Atica On Rails

Seminario sobre Ruby On Rails para el personal del Área de Tecnologías de la Información y las Comunicaciones Aplicadas de la Universidad de Murcia

BETA v0.1

(vamos. . . , aún sin terminar)

Juan José Vidal Agustín
<juanjova@um.es>
Área de Tecnologías de la Información y las Comunicaciones Aplicadas

UNIVERSIDAD DE MURCIA

03 de Diciembre de 2007

Resumen

Ruby on Rails, también conocido como RoR o Rails es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración. El lenguaje de programación Ruby permite la metaprogramación, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. Rails se distribuye a través de RubyGems, que es el formato oficial de paquete y canal de distribución de librerías y aplicaciones Ruby.

Índice general

Capítulo 1

Instalación

1.1. UNIX y GNU/Linux

1. Muchas distribuciones de UNIX y GNU/Linux ya vienen con Ruby instalado. Necesitaremos una versión de Ruby igual o superior a la 1.8.2. Podemos verificar la versión que tenemos instalada a través de la terminal, ejecutando el comando `ruby -v`. En caso de tenerlo instalado, pasaremos al paso 3 de esta lista.

2. Lo más normal hoy, es que la distribución de GNU/Linux que tengamos instalada lleve un sistema de gestión de paquetes, como podría ser `yum` o `apt`. En este caso simplemente tendríamos que instalar Ruby desde un terminal a través de estos programas (p.e.: `apt-get install ruby`). En caso contrario tendremos que instalar Ruby desde el código fuente:

```
juanjo> tar xzf ruby-x.y.z.tar.gz
juanjo> cd ruby-x.y.z
ruby-x.y.z> ./configure
ruby-x.y.z> make
ruby-x.y.z> make test
ruby-x.y.z> sudo make install
Password: <enter your password>
```

3. A continuación instalaremos RubyGems (*http://rubygems.rubyforge.org*). Lo descargaremos y realizaremos las siguientes acciones desde un terminal:

```
juanjo> tar xzf rubygems-0.9.5.tar.gz
juanjo> cd rubygems-0.9.5
rubygems-0.9.5> sudo ruby setup.rb
Password: <enter your password>
```

4. Ahora utilizaremos RubyGems para instalar *Ruby On Rails*. Desde el mismo terminal ejecutaremos el siguiente comando:

```
sudo gem install rails --include-dependencies --remote
```

1.2. Apple MacOS X

1. Apple MacOS X ya viene con Ruby instalado de serie. Podemos verificar la versión que tenemos instaladas a través de la terminal, ejecutando el comando `ruby -v`. En caso de no tenerlo instalado, podemos utilizar las instrucciones para hacerlo en GNU/Linux.

2. A continuación instalaremos RubyGems (*http://rubygems.rubyforge.org*). Lo descargaremos y realizaremos las siguientes acciones desde un terminal:

```
juanjo> tar xzf rubygems-0.9.5.tar.gz
juanjo> cd rubygems-0.9.5
rubygems-0.9.5> sudo ruby setup.rb
Password: <enter your password>
```

3. Ahora utilizaremos RubyGems para instalar *Ruby On Rails*. Desde el mismo terminal ejecutaremos el siguiente comando:

```
sudo gem install rails --include-dependencies --remote
```

1.3. Microsoft Windows

Podemos instalar el framework de diversas formas para su correcto funcionamiento en Microsoft Windows:

1.3.1. Componente a componente

Para empezar a trabajar con *Ruby On Rails* de forma básica, al menos necesitamos los siguientes componentes:

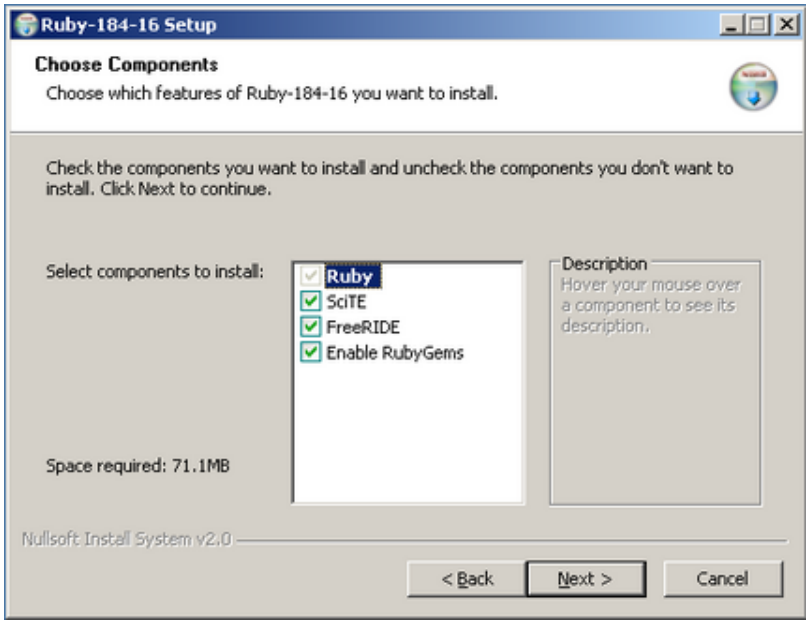
- Ruby
- RubyGems
- El framework *Ruby On Rails*

Instalación de Ruby

La instalación de Ruby no puede ser más simple:

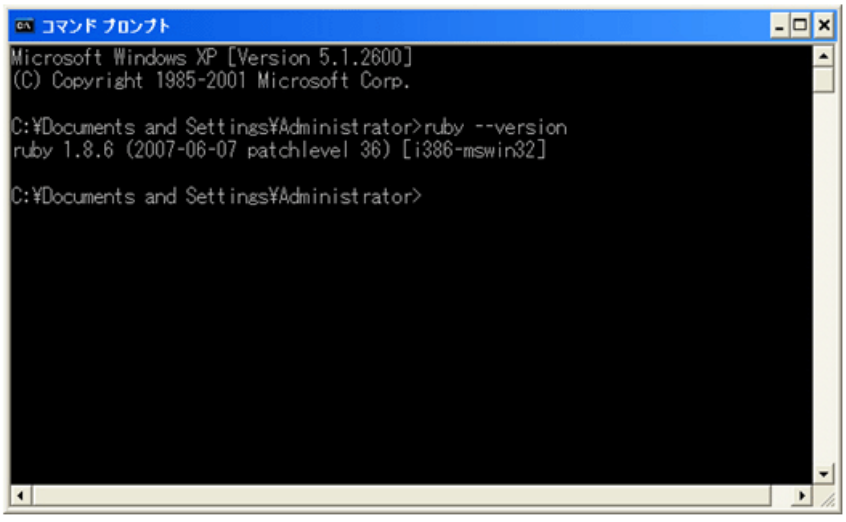
1. Descarga el último paquete instalador de Ruby para Windows (*http://rubyinstaller.rubyforge.org*). En el momento de escribir este artículo, la última versión es *ruby186-26_rc2.exe*).

2. Haz doble clic en el ejecutable que has descargado y sigue las instrucciones del instalador. A no ser que tengas requisitos especiales, simplemente pulsa *Enter* para aceptar todas las opciones por defecto.



3. Para comprobar que todo ha ido bien, desde la consola de comandos de Windows `(Inicio > Ejecutar > cmd)` tecleamos lo siguiente: `ruby --version`

4. Si todo ha ido bien, nos tendrá que aparecer algo parecido a esto en la consola: `ruby 1.8.6 (2007-12-04) [i386-mswin32]` Con este comando podremos conocer qué versión de ruby tenemos instalada en nuestra máquina.



