Ruby on Rails 4

para principiantes

sobre

Windows y Ubuntu

Sergio Alonso

¡Ud podría estar leyendo una copia vieja!

Chequee la versión en el encabezado de las hojas, contra la **última** en el sitio oficial,

http://r3uw.heroku.com

Desde allí encontrará el vínculo hacia la versión mas actual, y un listado muy útil de erratas, correcciones, e incluso una sección donde postear errores y sugerencias.

Este libro es tan gratis como una cerveza (free-as-a-beer). Cuente siempre con la última versión disponible en PDF. Mi objetivo es que lo lea la mayor cantidad posible de personas, y que éstas me retornen sus impresiones, errores encontrados y sugerencias en general.

De hecho, necesito de estos reportes para obtener un nivel apreciable de calidad, tal que el libro sea digno de ser llevado a una imprenta.

Una vez que el libro este listo, pretendo seguir dejándolo libre, publicarlo en la editorial Bubok (impresión a demanda), y quedarme como mucho con u$s 5 de regalías por libro... al menos para hacerle creer a mi esposa que hago dinero con esto. Creo que si el libro lo merece, la gente se cansará de leerlo en pantalla, y querrá tener un ejemplar en papel.

Además, le daré la oportunidad a los lectores de tener el libro en papel, y de convertirse en la envidia de compañeros y amigos.

Sergio A. Alonso

[Portada: Verónica V. Martínez]

[Copyleft 2015 Sergio A. Alonso]
[E-mail / MSN: sergio (at) eim.esc.edu.ar]
[Twitter: @karancho]
[http://ar.linkedin.com/in/alonsosergio]
[Blog: http://bunker-blog.blogspot.com.ar]
[http://about.me/elbunker]

[Hi, I am the root here. If you see me laughing,]
[you'd better have a backup ready.]

Índice de contenido

Introducción a la presente obra

Mis estimados lectores: les agradezco la apertura de estas paginas. Las mismas han sido confeccionadas con cariño, revisadas cientos de veces en buscas de errores e incompatibilidades, y espero sinceramente cumplan vuestras expectativas.

La confección de este libro apunta principalmente a dotar de documentación en español a aquellos desarrolladores que han decidido disfrutar de la programación Web. Efectivamente, lo único que le quita diversión a Rails, es *tener que leer la bibliografía en inglés*.

Lo cual es una paradoja, puesto que Rails ha sido concebido para hacer felices a los programadores: sin importar cuan veloces seamos leyendo en la lengua anglosajona, el factor de placer se reduce. Y aunque sea poquito... ya saben. Con el placer no se juega. A tales efectos, lo he redactado de tal manera, que se lo pueda maridar con una buena cerveza fría, de preferencia negra. No cualquiera, ojo.

OS Wars Disclaimer

```
<antiflama mode="activado">
```

Otra razón por la cual he acometido la confección de este libro, es acercar a los usuarios de Windows y Linux hacia este framework. Particularmente a los usuarios de Windows, porque todavía les cuesta mucho salir de los entornos unificados, centralizados y -al menos para mi- agobiantes. La descentralización del entorno de trabajo es una de las mejores experiencias que pueda tener un programador de espíritu libre.

Mediante los ejemplos, espero animarlos a que prueben Rails bajo Linux. La diferencia es mínima. Por cierto, el enfoque apunta a usar Ruby en una distribución abanderada por aquellos novatos que recién empiezan con el sistema operativo del pingüinito, el amable y confiable Ubuntu. Y no quiero dejar de lado algunas instrucciones para sus curtidos administradores, quienes probablemente vayan a alojar nuestra aplicación cuando esta comience a tambalearse bajo el peso de las consultas. Es decir, cuando sea rentable.

Deliberadamente, he dejado de lado a OS/X. Una pena, puesto que se ha erigido en estos últimos años como la panacea que junta lo mejor de ambos mundos. Sin embargo, no me siento culpable. Sucede que el tercer mundo, de donde provengo, una Mac cuesta el

equivalente a medio año de sueldo. Y por otro lado, ya hay mucha, y muy buena documentación: la comunidad de este sistema operativo ha sido quien más ha adoptado a Rails. Por supuesto, los ejemplos también son válidos para ellos.

Para no llenar de hojas extras el presente volumen, también he dejado de lado un sistema operativo que me gusta mucho: FreeBSD.

Es frecuente encontrar en las convenciones de rubystas, en los Google Summer of Code, y otros eventos, a varios programadores trabajando en un mismo proyecto. Si miran las pantallas de las notebooks, con sorpresa notaran que cada uno trabaja en el sistema operativo que se le antoja. Los mantiene unidos, en primer lugar, la fantástica herramienta "gems". Y en segundo lugar, algún manejador para gestionar concurrencia de versiones, como GIT. Ambas herramientas serán utilizadas frecuentemente en la presente obra.

Everybody loves Screenshots

Tiendo mucho a ejemplificar mediante capturas de pantalla. No solo ejemplifican muy bien, sino que demuestran *que alguien pudo hacer funcionar el ejemplo*[1].

Podrán notar que las capturas de pantalla alternan todo el tiempo entre Windows XP, Seven, 2003 server, y Ubuntu, e incluso entre varios windows managers (Gnome, KDE, Fluxbox). Y para completarlo, entre varios navegadores con distintos temas.

Debo confesar que la construcción de este libro fue realizada en cada maquina donde tuve un rato libre para sentarme. La plasticidad y compatibilidad de Ruby me permitió instalar gemas sobre toda clase de sistemas operativos. Mediante GIT, sincronizaba el código no importa donde estuviera. Basta unos instaladores a mano, o apt-get en Linux, y listo.

Finalmente, dejé las capturas así como la podrán ver. La idea es demostrar la capacidad que tendrá nuestra aplicación para ejecutarse indistintamente sobre cualquier plataforma.

1 A menos que de puro malvado, y mediante Gimp, muestre cosas que en realidad no funcionan... ¡muajajajah!

¿Por qué utilizar Ruby para escribir páginas Web?

Se me ocurre una multitud de razones. Para empezar, Ruby es un lenguaje nacido en la era de Internet. No tuvo que adecuarse a nuevos paradigmas ni incorporar funciones en forma confusa y apresurada. Es completamente libre, lo cual garantiza un buena cantidad de mantenedores y auditores del código. Y como sus ancestros conceptuales, hereda muchas buenas ideas, tales como la orientación a objetos, manejo de expresiones regulares, y varias funciones que veremos mas adelante[(2)].

Un aspecto destacable, es que Ruby es el lenguaje ideal para crear aplicaciones para la **Web 2.0**.

Este término, **Web 2.0**, es un concepto amplio y abstracto. Y es mas fácil entenderlo observando sus consecuencias actuales. Yendo a lo general, apunta a la idea de sitios colaborativos, compuestos por muchas personas (**y otros sitios**) interactuando. Las redes sociales son un buen ejemplo. La Web 2.0 tiene también un comportamiento visible, o mejor dicho, invisible, ya que esconde a la verdadera Web, e incluso Internet, para el usuario ocupado de hoy[(3)].

Ya quedaron en los 90 las primitivas interfaces de antaño, que delataban la sincronicidad de la Web, en la cual todo botón o evento de tipo **submit** ocasionaba la recarga completa de la página.

También van en disminución aquellas compuestas exclusivamente por Flash, con excepción ocasional de aquellos que han sido capaces de entender a tiempo a Flex y el protocolo MXML.

Hoy se apunta a un mezcla, un rescate de los controles originales <**input**>, pero con un **#toque** amigable, intuitivo, tal que el usuario apenas se de cuenta de la permanente necesidad que tiene la interface **cliente**, de comunicarse con la aplicación **servidora**.

Para lograrlo, la Web 2.0 se apoya en estándares, software libre, documentación abierta, y en general respeto a rajatabla por los 7 niveles del modelo OSI.

Sin embargo, toda esa interactividad invisible, no es fácil de lograr. Para ser exacto, es muy difícil de compaginar, debido a la cantidad de tecnología que forma la mezcla. El usuario, impaciente como un hámster seleccionando semillas de girasol, no tiene idea de la cantidad de cosas que se encadenan cada vez que hace click.

2 Ruby tiene mas abuelos que un club de jubilados. Hasta donde pude saber, incluye a los venerables: Perl, Eiffel, LISP, Smalltalk, y el asombroso *new kid on the block*, Python.

3 Si, estoy siendo sarcástico.

Hasta este punto, Ruby no difiere tanto de otros lenguajes para la Web. Armar una interface Web 2.0 puede tomar un tiempo apenas menor.

¿Cual es la diferencia?
La diferencia estriba en que los programadores Web, tarde o temprano deben aprender a usar un framework. Programar comportamientos enriquecidos a mano puede llegar a ser desquiciante. Los frameworks, en cambio, poseen librerías, scripts, patrones de diseño, que se encargan de lidiar con ciertos aspectos cuyas particularidades entorpecen el placer de la creación.
Y los frameworks de Ruby son los más fáciles de aprender del mercado. Todos ellos nos permiten (y obligan) a hacer uso de standares de la Web 2.0, tales como CSS, Ajax, XML, Json, YML, por mencionar solo algunas. De esta manera, es relativamente simple interactuar con Flex (Flash), APIs de los principales clouds, y bindings de todo tipo hacia Java, .NET y C++. También se lleva muy bien con toda clase de métodos que guardan o cachean información, tanto del lado servidor (SQL, NOSQL) como del lado cliente (SQLite, AIR).
Y como si todo esto fuera poco, la sintaxis de Ruby es natural, se lee más fácil, es original, limpia, atrevida. Por estas razones tildan a los programadores de Ruby de *metrosexuales,* lo cual quizás no esté exento de razón.

Mas piedras preciosas

Otras características que podemos encontrar en Ruby:

- Soporta multihilos... ¡hasta en MSDOS!
- Orientado completamente a objetos.
- Permite intervenir los objetos en tiempo de ejecución.
- Parecido en algunos aspectos al genial Python, pero a diferencia de éste, Ruby hasta el 2000 no estaba traducido al inglés
- Si, ya lo dije. Pero lo digo de nuevo: posee bindings hacia
  - C++
  - Java
  - .NET
  - Python
  - Tk
- Maneja excepciones.

- Es capaz de documentar una aplicación con solo usar el comando rdoc.
- Posee acceso a la documentación instalada del sistema, mediante los comandos **ri** y **fxri.**
- Posee una comunidad muy grande de desarrolladores, los cuales todo el tiempo fabrican clases y objetos asombrosos. Parecen personas muy felices compartiéndolo todo. Lo son, desde que les concedieron asilo político... cuando arribaron desde las ásperas costas de Java y C++
- Los programas de Ruby corren con la misma sintaxis en la mayoría de los sistemas operativos conocidos. Solo hay que cuidar que cuenten con las mismas librerías, o al menos, librerías (gemas) portadas al sistema operativo que estemos usando. Tambien esta es la razón por la cual Ruby es óptimo para realizar scripting e instrucciones de deploy.
- Finalmente, su portabilidad hacia varios sistemas operativos se pude observar en eventos tipo "Summer of Code" y congresos de rubystas, donde es frecuente ver programadores de Ruby conectados vía GIT / SVN, desde sus equipos con Windows, MAC OS/X, Linux, e incluso desde celulares corriendo Android, coexistiendo pacíficamente y colaborando juntos.

Capítulo 1

Capítulo 1: Instalación y Creación de un Marco de Trabajo

Entre las muchas discusiones que dividen a los programadores, hay una en particular que se distingue por su pasión: la integración del entorno de trabajo.

Hay programadores que se sienten cómodos dentro de entornos que le provean absolutamente todo. Entre ellos podemos encontrar a los usuarios de Visual Studio, Eclipse o NetBeans (Java).

Otros programadores, en cambio, prefieren la desmodularización de las partes, es decir, componentes pequeños, en lo posible libres y gratuitos.

Si adivina correctamente, el mundo de Ruby se construye desde el segundo enfoque. La naturaleza y libertad de Ruby ocasiona que existan muchas opciones para realizar una misma tarea. Si bien existen IDEs con mucha suma de partes, tales como "Eclipse" Aptana Studio o Rubymine, con el tiempo coincidirá conmigo en que un liviano editor y unas cuantas herramientas es todo lo que hace falta.

En este capítulo instalaremos Ruby, un manejador de paquetes llamado GEM, y a través de GEM, la gema Rails. También le recomendaré otras piezas de software que me han sido de mucha utilidad.

Espero que con ellas construya un marco agradable de trabajo, personalizado a su gusto. Por si quiere darse una vuelta, este el mío:
http://www.eim.esc.edu.ar/bunker.org.ar/incubadora.varios/ruby/

Aclaraciones previas

GEM

Es importante aclarar un aspecto que confunde a los novatos en Rails. *Este framework es una gema de Ruby*. Por lo tanto, debemos tener instalados tanto Ruby como GEM.
GEM es un inteligente instalador de paquetes, utilidades y librerías. Es análogo al gestor de paquetes apt-get de Debian / Ubuntu, pero funciona sobre cualquier sistema operativo que esté soportado por Ruby.

Bases de Datos

¿Cuál SQL estoy usando?
Cualquiera, en cualquier momento de las tres fases de desarrollo. Incluso, en instalaciones realmente grandes, podría no usar SQL[4].
Los programadores de Ruby suelen usar SQLite para las etapas Development y Testing, en sus propias maquinas de desarrollo, por un tema de comodidad y ahorro de recursos.
Cuando pasan a producción, suben la aplicación al servidor mediante GIT (o Mercurial, Subversion, etc) + Capistrano, y *deployan* sobre algún motor mas grande, como MySQL, PostgreSQL, Oracle, o MSSQL. Por supuesto, este comportamiento no es obligatorio, pero si es el mas recomendado por la comunidad. Igualmente, veremos como instalar servidores de producción tanto sobre Windows como sobre Ubuntu Linux.
Como habíamos mencionado, ActiveRecord se encarga de lidiar con las diferencias de sintaxis SQL propias de cada motor.

Próximamente veremos como el componente **Vista** de Rails, el **ActiveRecord**, busca que no lidiemos con las **particularidades** de las bases de datos. Como toda librería ORM, *trata de aislarnos* de las maneras propias y particulares en que "se hacen las cosas" dentro de un motor SQL en particular.

SQLite

SQLite Es un pequeño y ágil motor RDBMS, muy embebible y transportable. Se usa en

4 http://del.icio.us/karancho/nosql

montones de pequeñas aplicaciones y miniframeworks, usualmente del lado del cliente, tales como Google Gears, para las versiones offline de Gmail, Calendar y Reader, o Adobe Air, etc.

Estos lo aplican para cachear datos, bufferizarlos, o mantenerlos offline para posteriores sincronizaciones. Podemos usarlo cómodamente también en el lado del servidor, puesto que no requiere de un daemon ni configuración alguna.

Instalación en Ubuntu / Debian

Primero: Ruby. En Linux Ubuntu, abrimos una terminal, y escribimos la siguiente orden:

No Copie → Pegue del PDF: el formato de destino (ASCII, UTF8, ISO) puede variar en el destino. Escriba cada línea manualmente

- `sudo apt-get install curl gpg`
- `gpg --keyserver hkp://keys.gnupg.net –recv-keys \`
  `409B6B1796C275462A1703113804BB82D39DC0E3`
- `\curl -sSL https://get.rvm.io | bash -s stable`
- `rvm get stable --auto-dotfiles`

Salga de todas las sesiones de terminal, o ante la duda, reinicie la computadora.

Para ver todas las versiones disponibles de Ruby:

```
rvm list known
```

Si bien Ruby va por 2.2-stable, yo prefiero todavía utilizar la última rama estable en 2.0, en la cual debugger y otras gemas se compilan sin problemas. Ejemplo:

```
rvm install --default ruby-2.0.0 --auto-dotfiles
```

Una vez que termine la instalación, revise que tenga estas lineas en su ~/.bashrc o en su ~/.zshrc En mi caso, las agrego al ~/.profile, para que quede disponible tanto para Bash como para Zsh

```
[[ -s "$HOME/.rvm/scripts/rvm" ]] && . "$HOME/.rvm/scripts/rvm"
source ~/.rvm/scripts/rvm
```

Reinicie nuevamente las terminales (o escriba source ~/.rvm/scripts/rvm)

```
rvm --default use ruby-2.0.0
```

Compruebe que tiene la versión correcta escribiendo

```
ruby -v
=> ruby 2.0.0p598
```

Si así no fuera, revise la variable PATH

```
echo $PATH
```

Ruby podría estar siendo servido por los paquetes del propio Linux, el cual muchas veces viene con Ruby incluido, o es instalado automaticamente por solicitud de otros paquetes, como vim-gnome. Es fácil darse cuenta, ya que al final del path aparece

```
=> /usr/bin/ruby
```

… en lugar de la que hemos instalado vía RVM, algo así:

```
=> /home/USUARIO/.rvm/gems/ruby-2.0.0-p598/bin
```

Otra pista para darse cuenta, es lanzar el comando ruby y ver cual contesta:

```
ruby -v
=> ruby 2.0.0p598 (2015-02-25 revision 49749) [x86_64-linux]
```

Otra pista es usar el comando which

```
which ruby
=> /home/s/.rvm/rubies/ruby-2.0.0-p598/bin/ruby
```

Estos path se pueden forzar agregando la ruta del binario a nuestra instalación: algo así (comprobarla primero) en los archivos ~./bashrc (para el shell Bash) ~/.zshrc (para el shell Zsh), es decir.

```
export PATH="$PATH:$HOME/.rvm/rubies/default/bin"
```

El único detalle de este truco es que requiere de reiniciar el modo gráfico.

Problemas con el Proxy?

En Linux hay un seteo general en el que todas las aplicación se fijan, ubicado en **Sistema → Preferencias → Proxy de la Red.** Alternativamente, puede configurar CURL, el gestor de descargas que usan varias de las aplicaciones, editando el archivo **~/.curlrc** y agregando adentro la variable proxy. A modo de ejemplo:

```
proxy = proxy.mendoza.gov.ar:8080
```

Otros programas como RVM o GEM también pueden tomar valores proxy desde línea de comandos. Ejemplo:

```
gem1.8 install --http-proxy http://proxy.mendoza.gov.ar:8080 rails
```

Instalar GEM

Necesitaremos la utilidad **gem** para instalar varios componentes. En Ubuntu, **RVM lo debería haber dejado instalado para nosotros.** Otras alternativas (no recomendadas) en Linux Ubuntu es instalar la herramienta desde los repositorios, mediante el conocido **apt-get install gem**. Otra alternativa, tampoco recomendada, es seguir los siguientes pasos:

1. Bajar la ultima versión de Gem desde
   http://rubyforge.org/projects/rubygems

2. Descomprimir el archivo en una carpeta

3. Ejecutar

```
ruby setup.rb
```

```
sudo ruby setup.rb
```

4. Si gem ya está instalado, puede actualizarlo mediante

```
gem update --system
```

```
gem update --system
```

5. Probemos instalando una librería necesaria:

```
gem install execjs -V
```

```
gem install execjs -V
```

Framework Javascript

Rails necesita de algún interprete Javascript instalado en algún lado. Hay dos maneras de satisfacerle tal necesidad.

1) Mediante Ubuntu:

```
sudo apt-get install nodejs
```

2) O bajando directamente la gema al sistema

```
gem install therubyracer
```

Ahora si: Rails 4

```
gem install rails -V
```

Rails 4 instala en forma automática varias dependencias propias, y un componente llamado **bundle. Bundle** revisa el **Gemfile** de cada proyecto, y genera una instalación de las gemas que se encuentren indicadas adentro. Es muy útil para cuando agregamos posteriormente gemas al proyecto. Si cambiamos de equipo, basta con correr dentro del proyecto (no hace falta que lo haga ahora):

```
bundle install
```

Y de esta manera, la otra computadora tendrá también las gemas necesarias. Esto es muy útil cuando copiamos la aplicación al server de producción. Por supuesto, "shit could happens", pero es una ventaja enorme respecto de otros frameworks que no tienen siquiera pensado como migrar todo el proyecto entre cambios de librerías, o de la versión del lenguaje usado.
En el siguiente ejemplo, si estamos usando Rails con una versión menor a 3.1, el paso 3 es manual.

1)	`rails new cualquiera`	Fabrica el proyecto
2)	`cd cualquiera`	Entra al proyecto
3)	`bundle install`	Instala librerías necesarias para este proyecto Rails en particular

Eso es todo. Podemos borrar el proyecto *cualquiera*, ya que poseemos todas las gemas

necesarias en el sistema para empezar a trabajar. Este pequeño aprendizaje es muy útil para cuando construya una aplicación dependiente de sabrosas gemas que obtenga en internet, y desee que cuando el proyecto suba a un server de producción, por ejemplo Heroku. Usted *debería anotar en el archivo Gemfile* todas las gemas que se vaya bajando de internet, a fin que durante el proceso de deploy remoto, Rails solicite que le satisfagan las mismas gemas que tiene en la maquina de desarrollo.

Una vez terminada la instalación, conviene revisar la versión:

```
rails -v
```

Antes de comenzar a probar, decida que tipo de base de datos desea usar, y avance unos capítulos para chequear la instalación necesaria.

Rails on the Edge

Si quiere adelantarse a la próxima versión, o cubrirse de gloria contribuyendo con parches, como estos abnegados muchachos http://contributors.rubyonrails.org/edge/contributors, pase por los siguientes vínculos:

- Lista de parches a resolver: cuando esta cifra llegue a cero, probablemente los mantenedores del proyecto dictaminen que Rails 5 será la nueva versión estable.
  https://github.com/rails/rails/issues
- Aquí puede obtener la última versión del código, con correcciones, parches y novedades diarias.
  https://github.com/rails/rails
  Tenga en cuenta que como toda versión "edge" de Rails, es sumamente inestable y se debe usar solo para descubrir las novedades del próximo lanzamiento o "release".
- Una vez que se ha bajado el fuente, ha creado proyectos con él (ver instrucciones mas abajo), y ha descubierto algún bug, siga el siguiente protocolo para enviar su corrección: http://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html

A la fecha que pude probar esta versión, 3 de octubre de 2015, la manera de poner a correr esta versión beta o "edge", es instalar previamente, como requerimiento mínimo, Ruby 2.2.2 y clonar luego el árbol del proyecto, en lugar de obtenerlo con gem como se hace habitualmente con la versión estable.

Durante la instalación de Rails 4, en el capítulo anterior, habíamos preinstalado Ruby 2.0 para correrlo. Esta combinación es muy estable, medido en función de las gemas, que se compilan habitualmente sin errores.
Para no introducir conflictos en nuestro ecosistema, apelamos a la ayuda de RVM. Este script nos permite tener varias versiones de Ruby instaladas en simultáneo, en la misma computadora, y cambiar entre ellas a la orden "rvm use". Los pasos son los siguientes

```
rvm install ruby-2.2.2
rvm use ruby-2.2.2
gem install bundler
```

En Linux Ubuntu y LinuxMint, bajo terminal de tipo gnome-terminal, o terminator, puede hacer falta hacer previamente al comando **rvm use**, un **bash -login**.
Para dejar la terminal previamente configurada con esta opción, visite este artículo creado a tal efecto por el creador de rvm, presente en https://rvm.io/integration/gnome-terminal

Para clonar el proyecto del próximo Rails, ejecutamos

```
git clone https://github.com/rails/rails
cd rails
bundle install
```

Ya podemos crear un proyecto con Rails 5. Sin embargo, para no mezclar algúnos archivos como Gemfile, deberíamos crearlo en otra ruta,
Por ejemplo, para crear un proyecto situado un nivel de carpetas hacia arriba, ejecutamos la siguiente instrucción

```
bundle exec rails new ../proyecto-prueba-rails-nuevo –dev
```

Vamos luego al proyecto

```
cd ../proyecto-prueba-rails-nuevo
```

Y ya podemos lanzar algunos comandos como los que aparecen en este libro, que como ya he mencionado, está preparado para la versión anterior, estable, Rails 4.

Al lanzar estos comandos, prestemos atención de no usar el comando rails que está en el PATH, sino el que viene incluido en nuestro proyecto, en su propia carpeta bin. Ejemplo:

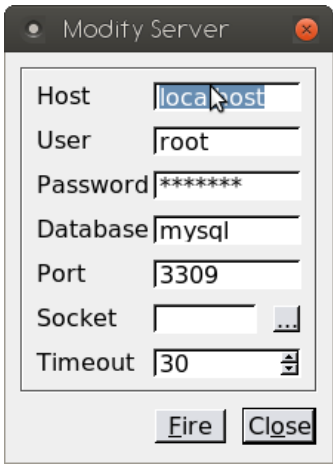
```
bin/rails generate scaffold Products nombre:string cantidad:integer
```

MySQL bajo Linux Ubuntu

¡Cuidado al copiar / pegar!

- `sudo apt-get install make ruby-dev mysql-server libmysqlclient-dev`
- `gem install -V mysql2`

Administradores Opcionales para MySQL:
Para revisar el estado de MySQL, y ocasionalmente administrar sus datos, se puede instalar los siguientes paquetes:




```
sudo apt-get install mysql-workbench
```

También puede bajar de internet la versión portable del magnifico Heidi SQL (http://www.heidisql.com/) para Windows.
Para correrlo en Linux solo necesita ejecutar el archivo heidisql.exe anteponiendo el comando **wine.**

Si extraña a phpMyadmin, puede instalarlo fácilmente mediante

```
sudo apt-get install tasksel
sudo taskel install lamp-server
sudo apt-get install phpmyadmin
sudo ln -s /usr/share/phpmyadmin /var/www/phpmyadmin
```

Luego, busque a phpmyadmin en http://localhost/phpmyadmin

Instalación de SQLite bajo Linux Ubuntu

```
sudo apt-get install sqlite3 libsqlite3-dev libdbd-sqlite3-ruby
gem install -V sqlite3-ruby
```

Administradores para SQLite en Linux

Posiblemente no necesite herramientas extras para administrar la base. Pero quizás desee saber si los datos se están escribiendo y las migraciones corriendo. No obstante recuerde que lo ideal es utilizar para estos menesteres a la consola rails (**rails console**) o la consola del motor sql (**rails dbconsole**). Aquí le presento tres opciones:

```
sudo apt-get install sqliteman sqlitebrowser
```

Alternativamente, le recomiendo instalar además el plugin de Firefox descripto en el apartado "Administradores para SQLite en Windows", unas paginas mas adelante.

Ventanas, en Linux

Si bien en este libro no cubrimos este fascinante tema, siempre hay gente que necesita o desea **ventanas**, al estilo de Visual Basic, sobre las cuales es mas fácil introducir elementos enriquecidos. Inspirado en el magnifico wxPython, tiendo a usar wxRuby, cuya instalación es la siguiente:

```
sudo apt-get install libwxgtk2.8-dev libwxbase2.8-0
gem install -r wxruby
gem environment | grep INSTALLATION
```

En el siguiente vínculo puede encontrar varias guías para desarrollar en poco tiempo aplicaciones enriquecidas: http://wxruby.rubyforge.org/wiki/wiki.pl?WxRuby_Tutorial
Otras muy buenas opciones disponibles en el mercado son TK (la opción oficial), Shoes, QT, y FXRuby (solo para Windows).

RDOC y RI

Quizás usted es de los programadores que no les gusta trabajar en manuales de sus aplicaciones. Manuales que probablemente sus usuarios no leerán. Entonces le alegrará saber que Ruby posee un comando llamado **rdoc**. Al estilo de **jdoc**, este comando, ejecutado en la raíz de la aplicación, puede crearnos automáticamente la mayor parte de la documentación.
También puede instalarlo en Debian / Ubuntu mediante apt-get.

Además, la presencia de este paquete evita algunos molestos warnings durante la instalación de algunas gemas. Alternativamente, puede ejecutar:

- `gem install -V rdoc`
- `gem install -V rdoc-data`

Además, Ruby ofrece una ayuda en pantalla sobre muchas de sus funciones. No es que desconfiemos de Google, pero **ri** (Ruby Information) nos puede desambiguar rápidamente las dudas. Primero, ejecute:

- `sudo rdoc-data --install`
- `gem rdoc --all --overwrite`

Sería buena idea comprobar si la ayuda en pantalla está funcionando. Por ejemplo, si no recordamos la nomenclatura de los métodos, y queremos ver un ejemplo:

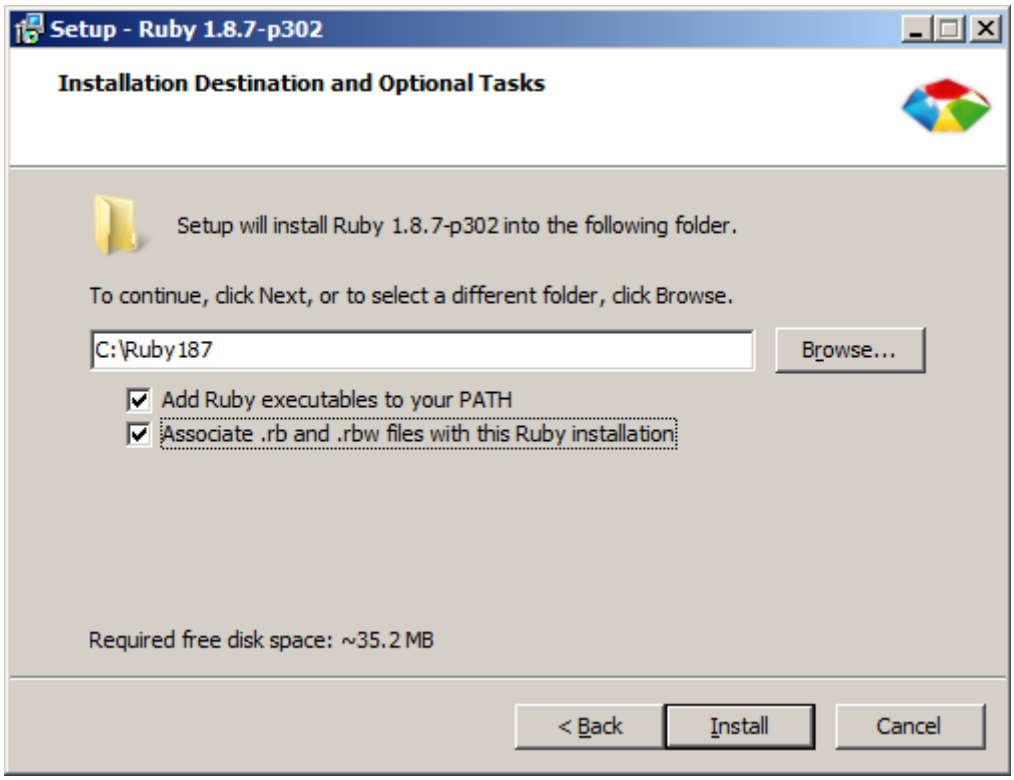
- **`ri methods`**

(**q** para salir del **less** invocado por **ri**)
Esto devuelve una lista de los métodos accesibles públicamente.

```
class Klass
  def kMethod()
  end
end
k = Klass.new
k.methods[0..9]    #=> ["kMethod", "freeze", "nil?", "is_a?",
                       "class", "instance_variable_set",
                       "methods", "extend", "__send__", "instance_eval"]
k.methods.length   #=> 42
```

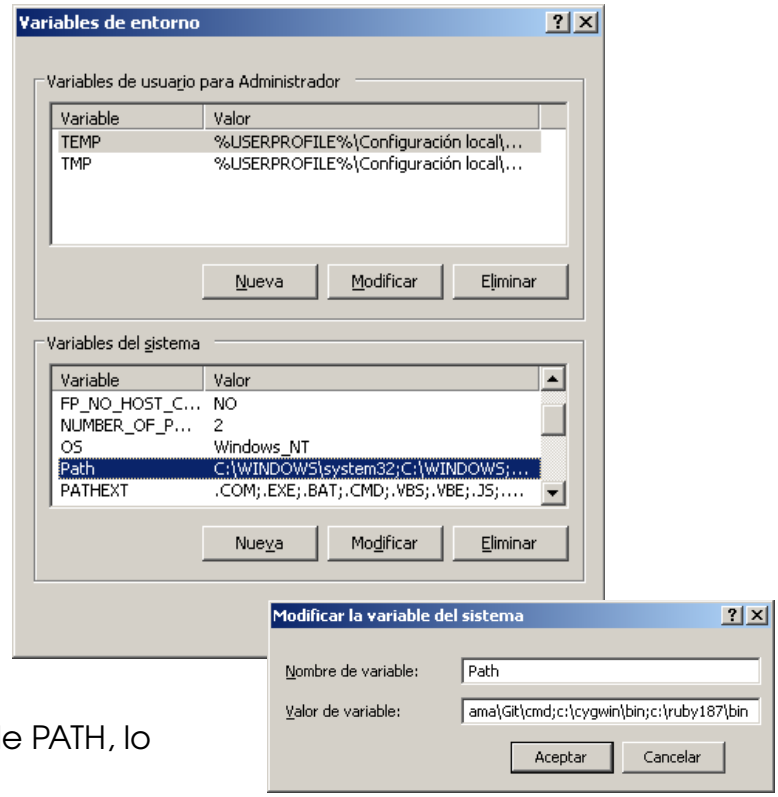
Por cierto, tanto **ri** como **rdoc** pueden ser instalados mediante **apt-get** , pero siempre será mas moderna la versión procedente de **gem.**

Instalación en Windows



Obtenemos Ruby desde www.rubyinstaller.org, proyecto argentino del próximamente legendario, Luis Lavena.

Allí tenemos la opción de bajarnos varias ramas de desarrollo. En términos generales, y a la fecha, setiembre de 2010, la rama 1.8 es mas compatible, en tanto que la 1.9 es mas rápida y provee varios cambios muy interesantes en su API. Retroceda hasta los comentarios sobre la instalación sobre Ubuntu para ver mas detalles.



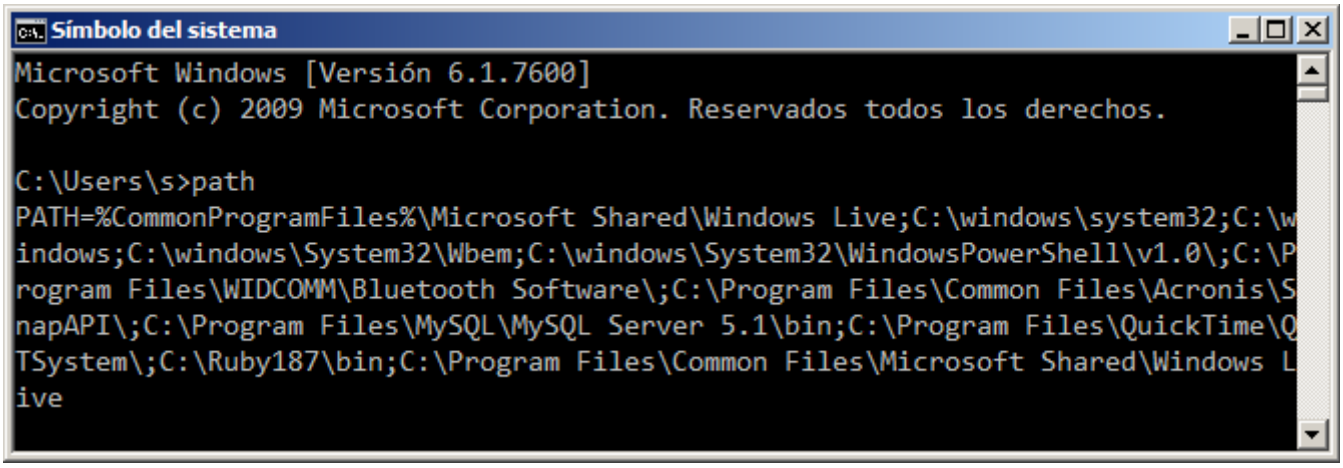
Cuando codeamos en Ruby, lo ideal es hacerlo desde el MSDOS con las variables PATH apuntando a la carpeta **c:\ruby<version>\bin**

Si desea activar manualmente la variable PATH, lo

puede hacer haciendo click derecho sobre **Mi Pc (Equipo)** → **Opciones Avanzadas** → **Variables de Entorno.**

Luego abra una (**nueva**) sesión de MSDOS y confirme escribiendo:

```
path
```

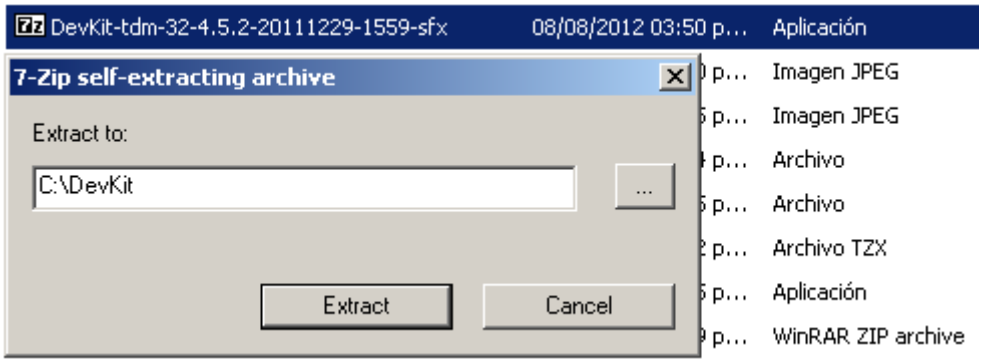


Sin embargo, Ruby Installer ya provee un acceso directo a tal efecto, que puede encontrarse en **Inicio** → **Programas** → **Ruby** → **Start Command Prompt with Ruby**

DevKit

Algunos pasos necesitaran de ciertas librarías instaladas en el sistema llamadas Development Kit. Para ellos bajamos el archivo autoextraible DevKit-tdm-32-x.x.x.sfx.exe de la pagina de Ruby Installer, presente en http://www.rubyinstaller.org/downloads/

Desinstalamos el contenido en alguna carpeta, C:\DevKit por ejemplo.



De acuerdo a las instrucciones de instalación de DevKit, presentes en https://github.com/oneclick/rubyinstaller/wiki/Development-Kit, abrimos el MSDOS, vamos hasta la carpeta donde se encuentra C:\DevKit, y ejecutamos los comandos

1. `ruby dk.rb init`
2. `ruby dk.rb install`
3. `gem install json --platform=ruby`

Este último paso (3) sirve para chequear que las librerías ha sido instalados (2) correctamente en el path de Ruby (1), en nuestro ejemplo, existente en C:\Ruby187

Instalación de GEM y Rails en Windows

RubyInstaller ya trae GEM incluido. Pero conviene actualizarlo. Podemos estar parados en cualquier carpeta y ejecutar:

`gem update --system`

Como primera medida, antes de instalar Rails, proveeremos algunas gemas para gestionar bases de datos: MySQL, SQLite y MSSQL.

SQLite bajo Windows

Para tener un buen soporte de SQLite bajo Windows, conviene entrar a http://www.sqlite.org/download.html y buscar los siguientes archivos:

Precompiled Binaries for Windows

sqlite-shell-win32-x86-3071300.zip (262.59 KiB)	A command-line shell for accessing and modifying SQLite databases. This program is compatible with all versions of SQLite through 3.7.13 and beyond. (sha1: fb04ed658956494b3660b192e4713efc6460dbcc)
sqlite-dll-win32-x86-3071300.zip (297.60 KiB)	This ZIP archive contains a DLL for the SQLite library version 3.7.13 for 32-bit x86 processors using the Win32 API. The DLL is built using SQLITE_ENABLE_COLUMN_METADATA so that it is suitable for use with Ruby on Rails. (sha1: 6985592d33048137fc1097e2826ff1c3458d3843)

Estos archivos deben ser descomprimidos en la carpeta **bin** de donde haya instalado Ruby, ejemplo: **C:\Ruby187\bin**
Además, conviene asimismo obtener el plugin de SQLite para Firefox que se destaca unas páginas mas adelante.

MySQL bajo Windows

MySQL puede ser descargado para Windows desde la página oficial, presente en www.mysql.org. Luego de la instalación, recuerde instalar una gema necesaria:

```
gem install mysql2
```

Una buena idea es buscar el archivo LIBMYSQL.DLL en su carpeta de instalación de MySQL, y copiarlo en c:\ruby<version>\bin

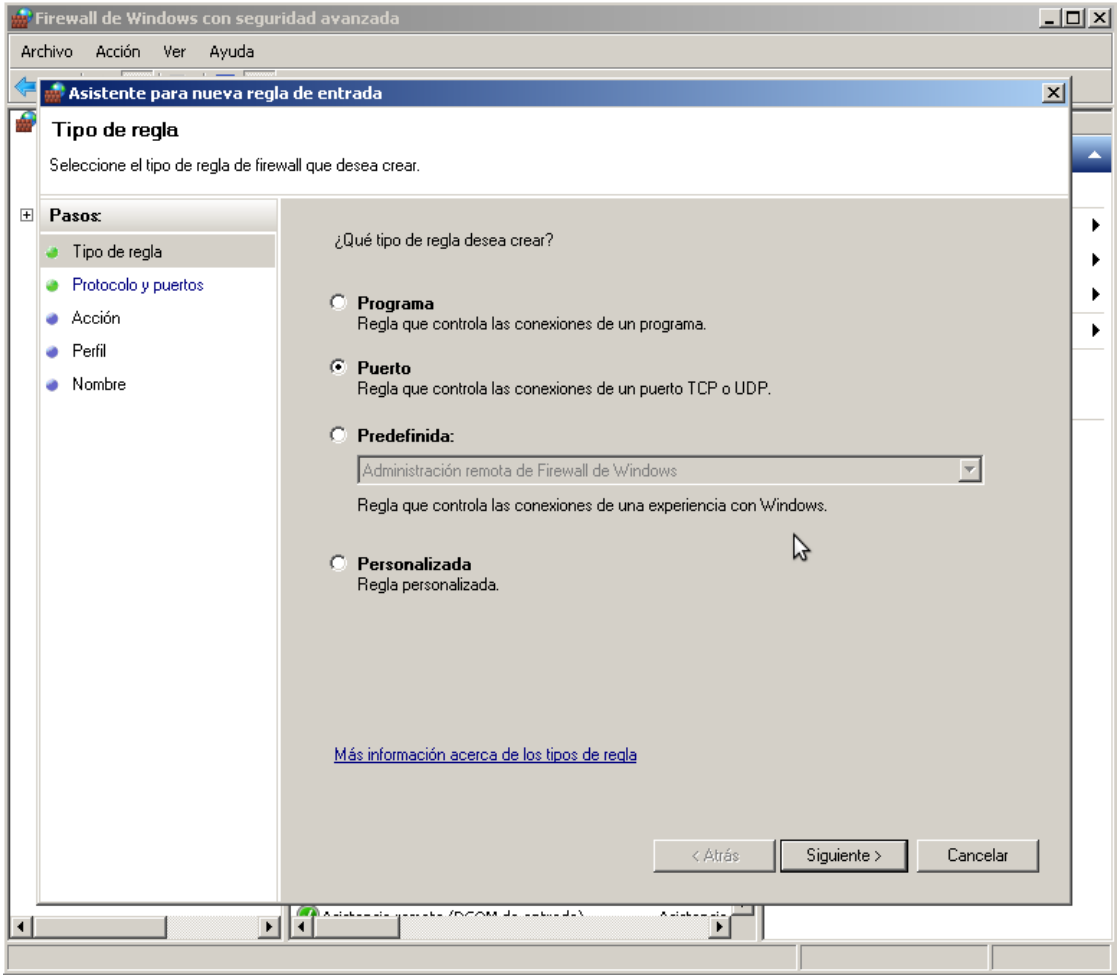
Bajo **Windows Seven** de 64 bits puede hacer falta seguir algunos pasos extras. Nuestro referente sobre Ruby y Windows, Luis Lavena, tiene un muy articulo que trata el tema, presente en http://blog.mmediasys.com/2011/07/07/installing-mysql-on-windows-7-x64-and-using-ruby-with-it/

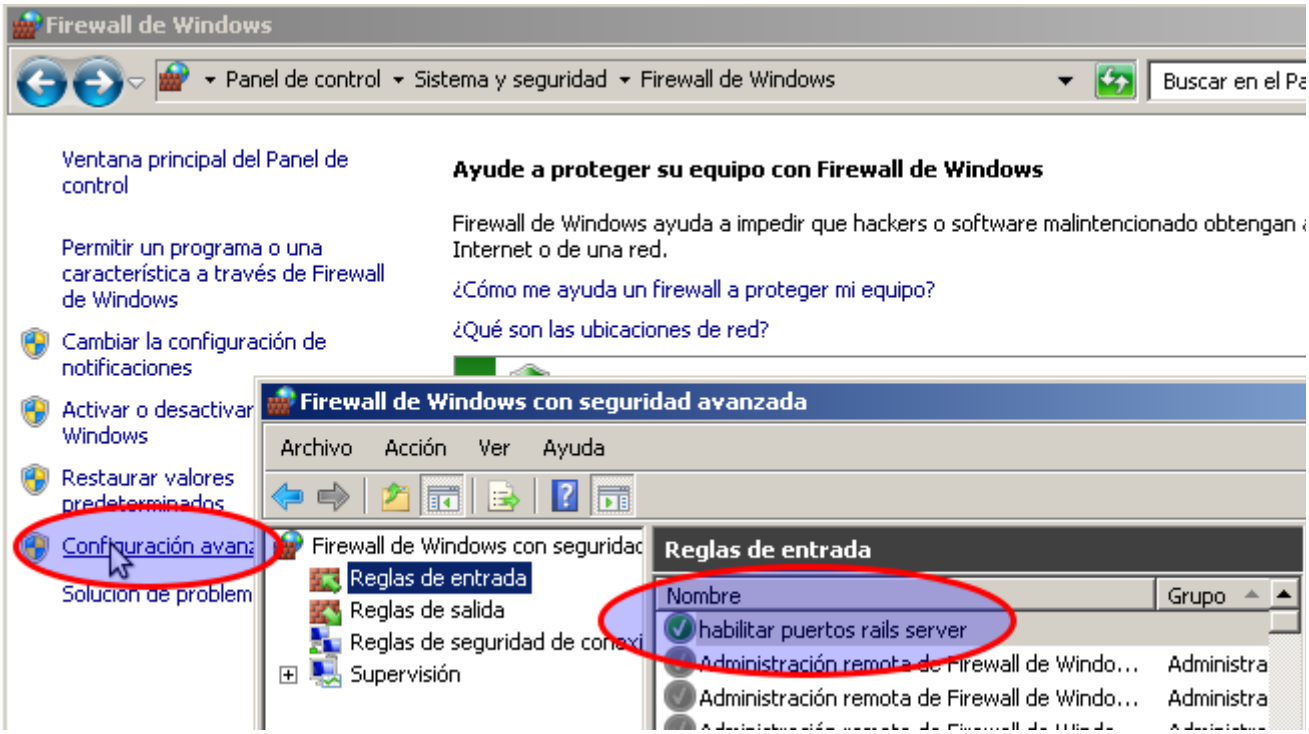
A veces querrá constatar el estado de las bases existentes. Mis recomendaciones:

a) HeidiSQL, presente en www.**heidisql**.com

b) Herramientas oficiales presentes en www.mysql.org: mysql-essentials y mysql-workbench.

