# El maldito libro de los Descarrilados


Ruby on Rails por quien ya recorrió el camino y no pereció en el intento.



Herrera

Tabla de contenidos

## Agradecimientos

Principalmente al lector por volverme escritor, secundariamente al descarrilado proponente que enriquece este texto día a día, y consecuentemente a mi familia por aguantarme mis deshoras.

La humanidad es inherentemente amante de la rutina y de la ilusión de control de ahí la resistencia al cambio.

# Prólogo

Un descarrilado en el contexto de este ensayo es un programador intentado entrar al mundo Rails y habiendo fracasado miserablemente con toda esa información en la web que al final no lo llevó a ninguna parte decide que este texto le puede ayudar, a su servidor le sucedió y decidió tomar sus apuntes y armar el maldito libro que nunca encontró, de ahí el sugerente y criticado título que solo supone una ayuda práctica en el mas puro de los propósitos.

El presente ensayo es solo un intento de iniciar al no iniciado, convertir al profano que proclama que con herramientas del milenio pasado se es aún competitivo, reubicar al infiel que trabaja con logicial privativo impactando negativamente la economía y la productividad y convencer al irreverente que no comprende el poder del trabajo bien realizado y la entrega a tiempo.

Todo esto no viene gratuita o fácilmente, en general la principal barrera es comprender que lo que se ha venido haciendo y le ha dado de comer al lector por muchos años es ahora absoleto o desactualizado, en general mi mejor consejo y el mas cruel es: Tomar todo lo que saben y tirarlo a la basura, mentalizarse a aprender, e iniciar otra vez desde el principio, si el lector no está dispuesto a esto, entonces este texto no es para usted.

Los requisitos mínimos los describiría algo así: Conocimientos técnicos básicos de GNU/BSD/UNIX, HTML, Javascript, SQL y algún lenguaje de 3era. generación orientado a objetos, programación por objetos, análisis de sistemas, diseño de base de datos, comunicaciones TCP/IP y Web 2.0, todo lo demás, aunque no cubriremos nada en profundidad, se verá en las siguientes páginas.

Lamentablemente al lector solo puedo llevarlo de la mano hasta la puerta, pero de ahí en adelante entrará solo, y deberá vencer su propia resistencia al cambio y seguir investigando y aprendiendo ya que Rails está en sus inicios, es imperfecto, es inmaduro y esta lleno de áreas para mejorar, y de hecho lo hace, este mismo ensayo se encuentra en su 6ta. iteración, ya que muchísimas de las acciones que se realizaban en las versiones anteriores ya no son necesarios y han dado paso a otras, en muchos casos, mas simples, pero algunas otras, mas crípticas.

Desafortunadamente la curva de aprendizaje es mucho mas amplia cuando se induce a un nuevo paradigma que cuando se induce a un nuevo lenguaje, es decir que un nuevo lenguaje se enseña en 1 mes, y el programador se hará productivo en 3 mas, mientras que un nuevo paradigma se enseña en 1 día y toma 3 meses asimilarlo, para poderse aplicar en producción al año. Este ensayo propone un lenguaje y un paradigma nuevos, de ahí que las cosas se deberán tomar con mucha serenidad

Afortunadamente una vez completada la etapa de asimilación este programador se volverá mucho mas productivo y mucho mas ordenado, y es por eso que todo el esfuerzo vale la pena.

Todo evoluciona y deberemos, el lector y su servidor, cambiar, sino seguiremos siendo... descarrilados.

---

Al final del texto se han colocado como capítulos propuestos aquellos temas que los descarrilados me han hecho llegar a través del correo electrónico, agradezco sinceramente las muestras de simpatía y las críticas que son siempre recibidas como constructivas, en el entendido de que estaré cubriendo esos temas en la medida de mis posibilidades, ya sea embebiéndolos en los capítulos ya existentes o escribiendo nuevos. Las actualizaciones estarán siempre en yottabi.com/mld.pdf, y raul.herrera@yottabi.com asunto MLD para las quejas, críticas, comentarios y aportaciones.

---

# Capítulo -1: Sobre logicial privativo

No lea este capítulo, es mas bien una catarsis

Comprendo en toda su cabalidad el impacto que ha causado microsoft y windows al mundo de la tecnología de la información, pasando por el de las comunicaciones y el del entretenimiento, entiendo completamente la inmensurable cantidad de personas que lleva el pan a la mesa de sus hogares día a día gracias a microsoft y también estoy de acuerdo como empresario que el mundo microsoft es una pequeña mina de oro.

Lo que no comprendo es: ¿¡Por que lo permitimos!? ... aparentemente consideramos correcto el mundo de la pantalla azul y el de los viruses, es normal que nuestras computadoras inicien en 10 minutos y consuman 475 watts en vez de 85 por que los discos duros no descansan, también se acepta el formateo como procedimiento normal de recuperación y no es para nada raro que al instalar un dispositivo nuevo la computadora se congele de siempre en siempre y mucho menos raro que cuando sale un nuevo windows nuestro equipo automáticamente es incompatible.

Estamos tan acostumbrados a este sistema no-operativo impuesto que la mayoría no se da cuenta que viene embebido, quiéralo o no tiene que pagarlo, no importa si lo va a usar o no, no importa si va a instalar un BSD o un GNU, usted tiene que pasar saludando al rey ofreciendo lealtad y obediencia con su cuota de US$39.99 a US$699.99 por lo bajo, dependiendo el grado de amor y lujuria que usted esté o no dispuesto a dar, al final eso no importará.

Los técnicos deberían ser millonarios, no conozco a ninguno que tenga un momento de ocio entre formatear y buscar drivers, los programadores deberían ganar la medalla olímpica por terminar el circuito "estar certificados y volverse obsoletos" en 45 segundos, los rockstars de los helpdesk deberían ser poseedores del trofeo a la paciencia por tener que explicar como la empresa mas rica del planeta puso "Formatear Disco" a la par de "Expulsar con seguridad", los sysadmin deberían ganar el Valium de oro por incrementar las ventas de Diazepán ya que solo así pueden conciliar el sueño ... y al usuario vil y mortal, la invitación a sembrar un árbol en la calle de filantropía por regalar nuestro dinero ganado con gran esfuerzo, sin obtener nada real a cambio.

La tan popular e inmisericorde interface gráfica es tan inútil para los técnicos y desarrolladores que se ven forzados a repetir una y otra vez las tareas comunes por que simplemente no hay como guardar historia y repetirla, y mucho menos una bitácora para auditoría, y por lo mismo el soporte ssh/telnet es muy limitado o nulo, en realidad hay que utilizar un delomelanicon para invocar algo a la LogMeIn para tener la ilusión de control, y por el amor de Dios no se le vaya a ocurrir darle "Expulsar" a un USB remotamente, ya que si no le dio formatear por error, habrá que reiniciar toda la máquina para montarlo otra vez.

Según el propio microsoft el 88% de los usuarios que pagaron windows vista y office 2007 usan windows xp y office 2003, y el 50% de los programadores que asistieron a los cursos de certificación .NET continúan utilizando visual basic 6. La media de "actualización" de vista a 7 es de US$119.99, el usuario promedio a través de 10 años de uso de productos microsoft y sus consecuencias gasta US$5430 por lustro y una PyME típica gringa gasta en productos microsoft y su consecuente mantenimiento un promedio de US $9850.00, por lustro en cada estación de trabajo.

Un sistema operativo ecológicamente consciente deberá proveer maneras de evitar la impresión de documentos, bajar consumo eléctrico, disminuir el impacto auditivo y visual, maximizar su eficacia para minimizar el tiempo usuario/máquina y prolongar la vida útil del equipo cuanto sea posible ... me he quedado sin aliento de tanto reir, estoy seguro que podríamos demandar a microsoft por daño irreparable al ambiente e irreverente uso de recursos no renovables.

Si en Latinoamérica hubiera una iniciativa real, venida de nuestros asesores gubernamentales de IT, de no usar logicial privativo nos ahorraríamos unos cuantos millones de dólares al año que actualmente le cedemos a empresas como microsoft, eset u oracle y tendríamos un poquito de dinero adicional para fomentar la inversión extranjera, disminuir la mortalidad infantil, mejorar la educación, combatir la delincuencia y en general ayudar a salir del condenado "tercer mundismo" que al final no se que significa sino es ser pobres babosos.

Por lo anterior, este ensayo no fomenta el uso de logicial privativo en ninguna de sus formas, el uso de las marcas registradas se ha realizado con fines ilustrativos y no pretenden ser palabras altisonantes o escatológicas.

Por cierto ... Rails es libre y hace el intento de correr en windows ... como todo lo de windows.

La fuente de las estadísticas ha sido una compilación basada en búsquedas a través de www.google.com, www.bing.com, office.microsoft.com, microsoft.com/presspass, eset.com, oracle.com, http://marketshare.hitslink.com, http://news.cnet.com, y por supuesto wiki.com

# Capítulo 0: Introducción

Rails es un marco de trabajo basado en Ruby para elaborar aplicaciones web, su nombre oficial es Ruby on Rails, se apega al patrón de diseño MVC, genera aplicaciones para la web 2.0, es agnóstico a la base de datos y reclama que desarrollar en él es 10 veces mas rápido que las herramientas tradicionales. Es tan radical que en muchos casos hay que tomar todo lo que el programador sabe y tirarlo a la basura, es por eso que su penetración en el medio empresarial ha sido lenta al igual que su popularidad, sin embargo otros marcos de trabajo toman a Rails de paradigma y lo siguen a buen paso aunque guardando la distancia, como es el caso de Django para Python o Symfony para PHP5.

Y como todo producto elitista esta basado en filosofías elitistas, es decir que dentro de la utopia de la que busca ser parte Rails existen lineas de pensamiento o principios que de una u otra forma intentan hacer más productivo y feliz al programador, he aquí una recopilación:

No te repitas, nunca escribas en dos lugares el mismo cálculo, o el mismo proceso o la misma función, si lo haces entonces nunca estarás seguro de donde viene tu problema, además ... duplicar esfuerzos no es inteligente, esta basado en el principio del origen único de la verdad, el reduccionismo metodológico y KISS.

Convención sobre configuración , la aplicación ya deberá funcionar sin interacción con el usuario, por que se han suplido las necesidades iniciales desde el mismo desarrollo, el usuario deberá poder cambiar estos valores para que se adapten a su necesidad en particular, en el caso del desarrollo provee las siguientes ventajas:

- Permitir a los desarrolladores nuevos aprender un sistema rápidamente.
- Promueve la uniformidad.
- Promueve el dinamismo.

Sin embargo ofrece también algunas desventajas:

- Se requiere familiarizarse con los defaults.
- Aumenta el peso de la aplicación.
- Puede ser difícil modificar un valor de fábrica.

Desarrollo Ágil, Reingeniería constante de tus acciones y tus ideas, es un método reactivo que provee soluciones al momento del problema, sus bases son las siguientes:

- **Las personas y sus interacciones** deben estar sobre los ***procesos y las herramientas***.
- **La aplicación funcionará** y no deberá ser necesaria una ***documentación detallada o abundante***.
- **La colaboración con cliente** por sobre la ***negociación***.
- **La necesidad** siempre estará sobre la ***planificación***.

En el desarrollo ágil los elementos de la izquierda son mas importantes que los elementos de la derecha esto es opuesto al desarrollo tradicional, la idea es encontrar lo siguiente:

- La satisfacción del cliente a través de una veloz y continua entrega de aplicaciones útiles.
- Aplicaciones útiles entregadas frecuentemente (semanas en vez de meses).
- La utilidad de las aplicaciones es la medida del progreso del proyecto.
- Todos los cambios son bienvenidos aún aquellos de último momento.
- Cooperación y acercamiento diario entre la gente del negocio y la de desarrollo.