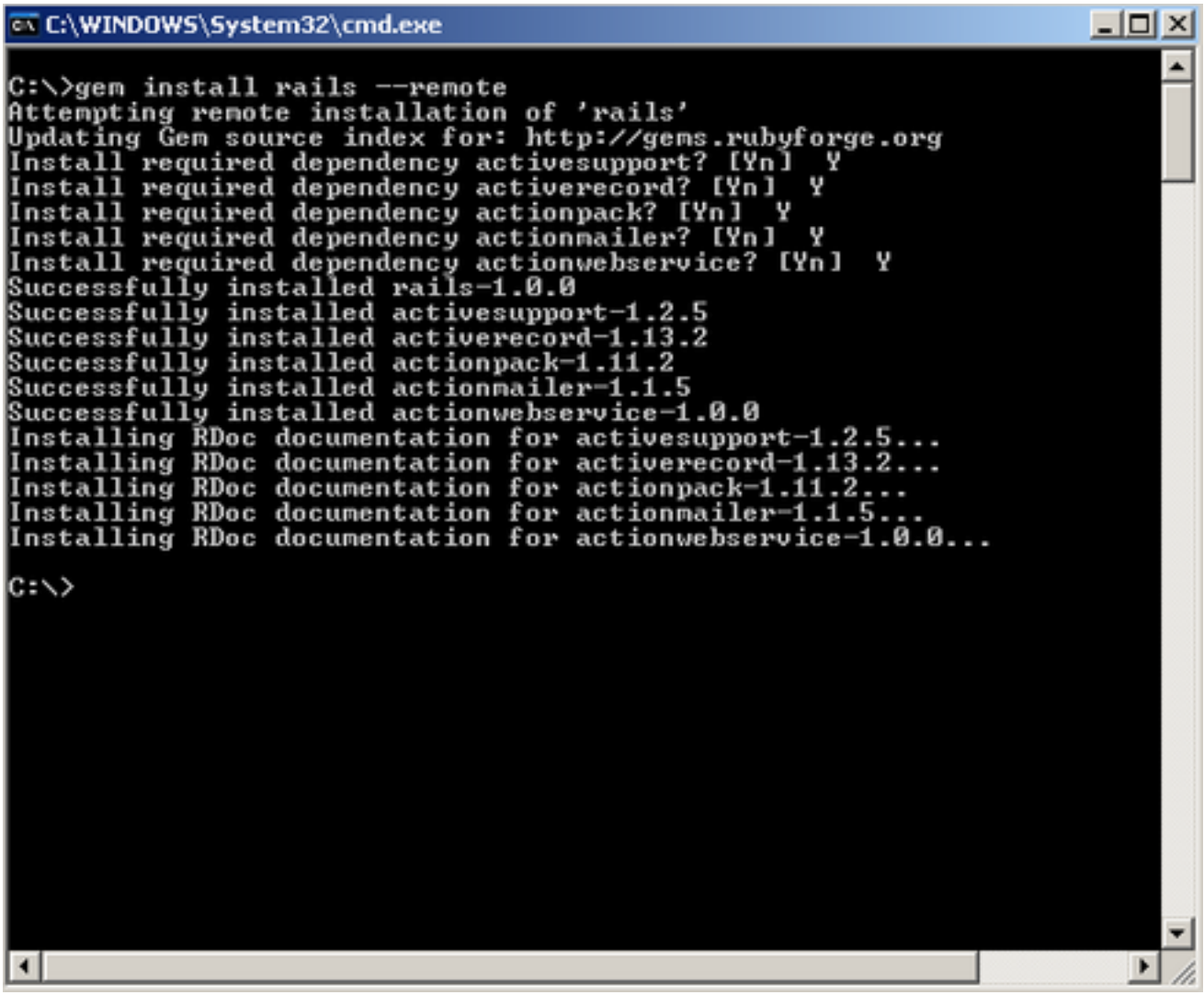
Nota: El instalador de Windows viene con el gestor de paquetes **RubyGems** (*http://rubygems.rubyforge.org/*) ya instalado.

Instalación de Ruby On Rails

Para instalar Rails, desde la consola escribimos lo siguiente:

```
gem install rails --include-dependencies --remote
```

Esto descargará unos paquetes desde Internet y los instalará. Una vez finalizado. Nos creamos un directorio para almacenar la Aplicación Web. Por ejemplo: `C:\railsapps`



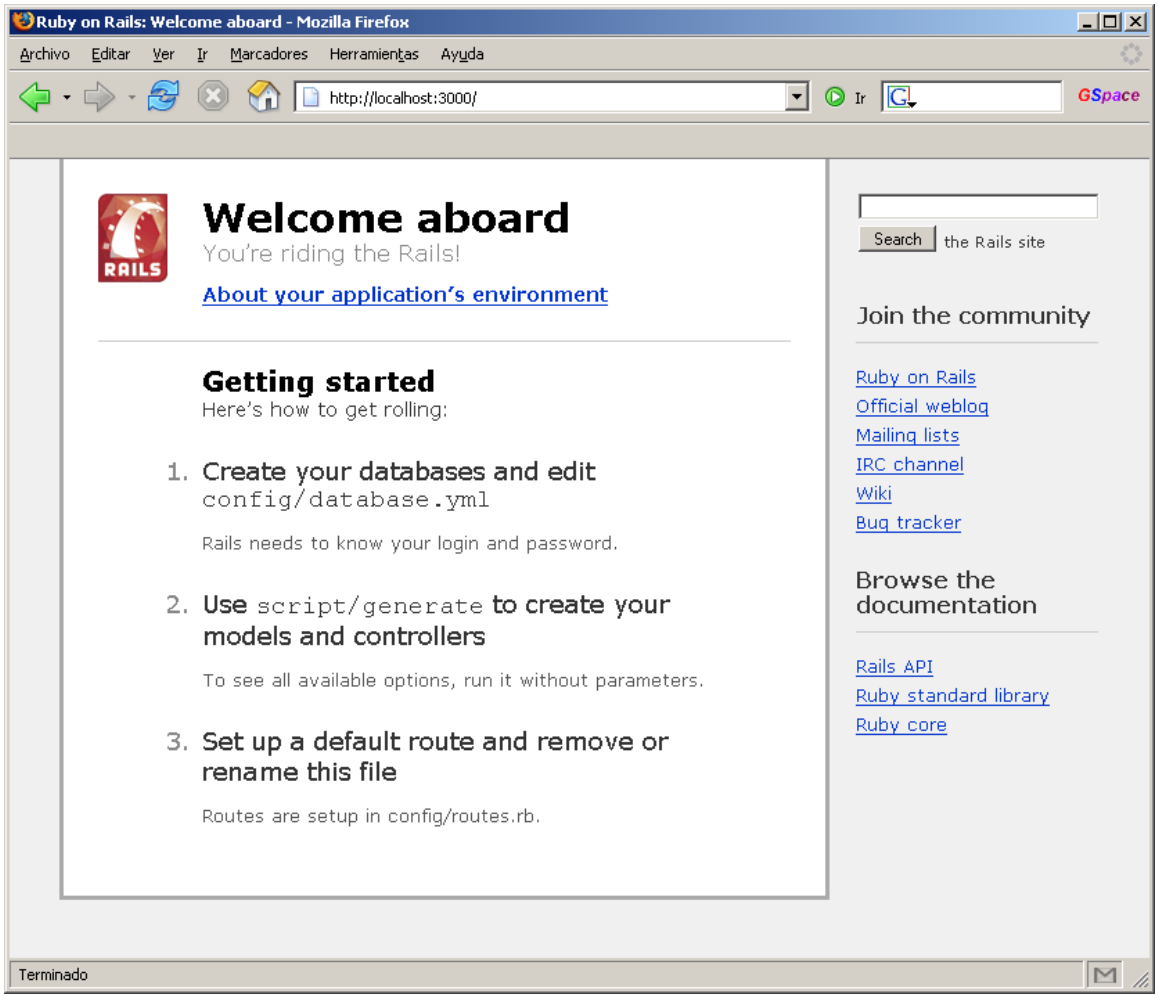
Desde linea de comandos escribimos `cd C:\railsapps` para situarnos en el directorio y luego escribimos rails ./proyecto1. Con eso, se nos crea un sitio Web en una carpeta llamada proyecto1 dentro de `C:\railsapps`.

Por último, nos colocamos en la carpeta de proyecto1 escribiendo `cd C:\railsapps\proyecto1` y escribimos lo siguiente:

```
ruby script/server
```

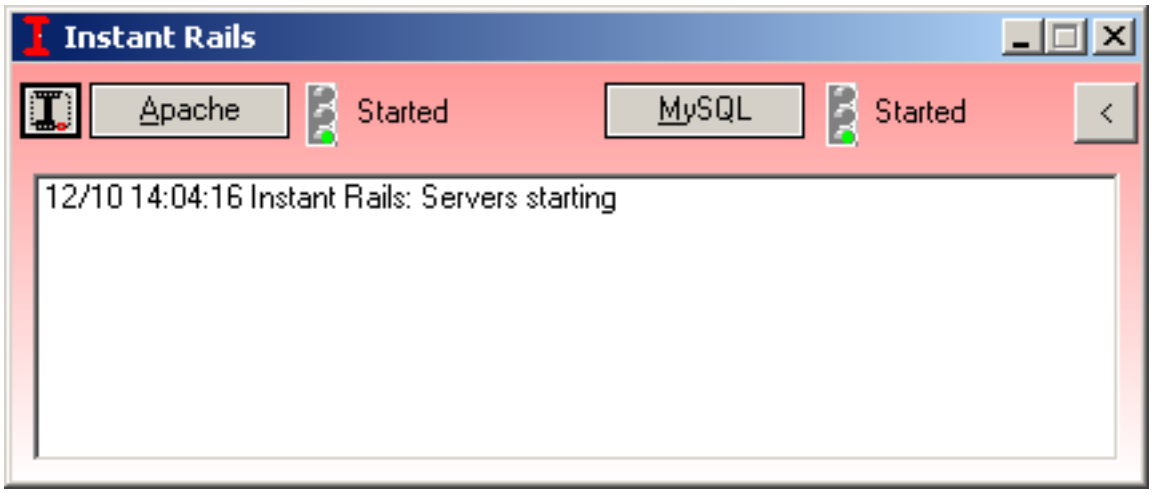
Ya solo queda ir al navegador y escribir la siguiente dirección: http://localhost:3000