Finalmente: no olvide abrir los siguientes puertos en el Firewall: 80, 3000, 3001, 3002, 8000, 3306



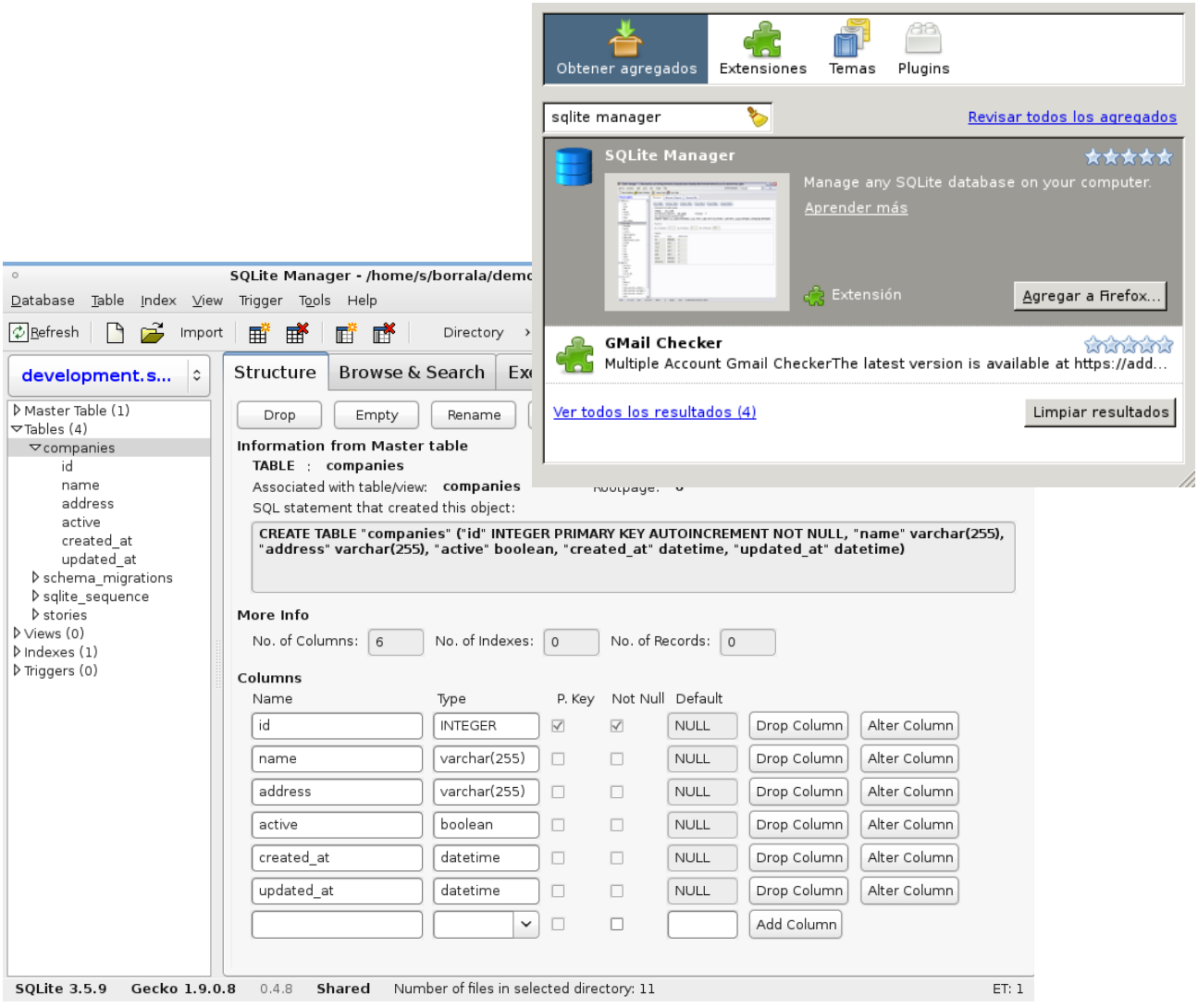


Administradores para SQLite, Windows

Como ya mencioné en el apartado de Linux, posiblemente no necesite herramientas extras para administrar la base. Pero quizás desee saber si los datos se están escribiendo y las migraciones corriendo. No obstante recuerde que lo ideal es utilizar para estos menesteres a la consola rails (**rails console**) o la consola del motor sql (**rails dbconsole**).

Este manager, curiosamente un plugin de Firefox[5], es el mejor de los existentes, y también está disponible en Firefox para Linux.

https://addons.mozilla.org/es-ES/firefox/downloads/latest/5817



5 En realidad no es tan raro que que ciertos plugins para Firefox ofrezcan mejores funcionalidades que programas clásicos: basta solo observar a fireftp o a firebug.

Microsoft SQL Server

Para conectar contra esta base de datos, pasee un momento por mis favoritos:

http://delicious.com/karancho/mssql+rails

Ahora si: Rails 4 en Windows

Los siguientes pasos de instalación deberían proveer una base de dependencias tal que satisfaga a Rails 4. Mucho cuidado al copiar las siguientes líneas:

```
gem install -r rails
```

No se preocupe por el error "File not found lib".

Por cierto, si este comando falla, **insista** varias veces: a veces los servidores se encuentran saturados. Ahora creamos una aplicación cualquiera para lanzar la instalación de gemas que por defecto venga con Rails, concretamente el instalador bundle, que actúa sobre el archivo **Gemfile.**
Para un proyecto simple, con base de datos SQLite:

```
rails new prueba
```

Si preferimos usar MySQL

```
rails new prueba --database=mysql
```

Eso es todo. El proyecto debería estar creado y listo para comenzar.

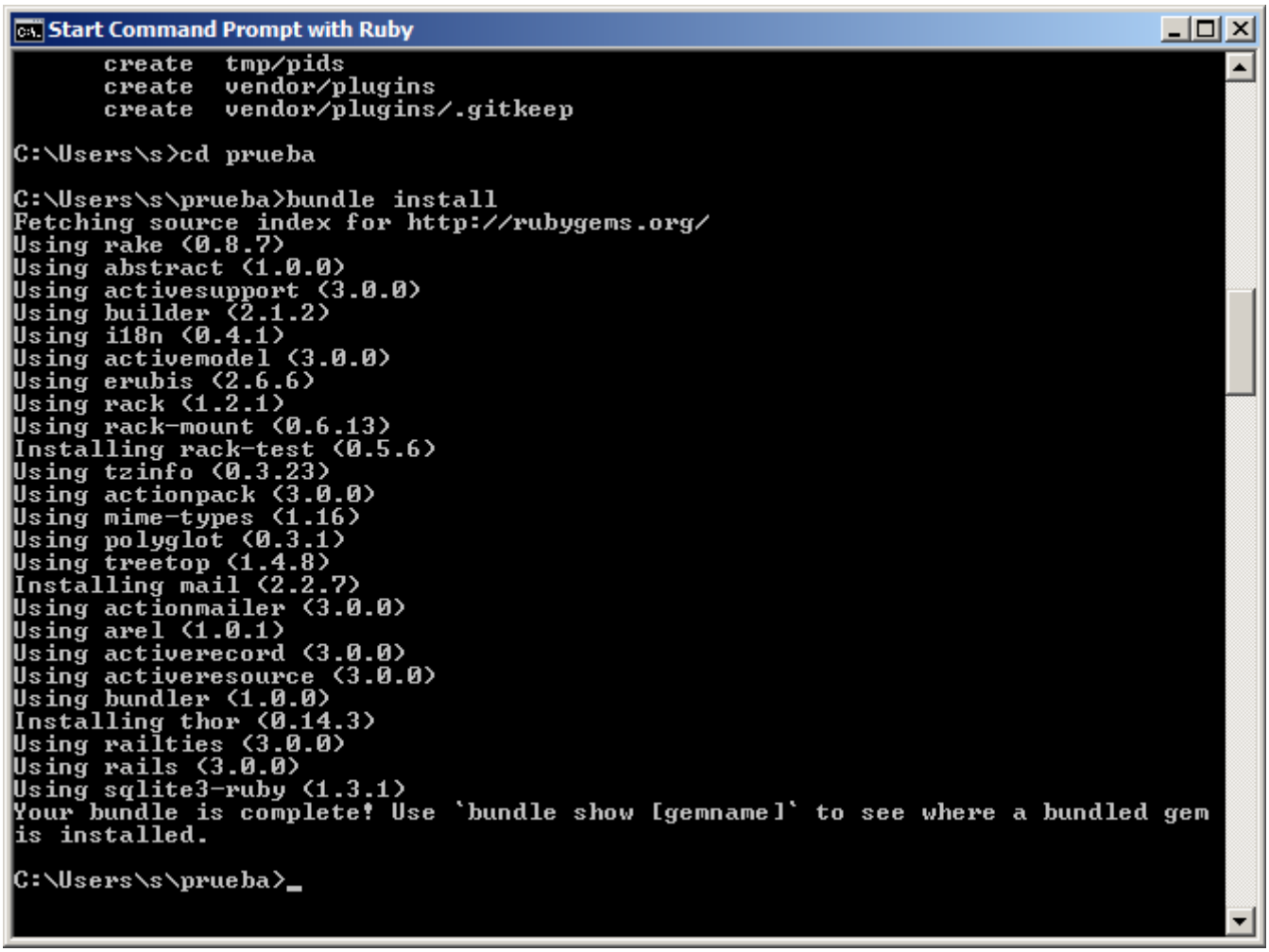
Como único comentario, y si presta atención, durante la creación del proyecto, se ha disparado un script llamado **bundle**, que se encarga un poco a la manera de Maven, de bajar las librerias necesarias para que el proyecto funcione. Si usted debiera incluir alguna librería extra, debe hacerlo dentro del archivo **Gemfile.**
Ejemplo: si hubiéramos solicitado soporte de MySQL, probablemente en **Gemfile** figure una línea:

```
gem 'mysql2'
```

Luego que ese archivo es alterado, se debe correr manualmente:

```
bundle install
```

- Si hemos escogido MySQL, debemos editar los campos *password* de config/database.yml, antes que comencemos con las migraciones.



Una vez terminada la ejecución de bundle, revise la versión instalada:

```
rails -v
```

Comprobar todo

Incluso una tecnología tan fiable como la aquí presentada, puede fallar en Windows, un sistema operativo que a veces suele comportarse como un campo minado para programadores.

Para evitar futuros problemas, pruebe realizar las siguientes acciones en la raíz de la aplicación:

```
rails generate scaffold Student name:string age:integer registered:boolean
rake db:create:all
rake db:migrate
rails server
```

Luego, cargue las direcciones:

- http://localhost:3000/rails/info/properties
- http://localhost:3000/students

Si algo falla durante algún paso:

- Asegúrese de haber intentado varias veces la misma línea. GEM no provee a la fecha en su capa de aplicación, opciones para decidir sobre un exceso de carga en el repositorio rubygems.org.
- Cuidado cuando copie y pegue desde la versión de este libro en PDF. A veces caracteres en un formato se pegan binariamente de otra forma en el destino. Por si acaso: transcriba palabra a palabra las órdenes aquí expuestas. Además es bueno para la memoria.
- Si usa MySQL, cada vez que cambie la contraseña en **config/database.yml,** reinicie Webrick (la orden **rails server**) mediante **Ctrl + C**.

Si persiste el mismo mensaje de error, entonces será el momento de revisar informes de último momento en

- http://guides.rails.info
- Y los recomendados de Luis Lavena, el genio creador de RubyInstaller:
  - http://github.com/oneclick/rubyinstaller/wiki/Tutorials
  - http://pragmaticstudio.com/blog/2010/9/23/install-rails-ruby-windows
  - http://skim.la/2010/08/21/rails-3-rc-on-windows-using-rubyinstaller-192/

Por supuesto, no deje acudir a algunas de las excelentes listas de correos

- En español:
  - http://lists.simplelogica.net/mailman/listinfo/ror-es
  - http://lista.rubyargentina.com.ar/listinfo.cgi/ruby-rubyargentina.com.ar
  - http://librelist.com/browser/ruby - enviar correo a rubysur@librelist.com y esperar

Reply

- https://groups.google.com/forum/?fromgroups#!forum/mendoza-rb

- En ingles y como curiosidad, ¡para chicas! http://railsgirls.com

Finalmente, puede mantenerse actualizado acudiendo a mis vínculos bookmarkeados en

- http://del.icio.us/karancho/rails3
- http://del.icio.us/karancho/rails
- http://del.icio.us/karancho/ruby
- http://del.icio.us/karancho/tutoriales

Editores

[IDEs] vs [Editores simples + IRB]

Coincido tanto con Fabio Akita, como con Michael Hartl, en que los grandes IDEs para Ruby molestan mas de lo que ayudan[6]. Si usted proviene de Java o .NET, inmediatamente buscará autocompletado, instrospección, y varias funciones que bajo Ruby no tienen mucha utilidad.
La razón se encuentra en que la mayoría de las funciones que necesite, ya están incluidas en la librería base. Y no son muchas. Al cabo de unas horas, notará que todo el tiempo ha estado llamando a los mismos métodos.
Lo ideal es utilizar "los polígonos de tiro": **irb** para Ruby, y **rails console** para Rails. En ellos puede probar pequeños trozos de código, y luego pegarlos en algún editor simple.

Supongamos que queremos averiguar todas aquellas cosas que podemos hacerle a una cadena. Vamos primero a consultar al **irb**

 **gnome-terminal,** o mejor aún: **terminator**

 **Inicio → Programas → Ruby → Start Command Prompt with Ruby**

Luego, llamamos a **irb**

  **irb**

```
irb(main):001:0> p = "pepe"
=> "pepe"
```

Averiguamos que podemos hacer con **p**:

```
irb(main):002:0> p.methods
=> ["upcase!", "zip", "find_index", "between?", "to_f", "minmax", "lines", "sub", "methods",
"send", "replace", "empty?", "group_by", "squeeze", "crypt", "gsub!", "taint", "to_enum",
"instance_variable_defined?", "match", "downcase!", "take", "find_all", "min_by", "bytes",
"entries", "gsub", "singleton_methods", "instance_eval", "to_str", "first", "chop!", "enum_for",
"intern", "nil?", "succ", "capitalize!", "take_while", "select", "max_by", "chars", "tr!",
"protected_methods", "instance_exec", "sort", "chop", "tainted?", "dump", "include?", "untaint",
"each_slice", "instance_of?", "chomp!", "swapcase!", "drop", "equal?", "reject", "hex",
"minmax_by", "sum", "hash", "private_methods", "all?", "tr_s!", "sort_by", "chomp", "upcase",
"start_with?", "unpack", "succ!", "enum_slice", "kind_of?", "strip!", "freeze", "drop_while",
"eql?", "next", "collect", "oct", "id", "slice", "casecmp", "grep", "strip", "any?", "delete!",
"public_methods", "end_with?", "downcase", "%", "is_a?", "scan", "lstrip!", "each_cons", "cycle",
"map", "member?", "tap", "type", "*", "split", "insert", "each_with_index", "+", "count", "lstrip",
```

6 http://www.akitaonrails.com/2009/01/13/the-best-environment-for-rails-on-windows - http://railstutorial.org/book#sec:development_tools

```
"one?", "squeeze!", "instance_variables", "__id__", "frozen?", "capitalize", "next!", "each_line",
"rstrip!", "to_a", "enum_cons", "ljust", "respond_to?", "upto", "display", "each", "inject", "tr",
"method", "slice!", "class", "reverse", "length", "enum_with_index", "rpartition", "rstrip", "<=>",
"none?", "instance_variable_get", "find", "==", "swapcase", "__send__", "===", "min", "each_byte",
"extend", "to_s", "rjust", "index", ">=", "size", "reduce", "tr_s", "<=", "clone", "reverse_each",
"to_sym", "bytesize", "=~", "instance_variable_set", "<", "detect", "max", "each_char", ">",
"to_i", "center", "inspect", "[]", "reverse!", "rindex", "partition", "delete", "[]=", "concat",
"sub!", "dup", "object_id", "<<"]
```

Vaya, ¿así que podemos Capitalizar la primer letra?

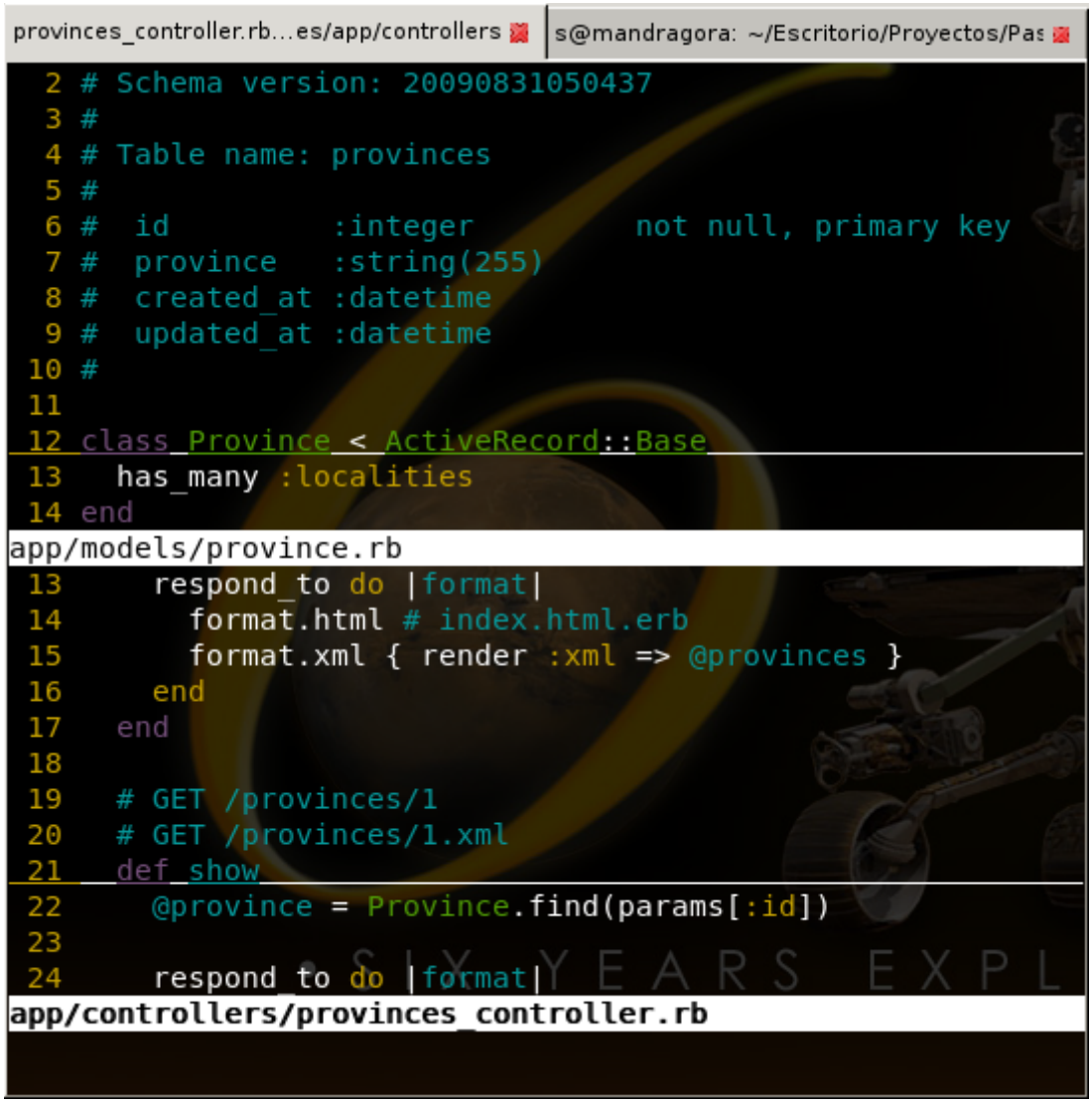
```
irb(main):003:0> p.capitalize!
=> "Pepe"
```

Ahora sí, sabiendo que esas porciones de código son seguras, nos vamos a algún editor y pegamos esa línea.

Editores

Se ofrecen aquí editores que pueden, en su mayoría, encontrarse con versiones para los tres sistemas operativos mas usados, Linux, Mac OS/X y Windows.

Vim (Windows y Linux)



Vim debe ser uno de los mejores y mas usados editores en la historia de la programación. Sin embargo, requiere antes de ser usado, buscar algún breve tutorial en Internet. Es decir, no es apto para impacientes. Además, al ser muy liviano, es el preferido de los **sysadmins**, quienes lo usan a menudo cuando se conectan remotos al servidor vía conexiones ssh, limpio, sin barras ni botones, y muy extensible. Quien se acostumbra a Vim, tiende a ver los demás IDE como si fueran travestis llenos de accesorios por encima. Por tal razón es el ideal para los

codemonkeys, geeks, aspirantes a hackers, dueños de maquinas limitadas, o simplemente autodidactas.

Vim puede ser llamado desde consola/msdos mediante el comando **vim** (austero, rápido, sin menú), o como **gvim** (gráfico, con ventanas, con barra de iconos, mas "amable").

También existe una versión de gvim aún mas fácil de usar llamada **Cream**.

Vim / Cream en Windows

Descargable desde vim.org. En el asistente de instalación, recuerde activar la opción **Full**, a fin de poder lanzarlo desde el propio MSDOS.

Lo primero que desagrada a los usuarios de Vim para Windows es la modalidad **Inserción:** se debe presionar `[i]` para comenzar a introducir texto. Para evitar esta modalidad, se sugiere lanzar vim mediante el icono **Vim Easy**, o bajar **Cream** desde cream.sourceforge.net

Vim / Cream en Ubuntu Linux

Vim

- `sudo apt-get install vim vim-gnome vim-rails`
  Se lo llama desde terminal como "**gvim**" o como "**vim**".

Cream

- `sudo apt-get install vim vim-gnome vim-rails cream`
  Se lo llama desde la terminal como "**cream**".

Ayuda!

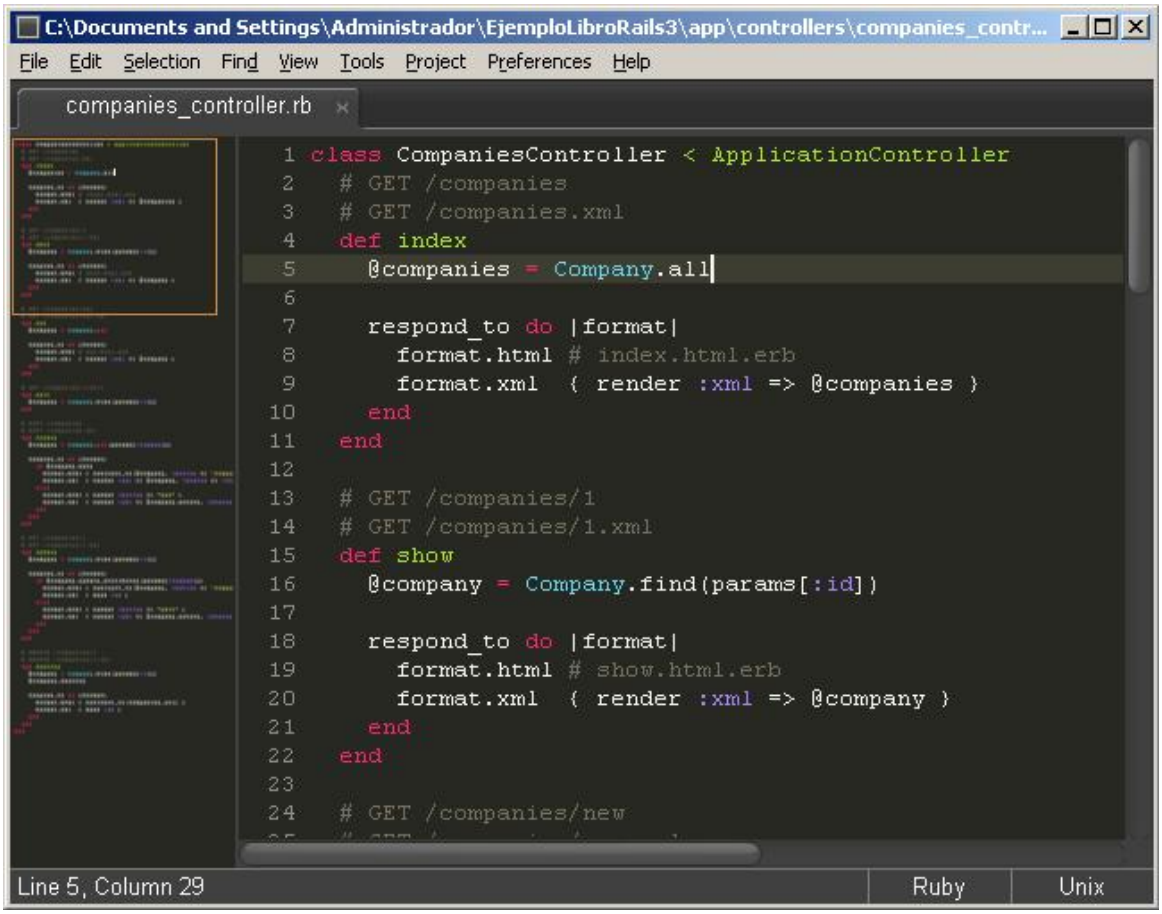
Aquí les dejo una serie de vínculos para dominar este magnífico editor:
http://delicious.com/karancho/vim+tutoriales

Además, una tarjeta de referencia rápida que puede ser estampada en una taza
http://www.eim.esc.edu.ar/incubadora
(La vendí durante un tiempo en Mercadolibre con cierto éxito, hasta que decidí liberarla).

Vim con Esteroides (Linux y Windows)

Avanzado: Wael M. Nasreddine muestra como aprovechar **exuberant-ctags** para agregarle varias comodidades al viejo y confiable Vim. Se recomienda leer la nota presente en https://github.com/carlhuda/janus

Sublime (Windows y Linux)



Michael Hart, el autor del excelente Ruby on Rails Tutorial (www.railstutorial.com) recomienda un editor para aquellos usuarios de Windows que no deseen lidiar con el "Unix Style" de Vim. Se trata de SublimeText (www.sublimetext.com), un editor muy agradable, que promete mucho. No posee todavía autocompletado, pero muchos snippets al estilo de Texmate (Mac).

Si bien solo se lo debe descomprimir y comenzar a usar, Michael ha escrito un interesante tutorial para hacerlo mas user friendy en https://github.com/mhartl/rails_tutorial_sublime_text

RedCar (Linux y Windows)

Es el nuevo favorito de las listas de correo. Libre, gratis, multiplataforma, liviano, e instalable simplemente como una gema. Soporta autocompletado, trae algunos snippets, y es el que trae mas themes oscuros. Las instrucciones para instalarlo pueden encontrarse en

http://wiki.github.com/danlucraft/redcar/installation

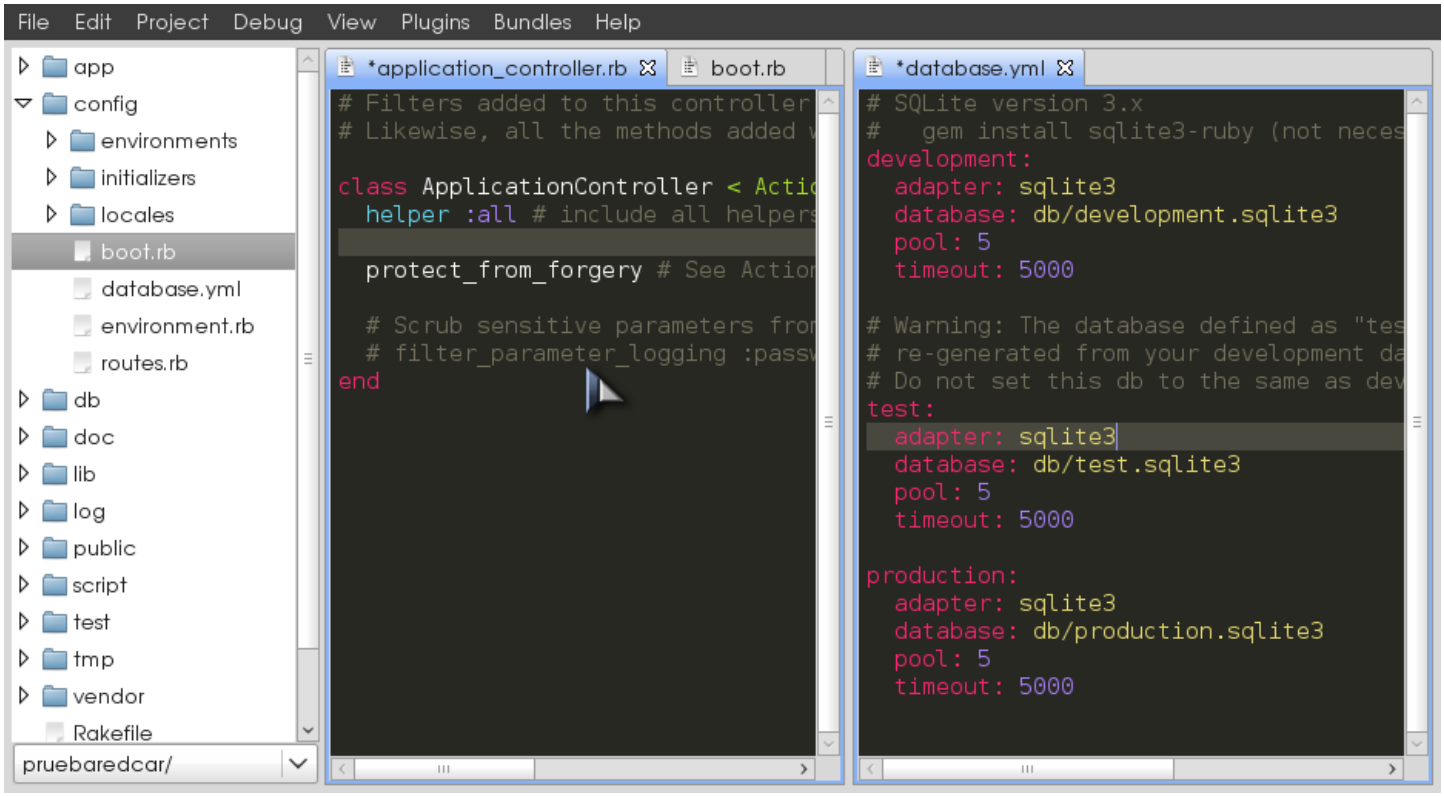
… las cuales, en cuentas resumidas, se componen de las siguientes instrucciones:

```
sudo apt-get  install libzip-ruby1.8
```

```
gem install -V redcar && redcar install
```

`gem install -V redcar`

`redcar install`



El instalador no fabrica una entrada en el menú, de modo que tendremos que hacerlo nosotros, o llamarlo simplemente desde la terminal, **parados en la raíz del proyecto**, mediante

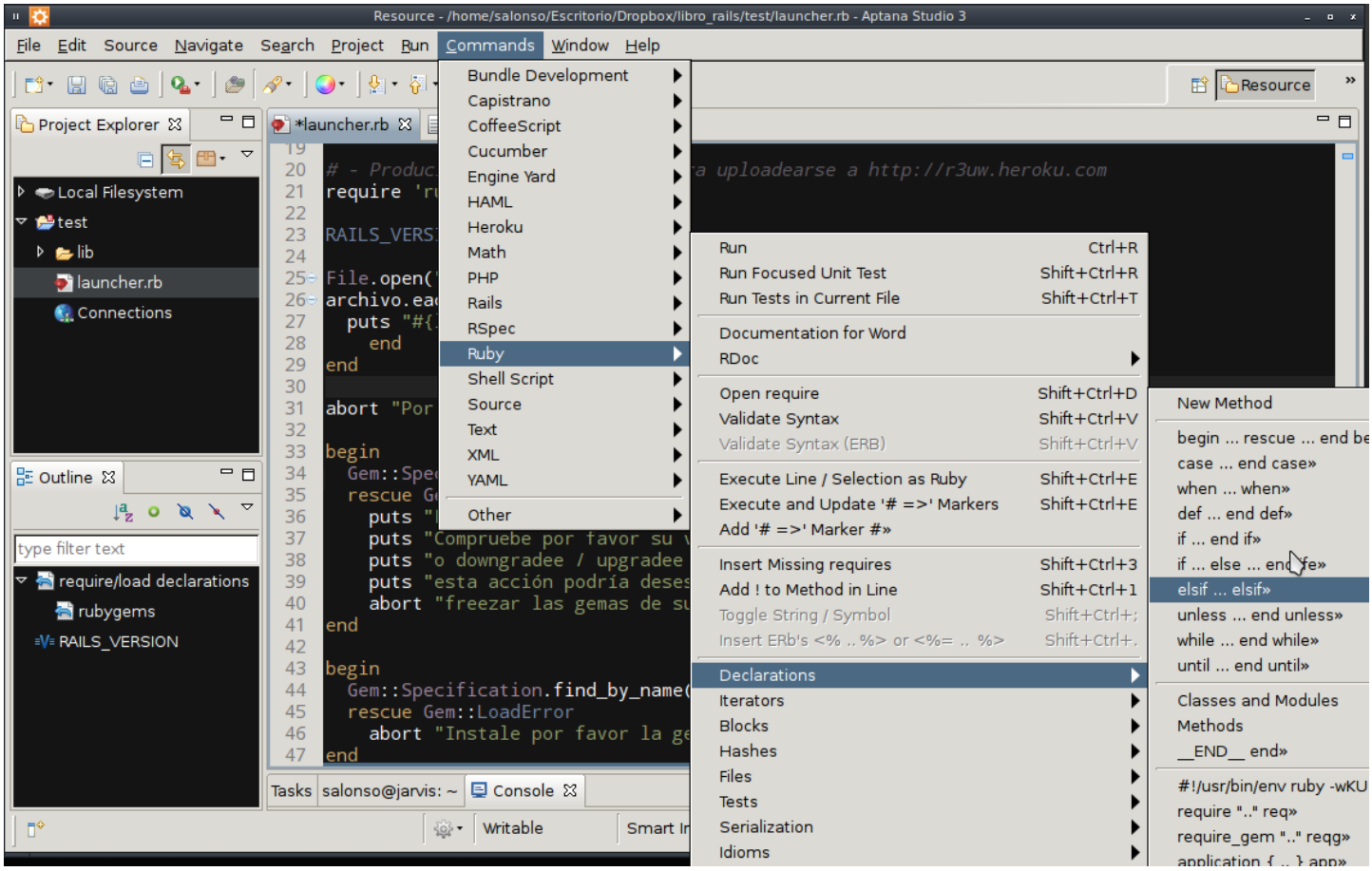
  `redcar .` (si, con el punto al lado: significa "carpeta actual")

IDEs

Hasta ahora veníamos comentando simples editores. A continuación se presentan entornos mas completos, capaces de gestionar servidores, debuggear y asistir a programadores mas experimentados. No se recomienda su uso a los novatos, ya que retrasarían inútilmente el aprendizaje inicial de Ruby.

  Aptana Studio (Linux y Windows)

Los fanáticos de Eclipse se alegrarán de conocer este fork para Ruby. De hecho, se encuentra a medio camino entre un editor y un IDE, ya que posee varia características interesantes.



Para empezar, provee soporte para lenguajes como: PHP, Python, Ruby, CSS, Ajax, HTML y Adobe AIR. Además, tiene la posibilidad de incluir complementos para nuevos lenguajes y funcionalidades. Compatible con aproximadamente 1000 extensiones. A saber:

- Debug integrado con Firefox, como detalle interesante, ya que Ruby trae un

debugger propio, que puede usted consultar en el Apéndice A.

- Asistente de código para HTML y Javascript.
- Librerías Ajax (jQuery, Prototype, Scriptaculous, Ext JS, Dojo, YUI y Spry entre otras).
- Conexión vía FTP, SFTP, FTPS, y snippets para Heroku, muy útil para agilizar deploys.
- IDE para GIT integrado, si el programa detecta un **git init .** o un **git clone** previo.
- Impresionante colección de snippets para realizar tareas comunes (ver captura).


Netbeans (Linux y Windows)

Netbeans es un entorno muy completo, repleto de atajos y opciones. Consume bastante RAM, 1 GB ~ 2 GB al menos para trabajar cómodamente. Es el único del conjunto que puede ser conseguido en español, consuelo de novatos.
Lo malo: al ser ignorado por parte de la comunidad de Oracle, esta compañía, menos generosa que la antaño Sun Microsystem, ha dejado oficialmente de soportar Ruby desde la versión 7. Una pena, ya que se nota un verdadero esfuerzo por parte de la comunidad de Netbeans hasta la versión 6.9.
Sin embargo ciertos detalles como los largos tiempos de despliegue, la ausencia de fastdebugger en la versión Windows, y otros aspectos, hace que la recomendación de un IDE integrado y gratuito pase por otras opciones, como Aptana Studio.

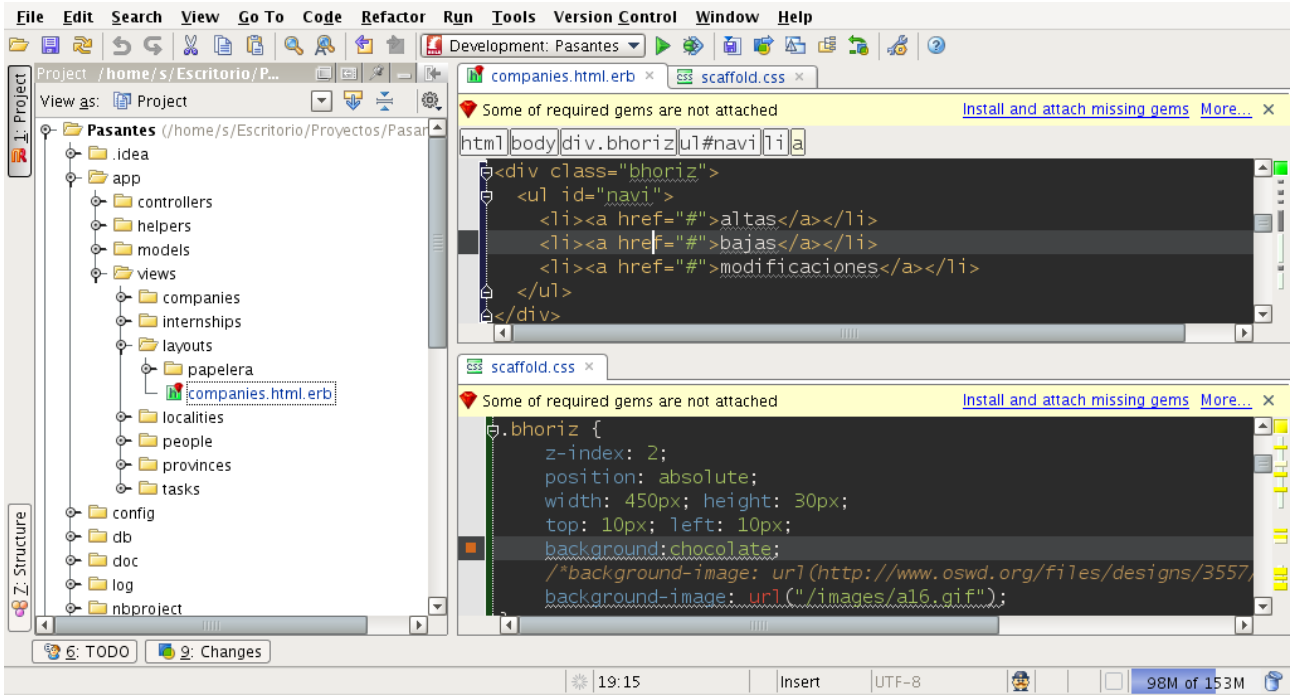
Rubymine (Linux y Windows)

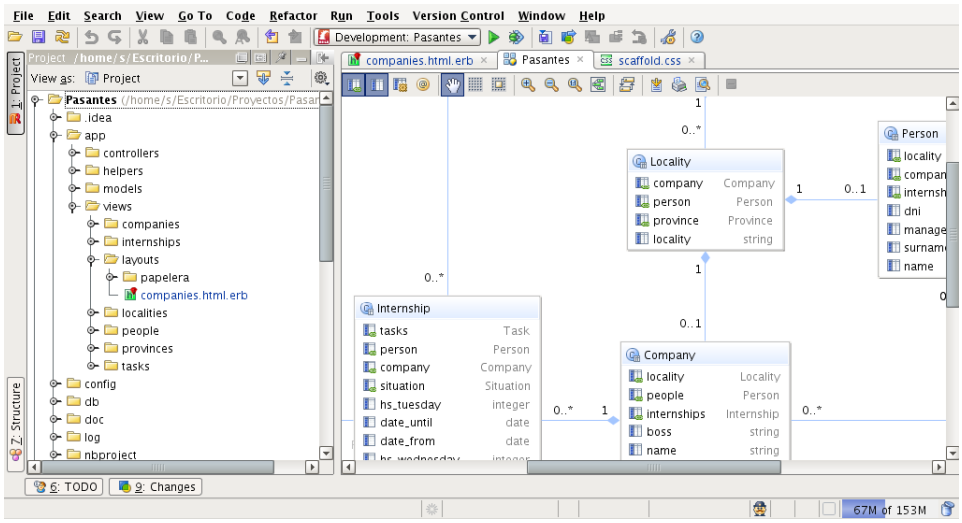
Es un editor comercial, aunque a un excelente precio: u$s 100.

Posee el mejor autocompletado del conjunto, sugerencias de optimización, detección de sintaxis, impresionante manejo de CSS, e incluso un generador de diagramas de relaciones has_many, belongs_to, etc. También posee varias ayudas para aquellos que usan testear antes de codificar (Test Data Driven), mediante rspec, cucumber, y otros.

Un profesional se sentirá muy a gusto con esta herramienta, en tanto que un novato podría sentirse intimidado por la cantidad de opciones.

Requiere de al menos 512 MB ~ 1 GB de RAM para trabajar cómodamente.





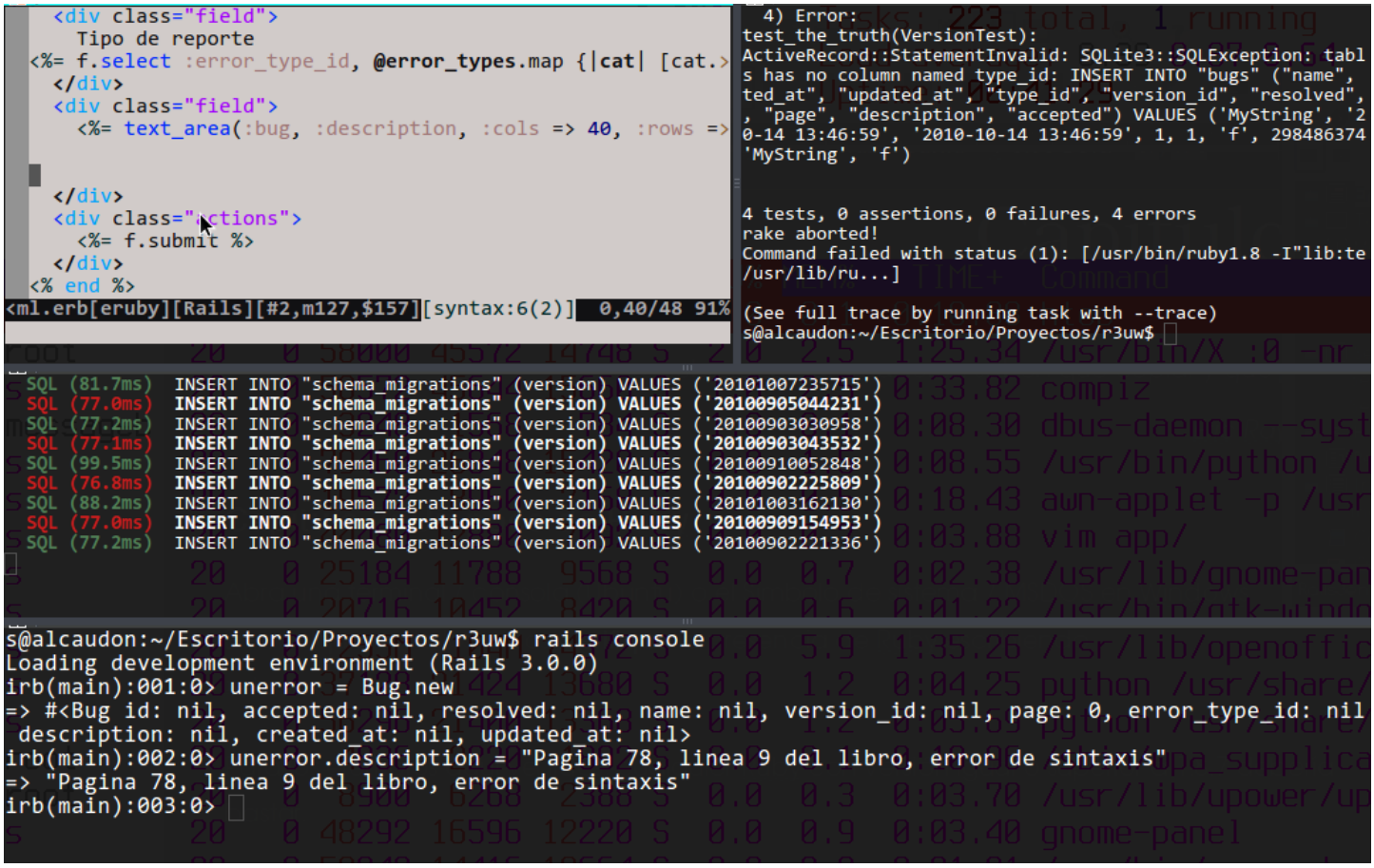
Otras herramientas útiles

"Dadme un punto de apoyo y moveré el mundo"

Arquimedes

Consola / Terminal / MSDOS

Si bien Ubuntu ya trae la excelente **gnome-terminal,** la experiencia de uso puede mejorarse agregando a **Terminator**, una consola con capacidad de dividir el espacio en varias consolas. De esta manera, se puede crear un IDE a medida que progresa la sesión de trabajo. Cuando queremos tener algo a la vista en simultáneo, simplemente dividimos:

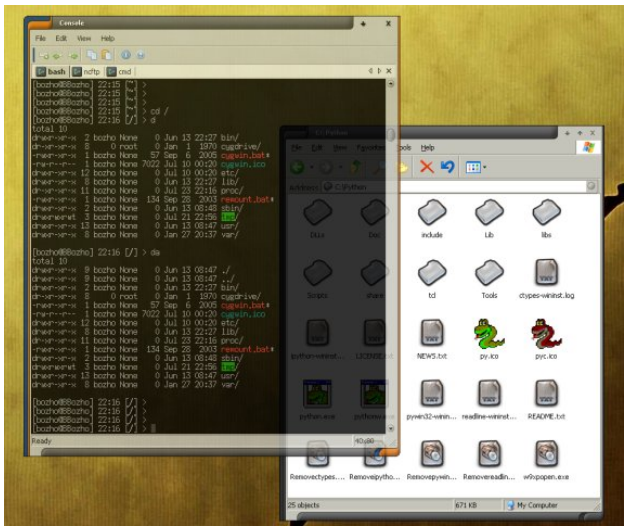


En la captura puede observarse una edición de las vistas (**vim**), un espacio en el que se está testeando (**rake test:units**), una porción que vuelca los logs del server (mediante **tail -f**), y la consola de Rails (**rails console**) probando trozos de código.

Exterminator se puede obtener directamente desde los repositorios, mediante

```
sudo apt-get install terminator
```

En Windows, puesto que va a pasar bastante tiempo con la nariz dentro del Símbolo de Sistema, o "MSDOS", sería conveniente que obtenga un reemplazo mas cómodo, ubicado en http://sourceforge.net/projects/console/files/console-devel/



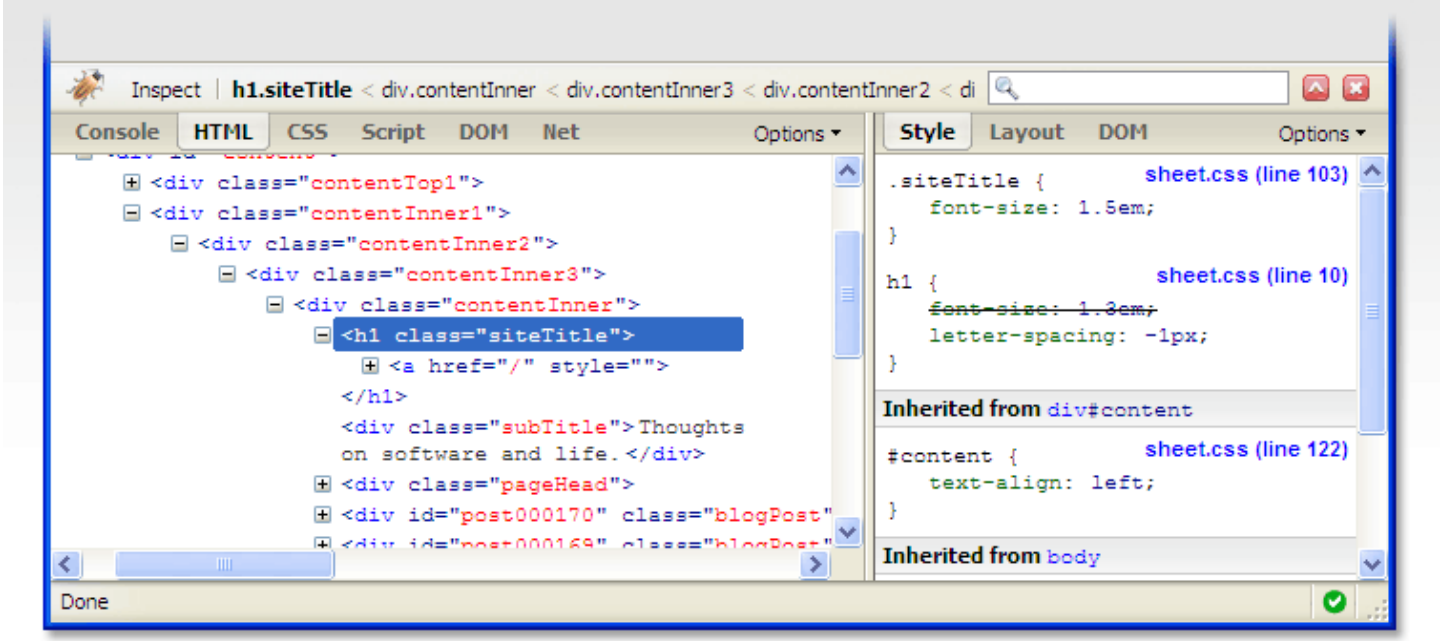
Console es un zip que puede ser descomprimido en cualquier parte. Pero no posee las variables de entorno de Ruby. Por tal razón, como primera medida el programa debería ejecutar el archivo **setrbvars.bat** que se encuentra en la carpeta bin de Ruby.