- La conversación cara a cara es la mejor forma de comunicación.
- Los proyectos están rodeados de gente motivada en la que se puede confiar.
- Atención continua a la excelencia técnica y un buen diseño.
- Simplicidad.
- Equipos de organización natural.
- Adaptación regular a circunstancias cambiantes.

Mejores Prácticas, codifica y luego recodifica para que quede legible para ti mismo y tu grupo de trabajo, dentro de 8 semanas no recordarás que hiciste y mucho menos como, comenta y documenta todo lo que puedas y apégate a los standards, tu aplicación debe contener exclusivamente el código que se ejecutará, no te conviene tener focos de confusión, maneja el error de forma elegante y minimiza la repercusión, trabaja simple, por que lo simple es bello por que es simple.

Desarrollo llevado por pruebas, Escribe una linea a la vez, coméntala y pruébala, has pruebas consecuentes con múltiples tipos de datos y ensaya una y otra vez el código, de esta forma aumentarás la probabilidad de tener código libre de errores a llegar a la fase de pruebas. En lo posible automatizalas.

Código Limpio, Toma el código que no ha sido realizado con las mejores prácticas y vuelve a escribirlo para que sea legible para ti y para tu grupo de trabajo, pásale las pruebas y asegúrate que quedo tan bien como cuando lo encontraste y que ahora es legible, has que al menos un compañero le haga la prueba de la legibilidad.

Versionado, utiliza alguna herramienta para manejar tus versiones, nunca sabrás cuando deberás dar algunos pasos hacia atrás, nunca sabrás que porciones de código te seguirán sirviendo o cuales habrá que reemplazar, y por sobre todo nunca sabrás cuando el siniestro acaecerá.

Compartir, informa de tus actividades al grupo de trabajo, y procura que ellos también lo hagan, mantengan una clara y eficiente comunicación en función de la planificación o de la necesidad, procúrate una bitácora interactiva para asignar, comentar y validar las actividades, toma en cuenta las holguras y ajusta diariamente el itinerario y hazlo del conocimiento de todos.

Agnosticismo, sé independiente, debes ser capaz de trabajar en cualquier escenario, utiliza solamente herramientas standard, no debes estar atado a sistemas operativos, editores, bases de datos, equipo o modas, ten claro tu objetivo y cúmplelo.

Profesionalismo, Sin importar si tienes entregas prontas o estás presionado por tus actividades, compórtate calmado y sereno ante todos los problemas, lo que menos conviene en el momento de la verdad es tener al capitán del barco con un ataque de arrebatos, incongruencia o legítima estupidez.

Visión, No te comprometas en lo profesional a actividades que no estás seguro que puedas realizar, tampoco ofrezcas entregas con tiempos demasiado cortos, entiende y respeta a tu grupo de trabajo comprendiendo sus capacidades y limitaciones. Entrénate a tí y a tu grupo de trabajo todo el tiempo para estar actualizados y preparados para los nuevos productos y servicios que deseas ofrecer.

Como toda regla tiene su excepción, estas son opiniones, sugerencias o simples experiencias compartidas de las diferentes comunidades, en conclusión deberíamos tomar las que nos sirvan y dejar pendientes las que no, mientas mas apliquemos mejor estaremos.

¡Atención! Tenga muy en cuenta que al copiar y pegar los ejemplos de este texto, es probable que no se transfieran correctamente los tabuladores, retornos de carro, comillas y líneas largas, causando errores muy ocultos que suelen ser indescifrables al usuario inexperto y por ende se puede perder la continuidad ... y la paciencia. Como este texto propone una actividad didáctica mi recomendación es que el lector los ingrese y los pruebe uno a uno.

# Capitulo 1: Instalación

Antes de decir cualquier cosa, algunos OS ofrecen a través de comandos de instalación de paquetes nativos la instalación de Rails, esto puede ser bueno en un principio, pero los repositorios no se actualizan a la velocidad que lo hacen las gemas, de tal suerte que es posible que el lector se quede anclado a una versión en particular sin poderse actualizar, le suplico lo intente con los métodos que detallo a continuación y juzgue por si mismo cual es el que mejor le conviene.

Nuestra plataforma de ensayos será un Linux Debian, cualquiera de sus múltiples sabores nos es útil, utilizaremos la utilería apt-get pero bien podría ser aptitude o cualquier otra en la que el lector se sintiera cómodo.

Sugiero que iniciemos con una instalación de LAMP ya que la inmensa mayoría de lectores está utilizando profesionalmente MySQL. Para nuestros ensayos, en este texto, utilizaremos SQLite que es una base de datos simple y portátil que aunque dicen que no está preparada para producción pero eso no es del todo cierto.

El proceso de instalación de LAMP se resume así:

1. Instalación del Sistema Operativo mismo
2. Instalación de Apache2 (httpd)
3. Instalación de MySQL
4. Instalación de PHP5
5. Instalación de phpMyAdmin

El Proceso de instalación de Rails con el método oficial se resume así:

1. Instalación de Ruby
2. Instalación de Rubygems
3. Instalación de Rails
4. Redireccionamiento a las gemas
5. Instalación del Servidor Webrick, Mongrel, Passenger
6. Instalación de las bibliotecas de base de datos
7. Instalación de gemas adicionales

Las 2 instalaciones son totalmente independientes y por lo tanto puede realizarse una sin la otra, en función de lo que el lector desee puede obviar la instalación de LAMP, solo será necesaria una instalación Linux cualquiera.

## Instalación LAMP

Por lo pronto iniciaremos con una instalación típica LAMP, esto bajo ninguna circunstancia es requisito, pero será muy adecuado para los usuarios interesados en el uso de MySQL y su manejador web mas popular phpMyAdmin, para esto necesitaremos, instalar Apache HTTP Server, MySQL Server, PHP5 y phpMyAdmin.

Y como este texto fomenta el trabajo en equipo instalaremos los demonios SSH y FTP que son muy útiles para el acceso remoto y el trasiego de datos.

Por supuesto se asume un Linux Debian como se dijo anteriormente, con un usuario capaz de instalar paquetes, para motivos ilustrativos nuestro usuario se llamará así "usuario" y nuestra computadora "host".

Iniciamos con una actualización de la base de datos de repositorios, y las bibliotecas esenciales:

```
usuario@host:~$ sudo apt-get update
usuario@host:~$ sudo apt-get build-essential
```

En este momento nos solicitará la clave, si no hemos invocado el comando sudo con anterioridad, sudo ejecuta el comando a su derecha con privilegiós de root, pero es necesario ser un sudoer para poder invocarlo.

Si por alguna razón tenemos la necesidad de actualizar el sistema a los últimos paquetes y no nos importa esperar entonces podemos correr:

```
usuario@host:~$ sudo apt-get upgrade
```

Para eliminar los paquetes no necesarios u obsoletos entonces podemos ejecutar:

```
usuario@host:~$ sudo apt-get autoremove
usuario@host:~$ sudo apt-get clean
```

# Apache2 (httpd)

## Instalación

```
usuario@host:~$ sudo apt-get install apache2
```

Una vez terminada la instalación, por favor vaya a su navegador y escriba el URL http://localhost si ve un mensaje que dice "It Works!" entonces la instalación fue exitosa.

Apache HTTP Server inicia automáticamente en cada inicio de servidor así que no es necesario nada adicional.

Para bajar el servicio:

```
usuario@host:~$ sudo service apache2 stop
```

Para subir el servicio:

```
usuario@host:~$ sudo service apache2 start
```

Para reiniciar el servicio:

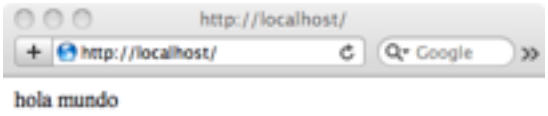
```
usuario@host:~$ sudo service apache2 restart
```

El directorio donde se aloja el contenido publicado por Apache2 es en Debian /var/www

Es posible en este momento modificar el archivo /var/www/index.html para hacer una prueba fehaciente de la ubicación de los archivos publicados bastará con escribir:

```
/var/www/index.html

hola mundo
```



# MySQL server

## Instalación

```
usuario@host:~$ sudo apt-get install mysql-server
```

Solicitará que se ingrese la clave de root del servidor MySQL para motivos de este texto ingresaremos 1234., nos pedirá repetirla y concluirá la instalación después de unos minutos. Por favor no confunda este usuario root con el usuario root de su sistema operativo son completamente distintos.

Una vez instalado podremos verificar la correcta instalación invocando:

```
usuario@host:~$ mysql --version
```

```
mysql  Ver 14.14 Ditrib 5.1.54, for debian-linux-gnu (i686) using readline 6.2
```

No necesariamente debe ser exacto a este mensaje, aquí nos dice que MySQL ha sido instalado correctamente. Por definición MySQL server iniciará automáticamente en cada inicio así que no será necesario ninguna acción adicional.

Para bajar el servicio:

```
usuario@host:~$ sudo service mysql stop
```

Para subir el servicio:

```
usuario@host:~$ sudo service mysql start
```

Para reiniciar el servicio:

```
usuario@host:~$ sudo service mysql restart
```

Para sacar una copia de respaldo de la base de datos:

```
usuario@host:~$ mysqldump -p1234 -u root nombrebd > nombrebd.sql
```

Para recuperar una copia de respaldo en la misma o en otra base de datos:

```
usuario@host:~$ mysql -p -u root nombrebddest < nombrebd.sql
```

Para una prueba mas realista del servidor podemos ejecutar:

```
usuario@host:~$ mysql -p -u root
```

En este momento tendremos la consola de mysql, donde crearemos una base de datos, una tabla, unas inserciónes una consulta y eliminaremos la base de datos, como se ve en el ejemplo:

```
mysql> create database prueba1;
Query OK, 1 row affected (0.00 sec)
mysql> use prueba;
Database changed
mysql> create table tabla1 (codigo integer primary key, descripcion varchar(255));
Query OK, 0 row affected (0.00 sec)
mysql> insert into tabla1 values (1,"uno");
Query OK, 1 row affected (0.00 sec)

mysql> insert into tabla1 values (2,"dos");
Query OK, 1 row affected (0.00 sec)

mysql> select * from tabla1;
+--------+-------------+
| codigo | descripcion |
+--------+-------------+
|      1 | uno         |
|      2 | dos         |
+--------+-------------+
2 rows in set (0.00 sec)
mysql> drop database prueba;
Query OK, 1 row affected (0.00 sec)

mysql> exit
```

Algunos comandos de mysql:

| | |
|---|---|
| Crear una base de datos | `create database nombrebd;` |
| Borrar base de datos | `drop database nombrebd;` |
| Mostrar bases de datos | `show databases;` |
| Usar una base de datos | `use nombrebd;` |
| Mostrar las tablas de la BD actual | `show tables;` |
| Mostrar la estructura de una tabla | `describe nombretabla;` |
| Salir | `exit` |

# PHP5

## Instalación

Si queremos instalar PHP5 sobre Apache para que este publique nuestras paginas .php entonces es menester tener instalado Apache2 a priori a esta instalación, si por alguna razón no se hace en ese orden entonces se vuelve necesario configurar a mano Apache2, por favor consulte la documentación de Apache2 para esta actividad.

