linux y Mac OS X).

Rake es un *make para Ruby*, un lenguaje tipo *make* escrito en Ruby. Rails utiliza Rake extensamente, especialmente para las innumerables pequeñas tareas administrativas que se necesitan cuando se desarrolla aplicaciones web respaldadas por una base de datos. El comando **`rake db:migrate`** es probablemente el más común, pero existen muchos otros; usted puede consultar una lista de tareas que puede realizar sobre la base de datos, utilizando **`-T db`**:

```
$ bundle exec rake -T db
```

Para ver todas las tareas disponibles del comando Rake, ejecute

```
$ bundle exec rake -T
```

Es probable que la lista sea abrumadora, pero no se preocupe, no tiene que saber todos (o la mayoría) de estos comandos. Al final del *Tutorial de Rails*, usted conocerá los más importantes.

URL	Acción	Funcionalidad
/users	`index`	página para enlistar a todos los usuarios
/users/1	`show`	página para mostrar al usuario con id `1`
/users/new	`new`	página para crear un nuevo usuario
/users/1/edit	`edit`	página para editar al usuario con id `1`

Tabla 2.1: La relación entre páginas y URLs para el recurso Users.

2.2.1 Un recorrido por User

Si visitamos la URL raíz: / (se lee "diagonal", como se señala en la Sección 1.3.4), obtenemos la misma página de Rails por defecto, mostrada en la Figura 1.9, pero al generar el recurso Users mediante la creación de estructuras temporales, también creamos varias páginas para manipular a los usuarios. Por ejemplo, la página para enlistar a todos los usuarios se encuentra en /users, y la página para crear un nuevo usuario está en /users/new. El resto de esta sección está dedicado a realizar un viaje relámpago através de estas páginas de usuario. Conforme avanzamos, puede ser útil consultar como referencia la Tabla 2.1, que muestra la relación entre páginas y URLs.

Empezaremos con la página que muestra a todos los usuarios de nuestra aplicación, llamada `index`; como es de esperarse, inicialmente no hay ningún usuario (Figura 2.4).

Para crear un nuevo usuario, visitamos la página `new`, como se muestra en la Figura 2.5. (Puesto que http://0.0.0.0:3000 o la parte de la dirección del IDE en la nube está implícita cada vez que desarrollamos localmente, la omitiré de aquí en adelante.) En el Capítulo 7, esta página se convertirá en la página para enrolar usuarios.

Podemos crear un usuario al ingresar los valores de nombre y correo electrónico en los campos de texto y luego presionar el botón Create User. El resultado es la página del usuario `show`, como se muestra en la Figura 2.6. (El mensaje verde de bienvenida se logra usando *flash*, el cual aprenderemos en la Sección 7.4.2.) Observe que la URL es /users/1; como puede suponer, el número `1` es simplemente el atributo `id` del usuario, como en la Figura 2.2. En la Sección 7.1, esta página se convertirá en el perfil del usuario.