Para mayor comodidad, yo suelo crear un archivo .bat en el escritorio que lanza todo el conjunto. El contenido del bat es el siguiente, sin espacios (adapte de acuerdo a sus rutas):

```
c:\Users\Sergio\Downloads\Console-2.00b148-Beta_64bit\Console2\Console.exe
C:\Windows\System32\cmd.exe /E:ON /K C:\Ruby193\bin\setrbvars.bat
```

Windows / Linux: Firebug

Firebug es un plugin para Firefox que permite examinar las páginas que vamos recibiendo



en el navegador. Si algo no anda bien en el diseño, simplemente pulsamos F12, y el navegador nos permitirá revisar fácilmente zonas , etiquetas, su correspondencia en las plantilla CSS, sus enlaces javascript, etc. Presente en https://addons.mozilla.org/en-US/firefox/addon/1843

Asimismo, Google Chrome trae ya incluida una funcionalidad equivalente al alcance de la misma tecla –F12--


Windows / Linux: Web Developer

Este plugin, acaso popularizado por la genial @raymicha y sus magníficos tutoriales de XHTML[7], permite intervenir -al igual que Firebug- en el diseño de la página, inspeccionar diseños ajenos, obtener información valiosa, e incluso cambiar las CSS en tiempo de ejecución. Esto es muy útil para probar y afinar detalles durante los maquetados.

Se encuentra disponible tanto para Firefox como para Chrome.

7 http://www.trazos-web.com/2009/09/16/como-convertir-un-psd-a-xhtml-y-css/

Capítulo 2

Los buenos programadores se pasan la vida programando.

Los excelentes programadores, saben *que copiar y que pegar.*

Eric Raymond

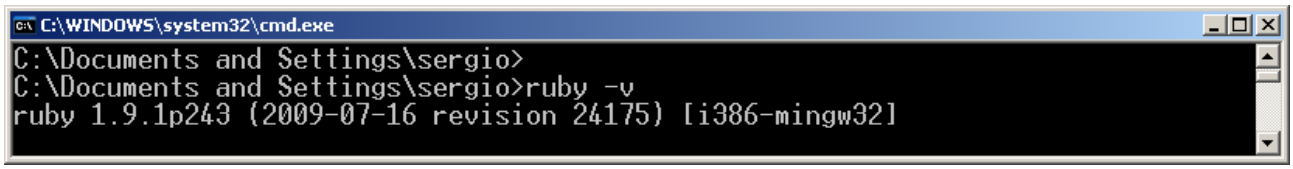
Capítulo 2: Un poco de Ruby

Abra una terminal / consola (Ubuntu) o **Inicio → Programas → Ruby → Start Command Prompt with Ruby** en Windows.

Verifique la presencia de Ruby escribiendo

```
ruby -v
```

Para los ejemplos aquí vertidos, cualquier Ruby con versión igual o superior a 1.8.7 debería bastar.



IRB... o "el polígono de tiro"

Llame al **Interprete Interactivo de Ruby**. Servirá para introducirnos un poco en el lenguaje sin necesidad de guardar archivos.

Escriba las palabras en **negrilla:**

irb

```
Welcome to interactive ruby!

"saludos".length
=> 7
"saludos".methods
=> ... (muchos metodos)
"saludos".reverse.capitalize
=> "Sodulas"
```

```
3.times do
  puts "holas "
end
holas
holas
holas

[3,1,7,0].sort.reverse
=> [7, 3, 1, 0]
```

For vs Iterar

```
for i in 1973..2008
 puts i
end

1973
...
2008

1973.upto(2008) do | anio |
  puts anio
  break if anio == 2007
end

1973
...
2007
```

Fechas

```
require 'date'
=> true
```

Llamamos y cargamos un *Módulo* de la librería base. Los módulos son colecciones de métodos, que no requieren de instanciación.

```
Time::now.year
=> 2007
```

```
sergio = Date.new(1973,2,26)

Time::now.year - sergio.year
=> 34

for i in sergio.year..2008
puts i
end

1973 .......... 2008

cristian = Date.new(1973,10,26)
cristian - sergio
=> Rational(242, 1)

cristian.month - sergio.month
=> 8
```

Arreglos

Nota: si observa el ejemplo, notará que los arreglos o "arrays" pueden contener toda clase de elementos: cadenas, enteros, objetos, otros arreglos...

```
arreglo=["1",2,0.3],[4,5,6]
=> [["1", 2, 0.3], [4, 5, 6]]
```

Es decir

arreglo			
	Columna 0	**Columna 1**	**Columna 2**
Fila 0	"1"	2	0,3
Fila 1	4	5	6

```
arreglo[1][1]
=> 5

lenguajes = ['Python', 'Java', 'Ruby']
lenguajes[1]
```

```
=> "Java"

lenguajes[0..2]
=> ["Java", "Python", "Ruby"]

lenguajes[0,2]
=> ["Java", "Python"]

for i in lenguajes
    puts i
end
=> ["Java", "Python", "Ruby"]

lenguajes << 'PHP'
=> ["Python", "Java", "Ruby", "PHP"]
```

Bloques vs For

```
lenguajes = ['Python', 'Java', 'Ruby']

lenguajes.each do |lenguaje|
  puts 'Me gusta ' + lenguaje + '!'
end

=> Me gusta Python
=> Me gusta Java
=> Me gusta Ruby

for i in lenguajes
   puts "me gusta " + i
end

=> Me gusta Python
=> Me gusta Java
=> Me gusta Ruby
```

Diccionarios (en rigor: Hash)

Los diccionarios son una especie de arreglos mixeados con índices manuales, como clave para encontrar un contenido.

Creamos un hash

```
legajos = { "uno" => "Alonso" , "dos" => "Pacifico" , "tres" => "Ramirez" }
```

Recuperamos datos del hash

```
legajos["dos"]
=> Pacifico
```

Creamos otro hash

```
diccionario = { 'uno' => 1 , 'dos' => 2 }
=> {"uno"=>1, "dos"=>2}
```

Le agregamos elementos

```
diccionario['tres'] = 3
=> 3
```

Inspeccionamos el objeto

```
diccionario
=> {"uno"=>1, "tres"=>3, "dos"=>2}
```

Le pasamos una clave ('uno') al objeto, para cambiar al contenido

```
diccionario['uno']='otra cosa'
=> "otra cosa"
```

Inspeccionamos nuevamente

```
diccionario
=> {"uno"=>"otra cosa", "tres"=>3, "dos"=>2}
```

Símbolos como Índices

Como índice de los hashes se pueden usar unas expresiones llamadas **símbolos**, las cuales vienen a ser un especie de etiqueta; una suerte de constante literal que no posee contenido.

Creamos un hash

```
persona = Hash.new
```

Lo llenamos

```
persona[:nombre] = 'Pedro'
persona[:apellido] = 'Picapiedra'
```

O también creamos y llenamos en un solo paso:

```
persona = {:nombre => 'Pedro', :apellido => 'Picapiedra'}
```

Volcamos algunos de valores almacenados

```
puts persona[:nombre]
=> Pedro
```

Modelo try -- catch - finally

No podía faltar algún mecanismo de catch

```
F = File.open("archivo")
begin
    #...
    rescue
    #...
     else
    puts "No hubo errores"
ensure
  if not f.nil?
        f.close
end
```

Acceso a bases de datos sin uso de ActiveRecord o Rails... todavía

Este ejemplo es útil para cuando necesitamos hacer alguna simple aplicación sin necesidad de desplegar el framework Rails.

-  En Linux requiere de agregar unas librerías:

  ```
  sudo apt-get install libmysql-ruby libdbd-mysql-ruby
  ```

-  En Windows, las librerías ya están incluidas.

Cuidado cuando copie y pegue estos ejemplos.

Acceso a MySQL

```
require 'dbi'
dbh = DBI.connect('DBI:Mysql:inventario', 'root', '')

sth = dbh.prepare('select * from material')
sth.execute

while row=sth.fetch do
    p row
end

sth.finish
```

Acceso a MSSQL desde Windows

```
require 'dbi'

DB = "Provider=SQLOLEDB.1;Password=coliflor;Persist Security Info=True;User
ID=sa;Initial Catalog=eith;Data Source=192.168.1.118"

sql = " select * from Alumno"
DBI.connect("DBI:ODBC:#{DB}") do | dbh |
  dbh.select_all( sql ) do |row|
     puts row[0] # primer campo
```

```
    puts row[1] # segundo campo
  end
end
```

Bindings gráficos

Se puede programar **ventanas** en Ruby? Por supuesto: Ruby puede utilizar diversos toolkits gráficos:

- Tk
- wxRuby (desmantenido)
- RubyGTK
- RubyQT
- CocoaRuby (Objetive C, para Mac)
- Shoes
- JRuby

Uno de los bindings mas avanzados, estables, multiplataforma y mejor documentada, es **wxRuby**. Solo requiere tener las librerías wxruby instaladas: ver sección de instalación.

Módulos

Definición: Módulo: colección de métodos, que no requieren de instanciación. Lo habíamos usado en el ejemplo

 irb

```
Time::now.year
```

¿Se puede crear módulos? ¡Por supuesto! Cumplen la función de las funciones en la programación estructurada.

```
#creamos un vector con cadenas
claves = ['python', 'java', 'ruby']

#solo por curiosidad, le agregamos un elemento
claves.push('eiffel')


def encriptar(v)
  #solo por curiosidad, ordenamos el vector enviado
  v.sort
```

```
  #creamos un nuevo vector
  vv = []

  #Representamos cada (each) elemento del vector con la variable i
  v.each do |i|

    #Ponemos el elemento en Mayúscula al principio
    i = i.capitalize

    #Lo damos vuelta
    i = i.reverse

    #Lo agregamos al vector nuevo
    vv.push i
  end

  return vv
end

#Probemos como quedó el módulo:
claves_encriptadas = encriptar(claves)

puts claves_encriptadas
```

Respuesta:

```
=> nohtyP
=> avaJ
=> ybuR
=> leffiE
```

Clases

Ejemplificaré el uso de clases, mediante un juego RPG. El siguiente programa guardelo en un archivo llamado **juego.rb**, y ejecutelo con el comando **ruby**:

```
class Personaje
  #Atributos accesores (getters y setters pero con estilo)
  #attr_reader
```

```
  #attr_writer
  #attr_accesor

  attr_reader :nombre, :fuerza, :magia
  @nombre
  @clase
  @fuerza
  @magia

  #constructor
  def initialize(n,c,f,m)
    @nombre = n
    @clase = c
    @fuerza = f
    @magia = m
  end
end

class Juego

  def initialize(p1,p2)
    @personaje1 = p1
    @personaje2 = p2
  end



  def combate
    if (@personaje1.fuerza + @personaje1.magia) <    (@personaje2.fuerza +
@personaje2.magia)
      puts "Gana " + @personaje2.nombre
    else
      puts "Gana " + @personaje1.nombre
    end
  end
end

#comienza juego
#escribo "main" para que se sienta como en ~~java~~ casa :)

#instanciamos personajes
```

```
arturo = Personaje.new("Arturo","Caballero",10,5)
smaugh = Personaje.new("Smaugh","Dragon",50,40)

#instanciamos un juego nuevo
SeniorDeLosLadrillos = Juego.new(arturo,smaugh)

#Let's figth! (parafraseando a Mortal Kombat ;)
SeniorDeLosLadrillos.combate
```

Herencia

La herencia es muy simple. Utiliza el operador <

```
class JuegoV2 < Juego
end
```

Clases temporales

Si necesitamos momentáneamente un método de una clase, y nada mas, podemos instanciar una clase sin ocuparnos de pensar en un nombre para el objeto, ni tener que destruirlo mas tarde.

```
Juego.new(arturo,smaugh).combat
```

Entrada por línea de comandos

Posiblemente desee probar pequeños trozos de código con interactividad del usuario, sin entrar a la complejidad de bindings gráficos como el mencionado wxRuby, ni accesos desde Web. El amigable **gets** nos permite leer entradas de teclado. Pruebe el siguiente ejemplo en un archivo con extensión **.rb**, o mas cómodamente, en el **irb**:

```
puts "Ingrese un numero: "
STDOUT.flush
primerNumero = gets.chomp
puts "Ingrese otro numero: "
STDOUT.flush
segundoNumero = gets.chomp
prom = primerNumero.to_i+segundoNumero.to_i
puts "El promedio de los dos numeros es: ", prom /2
```

Cuidado con los acentos usando **irb** en Windows

Métodos Bang!

Hay ocasiones en que por cada método, encontramos uno similar terminado con un símbolo !

Estos símbolos modifican el objeto que los llama. Y son muy cómodos. Use el comando **irb** para probar las siguientes instrucciones:

```
mensaje = 'Die die, piggy, die die!'
copia = mensaje.upcase
puts mensaje
>> Die die, piggy, die die!

puts copia
>> DIE DIE, PIGGY, DIE DIE!
```

En cambio con !

```
mensaje = 'Die die, piggy, die die!'
copia = mensaje.upcase!
puts mensaje
>> DIE DIE, PIGGY, DIE DIE!  <-------------------- (¡!)
puts copia
>> DIE DIE, PIGGY, DIE DIE!
```

Excepciones

Ejemplo se secuencia para atrapar el tipo de excepción, y afinarla para atraparla apropiadamente la siguiente vez.

La idea es no mostrar solo "error"

```
begin
  a = 1
  b = 0
  c = a / b
rescue
  puts "error"
  raise
end
```

Resultado:

```
error
ZeroDivisionError: divided by 0
```

Ahora ya podemos hacer un rescue específico para divisiones en cero, en vez de uno general

```
#Version mejorada
begin
  a = 1
  b = 0
  c = a / b
rescue ZeroDivisionError
  puts "Hey! esta usted usandome para dividir por cero?"
rescue
  puts "error"
  raise
end
```

Resultado:

```
error
Hey! esta usted usandome para dividir por cero?
```

Capítulo 3

Capítulo 3: Frameworks

Habíamos hablado durante la introducción, respecto de la conveniencia de utilizar frameworks para construir, mas que paginas web dinámicas, *aplicaciones Web 2.0.*
Si hacemos un resumen de todo mi sermón inicial, **un framework es un software que ayuda a construir otros softwares.**
En mas palabras, **es un gran aplicación incompleta, que se configura y conectan sus partes**.
Y siempre **obliga a seguir buenas practicas de construcción de software.** Típicamente, un framework puede incluir soporte de **programas**, **bibliotecas** y un **lenguaje de scripting,** entre otros softwares, para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Algunas características del framework Rails

- Provee salidas o "vistas" no solo en HTML: también genera JS y XML. Esto es muy útil cuando quien entra a nuestra aplicación no es una persona, sino otra aplicación. Ejemplo: formularios Flash construidos en MXML mediante el simpático Flex Builder.
- Posee un inteligente mapeo de clases mediante "Migraciones". Las migraciones permiten mantener actualizado un archivo, el schema.rb, que utiliza Rails como mapa para acceder a las tablas.
- Se adscribe a un movimiento denominado DRY - No Repitas (Don't Repeat Yourself). Por ejemplo, si en el modelo declaramos que "*el campo nombre no puede estar repetido"*, esto impedirá (y generará la validación por nosotros) la repitencia del nombre desde cualquier formulario, sea este un *new*, un *edit*, etc.
- Rails todo el tiempo realiza anotaciones de bitácora, o logging de transacciones, para ver que puede estar funcionando mal.
- Un aspecto interesante, es que para evitar las largas configuraciones, mapeos manuales de clases, tablas e índices que marean a todos los principiantes de Struts, Spring y otros frameworks, Rails prefiere convenir o "acordar" primero. Esto se llama "Convención sobre Configuración". Es decir, si respetamos los nombres que usa rails para sus tablas (plural), clases (en singular), índices (solo id), índices ajenos (_id), etc, él realizará un montón de trabajo automático por nosotros.

- Con Rails es fácil parametrizar URL fáciles de recordar, al estilo:
  - http://localhost:3000/tabla/edit/2
  - http://libros.com/revisiones/list/2012-08-11
- Compaginación: Rails maneja de manera muy limpia y elegante el diseño general, a través de layouts, los cuales veremos mas adelante. ¡Adiós frames, plantillas Dreamweaver y cosas raras!
- Todo el tiempo Rails gestiona instancias de Desarrollo, Prueba y Producción, incluso en el mismo servidor.

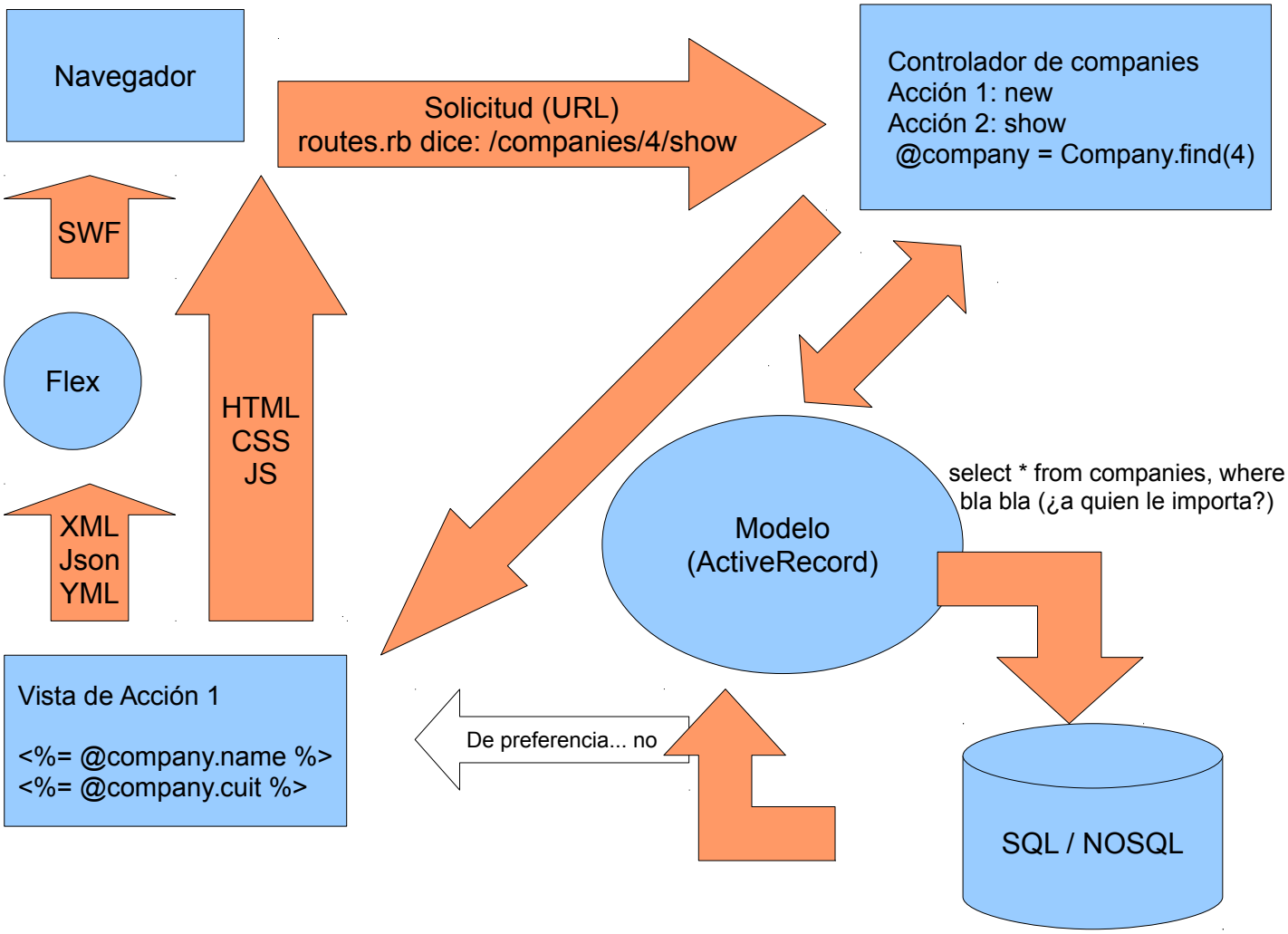
Frameworks MVC

Un clásico problema de la programación de paginas dinámicas es la mezcla entre la lógica, las conexiones, el SQL, la presentación, el manejo de las sesiones, etc.
El código resultante es confuso, difícil de leer, e incluso, de explicar. Incluso cuando el código es propio, al cabo de unos meses, mas de un programador se arranca los cabellos, mientras piensa
-"¿Que tenía en la cabeza cuando escribí esto...?

En los 80 se ideó un método para separar las partes esenciales de la aplicación, llamado MVC, correspondiente a Modelo - Vista - Controlador.
Usen o no Rails, quizás lo mas importante que se llevaran de esta lectura, es la noción de MVC. Muchos frameworks poseen sus propias implementaciones. Por mencionar algunos:

- Java: J2EE, Struts, Swing, ...
- Python: Django, TurboGears, ...
- PHP: QCodo, Smarty, phpMVC, Trax, Symfony, …
- ASP.NET, Windows Form, …

La gran diferencia, es que los programadores de Rails se acordaron que **MVC debía ser simple**.
Veamos un ejemplo básico



1) En el **navegador** se hace click en la compañía **#4**, action **show**
2) **routes.rb** descompone la solicitud, y decide **controlador** / **acción** a realizar. En este caso, **companies**/**show**. La URL puede incluir otros datos útiles, tales como el **#id**
3) En la **acción** se toman la decisiones, tales como la conveniencia o no de buscar la información solicitada.

   La petición de datos se realiza al **modelo**, mediante instrucciones de tipo **ORM** (Object Relational Map). Rails provee uno de los mejores **ORM**, llamado **ActiveRecord**. Ejemplo:

   ```
   @company = Company.find(4)
   ```

   Desde el **controlador** se podría pedir al **modelo** que se le proporcionen otras cosas que **ActiveRecord** no realize automáticamente. Ejemplo:

   ```
   @morosos = Company.buscar_activas
   ```

   Por supuesto, en ese caso, hay que aclarar el método **buscar_activas** en el modelo **Company**.

Las **acciones** suelen tener vistas asociadas. En este ejemplo: **show.html.erb**

4) En el **modelo** hay rutinas **find** capaces de encontrar lo que necesitamos. **ActiveRecord** hace todo lo posible para encargarse de la parte "dura", es decir inyectar complejas instrucciones SQL (o incluso NOSQL).
5) La idea de la vista es mantenerla limpia para concentrarse en el diseño. Mas de un diseñador puede espantarse cuando le pedimos que modifique una pagina escrita en PHP, y no deja de tener razón. Las decisiones quedan para el controlador. Las búsquedas, para el modelo.
   Por cierto, podemos emitir datos no solo en el convencional HTML, sino también en texto plano, JSON, YML, o XML. Otras capas y tecnologías, tales como Flex o AIR pueden aprovechar estas funciones.

Algo más respecto del Modelo

ActiveRecord, el ORM de Rails, se comunica con la base y devuelve a cambio objetos que representan registros o colecciones de registros. Los atributos del objeto obtenido son los campos.
Supongamos que veníamos desarrollando, en SQLite. Si en SQLite queremos saber cuantas empresas se encuentran activas, usamos

```
SELECT * FROM companies WHERE (active='t')
```

Mientras que en MySQL, la misma consulta, se representa como

```
SELECT * FROM companies WHERE active = true;
```

En un proyecto, cambiar a mitad del desarrollo de motor SQL puede ser una solicitud muy cruel[8]. Rails, mediante ActiveRecord, se mantiene por encima de esas necesidades y nos abstrae de los detalles:

```
Company.where(:active => true)
```

Aunque si necesitamos algo muy específico, estamos apurados, siempre podemos pasar porciones personalizadas a ActiveRecord:

```
Company.where(" active = 't' ")
```

8 http://picandocodigo.net/2009/video-hug-a-developer-abraza-a-un-desarrollador

Capítulo 4

Capítulo 4: Bienvenidos al Tren

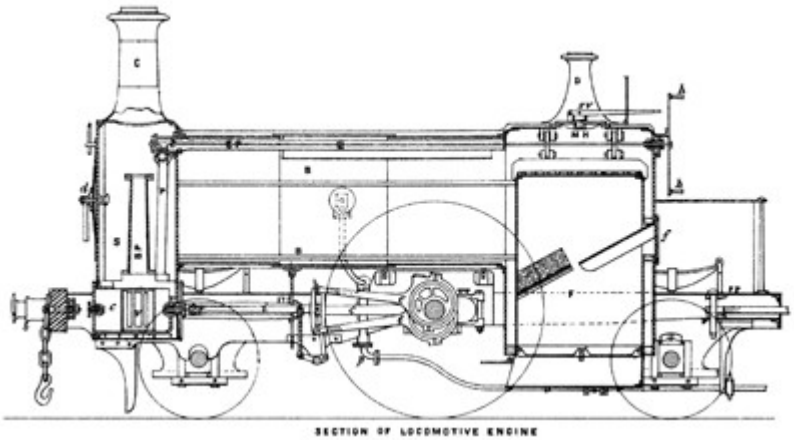
En esta parte intentaremos realizar, paso a paso, una aplicación con Rails.

Por el camino, se hará uso de algunas gemas y trucos propios de este *framework*. Los ejemplos aparecen en Windows solo para hacerles perder el miedo a los usuarios de este SO, a la vez que se demostrará las -leves- diferencias de trabajar con Linux.

Crear una Aplicación

Abrir una terminal o el Símbolo de Sistema, y ejecute el siguiente comando:

```
rails new demo
```



Esta orden fabrica el esqueleto de la aplicación Rails. Ademas, preconfigura el entorno para trabajar con una base de datos llamada SQLite.

Si deseamos en cambio trabajar con otras base de datos, tal como MySQL, Oracle, PostgreSQL, IBM DB2, etc, podemos preconfigurar algunos archivos escribiendo, por ejemplo

```
rails new demo --database=mysql
```

Mediante algún editor, abra los archivos de la carpeta **demo**, y observe las carpetas generadas. También observe atentamente las bases definidas en **config/databases.yml.** Volveremos a estos archivos mas adelante. Para correr el server Web de pruebas:

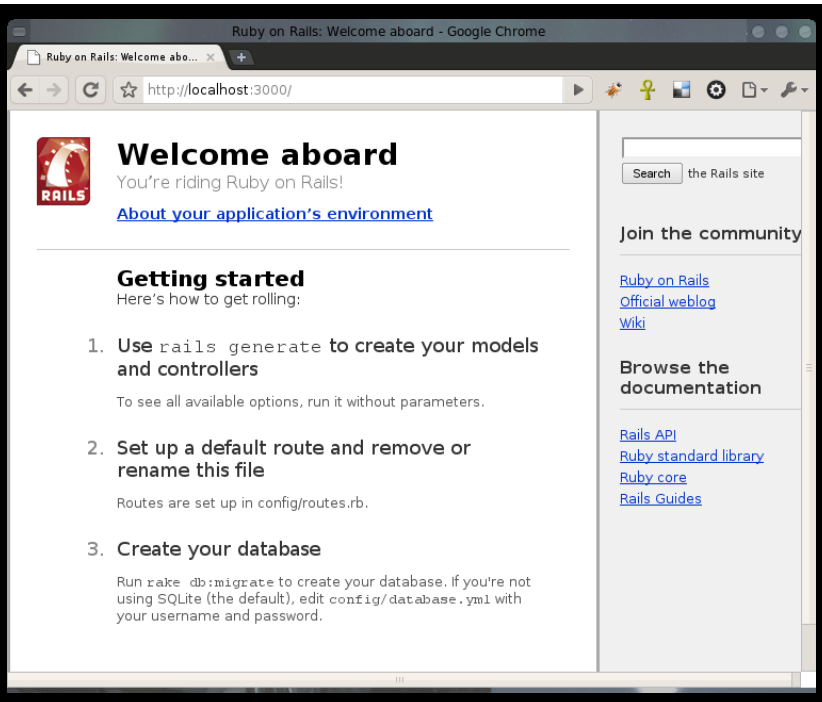
```
cd demo
```

```
rails server
```

```
ruby bin/rails server
```

Este comando corre un pequeño server web, escrito en Ruby, para realizar pruebas.

Cargue la dirección http://localhost:3000 en el navegador.



Si obtiene un error, cambie la palabra localhost por 127.0.01, o revise atentamente los pasos en el Capítulo 1: Instalación.

Vuelva a la consola / MSDOS, y observe la salida del comando rails server. Este muestra mucha información respecto del tráfico solicitado por el navegador:

```
rails server
```

```
ruby bin/rails server
```

```
=> Booting WEBrick
=> Rails 4.0.0.beta application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2010-03-02 15:24:01] INFO  WEBrick 1.3.1
[2010-03-02 15:24:01] INFO  ruby 1.8.7 (2010-01-10)
[2010-03-02 15:24:06] INFO  WEBrick::HTTPServer#start: pid=11684 port=3000
```

Recargue la pagina, cambiando la dirección por http://localhost:3000/zaraza

Observe el error generado, tanto en el navegador como en la salida del server Web.

Truco: un aspecto casi escondido de esta bienvenida, es la posibilidad de revisar si las gemas instaladas satisfacen a la aplicación. Es importante hacer click sobre

About your application's environment

(http://127.0.0.1:3000/rails/info/properties)

Si esto ocasiona un error:

We're sorry, but something went wrong

Significa que, por razones de seguridad, Rails se niega a mostrar el error. Para ver que es lo que le falta, debemos revisar la traza del comando rails server.

Webrick, por cierto, es un pequeño server para hacer pruebas simples. Cuando corra en modo producción, seguramente utilizará Apache o Mongrel. Si necesita mas potencia de respuesta en modo **development**, puede iniciar **mongrel** en lugar de **webrick**, desplegando el server de la siguiente manera.

```
gem install mongrel
```

Y luego

```
rails server mongrel
```

o también

```
rackup mongrel --port 3000
```

Integrar GIT al proyecto

Entre los programadores de elite existe una expresión denominada "Sistemas de Control de Revisiones", la cual se refiere a aquellos softwares capaces de automatizar las tareas de guardar, recuperar, registrar, identificar y mezclar versiones de archivos. Son muy útiles para crear ramas de desarrollo de archivos que son modificados frecuentemente. A esto se puede añadir que cobran verdadera utilidad cuando muchas personas trabajan en un mismo proyecto.
Se los utiliza particularmente para compartir y hacer el seguimiento de cambios en algoritmos informáticos, aunque también sirven para crear en forma colaborativa documentación, cartas y manuales.
Los sistemas de revisión mas conocidos son CVS, SVN, GIT, Mercurial y otros. GIT en particular es muy usado por la comunidad de programadores de Ruby.
En breve, y en nuestro proyecto, pronto comenzaremos a utilizar generadores de código. También instalaremos plugins y llamaremos scripts que modificaran varias partes del proyecto. Es común que un proyecto no se toca un solo archivo: como un efecto mariposa, los cambios en un archivo acarrean cambios en muchos otros. Al cabo de un tiempo, si queremos volver atrás, puede que nos encontremos con demasiados cambios que deshacer. El Ctrl + Z no es útil si ya hemos cerrado los archivos involucrados, y menos aun si hemos realizado múltiples cambios. Para peor, muchos de estos cambios son automáticos, es decir, son introducidos por los scripts.

Mientras vamos dejando cada vez mas profesional nuestro proyecto, utilizaremos GIT como ancla y maquina del tiempo.

Instalación de GIT

Bajo Linux

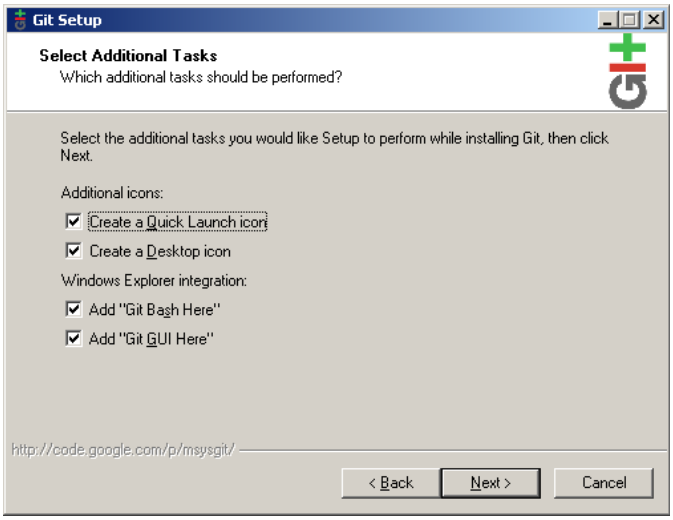
Bajo Linux Ubuntu, instalaremos GIT en modo consola, y en el mismo paso, varias cómodas interfaces gráficas, mediante un simple[9]

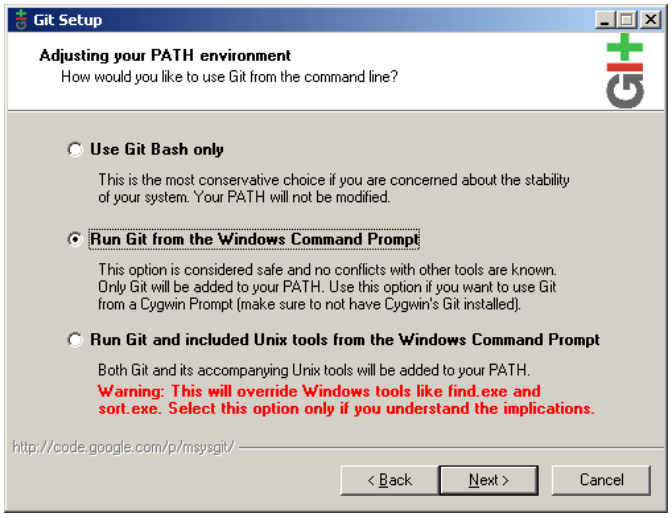
```
sudo apt-get install git-arch git-gui gitk tig qgit giggle meld
```

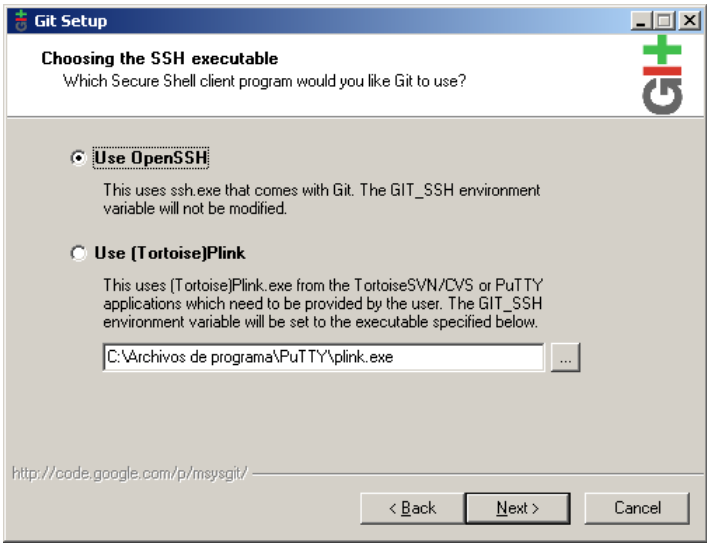
9 Nuevamente, si desea profundizar en el uso de **apt-get** / **aptitude,** baje una copia gratuita del Manual de Redes Libres, escrito también por quien le escribe ahora, y descargable desde http://www.eim.esc.edu.ar/incubadora/redes.pdf. El capítulo que trata el uso de apt-get es "Instalando binarios desde las fuentes".

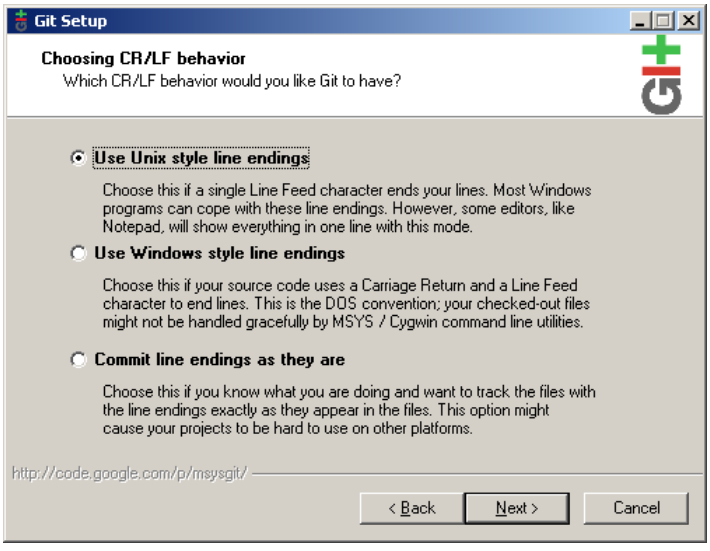
Bajo Windows

GIT ha sido portado de diversas formas a Windows. La mayoría de los usuarios que utilizan GIT bajo el sistema operativo de las ventanitas, coinciden en que la mejor implementación es la realizada por MsysGIT, la cual puede obtenerse en http://code.google.com/p/msysgit/ La instalación es muy sencilla, y como único comentario, debería mencionar la conveniencia de activar las siguientes opciones durante la instalación:









Utilizar GIT en forma local

El propósito de esta sección es mantener el código que vamos generando, aprender a volver atrás en el tiempo, invitar gente a participar, y fundamentalmente, comportamos como programadores con experiencia.

Si a medida que progresa en la construcción de los ejemplos del libro, se encuentra con errores, o tiene dudas respecto de como debería verse la aplicación final, recuerde que puede crear una carpeta en cualquier parte, y dentro de ella bajar mi versión de la aplicación, mediante la orden

```
git clone git@github.com:karancho/EjemploLibroRails3.git
```

Otra opción es solicitar a mi correo (escuelaint@gmail.com) un pequeño programita que me encuentro desarrollando que es capaz de desplegar el proyecto de este libro en su computadora, chequeando por el camino todos aquellos componentes que pudiera haberse olvidado instalar. Así podrá contrastar si el problema está en su código, en su software, o en mis ejemplos. En caso que encuentre un error en los ejemplos, sírvase por favor reportarlo en http://r3uw.heroku.com

Recordemos:

- Mediante el comando **rails new demo** habíamos creado el esqueleto de la aplicación.
- Habíamos entrado a la carpeta mediante **cd demo**
- Habíamos levantado un server de pruebas, con el comando **rails server**
  En este ultimo paso, la consola / MSDOS queda "capturada", es decir, no nos deja escribir otra cosa que Ctrl + C (interrumpir), a la vez que muestra las consultas de los navegadores.

Abrimos **otra** consola / MSDOS, entramos nuevamente a la aplicación mediante **cd demo**, y creamos un repositorio local:

```
cd demo
```

```
git init
```

```
Initialized empty GIT repository in .git /
```

El término "local" se refiere a que podemos gestionar tanto repositorios locales como

remotos.

Configuramos el repositorio con algunas variables útiles.

```
s@mandragora:~/demo$ git config --global user.name "Sergio Alonso"
s@mandragora:~/demo$ git config --global user.email "sergio@eim.esc.edu.ar"
s@mandragora:~/demo$ git config --global color.ui "auto"
```

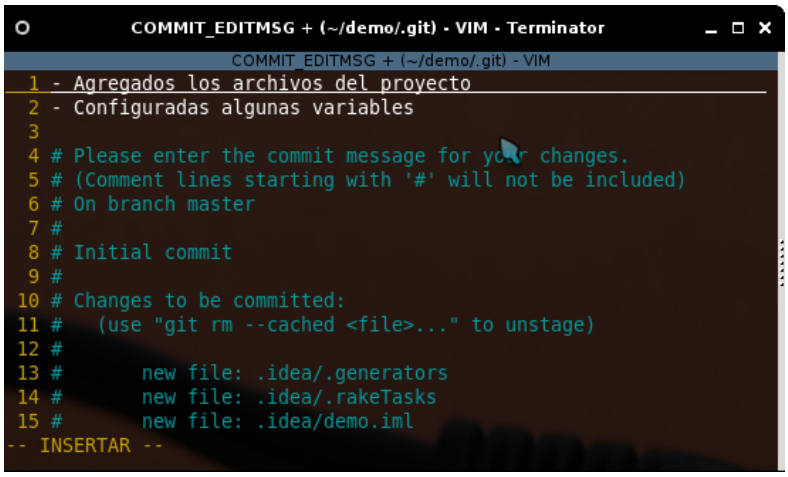
Agregamos **todos** los archivos del proyecto, y realizamos el primer commit, o mal traducido "entrega".

```
git add -A
```

```
git commit -a -m "Agregados los archivos del proyecto"
```

```
Created initial commit 073c820: - Agregados los archivos del proyecto
 128 files changed, 15218 insertions(+), 0 deletions(-)
```

En la línea **git commit**, GIT invocará a algún editor para que contemos que cambios hemos realizado. Se recomienda enfáticamente utilizar esta función como un diario personal.
El editor preferido de GIT es **vim**. Caso contrario buscará algún otro presente en el sistema, como **nano**, **mcedit** u otro.



- En el caso de **nano**, una vez que terminamos de describir nuestras aventuras, pulsamos Ctrl + X para salir y guardar.
- En el caso de **vim**, a no asustarse si nos cuesta operarlo. Solo debemos pulsar la tecla de **i**nserción de texto (**i**). Escribimos lo que se nos antoje, y volvemos al modo comando con **Esc**. La secuencia **:wq** grabará (**w**rite) los cambios, y saldrá (**q**uit) al sistema.
  Convengamos que **vim** no es muy amigable en su primera impresión. Tampoco en la segunda, pero su uso es muy aceptado en toda la comunidad, debido a diversas características que lo hacen probablemente el mejor editor en la historia[10]. Por esta razón existen cientos de tutoriales en internet para operar este magnífico editor. Pase

10 Si, si. Junto a **emacs**, por supuesto.

por el apartado de Editores para ver mas ayuda sobre el mismo.

Agregando y cambiando archivos

Vamos ahora a crear un archivo, ubicado en **public**/ llamado **bienvenida.html**
A fines educativos, su contenido será incompatible *ex profeso* con navegadores nuevos que no soporten XHTML. Hacer una pagina incompatible es muy simple. Tan solo debemos olvidarnos de incluir en la etiqueta <html> el DOCTYPE apropiado. Esto hace que el navegador entre en modo *quirks* o "raro", y al estilo de los viejos Explorer 4 y Netscape 4, ciertos símbolos como ó, é, ñ, etc deban ser expresados con los incómodos *caracters entities.*
Introduciremos a propósito el carácter **¡**, el cual, o bien debería contar con el DOCTYPE apropiado, o bien debería haber sido expresado como el entitie **¡**

```
<html>
        ¡Holas Manolas!
</html>
```

Para mas información sobre el modo quirks, lea la pagina 60 del magnifico libro gratuito **Introducción a CSS** de Javier Eguíluz Pérez , que puede encontrarse en librosweb.es

Ahora, estando parados en la raíz de la aplicación, agregamos este archivo al mecanismo de seguimiento de GIT mediante

```
git add public/bienvenida.html
```

Nota: también podemos agregar archivos sin especificar su ruta, mediante

```
git add -A
```

Nota: conveniencia de usar git add -A
El comando **git add** se puede utilizar acompañado de un archivo en particular al cual queremos que GIT trackee (persiga, rastree), o simplemente acompañado de un -A, en el

cual GIT revisa todo el árbol en busca de agregados, modificados o borrados.
El comando **git add -A** es muy cómodo cuando hemos realizado muchos cambios. Tantos, que no recordamos cuantos archivos hemos involucrado. Pero debemos tener en cuenta que quizás agregue cosas que no debería, tales como bases grandes, o archivos con contraseñas importantes.

Para evitar que **git add -A** tome en cuenta ciertos archivos, creamos un archivo en la raíz del proyecto con nombre

```
.gitignore
```

… cuyo contenido sea, por ejemplo

```
db/production.sqlite3
```

Rails 4 ya viene con un cómodo archivo .gitignore, que incluye ciertas carpetas que no tiene sentido que sean trackeadas, ni subidas a servidores de producción.

Luego del **git add**, si realizamos un **git status**, GIT anunciará que hay cambios en el árbol. Cambios de los cuales él no tiene copia. Para ser claros: no hemos hecho todavía la entrega (el commit).

  **git status**

```
# On branch master
# Changes to be committed:
#     new file:   public/bienvenida.html
```

Realizaremos una nueva entrega. Cuando no hay mucho que informar, y no queremos entrar al editor, podemos utilizar el modificador **-m**, seguido de una breve descripción.

  **git commit -m "Agregado archivo de Bienvenida"**

```
Created commit 8e5330c: agregado archivo de Bienvenida
 1 files changed, 3 insertions(+), 0 deletions(-)
 create mode 100644 public/bienvenida.html
```

Si el server continúa corriendo (rails server), y apuntamos el navegador hacia

http://localhost:3000/bienvenida.html descubriremos el error: nuestro carácter ¡ se ha convertido en algo parecido a Â¡

Ahora introducimos algunos cambios: modificamos el archivo **public/bienvenida.html:** para adecuarnos mas al standard que dicta la w^3, y le agregaremos algunas etiquetas.

Los cambios en el código van en **negrita**

```
<!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>
                <title>
                        Bienvenido al Sistema
                </title>
        </head>
        <body>
                ¡Holas Manolas!
        </body>
</html>
```

Ahora **git status** debería mostrar el status de "modified".

Informamos de los cambios mediante **git add public/bienvenida.html**, o el mas practico:

```
git add -A
```

Mandamos copia al repositorio, junto con una breve explicación de los cambios que se han llevado a cabo:

```
git commit -m "Agregadas etiquetas <head>, <title> y <body>"
```

El uso habitual de los comandos **git add -A** seguido de **commit** se pueden resumir en una sola expresión, conformada como **commit -a**

Si ahora queremos revisar los cambios introducidos

```
git show --color
```

```
commit c28bc8f3622dcef3e4c431a604e01eef80cf9645
Author: Sergio Alonso <sergio@eim.esc.edu.ar>
```

```
    Agregadas etiquetas <head>, <title> y <body>
diff --GIT a/public/bienvenida.html b/public/bienvenida.html
index cce29e9..3382961 100644
--- a/public/bienvenida.html
+++ b/public/bienvenida.html
@@ -1,3 +1,10 @@
 <html>
-       ¡Holas Manolas!
+       <head>
+               <title>
+                       Bienvenido al Sistema
+               </title>
+       </head>
+       <body>
+               &iexcl;Holas Manolas!
+       </body>
 </html>
```

Podemos observar en rojo (-) las líneas movidas o quitadas, y en verde (+) las agregadas.