La instalación de PHP5 se hace a través de:

```
usuario@host:~$ sudo apt-get install php5
```

Una vez instalado podremos verificar la correcta instalación invocando:

```
usuario@host:~$ php -v
```

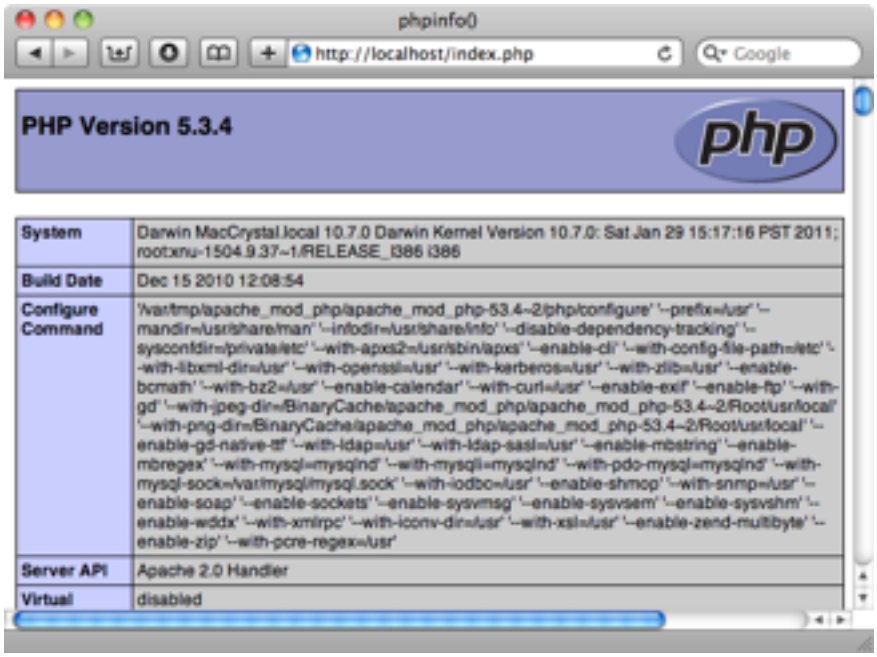
```
PHP 5.3.4 (cli) (built: Dec 15 2010 12:15:07)
Copyright (c) 1997-2010 The PHP Group
Zend Engine v2.3.0, Copyright (c) 1998-2010 Zend Technologies
```

No necesariamente debe ser exacto a este mensaje, aquí nos dice que PHP5 ha sido instalado correctamente.

Para una prueba completa de la integración con Apache2 habrá que crear un archivo de nombre index.php en /var/www que es donde Apache2 buscar los archivos de forma predeterminada en Debian, con el siguiente código, así podremos corroborar la instalación correcta.

```
/var/www/index.php

<?php
phpinfo();
?>
```




Esta es una presentación parcial de la información que phpinfo presenta, pero da una idea de lo que se espera.

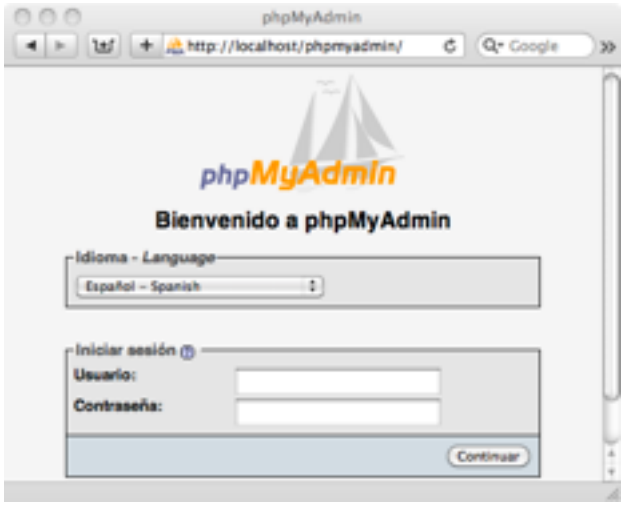
## Instalación phpMyAdmin

Para la instalación de phpMyAdmin se deben cumplir con los requisitos de tener instalado Apache HTTP Server, MySQL Server y PHP5, aunque es posible instalarlo sin tener MySQL instalado localmente, no está dentro del contexto de este texto ese tópico.

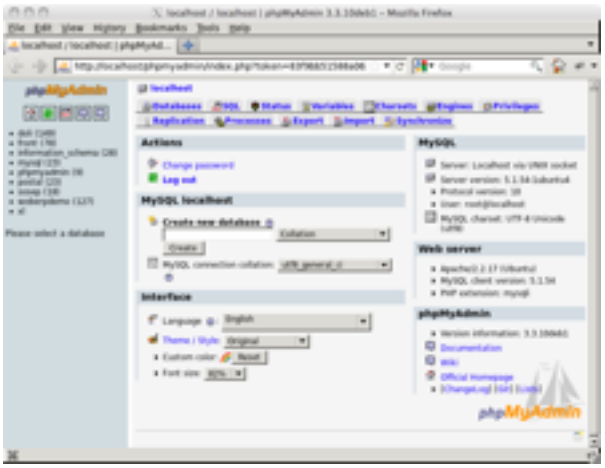
```
usuario@host:~$ sudo apt-get install phpmyadmin
```

- Una vez instalado nos pedirá la elección entre apache2 y lightpd, seleccionaremos apache2.
- Después de unos instantes nos mostrará el mensaje de requerimiento de la base de datos, esto se supone ya está instalado así que solo lo aceptaremos.
- Le diremos que si queremos que se autoconfigure.
- Ahora nos pedirá la clave de root de MySQL, le diremos 1234 como se estipuló en la instalación de MySQL.
- Ahora nos pedirá una clave para poder registrar la base de datos de phpMyAdmin en MySQL, aquí no nos la complicaremos y le colocaremos la misma 1234, y la repetiremos para confirmarla.

Si todo fué exitoso ingresaremos el URL http://localhost/phpmyadmin en nuestro navegador y si vemos la siguiente pantalla es porque tanto el Apache2, el php5 y le phpMyAdmin están funcionando perfectamente.




Ahora bien hay que probar el MySQL, así que ingresaremos usuario root y clave 1234, si vemos algo como esto, todo fue un éxito y tenemos un LAMP perfectamente instalado.



# Instalación de Ruby on Rails

## Método a la Linux

Este método lo daré sin mucho miramiento, no lo recomiendo y por lo mismo no lo detallo, en general funciona, a veces no lo hace, y no estoy muy interesado en estudiar el porque.

Para instalar:

```
usuario@host:~$ sudo apt-get install rails
```

Para desinstalar:

```
usuario@host:~$ sudo apt-get remove rails
```

Con este método no estoy muy seguro sobre el manejo de las gemas, por favor córralo bajo su entera responsabilidad y no me consulte sobre los problemas que seguramente encontrará

## Método Independiente

Este será el método de la salida cobarde, es decir ¨quiero que funcione pero no me interesa como¨, no recomiendo este método, está aquí debido a la petición de los descarrilados que necesitan donde ensayar sin tanta dificultad, así que tiene algún valor didáctico, pero hasta ahí, de tal suerte que será un listado simplista y corto.

Turnkey Linux, http://www.turnkeylinux.org/rails (Agradable, simple, Appliance)
Instant Rails, http://rubyforge.org/frs/?group_id=904 (Lento, Obsoleto, Window$)
RubyStack, http://bitnami.org/stack/rubystack (Actualizado, MultiOS, Standard)
Lomotive, http://sourceforge.net/projects/locomotive/ (Obsoleto, MacOS, Bellísimo)

Dentro de los males el menor, prueben RubyStack, es gratamente impresionante.

## Método Oficial

Si va a continuar con lo que resto del capítulo asegúrese que Rails esta desinstalado de la versión de los repositorios con el comando:

```
usuario@host:~$ sudo apt-get remove rails
```

Necesitaremos preparar el terreno para instalación así que nos aseguraremos que todo está listo con:

```
usuario@host:~$ sudo apt-get install build-essential
usuario@host:~$ sudo apt-get install libopenssl-ruby libfcgi-dev
```

Si va instalar aplicaciones adicionales como Passenger entonces también corra:

```
usuario@host:~$ sudo apt-get install libssl-dev zlib1g-dev
usuario@host:~$ sudo apt-get install libcurl4-openssl-dev
```

Rails es un framework full stack esto significa que viene con servidor de aplicaciones, utilerias de generación de todo tipo de módulos, generador de bitacoras pruebas, migraciones, y un gran etcétera y por lo mismo no se trata de una instalación a la ligera, y es muy importante que el lector conozca cada paso de la instalación ya que en no pocas ocasiones suceden errores que pueden localizarse y corregirse aisladamente del resto de la instalación.

La instalación completa tomará al rededor de 15 minutos en función de la velocidad de descarga y de instalación propiamente, los pasos son muy simples y se ha probado en muchas ocasiones este método sin embargo como todo evoluciona si por alguna razón alguna sección no funciona como debiere deberá consultar el problema puntualmente, recuerde que google es su amigo y el autor también.

Es muy importante que pruebe cada parte de la instalación por separado y conforme la vaya realizando, no de un paso adelante sin asegurar completamente el anterior.

## Instalación Ruby

La página oficial de Ruby es http://www.ruby-lang.org/es/, este lenguaje por si mismo es quizás, en mi humilde opinión, lo mejor que le pudo pasar a la programación por objetos y existe una enorme cantidad de información acerca de él, y es por mucho una de mis mas frecuentes recomendaciones de lectura técnica. Para verificar su instalación y versión corremos:

```
usuario@host:~$ ruby -v
```

```
ruby 1.8.7 (2010-08-16 patchlevel 302) [i686-Linux]
```

Si no ve algo como esto, entonces es menester que instale Ruby, en muchos casos el mismo OS le sugerirá el comando, hágale caso, pero tambien deberá instalar las bibliotecas de desarrollo para que Gem pueda efectuar los ajustes en los manejadores de base de datos y cualquier otra cosa que se presente. Por favor escriba:

```
usuario@host:~$ sudo apt-get install ruby irb
usuario@host:~$ sudo apt-get install ruby1.8-dev
```

Las lineas son comandos distintos, por favor presione <Enter> después de cada una.

El comando sudo permite la ejecución del comando que le sigue con privilegios de superusuario así que le pedirá su clave, si por alguna razón le da algún tipo de error tendrá que pedirle al administrador que le permita instalar paquetes en su máquina o que lo añada al grupo de usuarios Sudoers para poder ejecutar el comando sudo. Y por supuesto deberá estar conectado a la red para poder bajar el paquete correspondiente.

Una vez el comando anterior fue exitoso reintente verificar la versión de Ruby.

```
usuario@host:~$ ruby -v
```

---

Es muy importante resaltar que apt-get al igual que la mayoría de comandos en los nuevos ¨Unixes¨ poseen la característica de escritura rápida predictiva, es decir que bastará con tipear solo las primeras letras y presionar <Tabulador> para que el OS intente descifrar cual es el siguiente comando y si no puede, sugerir las alternativas, incluyendo los paquetes a instalar, es decir que solo basta con colocar ¨ru¨y presionar tabulador para que el comando sea sugerido con textos como ruby, ruby1.8, runit, rubygems, rumor, rubrica, solo por mencionar algunos.