Para modificar la información de un usuario, visitamos la página `edit`

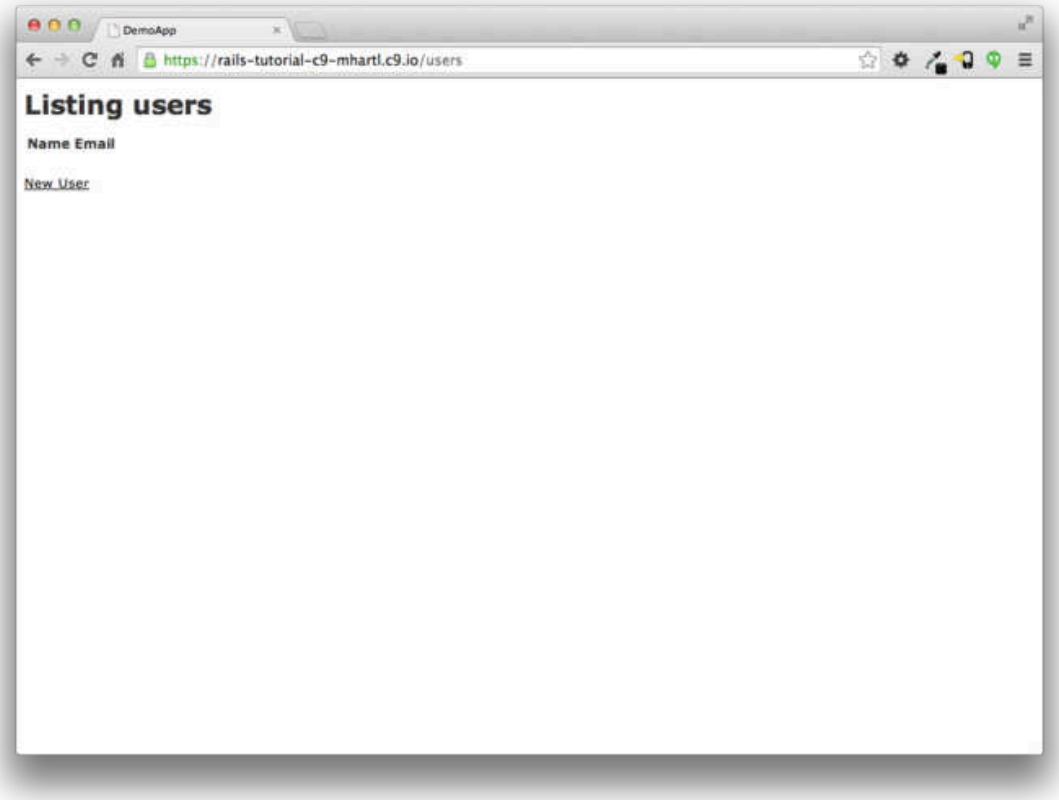

Figura 2.4: La página inicial de listado de usuarios (/users).

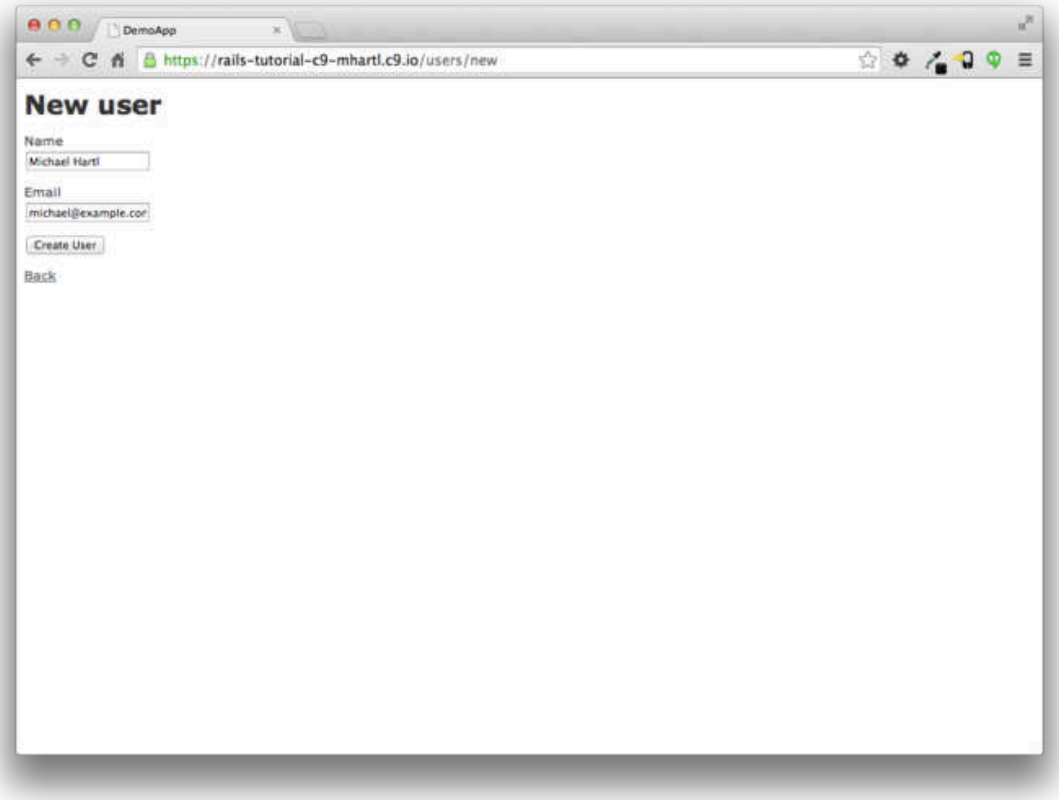

Figura 2.5: La página para crear un usuario nuevo (/users/new).