¡ No puedo salir del comando git log / status / show !

Si el prompt de la línea de comandos parece "atrapado", por el carácter **(:)**, y no nos devuelve el control de la consola / MSDOS, no se preocupe. Significa que la consola se encuentra bajo el comando **less**, el cual posee algunos comandos para perseguir cambios, tales como la tecla **/** (buscar una palabra), **n** (seguir buscando), y otras, como **q** (salir).

La bitácora del maquinista II

En este punto, sería interesante consultar el historial de los cambios:

```
git log
```

Recuerde que puede usar la tecla **q** para salir del **log**.

Si nota muy confusa la salida del comando **git log,** puede mejorar un poco la visualización jugando con las variables mencionadas en el manual *git log help*. Por ejemplo:

```
git log --pretty=oneline --color
```

Al respecto, Bart Trojanowski, en http://www.jukie.net/bart/blog/pimping-out-git-log recomienda crear un log personalizado, inyectando la siguiente línea a la configuración en el proyecto:

```
git config --global alias.lg "log --graph --pretty=format:'%Cred%h%Creset \
-%C(yellow)%d%Creset %s  %Cgreen(%cr) %C(bold blue)<%an>%Creset' \
--abbrev-commit --date=relative"
```

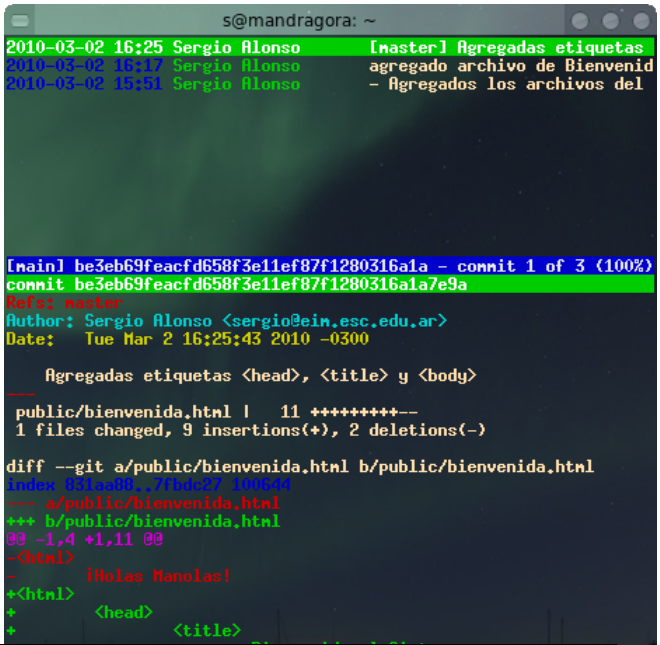
También puede ver todas las ramas que incidan sobre el proyecto haciendo

```
git log --pretty --graph --color
```

Ahora tendremos la posibilidad de hacer **git log** en nuestro proyecto y obtener un resumen mucho mas claro. Sírvase entrar a la página de Bart para ver algunos ejemplos.

Interface GIT en modo texto

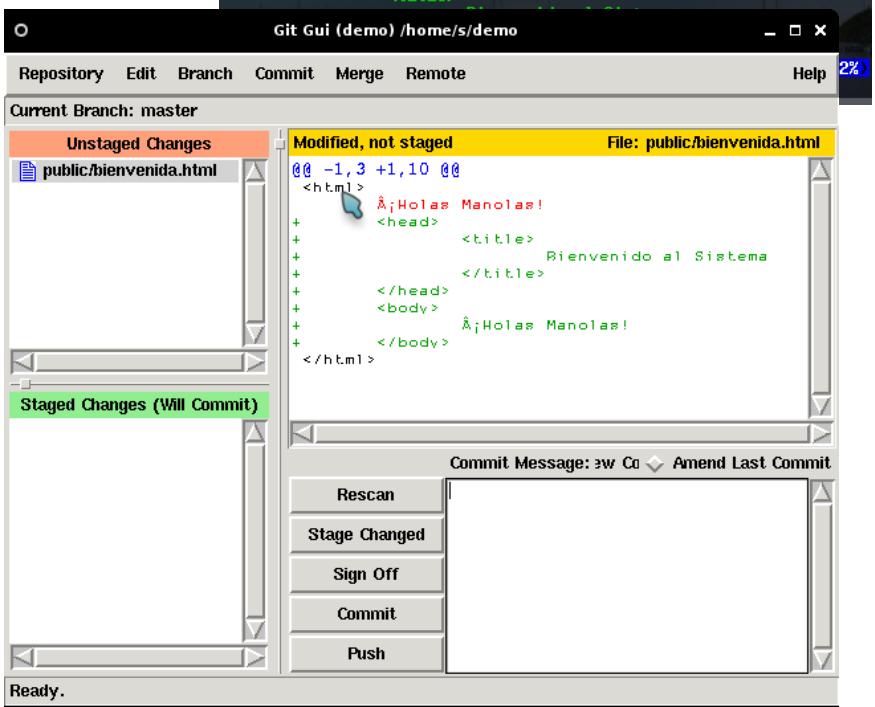
Tig. ¿Para qué mas? Sírvase observar la siguiente captura de pantalla.

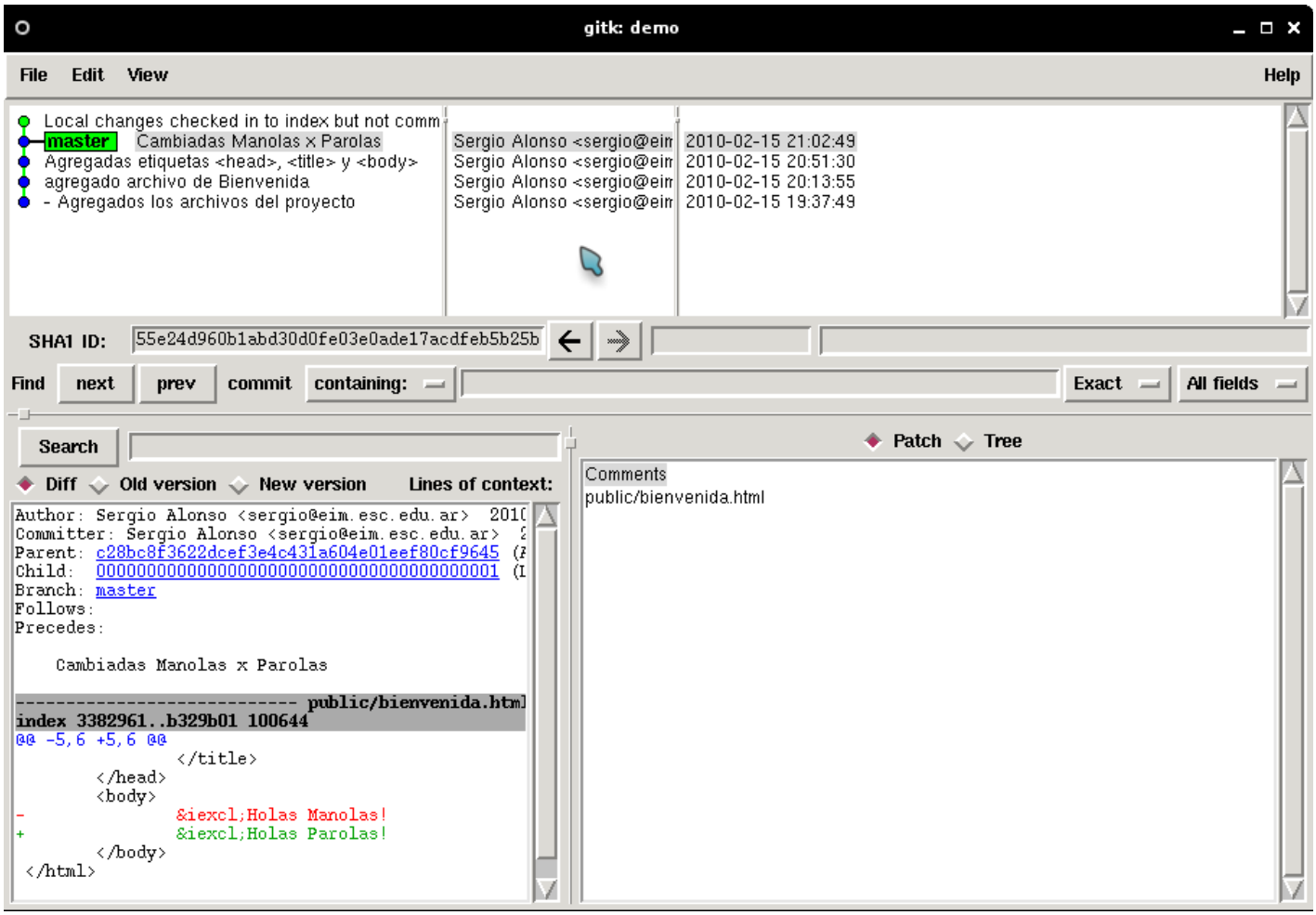


Interfaces gráficas

Si desea algo mas sofisticado, se alegrará saber de la existencia de **git-cola, git-gui, gitk, qgit, gitg** y el grandioso **meld** como comparador. Puede instalar cualquiera de estas herramientas mediante **apt-get**, y escogerlas llamándolas por su nombre para revisar el cambio en toda la carpeta, o haciendo un

```
git difftool archivo
```





Otra buena combinación consiste en observar los cambios, ejecutando en la raíz del proyecto a **git-cola.** Cuando hemos detectado alguno en particular, podemos comparar su historial de modificaciones yendo al menú Diff → Commits, y escoger commits anteriores (Start Commit), con respecto al último (End Commit). **Meld** se abrirá mostrando exactamente las lineas cambiadas**.**

Trabajar con ramas y números de versión

Las ramas de desarrollo son muy conocidas en el mundo del software libre. Y prácticamente una necesidad, ya que cobran cabal utilidad cuando muchas personas aportan trozos de código, *inestables hasta que se demuestre lo contrario*.

También en forma local podemos beneficiarnos de esta interesante propiedad que posee GIT: supongamos que deseamos introducir un cambio en nuestro sistema. Predecimos que la naturaleza de este cambio puede tener consecuencias en muchas secciones.

Entretanto, necesitamos una versión "estable" que ofrecer a nuestros clientes. Esto implica dos lineas de desarrollo. Al igual que en todos los proyectos de software libre, todo el tiempo hay ramas inestables, con funciones nuevas, en proceso de testing, y la rama estable, ya comprobada, y con menos sorpresas para los usuarios.

Aquí es donde las ramas de desarrollo y los números de versión aparecen para ayudarnos.

Todavía no hemos creado ramas, por lo tanto se considera que estamos en el tronco original de desarrollo, o *master*.

Respecto de los números de versiones, a modo de sugerencia, me permito explicar la forma en que se acostumbra a versionar en la comunidad de software libre. Bajo esta comunidad, a las versiones que todavía no están listas, se las expresa con decimales. Por ejemplo 0.1, 0.3, etc, reservando los enteros para los estables.

Algunos seguidores de creador del kernel Linux, (y del propio GIT), el emblemático Linus Torvalds, acostumbran a versionar la parte decimal, mediante números pares cuando se encuentran "listas para la comunidad". Los valores impares indican ramas para usuarios que testean o que requieren de funciones todavía no probadas.

Así, una versión 0.1, o 1.1 está pidiendo que la revisen, en tanto que una 0.2, o mejor aún, una 1.02 indica al menos un grado de estabilidad.

También se utilizan letras en los números de versión. Cuando una de las ramas testing ha llegado un cierto grado de madurez, y el lanzamiento es inminente, se espera que la comunidad colabore en el seguimiento de errores. En estos casos que se inicia una maratón de búsqueda de errores, se acompaña el número de versión por el término RC (Release Candidate).

De esta manera, vamos a bautizar nuestra versión como 0.1, dentro del tronco original *master*.


```
git tag 0.1 master
```


```
git tag 0.1
```

Supongamos ahora que queremos incluir una función de la cual no estamos completamente seguros. Realizaremos cambios por varias partes del proyecto, al punto que podríamos olvidarnos dentro de unos días cuantos cosas hemos cambiado. Prevemos también que enchufaremos gemas escritas por otras personas, que probablemente

modificaran varias partes automáticamente. A decir verdad, quisiéramos que esta versión sea utilizada solamente por el usuario que actúa de enlace con el sistema[11].

En los próximos capítulos realizaremos varios cambios. Si en un punto la aplicación empieza a soltar vapor por todos lados, como una locomotora desbocada, siempre podremos volver a nuestra rama estable.

A continuación, generamos una rama *testing*, basada en la actual *master:*

```
git branch testing master
```

¿En cual rama estamos parados?

```
git branch
```

```
* master
  testing
```

Cambiamos a la rama testing:

```
git checkout testing
```

```
M    public/bienvenida.html
Switched to branch "testing"
```

Borrando / modificando cosas antes del commit ("entrega")

Técnicamente, en todo momento podemos ocasionar un desastre e igualmente volver atrás. Supongamos que el gato pasa caminando sobre el teclado, y escribe[12]

```
rm -rf app/
```

```
del app /s /q && rd app /s /q
```

Esto equivale a destrozar la carpeta mas importante de la aplicación. De modo que le pedimos a GIT que reviva la aplicación tal como era en el último commit:

```
git add -A
```

11 En las metodologías ágiles de desarrollo, generalmente se involucra activamente a un representante de los clientes, o simplemente, "el usuario mas comprometido", quien se encarga de actuar como enlace entre los usuarios, sus necesidades, y los programadores. Para mas datos, revisar http://es.wikipedia.org/wiki/Desarrollo_ágil_de_software

12 Ah si. Mi gato puedo hacer eso.

```
git checkout -f
```

Si miramos ahora ¡ha vuelto todo a la normalidad!

Ten cuidado con lo que deseas

git checkout -f también puede quitar cosas de las que nos arrepentimos crear.
Por ejemplo, si decidimos pedirle a Rails que nos construya automáticamente una sección nueva, pero confundimos la sintaxis, podemos encontrarnos con decenas de archivos nuevos para borrar, regados a lo largo de toda la aplicación.
Mas adelante veremos en profundidad los mágicos scaffolds. Por ahora, soltamos la orden

```
rails generate scaffold Deposito domicilio:string encargado:string
capacidad:float
```

```
ruby bin\rails generate scaffold Deposito domicilio:string encargado:string
capacidad:float
```

Esta orden nos generara toda clase de cosas nuevas:

```
invoke active_record
create db/migrate/20100314175320_create_depositos.rb
create app/models/deposito.rb
invoke test_unit
create test/unit/deposito_test.rb
create test/fixtures/depositos.yml
route resources :depositos
invoke scaffold_controller
create app/controllers/depositos_controller.rb
invoke erb
create app/views/depositos
create app/views/depositos/index.html.erb
create app/views/depositos/edit.html.erb
create app/views/depositos/show.html.erb
create app/views/depositos/new.html.erb
create app/views/depositos/_form.html.erb
create app/views/layouts/depositos.html.erb
invoke test_unit
create test/functional/depositos_controller_test.rb
invoke helper
create app/helpers/depositos_helper.rb
invoke test_unit
create test/unit/helpers/depositos_helper_test.rb
invoke stylesheets
create public/stylesheets/scaffold.css
```

Mas adelante hablaremos de la conveniencia de nombrar en ingles a las clases, tablas y campos de los proyectos. "Deposito" se debería haber llamado "Depot", "encargado" debería haberse llamado "manager", y así. Es decir, hemos metido la pata.
¡A no preocuparse! En esta ocasión, hay toda clase de cosas nuevas. De modo que

ponemos al tanto a GIT de las novedades:

```
git add -A
```

Y como no hemos realizado todavía ninguna entrega ("commit"), rebobinamos todos los cambios, hasta el último commit.

```
git checkout -f
```

En realidad, Rails provee un mecanismo para destruir Scaffolds. En este caso hubiera bastado con ejecutar:

```
rails destroy scaffold Deposito
```

Pero aprendiendo a hacerlo con git, nos garantiza que podemos volver atrás muchas otras situaciones, independientemente del lenguaje o del framework elegido.

Borrando / modificando cosas después del commit

Esta es la situación mas frecuente. Realizamos cambios, y los entregamos ("commit") al repositorio. Luego nos arrepentimos.
Ejemplo: movidos por la inspiración, le agregamos algunas líneas al archivo **config/databases.yml**

```
echo "lineas tentativas" >> config/database.yml
```

```
echo "lineas tentativas" > config\database.yml
```

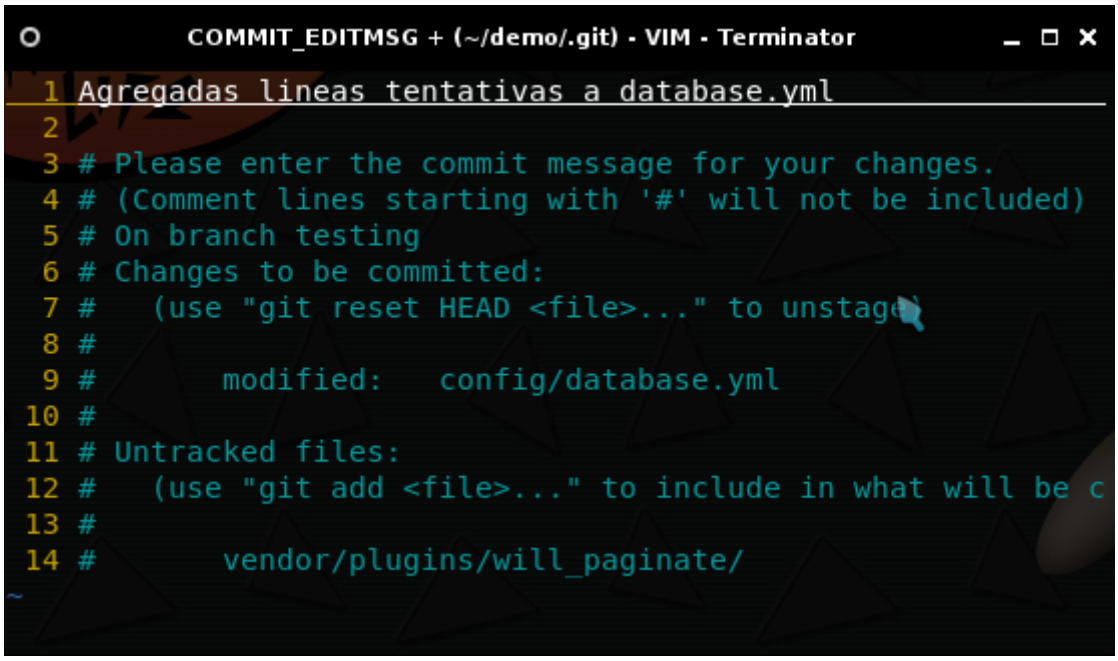
No habiendo archivos nuevos ni borrados, no hace falta (aunque no molesta) realizar el

```
git add -A
```

Es decir, al haber solo un archivo modificado, basta con realizar un git commit -a

```
git commit -a
```

Se abre el editor (vim o nano) solicitando le expliquemos la naturaleza de los cambios.



A continuación, borramos la carpeta doc. Al fin y al cabo, nadie la lee.

```
rm -rf doc/
```

```
del doc /s /q && rd doc /s /q
```

Al haber cambios en archivos que GIT conoce, utilizamos git add -A o simplemente agregamos -a al commit.

```
git commit -a -m "documentación borrada"
```

Finalmente, realizamos una acción típica de Rails: instalamos un componente de otra persona que nos facilita las cosas. En este caso, un plugin que permite con muy poco esfuerzo, presentar como "paginas" los listados muy extensos.

```
rails plugin install git://github.com/mislav/will_paginate.git
```

```
ruby bin\rails plugin install git://github.com/mislav/will_paginate.git
```

Este comando inicia una descarga de datos hacia la carpeta vendor/plugins.
Le decimos al repositorio que revise el árbol en busca de carpetas y archivos nuevos.

```
git add -A
```

```
git commit -m "agregado plugin para paginar fácilmente las salidas"
```

¡Sorpresa! Cuando iniciamos el server

```
rails server
```

```
ruby bin\rails server
```

Todo se colapsa miserablemente. El mensaje:

```
/usr/lib/ruby/1.8/yaml.rb:133:in `load': syntax error on line 23, col -1: `'
(ArgumentError)
     from /usr/lib/ruby/1.8/yaml.rb:133:in `load'
     from /usr/lib/ruby/gems/1.8/gems/railties-
3.0.0.beta/lib/rails/application/configuration.rb:64:in `database_configuration'
```

Algo en la configuración de las bases está mal. Al parecer nuestro cambio en **config/databases.yml** (hace tres commits), no fue del agrado del framework.

Sin embargo, no deseamos borrar ni el ultimo cambio (la instalación del plugin), ni el penúltimo (el borrado de doc/).
Si deseamos corregir errores puntuales, revisamos el log

```
git log --pretty=oneline
```

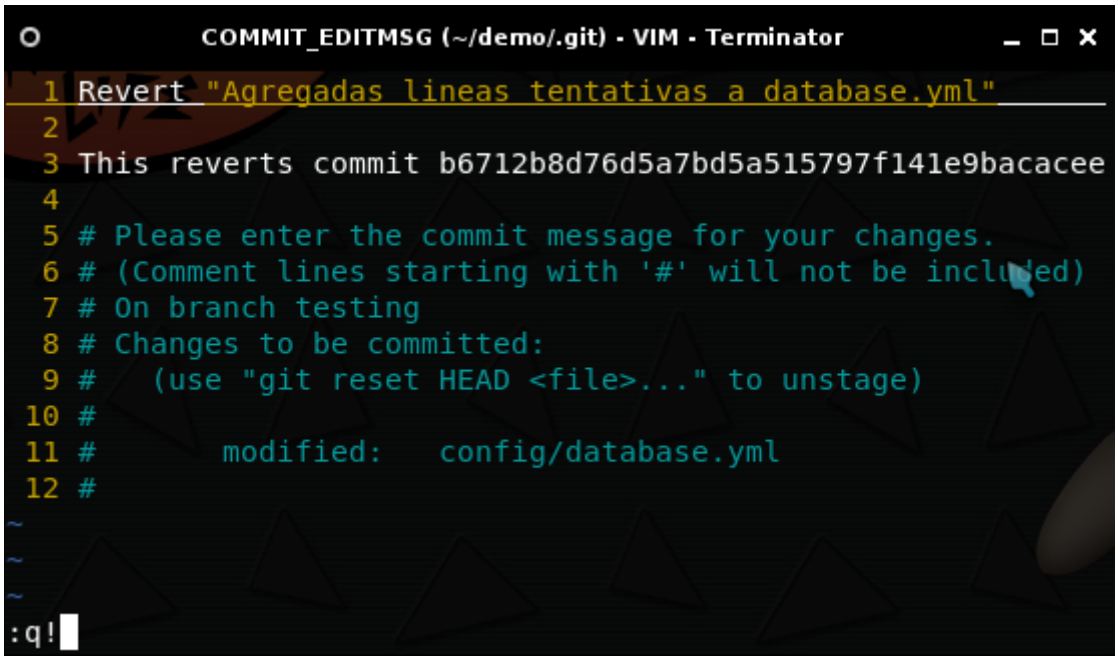
```
9b7a92b6cc7be5dbf259ac4a19e8515a25978e65 agregado plugin para paginar fácilmente las
salidas
9139194255481f5d564c68732f9a62b92a3afd0c borrada carpeta doc/
b6712b8d76d5a7bd5a515797f141e9bacacee5a0 Agregadas lineas tentativas a database.yml
be3eb69feacfd658f3e11ef87f1280316a1a7e9a Agregadas etiquetas <head>, <title> y <body>
53873d5e91f9e527ae765e80faf7bb00b66fdf91 agregado archivo de Bienvenida
8af7a6da99cf51fdb808e7571de87a8db8e22255 - Agregados los archivos del proyecto -
Configuradas algunas variables
```

(En rojo el cambio que nunca debió ser hecho).

Por lo tanto hago un revert "solo" a ese momento en el tiempo.

```
git revert b6712b8d76d5a7bd5a515797f141e9bacacee5a0
```

Un revert es como un commit: se abre el editor pidiendo las razones del arrepentimiento, aunque podemos ignorarlas simplemente saliendo.



Si ahora corremos nuevamente el server de pruebas, este debería iniciar sin problemas.

Duro de Resetear

Luego de un tiempo, descubrimos que el plugin que habíamos instalado... en realidad nos estorba. Y además, sacar la carpeta doc/ fue muy mala idea.

Podemos volver en el tiempo absolutamente todo, destruyendo todos los cambios, hasta el último commit:

```
git reset --hard HEAD^
```

Si deseamos volver # versiones hacia atrás (revisar primero con git log)

```
git reset --hard HEAD-#
```

La pregunta es: ¿cuando mi aplicación realmente me gustaba?
Si no deseamos "contar con los dedos" cuantos cambios se han ido sucediendo, podemos escoger uno de los hash que figuran en git log, mediante git reset --hard SHA1_HASH
En este caso:

 **git log --pretty=oneline**

```
9b7a92b6cc7be5dbf259ac4a19e8515a25978e65 agregado plugin para paginar fácilmente las salidas
9139194255481f5d564c68732f9a62b92a3afd0c borrada carpeta doc/
b6712b8d76d5a7bd5a515797f141e9bacacee5a0 Agregadas lineas tentativas a database.yml
be3eb69feacfd658f3e11ef87f1280316a1a7e9a Agregadas etiquetas <head>, <title> y <body>
53873d5e91f9e527ae765e80faf7bb00b66fdf91 agregado archivo de Bienvenida
8af7a6da99cf51fdb808e7571de87a8db8e22255 - Agregados los archivos del proyecto - Configuradas algunas variables
```

Volvemos casi al principio de la rama testing:

 git reset --hard be3eb69feacfd658f3e11ef87f1280316a1a7e9a

Si ahora hacemos

 **git log --pretty=oneline**

```
be3eb69feacfd658f3e11ef87f1280316a1a7e9a Agregadas etiquetas <head>, <title> y <body>
53873d5e91f9e527ae765e80faf7bb00b66fdf91 agregado archivo de Bienvenida
8af7a6da99cf51fdb808e7571de87a8db8e22255 Agregados los archivos del proyecto - Configuradas algunas variables
```

Todos los commits y cambios han dejado de existir y de ser tenidos en cuenta por GIT.

¡Basta de jugar! Volvemos a la rama master

```
git checkout master
```

Borramos el patio de juegos

```
git branch -d testing
```

Revert, Reset, Checkout

Supongamos que pudiéramos borrar un hecho puntual a través del tiempo. Esto sería el equivalente a un **revert**, que nos permite cambiar cosas que sucedieron el pasado.

Por su parte, **reset --hard** "rebobina la historia". Si a este comando le otorgáramos el libre albedrío, nos podría dejar parados en el medioevo sin posibilidad de volver.

Otra forma de cambio puntual es **checkout**, pero con la diferencia que, al estilo de "Volver al Futuro", nos lleva a un punto del pasado en el que todo cambio inicia una realidad (o rama) alternativa. Para mas datos, consultar http://crypto.stanford.edu/~blynn/gitmagic/intl/es/ch02.html

Resumen de este capítulo

Resumiré aquí los pasos importantes que hemos realizado:

Creamos la aplicación mediante

```
rails new demo
```

Entramos a la carpeta, mediante

```
cd demo
```

Corremos el server en modo testing, y lo dejamos mostrando sus operaciones internas:

```
rails server
```

```
ruby bin\rails server
```

Instalamos GIT. Posteriormente, creamos un repositorio local.

```
git init
```

Agregamos algunos archivos sin importancia (git add -A) entre commit y commit (git commit), los corregimos.

Luego, para probar algunos cambios mas complejos, creamos una rama testing, mediante la orden

```
git branch testing
```

La llenamos de basura, y la rompimos un poco. Luego deshicimos algunos (y todos) los cambios. Volvimos a la rama master, mediante

```
git checkout master
```

Y finalmente borramos la rama testing.

```
git branch -d testing
```

Capítulo 5

Capítulo 5: Hola Mundo en Rails

Cuando el navegador hace un solicitud al servidor Web, y esta es parseada a la aplicación Rails, la URL es descompuesta en busca de una **acción** asociada. Las **acciones** están definidas dentro de los **controladores**, quienes deciden que hacer con la URL cargada en el navegador.

Esto que en principio suena confuso, es común (y mucho mas difícil de configurar), en los demás frameworks del mercado. Sin embargo, es muy útil para crear direcciones legibles por seres humanos, y *spiders* de buscadores. Convengamos que es mucho mas fácil recordar:

```
http://www.mascotas.org.ar/adopciones/modificar/46
```

que la dirección.

```
http://www.mascotas.org.ar/adopciones.php?id=46&accion=modificar
```

Es decir: Rails analiza la URL (dirección) y la descompone para encontrar

- El controlador
- La acción
- Otros parámetros

En el siguiente capítulo vamos a intentar nuestra aplicación responda de forma distinta a las siguientes solicitudes:

- /hola

  → En este caso el navegador debe mostrar un simple "hola Mundo"

- /hola/4

  → Aquí en cambio, el 4 debe ser redirigido hacia alguna sección donde salude "calurosamente", es decir, cuatro veces.

Controladores y Acciones

"Tu ruta es mi ruta"

Anibal, "el Number One"

Antes de comenzar, vamos a crear nuevamente una rama donde probar cosas, sin poner en riesgo la aplicación tal como está. No es cuestión que nos echen la culpa desde el Colisionador de Hadrones cuando las cosas comiencen a explotar.

```
git branch holamundo
```

Cambiamos a esta nueva rama

```
git checkout holamundo
```

Opcional: esta rama tiene su origen en la versión 0.1 de master. Es decir, si alguien consulta su número de versión (**git tag**), la considerará estable[13].

Para dejar claro que se trata de un polígono de tiro, y que sus cambios solo deberían ser utilizados por desarrolladores cercanos al proyecto, le podemos situar un valor impar detrás, como **.1**

```
git tag 0.1.1 holamundo
```

Mas adelante veremos como situar nuestro proyecto en un repositorio público. Estos repositorios tienen la posibilidad de permitir a los usuarios que tiren de la rama que deseen. Si no se especifica otra cosa, tiraran de la master.

13 Nuevamente, ejemplos con el kernel linux:

- 1.**2**.1 → estable
- 1.**3**.5 → inestable
- 2.**5**.x → inestable
- 2.6.48 → estable

Volvamos a Rails, y al tratamiento de las URLs: el archivo que traduce las direcciones y dirige el tráfico hacia las acciones, se encuentra en **config/routes.rb**. Lo abrimos, y le agregamos las líneas en negrita:

Cuidado al copiar. Ciertas líneas, cuyo origen en el PDF es el ISO8859-1, se traducen mal en el UTF-8 de la terminal (Linux) o en el Windows-1852 (MSDOS)
Consejo: no sea vago y escriba manualmente los comandos.

```
Demo::Application.routes.draw do |map|
  get ':controller(/:action(/:id(.:format)))'
end
```

De esta manera, Rails descompone la dirección URL y la reenvía al *controlador → acción* adecuado. Crearemos un controlador

 `rails generate controller hola`

 `ruby bin\rails generate controller hola`

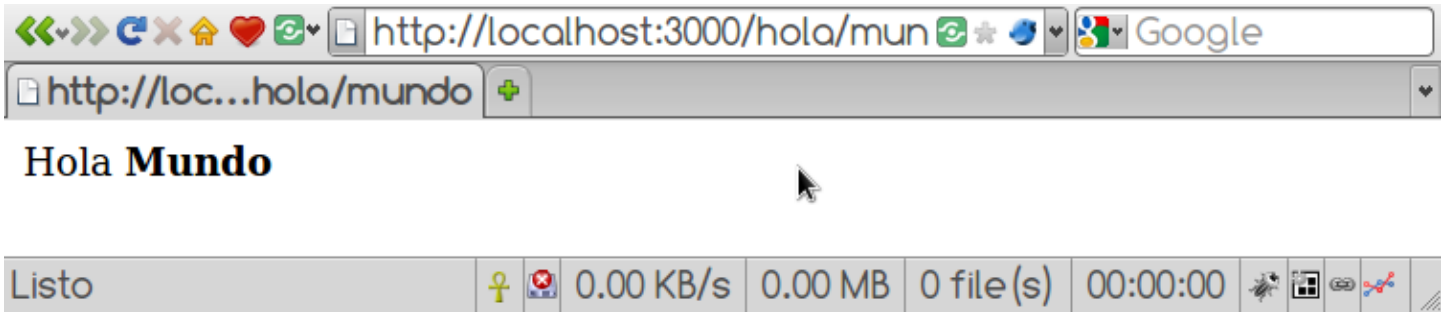
Abrimos **app/controllers/hola_controller.rb,** y le agregamos dos acciones:

```
class HolaController < ApplicationController
  def mundo
    render :text => "Hola <b>Mundo</b>"
  end

  def caluroso
    @valor=params[:id].to_i
  end
end
```

Reiniciamos (**Ctrl + C**) el servidor Web, y cargamos en el navegador la dirección

http://localhost:3000/hola/mundo

En este caso, lo que ha sucedido es muy simple: la acción **mundo** ha creado una pequeña vista mediante la orden **render,** con un breve mensaje.



Sin embargo, *deberíamos* haber creado un archivo llamado
`app/views/hola/mundo.html.erb`

A un purista del MVC no le gustaría este ejemplo. Hemos ensuciado una acción (¡en las barbas del controlador!) con etiquetas <b> de tipo html.
Así es: en ocasiones, para respuestas simples, se instala código propio de la "**Vista**" mediante **render**, en plena **Acción**. Por supuesto, no es conveniente, ya que con el tiempo se complicara la visualización y la separación de contenido.
Vamos al otro ejemplo. La idea es que la maquina salude "calurosamente": varias veces, de acuerdo a un valor pasado por URL.
Sin embargo, si intentamos con la dirección http://localhost:3000/hola/caluroso/4, obtendremos un error de Template (o "Vista"), lo cual es correcto: recordemos que en la segunda acción, no creamos una vista mediante render, sino que nos limitamos a recibir los datos, procesarlos (si hace falta), y dejarlos disponibles para la vista asociada. Aquí repito la parte en cuestión:

```
def caluroso
  @valor=params[:id].to_i
end
```

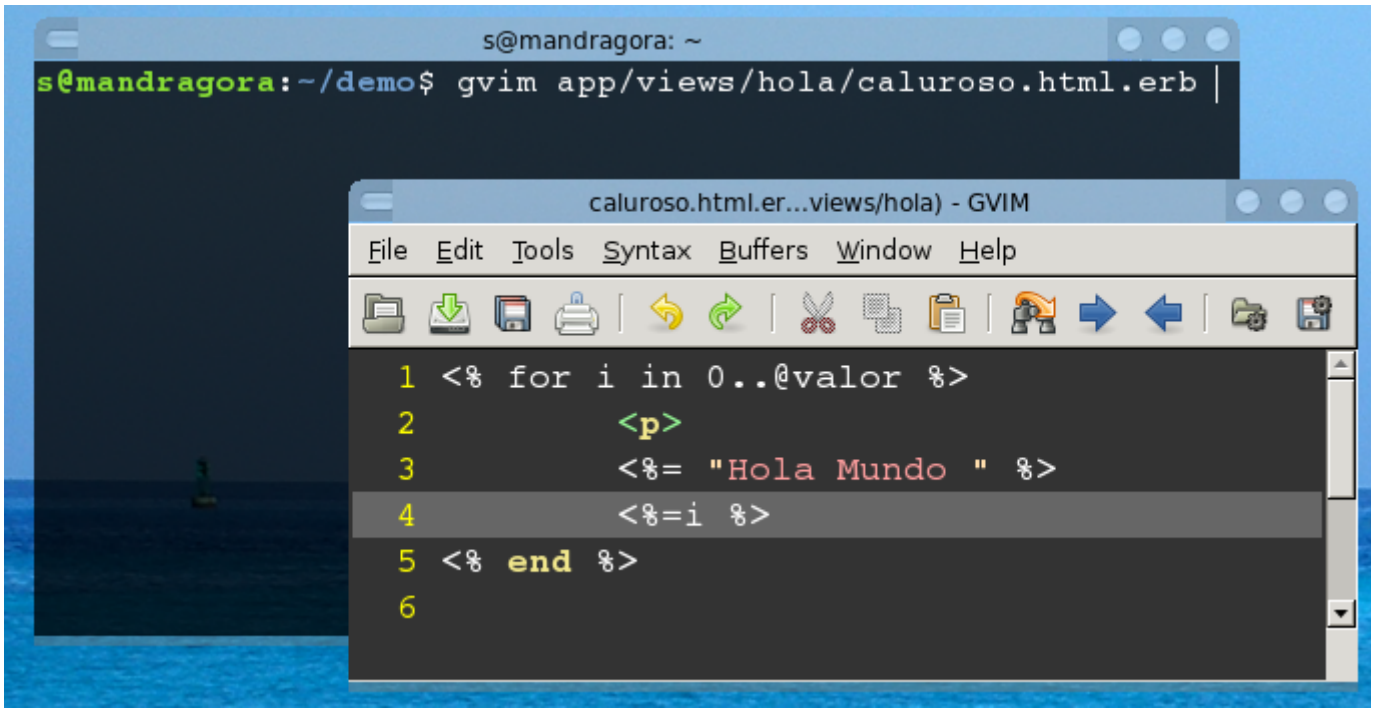
Podemos observar que el valor 4 pasado desde la URL, ha quedado disponible bajo la variable **params(:id)**. Por cierto, los valores que se pasan de esta manera, quedan expresados como cadenas, de modo que si queremos realizar operaciones matemáticas sobre él, debemos convertirlo a entero (.to_i).
El valor 4 es puesto a recaudo en un atributo **@valor,** accesible también por la vista, la cual se dispara inmediatamente después que termina la ejecución de la acción. En Rails, **cada acción tiene una vista asociada**. El error recién obtenido explica claramente que no hay

una vista (template) disponible.

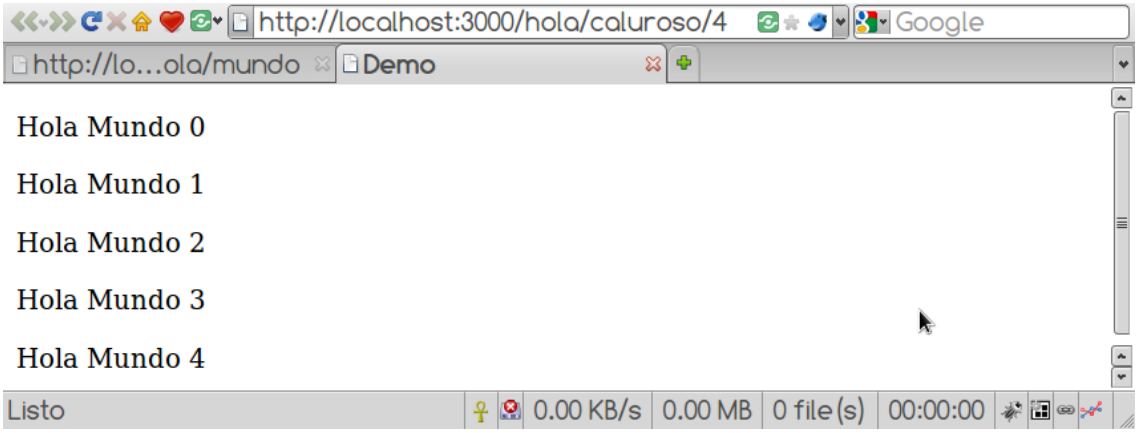
Para el caso de esta acción, fabricamos un archivo en **app/views/hola/caluroso.html.erb**, cuyo contenido debe parecerse al siguiente:

De nuevo: cuidado al copiar → pegar: la codificación de los caracteres puede variar en el destino.

```
<% for i in 0..@valor %>
      <p>
      <%= "Hola Mundo " %>
      <%=i %>
<% end %>
```



En estas líneas, recorremos de 0 a 4, saludando calurosamente en cada vuelta.



Para finalizar: este último archivo, **app/views/hola/caluroso.html.erb,** no existe en el repositorio:

```
git add -A
```

Recuerden que si algo anda mal, podemos empezar de nuevo la creación del hola mundo, mediante

```
git add -A
git checkout -f
```

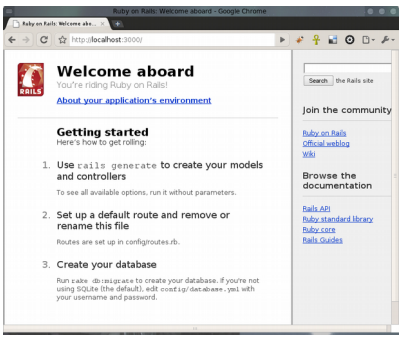
Reapuntando la página principal

Ahora que nuestra aplicación responde según los parámetros enviados por dirección http://, ya estamos en condiciones de enviar la simple solicitud http://localhost:3000 a una nueva página principal, la cual será lanzadera de toda la aplicación.

Como primera medida, borramos la página principal de Rails.

```
rm public/index.html
```

```
del public\index.html
```



Ahora abrimos nuevamente el archivo **config/routes.rb**, y le agregamos una línea

```
root :to => "hola#principal"
```

Si bien el controlador **hola** existe y tiene dos acciones adentro, la acción **principal** aún no existe. Editamos **app/controllers/hola_controller.rb** y le agregamos

```
    def principal
    end
```

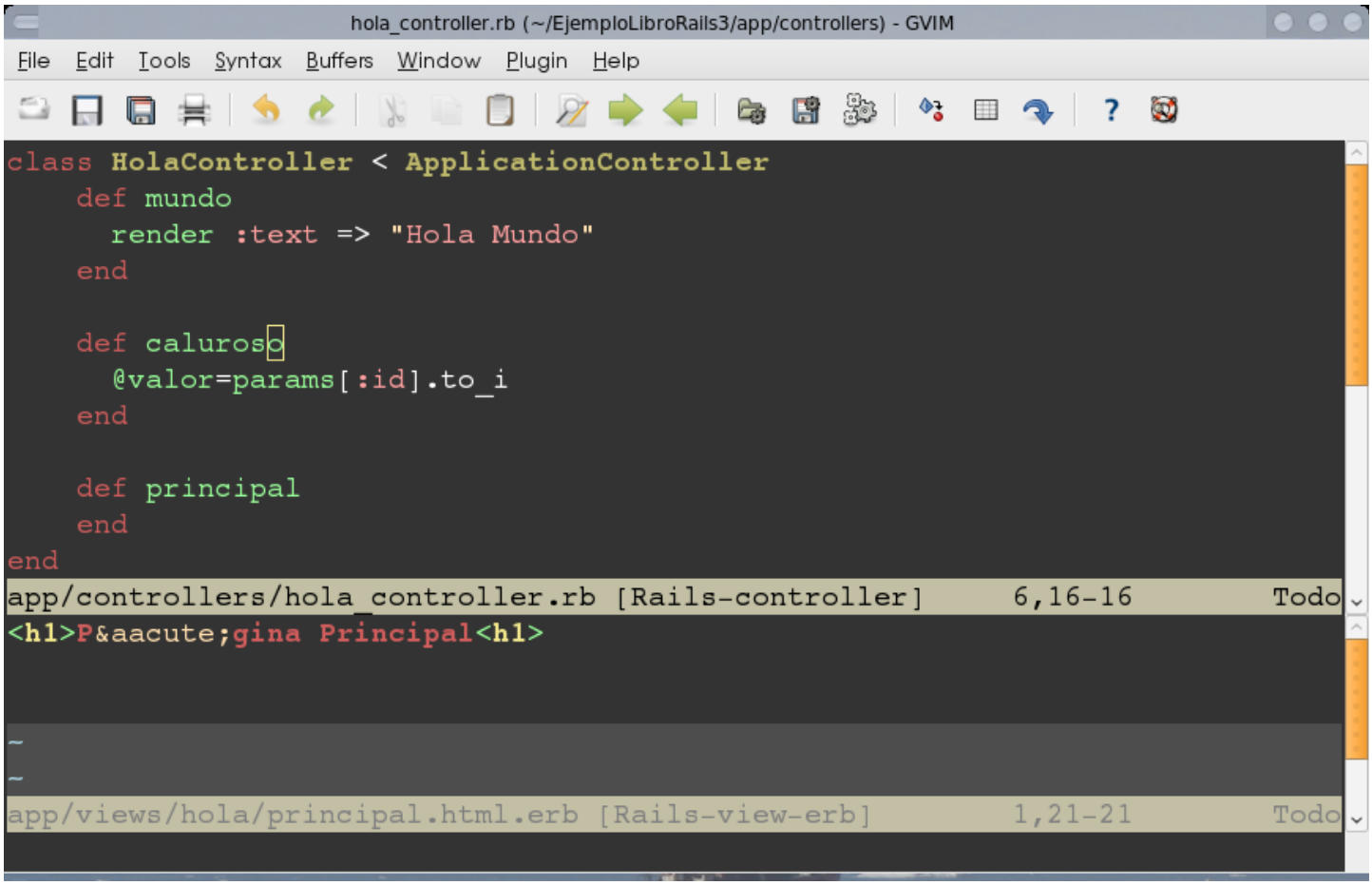
No es una acción muy impresionante que digamos. Pero cumple una simple función: cuando termine su ejecución, Rails buscará una vista asociada en **app/views/hola/principal.html.erb**

Este archivo no existe, de modo que lo creamos con el siguiente contenido

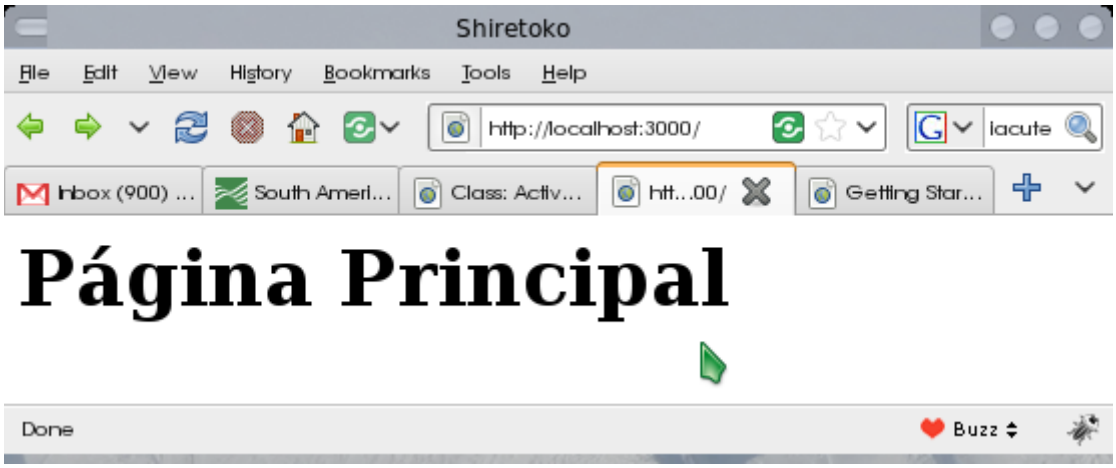
```
<h1>Página principal</h1>
```

o también, aunque ya no es necesario

```
<h1>P&aacute;gina Principal</h1>
```



Si todo anduvo bien, ahora http://localhost debería mostrarnos



Para finalizar, entregamos los cambios a GIT

```
git commit -m "holamundo funcionando"
```

Si recuerdan, estamos parados en la rama **holamundo.** Volvemos a la zona estable.

```
git checkout master
```

Y desde la rama master, mezclamos ("merge") las nuevas y flamantes innovaciones presentes de la rama holamundo.

```
git merge holamundo
```

Opcional: también podríamos actualizar la versión de la rama estable:

```
git tag 0.2 master
```

Capítulo 6

Capítulo 6: ActiveRecord

ActiveRecord es una biblioteca capaz de realizar *Mapeos Relacionales de Objeto*s (ORM). Las abstracciones ORM sirven para acceder a los datos de los motores SQL, sin preocuparse por la sintaxis de cada motor en particular. Además otorga la facilidad de tratar a las tablas como objetos, y a sus campos como atributos.

Esta librería puede ahorrarnos montones de líneas de código. Sin embargo, para lograr su magia, realiza una especie de pacto con el programador, que conviene respetar.