---

## Instalación Ruby Gems

Los lenguajes de alto nivel se caracterizan hoy por hoy por tener repositorios, es decir que las aplicaciones que se realizan con ellos ,que son de dominio público, son almacenadas en contenedores de donde cualquiera los puede bajar, esto es una cosa maravillosa por que al igual que los paquetes verifican los prerequisitos para garantizar estabilidad y desempeño, pero también cada lenguaje tiene su propio manejador de paquetes, en el caso de Ruby son las Gemas o Gems que son aplicaciones armadas y listas para producción que simplemente se corren, Rails es una gema y por lo tanto deberemos instalar el instalador, la página oficial es http://rubygems.org

Primero verificamos la instalación y versión de Ruby Gems, sin embargo Ruby Gems es muy distinto a lo que hemos instalado antes, tiene un frente y una aplicación de soporte, rubygems es la aplicación, pero gem es el frente, así que verificamos la versión e instalación con:

```
usuario@host:~$ gem -v
```

```
1.3.7
```

En general es conveniente tener la última versión de Ruby Gems instalado así que lo instalaremos:

```
usuario@host:~$ sudo apt-get install rubygems
```

Si no planea usar la última versión de Rails entonces la version 1.3.7 será adecuada, sin embargo de no ser así actualice con:

```
usuario@host:~$ sudo gem install rubygems-update
usuario@host:~$ sudo /var/lib/gems/1.8/bin/update_rubygems
```

Desafortunadamente no existe un solo método para actualización de Ruby Gems ya que esto esta en función del sistema operativo, el Ruby Gems actual, los repositorios etc. Es posible hacerlo a través de:

```
usuario@host:~$ sudo gem update --system
```

O bajando el código y compilándolo

```
usuario@host:~$ wget http://production.cf.rubygems.org/rubygems/rubygems-1.8.5.tgz
usuario@host:~$ tar xzf rubygems-1.8.5.tgz
usuario@host:~$ cd rubygems-1.8.5
usuario@host:~$ sudo ruby setup.rb
```

Para ver el listado de las gemas instaladas y sus versiones invocamos el comando:

```
usuario@host:~$ gem list
```

Si tenemos gemas instaladas y queremos verificar el servidor de gemas lo podemos invocar con:

```
usuario@host:~$ gem server
```

```
Server started at http://[::ffff:0.0.0.0]:8808
Server started at http://0.0.0.0:8808
```

Ahora podemos ir a cualquier navegador de internet e ingresar la dirección en la casilla del URL http://localhost:8808 ahí podremos tener un listado de las gemas instaladas y siempre podemos revisar que nuestras gemas esten correctas aqui mismo. Cuidado, el servidor de gemas enviara un error "/var/llib/gems/1.8 does not appear to be a gem repository" si no hay gemas instaladas.

La instalación del Ruby Gems es por mucho el paso mas importante dentro de la instalación de Rails, en ocasiones las aplicaciones en versiones mas antiguas de Rails manifiesten precauciones por el uso ya obsoleto de métodos o funciones. Si su aplicación en particular no corre como espera o no corre del todo, intente instalando una versión anterior de Ruby Gems con el comando:

```
usuario@host:~$ sudo gem update --system 1.6.2
```

Donde 1.6.2 es la versión a la cual se quiere retornar, es muy probable que necesite repetir la instalación de las gemas en esta versión, sin embargo en muchos casos funcionará sin preámbulos.

---

Existe una instrucción que nos permitirá correr aplicaciones antiguas de Rails en Ruby Gems mas modernos sin ningún cambio aunque nos enviara mensajes de precaución. requiere 'thread' Debe ser insertado en el archivo config/boot.rb inmediatamente abajo de RAILS_ROOT

---

## Instalación Rails

Rails es el tema de este texto, así que no tiene mucho sentido definirlo una vez mas, la página oficial es http://rubyonrails.org/, si llegó aquí es porque eligió el camino largo y difícil y asumo que no tiene instalada la versión de Rails por repositorios de OS, verifíquelo de la siguiente forma:

```
usuario@host:~$ rails -v
```

```
Rails 2.3.8
```

Si no ve el mensaje anterior sino algún tipo de error de no poder encontrar rails, entonces no hay problema..

Para instalar una versión anterior basta con colocar la versión con el modificador -v= esto instruye a Ruby Gems que versión es la que necesita, es posible tener múltiples versiones instaladas pero solo la última servirá para desarrollo desde ○, aunque todas las demás darán soporte a las aplicaciones en sus respectivas versiones.

Para instalar la versión mas actual corra:

```
usuario@host:~$ sudo gem install rails
```

O si lo el lector lo prefiere la versión para este ensayo

```
usuario@host:~$ sudo gem install rails -v=2.3.8
```

Cabe mencionar una vez mas que si se está instalando un servidor de aplicaciones que no se usará para desarrollo entonces es muy conveniente tener todo el conjunto de versiones, así que es muy conveniente instalar ambas.

Después de unos minutos de mensajes de instalación, crear documentación y configuración tendrá la versión mas actualizada de Rails y una versión anterior que nos servirá para este ensayo, sin embargo al correr el comando rails -v da un error de no poder encontrar Rails entonces necesitamos la siguiente instrucción:

```
usuario@host:~$ export PATH=$PATH:/var/lib/gems/1.8/bin/
```

Y también habrá que agregarla al final del archivo .bashrc

```
usuario@host:~$ cd
usuario@host:~$ nano .bashrc
```

Agregar al final del archivo la línea `export PATH=$PATH:/var/lib/gems/1.8/bin/` y grabar. Verifique la versión con:

```
usuario@host:~$ rails -v
```

```
Rails 3.0.9
```

O quizás 2.3.8 en función de la elección de versión

Ruby Gems versión 1.3 en adelante instalará automáticamente las gemas en dependencia de la gema actual, así que como Rails requiere de la gema Rake esta es automáticamente instalada, lo verificamos con:

```
usuario@host:~$ rake --version
```

```
rake, version 0.9.2
```

Si todo salió bien, en este momento tenemos Rails perfectamente instalado.

Si queremos ver las funciones que rake puede realizar entonces podemos invocarlo con rake -T pero es necesario crear un ambiente rails o tener los archivos rakefile, Rakefile, rakefile.rb o Rakefile.rb, el procedimiento es el siguiente:

```
usuario@host:~$ mkdir rails
usuario@host:~$ cd rails
```

Ahora invocamos el generador de aplicaciones:

```
usuario@host:~/rails$ rails prueba1
usuario@host:~/rails$ cd prueba1
usuario@host:~/rails/prueba1$ rake -T
```

Ahora solo nos falta una cosa más ... la base de datos.

## Instalación SQLite 3 (recomendado para este ensayo)

SQLite 3 es una excelente base de datos relacional, es muy rápida, muy confiable, tiene muy poco peso, y de alguna forma es ubicua, ya que la instalación completa ocupa menos de 300k y es común que se utilice en handhelds, teléfonos celulares, sistemas empotrados y por supuesto computadoras, pero quizás dentro de sus mayores atractivos esta su licencia que dice algo así:

HARÁS EL BIEN Y NO EL MAL.
ENCONTRARÁS PERDÓN PARA TI Y PARA LOS OTROS.
COMPARTIRÁS LIBREMENTE, NUNCA TOMANDO MÁS DE LO HAS DADO.

Rails utiliza esta base de datos como default, pudiendola cambiar en el momento de la creación o en cualquier momento en la etapa de desarrollo, la página oficial es http://www.sqlite.org/, pero como nada es perfecto tiene 2 defectos que no la hacen viable para producción a nivel teórico ... no es segura y tampoco maneja muy bien la concurrencia, es decir que cualquiera puede ver lo que en la base de datos hay y no funciona muy bien cuando hay muchos usuarios, sin embargo esto en la práctica no es ninguna limitante por que al fin y al cabo la BD está en el servidor que es inherentemente seguro y con lo de la concurrencia siempre que no sean 1000 búsquedas por segundo todo ira bien.

Verificamos instalación y versión con:

```
usuario@host:~$ sqlite3 -version
```

```
3.7.2
```

Si no ve algo como lo anterior entonces deberá instalarla con el siguiente comando y deberá volver a verificar la instalación y la versión, pero como estamos destinando SQLite para trabajar con Rails de una vez instalamos la biblioteca de conexión y la de desarrollo para que la gema pueda encontrar donde conectarse.

```
usuario@host:~$ sudo apt-get install sqlite3
usuario@host:~$ sudo apt-get install libsqlite3-ruby
usuario@host:~$ sudo apt-get install libsqlite3-dev
```

Y por último la gema para que Rails pueda acceder a SQLite 3

```
usuario@host:~$ sudo gem install sqlite3
```

## Instalación MySQL

MySQL es por llamarle de alguna forma el estándar de las bases de datos para web, sin embargo esa definición se queda muy corta por que en realidad MySQL se utiliza en cada ámbito donde cualquier otra base de datos ha también estado, es muy confiable, muy rápida y muy segura, esta diseñada para producción a nivel empresarial y tiene una licencia muy económica para producción y gratuita para aprendizaje o para aplicaciones sin fines de lucro, la página oficial es http://www.mysql.com/. Cabe mencionar que recientemente la adquirió Oracle, eso significa que no es una cosa que hay pasar a la ligera.

Para verificar su instalación y versión corremos

```
usuario@host:~$ mysql --version
```

```
mysql  Ver 14.14 Ditrib 5.1.49, for debian-linux-gnu (i686) using readline 6.1
```

Si no vió algo como lo anterior por favor instalelo de la siguiente forma:

```
usuario@host:~$ sudo apt-get install mysql-server
```

Tome en cuenta que le pedirá en medio de la instalación la clave de superusuario (root) y la deberá ingresar 2 veces, como cualquier clave no la vaya a olvidar y contrario a las mejores prácticas apúntela en algún lugar visible, al fin y al cabo es solo para los ensayos. El comando anterior instalara tanto el servidor como el cliente, así que no es necesaria una instalación posterior para poder acceder a la base de datos.

Ahora instalamos las bibliotecas de soporte de Ruby y también las de desarrollo para que la gema sepa donde conectarse.

```
usuario@host:~$ sudo apt-get install libmysql-ruby
usuario@host:~$ sudo apt-get install libmysqld-dev
usuario@host:~$ sudo apt-get install libmysqlclient-dev
```

Una vez instalados todos los paquetes sin errores entonces podemos instalar la gema, sin embargo es posible que su versión de MySQL no corra con la gema mas actual, así que instalaremos ambas.

```
usuario@host:~$ sudo gem install mysql
usuario@host:~$ sudo gem install mysql2
```

## Instalación Mongrel (recomendado para este ensayo)

Rails al ser full stack viene con un servidor para desarrollo llamado WEBebrick, muy adecuado y rápido para pruebas en una pequeña red de desarrollo, pero definitivamente no es para producción ya que carece de un buen manejo del tráfico y en la multitud de llamadas al sistema es posible que cree una latencia significativa, en mi experiencia nunca he tenido problemas de estabilidad sino simplemente de desempeño.

Mongrel es un servidor totalmente adecuado para producción y reemplaza a WEBrick en su totalidad, al instalarlo automáticamente se iniciará junto con la aplicación a menos que se llame por nombre a WEBrick. Y es una excelente práctica instalarlo en desde el inicio ya que es excelente para producción como para desarrollo.

Para instalarlo correremos:

```
usuario@host:~$ sudo gem install mongrel
```

Para probarlo bastara con iniciar una aplicación y deberá aparecer algo como esto:

```
=> Booting Mongrel (use 'script/server webrick' to force WEBrick)
=> Rails 2.3.8 application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
```

## Instalación Phusion Passenger Standalone
**(no recomendado, solo para producción)**

Passenger es el servidor mas avanzado para aplicaciones Rails, con muy buen desempeño y excelente manejo de errores y recuperación de fallas, solo es superado por el Apache mismo, adecuado cuando las aplicaciones requieren de uploads masivos de usuarios que es un escenario en el que otros servidores esperarían a que la carga terminara, Passenger genera instancias añadidas para poder atender a los usuarios posteriores.

Para instalarlo basta con:

```
usuario@host:~$ sudo gem install passenger
```

Pero algo tan bueno no se configura tan rápidamente, afortunadamente el fabricante dejo un excelente script que analiza el primer arranque y configura automáticamente todo.

Cabe decir que Passenger puede trabajar conjuntamente con Apache.

Para concluir la instalación necesitaremos estar en la raíz de una aplicación del Rails más actualizado y es por eso que donde iniciaremos Passenger no será adecuado para desarrollo por que en general inutiliza las versiones anteriores de Rails para desarrollo desde ○, pero es posible instalar múltiples versiones para soporte de tiempo de ejecución, para configurarlo y terminar de instalarlo necesitamos ingresar:

```
usuario@host:~/rails/prueba1$ passenger start
```

- Nos indicará cuales son los requerimientos del sistema y podremos elegir entre continuar la instalación o abortar.
- Es posible que pida que corramos los siguientes comandos

```
usuario@host:~$ sudo apt-get install libcurl4-openssl-dev
usuario@host:~$ sudo apt-get install libssl-dev
```

- Una vez concluida la instalación volveremos a correr passenger start y ahora esperaremos la instalación completa.
- Para detenerlo presionamos <CTRL-C>

## La prueba

### Preparación de la base de datos MySQL.

Si tiene como objetivo trabajar con MySQL entonces siga los siguientes pasos, si va a utilizar SQLite entonces pase directamente a crear la aplicación inicial de prueba.