Si todo ha ido bien, tendremos que ver una pantalla como esta:



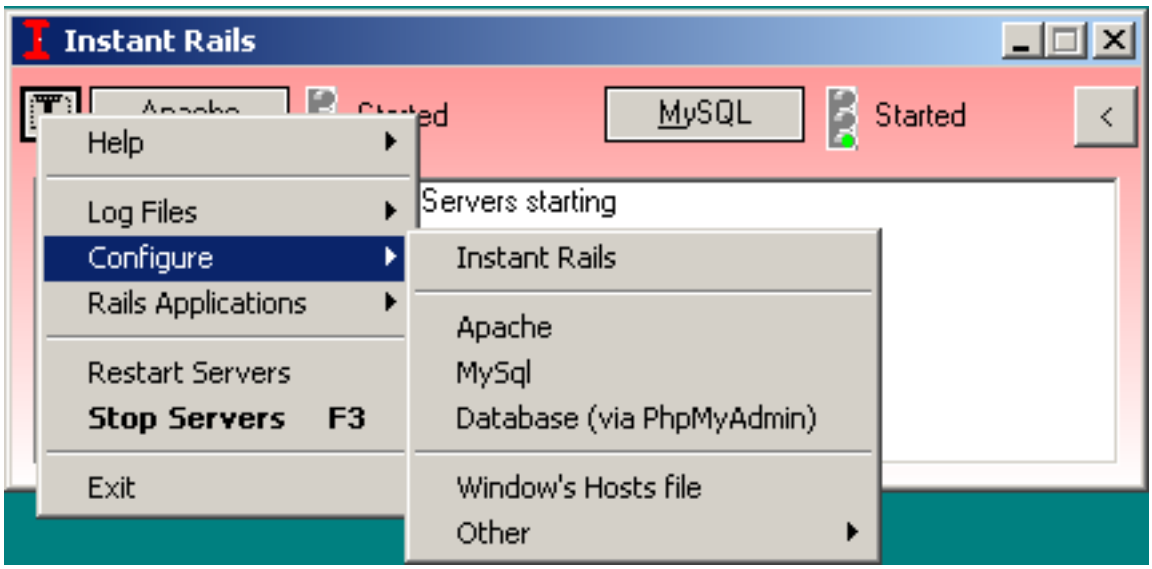
1.3.2. InstantRails

InstantRails (*http://instantrails.rubyforge.org/*) es una solución sencilla para poner a funcionar Ruby y Rails en nuestro Microsoft Windows. Sin afectar al resto de programas que podamos tener ya instalados, instala Ruby, *Ruby On Rails* Apache, MySQL y PHPMyAdmin, todo ello en un solo directorio.



Pasos a seguir

1. Nos descargamos el zip desde *http://instantrails.rubyforge.org*
2. Descomprimimos el fichero en una carpeta (que no contenga espacios)
3. Ejecutamos *InstantRails.exe*
4. Una vez instalado, arrancarán los servidores Apache y MySQL: Si aparece algún mensaje de error, revisa la causa. Por ejemplo, no arranca el servidor MySQL porqué ya se está ejecutando.
5. En la barra de tareas te aparecerá un icono (una I mayúscula roja con una lucecita parpadeante) a través del cual podrás acceder a la configuración.



Enlaces de interés

- Instalar *Ruby On Rails* en Microsoft Windows XP con Apache
  http://ubertinodacasale.wordpress.com/2007/04/05/instalar-ruby-en-apache-bajo-windows-xp/

1.4. IDEs de desarrollo

- **Opción recomendada:** Eclipse + Aptana Extension + RadRails Extension + Subclipse
- Aptana Studio Community Edition
- NetBeans 6.0
- RadRails
- Vi

1.5. Actualización de Ruby On Rails

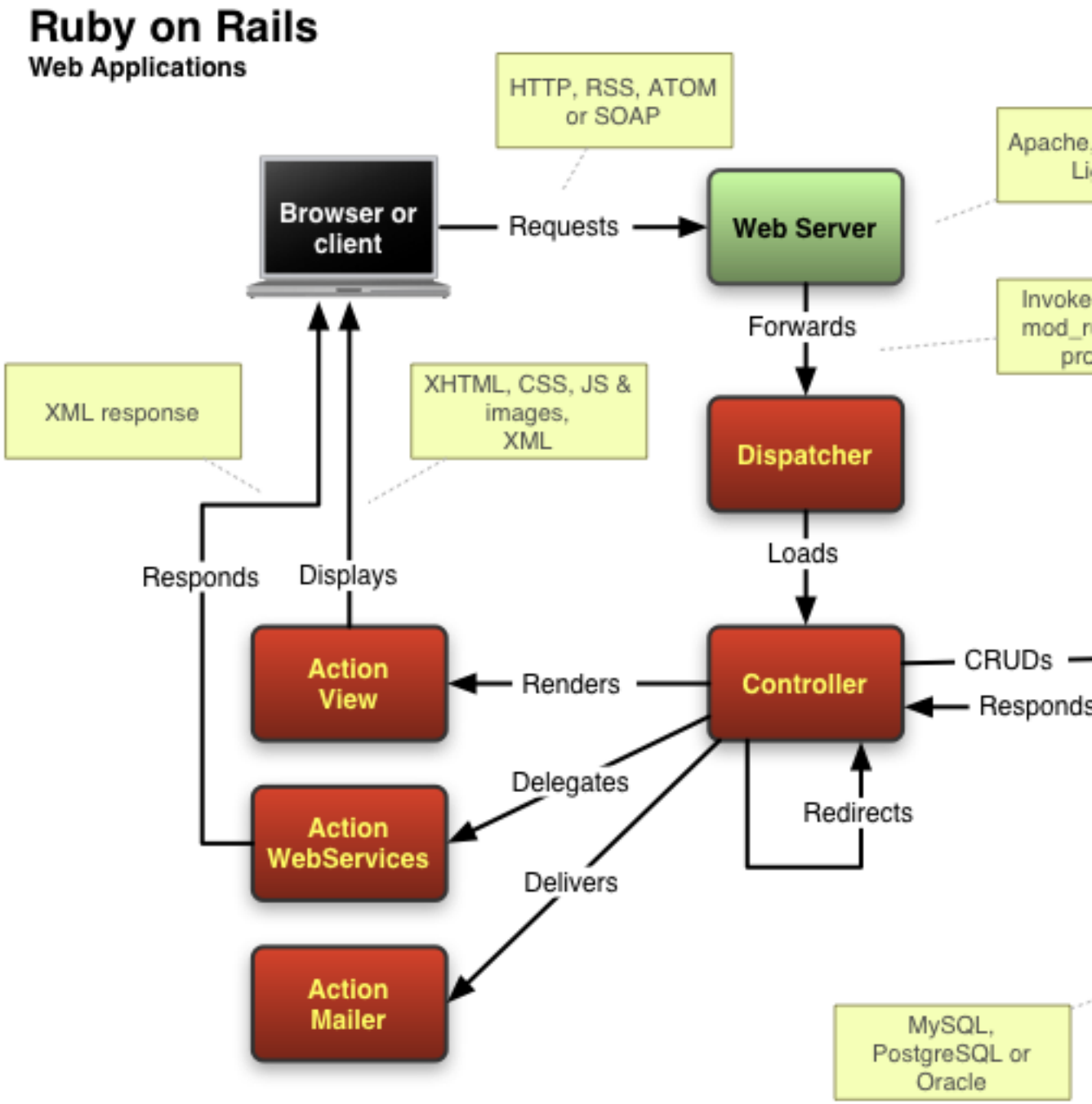
Una vez instalado *Ruby On Rails* en cualquiera de los sistemas que más arriba comentamos, su actualización es en extremo simple. Sólo nos basta con abrir una terminal y ejecutar el siguiente comando:

```
gem update rails
```