Para empezar, Rails es un framework que se adscribe a una filosofía denominada ***Convención en vez de Configuración***, lo cual acelera en gran medida el desarrollo.

Una de sus convenciones es trabajar en ingles. Esto no sorprende ni molesta a ningún programador acostumbrado a trabajar en factorías de software. Incluso es muy útil si se desea abrir y acelerar el desarrollo del proyecto atrayendo a programadores de la comunidad de Software Libre.

Así, si vamos a tener una tabla "Compañías", la trataremos en inglés. También debemos atenernos a las convenciones de Rails, y cuidando mayúsculas y tiempos verbales. Ejemplo:

- Tabla: **`companies`**
- Modelo: **`Company`**
- Archivo que contendrá la definición: **`app/model/company.rb`**
- La clave principal de las tablas siempre será **`id`**, no "id_company", "cod_company" o engendros similares.
- Las claves ajenas siempre terminarán en **`_id`**, ejemplo: `category_id, status_id, etc`

Se puede utilizar nombres compuestos "CamelCase" en los modelos, pero en ciertas partes para las cuales Rails debe mantener compatibilidad, se utilizará el _ separando ambos términos. Por ejemplo, en el seguimiento de erratas de este libro, utilizo un modelo **ErrorType** que describe el tipo de error descubierto por el lector (Técnico, Ortografía, Sugerencia). La tabla asociada será **error_types**. Y si otra tabla posee un índice ajeno que apunta a esta

tabla, el nombre de la columna será **error_type_id**. La relación descripta en el modelo de la otra tabla, por ejemplo **belongs_to**, será hacia **:error_type**.

Estas convenciones impacientan a muchos programadores, pero agrada a la mayoría, ya que le permite a Rails trabajar automáticamente, y prescindir de fastidiosas configuraciones XML, como las existentes en Struts, Spring, o diversos métodos de persistencia.

Por esta razón los frameworks que usan *Convención en vez de Configuración* son tildados de "fascistas". En en el buen sentido de la palabra: fascismo se refiere a la imposición de una idea por parte de muchas personas. Pero sucede que las ideas de Rails son muy buenas, y han sido consensuadas por muchos y excelentes programadores.

Este aspecto fascista se detecta en dos aspectos: el idioma, y las convenciones.

Ya vimos las convenciones. En cuanto al idioma, se recomienda trabajar lo mas posible en ingles, puesto que Rails "intuye" mejor en esta lengua. Por supuesto, hay diversos hacks o "inflections", que permiten trabajar con un Rails españolizado. Sin embargo, no recomiendo molestarse en hackear Rails: el inglés, nos guste o no, es la lengua que se habla en el mundo de la programación.

No recibiremos mucha ayuda en los foros si nos empeñamos en que chinos, rusos, árabes, alemanes, etc, deban entender que el significado de la palabra "deposito". Si tenemos una buena idea, y creemos que se puede convertir en algo realmente grande, podemos subir nuestro proyecto a un repositorio publico y llamar a toda la comunidad de software libre para que colabore. Todos agradecerán que nos hayamos molestado en acudir al Google Translate, para cambiar el término "Depósito" por "Depot".

Coincidirá conmigo en que aumentarán la posibilidades que otras personas nos ayuden en nuestro proyecto, si mantenemos compatibles todas las piezas.

Usted mismo entrará muchas veces a RubyForge o a GitHub a buscar algún pedazo de código que le ahorre semanas de trabajo. Le aseguro que agradecerá a mas de un hindú, que éste haya ignorado su natal *hindi,* y en lugar de utilizar "जमा" para definir un "deposito", lo haya expresado como "depot". Al menos en español compartimos el mismo juego de símbolos latinos. ¡Piense en el trabajo que le ha tomado al hindú, cambiar hasta su alfabeto de nacimiento!

Finalmente, el hecho que la aplicación *por dentro* piense en inglés, no significa que *hable en inglés con el usuario*. Rails viene muy bien preparado para internacionalizar ("i18n") la

interface. Nuestros usuarios, y los usuarios de nuestros programadores hindúes, podrán cambiar el idioma a elección: es lo que se dice... ¡una aplicación bien construida!

El segundo aspecto a analizar es la nomenclatura que *deberíamos* respetar en tablas, índices y nombres de clase. La cual no debería ser complicada de seguir... a excepción que esté pensando en reprogramar una aplicación anterior. No se preocupe: empleando la aclaración necesaria en los modelos, se puede usar cualquier nombre en las tablas e índices existentes.

Ejemplo de creación de modelo, y alteración de las convenciones

Atención: esta sección ha sido construida con el propósito que Ud. entienda la forma en que razona Rails durante sus procesos automáticos, e incluso aprenda a modificar su comportamiento, respecto de aplicaciones escritas con anterioridad en otro lenguaje. Sin embargo, la idea no es personalizar cada nueva aplicación que inicie. Las convenciones de Rails deberían respetarse simplemente porque *introducen a buenas prácticas de programación*.

```
rails generate model Empresa nombre:string cod_empresa:integer
```

```
ruby bin\rails generate model Empresa nombre:string cod_empresa:integer
```

```
      invoke  active_record
      create    db/migrate/20100314191942_create_empresas.rb
      create    app/models/empresa.rb
      invoke    test_unit
      create      test/unit/empresa_test.rb
      create      test/fixtures/empresas.yml
```

En este punto Rails calcula que la tabla se llamará empresas**,** y que el id será un campo llamado **id**. Sin embargo, supongamos que tenemos una tabla (sin datos, cuidado!) llamada **empresa** (en singular), con clave principal **cod_empresa**, y un campo **nombre.**

1. Durante la orden **migrate**, hemos creado un archivo
   db/migrate/20100314191942_create_empresas.rb
   Este archivo cumple dos propósitos: en unos instantes, será utilizado por un script llamado rake, que utilizará las definiciones existentes en él para:

- Crear físicamente las tablas en base de datos.
- Crear / Actualizar un archivo **db/schema.rb**, el cual posee clases con información de todas las tablas existentes en la base de datos: el corazón de ORM o "Object Relational Map".

Deberíamos *interceptar* este archivo antes que Rails lo procese.

a) Lo abrimos y lo alteramos de la siguiente manera (cambios en negrita):

```
class CreateEmpresas < ActiveRecord::Migration
  def self.up
    create_table :empresa do |t|
      t.column :nombre, :string
      t.column :cod_empresa, :integer
      t.timestamps
    end
    add_index :empresa, :cod_empresa
  end

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

b) Ejecutamos rake:

```
rake db:migrate
==  CreateEmpresas: migrating
================================================
-- create_table(:empresa) -> 0.0016s
-- add_index(:empresa, :cod_empresa) -> 0.0003s
==  CreateEmpresas: migrated (0.0021s)
=======================================
```

Antes de seguir, podemos constatar la creación física de la tabla, mediante el plugin **SQLite Manager** de Firefox (Herramientas → SQLite Manager). También es recomendable mirar dentro de **db/schema.rb**, la definición de la clase Empresa. EL otro detalle, no necesario, es el índice que hemos creado con add_index, el cual tiene por solo propósito mejorar la velocidad de acceso a los datos.

2. Así como en **schema.rb** están las definiciones de las tablas existentes en nuestra

aplicación, cada una de las tablas tiene un modelo relacionado.

Abrimos **app/models/empresa.rb** y lo alteramos según nuestras necesidades. En este caso:

```
class Company < ActiveRecord::Base
  set_table_name 'empresa'
  set_primary_key 'cod_empresa'
end
```

Recuerde (una vez mas), que si durante este capítulo ha cometido un error, puede comenzarlo de vuelta borrando los cambios:

```
git add -A
git checkout -f
```

Comprobar todo

Una manera de averiguar si Ruby es capaz de utilizar esta tabla díscola, es decir, no atada a las convenciones, es crear algunos datos desde la consola:

 **rails console**

 **ruby bin\rails console**

```
irb(main):001:0> unaempresa = Empresa.new
=> #<Empresa id: nil, nombre: nil, cod_empresa: nil, created_at: nil, updated_at:
nil>

irb(main):002:0> unaempresa.nombre = "Pastas Alfredo"
=> "Pastas Alfredo"

irb(main):003:0> unaempresa.save
=> true
```

¿Será cierto que ahora Rails respeta nuestro cod_empresa en lugar de su id?

```
irb(main):004:0> unaempresa
=> #<Empresa id: 1, nombre: "Pastas Alfredo", cod_empresa: 1, created_at: "2010-
03-14 19:54:05", updated_at: "2010-03-14 19:54:05">
```

```
irb(main):005:0> unaempresa.cod_empresa
=> 1
irb(main):006:0> unaempresa.id
=> nil
```

Nota: de realizar algún cambio en los modelos, se debe volver a ingresar a **rails console**, y reiniciar **rails server**

Resumen

Espero que haya disfrutado de esta sección. Si bien es compleja, es muy posible que deba tenerla en cuenta para cuando migre sus aplicaciones actuales a Rails.
No obstante, **no utilizaremos la clase Empresa ni sus modelos**, sino sus equivalentes en inglés, de modo que daremos vuelta atrás todas estas anti convenciones:

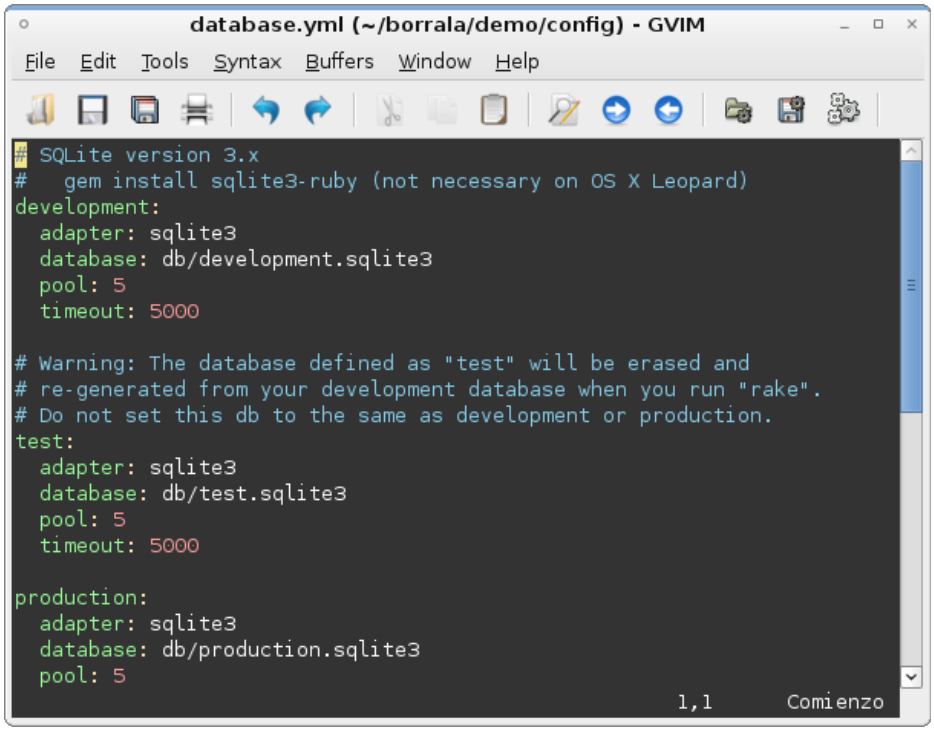
 `git add -A`

 `git checkout -f`

Desarrollar, Testear, Producir

Vamos a conocer una de las prácticas que mas sorprende a aquellos que se acercan a Rails. Empecemos abriendo el archivo **`config/databases.yml`**

Una vez adentro, podremos constatar tres entornos de producción en Rails: **Development, Test y Production.**



Todos estos entornos pueden trabajar en simultáneo: incluso en el server de producción podemos hacer pruebas, e invitar a los usuarios mas valientes a que entren por el puerto 3000 a probar las innovaciones.

Asociados a estos entornos, se encuentran las bases, las cuales pueden ser de distintos fabricantes. Una herramienta mágica llamada **rake** se encargará de migrar datos entre ellas. Todavía no usamos **rake**, pero si a titulo de curiosidad quiere asombrarse de algunas de sus posibilidades, pruebe ejecutar

```
rake -T
```

Como sugerencia, una manera de no preocuparse por los detalles de implementación de la base de datos, consiste en utilizar SQLite, al menos para las etapas development y test. Luego puede usar adaptadores para otras cosas mas potentes. Entonces rake se encargará de crear una basada **production** sobre, por ejemplo, PostgreSQL, con la estructura de la base development.

Las bases RDBMS normalmente atienden en un puerto, custodiado por un servicio o "daemon", y ocupando recursos incluso cuando no se las utiliza. Pero para usar SQLite, no hace falta instalar mas que unos pocos paquetes básicos y las gemas. Una mala comparación, podrían ser los archivos .mdb de Access.

Las bases, al estilo de Access, o de FoxPro, se guardan en la carpeta **/db**.

Por que utilizar SQLite? En realidad puede utilizar toda clase de motores SQL. Pero este pequeño motor se recomienda mucho en la comunidad de software libre por las siguientes razones:

- Es muy rápido para consultas

- Es muy portable: puede ser empaquetado en motorcitos Ajax como Google Gears, Adobe AIR y otros, para cachear datos del lado del cliente.

El principio de trabajo de SQLite es muy simple. **Podríamos** crear la base mediante algún administrador de SQLite, o realizando:

```
sqlite3 db\development.sqlite3
```

… pero esto implicaría que *queremos* aprender a hacer las cosas al estilo SQLite3. Pero recordemos que nuestro ORM, ActiveRecord, desea que **no nos preocupemos por el motor SQL.**

Capítulo 7

Capítulo 7: A levantar los andamios: Scaffolds

En cierta manera, los scaffolds son los caballos de batalla de Rails.

Se trata de unos scripts capaces de crear modelos, vistas y controladores a partir de unos pocos datos. Son muy buenos para crear ABMs (altas, bajas y modificaciones) en forma rápida, para luego personalizarlos a gusto([14]).

Comenté hace unas pocas hojas, la idea de trabajar, mediante las convenciones en ingles. Por ello hemos borrado toda referencia a **Empresa**, a la vez que comenzamos con un nuevo modelo **Company**. El script scaffold nos creará además las rutas, algunas vistas, e incluso el controlador.

 `rails generate scaffold Company name:string address:string active:boolean`

 `ruby bin\rails generate scaffold Company name:string address:string active:boolean`

```
      exists  app/models/
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/companies
      exists  app/views/layouts/
      exists  test/functional/
      exists  test/unit/
      create  test/unit/helpers/
      exists  public/stylesheets/
      create  app/views/companies/index.html.erb
      create  app/views/companies/show.html.erb
      create  app/views/companies/new.html.erb
      create  app/views/companies/edit.html.erb
      create  app/views/layouts/companies.html.erb
      create  public/stylesheets/scaffold.css
      create  app/controllers/companies_controller.rb
      create  test/functional/companies_controller_test.rb
```

14 Los ABMs pueden ser encontrados en páginas en inglés como CRUD (Create, Read, Update, Delete).

```
      create  app/helpers/companies_helper.rb
      create  test/unit/helpers/companies_helper_test.rb
       route  map.resources :companies
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
      create    app/models/company.rb
      create    test/unit/company_test.rb
      create    test/fixtures/companies.yml
      create    db/migrate
      create    db/migrate/20100111165913_create_companies.rb
```

Corremos la migración:

  **rake db:migrate**

```
==  CreateCompanies: migrating ================================================
-- create_table(:companies)
   -> 0.0057s
==  CreateCompanies: migrated (0.0061s) =======================================
```

Abrimos Firefox en la dirección http://localhost:3000/companies

Nada mal para 2 (dos) líneas.

Observe la línea que construye la clase o "modelo", app/models/company.rb. También

podrá apreciar la creación del controlador, y de varias vistas correspondientes a acciones ABM (index, new, edit, show).

Los scaffolds son útiles para realizar rápidamente el ABM de una tabla, y aprender un poco de la *rails ways* de hacer las cosas.

Avisamos a GIT que comience a seguir a los nuevos archivos:

```
git add -A
```

Si está todo bien, entregamos los cambios mediante

```
git commit -m "Scaffold de Compañias"
```

Caso contrario

```
git checkout -f
```

En esta fase, es imprescindible revisar la magia detrás de scaffold

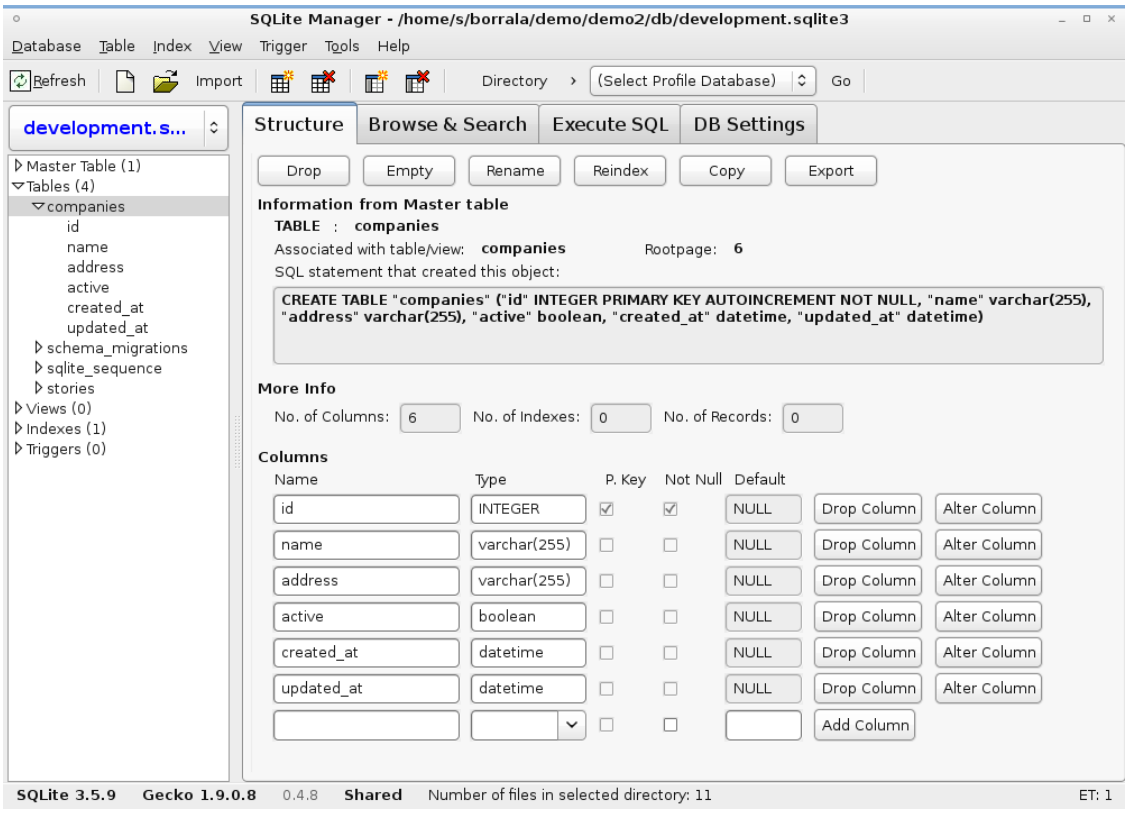
Caminar hasta la locomotora

¡Conozcamos al maquinista! Vamos al corazón del **Modelo Vista Controlador**, la carpeta **\app**.

Allí podremos constatar la creación de un montón de cosas. **Deberíamos tomarnos unos largos minutos en revisar:**

- app**controllers**\companies_controller.rb
- app**models**\company.rb
- app**views**\companies
  - edit.html.erb
  - index.html.erb
  - new.html.erb
  - show.html.erb

Muy bien los andamios... pero ¿donde almacenan los datos?



Podemos constatar las inserciones mediante algún administrador para SQLite, tal como **SQLite Manager** (plugin de Firefox). Vimos sus detalles de instalación en el Capítulo 1.

Mediante SQLite Manager podemos abrir el archivo **db\development.sqlite3.** Allí podremos constatar la creación de la tabla y su eventual llenado desde la vista new.

Por cierto: **rake** nos ha construido todos los campos, mas el **id**, y dos campos muy útiles: **created_at** y **updated_at**

¿Cómo sabía **rake** que cosas crear?

Si recordamos la salida de *generate scaffold,* notaremos la presencia de un archivo

db/migrate/20100506030728_create_companies.rb

Este archivo contiene la definición respecto de como debería ser creada la clase, su reflejo en db/schema.rb, y sus campos.

¿Como lo hizo **rake**? Es un buen momento para revisar el archivo de bitácora.

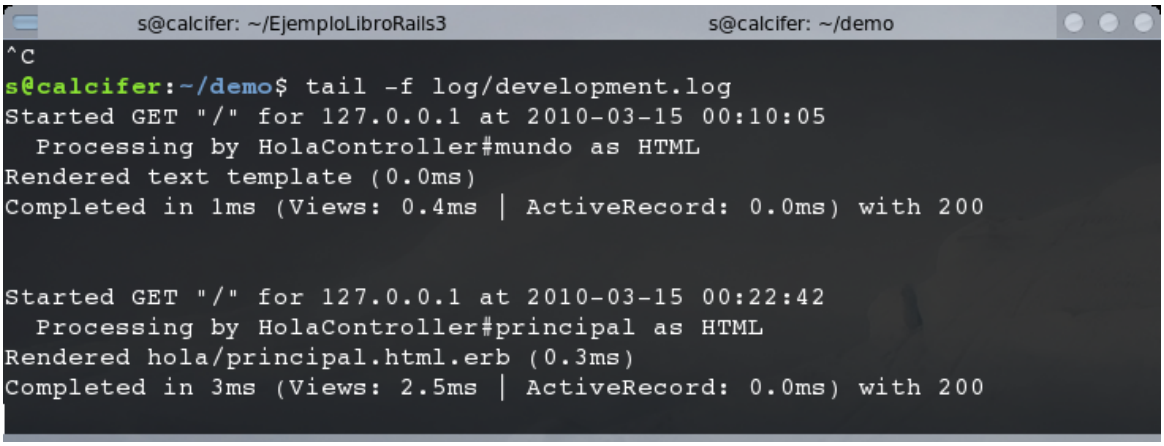
La bitácora del maquinista

Todo el tiempo Rails lleva un registro de todo lo que hace.

```
more log/development.log
```

```
more log\development.log
```

Allí nos encontraremos con un montón de instrucciones SQL que hecho ActiveRecord por nosotros: instrucciones CREATE TABLE, CREATE DATABASE, etc.



Nota 1: En **Windows,** debido a que este archivo está escrito en utf8, y MSDOS trabaja en ASCII, la salida en pantalla contendrá algunos caracteres extraños.

Nota 2: En Linux se acostumbra a usar **tail -f** en lugar del anticuado **more**. En este caso,

```
tail -f log/development.log
```

Este comando mantiene capturada la terminal mientras relata ("**tail**") los cambios al final de un archivo. Si queremos cortarlo y retomar la consola, pulsamos **Ctrl + C**

En nuestro mismo código también podemos volcar al log. En cualquier parte de una acción, podemos agregar líneas al estilo:

```
logger.info("#{Time.now} Borrando una empresa cuyo ID es el ##{@company.id}")
```

Esto es muy útil para dejar registro en log/ sobre ciertos sucesos importantes de la vida de la aplicación. Yo uso estos registros para monitorear momentos críticos, como ciertas noches en que mis servidores sincronizan sus bases.

Capítulo 8

Capítulo 8: Validaciones

Validar es un paso aburrido durante el desarrollo de toda aplicación. Su objeto es impedir la entrada de datos que corrompan la base. No, no se trata de dinero. Se trata de valores vacíos, repetidos, letras en lugar de números, etc. Desgraciadamente, es un paso que los programadores deben hacerlo por duplicado en todas las etapas de la aplicación: durante los new, los edit, etc.

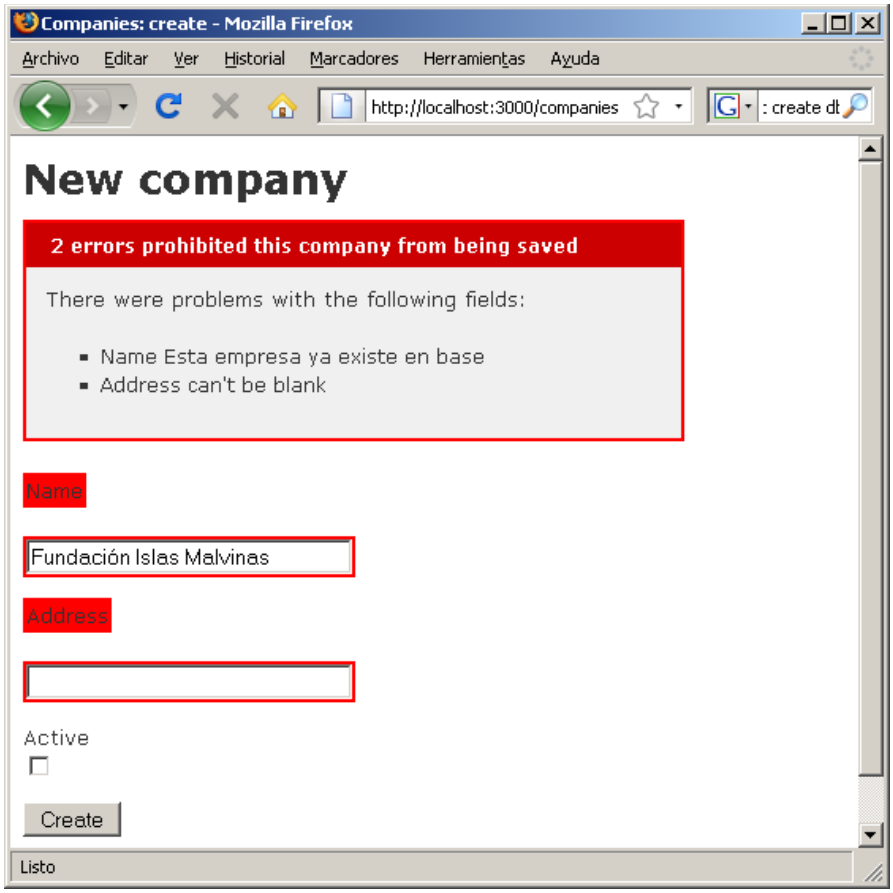
Rails habíamos mencionado que se adscribe al modelo DRY (no te repitas), de modo que aclaramos nuestra regla de validación en el modelo, y ActiveRecord cuidará que **nunca** ingresen datos sucios.

Abrimos **app\models\company.rb**, y lo dejamos de la siguiente forma:

```
class Company < ActiveRecord::Base
     validates_uniqueness_of :name, :message => "Esta empresa ya existe en base"
     validates_presence_of :name, :message => "necesito un nombre"
     validates_presence_of :address
end
```

¡Vaya! ¡Eso fue muy rápido!

Y si Ud, mi querido programador de PHP está lagrimeando de la emoción, busque otro pañuelo, e intente editar un registro y borrar el valor en *address*.



Capítulo 9

Capítulo 9: La imagen no es nada: Layouts y CSS

Los layouts se renderean antes que las vistas, y técnicamente, las "envuelven".

Sirven para alojar barras de botones, encabezados, publicidad, etc. Por cada controlador generado mediante scaffold, hay un layout general. Y si deseamos, podemos crear un layout general, y hacer que todos los controladores busquen al mismo antes de renderear sus vistas.

Los layouts son opcionales, pero nos aligeran el trabajo de tener que diseñar estos elementos en CADA componente ABM.

Por ejemplo, para todas las vistas alojadas en **app/views/movies**, existirá un layout en **app/views/movies/movies.html.erb**

De esta manera, los layouts son una solución elegante, que hasta ahora solo se había solucionado con frames, mas propios de los años 90, o mediante las plantillas de Dreamweaver (no CSS, y dependientes de este oneroso editor).

En un alarde de metrosexualismo, los creadores de Rails diseñaron los layouts para que se conectaran a las vistas mediante DIVs (por CSS), y no mediante tablas.

Esto, en un principio resulta molesto, ya que debemos entender un poco de plantillas CSS. Sin embargo, **es una correcta práctica de programación**.

Citando al libro Introducción a CSS, escrito por Javier Eguíluz Pérez, y presente gratuitamente en www.librosweb.es:

"El diseño de las páginas web habituales se divide en bloques: cabecera, menú, contenidos y pie de página. Visualmente, los bloques se disponen en varias filas y columnas. Por este motivo, hace varios años la estructura de las páginas HTML se definía mediante tablas.

El desarrollo de CSS ha permitido que se puedan realizar los mismos diseños sin utilizar tablas HTML. Las principales ventajas de diseñar la estructura de las páginas web con CSS en vez de con tablas HTML son las siguientes:

- ***Mantenimiento**: una página diseñada exclusivamente con CSS es mucho más fácil de mantener que una página diseñada con tablas. Cambiar el aspecto de una página creada*

con CSS es tan fácil como modificar unas pocas reglas en las hojas de estilos. Sin embargo, realizar la misma modificación en una página creada con tablas supone un esfuerzo muy superior y es más probable cometer errores.

- ***Accesibilidad**: las páginas creadas con CSS son más accesibles que las páginas diseñadas con tablas. De hecho, los navegadores que utilizan las personas discapacitadas (en especial las personas invidentes) pueden tener dificultades con la estructura de las páginas complejas creadas con tablas HTML. No obstante, diseñar una página web exclusivamente con CSS no garantiza que la página sea accesible.*
- ***Semántica**: aunque resulta obvio, las tablas HTML sólo se deben utilizar para mostrar datos cuya información sólo se entiende en forma de filas y columnas. Utilizar tablas para crear la estructura completa de una página es tan absurdo como utilizar por ejemplo la etiqueta <ul> para crear párrafos de texto.*

Por estos motivos, la estructura basada en tablas ha dado paso a la estructura basada exclusivamente en CSS. Aunque crear la estructura de las páginas sólo con CSS presenta en ocasiones retos importantes, en general es más sencilla y flexible. "

¡Ánimo! No es tan difícil: en http://del.icio.us/karancho/css menciono varias fuentes de inspiración, generación de layouts, menúes, y todo lo necesario para aprender rápidamente diseños e interfaces profesionales.

Capítulo 10

Capítulo 10: Uno para Todos...

"*Encuentra un trabajo que te guste*
y no volverás a trabajar
ni un sólo día de tu vida".
Confucio

En esta fase ahondaremos en algunos aspectos mas avanzados de Rails.
Debo admitir que todo lo que habíamos logrado hasta ahora, lo podríamos haber realizado en PHP, sin tener que aprender un lenguaje nuevo.
Muchos principiantes prefieren trabajar desde el principio en forma casi artesanal (PHP / ASP), para descubrir que un proyecto puede crecer hasta hacerse muy difícil de mantener.
Cuando en un proyecto, sus tablas, relaciones, tiempo de carga, etc, comienzan a crecer, es cuando los frameworks demuestran su valía. Tradicionalmente se emigra a Java / .NET, con el larguísimo e intenso entrenamiento que esto supone, al menos de 1 ½ año, y practicando como un violinista[15].
Afortunadamente, en estos últimos años Python (con Dyango), y Ruby (con Rails, Sinatra y otros) han traído opciones frescas, originales y potentes al mundo del desarrollo web.
En el mundo de los frameworks web reinan indiscutiblemente Java (Struts, Spring, JSF), debido a su comprobada flexibilidad y robustez.
Sin embargo, estos últimos años, los mismos programadores que pasan su semana laboral en Java y .NET, se conectan los fines de semana a los foros de Python y Ruby, lo cual indica un inminente cambio de paradigma. Los moderadores de los foros concuerdan: al menos, programar en Ruby y en Python es *divertido*[16].

15 Aproximadamente 4 (cuatro) horas diarias.

16 Fuente:
https://web.archive.org/web/20111025122551/http://www.vivalinux.com.ar/articulos/python-y-ruby-mas-divertidos

Tablas Combinadas

Las compañías poseen mas de un departamento: **Ventas**, **Atención al Cliente**, etc.

Generaremos modelo y controlador de departamentos en un solo paso:

 `rails generate scaffold department company_id:integer name:string`

 `ruby bin\rails generate scaffold department company_id:integer name:string`

Si observamos la salida generada, veremos que Rails ha generado archivos para cuando estemos listos para "migrar", esto es, crear en el motor SQL un reflejo funcional de nuestro modelo.

```
exists  app/models/
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/departments
      exists  app/views/layouts/
      exists  test/functional/
      exists  test/unit/
      exists  test/unit/helpers/
      exists  public/stylesheets/
      create  app/views/departments/index.html.erb
      create  app/views/departments/show.html.erb
      create  app/views/departments/new.html.erb
      create  app/views/departments/edit.html.erb
      create  app/views/layouts/departments.html.erb
   identical  public/stylesheets/scaffold.css
      create  app/controllers/departments_controller.rb
      create  test/functional/departments_controller_test.rb
      create  app/helpers/departments_helper.rb
      create  test/unit/helpers/departments_helper_test.rb
       route  map.resources :departments
  dependency  model
      exists    app/models/
      exists    test/unit/
      exists    test/fixtures/
      create    app/models/department.rb
```

```
      create    test/unit/department_test.rb
      create    test/fixtures/departments.yml
      exists    db/migrate
      create    db/migrate/20100113040955_create_departments.rb
```

Concretamente, y entre muchas otras cosas, ha generado un archivo

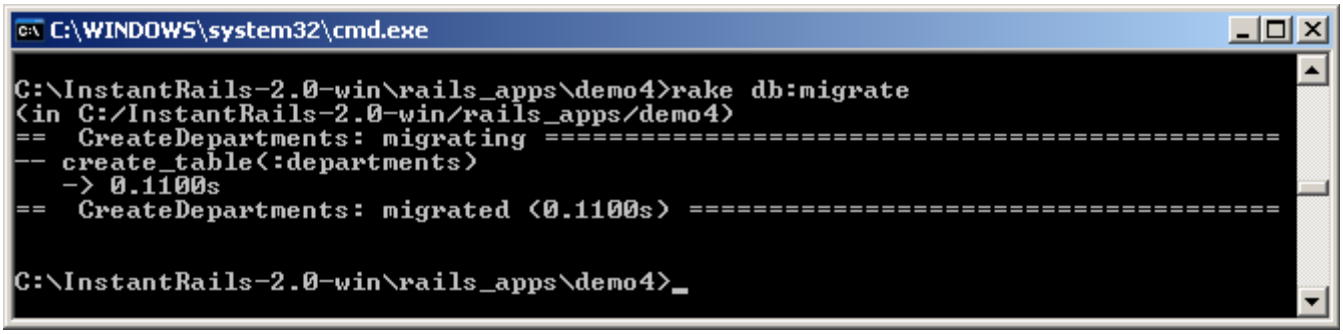
db/migrate/2009nnnnnnnnnn_create_departments.rb

Le le damos una mirada, y chequeamos que aparece un campo (creado por nosotros durante el scaffold) llamado a propósito **company_id:integer** esto significa dos cosas:

- Debe existir una clase (modelo) Company por algún lado. El scaffold la ha creado por nosotros en **app\models\department.rb**
- Según la convención de Rails, cuando mencionamos un campo terminado en **_id**, es por que lo estamos declarando como **clave externa de otra tabla**.

  **rake db:migrate**

```
==  CreateDepartments: migrating ==============================================
-- create_table(:departments)
   -> 0.0054s
==  CreateDepartments: migrated (0.0059s) =====================================
```

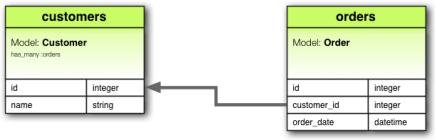


Es decir que por abajo, si miramos el archivo **log\development.log**, nos encontraríamos una charla entre ActiveRecord y el motor SQLite, de tipo CREATE TABLE, etc.

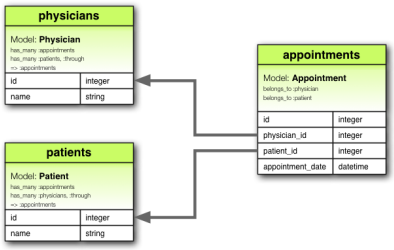
Formalizar la relación

Si bien hemos establecido la relación entre ambas tablas, mencionando el índice de una como clave ajena de otra, no hemos aclarado si es una relación de **Uno a Uno, de Uno a Muchos, o de Muchos a Muchos.**

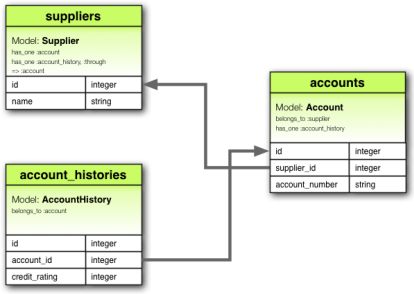
La presente explicación tiene por propósito introducir solo a los ejemplos aquí presentados. Para ver una muy completa guía sobre las relaciones entre los modelos de ActiveRecord, se recomienda enfáticamente concurrir a la excelente guía presente en http://guides.rubyonrails.org/association_basics.html




```
class Customer < ActiveRecord::Base
  has_many :orders
end
```




```
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, :through => :appointments
end

class Appointment < ActiveRecord::Base
  belongs_to :physician
  belongs_to :patient
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :physicians, :through => :appointments
end
```

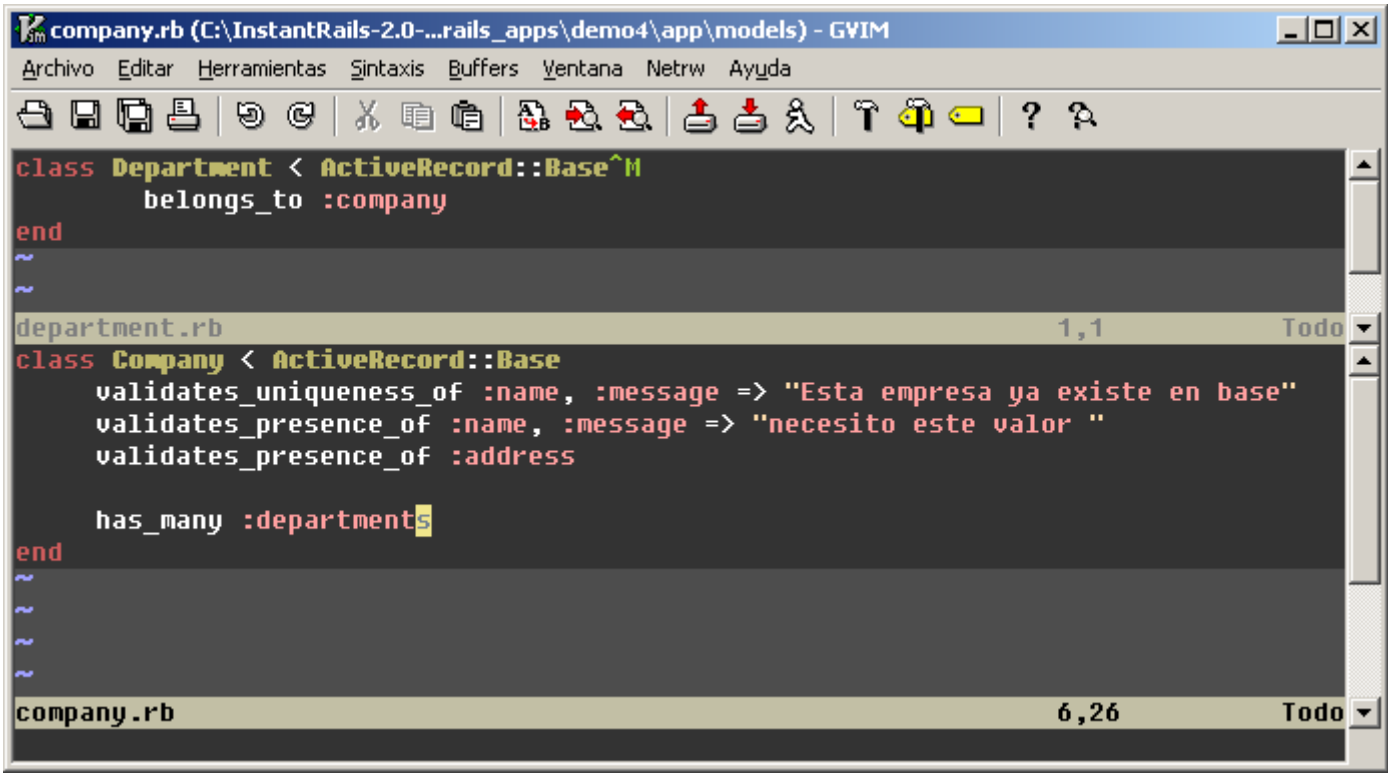



```
class Supplier < ActiveRecord::Base
  has_one :account
  has_one :account_history, :through => :account
end

class Account < ActiveRecord::Base
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ActiveRecord::Base
  belongs_to :account
end
```

En términos generales, abrimos ambos modelos, y les explicamos a Rails *que una compañía puede tener varios departamentos*. En tanto que *un departamento pertenecerá a una sola compañía*.

Si recuerdan las reglas de integridad referencial, recordarán el viejo adagio "en una relación de Uno a Muchos, la clave de los unos (Compañía) se encuentra del lado de los muchos (Departamentos). Esta es la razón por la cual un departamento posee un campo

company_id. Y lo aclaramos mediante "belongs_to".



Lo hacemos en la carpeta **app\models.** En esta captura de pantalla, gvim edita ambos modelos a la vez.

Integridad Referencial

A nivel ActiveRecord

Una pregunta que se hacen todos los programadores que llegan de otros lenguajes, es como maneja Rails la integridad referencial. Al final del vinculo mencionado, David Heinemeier Hansson menciona que desde el punto de vista de Active Record, y de mediar bien las validaciones en los modelos, no hace falta. De hecho en la guía oficial de asociaciones, http://guides.rubyonrails.org/association_basics.html, menciona algunos trucos. Por ejemplo:

```
class Customer < ActiveRecord::Base
  has_many :orders, :dependent => :destroy, :validate => :false
end
```

Aquí se puede observar que en el caso que se borre un cliente, se borran todas sus facturas asociadas. Otra opción alternativa a destroy es usar :delete_all, lo cual borra el objeto sin

ejecutar el método destroy, ignorando todas las precondiciones que este tuviera. En nuestro ejemplo sería:

```
has_many :orders, dependent: :delete_all, validate: :false
```

Y por supuesto hay una opción inversa a :destroy, llamada :restrict, la cual, si adivinan bien, inhibe el borrado disparando un feo mensaje de tipo ActiveRecord::DeleteRestrictionError. Para controlar este mensaje, lo ideal es usar alguna de sus variantes:

:restrict_with_exception – dispara una excepción, si hubieran registros asociados. Esto es mucho mejor, ya que como sabemos, las excepciones son atrapables.

```
has_many :orders, dependent: :restrict_with_exception
```

:restrict_with_error – es lo mismo, pero genera un error en lugar de una excepción. Es, en realidad, una excepción atrapada.