Como MySql es una base de datos multiesquema entonces hay que crear un esquema para almacenar nuestros datos, lo hacemos de la siguiente forma, asumiendo siempre que la clave es 1234:

```
usuario@host:~$ mysql -u root -p
```

```
Enter password: 1234
Welcome to the Mysql monitor.  Commands end with ; or \g
Your MySQL connection id is 51.
Server version: 5.1.49-1debian11 (Debian)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database misdatos;
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye
```

Esencialmente todo se resume en create database misdatos; así nos aseguramos que la base de datos está levantada y activa. misdatos es ahora el nombre de la base de datos a conectarse y se deberá configurar en el database.yml, también se ingresará ahí mismo la clave, por default Rails 3 se configura con el manejador mysql2 pero podría ser necesario cambiarlo a mysql por motivos de compatibilidad, para eso ya estamos preparados, lamentablemente el tema de MySQL se sale del ámbito de este texto así que ya no lo tocaré mas.

Habrá que modificar el archivo config/database.yml, en general el bloque que nos interesará inicialmente será el development, habrá que uniformarlo con esto o algo como esto, cabe mencionar que la línea socket puede variar de distribución en distribución no digamos de OS a OS:

```
development:
  adapter: mysql
  encoding: utf8
  reconnect: false
  database: misdatos
  pool: 5
  username: root
  password: 1234
  host: localhost
  socket: /var/run/mysqld/mysqld.sock
```

## Creación de la aplicación inicial de prueba

Ya tenemos todo listo, ahora veamos si realmente esta cosa funciona como debe, crearemos un directorio de trabajo para poder alojar nuestras aplicaciones, no importa donde se coloque solo es necesario tener permisos de lectura-escritura.

```
usuario@host:~$ mkdir rails
usuario@host:~$ cd rails
```

Ahora invocamos el generador de aplicaciones:

```
usuario@host:~/rails$ rails prueba1
```

```
create
      create
      create  README
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      (se omite el resto)
```

Si lo anterior fue exitoso después de unos segundos, veremos texto fluyendo por la pantalla que nos indica que algo se ha ejecutado, podrá ver muchos "create", en este momento ya tenemos media aplicación ya programada, para verificar:

```
$ cd prueba1
```

En este momento ya tenemos media aplicación armada, todo lo que haremos estará alojado en el directorio ~/rails/prueba1, para motivos de trasiego todo el directorio podrá ser movido a otra posición, la aplicación no tiene ningún componente fuera de este directorio.

Ahora hay que darle fuego y ver que pasa.

```
usuario@host:~/rails/prueba1$ ./script/server
```

```
=> Booting Mongrel
=> Rails 2.3.8 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
```

Esto ejecutará el servidor en la maquina local publicando una aplicación en el IP 0.0.0.0 puerto 3000, es decir que en nuestro navegador introduciremos el URL, y deberemos ver una página similar a la ilustración.

http://0.0.0.0:3000


En muchos casos querremos publicar nuestra aplicación en la LAN para pruebas o demostraciones, el ip 0.0.0.0 es exclusivo de nuestra máquina, entonces utilizaremos el modificador -b

```
$ ./script/server -b 192.168.39.122 -p 3008
```

Donde 192.168.39.122 es el ip de nuestra maquina y 3008 el puerto, de esa forma los clientes de la LAN podrán acceder a nuestra aplicación con el URL: http://192.168.39.122:3008

La ayuda se invoca con:

```
$ ./script/server --help
```

```
Usage: rails server [mongrel, thin, etc] [options]
    -p, --port=port                  Runs Rails on the specified port.
                                     Default: 3000
    -b, --binding=ip                 Binds Rails to the specified ip.
                                     Default: 0.0.0.0
     -c, --config=file                   Use custom rackup configuration
file
    -d, --daemon                     Make server run as a Daemon.
     -u, --debugger                       Enable ruby-debugging for the
server.
     -e, --environment=name             Specifies the environment to run
this server under (test/development/production).
                                     Default: development
    -P, --pid=pid                    Specifies the PID file.
                                     Default: tmp/pids/server.pid

    -h, --help                       Show this help message.
```

Los ambientes de ejecución son Desarrollo, Prueba y Producción, siendo el de desarrollo el estándar, cada uno de ellos accede a una base de datos distinta y varía la modalidad de presentación de advertencias y errores.

Y para detenerlo presionamos <Control-C>

# Capítulo 3: Análisis

Ahora crearemos nuestra aplicación de introducción, en este caso será un pequeño foro, para que estemos claros, un foro tiene usuarios que crean hilos y cada hilo tiene comentarios, por lo tanto al llevarlo a 3era. forma normal los datos nos quedan de la siguiente forma:

Descripción de los datos para la aplicación "Foro"

| Hilos | | Comentarios | |
|---|---|---|---|
| texto varchar(100) | Texto del encabezado del hilo. | usuario_id integer | Llave foránea a la tabla usuarios. |
| usuario_id integer | Llave foránea a la tabla usuario | hilo_id integer | Llave foránea a la tabla Hilos. |
| estado integer | Activo, Inactivo, Invisible, de forma que inactivo solo permite leer, pero no agregar comentarios. | texto varchar(400) | Texto del comentario. |
| | | estado integer | Modificable o lectura solamente. |
| **Usuarios** | | | |
| nombre varchar(12) | nombre único del usuario, llave candidata | | |
| clave varchar(20) | Clave encriptada del usuario (por favor omítase en este momento, nos servirá para ejemplo posteriormente) | | |

La convención sobre configuración difiere del análisis tradicional ya que al ser una herramienta de ultima generación no es necesario definir llaves primarias, ni fechas de creación o fechas de actualización, esos datos son automáticos, adicionalmente tome nota que los campos que terminen con _id se asumirán llaves foráneas a las tablas que tengan como prefijos

Añadido a lo anterior, si la entidad es Usuario, entonces la tabla se creará automáticamente con el nombre Usuarios, como es una pluralización en sajón, no necesariamente es la más adecuada, y por lo tanto habra que cambiar el archivo de locales para que se la correcta en castellano.

En esta fase, no espere ver DDL, tampoco espere ver relaciones o tablas, no habrá esquemas, ni invocaciones al cliente de la base de datos, Rails se encargará de tomar lo que sabemos y convertirlo en lo que queremos.

## Capítulo 4: Elaboración

Aunque en la fase de desarrollo formal no se espera que se utilicen generadores, estos están aquí por una y solo una razón, "eliminar un montón de trabajo", y por lo tanto una vez se comprende lo que en realidad hacen, son una excelente herramienta para iniciar, los siguientes comandos corren igualmente en ~/rails/prueba1

```
$ rails g scaffold hilo texto:text usuario_id:integer estado:integer

$ rails g scaffold comentario usuario_id:integer hilo_id:integer texto:text estado:integer

$ rails g scaffold usuario nombre:string
```

Como podrá darse cuenta no tienen un orden especifico, los usuarios son necesarios para los comentarios y los hilos, sin embargo fueron creados al último. Ahora la instrucción para asentar la migración, que es simplemente crear las tablas en la base de datos:

```
$ rake db:migrate
```

---

Si obtiene un mensaje del tipo **rake aborted! no such file to load -- sqlite3** por favor corra **sudo apt-get install libsqlite3-ruby1.8**

---

Y listo, tenemos una aplicación perfectamente funcional aunque aun dista un poco de ser útil.

Iniciamos el servidor.

```
$ rails s
```

Y coloquemos en el URL del visualizador

**http://0.0.0.0:3000/usuarios**

Podremos ver un ABCD de la tabla usuario, simplista pero al menos completo, en realidad tenemos 3 módulos funcionales, de ahora en adelante les llamaré controladores:

**http://0.0.0.0:3000/usuarios**
**http://0.0.0.0:3000/comentarios**
**http://0.0.0.0:3000/hilos**

Si cometemos un error en un scaffold basta con invocar el script destroy, es decir:

```
$ rails destroy scaffold usuario
```

Y tendremos una reversión muy buena, pero habrá que ir a la BD y administrarla ya que la tabla seguirá existiendo. Vea el capítulo 14 para invocar la consola de la base de datos.

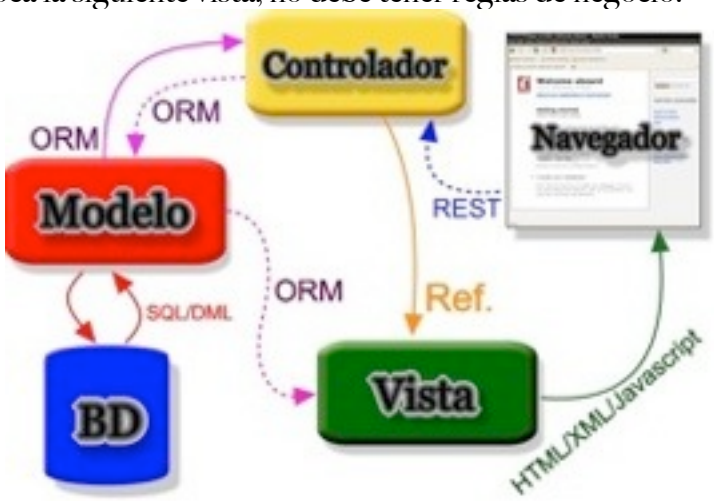
## Capítulo 5: MVC

El patrón de diseño MVC, significa que todo se distribuirá en Modelos, Vistas y Controladores, cada elemento tiene una muy bien definida función.

En los modelos está el código que interactúa con la base de datos, se encarga de las reglas del negocio, relaciones entre tablas, métodos y atributos virtuales entre otras cosas.

La vista es un generador de código visual aunque en Rails el único código visual que genera es HTML y Javascript, y consecuentemente AJAX. La vista puede acceder al modelo y presentar cualquier tipo de información pero no debe, en general, hacer cálculos.

El controlador se encarga de la interacción entre el modelo y la vista, en general recopila los datos, los procesa e invoca la siguiente vista, no debe tener reglas de negocio.




El flujo normal inicia en el controlador inicial, que extrae datos del modelo, y los presenta en una vista, una vez esta vista retorna la petición el controlador redirecciona esos datos al modelo donde son aceptados o no, el controlador actúa en función de esa aceptación o rechazo para volver a hacer otra petición al modelo y redireccionar el flujo a la vista correspondiente, este ciclo se repetirá indefinidamente.

El controlador inicial será el que aparece en el archivo de rutas, por el momento nos tendremos que conformar con saber que el controlador con el que interactuamos se extrae del URL, en la forma:


El controlador recibe en este caso como parámetros el Id y la Acción para que pueda saber como actuar y que vista presentar, dentro de sí el controlador tiene un método con el nombre de la acción y cada acción tiene relacionada una vista con el mismo nombre. El capítulo 12 habla de las rutas, y el 7 de los controladores y las vistas.

# Capítulo 6: El esqueleto

Cuando se invoca el comando rails, este genera un conjunto de directorios y archivos, a esto se le denomina el esqueleto de la aplicación:

```
Gemfile
app
doc
script
Gemfile.lock
config
lib
test
README
config.ru
log
tmp
Rakefile
db
public
vendor
```

El archivo README contiene una muy buena descripción de lo que contienen y cual es su función, en general toda la acción estará en el directorio "app" ya que dentro de sí mismo están los controladores, modelos y vistas, para cada tabla se ha creado un controlador y un modelo, y un directorio con 4 vistas, cada una de ellas para los eventos index, edit, new, show. Se usará usuario como ejemplo:

- Vista *index*, presenta una matriz con el contenido de la tabla.
- Vista *edit*, presenta una forma para modificar los datos de la tupla.
- Vista *new*, presenta una forma para ingresar una tupla nueva.
- Vista *show*, presenta los datos de una sola tupla.