Capítulo 2

Ruby On Rails

2.1. Arquitectura de una aplicación Rails



2.2. Generando el proyecto

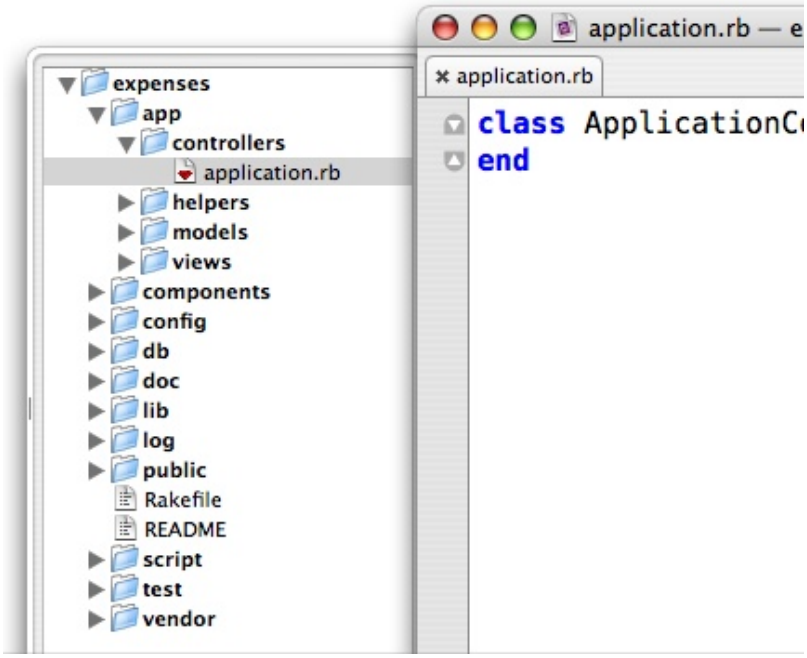
Para ver la estructura de un proyecto generaremos un proyecto de ejemplo sobre el que poder trabajar. Para ello ejecutaremos el siguiente comando desde una terminal (podríamos también hacerlo de forma gráfica desde el IDE que utilicemos para trabajar con *Ruby On Rails*):

```
ishka@bujia:~/workrails$ rails aticaonrails
```

Este comando nos generará un conjunto de directorios, que compondrán el proyecto.

```
ishka@bujia:~/workrails/aticaonrails$ cd aticaonrails

ishka@bujia:~/workrails/aticaonrails$ ls -l
total 64
drwxr-xr-x 6 ishka ishka 4096 2007-12-03 17:22 app
drwxr-xr-x 2 ishka ishka 4096 2007-12-03 17:22 components
drwxr-xr-x 3 ishka ishka 4096 2007-12-03 17:22 config
drwxr-xr-x 2 ishka ishka 4096 2007-12-03 17:22 db
drwxr-xr-x 2 ishka ishka 4096 2007-12-03 17:22 doc
drwxr-xr-x 3 ishka ishka 4096 2007-12-03 17:22 lib
drwxr-xr-x 2 ishka ishka 4096 2007-12-03 17:22 log
drwxr-xr-x 5 ishka ishka 4096 2007-12-03 17:22 public
-rw-r--r-- 1 ishka ishka  307 2007-12-03 17:22 Rakefile
-rw-r--r-- 1 ishka ishka 9102 2007-12-03 17:22 README
drwxr-xr-x 4 ishka ishka 4096 2007-12-03 17:22 script
drwxr-xr-x 7 ishka ishka 4096 2007-12-03 17:22 test
drwxr-xr-x 6 ishka ishka 4096 2007-12-03 17:22 tmp
drwxr-xr-x 3 ishka ishka 4096 2007-12-03 17:22 vendor
```

2.2.1. Explicación de la estructura



app Controladores, Modelos, Vistas y Helpers

components Miniaplicaciones que pueden utilizar controladores, modelos y vistas de forma conjunta.

config Configuración de base de datos, de rutas y ajustes de entorno.

db Ficheros de esquema de base de datos y ficheros de migración Rails

doc Documentación de la aplicación

lib Código de la aplicación que no pertenece a controladores, modelos o helpers. Por ejemplo, unas librerías para la generación de PDF

log Ficheros de log de acceso y de error de la aplicación.

public CSS, Javascripts, imágenes y otros ficheros estáticos.

script Scripts de generación de código, herramientas de depuración y utilidades de depuración

test Ficheros para testear la aplicación, test de unidad, de integración de código, fixtures...

tmp Directorio dónde se encuentran ficheros de sesión, caché, sockets...

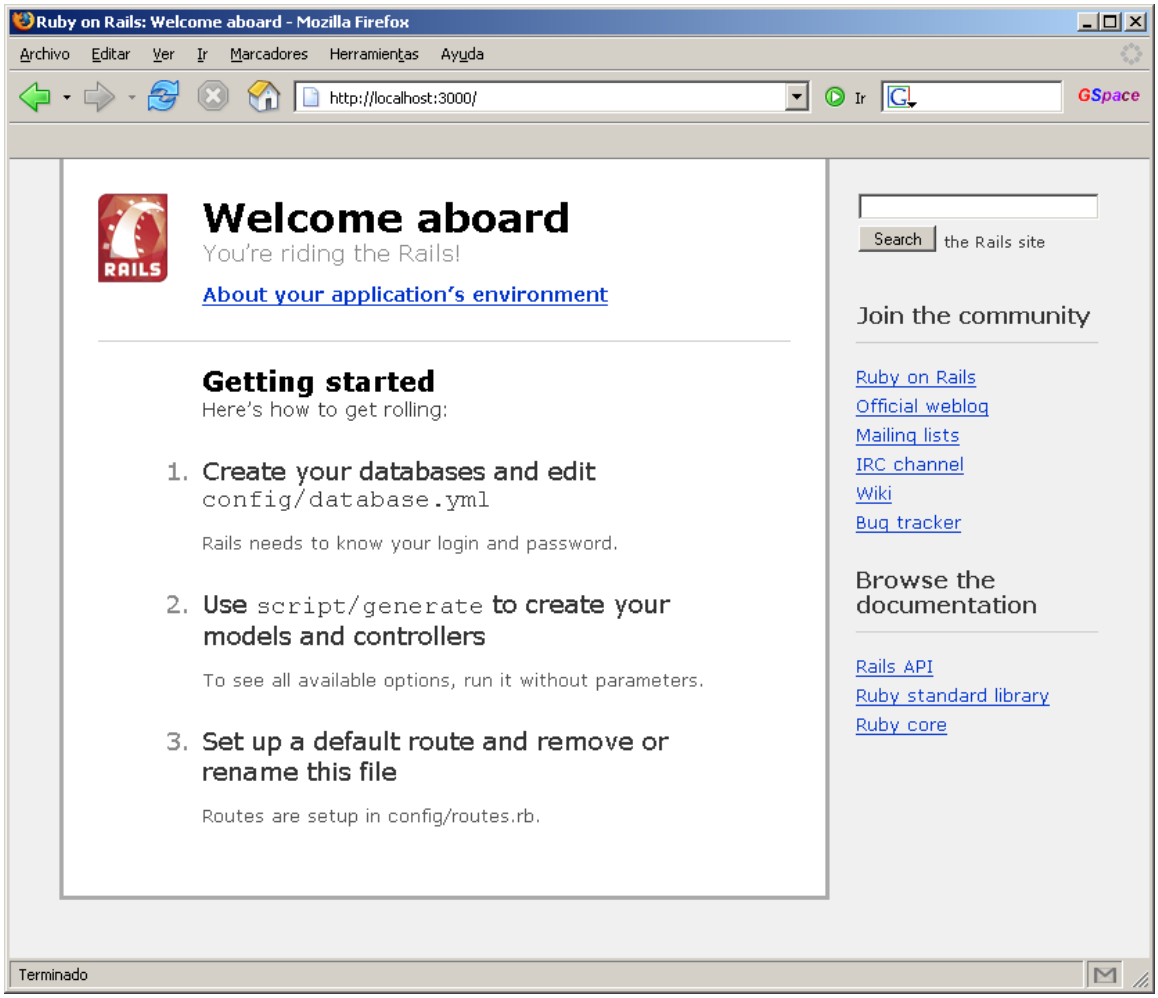
vendor Lugar dónde son instalados los plugins de la aplicación

2.2.2. Probando el proyecto

Para ver si lo que acabamos de generar (el proyecto *AticaOnRails*), funciona correctamente, ejecutaremos los siguientes comandos:

```
 ishka@bujia:~/workrails/aticaonrails$ ruby script/server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2007-12-03 17:42:45] INFO  WEBrick 1.3.1
[2007-12-03 17:42:45] INFO  ruby 1.8.6 (2007-06-07) [x86_64-linux]
[2007-12-03 17:42:45] INFO  WEBrick::HTTPServer#start: pid=25970 port=3000
```

Esto nos arrancará un servidor web (mínimo) que lleva ya la propia aplicación. Si nos fijamos en la última línea, la salida por pantalla nos informa que el puerto 3000 está la aplicación arrancada; por lo que iremos a `http://localhost:3000`



La página que estamos viendo, es la página por defecto para cualquier proyecto *Ruby On Rails*. En ella ya nos indica unas instrucciones mínimas para seguir trabajando.

2.3. Creando la base de datos

Supondremos que el usuario sabe cómo crear una base de datos en el sistema de base de datos con el que suele trabajar: MySQL, PostgreSQL, Oracle..., o puede ponerles partes de trabajo a algún

administrador de sistemas de su organización.