```
has_many :orders, dependent: :restrict_with_error
```

:restrict_with_error es muy útil, porque si sabemos con antelación que algún día el método destroy fallará (por instrucción del modelo), podemos reformular su destrucción de una manera mas elegante, preguntando (if) si pudo ser destruido, en lugar de destruirlo sin mas. Caso contrario, mandamos un mensaje:

```
def destroy
    if @order.destroy
      respond_to do |format|
        format.html { redirect_to orders_url }
        format.json { head :no_content }
      end
    else
      redirect_to orders_url, notice: "Factura #{@order.nombre}, debido a reglas de integridad, no debe ser borrado"
    end
  end
```

Un último comentario: En el caso de **has_and_belongs_to_many** hay que hacer un paso extra, ya que este helper no soporta :restrict_with_error. De hecho su comportamiento habitual es realizar un delete_all

Esto es peligroso si alguien anda borrando registros en tablas auxiliares, por ejemplo un registro de localidades. Una solución alternativa es usar has_many de ambos lados, o usar clear cuando se quiere borrar solo las combinaciones en la tabla relación.

Integridad a nivel base de datos

La integridad referencial explicada anteriormente es a nivel de Active Record. Si además se desea utilizar la integridad referencial propia de la base de datos, la documentación oficial recomienda usar
http://github.com/harukizaemon/redhillonrails/tree/master/foreign_key_migrations/

En el README de esta gema se explica como generar desde las migraciones claves ajenas, con instrucciones on delete, on update, etc.

Personalmente pude comprobar la validez de foreign_key_migrations, ya que siempre me ha inquietado compartir la base con programadores que desarrollen en otros lenguajes: la base debería protegerse a sí mismo ante registros huérfanos o duplicados.

Jugar con Tablas Combinadas en Rails Console

Estando ambas tablas creadas, es una buena ocasión para ver si el modelo responde a nuestras expectativas. También es una muy buena forma para detectar errores de diseño.

1.Creamos una Compañía y tomamos nota de su índice, creado automáticamente. Por ejemplo: 3

2.Creamos un Departamento, y en el campo company_id, fijamos el valor 3

3.Cuando queremos saber a que compañía pertenece un Departamento, en lugar de realizar un

```
SELECT companies.name

FROM companies, departments

WHERE companies.id = departments.companies_id
```

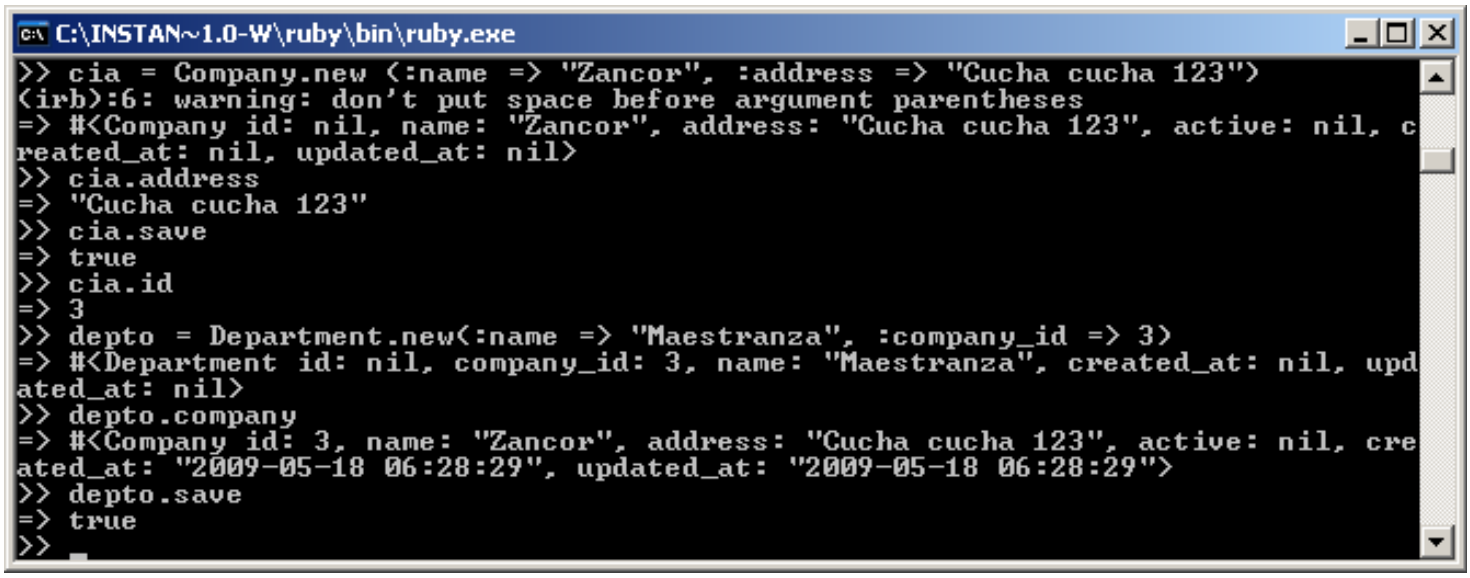
… simplemente instanciamos un objeto de tipo Departament. Una de sus propiedades es la Company a la que pertenece. A esto se le llama expresarlo en forma de ORM.

Si bien estos pasos los podemos hacer creando cada registro en el **SQLite Manager,** conviene aprender a escribirlo en formato ORM, ya que así lo expresaremos cuando programemos **controladores**. Para ello llamamos a una versión del IRB (el interactive Ruby) ya precargado con la clase ActiveRecord y sus herencias... nuestros modelos.

Esto se le conoce como la **Rails Console**.

Observar atentamente el "chateo" con el modelo. Podemos jugar a ser **controladores** antes de sentarnos a programar.

Aquí podemos ver los pasos realizados en un Rails Console.



Observar las ordenes introducidas, comenzadas con >>

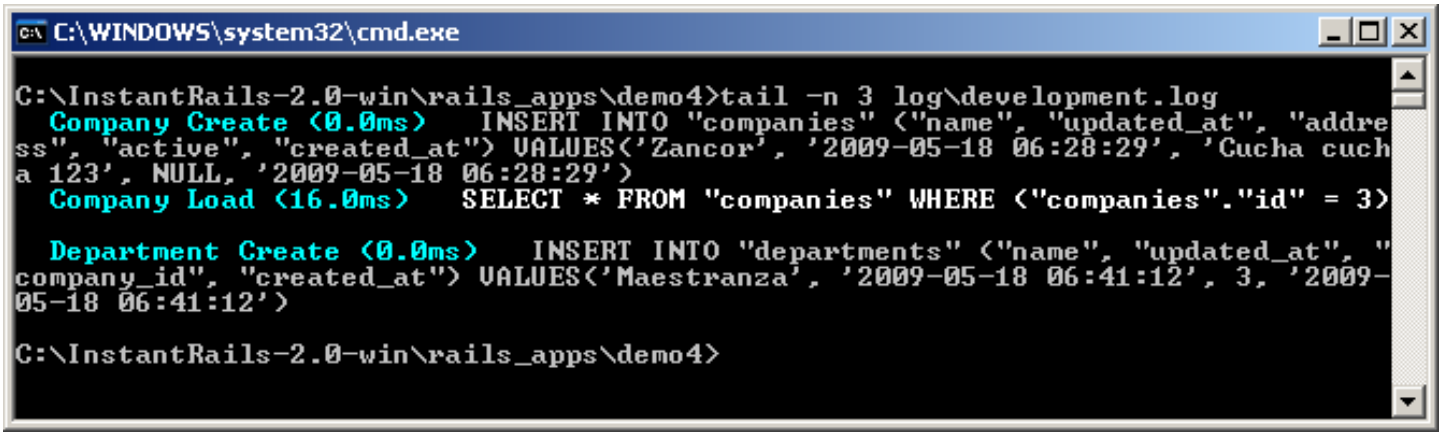
Obtenemos la Rails Console en la raíz del proyecto, mediante la orden

 **ruby bin\rails console**

 **rails console**

Querido diario...

Por abajo están ocurriendo las verdaderas transacciones SQL.



En esta captura se está utilizando **tail** mediante **cygwin** para leer la traza del sistema, desde el archivo **log/development.log**

Puede utilizarse en su lugar **TailXP**, o el mismo **notepad** de Windows, aunque este último no actualiza en tiempo real los cambios que Rails va escribiendo sobre este archivo.

Partials y Helpers

Si alguna vez han tenido piezas de código recurrente, que las han tenido que programar en forma parecida muchas veces, es posible que las hayan movido a archivos, para poder llamarlas a la siguiente vez, y para no tener que reprogramarlas una y ortra vez, con el peligro de tener versiones distintas entre sí.

Una variación de esta manera de programar es incluir alguna función o procedimiento en estos archivos, incluir el nombre del archivo al principio del nuestro, y llamar a estas funciones enviando ocasionalmente algún parámetro.

Rails maneja este concepto de dos maneras. Por un lado tiene los **helpers**, los cuales son módulos (procedimientos o funciones) que se crean en app/helpers. Los procedimientos allí guardados, una vez que se ejecutan, como todos los procedimientos, no devuelven nada, o como todas las funciones, devuelven (solo un) un valor que pasamos por return.

No obstante, a veces necesitamos que estas piezas de código hagan muchas cosas: no solo que devuelvan un valor. Y esto se nota en la vista, cuando a veces tenemos que mostrar lo mismo una y otra vez, o cuando tenemos una porción grande de código que sabemos que deberemos programarla muy parecida en otra parte. El caso clásico son los formularios new y update: ambos tienden a tener los mismos campos, la misma lógica de validaciones, etc.

Los partials son muy útiles en estos casos: como trozos de código que se insertan en nuestra pagina, y se utilizan así:

```
<%= render 'mi_partial', :locals => {:variable1 => 'Podrias procesar esto por mi', :veces => 36} %>
```

En este ejemplo, estamos insertando un archivo mi_partial.html.erb, el cual ha procesado las variables :variable y :veces

El scaffold de Rails, probablemente a propósito y a efectos pedagógicos, incluye un partial llamado _form.html.erb de ejemplo en su ABM.

En el próximo capitulo, Uno a Muchos volveremos a este tema, con un ejemplo.

¡Me olvidé de agregar unos campos! → scaffold_controller

¿Qué sucede si creamos **nuevos** campos (columnas) en la base de datos?.

Idealmente: cree o modifique todo lo que quiera. Pero hágalo mediante migraciones, a fin de mantener actualizada la espina dorsal del sistema de modelos: el archivo **db/schema.rb**

Para el ejemplo de las compañías, los pasos serían:


```
rails generate migration AddCuitToCompany cuit:string
```


```
ruby bin\rails generate migration AddCuitToCompany cuit:string
      exists  db/migrate
      create  db/migrate/20100111170149_add_cuit_to_company.rb
```

Por curiosidad, revisemos el archivo creado:

```
************************************************
class AddCuitToCompany < ActiveRecord::Migration
  def self.up
    add_column :companies, :cuit, :string
  end
  def self.down
    remove_column :companies, :cuit
  end
end
************************************************
```

Corremos la migración, para que se tomen los cambios en la estructura de la base:

```
rake db:migrate

==  AddCuitToCompany: migrating ===============================================
-- add_column(:companies, :cuit, :string)
   -> 0.0028s
==  AddCuitToCompany: migrated (0.0032s)
```

El problema

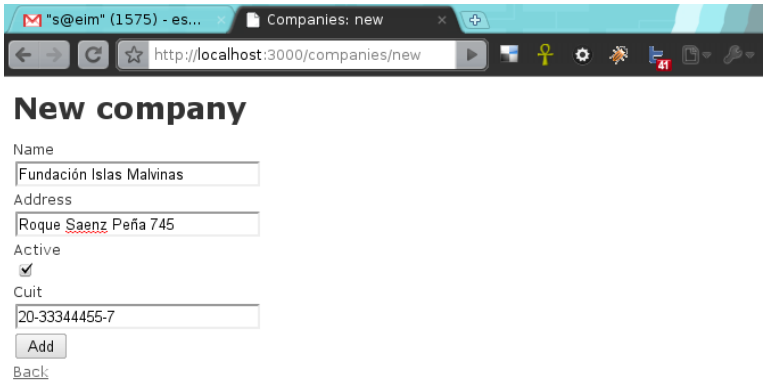
Si bien modelos y controladores están actualizados, las vistas no lo están, y por lo tanto seguirán mostrando los campos antiguos. Así, **tendremos que crear manualmente los nuevos campos**, *editando las vistas*. Si somos vagos, y queremos evitar este paso, podemos recurrir

nuevamente a **scaffold_controller** para mantener todo actualizado:

```
rails generate scaffold_controller Company name:string address:string
active:boolean cuit:string
```

```
ruby bin\rails generate scaffold_controller Company name:string address:string
active:boolean cuit:string
```

Como se puede apreciar, se le pasa como argumento los campos existentes, y el nuevo campo.



Uno a Muchos => Combo SELECT, Radio Buttons

Las cajas **selects** pueden expresarse en forma **estática**, o en forma **dinámica** (tomando datos desde la base). La forma **estática** sería equivalente a

```
<%= select "department","name",["Ventas","RRHH","etc"] %>
```

En el caso de la forma **dinámica,** necesitamos tener una relación **has_many** en el modelo, y una **variable repetida** varias veces en el controlador. La relación ya la tenemos, de modo que en el controlador **[app/controllers/departments_controller.rb]**, concretamente en las acciones acciones `edit, new, update y create.` En ellas creamos un atributo llamado `@companies`, compuesto por TODAS las compañías que le proporcione ActiveRecord.

Se lo crea con @ adelante para que las vistas también tengan acceso.

```
    @companies = Company.all
```

Recuerde realizar este paso. De otra manera, el formulario disparará el siguiente error cuando devuelva los datos a las acciones update o create:

```
You have a nil object when you didn't expect it!
You might have expected an instance of Array.
The error occurred while evaluating nil.map
```

Ahora el código del SELECT desplegable.

En el partial, **[app/views/departments/_form.html.erb]**, lo dejamos parecido a

```
<%= error_messages_for 'department' %>
<!--[form:department]-->
<div>
  <select name="department[company_id]">
   <% @companies.each do |cia| %>
       <option value="<%= cia.id %>"
         <%= 'selected' if cia.id == @department.company_id %> >
         <%= cia.name %>
       </option>
   <% end %>
  </select>
  <%= f.label :name %><br/>
  <%= f.text_field 'name'  %>
</div>
<!--[eoform:department]-->
```

Por si lo han notado, en lugar de dejar el fuente en **new.html.erb** o en **edit_html.erb**, lo

hemos dejado en un partial **_form.html.erb** común a ambos. Un partial es como un trozo de código que se inserta y se ejecuta allí donde lo llamemos. Mas adelante veremos como crear nuestros propios partials.

Mucho código? ¡la ultima moda en Rails!: un helper. Logra el <SELECT> anterior con... una sola línea. Probablemente no se lleve bien con Ajax, pero es bastante sintética.

```
<%= f.select :company_id, @companies.map {|comp| [comp.name, comp.id]} %>
```

Tercera forma de realizar la caja del SELECT (no funciona con partials):

```
  <select id="department_company_id" name="department[company_id]">
    <%= options_from_collection_for_select @companies, "id", "name",
@company.department_id %>
  </select>
```

ORM... de nuevo

Recordemos: en todo ORM (Object Relational Map), las clases mapean tablas, los atributos a los campos, los métodos a las acciones, y un objeto puede ser un registro o una colección de ellos.

Abrimos una consola irb mediante el comando

 **rails console**

 **ruby bin\rails console**

y probamos

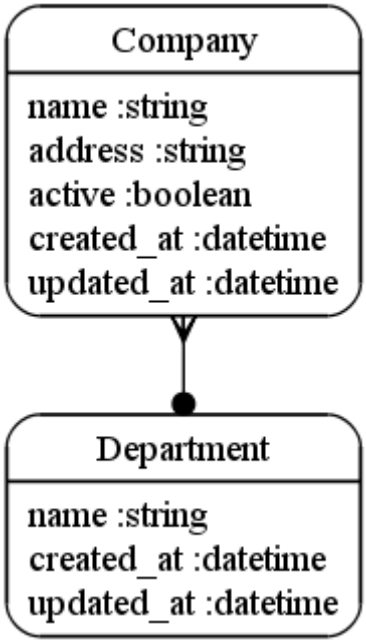
1.Crear un objeto **deptos** que represente una colección de todos los registros disponibles

2.Lo iteramos. Es decir, lo recorremos, mostrando con el comando p (puts para holgazanes).

```
deptos=Department.find(:all)
    ==> [#<Department id: 1, company_id: 1, name: "Secretaria Academica",
created_at: nil, updated_at: nil>, #<Department id: 2, company_id: 3,
name: "Maestranza", created_at: "2009-05-18 06:41:12", updated_at: "2009-
05-18 06:41:12">]


irb --> deptos.each do |i|
    \-+ p i.name
>> end
"Secretaria Academica"
"Maestranza"
deptos[1].name
    ==> "Maestranza"
```

También lo podríamos haber recorrido con un for, un do loop, etc. Pero horrorizaría a cualquier programador de Ruby.

¡Tengo muchas tablas, índices y campos!



Estrategia 1

Crear automáticamente diagramas de modelos y controladores

En una fase avanzada del proyecto, puede confundirnos la cantidad de tablas -modelos- y controladores interoperando entre sí. Algunos IDE pagos, como **Rubymine**, poseen funcionalidades muy buenas en tal sentido. Sin embargo, podemos seguir manteniéndonos "libres", y usar **Railroady** en su lugar.

 `sudo apt-get install graphviz`

 `Bajar e instalar desde` http://www.graphviz.org

Luego, incorporar a Gemfile

```
group :development, :test do
    gem 'railroady' #esta gema a veces confunde a rake en Windows
end
```

Los siguientes comandos son auto explicativos

- `bundle install`
- `mkdir doc`
- `rake diagram:all`
-  `nautilus doc/`  `start doc\`

En la carpeta doc/ deberían aparecer los diagramas. El formato es **.svg,** es decir vectorial xml, formateable con cualquier herramienta de dibujo como **Inkscape, Corel Draw, Ilustrator**, etc, e incluso ¡un editor de textos!

Puesto que Windows no abre en forma nativa el formato svg, se puede modificar el comportamiento de creación para crear png en lugar de svg (lea mas abajo). Pero en tanto que svg es fácilmente formateable mediante los programas mencionados, los archivos png requieren de toda clase de maquillaje por parte de algún programa de edición de fotos.

Podemos generar la version de los archivos png en lugar de svg, mediante los comandos

```
railroady -C | dot -Tpng > doc/controladores.png
railroady -M | dot -Tpng > doc/modelos.png
```

Si algo falla, es porque no están bien trazadas las órdenes belongs_to, has_many. Tambien el comando puede fallar si no ha sido creada la carpeta doc, o si hay migraciones sin correr. ¡Lea la salida del error!

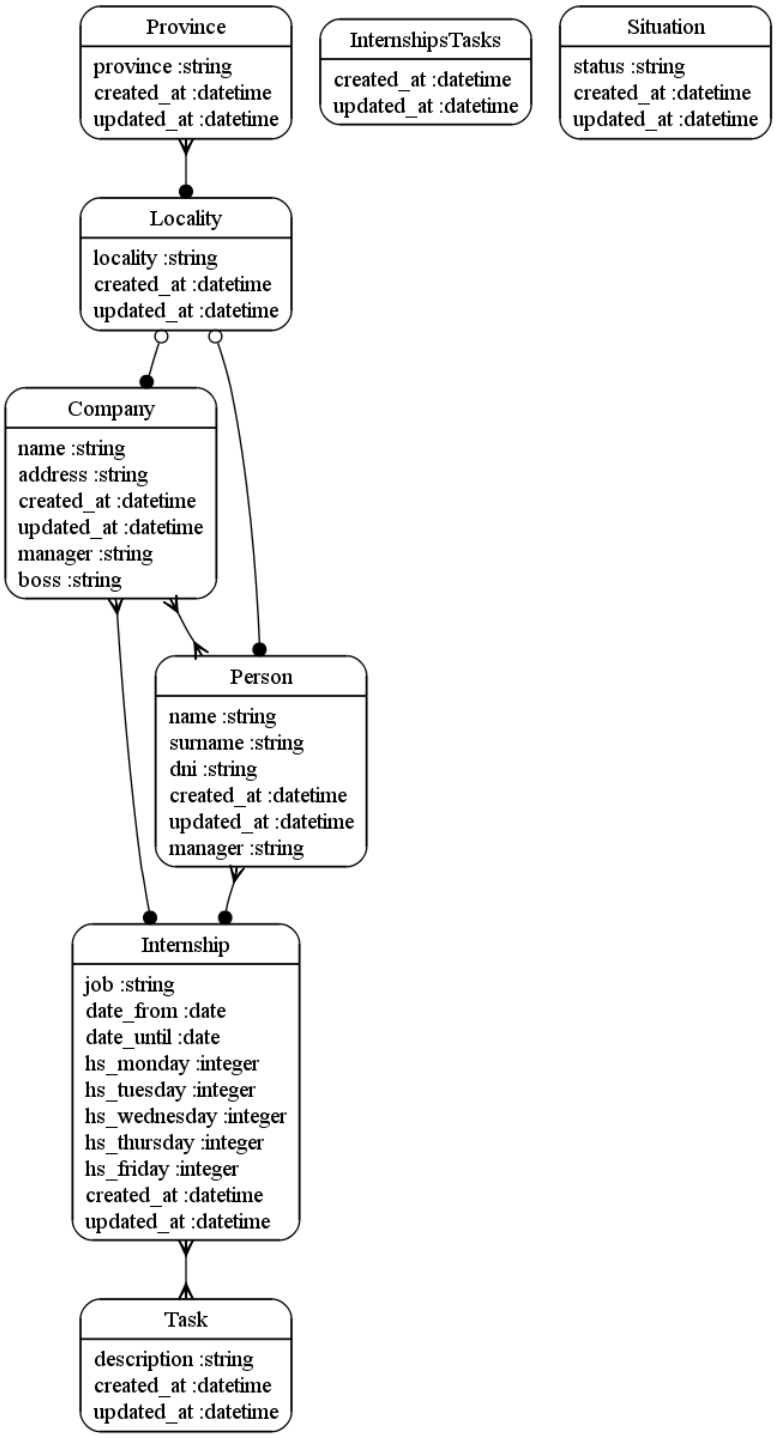
Truco: para poder visualizar los diagramas en formato **svg** en Windows:

 Windows:

- Instalar plugin SVG de Adobe para navegadores Explorer y Firefox: http://www.adobe.com/svg/viewer/install/

Linux:

- Para solamente ver los archivos creados, se puede llamar simplemente al propio viewer de imágenes de gnome (Ejemplo: **`eog modelos.svg`**).
- Para editar, agregar títulos, etiquetas, comentarios, abrir con **inkscape** (**`apt-get install inkscape`**).



Estrategia 2

Annotate Models

Cuando el proyecto avanza mucho, los creadores de rails recomiendan utilizar un plugin que simplemente va anotando dentro de los modelos, los campos que vamos utilizando. Esto se debe a que cuando va progresando el código, recurrimos frecuentemente al modelo para cambiar cosas. Y todo el tiempo vamos a la base a revisar los campos existentes. Es un ejercicio aburrido.
Ejemplo de instalación, directamente desde la página de sus creadores:

```
gem install annotate
```

```
gem install -rV annotate
```

Alternativamente: entrar a http://github.com/ctran/annotate_models
Ahora si: para dejar a mano los nombres de campos, dentro de los modelos:

```
annotate
```

```
… Annotating Company
… Annotating Department
```

En Linux, si la salida es distinta, puede ser que se encuentre presente el programa /usr/bin/annotate del paquete gd. En tal caso no hace falta instalar este paquete del sistema operativo. Solo hay que hacer:

```
rails g annotate:install
```

```
rake annotate_models
```

El efecto final de esta gema funcionando es que si ahora miramos dentro de **app\models\company.rb**... todas nuestras columnas y sus tipos estarán allí.
Muy útil por ejemplo para armar las validaciones.

```
# == Schema Information
# Table name: companies
#
#  id         :integer         not null, primary key
```

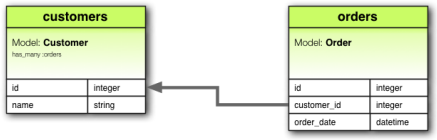
```
#  name       :string(255)
#  address    :string(255)
#  active     :boolean
#  created_at :datetime
#  updated_at :datetime

class Company < ActiveRecord::Base
     has_many :departments
end
```

Capítulo 11

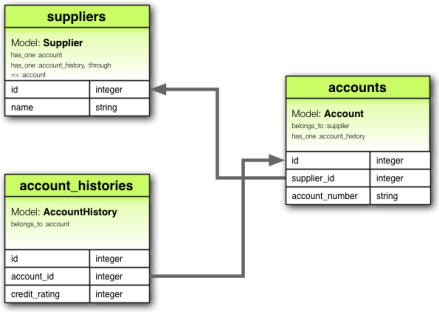
Capítulo 11: Muchos a Muchos => Checkboxs

Es un hecho que si en una relación de **uno a muchos**, la representación será una caja desplegable **SELECT**, (o unos **RADIO)**, en cambio en una relación de **muchos a muchos**, su representación serán los **CHECKBOX**.
Rails es capaz de manejar de varias formas las relaciones muchos a muchos. Aquí veremos la forma mas simple, la has_and_belongs_to_many.

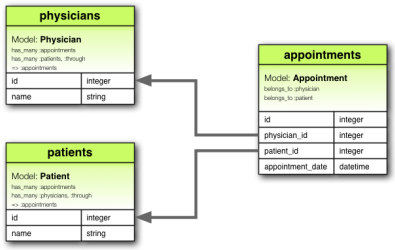
Nuevamente...
La presente explicación tiene por propósito introducir solo a los ejemplos aquí presentados. Para ver una muy completa guía sobre las relaciones entre los modelos de ActiveRecord, se recomienda enfáticamente concurrir a la excelente guía presente en
http://guides.rubyonrails.org/association_basics.html




```
class Customer < ActiveRecord::Base
  has_many :orders
end
```




```
class Supplier < ActiveRecord::Base
  has_one :account
  has_one :account_history, :through => :account
end

class Account < ActiveRecord::Base
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ActiveRecord::Base
  belongs_to :account
end
```




```
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, :through => :appointments
end

class Appointment < ActiveRecord::Base
  belongs_to :physician
  belongs_to :patient
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :physicians, :through => :appointments
end
```

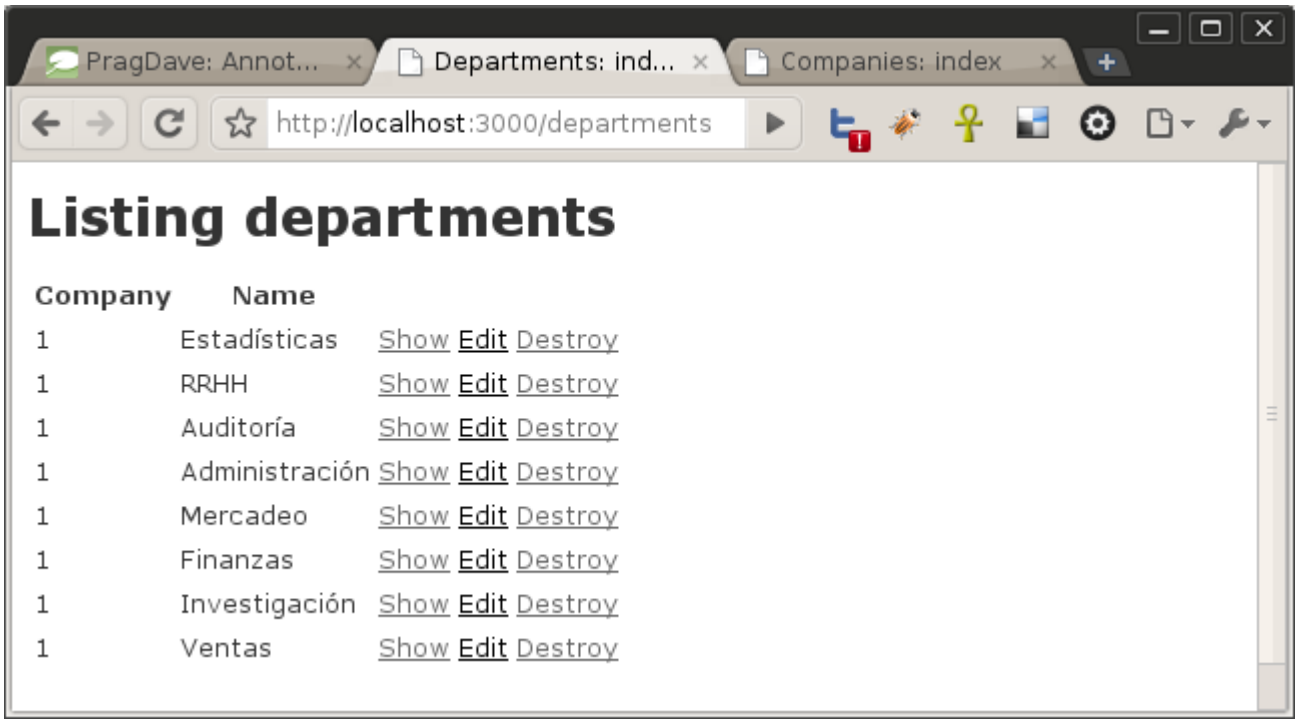
*Supongamos que en la compañía, **varios departamentos** colaboran para resolver **algunos proyectos**.*

Cargar departamentos

Este paso puede realizarlo mediante fixtures que todavía no vemos), cargar directamente usando el Manager de SQLite, o mediante el ABM de departamentos que habíamos hecho con Rails, es decir:

1. Lance el server de pruebas

   `rails server`
2. En http://localhost:3000/departments cargue los departamentos:
   - Estadísticas
   - RRHH
   - Auditoría
   - Administración
   - Mercadeo
   - Finanzas
   - Investigación
   - Ventas



Es decir, debería tener algo parecido a la captura de pantalla:

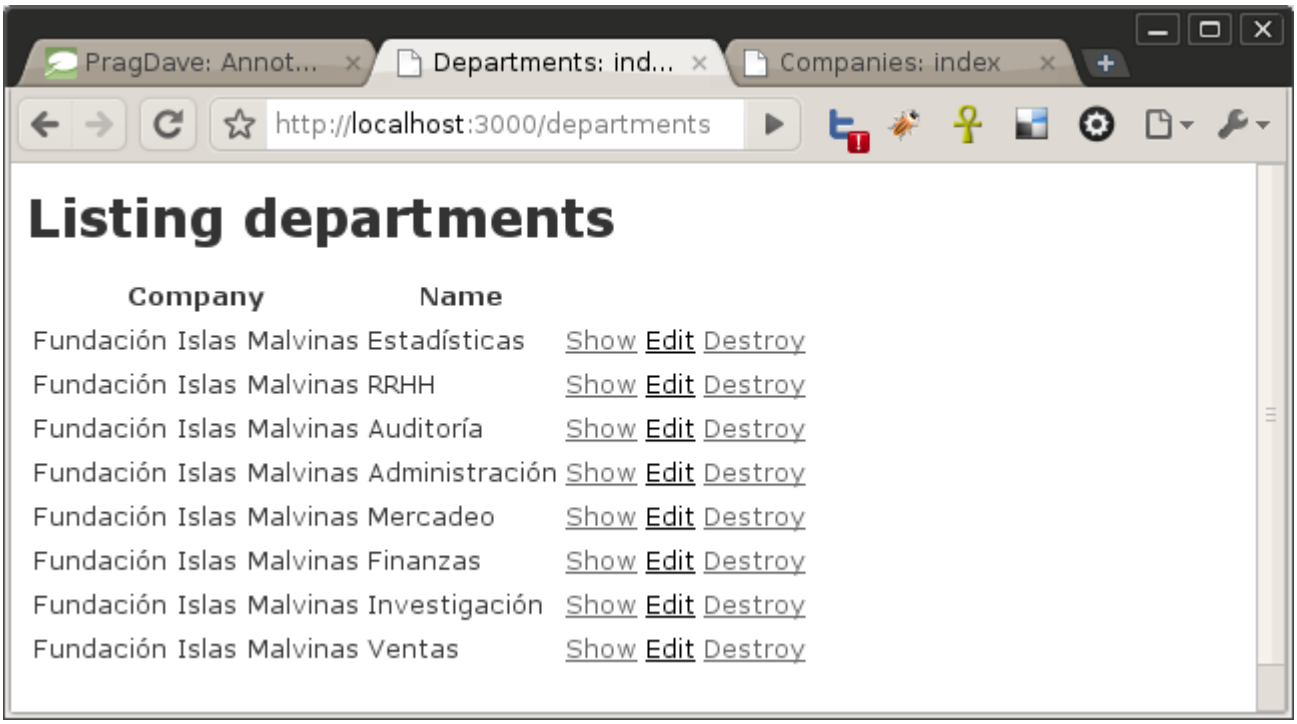
3. Upps, en la primer columna, en la que dice Company, figura el **company_id** en lugar del **nombre** de la **compañía**. Algo que con otros frameworks tardaríamos un rato en arreglar, con Rails es tan simple como

- Abrir **app/views/departments/index.html.erb**
- Cambiamos la línea que dice

```
<td><%=h department.company_id %></td>
```

por otra que diga

```
<td><%=h department.company.name %></td>
```



Creación del ABM de proyectos

Si, adivinan.

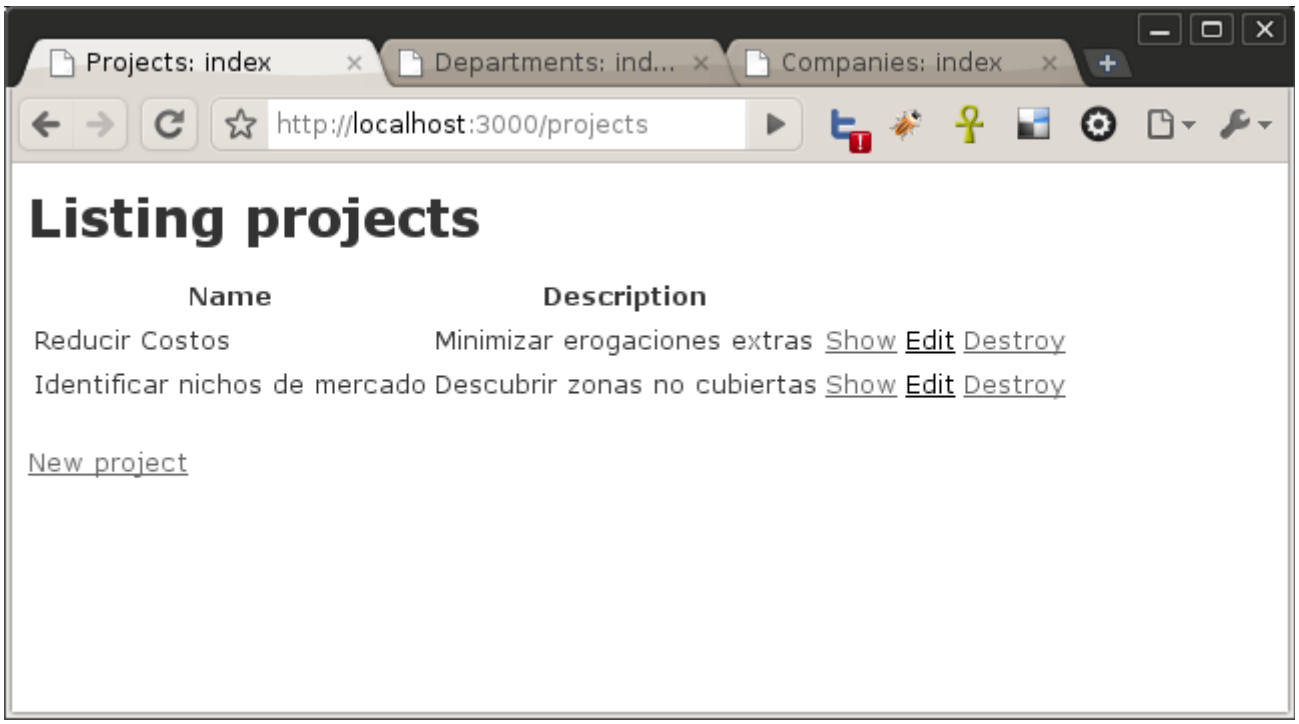
```
rails generate scaffold Project name:string description:string
```

```
ruby bin\rails generate scaffold Project name:string description:string
```

```
rake db:migrate
```

En http://localhost:3000/projects cargamos dos proyectos con los cuales jugar a:

- Reducir costos
- Identificar nichos de mercado



Crear tabla relación

Necesitamos una tabla "relación", que conserve los id de aquellas tablas con las que queremos iniciar la relación muchos a muchos:

 `rails generate scaffold departments_projects project_id:integer department_id:integer`

 `ruby bin\rails generate scaffold departments_projects project_id:integer department_id:integer`

Rails tiende a crear por nosotros un campo **id** para la propia tabla. Sin embargo, en este caso, **lo evitaremos**, ya que los índices (mas adelante) serán las claves ajenas **project_id** y **department_id**.

Por esta razón, antes de migrar, introduciremos el siguiente cambios (en rojo) en el archivo de migración **db/migrate/20100116190300_create_projects_departments.rb**

```
class CreateProjectsDepartments < ActiveRecord::Migration
   def self.up
     create_table :projects_departments, :id => false do |t|
       t.integer :project_id
```

```
      t.integer :department_id

      t.timestamps
    end
  end

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

Asimismo, quitaremos la linea que figura tachada (~~t.timestamps~~) para evitar lidiar con *constraints* de tipo *not null* en Sqlite.

Finalmente, un tercer aspecto es cuidar (solo para estos casos de muchos a muchos) usar plurales en la tabla relación: en este ejemplo, projects_departments. También se puede evitar el estilo camelCase se se desea, y es perfectamente valido.

Ahora si, corremos la migración:

  **rake db:migrate**

```
==  CreateProjectsDepartments: migrating ======================================
-- create_table(:projects_departments, {:id=>false})
   -> 0.0050s
==  CreateProjectsDepartments: migrated (0.0055s) =============================
```

Formalizamos la relación entre las tablas

En **app/models/department.rb**

```
class Department < ActiveRecord::Base
    belongs_to :company
    has_and_belongs_to_many :projects
end
```

Y en **app/models/project.rb**

```
class Project < ActiveRecord::Base
  has_and_belongs_to_many :departments
end
```

Controlar la relación

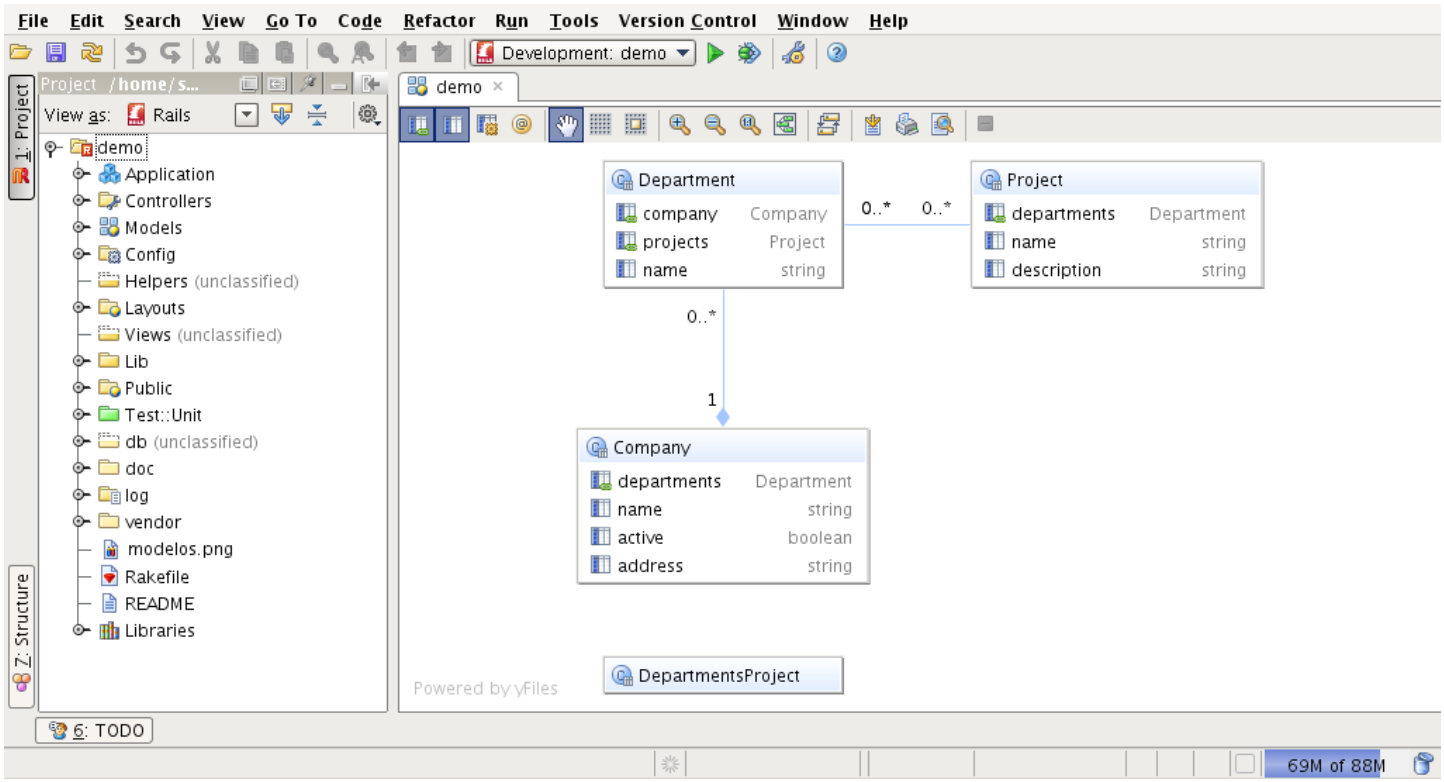
A los programadores de Ruby les gusta mucho **probar antes de programar**. Todavía no entramos a **rspec**, **cucumber** y otras herramientas para hacer algo así, pero igualmente **podemos descubrir errores en los Modelos que hemos diagramado, antes de echarle la culpa a los Controladores o a las Vistas**.

Testeo rápido mediante diagramas automáticos

Mediante el ya visto plugin **railroady,** o mediante el editor **rubymine,** realizamos un diagrama de los modelos.

Si estos programas no dibujan la relación,
o se cuelgan, nos estamos equivocando en algún paso.

Revise mayúsculas, singulares, plurales utilizados en el tutorial.


File Edit Search View Go To Code Refactor Run Tools Version Control Window Help
Development: demo
Project /home/s...
View as: Rails
demo
Application
Controllers
Models
Config
Helpers (unclassified)
Layouts
Views (unclassified)
Lib
Public
Test::Unit
db (unclassified)
doc
log
vendor
modelos.png
Rakefile
README
Libraries
1: Project
7: Structure
6: TODO
demo
Department
company Company
projects Project
name string
0..*
0..*
Project
departments Department
name string
description string
0..*
1
Company
departments Department
name string
active boolean
address string
DepartmentsProject
Powered by yFiles
69M of 88M

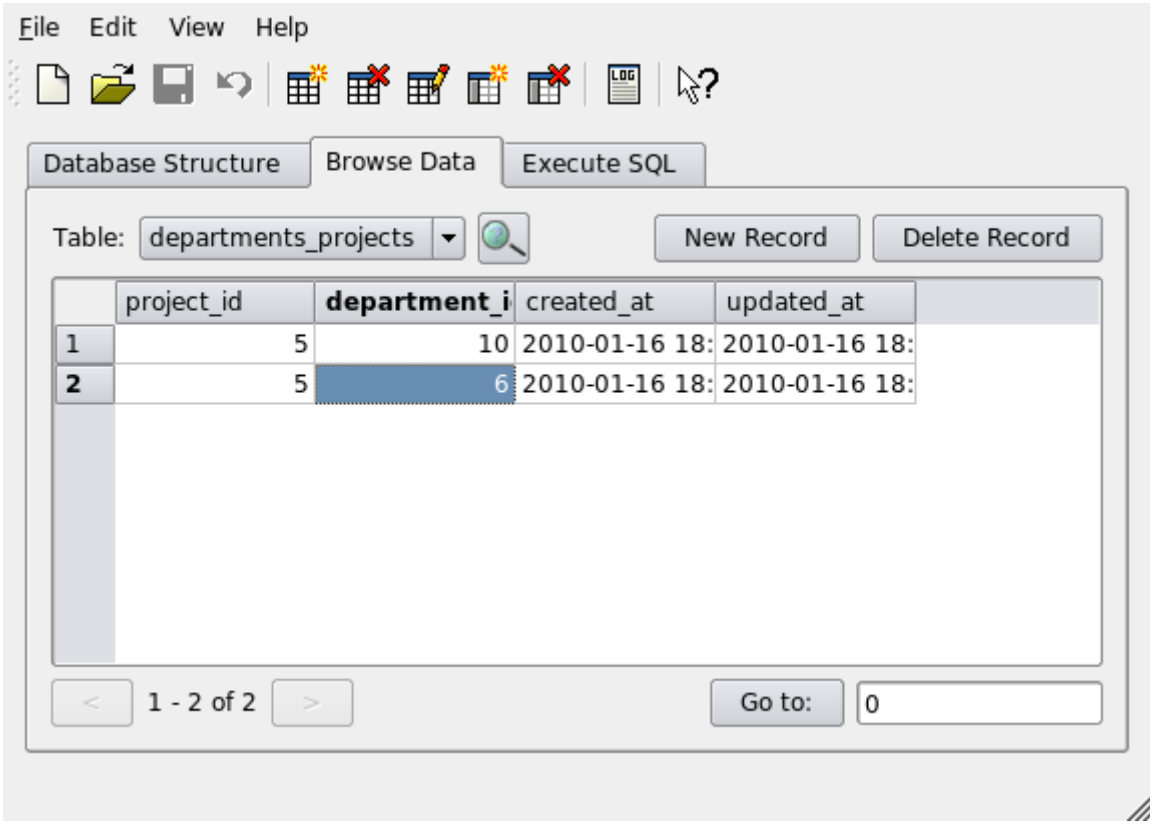
Testear relación mediante la consola de Rails + HIRB

Aviso para impacientes

A continuación se presentan cuatro hojas de comandos, que ayudarán a entender la manera en que se utilizará ActiveRecord desde los controladores. La consola de Rails, y el mini framework HIRB son una especie de polígono de tiro para debuggers, testeadores, y programadores novatos.

Si desea, puede saltear esta parte y avanzar directamente a la construcción de los CHECKBOX.

Recuerde que en las próximas hojas, a medida que va estableciendo relaciones **muchos a muchos**, puede ir controlando todo mediante alguno de los managers de SQLite recomendados en el **Módulo 1: Instalación**. Aquí se muestra **sqlitebrowser**.



En la raíz de la aplicación, lanzamos una versión de **irb** que precarga los modelos, y los hereda con respecto a la superclase **ActiveRecord**.

Ruby bin\rails console

rails console

```
Loading development environment (Rails 2.3.5)
Welcome to interactive ruby!
irb -->
```

Creando proyectos desde consola

Asignamos un proyecto para que sea realizado por determinados departamentos. Si un proyecto no existe aún, incluso lo podemos crear aquí.

```
irb --> proyectoColas = Project.new
    ==> #<Project id: nil, name: nil, description: nil, created_at: nil,
updated_at: nil>

irb --> proyectoColas.name = 'Colas de espera'
    ==> "Colas de espera"

irb --> proyectoColas.description = 'Reducir colas de espera'
    ==> "Reducir colas de espera"

irb --> proyectoColas.save
    ==> true

irb --> proyectoColas.id
    ==> 5
```

¡Muy bien! Hemos creado un proyecto mediante la consola, y sin usar SQL ni las vistas. Fabriquemos **otro proyecto**. Solo por curiosidad, realizaremos todas las instrucciones anteriores usando solo UNA línea, utilizando **.create** en lugar de **.new** + **.save**

```
irb --> proyectoObsoletos = Project.create (:name => 'Obsoletos', :description =>
'Identificar compus obsoletas para donarlas')
    ==> #<Project id: 6, name: "Obsoletos", description: "Identificar compus
obsoletas para donarlas", created_at: "2010-01-18 04:03:23", updated_at: "2010-
01-18 04:03:23">
```

Ya tenemos dos proyectos, que se suman a los otros dos que habíamos cargado desde http://localhost:3000/projects, ¿recuerdan? eran:

- **Reducir costos**
- **Identificar nichos de mercado**

Podemos constatar **cuantos** proyectos tenemos, usando un **for** (como en el **Módulo 2: Introducción**), o mediante un **iterador**. Vamos a usar el estilo Ruby, el **iterador**.
Atención a las **negritas**:

```
irb --> Project.find(:all).each do |p|
\-+       puts p.name
>>      end
Reducir Costos
Identificar nichos de mercado
Colas de espera
Obsoletos
(un montón de otras cosas)
    ==>
```

De la misma forma, podemos ver cuantos departamentos figuran en base

```
irb --> Department.find(:all).each do |d|
\-+       puts d.name
>>      end
Estadísticas
RRHH
Auditoría
Administración
Mercadeo
Finanzas
Investigación
Ventas
(un montón de otras cosas)
    ==>
```

Buscando cosas en consola

Ejercicio: en el proyecto "**Colas de espera**", cuyo objeto instanciado es **proyectoColas**, se

deberían involucrar todos aquellos departamentos donde se producen largas esperas de personas. Estos son los departamentos **Ventas** y **Administración**.
Instanciemos como un objeto llamado **ventas**, que referencie aquel registro que posea el departamento **Ventas**. Si conocemos su **id**, podemos usar:

```
irb --> ventas = Department.find(10)
    ==> #<Department id: 10, company_id: 1, name: "Ventas ", created_at: "2010-
01-16 18:22:18", updated_at: "2010-01-16 18:22:18">
```

Si no conocemos su **id**, podemos buscarlo como cadena:

```
irb --> ventas = Department.find(:all, :conditions => " name like '%Ventas%' ")
    ==> [#<Department id: 10, company_id: 1, name: "Ventas ", created_at: "2010-
01-16 18:22:18", updated_at: "2010-01-16 18:22:18">]
```

Asignemos a la gente de **Ventas**, al proyecto **Colas**:

```
irb --> proyectoColas.departments << ventas
    ==> [#<Department id: 10, company_id: 1, name: "Ventas ", created_at: "2010-
01-16 18:22:18", updated_at: "2010-01-16 18:22:18">]
```

Busquemos **Administración**, y asignemos también a **Colas**:

```
irb --> adm = Department.find(:all, :conditions => " name like '%Administrac%' ")
    ==> [#<Department id: 6, company_id: 1, name: "Administración ", created_at:
"2010-01-16 18:20:53", updated_at: "2010-01-16 18:20:53">]

irb --> proyectoColas.departments << adm
    ==> [#<Department id: 10, company_id: 1, name: "Ventas ", created_at: "2010-
01-16 18:22:18", updated_at: "2010-01-16 18:22:18">, #<Department id: 6,
company_id: 1, name: "Administración ", created_at: "2010-01-16 18:20:53",
updated_at: "2010-01-16 18:20:53">]
```

Veamos cuantos departamentos se encuentran asignados al proyecto Colas.

```
irb --> proyectoColas.departments.count
    ==> 2
```

¿Cuáles?

```
irb --> proyectoColas.departments
    ==> [#<Department id: 10, company_id: 1, name: "Ventas ", created_at: "2010-
01-16 18:22:18", updated_at: "2010-01-16 18:22:18">, #<Department id: 6,
company_id: 1, name: "Administración ", created_at: "2010-01-16 18:20:53",
updated_at: "2010-01-16 18:20:53">]
```

O, mejor presentado:

```
irb --> proyectoColas.departments.each do |x|
\-+       puts x.name
>>      end
Ventas
Administración
```

(y un montón de basura)

HIRB

Realmente la salida reciente

```
irb → proyectoColas.departments
    ==> [#<Department id: 10, company_id: 1, name: "Ventas ", created_at: "2010-
01-16 18:22:18", updated_at: "2010-01-16 18:22:18">, #<Department id: 6,
company_id: 1, name: "Administración ", created_at: "2010-01-16 18:20:53",
updated_at: "2010-01-16 18:20:53">]
```

… se ve bastante desordenada. En registros con muchos campos cuesta encontrar las cosas. Mejoraremos un poco la presentación mediante un plugin llamado HIRB.

Bajo Ubuntu y Rails 4, instalamos el plugin mediante:

```
gem install hirb
```

```
gem install hirb
```

Si además queremos que funcione en la Rails console, la agregamos al Gemfile:

```
group :development, :test do
  gem 'hirb'
end
```

Como siempre después que agregamos algo a este archivo, ejecutamos **bundle install**

Luego agregamos las siguientes líneas a nuestro archivo irbrc. Si no existe, lo creamos.