Como ejemplo, vea el directorio ./app/views/usuarios/

El controlador contiene 7 acciones, index, show, new, edit, create, update, y destroy.

- Acción *index*, extrae todas las tuplas y las envía a la vista *index*
- Acción *show*, extrae una tupla y la envía a la vista *show*
- Acción *new*, genera una tupla en blanco y la envía a la vista *new*
- Acción *edit*, extrae una tupla y la envía a la vista *edit*
- Acción *create*, toma la respuesta de la vista *new*, graba, y presenta la vista *show*, si hay error invoca la vista *new* otra vez.
- Acción *update*, toma la respuesta de la vista *edit*, graba y presenta la vista show. si hay error invoca la vista *edit* otra vez.
- Acción *destroy*, toma la respuesta la vista que lo invoca, borra la tupla e invoca la vista *index*.

Como ejemplo vea el archivo ./app/controllers/usuarios_controller.rb

Curiosamente el modelo no contiene nada útil, mas que la definición de él mismo, sin embargo será en el donde la magia se creará. Vea el archivo ./app/models/usuario.rb

El directorio "config", contiene 3 archivos importantes, el environment.rb, routes.rb y database.yml, como hemos hecho nuestro mejor esfuerzo por configurar lo menos posible estos archivos permanecerán en la incógnita por el momento, sin embargo, presentan los puntos de configuración de Rails, cada uno tendrá su propio capítulo mas adelante.

El directorio "db" contiene en nuestro caso la base de datos, es decir que en la configuración por defecto, el archivo development.sqlite3 ¡es nuestra base de datos!, Sqlite3 es un excelente gestor de base de datos y tiene el particularidad que todo se guarda en un solo archivo, también contiene el directorio "migrate" que contiene los cambios realizados a nivel de DDL a la base de datos, también merece un capitulo aparte.

El directorio "public" es donde se almacenan los archivos expuestos al usuario final, es la raíz de nuestra aplicación y el archivo index.html es la página de bienvenidos de Ruby on Rails, también requerirá de un capitulo aparte.

# Capítulo 7: La vista index

Al usar "usuario" como ejemplo ./app/views/usuarios/index.html.erb, primero debemos revisar ./app/controllers/usuarios_controller.rb, que es su controlador:

Lo que vemos aquí, en el método index, no es mas que una petición al modelo Usuario, para que exponga todas sus tuplas, estas se almacenan en @usuarios que es una variable de instancia, es decir que transcenderá hasta la vista, la cláusula de respuesta está en función si lo que pedimos es una página html, o un archivo XML.

Index es siempre el método por defecto, es decir que se puede hacer una llamada del tipo http://0.0.0.0:3000/usuarios en donde se asumirá que se busca la acción index, se ejecutará el método y presentará la vista index.html.erb. En contraparte, una llamada del tipo http://0.0.0.0:3000/usuarios.xml instruirá a la cláusula de respuesta que no queremos un HTML sino un XML y generará el archivo pertinente.

Aquí tenemos HTML simple, donde se insertan instrucciones en Ruby a través de <% %>. el = significa print, y la h filtrar HTML.

@usuarios es la variable de instancia que creamos en el método index, el método .each significa que tome de a uno por uno los usuarios, y lo nombre como usuario, el método nombre del objeto usuario se refiere al atributo nombre de la tupla usuario.

```
./app/views/usuarios/index.html.erb
<h1>Listing usuarios</h1>

<table>
  <tr>
    <th>Nombre</th>
    <th>Hilo</th>
    <th>Texto</th>
    <th>Estado</th>
  </tr>

<% @usuarios.each do |usuario| %>
  <tr>
    <td><%=h usuario.nombre %></td>
    <td><%=h usuario.hilo_id %></td>
    <td><%=h usuario.texto %></td>
    <td><%=h usuario.estado %></td>
    <td><%= link_to 'Show', usuario %></td>
    <td><%= link_to 'Edit', edit_usuario_path
(usuario) %></td>
    <td><%= link_to 'Destroy', usuario, :confirm =>
'Are you sure?', :method => :delete %></td>
  </tr>
<% end %>
</table>

<br />

<%= link_to 'New usuario', new_usuario_path %>
```

La instrucción link_to maneja rutas simplificadas: usuario a la acción show, edit_usuario_path(usuario) ira a la accion edit con el id del usuario y new_usuario_path irá a la accion new del controlador usuario.

Destroy devolverá un flujo REST tipo DELETE de ahí que es distinta.

## Capítulo 8: La vista new y edit

Capítulo obsoleto las vistas generadas por Rails 3 utilizan parciales. Las vistas new y edit son idénticas en función a excepción de algunos cambios de forma son llamadas con http://0.0.0.0:3000/usuarios/new y http://0.0.0.0:3000/usuarios/1/edit, donde obedecen completamente a la lógica del URL.

Me resulta muy interesante ver que existe un generador de XML en blanco que solo contiene la estructura pero no los datos.

En ambos casos se genera la variable @usuario, en singular por que solo es uno, el modelo Usuario puede reservar un espacio para una nueva tupla con el método new, en contra parte el modelo Usuario y el método find pueden extraer de los parámetros enviados de la vista el Id para extraer esa tupla en particular.

Por simplicidad solo mostrare new.html.erb, edit.htm.erb es completamente equivalente.

```
./app/controllers/usuarios_controller.rb
  def new
    @usuario = Usuario.new

    respond_to do |format|
      format.html # new.html.erb
      format.xml  { render :xml => @usuario }
    end
  end

  def edit
    @usuario = Usuario.find(params[:id])
  end
```

La instrucción form_for genera una forma que presenta y recauda los datos del objecto f que fue inicializado con los datos de @usuario, lo interesante no sucede aquí, sino en el retorno, donde si la llamada fue generada por new, entonces retornara a la acción create, pero si fue generada por edit, entonces retornara a la acción update, eso lo sabe el enrutador que se configura en ./config/routes.rb, gracias al estandard REST.

El enrutador evalúa si el REST es un PUT entonces efectúa update, si es un POST entonces efectúa new, POST y PUT son flujo ocultos y a veces encriptados de datos que van de la vista al controlador para grabaciones, en contraparte de GET que es solo de consulta.

```
./app/views/usuarios/new.html.erb
<h1>New usuario</h1>

<% form_for(@usuario) do |f| %>
  <%= f.error_messages %>

  <p>
    <%= f.label :nombre %><br />
    <%= f.text_field :nombre %>
  </p>
  <p>
    <%= f.label :hilo_id %><br />
    <%= f.text_field :hilo_id %>
  </p>
  <p>
    <%= f.label :texto %><br />
    <%= f.text_area :texto %>
  </p>
  <p>
    <%= f.label :estado %><br />
    <%= f.text_field :estado %>
  </p>
  <p>
    <%= f.submit 'Create' %>
  </p>
<% end %>
<%= link_to 'Back', usuarios_path %>
```

## Capítulo 9: Las acciones create, update y destroy

Existen 4 flujos REST que genera el visualizador, GET, PUT, POST y DELETE, en el momento en que se recibe una página, también se recibe el método con que se devolverá el resultado, típicamente solo se utilizan 2, POST y GET, pero el estándar Web 2.0 requiere de los 4.

La acción create se ejecutara cuando se devuelva un REST POST, la acción update cuando se devuelva un REST PUT y la destroy cuando se devuelva un REST DELETE. Es por esa razón que el link_to de la forma index para destroy es distinta, REST GET es inofensivo así que se utiliza para index y show.

Atención a lo que hace create, extrae de lo que ha enviado lo la vista la tupla usuario, y en la misma instrucción genera una nueva tupla, despues la graba y si el resultado es satisfactorio se va a la acción show con la instruccion redirect_to (@usuario)

Update es muy similar, busca los datos temporales en el modelo con el id que le envía la vista e intenta hacer una actualización de atributos.

flash[:notice] es un mensaje que se envía al template de la aplicación.

Destroy es muy simple de entender y no tiene control de errores, busca el Id del usuario y lo borra. Todas las cláusulas respond_to contiene código para evaluar si se devolverá una vista HTML o XML no las comentaré por que no estamos interesados en este momento en el XML.

./app/controllers/usuarios_controller.rb

```
  def create
    @usuario = Usuario.new(params[:usuario])

    respond_to do |format|
      if @usuario.save
        flash[:notice] = 'Usuario was successfully created.'
        format.html { redirect_to(@usuario) }
        format.xml  { render :xml => @usuario, :status
=> :created, :location => @usuario }
      else
        format.html { render :action => "new" }
        format.xml  { render :xml =>
@usuario.errors, :status => :unprocessable_entity }
      end
    end
end

  def update
    @usuario = Usuario.find(params[:id])

    respond_to do |format|
      if @usuario.update_attributes(params[:usuario])
        flash[:notice] = 'Usuario was successfully updated.'
        format.html { redirect_to(@usuario) }
        format.xml  { head :ok }
      else
        format.html { render :action => "edit" }
        format.xml  { render :xml =>
@usuario.errors, :status => :unprocessable_entity }
      end
    end
end

  def destroy
    @usuario = Usuario.find(params[:id])
    @usuario.destroy

    respond_to do |format|
      format.html { redirect_to(usuarios_url) }
      format.xml  { head :ok }
    end
end
```

# Capitulo 10: El modelo

El modelo contiene las reglas del negocio y las relaciones entre modelos, por reglas de negocio debemos entender todas esas funciones de verificación que permiten a los datos ser o no legítimos, y por relaciones la interacción lógica de las tablas que a su vez están representadas por modelos. Para nuestra aplicación "Foro" que por cierto ya se me estaba olvidando, tenemos varias relaciones.

La tabla usuarios tiene una relación de uno a muchos con respecto a las tablas hilos y comentarios, por lo tanto debemos denotar eso en nuestro modelo agregando las cláusulas has_many, adicionalmente no es una buena idea que un usuario no tenga nombre, entonces validamos la presencia de ambos atributos.

```
./app/models/usuario.rb
class Usuario < ActiveRecord::Base
      has_many :hilos
      has_many :comentarios

      validates_presence_of :nombre
end
```

La tabla hilo tiene una relación de uno a muchos con la tabla usuarios, así que agregamos esa relación con un belongs_to, véase la singularización de usuario.

Y como un hilo puede tener muchos comentarios agregamos las instrucciones pertinentes, también no es una buena idea que alguien deje un hilo sin texto, tampoco nos conviene que hayan 2 hilos repetidos, y pensandolo bien el comentario también se debe validar para evitar que tenga textos en blanco

```
./app/models/hilo.rb
class Hilo < ActiveRecord::Base
      belongs_to :usuario
      has_many :comentarios

      validates_presence_of :texto
      validates_uniqueness_of :texto
end
```

El comentario debe hacer referencia a un usuario y a un hilo y por supuesto debe tener texto, de ahí los 2 belongs_to y el validates. Por cada has_many debe haber una contraparte belongs_to en el otro modelo.

```
./app/models/comentario.rb
class Comentario < ActiveRecord::Base
      belongs_to :usuario
      belongs_to :hilo

      validates_presence_of :texto
end
```

Por favor haga los cambios en sus archivos y reinicie el servidor, los modelos son recreados en cada inicio, este paso no es necesario cuando se hacen cambios en los controladores o las vistas.

Ahora intente crear un usuario sin nombre y se dará una grata sorpresa, el manejo de errores es virtualmente mágico en Rails.

En el capítulo 13 se verá la importancia de has_many y belongs_to.