2.3.1. Configuración base de datos

Para configurar la base de datos sobre la que trabajaremos habremos de editar el fichero `/config/database.yml`

```
 # MySQL (default setup).  Versions 4.1 and 5.0 are recommended.
#
# Install the MySQL driver:
#   gem install mysql
# On MacOS X:
#   gem install mysql -- --include=/usr/local/lib
# On Windows:
#   gem install mysql
#       Choose the win32 build.
#       Install MySQL and put its /bin directory on your path.
#
# And be sure to use new-style password hashing:
#   http://dev.mysql.com/doc/refman/5.0/en/old-client.html
development:
  adapter: mysql
  database: aticaonrails_development
  username: root
  password:
  socket: /var/run/mysqld/mysqld.sock

# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
  adapter: mysql
  database: aticaonrails_test
  username: root
  password:
  socket: /var/run/mysqld/mysqld.sock

production:
  adapter: mysql
  database: aticaonrails_production
  username: root
  password:
  socket: /var/run/mysqld/mysqld.sock
```

El fichero que ahora vemos está escrito en YAML (*YAML Ain't Another Markup Language*), lenguaje utilizado para escribir ficheros de configuración esencialmente. Como vemos, está escrito en ternas cla-

ve:valor y es fácilmente entendible. Para configurarlo hemos de poner los datos de nuestras base de datos (previamente creada, claro está). *Ruby On Rails* utiliza para sus proyectos 3 bases de datos por defecto:

- development Base de datos de desarrollo
- test Base de datos para realizar los test.
- production Base de datos de producción

La configuración que yo suelo utilizar durante el desarrollo es la siguiente. Básicamente utilizo esta y no otra por comodidad, para que la base de datos de desarrollo sea la de test y la de producción al mismo tiempo (por desidia :P)

```
defaults: &defaults
  adapter: mysql
  username: root
  password: **********
  host: localhost
  socket: /var/run/mysqld/mysqld.sock

development:
  database: aticaonrails_dev
  <<: *defaults

test:
  database: aticaonrails_test
  <<: *defaults

production:
  database: aticaonrails_pro
  <<: *defaults
```

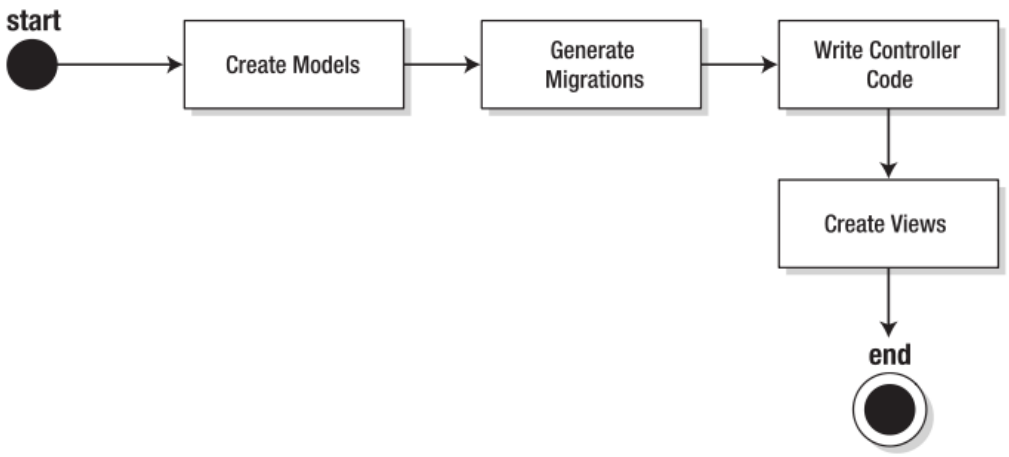
2.3.2. Bases de datos soportadas

- MySQL
- PostgreSQL
- SQLite
- SQL Server
- IBM DB2
- Informix
- Oracle 8i, 9i, y 10g
- Firebird/Interbase
- LDAP - No es exactamente un driver de base de datos, pero ActiveLDAP puede utilizarse con Rails.
- SybaseASA (Sybase Adaptive Server Anywhere conocido SQL Anywhere Studio)

Trabajando en Rails con Oracle

- Ruby on Rails on Oracle: A Simple Tutorial
  http://www.oracle.com/technology/pub/articles/haefel-oracle-ruby.html
- Rails Wiki HowtoConnectToOracle
  http://wiki.rubyonrails.org/rails/pages/HowtoConnectToOracle
- Ruby/OCI8
  http://rubyforge.org/projects/ruby-oci8/
- Ruby on Rails with Oracle FAQ
  http://www.oracle.com/technology/pub/articles/saternos-ror-faq.html

2.4. Creando el modelo



2.4.1. Cómo se nombran las cosas

ActiveRecord, al igual que el resto de Rails, favorece la convención frente a la configuración, es decir, seguir unas normas (convenciones) ya establecidas a la hora de, por ejemplo, llamar a las tablas, campos, ficheros, etc... Esto nos evita (o minimiza) el tener que crear y mantener ficheros de configuración y parametrización.

Los nombres de las tablas deben ser todos en plural: Coches, Productos, Personas, mientas que los Objetos que las referencian iran en singular Coche, Producto, Persona (Rails es lo suficientemente inteligente como para si creamos un objeto de tipo Coche automáticamente saber que su tabla asociada es Coches, si trabajamos con los nombres de las tablas en inglés, muy recomendable, sera capaz de asociar Person con people, así como muchas otras formas de plural irregulares).

Campos especiales en las migraciones

En cuanto al nombre de los campos, rails nos permite total libertad. Por supuesto existen una serie de convenciones que nos facilitarán y simplificarán la tarea de crear los campos de las tablas: si un campo acaba en _at (`vendido_at`) rails espera (y lo tratará), un campo de fecha-hora mientras que si acaba en _on (`visitado_on`) espera un campo de fecha.

Si nuestra tabla incluye un campo llamado lock_version rails implementará de manera automática bloqueos optimistas a la hora de gestionar la concurrencia (2 ó más usuarios accediendo al mismo registro).

Si añadimos un campo llamado `created_at` o `created_on` rails se encargará de insertar la fecha/hora de creación cuando demos de alta un nuevo registro. Para control de modificaciones podemos crear un campo llamado `updated_at` o `updated_on`

2.4.2. Migraciones en Rails

Las migraciones de Rails son una característica que nos permite modificar el esquema de la base de datos usando Ruby.

Alguien se puede preguntar cuáles son las ventajas reales de manipular la base de datos usando Ruby en lugar de SQL. La primera de ellas es que sólo tendremos que lidiar con las peculiaridades de Ruby, en vez de tener que andar consultando las referencias del dialecto de SQL que haya tenido a bien implementar el fabricante de nuestro gestor de bases de datos. La segunda es que, además de modificar el esquema de la base de datos añadiendo tablas, columnas, etc. con las migraciones podemos modificar los datos ya existentes en nuestras tablas.

```
 # db/migrate/001_create_users.rb
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
    end
  end

  def self.down
    drop_table :users
  end
end
```

El método 'up' es el que va a crear nuestras tablas y sus campos correspondientes, al contrario del método 'down' que va a deshacer la migración.

Para crear una tabla (como podemos ver en el archivo) se hace con create_table(name, options = , &block).

Dentro del bloque creamos los campos con column(name, type, options =).

```
# db/migrate/001_create_users.rb
def self.up
  create_table :users do |t|
    t.column :name, :string
    t.column :username, :string, :limit => 20
    t.column :password, :string
    t.column :email, :string, :null => false
    t.column :admin, :boolean, :default => false
    t.column :average, :decimal, :precision => 5, :scale => 2
  end
end
```

type puede ser cualquiera de los siguientes:

- string
- int
- float
- decimal
- time
- timestamp
- datetime
- date
- binary
- boolean

Podemos agregar índices con add_index(table, columns, options =), en donde options puede ser:

- :unique =>true
- :name =>'nombre'

Además tenemos acceso a nuestros modelos para crear registros 'predeterminados'

```
 # db/migrate/001_create_users.rb
def self.up
  create_table :users do |t|
    t.column :name, :string
    t.column :username, :string, :limit => 20
    t.column :password, :string
    t.column :email, :string, :null => false
    t.column :admin, :boolean, :default => false
    t.column :average, :decimal, :precision => 5, :scale => 2
  end
  add_index :users, :name  # <- indice simple
  add_index :users, [:username, :email] # <- indice compuesto

  User.create(:name => 'Edgar Suarez', :username => 'edgarjs',
          :email => 'edgarjs@mimbles.net', :admin => true)
end
```

Ya que tenemos nuestro archivo definido, hacemos esta migración efectiva con:

```
 rake db:migrate
```

Si exploran su base de datos podrán ver una tabla llamada 'schema_info'. Esta tabla lleva el control de las migraciones y contiene un único campo version, cada que hacemos una nueva migración, se incrementa el número de version. Lo entenderemos mejor con lo que sigue.

Supongamos que se nos olvidó agregar el campo para la fecha de registro del usuario. Para solucionar esto podemos hacer 2 cosas:

1. Deshacer la migración, agregar el campo en la definición del archivo, y volver a crear la migración.
2. Crear otro archivo que agregue la columna olvidada.