 `~/.irbrc`

 `%HOME%\.irbrc`

Estas cargarán hirb en cada llamada a irb, y consecuentemente a rails console:

```
require 'rubygems'
require 'hirb'
require 'active_record'
Hirb.enable
ActiveRecord::Base.logger = Logger.new(STDOUT)
```

Prueben la siguiente combinación:

 **`rails console`**

 **`ruby bin\console`**

```
irb(main):001:0> table ['pepe', 'pipa']
+-------+
| value |
+-------+
| pepe  |
| pipa  |
+-------+
2 rows in set
=> true
```

Si tenemos problemas de instalación bajo Windows, alternativamente podemos seguir las instrucciones "oficiales" que figuran en la documentación del proyecto, y utilizar el instalador de plugins propio de Rails 4, el cual permite llegar hasta el mismo corazón del proyecto, y obtener la ultima versión posible. Este paso requiere de tener GIT instalado.

 `ruby bin\rails plugin install git://github.com/cldwalker/hirb.git`

Todo debería estar funcionando. Excepcionalmente, podría hacer falta activar la gema manualmente, es decir:

```
rails console
irb(main):001:0> require 'hirb'
```

```
irb(main):002:0> extend Hirb::Console
irb(main):003:0> Hirb::View.enable
```

Otra manera de ordenar la desordenada salida de variables es instalando en el archivo Gemfile una línea

```
gem 'awesome_print'
```

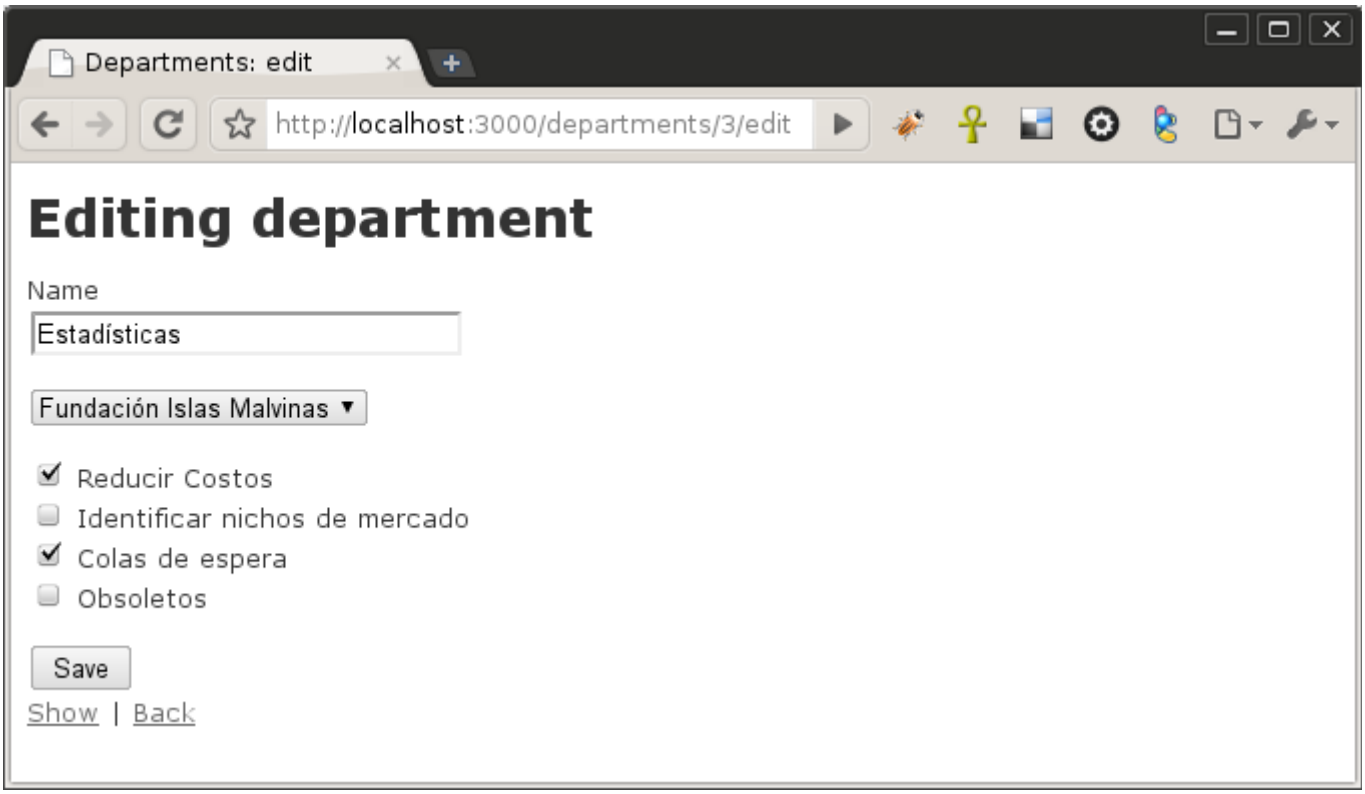
Esta gema tabula automáticamente la salida de información, e introduce la orden "ap" que reemplaza mucho mejor a p (puts).

Ejercicios propuestos

1) *Asignar al proyecto "**Reducir costos**", los departamentos de **Estadísticas, RRHH, Auditoría, Administración, Mercadeo y Finanzas**.*
2) *Asignar al revés: Inscribir al departamento **Investigación** en los proyectos "**Identificar nichos de mercado**", y "**Obsoletos**"*

Ahora sí: los CHECKBOX

Vamos a tratar que en la acción **Edit** de los **Departamentos**, cuya vista asociada se encuentra en el parcial **app/views/departments/_form.html.erb**, muestre una cajas de selección como las que se puede apreciar en la siguiente captura:



El código del parcial es el siguiente:

```
<%= error_messages_for 'department' %>
<!--[form:department]-->
<div>
  <p>
    <%= f.label :name %><br/>
    <%= f.text_field 'name'  %>
```

```
  </p><p>
    <%= f.select :company_id, @companies.map {|cat| [cat.name, cat.id]} %>
  </p><p>
  <% for project in Project.find(:all) %>
        <div>
          <%= check_box_tag "department[project_ids][]", project.id,
@department.projects.include?(project) %>
          <%= project.name %>
        </div>
    <% end %>
  </p>
</div>
<!--[eoform:department]-->
```

Cambios en el controlador

En app/controllers/departments_controller.rb, al principio de la acción **update** (la que recibe los valores de la vista **edit**), para evitar el problema citado en http://asciicasts.com/episodes/17-habtm-checkboxes respecto de cuando ningún checkbox ha sido clickeado, hay que poner lo siguiente:

```
  def update
     params[:department][:project_ids] ||= []
     @department = Department.find(params[:id])
```

En tanto que aquellos que estén probando este ejemplo sobre Rails 4 (rails -v muestra la versión), deben dejar como primera linea

```
class Department < ActiveRecord::Base
  attr_accessible :company_id, :name, :project_ids
  belongs_to :company
  has_and_belongs_to_many :projects
end
```

El error que aparece, de no cumplir ese paso, es el *ActiveModel::MassAssignmentSecurity::Error in DepartmentsController#update*

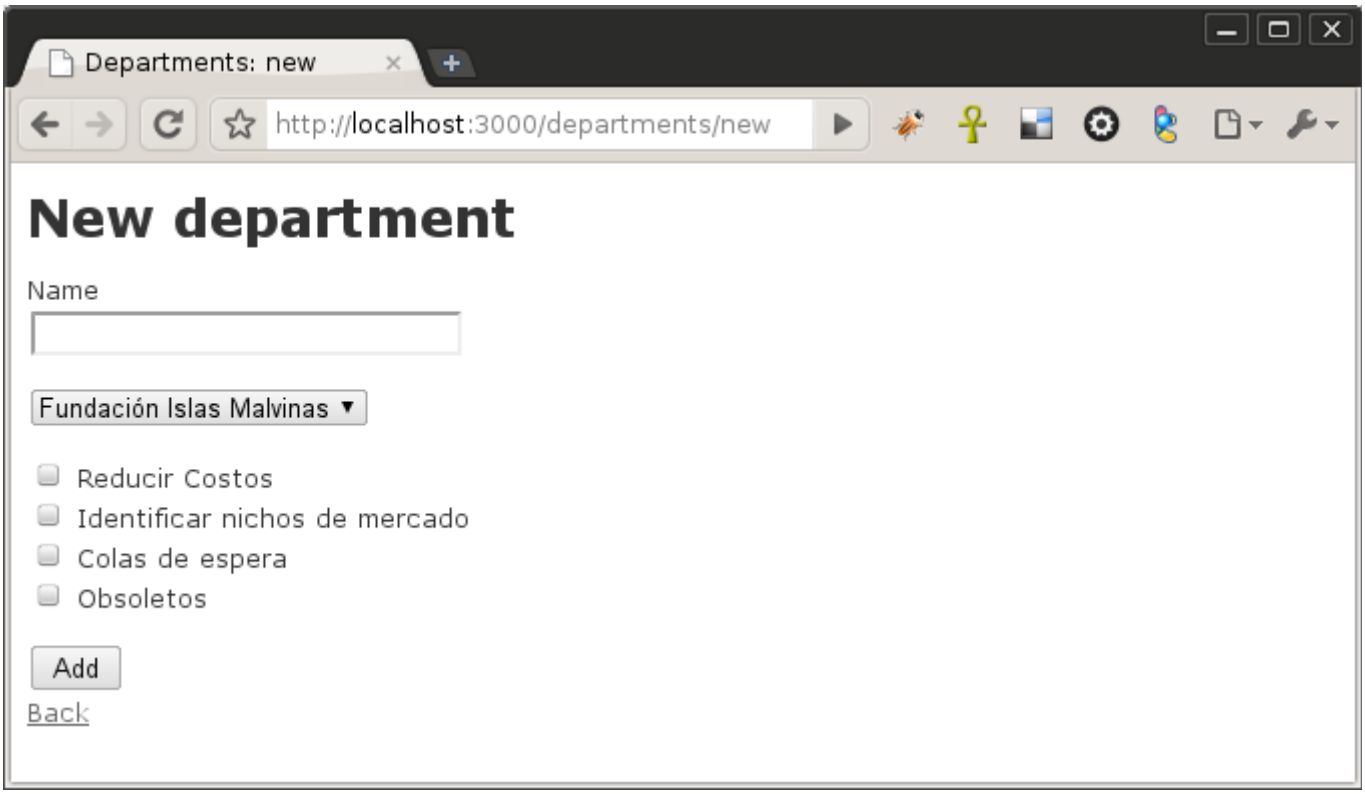
Atención en Rails 4, estos parámetros se administran desde el controlador. Ejemplo:

```
def request_params
```

```
  params.require(:department).permit(:company_id, :name, :project_ids => [])
end
```

Algo mas respecto de las vistas:

Lo interesante de haber realizado los cambios en el **parcial** **app/views/departments/_form.html.erb** en lugar de la vista **app/views/departments/edit.html.erb**, es que ahora también la vista **new** se beneficia con los checkbox:

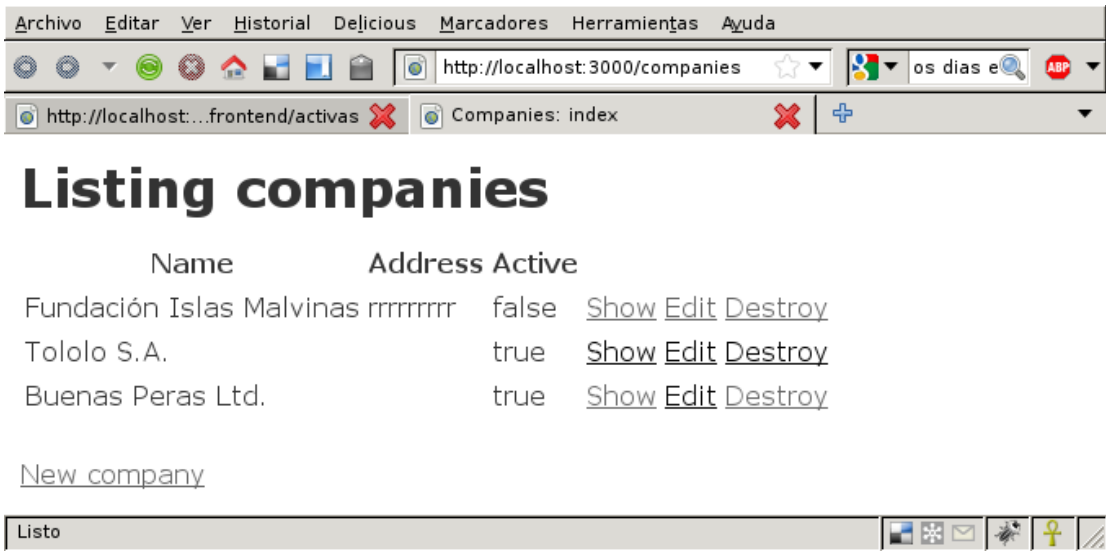


¿Y el show? (¡yapa!)

Las vistas show no poseen checkbox, lo cual es lógico, puesto que no se enviarán datos. Pero sería útil poder visualizar en modo "solo lectura" aquellos proyectos en los que se encuentra anotado el departamento. Para ello modificamos **app/views/departments/show.html.erb** para que quede mejor presentado:

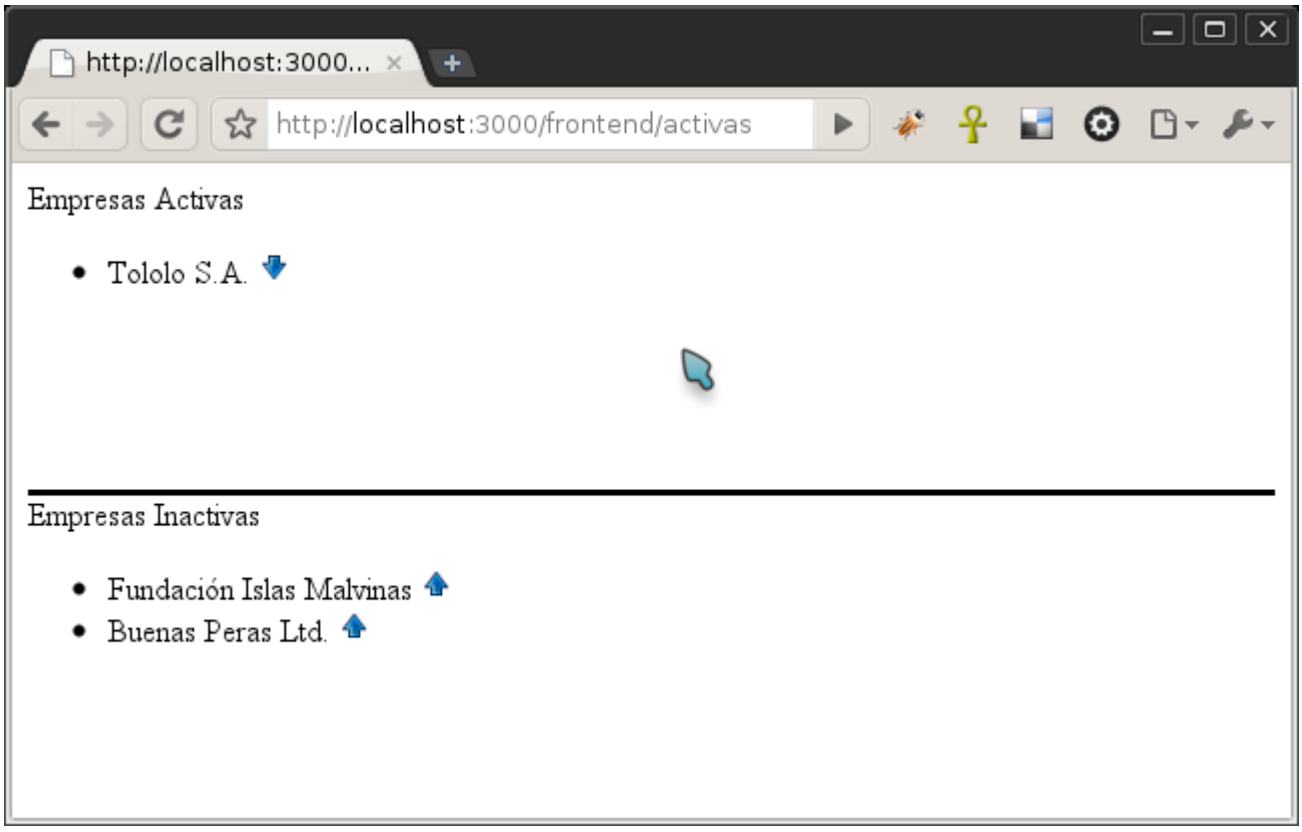

```
<p>
  <b>Company:</b>
  <%=h @department.company.name %>
</p>
<p>
  <b>Department:</b>
  <%=h @department.name %>
</p>
<p>
  Proyectos relacionados
</p>
<ul>
  <% @department.projects.each do |project| %>
     <li><%= project.name %> - <%= project.description %></li>
  <% end %>
</ul>

<%= link_to 'Edit', edit_department_path(@department) %> |
<%= link_to 'Back', departments_path %>
```

Jugando con MVC



Al principio habíamos hablado un poco de MVC. Sin embargo, no es un concepto fácil de abstraer. Ademas, los scaffold tienden a mal acostumbrar, de modo que vamos a trabajar un poco "a mano".

Es común que las las aplicaciones se dividan en el frente o "**frontend",** y en la trastienda o "**backend**". Es un anglicismo que se explica por si solo, y útil para ejemplificar aquí.



Comenzaremos generando un controlador "frontend" con una acción "activas" adentro. Su función será mostrar aquellos solo registros de compañías que tengan su campo **active** como *true*.

Es decir, haremos nuestra propia versión del ABM de companies:

Como aparece en la captura, intentaremos separar visualmente las empresas según su campo active. También le habilitaremos un botón al usuario para que cambie fácilmente el estado, con un solo click. Lo realizaremos **sin usar scaffolds**[17]:

Usaremos un generador de controladores, cuyo uso es

```
rails generate controller controlador acción1 acción2 acción3 …
```

```
ruby bin\rails generate controller controlador acción1 acción2 acción3 …
```

Por lo tanto, utilizamos la orden

```
rails generate controller frontend activas
```

```
ruby bin\rails generate controller frontend activas
```

```
      exists  app/controllers/
      exists  app/helpers/
      exists  app/views/frontend
      exists  test/functional/
      exists  test/unit/helpers/
   create  app/controllers/frontend_controller.rb
   create  test/functional/frontend_controller_test.rb
   create  app/helpers/frontend_helper.rb
   create  test/unit/helpers/frontend_helper_test.rb
   create app/views/frontend/activas.html.erb
```

Si observamos atentamente, paralela a la acción se ha creado una vista con el mismo nombre.

Mapear Rutas y Controladores

Los controladores no son públicos. Para eso existe la carpeta **public**/

Si deseamos acceder directamente al controlador vía

17 ¡Mira mama! ¡sin manos!

http://localhost:3000/frontend/activas, debemos mencionarlo en **config/routes.rb**

```
map.resources :activas
```

La Vista

Realizaremos dos divisiones ** ** que ocupen cada una la mitad de la pantalla. Esto es muy simple, y lo realizamos mediante la inclusión de css dentro del . Al ser tan localizada la necesidad, puede perdonarme no mandar **height: 50%** al archivo general .**css**, y emplearlo directamente usando el modificador **style.**

También le pondremos una simple línea inferior a la caja superior para separar ambas. Esto se logra con **border-bottom:solid;**

En lugar de los enlaces, usaremos dos botones: up.png  y down.png 

En cada uno de los ** ** usaremos bucles muy parecidos:

```
<% Company.find(:all, :conditions => {:active => true}).each do |c| %>
```

y

```
<% Company.find(:all, :conditions => {:active => false}).each do |c| %>
```

Si recuerdan el Módulo 2: Introducción, recordarán que los programadores de Ruby gustan de usar iteradores. En el ejemplo iteraremos -crearemos y mataremos muchas veces- un objeto **c**, que posee todos los datos del registro.

Dentro de los bucles, usaremos la función **link_to**, la cual puede emplear toda clase de parámetros. Para nuestras necesidades le suministraremos:

1. Algo para mostrar y convertir en un enlace: una palabra, o una imagen. Las imágenes pueden vincularse mediante otra función, image_tag.
2. Datos para conformar el enlace: en él debemos especificar
    1. Controlador hacia donde van los datos
    2. Acción a gestionar el dato
    3. El ID del registro a mostrar
3. Confirmación

Todos estos parámetros pueden anidarse usando llaves { }

Ahora sí: copiemos el siguiente código dentro de app/views/frontend/activas.html.erb

```
<div id="arriba" style="border-bottom:solid; 1px; height: 50%">
  Empresas Activas
  <ul>
    <% Company.find(:all, :conditions => {:active => true}).each do |c| %>
        <li>
          <%= h c.name %>
          <%= link_to(image_tag("/images/down.png" ),
                      { :controller => "frontend",
                        :action => "activas",
                        :id => c
                      }, {
                          :confirm => "Está seguro?",
                          }
          )
          %>
        </li>
    <% end %>
  </ul>
</div>


<div id="abajo" style="height: 50%">
  Empresas Inactivas
  <ul>
   <% Company.find(:all, :conditions => {:active => false}).each do |c| %>
        <li>
          <%= h c.name %>
          <%= link_to(image_tag("/images/up.png" ),
                      { :controller => "frontend",
                        :action => "activas",
                        :id => c
                      }, {
                          :confirm => "Está seguro?",
                          }
          )
          %>
        </li>
    <% end %>
```

```
  </ul>
</div>
```

El Controlador

El archivo app/controllers/frontend_controller.rb contendrá:

```
class FrontendController < ApplicationController
  def activas
    if params[:id]
      modificar = Company.find(params[:id])

      if modificar.active == true
        @modificar.active = false
      else
        modificar.active = true
      end

      modificar.save
    end
  end
end
```

En este controlador recibimos el id (¿recuerdan al iterador **c**?). Partimos de la siguiente idea: si acaban de arribar datos desde la vista (ver **if params(:id)**), es porque el operador ha

hecho click en algún enlace. En este caso, en  o en  , y por lo tanto, desea el

estado de campo *true* por *false*, o *false* por *true*.
Primero buscamos un registro con ese **:id**, y lo instanciamos en un objeto **modificar**

```
modificar = Company.find(params[:id])
```

Luego modificamos en memoria su campo

```
modificar.active = false o modificar.active = true
```

y luego lo guardamos.

```
modificar.save
```

La vista se dispara automáticamente cuando la acción termina su ejecución. Mostrando como la empresa cambia entre cajas

Capítulo 12

Server de Producción

Si se observa en **config/databases.yml**, se verán unas líneas que corresponden a la base de datos que *efectivamente ven los usuarios.*
En este archivo, se pueden apreciar tres entornos de trabajo: **development**, **testing** y **production**. Cada una de estas instancias se corresponden de su propia base, las cuales pueden incluso residir en distintos motores.
De esta manera, y a título de ejemplo, podemos realizar nuestras pruebas y desarrollos en el liviano SQLite; cuando consideramos estables los cambios, los aplicamos en la base producción, donde reside un MySQL, Oracle, MSSQL, PostgreSQL, etc.
En términos generales, el ciclo de vida de un desarrollador Rails *junior* que típicamente tiene su entorno en Windows, es el siguiente.

1) Trabaja en modo development, usando el liviano SQLite como base de datos.
2) Cuando considera que la aplicación se encuentra estable, hace una copia de la misma a una carpeta distinta.
3) Sobre la segunda carpeta creada a tal propósito (si usa SQLite incluso para producción), corre las migraciones en modo producción. Es decir, realiza un `rake db:migrate RAILS_ENV="production"`
4) Configura Apache para que sirva la versión en producción desde esta última carpeta. Es decir, publica.
5) Configura su Windows para que resuelva el nombre de la aplicación como local.
6) Deja instalados Apache y (Mongrel+Aplicación Demo) como *servicios* de Windows. Esto es muy útil, puesto que si desea liberar ciclos de CPU para correr otra cosa (a mi megustan los juegos de Rol), puede detener estos servicios.
   Veremos mas adelante como realizar este paso, es decir, configurar el equipo al estilo de un server Unix.

En tanto que un desarrollador Senior probablemente involucre un servidor Linux y realice los siguientes pasos:

1) Establece un repositorio propio, normalmente GIT o SVN.
2) Trabaja en modo development, también con SQLite

3) Testea:
    1. Se asegura de tener poblado un archivo seeds.rb (conocido como "fixtures") con datos básicos para tablas auxiliares, típicamente localidades, provincias, y al menos un usuario administrador que pueda dar de alta otros usuarios.
    2. Corre un comando que se encarga de limpiar la base testing, y la reconstruye (vacía) basada en la base development.
    3. Otro script carga *todos* los formularios utilizando los datos de los fixtures, a la vez que prueba el código.
       El testeo es muy útil para las ocasiones en que cambios en alguna parte de la estructura "ofende" a otros formularios cuya estabilidad damos por descontada. Esto permite superar el engorro de probar toda la aplicación cada vez. No obstante, no es obligatorio testear en Rails.
4) Si pasa los errores, es decir, los fixtures se cargan exitosamente en la base testing, informa mediante commits al repositorio local. Este repositorio le permite mantener varias ramas de desarrollo, e incluso realizar rollbacks entre ellas.
5) Invita a cliente a que pruebe los cambios (:3000)
6) En el server de producción, pide que se configure GIT, Apache y Passenger. Es decir, deja al BOFH a cargo rascándose la cabeza[18], y lo hace él mismo. Por suerte, la mayoría de los hostings con CPANEL actualizado ya traen configurados estos *daemons.*
7) Utiliza algún script automático como Capistrano, que se encarga de
    1. Subir la rama de desarrollo vía GIT o SVN.
    2. Clonar (dentro del server de producción) la nueva versión de la aplicación, en una carpeta configurada para ser servida por Apache.
    3. Correr la migración sobre la base de producción.

Hasta ahora hemos trabajado en el puerto 3000, con un pequeño servidor de pruebas llamado Webrick (o Mongrel, si se encuentra presente). Pueden adivinar la razón: hemos dejado libre el puerto 80 para la instancia **production** de nuestro sistema. A fines didácticos, enfocaremos las dos formas clásicas: proxy y módulo.

- En la primer forma, y para nuestro ejemplo con **Windows**, el puerto 80 lo gestionará **Apache**, quien es capaz de gestionar miles de consultas en simultaneo, y a la vez parsearlas a las distintas aplicaciones Rails que convivan en el mismo equipo.

18 Bastard Operator from Hell.

En estas aplicaciones, se encontrará escuchando un pequeño servidor llamado **Mongrel.** Mediante un mecanismo de proxy, Apache realizará las consultas a **Mongrel**, y se guardará aquellas que sean frecuentes.
Esto supone lidiar con algunas configuraciones extras, pero nos beneficiaremos de las ventajas de velocidad que supone intercalar un sistema de cache de consultas frecuentes.

- Bajo **Linux**, realizaremos la instalación mediante un mod-rails llamado **Phusion Passenger**, el cual desgraciadamente no existe para **Windows.** Es un procedimiento simple, acorde a los cómodos y confiables módulos que los viejos BOFH acostumbran a enchufar en Apache[19].

Discusión: ¿Linux, Windows, o un Cluster?

Se dice que "en Linux es difícil hacer muchas cosas". Sin embargo, esto no es del todo cierto. Linux parece haber sido construido para hacer fácilmente muchas cosas que en Windows son difíciles. Estas son son actividades que los usuarios finales no hacen a menudo, y por lo tanto no aprecian.
Me explico, instalar proxys, bases de datos, daemons de versionado, agentes de acceso remoto, servidores web, de backup o de correo, son asuntos triviales, rápidos y gratis bajo el sistema operativo del pingüinito.
Por esta razón, se tiende a trabajar con las bases **development** y **testing** bajo Windows (o MAC OS/X), y a desplegar la base **production** sobre Linux.
Lo mismo sucede en el mundo de PHP. Usualmente los programadores de PHP prueban en Windows, y deployan en Linux remotos, contratados mediante planes de hosting.
Así, muchos rubystas consideran que instalar un server en producción de Rails en Windows es muy difícil, a la vez que inseguro, por la naturaleza misma de Windows. Intentaremos aquí demostrar lo contrario[20].
Dejando claro los primeros conceptos sobre publicación, animaré a nuestros lectores palmípedos a que prueben la rapidez y sencillez con la que se configura un servicio equivalente sobre Linux.
Finalmente, si no deseamos complicarnos demasiado, o predecimos una alta carga sobre el sistema, veremos como desplegar hacia un cloud. Esta forma de trabajar es muy cómoda,

19 En esencia, los administradores de Linux, son (somos) unos vagos. La razón de aprender a administrar Linux... es tener tiempo libre dentro de la empresa.

20 Demostraremos que no es difícil montar un server de producción en Windows. *No que sea seguro.*

ya que se paga solamente lo consumido, y a la vez contamos con algunas comodidades muy interesantes.

Server de Producción en Windows

Para el siguiente ejemplo utilizaremos **InstantRails** o **Rubystack**, el cual ya viene con Apache configurado para proxear solicitudes a Mongrel.

Preparar Base

1) Como primer paso, copiamos la aplicación hacia otra carpeta que debe estar destinada a producción, por ejemplo "demo-production"

A tener en cuenta: si se encuentra usando SQLite, y en la carpeta destino existe un archivo **db\production.sqlite3**, no lo sobrescriba.

2) En la nueva carpeta, corremos un comando que nos permite crear la base de producción si esta no existiera, o realizarle cambios en su estructura. Estos cambios se corresponden a aquellos que hemos realizado sobre la base development.

Aviso: a menos que realicen permanentes test sobre las migraciones, **haga un backup de la base de producción**. En el caso de SQLite, es tan simple como poner a buen recaudo el archivo **db\production.sqlite3**

A tal fin, realice un

```
rake db:migrate RAILS_ENV="production"
```

Este comando realizará las siguientes acciones

- Si la base production no existe, rake creara una versión vacía, basada en la estructura de la base development.
- Si la base existe, modificará su estructura, aplicando las migraciones pertinentes.

Instalar Mongrel como servicio

Instalaremos a Mongrel como servicio de Windows. Cuidado al copiar / pegar:

```
mongrel_rails service::install -N demo -c c:\InstantRails-2.0-
win\rails_apps\demo-production -p 4000 -e production
```

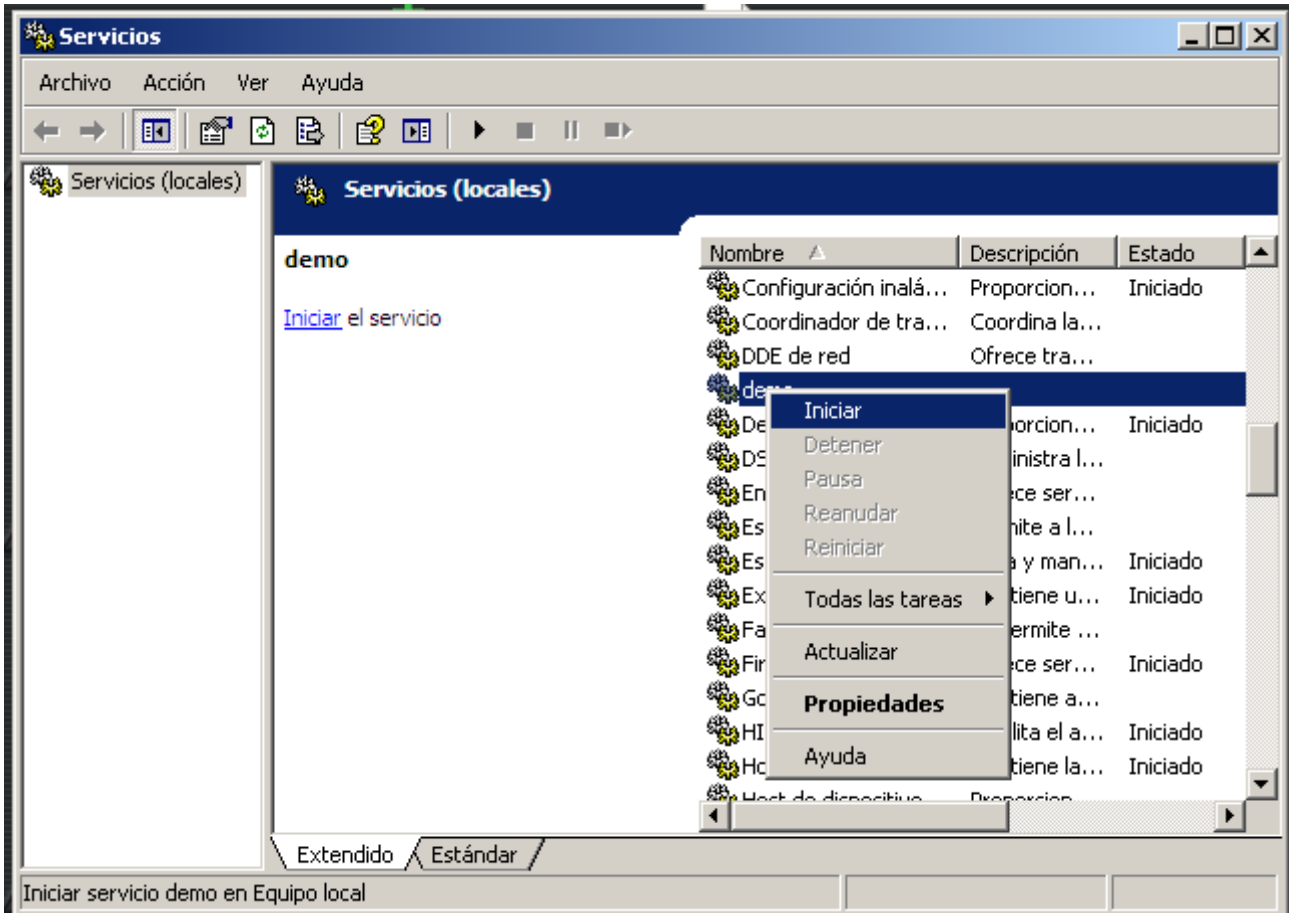
```
Mongrel service 'demo' installed as 'demo'.
```

Si examinan con cuidado la línea de órdenes, Mongrel atenderá esta aplicación en el puerto 4000. Otras aplicaciones que realicemos con Rails deberían llevar otro número de puerto.

Podemos iniciar el servicio de dos maneras:

- Por MSDOS

  `net start demo`
- Mediante la Consola de Administración de Windows

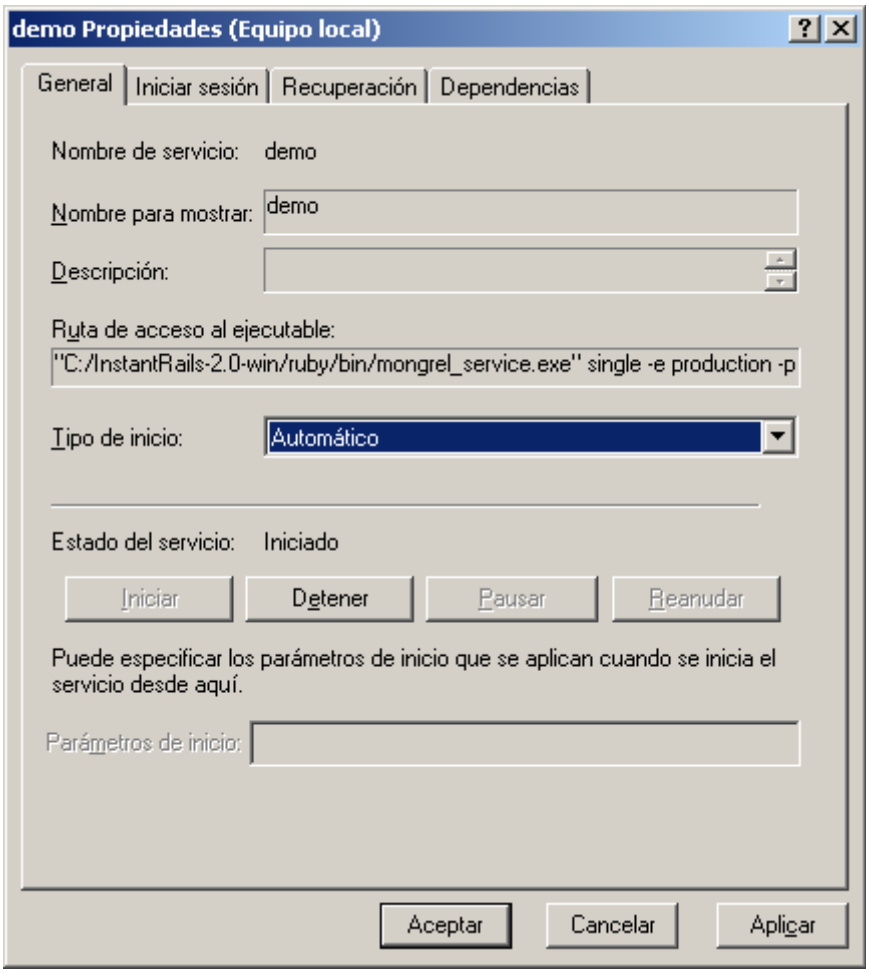


Deberíamos ver a Mongrel en producción, si apuntamos el navegador a

http://127.0.0.1:4000

Por cierto: en la Consola de Administración deberíamos dejar corriendo tanto la aplicación demo, como Apache; cada vez que inicie la computadora. Revise las propiedades de ambos servicios, y como aparece en la siguiente captura, actívelos como "Automático".

Configurar Apache



Hasta aquí tenemos la aplicación lista para servir a los usuarios. Sin embargo, habíamos mencionado que por razones de carga y estabilidad, es conveniente poner adelante de Mongrel al viejo, gruñón y confiable Apache. Bajo InstantRails el procedimiento es muy sencillo:

Debemos abrir **c:\InstantRails-2.0-win\conf_files\httpd.conf** y agregar al final las siguientes líneas:

```
<VirtualHost *>
    ServerName demo
    ProxyPass / http://localhost:4000/
    ProxyPassReverse / http://localhost:4000
</VirtualHost>
```

Resta que Apache quede como servicio (**cuidado con la ruta**)

```
C:\InstantRails-2.0-win\apache>apache -k install
Installing the Apache service
The Apache service has been installed successfully.
```

Podemos probarlo entrando a http://127.0.0.1

Ánimo, falta poco!

Finalmente, a nivel local, deberíamos agregar al archivo c:\windows\system32\drivers\etc\hosts, una línea parecida a a la siguiente

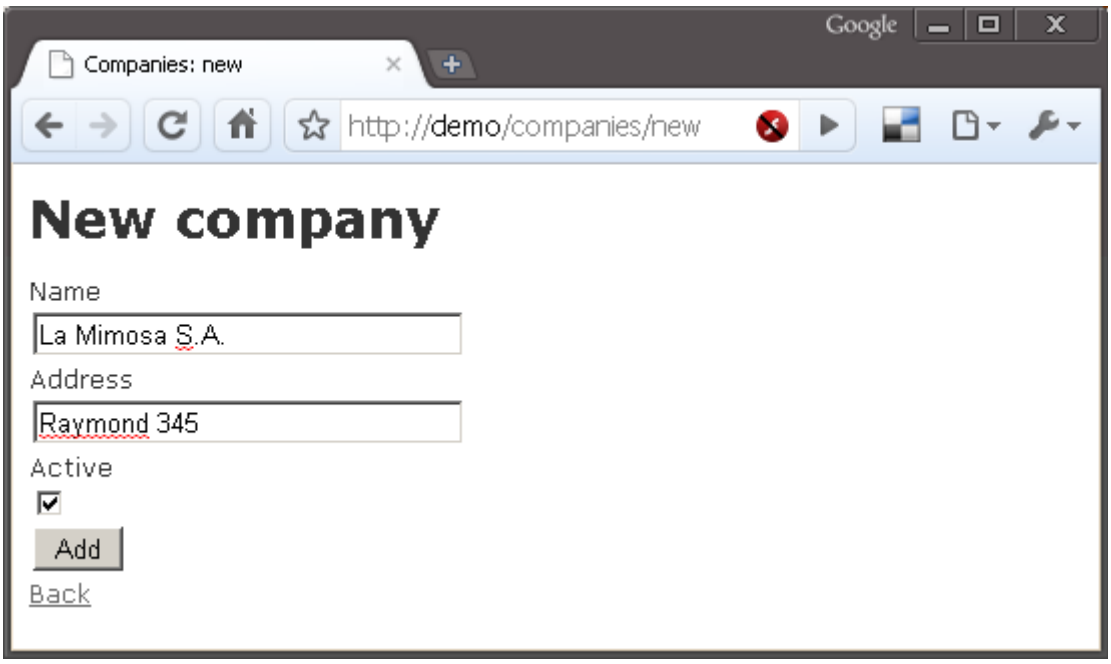
```
127.0.0.1       localhost mandragora mandragora.bunker.org.ar demo
```

Si deseamos que las otras computadoras de la LAN vean nuestra aplicación, deberíamos, o

bien configurar un Cache DNS local([21]), o alterar el archivo hosts de cada maquina, **agregando** (no cambiando) la siguiente línea.

```
192.168.1.33      demo
```

Si adivinan, esta linea apuntará referencias del tipo "http://demo" hacia la ip de la computadora que se encuentra sirviendo. En el ejemplo: 192.168.1.33



Errores

Si obtenemos errores:

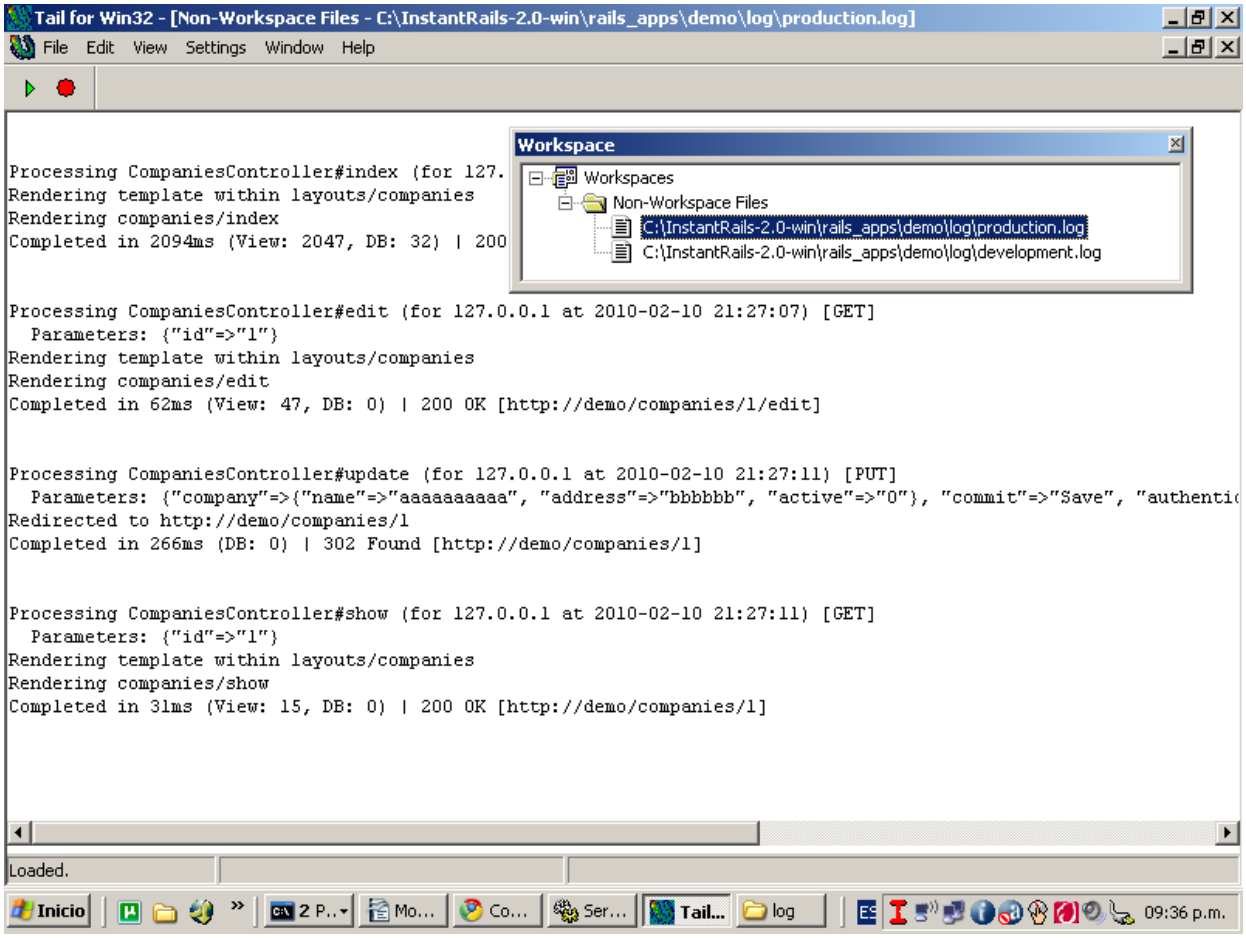
- Debemos asegurarnos efectivamente en la Consola de Administración de Windows (Inicio → Ejecutar → services.msc) que tanto la aplicación **demo** como **Apache** se encuentran **iniciados**. Si alteramos alguno de los archivos de configuración, deberíamos reiniciar preventivamente estos servicios.

3) Si obtenemos un error GET, el problema ocurre cuando Apache interroga a Mongrel, el cual podría estar detenido, o funcionar mal. Podemos probar como está funcionando el modo producción, con base y todo.

```
ruby bin\server --environment="production"
```

21 Los pasos para instalar un CacheDNS bajo Linux puede consultarlos en el Manual de Redes Libres, del mismo autor, en la dirección http://www.eim.esc.edu.ar/incubadora/redes.pdf

Si el puerto 3000 ya se encuentra ocupado por nuestra instancia de development,



puede pasar a script\server el parámetro -p (otro puerto).Si obtenemos un error "desconocido", deberíamos revisar el archivo **log\production.log** en busca de problemas. Este archivo, si bien puede ser abierto por cualquier editor, no debemos olvidar que todo el tiempo es alterado por Mongrel, de modo que si deseamos tener una vista "en tiempo real" de la aplicación, deberíamos obtener un equivalente al comando tail -f de Linux. Para Windows podemos recurrir a http://tailforwin32.sourceforge.net

Recuerde que **no todos los errores surgen ante el navegador**. Por razones de seguridad, solo se vuelcan al archivo de log para ser revisados por el administrador.

En caso que desee desbloquear ciertos errores, le aconsejo revise el documento http://guides.rubyonrails.org/debugging_rails_applications.html

Links recomendados:

http://mongrel.rubyforge.org

http://mongrel.rubyforge.org/wiki/Win32

Server de Producción en Linux Ubuntu

Como primera medida, instalamos Apache mediante la utilidad apt-get / aptitude. Si desea profundizar este tema, sírvase leer el capítulo "**Instalando binarios desde las fuentes**", presente en el Manual de **Redes Libres**. Este libro, del mismo autor, se encuentra disponible libremente en el sitio http://www.eim.esc.edu.ar/incubadora/redes.pdf

Para los ansiosos, la secuencia correcta es mediante la siguiente orden

```
s@mandragora:~$  sudo apt-get install apache2 apache2-prefork-dev libapr1-dev
libaprutil1-dev
```

A continuación instalamos Passenger, el cual nos pedirá que realicemos manualmente algunos pasos. También nos recordará la importancia de aprender la forma en que razona Apache.

```
s@mandragora:~$ gem install -r passenger
Building native extensions.  This could take a while...
Successfully installed passenger-2.2.9
1 gem installed
Installing ri documentation for passenger-2.2.9...
Installing RDoc documentation for passenger-2.2.9...

s@mandragora:~$ passenger-install-apache2-module
Welcome to the Phusion Passenger Apache 2 module installer, v2.2.9.
This installer will guide you through the entire installation process. It
shouldn't take more than 3 minutes in total.
Here's what you can expect from the installation process:

 1. The Apache 2 module will be installed for you.
 2. You'll learn how to configure Apache.
 3. You'll learn how to deploy a Ruby on Rails application.

Don't worry if anything goes wrong. This installer will advise you on how to
solve any problems.
Press Enter to continue, or Ctrl-C to abort.

--------------------------------------------
```

```
Checking for required software...



Some required software is not installed.
But don't worry, this installer will tell you how to install them.
Checking for required software...

 * GNU C++ compiler... found at /usr/bin/g++
 * Ruby development headers... found
 * OpenSSL support for Ruby... found
 * RubyGems... found
 * Rake... found at /usr/bin/rake
 * rack... found
 * Apache 2... found at /usr/sbin/apache2
 * Apache 2 development headers... found at /usr/bin/apxs2
 * fastthread... found
 * Apache Portable Runtime (APR) development headers... found at /usr/bin/apr-1-
config
 * Apache Portable Runtime Utility (APU) development headers... found at
/usr/bin/apu-1-config


--------------------------------------------
Compiling and installing Apache 2 module...
--------------------------------------------
The Apache 2 module was successfully installed.

Please edit your Apache configuration file, and add these lines:


   LoadModule passenger_module /usr/lib/ruby/gems/1.8/gems/passenger-
2.2.9/ext/apache2/mod_passenger.so
   PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-2.2.9
   PassengerRuby /usr/bin/ruby1.8

After you restart Apache, you are ready to deploy any number of Ruby on Rails
applications on Apache, without any further Ruby on Rails-specific
configuration!
```

Si prestamos atención, veremos que el script, antes de proseguir su instalación, nos solicita agregar las líneas en azul algunas lineas en el archivo de configuración de Apache. Tal

como se recomienda en la excelente obra Agile Web Development with Rails 4ta, Capítulo "Installing Passenger", buscamos el archivo de configuración de Apache. Este archivo puede diferir según la versión de Linux, Unix o Windows que estemos usando. Por suerte hay un comando para encontrarlo:

```
s@mandragora:~$ apache2ctl -V | grep SERVER_CONFIG_FILE
 -D SERVER_CONFIG_FILE="/etc/apache2/apache2.conf"
```

En el caso de Ubuntu y Debian, el archivo a modificar es **/etc/apache2/apache2.conf**

VirtualHosts

Respecto del script de instalación, no quiero dejar de comentar la importancia de configurar los VirtualHosts[22].