## Tabla diferente a la esperada

Si iniciamos con Rails, no tendremos que hacer esto, pero que hay si ya existe la base de datos y los nombres de las tablas son cualquier cosa menos nuestro estandard, es por eso que tenemos que especificar el mismo al igual que las llaves foráneas, es decir que si la tabla fuera otra entonces utilizamos el comando set_table_name, similarmente tenemos las llaves foráneas que las podemos definir con el símbolo foreign_key, por funcionalidad las dejamos idénticas al default.

```
./app/models/hilo.rb
class Hilo < ActiveRecord::Base
   set_table_name "hilo"

   belongs_to :usuario, :foreign_key=>"usuario_id"
   has_many :comentarios, :foreign_key=>"comentario_id"

   validates_presence_of :texto
   validates_uniqueness_of :texto

   def initialize(attribs={})
      super(attribs)
      if !self.texto.present?
         self.texto = "Escriba aquí ..."
      end
      self
   end
end
```

## Valores por defecto

Esta es una de las acciones típicas dentro de una aplicación pero hay que tomarlo con calma. es decir que no es correcto inicializar una atributo ya inicializado, sino hay que saber controlar cuando esta variable ya contiene algo útil, esto se hace con el método present?, este método evalúa si tiene nulo y así sabemos que no ha sido asignada previamente, el signo de admiración es una negación del método. es decir si "no ha sido asignada previamente" entonces asígnele el texto "Escriba aquí ..."

self es el envío del objeto mismo para que los atributos sean modificados y super(attrib) es el que permite que los atributos de la clase sean accesibles a este método para poderlos modificar, attribs={} es el conjunto de atributos que la clase le envía al constructor initialize.

## Otras validaciones

No existe solamente validates_presence_of o validates_uniqueness_of es posible validar si un campo contiene un número, o tiene un formato en lo específico, etc. Todas estas validaciones están documentadas en http://api.rubyonrails.org/

| | |
|---|---|
| validates_acceptance_of | validates_inclusion_of |
| validates_associated | validates_length_of |
| validates_confirmation_of | validates_numericality_of |
| validates_each | validates_presence_of |
| validates_exclusion_of | validates_size_of |
| validates_format_of | validates_uniqueness_of |

## Integridad Referencial

Rails tiene una forma particular de manejar este tema, al parecer como que no es amante de la integridad referencial o quiere dejar que la base de datos la maneje, en fin, Rails es totalmente destructivo o mas bien no constructivo.

En general una integridad referencial decente debería evitar que un registro padre fuera eliminado si tiene hijos, o crear un registro hijo sin padre, la solución que Rails propone es la siguiente:

En un has_many puede haber un símbolo :dependent que puede apuntar a :destroy, :delete_all o :nullify, si es :destroy entonces todos los objetos asociados son destruidos cuando este objeto llame al método destroy, es decir cuando sea borrada la tupla padre se destruirán los hijos uno por uno, si es :delete_all similar a :destroy solo que se borrarán en grupo así que si en el modelo del hijo existen métodos especiales para cuando sean borrados, estos no se ejecutarán, y :nullify que me parece la más estúpida de las 3, dice que las llaves foráneas de los hijos tendrán nulos.

En un belongs_to, el simbolo :dependent solo puede apuntar a :destroy y :delete donde :destroy quiere decir que esta tupla se borrará si el objeto padre es borrado, y :delete idéntico a :destroy pero con la diferencia que si hay métodos asociados a la destrucción de esta tupla, no se ejecutarán.

```
./app/models/hilo.rb
class Hilo < ActiveRecord::Base
   set_table_name "hilo"

   belongs_to :usuario, :foreign_key=>"usuario_id"
   has_many :comentarios, :foreign_key=>"comentario_id", :dependent=>:destroy

   validates_presence_of :texto
   validates_uniqueness_of :texto

   def initialize(attribs={})
      super(attribs)
      if !self.texto.present?
         self.texto = "Escriba aquí ..."
      end
      self
   end
end
```

Para nuestro ejemplo agregaremos un :dependent=>:destroy a comentarios, es decir que si este hilo es borrado destruiremos automáticamente sus comentarios.

Tome en cuenta que para cada asociación del tipo hilo.usuario.nombre deberá haber un rescue, especialmente si estará utilizando :nullify en la tabla usuario.

# Capítulo 11: La plantilla

En ninguna parte de la aplicación se han declarado elementos visuales tales como distribución, coloración, tipos de letra o íconos, esta omisión es totalmente deliberada, pero al fin y al cabo es una aplicación web y se supone debe lucir agradable.

La parte visual no debe interferir con la funcional en ningún momento, de esa forma el programador esta perfectamente enfocado en su trabajo y el maquillaje se realizará muy a gusto del cliente sin necesidad de tocar una linea del código ya estable.

- Copiaremos ./app/views/layouts/usuario.html.erb a ./app/views/layouts/application.html.erb
- Eliminaremos usuarios.html.erb, hilos.html.erb y comentarios.html.erb

El propósito de la eliminación es evitar que los controladores busquen sus propias plantillas y se apeguen a la de aplicación, que es una plantilla pública en donde colocaremos un menú simple de navegación.

Habrá que cambiar el código original por este o alguno parecido, este intenta ser un ejemplo muy simplista de lo que se puede hacer.

./app/views/layouts/application.html.erb

```
<!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" xml:lang="en" lang="en">
<head>
  <title>El foro mas grande del mundo</title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>
<a href="/usuarios">Usuarios</a>  
<a href="/hilos">Hilos</a>  
<p style="color: green"><%= flash[:notice] %></p>
<%= yield %>
</body>
</html>
```

yield es donde se insertará el código de la vista, es decir que toda la vista estará embebida en ese espacio, así que tendremos un encabezado y un pie de pagina común para todas las vistas, stylesheet_link_tag apunta a ./public/stylesheets/scaffold.css y es ahí donde modificaremos el css.

El capitulo sobre el maquillaje muestra un ejemplo, sin embargo la plantilla debe ser de índole concretamente funcional, es decir que colocaremos menues, avisos, alguna distribución necesaria, etc. pero no elementos de mejora visual, ya que eso es tarea exclusiva del CSS.

Siempre deje al último la parte de mejora visual, en muchos casos se necesitará de un profesional en el área o de encontrar algún template pre-armado que nos sirva, recuerde que el cliente puede elegir las mejoras visuales arbitrariamente y por lo mismo será un esfuerzo en vano hacerlo en este momento.

# Capitulo 12: Las rutas

El URL identifica al controlador, la acción, el identificador y en algunos casos, parámetros adicionales, sin embargo no estamos restringidos a esa distribución, ya que podemos modificar las rutas en el archivo ./config/routes.rb

```
./config/routes.rb
ActionController::Routing::Routes.draw do |map|
  map.resources :usuarios

  map.resources :comentarios

  map.resources :hilos
  map.connect 'mishilos', :controller=>'hilos',:action=>'mios'

  map.root :controller => "hilos"

  map.connect ':controller/:action/:id'
  map.connect ':controller/:action/:id.:format'
end
```

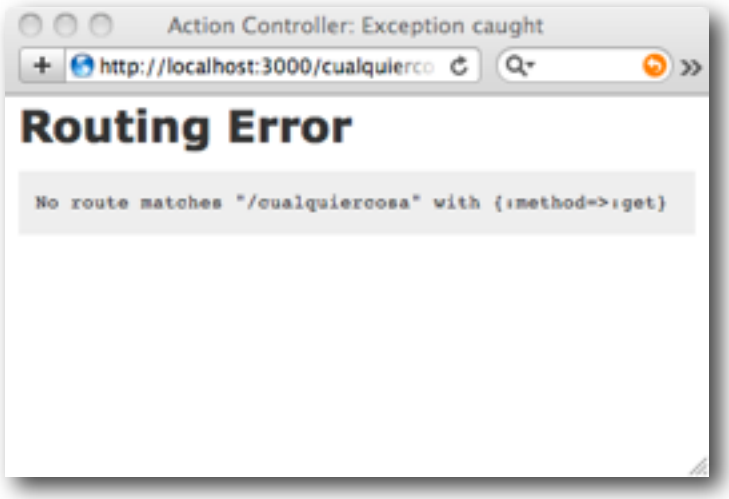
Las líneas con # han sido eliminadas por razones de espacio, estas líneas son comentarios y sirven de ejemplo para las distintas formas que puede tomar una ruta, en nuestro caso haremos 2 cambios, primero cambiaremos el controlador default con map.root apuntandolo a hilos, e inmediatamente después borrar el archivo ./public/index.html con esto logramos que Rails busque en las rutas el controlador hilos en vez de la página de bienvenida.

Las declaraciones map.resources, crean 7 rutas para index, show, new, edit, create, update y delete, para cada controlador, sin embargo si es necesario crear alguna ruta adicional, por ejemplo http://0.0.0.0:3000/mishilos, entonces simplemente declaramos un map.connect, y elegantemente aislamos al usuario del controlador y la acción a utilizar.

Es posible enmascarar cualquier tipo de acción, pero también es menester crearla en el archivo de rutas si no es una de las 7 mencionadas arriba.

En el caso de map.connect ':controller/:action/:id.:format' esto indica a Rails de que forma manejará los controladores, las acciones, el identificador y el formato, los símbolos antecedidos de 2 puntos indican la posición, ya que las rutas funcionan con prioridad descendente si se declarara otro orden arriba de esta instrucción entonces primeramente se intentaría con la ruta agregada y a posteriori con esta que es la default.

## Error de rutas




El error en las rutas es algo normal, a lo mejor un usuario queriendose pasar de listo nos coloca una dirección que no está contemplada en el ruteo, causando el error de rutas, al no ser un 404 ni nada que el usuario está acostumbrado da la sensación de descontrol y de mala calidad en la aplicación.

La solución es simple, en el archivo ./app/controllers/applicaction_controller.rb donde insertamos el método rescue_action donde exception es el error, así que redireccionaremos hacia la vista error del controlador application

./app/controllers/applicaction_controller.rb

```
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time
  protect_from_forgery # See
ActionController::RequestForgeryProtection for details

  # Scrub sensitive parameters from your log
  # filter_parameter_logging :password

  def rescue_action(exception)
    if ::ActionController::RoutingError === exception
       @exception=exception
       render :controller => 'application', :action=>"error"
    else
       super
    end
  end
end
```

El controlador application ya existe, pero curiosamente no tiene vistas, de hecho no tiene ni el direcctorio para alojarlas, aunque es posible direccionar a cualquier vista, en general podemos armar nuestros mensajes de error personalizados para cualquier cosa, asi que creamos el directorio ./app/views/application y dentro de él agregamos el archivo ./app/views/application/error.html.erb donde colocaremos un texto simple para motivos ilustrativos.

Con esto tenemos un error personalizado, por favor vease que se hace referencia

./app/views/applicaction/error.html.erb

```
Error de rutas
```

a ::ActionController::RoutingError y se compara con exception para poder determinar el tipo de error y poder actuar de forma afín, por favor vea el capítulo 17 donde se habla de otros tipos de errores.