Por ahora tomaremos la opción 2:

```
ruby script/generate migration add_register_date

      exists  db/migrate
      create  db/migrate/002_add_register_date.rb
```

Y en el archivo creado añadimos la columna con add_column(table, name, type, options =).
También hay que poner lo que deshaga esto: remove_column(table, name)

```
 # db/migrate/002_add_register_date.rb
class AddRegisterDate < ActiveRecord::Migration
  def self.up
    add_column :users, :created_at
  end

  def self.down
    remove_column :users, :created_at
  end
end
```

Y aquí aprovecho para anotar algo:

- La columna con nombre 'created_at' y type=datetime inserta automaticamente la fecha y hora al crear un registro.
- La columna con nombre 'created_on' y type=date inserta automaticamente la fecha al crear un registro.
- La columna con nombre 'updated_at' y type=datetime inserta automaticamente la fecha y hora al actualizar un registro.
- La columna con nombre 'updated_on' y type=date inserta automaticamente la fecha al actualizar un registro.

Pero bueno, continuemos... migramos el archivo:

```
rake db:migrate
```

Como se podemos ver, rake db:migrate sólo toma los archivos que no se han migrado, esto lo hace comparando el número de version en la tabla schema_info y el número del archivo generado.

Bien, ahora supongamos que deseas regresar a la primera versión, en donde no se ha añadido la columna created_at, simplemente ejecutas:

```
rake db:migrate VERSION=1
```

2.5. Creando el controlador

POR HACER

2.5.1. Creando las vistas

POR HACER

2.6. Añadiendo estilo

- **Layout:** Podemos definir nuestra plantilla (layout) en el fichero `/app/views/laout/application.rhtml`. Esta plantilla será utilizada por todas las vistas:

```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  <meta http-equiv="Content-Language" content="es-es" />
  <title>AticaOnRails</title>
  <%= stylesheet_link_tag "lacssdeatica" %>
</head>
<body id="rails-list">
<div id="container">
  <div id="header">
    <h1>AticaOnRails</h1>
    <h3>Hasta el infinito y más allá...</h3>
  </div>
  <div id="content">
    <%= yield -%>
  </div>
  <div id="sidebar"></div>
</div>
</body>
</html>
```

- **CSS:** Los ficheros CSS irán en `/public/stylesheets/`, y tendremos que enlazarlos en la cabecera desde el layout con `<% stylesheet_link_tag "lacssdeatica" %>`

- **Images:** Imágenes en `/public/images/`

Una forma fácil de obtener un buen template CSS con el que poder trabajar es:

1. Descargarse un template desde *http://www.freecsstemplates.org/css-templates/*
2. Descomprimir el .zip y dejar cada cosa en su sitio: imágenes en /public/images, css en /public/stylesheets
3. Editar el fichero CSS que acabamos de copiar, y reemplazar la palabra "images" por "../images". Esta acción la realizamos por estar en diferente ruta las imágenes, claro está.
4. A partir del fichero index.html que se nos descarga, crear un fichero /app/views/layout/application.rhtml. Hemos de acordarnos de poner la línea que include la CSS (`<% stylesheet_link_tag "lacssdeatica" %>`), y yield en el content del fichero index.
5. Voilá

Capítulo 3

Rails un poco más avanzado, pero poco poco más...

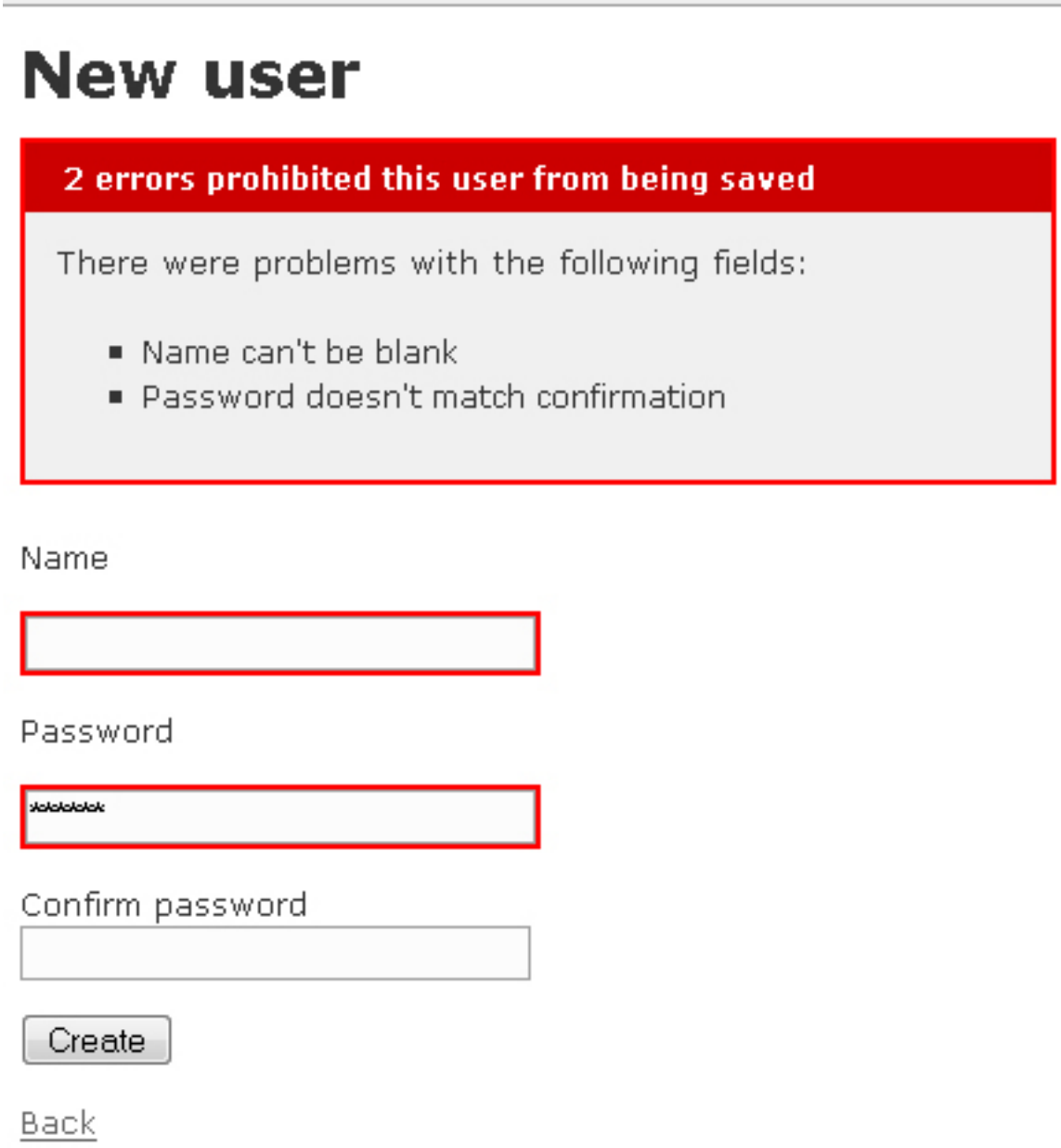
3.1. Validaciones de datos

Fuente: Mimbles! V.2 (*http://mimbles.net/2007/09/24/validaciones-en-rails/*)

Cuando tenemos un formulario, siempre es importante validar los datos que el usuario ingresa. Muchos suelen hacer esto con javascript y nada más, pero no es muy recomendable.

Una práctica buena sería validar el formulario con javascript y hacer otra validación en el lado del servidor.Y seguro otros tantos ni siquiera se molestan en validar porque es una pesadez y quieren evitar la fatiga, y eso es un problema.

Rails incorpora varios métodos para la validación de nuestros formularios, estos son llamados como callbacks antes de guardar el modelo y si éste contiene datos inválidos, permite al usuario editar estos datos mostrando de nuevo el template del formulario con los errores listados.



Los métodos para validar más comunes son:

validates_acceptance_of Valida que el valor del checkbox sea verdadero. Se usa normalmente para cuando se muestra algo como “Acepte los términos y condiciones de uso”.

validates_confirmation_of Valida que el valor del campo sea igual al valor del campo con el nombre campo_confirmation (vg. password_confirmation). Normalente para confirmar contraseñas.

validates_exclusion_of Valida que el valor del campo *no* se encuentre en un grupo de valores. (vg. validates_exclusion_of :username, :in => %w(admin superuser), :message =>Üsername incorrecto").

validates_format_of Valida que el valor del campo concuerde con la expresión dada. (vg. validates_format_of :email, :with =>/\A([^@\s]+)@((?:[-a-z0-9]+.)+[a-z]{2,})\Z/i, :message =>Çampo Email no válido").

validates_inclusion_of Valida que el valor del campo se encuentre en un grupo de valores.

validates_length_of Valida que la longitud sea o se encuentre entre los valores especificados. (vg. validates_length_of :user_name, :within =>6..20)

validates_numericality_of Valida que el valor del campo sea numérico.

validates_presence_of Valida que el valor del campo no esté en blanco.

validates_uniqueness_of Valida que el valor del campo no se encuentre en la base de datos, que sea único por tanto.