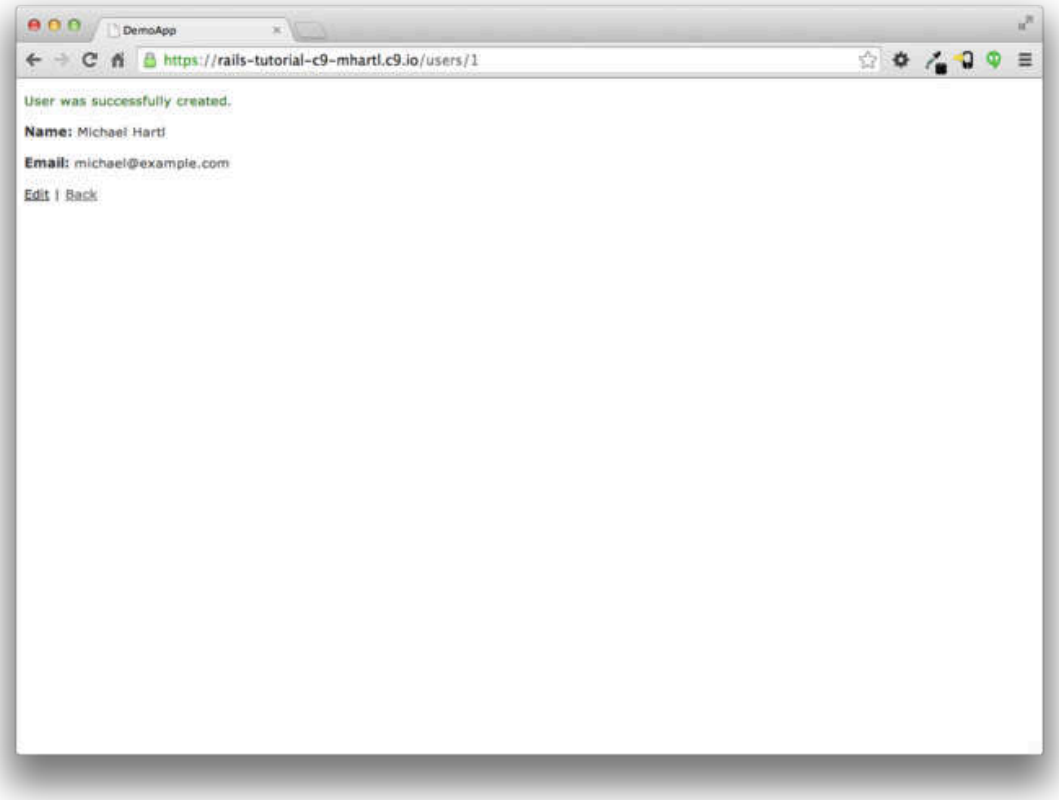

Figura 2.6: La página para mostrar un usuario (/users/1).

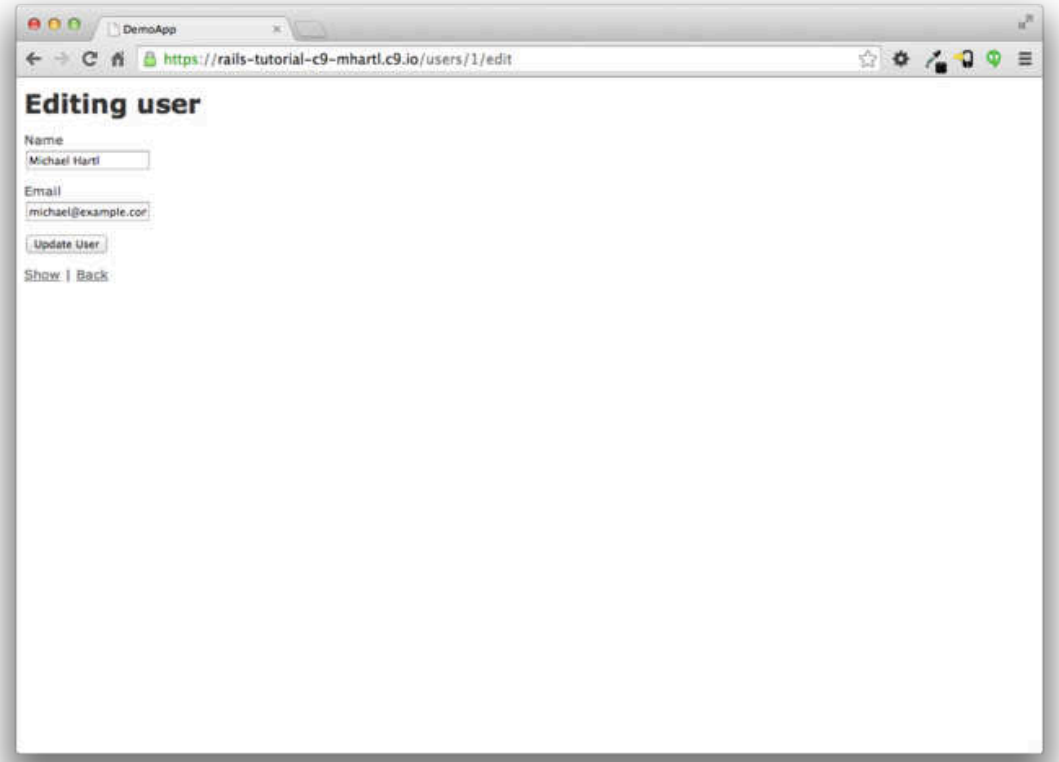

Figura 2.7: La página para editar un usuario (/users/1/edit).

(Figura 2.7). Al cambiar la información del usuario y presionar el botón Update User, conseguimos actualizar la información del usuario en la aplicación de juguete (Figura 2.8). (Como veremos en detalle al iniciar el Capítulo 6, estos datos de usuario son almacenados en una base de datos en el servidor.) Agregaremos funcionalidad para editar/actualizar usuarios en la Sección 9.1.

Ahora crearemos un segundo usuario volviendo a visitar la página `new` y enviando un segundo conjunto de datos de usuario; el listado de usuarios resultante `index` se muestra en la Figura 2.9. En la Sección 7.1 desarrollaremos una versión más refinada de este listado.

Habiendo mostrado cómo crear, mostrar y editar usuarios, finalmente llegamos a revisar cómo borrarlos (Figura 2.10). Verifique que al dar click en el