Para comenzar, en el archivo **/etc/apache2/apache2.conf**, buscamos las siguientes líneas, y las dejamos de la siguiente manera:

```
NameVirtualHost *:80
Include /etc/apache2/sites-enabled/
```

Aquí estamos indicando, en primer lugar, que este servidor *no atiende una sola aplicación*, sino varias. Como segunda medida, habilitamos el *new style* que tiene los administradores Linux (no Windows, no Unix) de repartir la configuración de cada aplicación en un archivo distinto. Esto es útil para no sobrecargar ni mezclar ítems en el archivo httpd.conf, o el apache2.conf

Luego, usando el *sudo power,* creamos un archivo **demo.conf**, en la ruta **/etc/apache2/sites-available**, cuyo contenido será equivalente a

```
<VirtualHost *:80>
  ServerName demo
  #Estamos trabajando en entorno de producción, pero si deseamos otro entorno:
```

22 La expresión "VirtualHost" hace referencia a la capacidad de Apache, de servir varios hosts, tales como http://pachanga.com, http://vivalapepa.org.ar, http://demo, etc, sobre una misma maquina, es decir, sobre un mismo host. Tambien se refiere al aprovechamiento de las capacidades de cache estático y multihilo de Apache, que le permite realizar Reverse Proxy hacia servicios que están escuchando en otros puertos, o en otros hosts (un escenario muy frecuente).

```
  #RailsEnv development
  DocumentRoot /home/s/demo/public
  <Directory /home/s/demo/public>
     Options FollowSymLinks
     Require all granted
  </Directory>
</VirtualHost>
```

Ahora, nuestro archivo demo, que se encuentra en /etc/apache2/sites-available, debería ser vinculado hacia una carpeta ubicada en /etc/apache2/sites-enabled

Si bien podemos hacer este paso en forma manual, es decir

```
ln -s /etc/apache2/sites-available/demo.conf /etc/apache2/sites-enabled/demo.conf
```

Podemos utilizar en cambio, un comando de Apache 2, mucho mas seguro:

```
s@mandragora:/etc/apache2$ sudo a2ensite demo.conf
Site demo installed; run /etc/init.d/apache2 reload to enable.
s@mandragora:/etc/apache2$ sudo /etc/init.d/apache2 reload
Reloading web server config apache2
Algunos mensajes... controlar solamente el [OK]
```

Por curiosidad, si corremos el siguiente comando

```
s@mandragora:/etc/apache2$ ls -l /etc/apache2/sites-enabled/
lrwxrwxrwx 1 root root 33 2010-02-11 22:18 demo.conf -> /etc/apache2/sites-available/demo.conf
```

Podremos ver que el archivo /etc/apache2/sites-available/demo posee un acceso directo que lo apunta desde /etc/apache2/sites-enabled/

Si ahora probamos cargar en el navegador las direcciones

http://127.0.1.1/ o http://127.0.0.1/

Deberíamos ver la aplicación corriendo.

Si en lugar de ello viéramos un mensaje de Apache indicando "It works" significa que hay un sitio llamado default (y otro llamado default-ssl) compitiendo en la misma carpeta donde habíamos dejado demo, es decir, en sites-avalilable.

Mi truco en este caso es, o bien modificar el archivo /etc/hosts, o simplemente

1) Mover ambos archivos *default* a otro lugar (por si los necesito para otra cosa), y

2) Crear un link simbólico que haga que mis sitio sea el sitio por default:

```
s@mandragora:/etc/apache2/sites-available $ sudo ln -s demo 000-default
```

3) Reiniciar Apache

```
s@mandragora:/etc/apache2/sites-available $ sudo /etc/init.d/apache2 restart
```

Un detalle a partir de Rails 4.2, es la exigencia en producción de generar una llave que Rails usa para varias cosas, mediante el comando

```
rake secret
```

Este comando lanza una larga cadena que debemos poner en **config/secrets.yml** en la linea

```
production:
  secret_key_base:
```

Multihosting

Supongamos que tenemos **otra** aplicación Rails en el mismo servidor. La llamaremos Pasantes (es uno de mis proyectos, para administrar pasantías). La crearemos, naturalmente, mediante el comando

s@mandragora:~$ rails pasantes

Al igual que con **demo**, generamos un archivo **pasantes** en /etc/apache2/sites-available

Su contenido es una copia muy parecida al archivo demo, presente en la misma carpeta.

```
<VirtualHost *:80>
  ServerName pasantes
  #Estamos trabajando en entorno de producción, pero si deseamos otro entorno:
  #RailsEnv development
  DocumentRoot /home/s/pasantes/public
  <Directory /home/s/pasantes/public>
    Options FollowSymLinks
    Require all granted
  </Directory>
</VirtualHost>
```

Lo habilitamos con

```
s@mandragora:$ sudo a2ensite pasantes
Site pasantes installed; run /etc/init.d/apache2 reload to enable.
s@mandragora:$ sudo /etc/init.d/apache2 reload
```

Además, en el archivo **/etc/hosts** nos aseguramos que la pila TCP, concretamente aquella que resuelve nombres, haga a 127.0.0.1 las solicitudes de nuestras aplicaciones:

```
127.0.0.1 localhost demo pasantes
127.0.1.1 mandragora demo pasantes
```

Alojar el proyecto sobre un Cloud

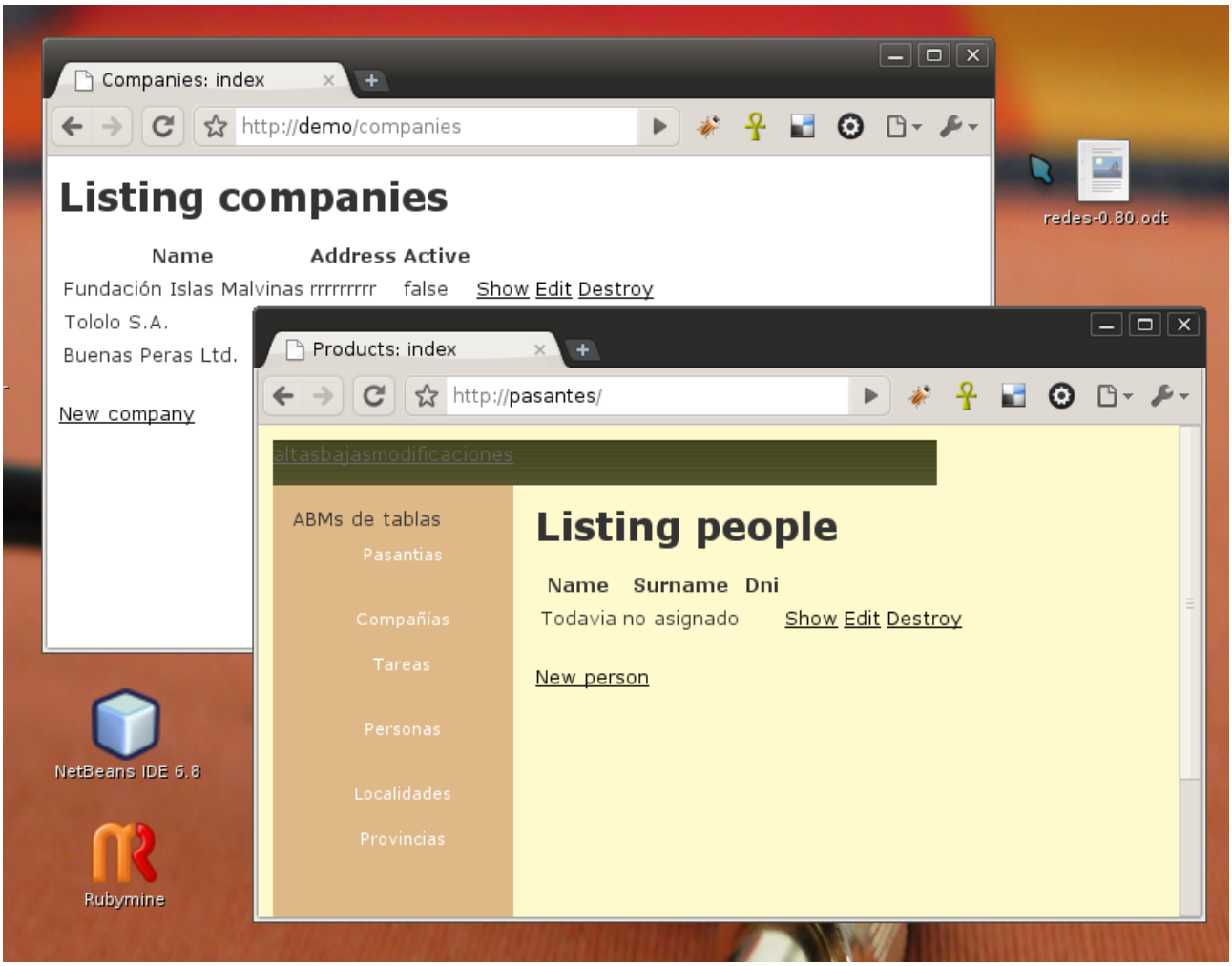
Un cloud es un racimo de servidores alojado en alguna parte de internet. Facebook, Gmail, Twitter y muchas otras aplicaciones residen en clouds o "nubes".
Cuando no queremos complicarnos con los detalles de instalación, o no queremos instalar un server en la empresa solo para correr con Rails, y no nos importa gastar unos dolares, podemos hacer uso de los muchos clouds disponibles en internet.

¿GitHub o Heroku?

A no confundir: a nivel gratuito

- Heroku ofrece hosting y algunos rudimentos de GIT.
- GITHUB no incluye hosting, pero a cambio ofrece muchas opciones para gestionar un proyecto colaborativo, y sin limites de tamaño.

En realidad la respuesta es GitHub "y" Heroku: GIT es capaz de regar código en ambos servicios mediante el comando push.



Para el siguiente ejemplo, haremos uso del servicio gratuito de Heroku. En caso que necesitemos mas "tracción de hosting", siempre podremos contratar allí servicios extras como Dynos o Workers. Si necesitamos mas potencia de gestión de proyecto, paralelamente podemos usar github, configurándolo como figura en http://teachmetocode.com/screencasts/getting-started-with-github

Alojar y Compartir en Heroku

Para alojar y compartir un proyecto en Heroku, deberíamos obtener una cuenta (es gratis), bajar e instalar Heroku Toolbelt. En el caso de Linux, en lugar de instalar el Toolbelt, alternativamente podemos correr los siguientes comandos:

```
s@mandragora:~/demo$ gem update
s@mandragora:~/demo$ gem update --system
Updating RubyGems
Nothing to update
s@mandragora:~/demo$ gem install heroku -V
s@mandragora:~/demo$ gem install taps -V
```

...

Generamos un conjunto de llaves:

 (Solo en Windows: los ssh-keygen se deben ejecutar desde el ícono **git bash**)

 **ssh-keygen -t dsa -f ~/.ssh/id_dsa -N ''**

 **ssh-keygen -t dsa -f ~/.ssh/id_rsa -N ''**

 **heroku keys:add**

 **heroku login**

Debemos tener en cuenta que Heroku usa PostgreSQL como motor de base en producción. Si queremos dejar Sqlite en nuestra modesta maquina de desarrollo, y PostgreSQL "arriba", debemos incluir la siguiente sentencia en el **Gemfile**

```
gem 'sqlite3', :group => [:development, :test]
```

```
group :production do
  gem 'pg'
end
```

Luego conviene instalar las siguientes dependencias para construir la gema pg:

```
sudo apt-get install libpq-dev build-essential
```

Y ahora si, como siempre luego de cualquier cambio en **Gemfile**

  `bundle install`

Ahora creamos un proyecto dentro del cluster Heroku. **No use como nombre de proyecto** "demo" **- ya se encuentra usada... por mi** :P

```
s@mandragora:~/demo$ heroku create demo
Enter your Heroku credentials.
Email: "escuelaint@gmail.com"
Password:
Uploading ssh public key /home/s/.ssh/id_rsa.pub
Creating demo.... done
Created http://demo.heroku.com/ | git@heroku.com:demo.git
```

Si no lo hemos hecho, pedimos a git que se instale su árbol de cambios

```
git init .
```

Agregamos la fuente remota

```
git remote add heroku git@heroku.com:demo.git
```

Técnicamente, la orden **git remote add** agrega por nosotros en el archivo **.git/config** una sección que reza:

```
[remote "heroku"]
      url = git@heroku.com:demo.git
      fetch = +refs/heads/*:refs/remotes/heroku/*
```

Configuramos algunos valores mas

```
s@mandragora:~/demo$ git config --global user.name "Sergio Alonso"
s@mandragora:~/demo$ git config --global user.email "escuelaint@gmail.com"
s@mandragora:~/demo$ git config --global color.ui "auto"
```

Antes de hacer el push hacia heroku, nos aseguramos de haber entregado localmente todos los cambios

```
git add -A
git commit -a
```

Ahora si, subimos la aplicación.

```
s@mandragora:~/demo$ git push heroku master
```

Técnicamente, todo esta listo para encontrar nuestra aplicación corriendo en

http://***(nuestra aplicación)***.heroku.com

Sin embargo, esto disparará un error en la versión online, puesto que falta subir la base. Ya había mencionado que los proyectos normalmente traen un archivo en la raíz llamado **.gitignore**, que sirve para que durante los push no se suban ciertos archivos y carpetas. Si miramos nuestro archivo **.gitignore**, encontraremos que nuestra modesta base de datos, la cual se encuentra en **db/development.sqlite3**, probablemente no ha subido:

```
.bundle
db/*.sqlite3
log/*.log
tmp/**/*
```

La manera obvia de permitir subir a la base **db/development.sqlite3** a producción sería quitar db/*.sqlite3 de .gitignore, pero si uno se detiene a pensarlo, lo coherente sería correr las migraciones ***sobre postgresql, al igual que heroku***, el cual sin duda debe estar muy optimizado, puesto que sirve decenas de miles de aplicaciones por segundo.
Igual, no hay que preocuparse con esto, porque ActiveRecord nos da independencia respecto de la base que se usa en producción (podría haber sido Mongodb, MySQL u otra cosa).
La línea db/*.sqlite3 existe por defecto por si utilizamos sqlite también en el server de producción, y no queremos sobrescribir con verdura los datos que en definitiva, nos están dando de comer.
Si estamos seguros de no sobrescribir *arriba* datos importantes, y tenemos la gema **taps**,

podemos quitar esa línea, y "empujar" la base entera, con migraciones y todo:

```
s@mandragora:~/demo$ heroku db:push
```

Ahora si, a probar la aplicación. Recuerde ejecutar **heroku logs** en caso de errores durante el deploy, o **heroku run rails console** para reparaciones de emergencia.

Retomar o Colaborar con un proyecto alojado en Heroku

Si deseamos tanto colaborar con un proyecto alojado en Heroku como continuar con uno propio nuestro, desde otro equipo, deberíamos comenzar creando un juego de llaves, instalar la gema **heroku**, enviar las llaves, y realizar un **clone** del proyecto. En pocas palabras:

```
s@mandragora:~$ ssh-keygen -t dsa -f ~/.ssh/id_dsa -N ''
s@mandragora:~$ ssh-keygen -t dsa -f ~/.ssh/id_rsa -N ''
s@mandragora:~$ gem install heroku
s@mandragora:~$ heroku keys:add
```

No olvidar situarse en alguna carpeta donde alojemos nuestros proyectos, y...

```
s@mandragora:~$ git clone git@heroku.com:demo.git
```

Entramos al proyecto, ponemos algo que no estaba antes...

```
s@mandragora:~$ cd demo
s@mandragora:~/demo$ touch archivo_nuevo
```

Touch no existe en Windows: solo fabrique un archivo en blanco.

Ponemos al día el repositorio local, explicamos la novedad...

```
s@mandragora:~/demo$ git add -A
s@mandragora:~/demo$ git commit -a -m "un archivo nuevo"
```

Si tenemos mas de un repositorio donde realizar nuestra entregas, como heroku, y a la vez github, a veces, antes de realizar el push por defecto (origin to master) conviene revisar las entradas existentes en .git/config.

Puede ser que hayamos alterado la estructura de las tablas, agregado o borrado cosas. En tal caso, sincronizamos

```
s@mandragora:~/demo$ heroku run rake db:migrate
```

Y subimos nuestros cambios al repositorio configurado por defecto.

```
s@mandragora:~/demo$ git push origin master
```

O, si tenemos mas de un repositorio configurado en .git/config, como github, especificamos heroku como destino

```
s@mandragora:~/demo$ git push heroku master
```

De nuevo, recuerde ejecutar **heroku logs** en caso de errores luego del deploy.

Si además queremos bajar la base con los datos en producción, sobreescribiendo la nuestra en development, para backupear, desarrollar o testear con datos reales (lo que se conoce como hacer "el restore" de la base), podemos hacer un

```
s@mandragora:~/demo$ heroku db:pull
Receiving schema
Receiving data
n tables, m records
users:          100% |==========================================| Time:
00:00:00
pages:          100% |==========================================| Time:
00:00:00
comments:       100% |==========================================| Time:
00:00:00
tags:           100% |==========================================| Time:
00:00:00
Receiving indexes
Resetting sequences
```

Apéndice A

¡Ayuda!

No quiero explicarle como buscar información en Internet. Pero es posible que haya estado buscando en el sitio erróneo. Veamos algunas técnicas para encontrar rápidamente el error. Para empezar, debemos tenerlo claro:

Debuggear en Ruby

Cuando un programa no hace lo que debe, siempre tenemos la opción de inspeccionarlo. A lo largo de la historia de Ruby han habido varias gemas para debuggear, tales como rdebug, debugger, y la última aceptada en Rails 4.2, byebug. No todas compilan en todos los sistemas operativos. Pero como su comportamiento es similar, mi técnica es instalar cualquiera de ellas.

Los programas escritos en ruby (archivos .**rb**) se inspeccionan mediante el comando **rdebug**, o tambien **ruby -rdebug mi_programa.rb**

```
gem install byebug
```

```
gem install byebug
```

Si esta falla, otras similares son debugger, ruby-debug y rdebug

El debugger se controla mediante palabras como **list**, **continue, next,** y otras que veremos en el siguiente capitulo.

Debugger en Rails

Quizas este es uno de los mejores argumentos que posee Rails como herramienta de trabajo. En tanto que otros lenguajes orientados a la web exigen de instalar costosas herramientas y plugins del lado del navegador, Rails resuelve todo por terminal, a la manera

de Ruby. Solo debemos cumplir algunos pasos extras para incluír a **rdebug**. Comenzamos agregando a la sección de Desarrollo del archivo **Gemfile** la gema necesaria:

```
group :development, :test do
     #repito: otras similares son debugger, ruby-debug y rdebug
     gem 'byebug'
end
```

Y luego corremos

```
bundle install
bundle update
```

Ahora corremos el server de pruebas con un añadido al final:

```
rails server --debugger
```

Una vez que haya terminado la sesión de debugging, corte la ejecución del servidor con Ctrl + C (o con la palabra **quit**). Recuerde que a diferencia del modo server "normal", el modo --debugger no vuelve a cargar cambios en el código automáticamente.
La manera simple de saber lo que esta haciendo una variable es escribir en cualquier parte

```
puts "la variable contiene ", variable
```

… y mirar dentro del log lo que sucede. Dejar "banderitas" por el camino a veces es la mejor idea para ir controlando el flujo logico. O utilizar logger, como fuera explicada anteriormente:

```
logger.info("#{Time.now} Borrando una empresa cuyo ID es el ##{@company.id}")
```

Para debuggear paso a paso

Otra manera, mas sofisticada, es insertar la palabra **debugger** o **byebug** en la linea del controlador donde deseamos inspeccionar.
En el caso de la vista hay que insertar <%= **debugger** %> o <%= **byebug** %>

Ejemplo:

```
 def check
    @bugs = Bug.all
    @bugs.each do |bug|
```

```
      params[:cb_].each do |parametro|
        if bug.id == parametro.id
          #En
     http://edgeguides.rubyonrails.org/debugging_rails_applications.html, a
partir de Rails 4.2, la siguiente línea es console
          debugger
          bug.accepted = true
          bug.save
        end
      end
    end
end
```

La linea **debugger (o byebug)** pausa la ejecución. Si vamos a la salida del **rails server**, descubriremos que la ejecución *se ha detenido*, y que nos espera un prompt para introducir comandos.

El comando **help** nos proveerá ayuda:

```
(rdb:2) help
ruby-debug help v0.10.3
Type 'help <command-name>' for help on a specific command

Available commands:
backtrace  delete   enable  help    next  quit     show    undisplay
break      disable  eval    info    p     reload   source  up
catch      display  exit    irb     pp    restart  step    var
condition  down     finish  list    ps    save     thread  where
continue   edit     frame   method  putl  set      trace
```

Por ejemplo, **list, o list <**numero de linea**>** nos mostrará la sección que estamos revisando:

```
(rdb:41) list
[3, 12] in
/home/s/Escritorio/Proyectos/r3uw/app/controllers/checklist_controller.rb
   3      @bichos = Bug.all
   4    end
   5
   6    def check
   7      debugger
=> 8      @bugs = Bug.all
   9      @bugs.each do |bug|
```

```
10        params[:cb_].each do |parametro|
11          if bug.id == parametro.id
12            bug.accepted = true
```

La orden **continue** seguirá la ejecución hasta el final (o hasta que encuentre una palabra **breakpoint** ;)

```
(rdb:2) continue
```

Otra manera de avanzar de a poco es decirle a **continue** hasta donde debe avanzar. Así, **continue 9** o **c 9** avanzará hasta dicha línea.
de Podemos también avanzar de a poco usando **step** o **next.** La diferencia entre uno y otro estriba en que si bien **next** continuará a la línea 9 del ejemplo, si lo volvemos a ejecutar allí, luego saltará la explicación del bucle **(next** no se meterá al bucle). En cambio **step** entrará a la 9. Pero si volvemos a ejecutarlo allí, en lugar de pasar a la 10, descenderá al nivel de las librerías de Rails (el stack) para mostrarnos ¡lo que ocurre dentro de .each!, lo cual requiere de mayores conocimientos. Si llega a suceder que el debugger se mete demasiado "por abajo" de las librerias, y queremos continuar la ejecución en un punto determinado, es decir "ser rescatados" de los intestinos de Rails, podemos pedirle que vuelva a subir en el stack mediante **up n** (siendo **n** la cantidad de niveles que queremos subir), o mediante **continue** (con **c**).
Cuidado con **continue**, pues una vez rescatados, si seguimos pulsando **c**, la ejecución se disparará hasta el final: ponga varios **breakpoints** por el camino. Use **continue** cuando se canse de debuggear o quiera ser rescatado de un bucle demasiado profundo.
También marcar un punto en particular con **b** o con **breakpoint**, de la siguiente manera:

```
b app/controllers/my_controller.rb:10
```

O si estamos parados en el mismo archivo que estamos revisando, simplemente

```
b 10
```

Si ha puesto muchos breakpoints (¡ponga todos los que pueda!), y no los recuerda, utilice

```
info b
```

En pocas palabras: utilice (n) **next** para avanzar, (s) **step** para descender niveles, (c) **continue** para seguir hasta el final. Por cierto, **next** si se ha usado al menos una vez, *activa*

*como atajo la tecla **Enter.***

Otra orden importante es **instance_variables**, la cual lista todas las variables en ejecución.

```
(rdb:195) instance_variables
["@_params", "@_config", "@lookup_context", "@_action_name", "@_status",
"@view_context_class", "@_response", "@_headers", "@action_has_layout",
"@_response_body", "@_env", "@_request", "@bugs"]
```

Si por ejemplo, queremos investigar la variable @_params, podemos revisarla al igual que en rdebug con la orden **display @params**, o en el caso de Rails, simplemente lo mencionamos en el debugger:

```
(rdb:195) @_params
{"cb_"=>{"6"=>"6", "7"=>"7", "13"=>"13", "27"=>"27", "28"=>"28", "2"=>"2",
"18"=>"18", "3"=>"3", "4"=>"4", "21"=>"21", "5"=>"5"}, "commit"=>"Submit
Selected", "authenticity_token"=>"n0wBjYDJqzEr1vi3ECVhNkqFnU1LqEd9g4TQK4Y5psg=",
"utf8"=>"\342\234\223", "action"=>"check", "controller"=>"checklist"}
```

Finalmente, si queremos hacer **watch** de la variables, lo que en el argot de los programadores significa "tener todo el tiempo un ojo encima" de la variable, algo muy útil en los bucles donde las variables van cambiando de estado, podemos hacer **display** <**variable**>.
Por ejemplo **display x** nos estará mostrando, al cabo de cada **next**, **step** o **continue** (hasta su breakpoint) el estado de **x**

Recuerde que si cambia el código fuente durante una sesión del debugger, no siempre podrá continuar **(continue)** hasta el final y volver a lanzar el código (**F5** en el navegador). *Seguirán figurando los mismos errores* a pesar de haber sido corregidos. Deberá hacer **Ctrl + C** en la salida del server, y recomenzar una sesión fresca. Es decir, debe hacer lo mismo que si estuviera con cualquier lenguaje.

Finalmente, algo magnífico de este debugger es que podemos llamar a **irb** en cualquier momento para jugar con las variables que vamos revisando.

En los siguientes sitios podrá encontrar una mayor explicación sobre el uso de esta herramienta

- http://www.tutorialspoint.com/ruby/ruby_debugger.htm

- http://guides.rubyonrails.org/debugging_rails_applications.html

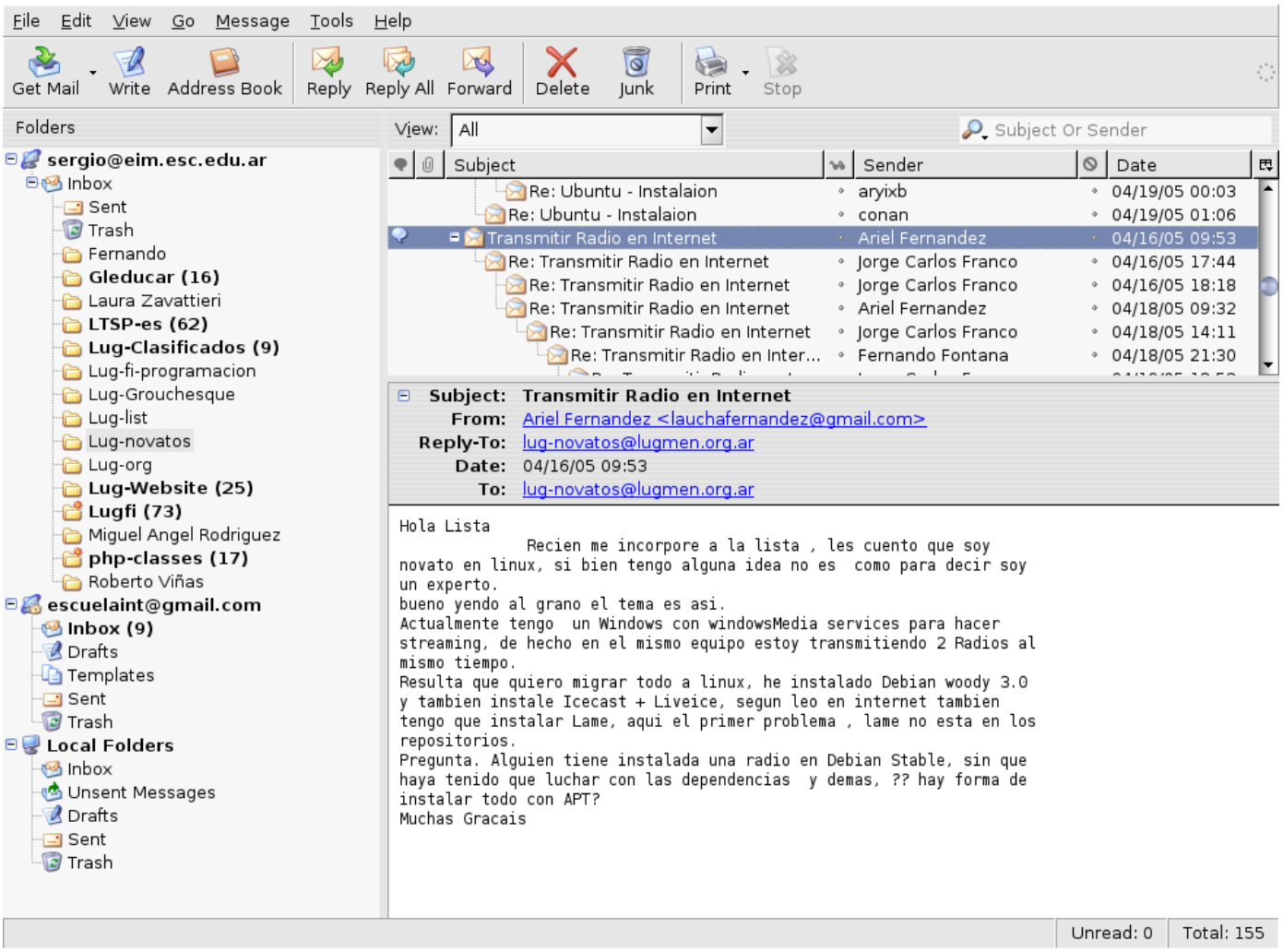
Listas de Correo

Las listas de correo son la principal fuente de resolución de problemas. Anímese, y anótese: se sorprenderá de encontrar una comunidad muy amable, compuesta con expertos que se sienten a gusto planteando problemas y soluciones.

Lista Española: http://lists.simplelogica.net/mailman/listinfo/ror-es

Lista Argentina: http://lista.rubyargentina.com.ar/listinfo.cgi/ruby-rubyargentina.com.ar

Para leer los correos que llegan, le recomiendo que utilice un cliente como Thunderbird, Kmail, Sylpheed y Evolution. Éstos programas permiten manejar las conversaciones



mediante threads (hilos). Los filtros de Gmail también son muy buenos anidando conversaciones.

Imagen obtenida de mi otro libro, Redes Libres, Apéndice A "Ayuda", una versión extendida de este Apéndice. Puede descargarlo desde www.eim.esc.edu.ar/incubadora/redes.pdf

Twitter

La comunidad de Software Libre, y particularmente la de Ruby, se encuentran conectadas casi exclusivamente mediante esta red social. Aquí hay noticias de primera mano, del mismo frente de combate en Ruby, y de los mismos creadores de Rails y sus gemas.

Si es usuario de Twitter, sabrá que el secreto consiste en seguir ("Follow") solamente gente interesante. Como ejemplo, le recomiendo siga a mis contactos: sus tweets son verdaderos mazazos de información útil:

Autores sobre Tutoriales:

- https://twitter.com/railstutorial - Michael Hartl, el creador de un libro mejor que este.
- https://twitter.com/RubyLearning - Satish Talim, Guru
- https://twitter.com/samruby - Otro gran escritor
- https://twitter.com/raymicha - Raymi Saldomando, la experta en maquetado

Noticias frescas:

- https://twitter.com/rails
- https://twitter.com/rubyflow
- https://twitter.com/RubyInside
- https://twitter.com/ruby_news
- https://twitter.com/rails_bookmarks

Programadores y "mineros de gemas" que *hay que seguir*:

- https://twitter.com/dhh - David Heinemeier Hansson, El creador de Ruby
- https://twitter.com/soveran: Michel Martens
- https://twitter.com/AkitaOnRails: Fabio Akita
- https://twitter.com/luislavena: Luis Lavena
- https://twitter.com/drnic – El increíble Dr Nic

- https://twitter.com/karancho – Yo :-)

Cheatsheets y RI

Habíamos ya mencionado a **ri** o "Ruby Information", un comando capaz de obtener desde consola, ayuda sobre los comandos y librerías de Ruby.

Por otro lado, están las cheat sheets o "tarjetas de referencia rápida": resúmenes rápidos y concisos, listos para ser pegados a un costado del monitor. Increíblemente, hay una gema en Ruby donde se centralizan todas las ayudas generadas por la comunidad.

Podemos encontrar varias en formato PDF, en las direcciones

- http://cheat.errtheblog.com
- https://www.dropbox.com/gallery/4822841/1/Rails%203%20Cheat%20Sheets?h=d08610#/

Pero siempre será mas *cool* solicitarlas vía línea de comandos:

 `gem install cheat`

 `gem install cheat`

Para ver los comandos disponibles

```
cheat cheat
```

Para ver una lista de las cheat sheets disponibles

```
cheat sheets
```

Ejemplo de cheat sheets que valen la pena mirar

```
cheat rails_console
cheat vim
cheat has_many
cheat rails
cheat unix
cheat cucumber
cheat wife
```

Apéndice B

Links Recomendados & Imperdibles

En Ingles

- Libros
  - Gratis:
    - www.railstutorial.org
  - Pagos
    - Agile Web Development with Rails, Cuarta Edición – prácticamente la biblia de Rails
  - Revistas, gratis:
    - http://railsmagazine.com
- ScreenCastings y videos gratis
    - www.railscasts.com
    - http://globalplaza.org/spaces/conferencia-rails/videos
    - http://teachmetocode.com/screencasts
- Tutoriales asistidos
  - Gratis
    - http://tryruby.org – Pequeño Tutorial online sobre Ruby
    - http://railsforzombies.org - Tutorial online sobre Ruby, muy simpático. Tiene otras secciones pagas, pero que valen la pena.
    - Consulte otros que voy dejando sincronizados con mi Firefox, en http://del.icio.us/karancho/ruby+tutoriales
  - Pagos
    - www.railstutorial.org (videos pagos).

En Español

- Libros
  - Gratis
    - La presente obra, y AFAIK[23], ningún otro.
  - Pagos
    - La presente obra, versión en papel, ideal para la cartera de la dama, y el

23 As Far I Know – Hasta donde pude saber.

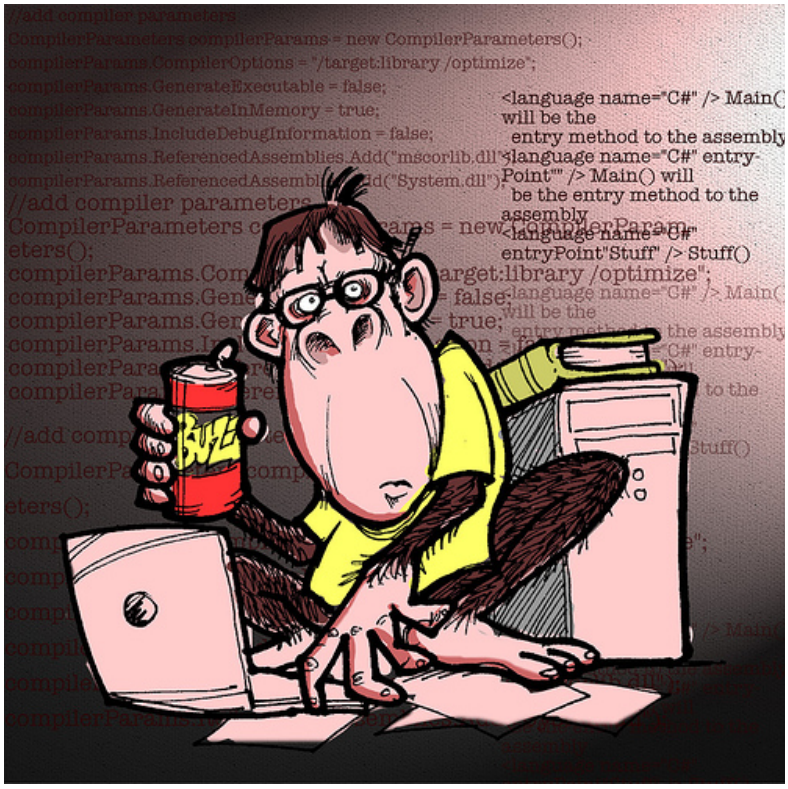
bolsillo del caballero.

- Casting, Videos:
  - http://es.asciicasts.com

Polígono de tiro para capítulos nuevos

Codemonkeys trabajando:

Entrar con casco



Sistema de login en rama testing de GIT

Ejemplo empleando Devise. Para algo mas artesanal, hay un port de AuthLogic para Rails 3 presente en https://github.com/csanz/authlogic-rails3-port

Bibliografía:

1. http://github.com/plataformatec/devise/
2. http://railscasts.com/episodes/209-introducing-devise (y su version asciicasts)
3. http://railscasts.com/episodes/210-customizing-devise (y su version asciicasts)

Los pasos, en términos generales, son:

- Definir en Gemfile la entrada

```
gem 'devise'
```

- Agregar devise al Gemfile, lanzar bundle

```
bundle install
```

- Instalar el nuevo generador

```
rails generate devise:install
```

- Tal como se recomienda por el envío de correo, agregar a config/environments/development.rb la linea

```
config.action_mailer.default_url_options = { :host => 'localhost:3000' }
```

Esta línea debe ser corregida cuando la aplicación sea subida en producción en un server con capacidad de envío de correo.
Queda indicar alguna ruta por defecto en **config/routes.rb** -ojo el plural-. Yo escogi (ADAPTAR)

```
root :to => "bugs#index"
```

Salir de config/routes.rb, porque en seguida el generador devise va a tocarlo.

```
rails generate devise User
```

```
      invoke  active_record
      create    app/models/user.rb
      invoke    test_unit
      create      test/unit/user_test.rb
      create      test/fixtures/users.yml
      inject  app/models/user.rb
      create  db/migrate/20100412200407_devise_create_users.rb
       route  devise_for :users
```

Cuando el generador devise genera (valga la redundancia) un modelo que sirva para que los demás se autentiquen con él, le da la forma que se puede apreciar en:
`app/models/user.rb`
`db/migrate/20100412200407_devise_create_users.rb`

… aunque en la nueva versión de la gema no hace falta cambiar nada, conviene leer para que sirven los items de adentro y adaptar si hiciera falta. La gema se encuentra muy bien documentada en http://github.com/plataformatec/devise/.

```
rake db:migrate
```

Conviene revisar las rutas con **rake routes**, ya aparece cierta información muy útil. Ejemplo:

Ruta	Método	Formato	Directorio/controlador/acción
new_user_session	GET	/users/sign_in(.:format)	devise/sessions#new
user_session	POST	/users/sign_in(.:format)	devise/sessions#create
destroy_user_session	DELETE	/users/sign_out(.:format)	devise/sessions#destroy

Estos datos que a primera vista no tienen sentido, son muy útiles cuando examinamos la traza del **rails server** (o del log equivalente en la carpeta **log**/).
Por ejemplo, la última línea es la ruta que elige el dispatcher cuando vemos pasar un

```
Started DELETE "/users/sign_out" for 127.0.0.1 at 2013-02-06 16:58:33 -0300
Processing by Devise::SessionsController#destroy as HTML
```

Significa que un usuario ha realizado un sign_out, y por lo tanto se ha destruido su sesión.

Rails y otros frameworks, como Sinatra y Padrino, hace uso de estas simples instrucciones

REST para quien desee lo desee usar como una API de Webservices.

Para que nos sirve esto de las rutas, me preguntaran. Bueno, pues nos permiten escribir fácilmente código. Volvamos al ejemplo de la línea

destroy_user_session	**DELETE**	/users/sign_out(.:format)	devise/sessions#destroy

Con esto podemos escribir un logout de la siguiente forma:

```
<%= link_to "Salir", destroy_user_session_path, :method => :delete %>
```

En este punto, en http://railscasts.com/episodes/209-introducing-devise proponen agregar al layout existente en **app/views/layouts/application.html.erb,** una pequeña zona que informe el estado de la sesión de usuario:

```
  <div id="user_nav">
   <% if user_signed_in? %>
     Bienvenido <%= current_user.email %>. No eres esta persona?
     <%= link_to "Salir", destroy_user_session_path, :method => :delete %>
   <% else %>
     <%= link_to "Inscribirse", new_user_registration_path %> or
     <%= link_to "Entrar", new_user_session_path %>
   <% end %>
 </div>
```

Lo que se traduce en que si el usuario no está logueado, le aparecerán las opciones

Inscribirse (Sign up) / Entrar (Sign In)

Si por el contrario, ya se encuentra logueado, mostrará solamente la opción para hacer

Salir (Logout).

Tu Ruta es mi Ruta (link_to & rake routes)

Por cierto, habrá notado que en Rails se tiene por costumbre utilizar **link_to** en forma recurrente, en lugar del clásico enlace **<a>**. Ejemplo de recién:

```
<%= link_to "Entrar", new_user_session_path %>
```

Este estilo de manejar los enlaces ha sido concebido para no tener que reescribir los vínculos de los modos **development**, cuya url contendrá **:3000/etc**, y del modo **production**,

que contendrá el dominio verdadero "**www.suempresa.com/etc**".
Sin embargo, subsiste la extraña manera de referirse al controlador/acción.
En el ejemplo... ¿de dónde sale **new_user_session_path**? En principio, del **routes.rb**, pero, ¿adonde va? . La mejor manera de saberlo es mirando todas las rutas disponibles mediante el comando de recién, **rake routes**. Copio nuevamente la primer línea, y remarco en color azul lo que usted debería copiar:

new_user_session	GET	/users/sign_in	{:controller=>"devise/sessions", :action=>"new"}

Así es como se construye el Entrar del ejemplo, que recordemos, se veía como:

```
<%= link_to "Entrar", new_user_session_path %>
```

Su identificación, por favor

Queda asegurar -en los controladores, lógicamente- el acceso a las acciones.
Para ello basta instalar al principio del controlador, una orden equivalente a:

```
before_filter :authenticate_user!, :except => [:show, :index]
```

Habilitamos la creación de vistas personalizadas

```
rails generate devise:views
```

Y generamos varias vistas personalizadas para los usuarios

```
rails generate devise:views users
```

Si ya hemos creado nuestro usuario, y quisiéramos inhibir la creación de usuarios nuevos, tenemos varias maneras:
La forma obvia es quitar de **app/views/layouts/application.html.erb** la linea

```
<%= link_to "Sign up", new_user_registration_path %> or
```

Otra manera, y que por seguridad también puede activarse, es buscar app/views/devise/registrations/new.html.erb y comentar la linea

```
<p><%= f.submit "Sign up" %></p>
```

De modo que quede algo así como

```
<p><%#= f.submit "Sign up" %></p>
Creación de nuevos usuarios: <blink>deshabilitado</blink>
```

Una tercer manera es crear nuestros propios controladores, o comenzar a manejar roles, como se detalla en http://github.com/plataformatec/devise

Queda:

- Fusionar con app del principio, demo.

Internacionalización (i18n)

Machete:

Según guía oficial: http://guides.rubyonrails.org/i18n.html

Modificar linea config.i18n.default_locale = :de de config/application.rb

Agregar a app/controllers/application_controller.rb

```
 before_filter :set_locale

 def set_locale
   # if params(:locale) is nil then I18n.default_locale will be used
   I18n.locale = params(:locale)
 end
```

Crear es.yml en config/locales al estilo de en.yml, y poner las traducciones. Idealmente: merguear con algunas localizaciones muy completas presentes https://github.com/svenfuchs/rails-i18n/tree/master/rails/locale, a la fecha la mas completa es la mexicana (es-MX).

```
hello_world: Hola Mundo
hello_flash: Hola Flash

Usar el helper t, ejemplos
Si en el controlador teníamos:
flash[:notice] = "Hello flash!"
Ahora podemos tener
flash[:notice] = t(:hello_flash)
Si en la vista teníamos
<h1>Hello world!</h1>
<p><%= flash[:notice] %></p>
Ahora podemos tener
<h1><%=t :hello_world %></h1>
<p><%= flash[:notice] %></p>

Para mayor comodidad, también se puede llamar al helper t dentro de los
```

f.label Por ejemplo, si en config/locales/es.yml hemos definido un hash: :locality_id

Entonces podemos intercalar el helper t al label

```
<%= f.label :locality_id %><br />
```

De tal modo que quede así:

```
<%= f.label t :locality_id %><br />
```

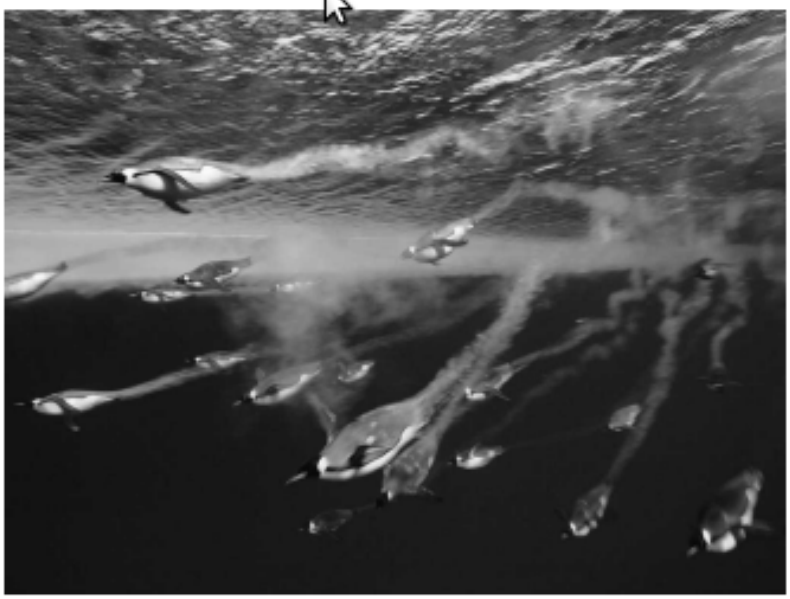
Autobombo: mas libros gratis

REDES LIBRES

V0.83 beta - 6 de setiembre de 2010

Redes Libres

Técnicas para armado de redes LAN,
utilizando Software Libre sobre ambientes mixtos



Si le gustó esta obra, quizás disfrute también de este otro libro gratuito.

Es un tratado sobre formas de integrar Linux en ambientes Windows, convirtiéndolo en una suerte de muleta para los demás servidores; una navaja suiza capaz de aligerar la carga que poseemos cada día los administradores de sistemas.

Escribí este libro como unos apuntes para mis alumnos de Análisis en Sistemas, y como un machete (ayuda de memoria, en Argentina) de aquellos pasos que en su momento me costó de varios días de experimentación. Allí he volcado mis experiencias configurando Apache, BIND, PHP, Squid Proxy, NAT, FTP, SSH, Samba, administrando usuarios, y velando por la seguridad y estabilidad de la red.

Creame: se puede disfrutar del trabajo de Sysadmin: el pingüinito Tux es la llave.

Puede descargarlo desde http://www.eim.esc.edu.ar/incubadora → redes.pdf