Para cada método se puede especificar cuándo se debe realizar dicha validación, puede ser :on =>:create, :on =>:update, :on =>:save

La forma de utilizar estos métodos es simplemente llamándolos en el modelo con el nombre del campo a validar como parámetro.

```
 # app/models/user.rb

class User < ActiveRecord::Base

        validates_presence_of :name

        validates_confirmation_of :password

end
```

Y en nuestra vista tenemos que llamar al método error_messages_for :user.

```
 <h1>New user</h1><%= error_messages_for :user %>

<% form_for(:user, :url => users_path) do |f| %>
  <p>
    <label for="user_name">Name</label><br />
    <%= f.text_field :name %>
  </p>
  <p>
    <label for="user_password">Password</label><br />
    <%= f.password_field :password %>
  </p>
  <p>
    <label for="user_password_confirmation">Confirm password</label><br />
    <%= f.password_field :password_confirmation %>
  </p>
```

```
  <p>
    <%= submit_tag "Create" %>
  </p>
<% end %>

<%= link_to 'Back', users_path %>
```

Cuando se valida un modelo y existen campos inválidos, entonces se agregan errores al objecto errors y luego éste se renderiza en la vista al llamar `error_messages_for :model`

Otro aspecto importante es que cuando se muestran los errores en la vista, si obervamos el código podremos descubrir cómo rails maneja el CSS de los campos con errores.

```
  <p>
    <label for="user_name">Name</label><br />
    <div class="fieldWithErrors">
<input id="user_name" name="user[name]" size="30" type="text" value="" />
    </div>
  </p>
```

Y esto es algo que no nos ayuda mucho a la hora de personalizar nuestra hoja de estilos. Sería mejor si al campo con errores sólo se le agregara una clase, por ejemplo `error_field`.

Añadiendo lo siguiente a nuestro `environment.rb` cambiará el comportamiento de Rails dándonos lo que queremos.

```
# config/environment.rb
# ...

# Include your application configuration below

ActionView::Base.field_error_proc = Proc.new do |html_tag, instance|

    msg = instance.error_message

    error_class = 'error_field'

    if html_tag =~ /<(input|textarea|select)[^>]+class=/

        class_attribute = html_tag =~ /class=['"]/

        html_tag.insert(class_attribute + 7, "#{error_class} ")

    elsif html_tag =~ /<(input|textarea|select)/

        first_whitespace = html_tag =~ /\s/

        html_tag[first_whitespace] = " class='#{error_class}' "
```

```
    end

    html_tag

end
```

Ahora podemos poner el borde rojo a nuestros campos con error más "cómodamente" en la CSS:

```
.error_field {
  border: 1px solid red;
}
```

Para más información sobre validaciones, ir a la API de *Ruby On Rails* en *http://ar.rubyonrails.com/classes/ActiveRecord/Validations/ClassMethods.html*

3.1.1. Crear nuevas validaciones

Fuente: Defining a custom validates in rails (*http://snippets.dzone.com/posts/show/822*)

3.2. Callbacks

Los callbacks son a Rails como los Trigges a las base de datos. Existen una serie de métodos del tipo:

- before_validation
- after_validaton
- before_save
- after_update
- before_create
- after_save
- ...

en los que podremos insertar nuestro código. Rails nos permite crear callbacks comunes a varios modelos de datos. Recordemos que Rails es un framework DRY (Don't Repeat Yourself).

Más información sobre **Callbacks** en la API de *Ruby On Rails* (*http://api.rubyonrails.org/classes/ActiveRecord/Callbacks.html*)

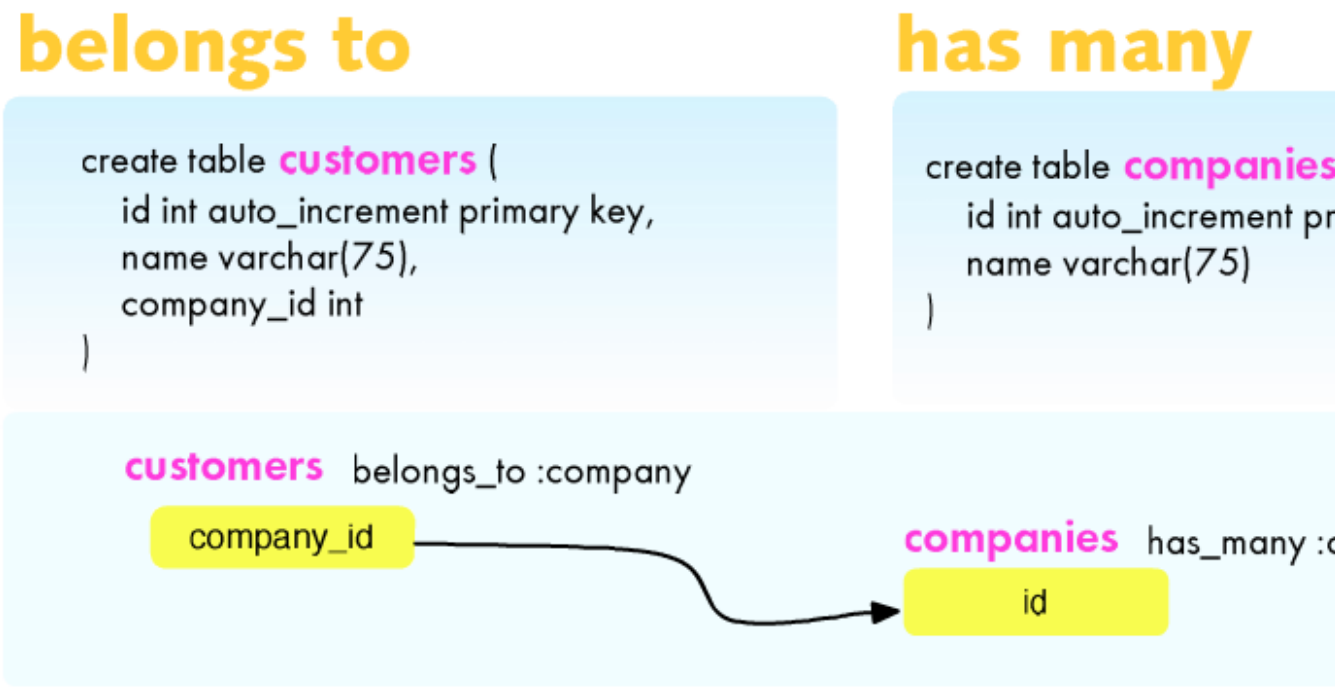
Ejemplo de uso de callbacks

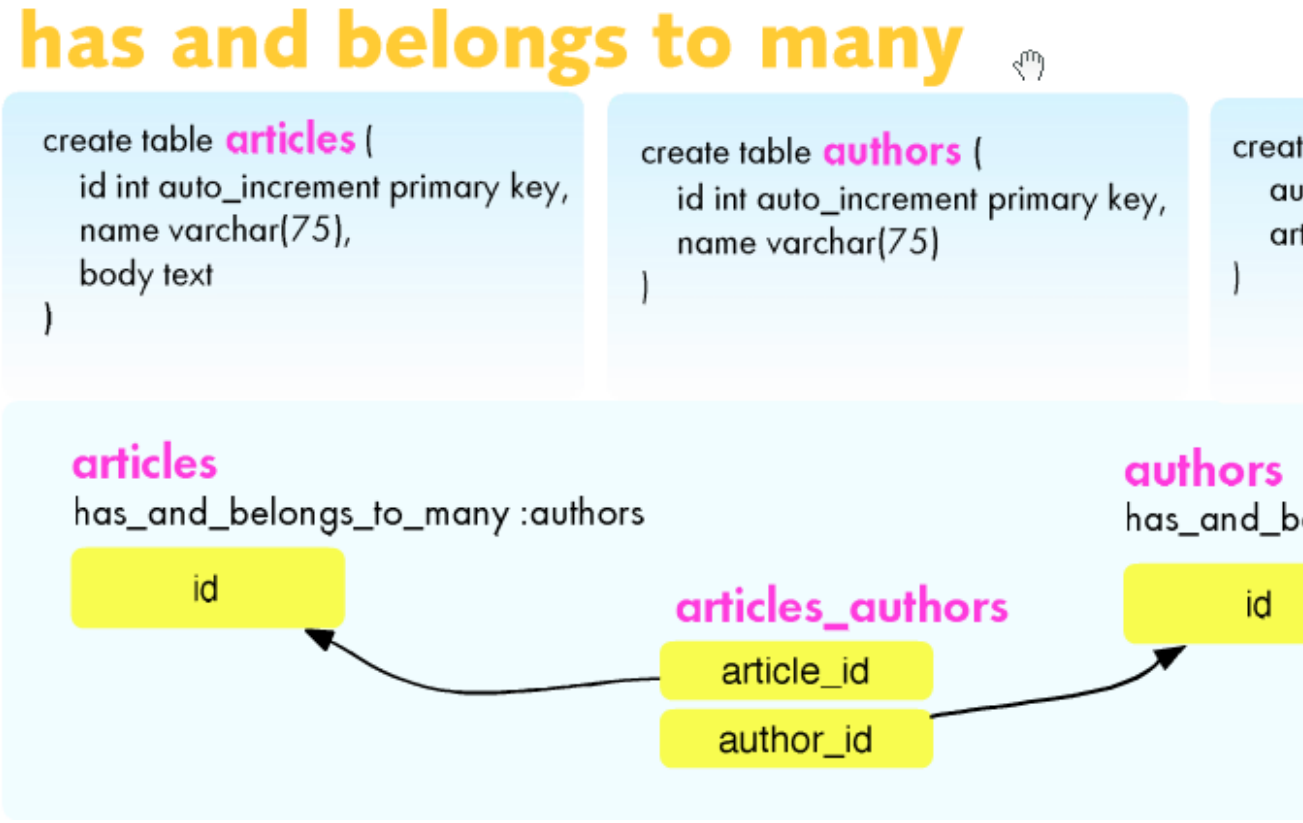
```
class Subscription < ActiveRecord::Base
  before_create :record_signup

  private
    def record_signup
      self.signed_up_on = Date.today
    end
end
```

3.3. Relaciones entre tablas

Rails implementa las relaciones entre tablas mediante etiquetas:

- `:has_one`
- `:has_many`
- `:belongs_to`
- `:has_and_belongs_to_many` (para relaciones de muchos a muchos, tablas pivote)





Al definir las relaciones entre tablas se crean una serie de metodos de instancia muy útiles en su posterior manejo.

Una ayuda muy útil es la que nos proporciona para mantener un contador automático y evitarnos constantes consultas del tipo "select count(*)"mediante el uso de un campo `count`.

Otra de las ventajas de Rails es la herencia. Al trabajar con objetos podemos aprovechar la herencia para crear modelos de datos que comparten una única tabla de la base de datos. Por ejemplo: podemos tener una tabla llamada vehículos y crear nuestros modelos de datos de la siguiente forma:

Más información sobre **relaciones entre tablas** en *http://www.railsforum.com/viewtopic.php?id=265*

3.4. Herencia

class Vehiculo <ActiveRecord::Base Esto generaría el modelo de datos maestro. Gracias a la herencia ahora podemos hacer:

Turismo <Vehiculo

Furgoneta <Vehiculo

Rails se encarga de diferenciar automáticamente los diferentes tipos de registros (Turismos, Furgonetas) gracias al empleo de un campo type (que habremos de definir en la tabla maestra).

3.5. Rutas

Capítulo 4

Utilizando plugins y engines

Capítulo 5

Material adicional

5.1. Enlaces de interés

- Sitio Oficial de Ruby On Rails
  http://www.rubyonrails.org/
- Wiki Oficial de Ruby On rails
  http://wiki.rubyonrails.org/rails
- RubyForge
  http://rubyforge.org/
- RubyCorner
  http://www.rubycorner.com/
- Planet Rails
  http://www.planetrubyonrails.org/
- Top 30 Ruby on Rails Tutorials
  http://webdeveloper.econsultant.com/ruby-rails-tutorials/
- Meshflex - The Tutorial Database
  http://www.meshplex.org/wiki/Ruby/Ruby_on_Rails_programming_tutorials
- Try Ruby!
  http://tryruby.hobix.com/
- How to use Rails with Subversion
  http://wiki.rubyonrails.org/rails/pages/HowtoUseRailsWithSubversion
- Rails Plugins
  http://agilewebdevelopment.com/plugins
- Desarrollo REST con Rails
  http://www.b-simple.de/download/restful_rails_es.pdf

5.1.1. Screencasts/Podcasts

- Railscasts - a free Ruby On Rail Screencasts
  http://railscasts.com
- PeepCode
  http://peepcode.com/
- Rails Envy
  http://www.railsenvy.com/podcast
- Rails Podcast
  http://podcast.rubyonrails.com/
- Odeo
  http://odeo.com/find/rails

5.1.2. Lista de opciones de Rake

- rake cache:clear ->Clears all cached pages
- rake db:bootstrap ->Loads a schema.rb file into the database and then loads the initial database fixtures.
- rake db:bootstrap:copy_default_theme ->Copy default theme to site theme
- rake db:migrate ->Migrate the database through scripts in db/migrate. Target specific version with VERSION=x
- rake db:schema:dump ->Create a db/schema.rb file that can be portably used against any DB supported by AR
- rake db:schema:load ->Load a schema.rb file into the database
- rake db:bootstrap:load ->Load initial database fixtures (in db/bootstrap/ *.yml) into the current environment's database. Load specific fixtures using FIXTURES=x,y
- rake db:fixtures:load ->Load fixtures into the current environment's database. Load specific fixtures using FIXTURES=x,y
- rake db:sessions:clear ->Clear the sessions table
- rake db:sessions:create ->Creates a sessions table for use with CGI::Session::ActiveRecordStore
- rake db:structure:dump ->Dump the database structure to a SQL file
- rake db:test:clone ->Recreate the test database from the current environment's database schema
- rake db:test:clone_structure ->Recreate the test databases from the development structure
- rake db:test:prepare ->Prepare the test database and load the schema
- rake db:test:purge ->Empty the test database

- rake deploy ->Push the latest revision into production using the release manager
- rake diff_from_last_deploy ->Describe the differences between HEAD and the last production release
- rake doc:app ->Build the app HTML Files
- rake doc:clobber_app ->Remove rdoc products
- rake doc:clobber_plugins ->Remove plugin documentation
- rake doc:clobber_rails ->Remove rdoc products
- rake doc:plugins ->Generate documation for all installed plugins
- rake doc:rails ->Build the rails HTML Files
- rake doc:reapp ->Force a rebuild of the RDOC files
- rake doc:rerails ->Force a rebuild of the RDOC files
- rake edge ->freeze rails edge
- rake log:clear ->Truncates all
- .log files in log/ to zero bytes
- rake rails:freeze:edge ->Lock to latest Edge Rails or a specific revision with REVISION=X (ex: REVISION=4021) or a tag with TAG=Y (ex: TAG=rel_1-1-0)
- rake rails:freeze:gems ->Lock this application to the current gems (by unpacking them into vendor/rails)
- rake rails:unfreeze ->Unlock this application from freeze of gems or edge and return to a fluid use of system gems
- rake rails:update ->Update both configs, scripts and public/javascripts from Rails
- rake rails:update:configs ->Update config/boot.rb from your current rails install
- rake rails:update:javascripts ->Update your javascripts from your current rails install
- rake rails:update:scripts ->Add new scripts to the application script/ directory
- rake remote_exec ->Execute a specific action using the release manager
- rake rollback ->Rollback to the release before the current release in production
- rake show_deploy_tasks ->Enumerate all available deployment tasks
- rake stats ->Report code statistics (KLOCs, etc) from the application
- rake test ->Test all units and functionals
- rake test:functionals ->Run tests for functionalsdb:test:prepare

- rake test:integration ->Run tests for integrationdb:test:prepare
- rake test:plugins ->Run tests for pluginsenvironment
- rake test:recent ->Run tests for recentdb:test:prepare
- rake test:uncommitted ->Run tests for uncommitteddb:test:prepare
- rake test:units ->Run tests for unitsdb:test:prepare
- rake tmp:cache:clear ->Clears all files and directories in tmp/cache
- rake tmp:clear ->Clear session, cache, and socket files from tmp/
- rake tmp:create ->Creates tmp directories for sessions, cache, and sockets
- rake tmp:pids:clear ->Clears all files in tmp/pids
- rake tmp:sessions:clear ->Clears all files in tmp/sessions
- rake tmp:sockets:clear ->Clears all files in tmp/sockets
- rake update_dialog_helper ->Copies the latest dialog.js to the application's public directory

Bibliografía

[DN08] Derek DeVries and Mike Naberezny. *Rails for PHP Developers*. Pragmatic Programmers, 2008.

[Fow06] Chad Fowler. *Rails Recipes*. Pragmatic Programmer, Junio 2006.

[HG07] Stuart Halloway and Justin Gehtland. *Rails for Java Developers*. Pragmatic Programmer, 2007.

[Ors07] Rob Orsini. *Rails Cookbook (Cookbooks (O'Reilly))*. O'Reilly Media, Inc., January 2007.

[Ray07] Scott Raymond. *Ajax on Rails*. O'Reilly Media, Inc., January 2007.

[TDHHS06] Dave Thomas, Mike Clark James Duncan Davidson Justin Gehtland David Heinemeier Hansson, with Leon Breedt, and Andreas Schwarz. *Agile Web Development with Rails*. Pragmatic Bookshelf, segunda edition, Diciembre 2006.

[TFH04] Dave Thomas, Chad Fowler, and Andy Hunt. *Programming Ruby: The Pragmatic Programmers' Guide, Second Edition*. Pragmatic Bookshelf, October 2004.