# Capitulo 13: La consola y el ORM

Rails tiene una consola de pruebas muy efectiva para probar algoritmos, administrar datos, ejecutar metodos y dar seguimiento a variables u objetos, en realidad no es propio de Rails sino de Ruby y es la IRB que es la consola interactiva de Ruby, sin embargo al invocarla dentro de Rails, todos los modelos, helpers, bibliotecas y controladores estarán disponibles.

```
usuario@host:~/rails/prueba1$ ./script/console
```

```
Loading development environment (Rails 2.3.8)
>>
```

El ORM es el mapeo de objetos relacionales, y se encarga de ser la cara de los modelos y la relación con los otros modelos. En este momento vamos a ingresar datos por la consola, los cambios podrán ser vistos en el visualizador cuando se invoque el servidor. El >> indica lo que nosotros escribimos y => la respuesta.

```
>> Usuario
=> Usuario(id: integer, nombre: string, created_at: datetime, updated_at: datetime)
>> a=Usuario.new
>> a.nombre="Perico"
>> a.save
=> true
```

Como tenemos reglas en los modelos estas evitarán que se grabe un hilo sin texto.

```
>> Hilo
=> Hilo(id: integer, texto: text, usuario_id: integer, estado: integer, created_at:
datetime, updated_at: datetime)
>> h=Hilo.new
>> Usuario.all
=> [#<Usuario id: 1, nombre: "Perico", created_at: "2010-07-31 03:05:22", updated_at:
"2010-07-31 03:05:22">]
>> h.usuario_id=1
>> h.estado=1
>> Hilo.all
=> []
>> h.save
=> false
>> h.texto="Que le parece Ruby on Rails??"
>> h.save
=> true
```

Otro hilo más para ejemplo, y revisamos que todo esté en orden.

```
>> h=Hilo.new
>> h.usuario_id=1
>> h.estado=1
>> h.texto="El ORM es maravilloso!!"
>> h.save
=> true
>> Hilo.all.count
=> 2
```

Ahora un poco de la magia del ORM

```
>> h=Hilo.find(:first,:conditions=>"texto like '%ORM%'")
=> #<Hilo id: 2, texto: "El ORM es maravilloso!!", usuario_id: 1, estado: 1, created_at:
"2010-07-31 03:09:37", updated_at: "2010-07-31 03:09:37">
>> h.usuario
=> #<Usuario id: 1, nombre: "Perico", created_at: "2010-07-31 03:05:22", updated_at:
"2010-07-31 03:05:22">
>> h.usuario.nombre
=> "Perico"
>> u=h.usuario
=> #<Usuario id: 1, nombre: "Perico", created_at: "2010-07-31 03:05:22", updated_at:
"2010-07-31 03:05:22">
>> u.hilos.count
=> 2
>> u.hilos.collect(&:texto)
=> ["Que le parece Ruby on Rails??", "El ORM es maravilloso!!"]
```

Gracias a las instrucciones has_many y belongs_to tenemos que los modelos adoptan a otros modelos como propios, en una estructura n-dimensional donde las herencias pierden el sentido de antecesores y sucesores. De ahí que podemos hacer cosas como esta:

```
>> Hilo.first.usuario.hilos[1].texto
=> "El ORM es maravilloso!!"
>> Usuario.all[0].hilos[1].usuario.nombre
=> "Perico"
```

El ORM se encarga de manejar la lógica relacional y no tenemos nunca que preocuparnos de recuperar datos aisladamente, si el modelo esta relacionado con otro modelo, entonces sus datos puede ser recuperados por relación sin importar cuantos modelos pueda tener en el intermedio, ingresemos un comentario para el ejemplo.

```
>> c=Comentario.new
>> c.usuario_id=1
>> c.hilo_id=2
>> c.texto="Es increible lo bien que funciona"
>> c.estado=1
>> c.save
=> true
```

Ahora podemos involucrar una tabla más

```
>> comentario=Comentario.first
=> #<Comentario id: 1, usuario_id: 1, hilo_id: 2, texto: "Es increible lo bien que
funciona", estado: 1, created_at: "2010-07-31 04:02:50", updated_at: "2010-07-31
04:02:50">
>> comentario.hilo.texto
=> "El ORM es maravilloso!!"
>> comentario.hilo.usuario.nombre
=> "Perico"
```

El ORM de Rails se llama ActiveRecord y puede utilizarse por separado, pero no veo por que razón alguien quisiera hacer eso.

---

El ORM es una de las piezas mas valiosas y sólidas dentro del abanico de herramientas que ofrece Rails, sin embargo también se puede abusar de él con instrucciones del tipo: Hilo.find_by_sql("select * from usuarios u, hilos h where h.usuario_id=u.id")

---

# Capítulo 14: La consola de la base de datos y la base de datos

La consola de la base de datos es siempre la de la base de datos conectada y se invoca:

```
usuario@host:~/rails/prueba1$ rails dbconsole
```

En función de como fue creada la base de datos así es como deberemos seguir su mantenimiento, es decir que si nosotros la creamos a nivel de migraciones entonces se vuelve menester hacerlo de esa manera. Sin embargo en no pocos casos tendremos que trabajar con bases de datos ya establecidas y probablemente en producción, por lo mismo, la opción de la migración se vuelve inadecuada y queda en total potestad del DBA y su DDL el hacer los cambios al esquema.

Hay 3 ambientes: Producción, prueba y desarrollo, el ambiente de desarrollo, es el default y los otros 2 se invocarán en el momento de iniciar el servidor o modificando el archivo ./config/environment.rb agregando **ENV['RAILS_ENV'] ||= 'production'**

El archivo ./config/database.yml contiene la información pertinente de la base de datos de cada ambiente, de ahí que la consola de la base de datos puede variar.

Quizás la forma mas simple de crear una aplicación para una base de datos puntual es desde el inicio con el comando rails -d donde las opciones son mysql, oracle, postgresql, sqlite2, sqlite3, frontbase, ibm_db. Por default trabajará con sqlite3.

```
 ./config/database.yml
# SQLite version 3.x
#   gem install sqlite3-ruby (not necessary on OS X Leopard)
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000

production:
  adapter: sqlite3
  database: db/production.sqlite3
  pool: 5
  timeout: 5000
```

Al ser agnóstico a la base de datos Rails no necesariamente promueve acciones propias de la misma como la integridad referencial, procedimientos embebidos o índices compuestos, estas acciones vienen en código dentro de Rails y pueden impactar negativamente en el desempeño y si la base de datos tiene reglas que riñen con las de la aplicación entonces se puede generar un error fatal, así que es necesario que ambos, aplicación y base de datos estén muy de acuerdo.

Al parecer el desempeño es el punto débil de Rails ya que los puristas lo han considerado lento, pero hay mejoras constantes en esta área y estoy convencido que la versión actual dará resultados mas que satisfactorios a la mayoría.

## Capítulo 15: La aplicación

Le daremos funcionalidad a los hilos y lo castellanizaremos, en primer lugar como ya se vió, la vista new y la vista edit son virtualmente idénticas, por lo tanto podemos aplicar una vista parcial, donde tomamos el código común de ambas y lo pegamos en ./app/views/hilos/_edit.html.erb

```
./app/views/hilos/edit.html.erb
<h1>Modificar Hilo</h1>

<% form_for(@hilo) do |f| %>
  <%= render :partial=>"edit", :locals=>{:f=>f} %>
    <%= f.submit 'Actualizar' %>
  </p>
<% end %>

<%= link_to 'Mostrar', @hilo %> |
<%= link_to 'Retornar', hilos_path %>
```

```
./app/views/hilos/new.html.erb
<h1>Nuevo Hilo</h1>

<% form_for(@hilo) do |f| %>
  <%= render :partial=>"edit", :locals=>{:f=>f} %>
    <%= f.submit 'Crear' %>
  </p>
<% end %>

<%= link_to 'Retornar', hilos_path %>
```

```
./app/views/hilos/_edit.html.erb
<%= f.error_messages %>
<p>
  <%= f.label :texto %><br />
  <%= f.text_area :texto, :rows=>2, :cols=>40 %>
</p>
<p>
  <%= f.label :usuario_id %><br />
  <%= f.collection_select :usuario_id,Usuario.all,:id,:nombre
%>
</p>
<p>
  <%= f.label :estado %><br />
  <%= f.radio_button :estado,"1" %> Activo
  <%= f.radio_button :estado,"2" %> Inactivo
```

Y creamos el archivo parcial que tiene como prefijo un guión mayor _

La belleza de esto radica en que un cambio en el archivo _edit.html.erb afecta tanto a new como a edit.

La instrucción render visualiza la vista parcial y el símbolo locals envía a la vista parcial cualquier objeto o colección necesaria.

En el caso de f.text.area tiene un área de texto enorme que quizás deba ser modificada para que sea mas ancha y menos alta así que simplemente enviamos los parámetros adecuados.

En el caso del usuario_id colocaremos una muy conveniente selección para apoyar la integridad referencial además que luce mucho mas profesional.

También habrá que colocar estados mas legibles, y declararemos 2, activo e inactivo con un par de radio button.


Utilice siempre que le sea posible vistas parciales esto modularizará su código y evitara repetirlo, lo cual siempre le dará la certeza de donde modificar en cuanto esto se vuelva necesario.

Así como hay radio_button, existe check_box, date_select, text_area, text_field, hidden_field e inclusive file_field entre otros, estos objetos son de la clase form_for. La documentación completa del API de Rails está en http://api.rubyonrails.org/

En el caso de index se requerirá de mucho maquillaje ya que debe ser coherente con la aplicación:

```
./app/views/hilos/index.html.erb
<h1>Listado de Hilos</h1>
<table>
  <tr>
    <th>Texto</th>
    <th>Usuario</th>
    <th>Estado</th>
  </tr>
<% @hilos.each do |hilo| %>
  <tr>
    <td><%= link_to hilo.texto, hilo %></td>
    <td><%=h hilo.usuario.nombre rescue “Usuario no asignado” %></td>
    <td><%= case hilo.estado
                when 1
                        "Activo"
                when 2
                        "Inactivo"
            end
        %></td>
    <td><%= link_to image_tag("edit.png"), edit_hilo_path(hilo) %></td>
    <td><%= link_to image_tag('delete.png'), hilo, :confirm => '¿Está seguro?', :method => :delete %></td>
  </tr>
<% end %>
</table>
<h2><%= @hilos.count %> Comentarios</h2>
<br />
<%= link_to 'Crear un nuevo Hilo', new_hilo_path %>
```

En vez de tener una opción show a la derecha se modificó la instrucción para que el link sea ahora el texto del hilo, por otro lado usando el ORM se ha extraído el nombre del usuario en vez de solamente usuario_id con la instrucción hilo.usuario.nombre, la cláusula rescue es sumamente importante, ya que controla y minimiza el error colocando el texto cuando se suceda un error.

## Control de errores

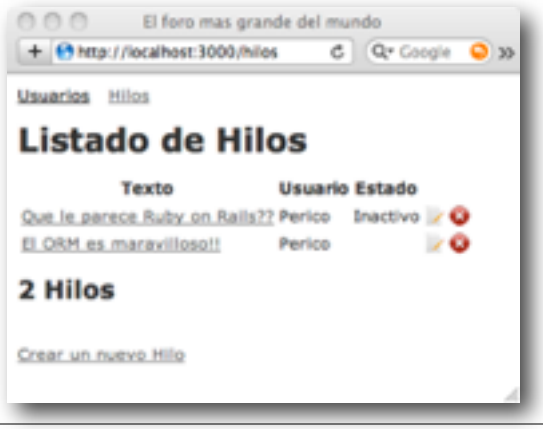
rescue después de cualquier instrucción evalúa la expresión de la derecha si se encuentra con algún tipo de error, en el caso de hilo.usuario.nombre, si hilo.usuario_id es nulo entonces el objeto hilo.usuario no es viable de ahí que Rails nos envíe un error de método no encontrado "nombre", es una forma muy elegante de enviar un mensaje de error y no lo parezca, añadido a que la aplicación podrá continuar sin mayores percances.

Por favor vea la elegancia de Ruby en la condición del estado, el valor de retorno será el string del condición verdadera.

Los link_to pueden recibir como parámetros image_tag que por default buscará las imágenes en el directorio ./public/images, estas imágenes fueron colocadas ahí manualmente. Los paths hilo y edit_hilo_path son generados automáticamente por las rutas así que no hay que preocuparse por crearlos en algún lugar.

El link_to del delete, funciona enviando los datos de hilo pero con un método REST de tipo DELETE de ahí que el controlador sabe que debe borrar el hilo que le envían.

Se colocó el número de hilos con el método count que simplemente cuenta el número filas de la colección a la que pertenece




No es una buena idea colocar el botón de borrar a la par del botón de edición, se ha dejado así por motivos ilustrativos.

Show debería mostrar el hilo que actualmente vemos, sus comentarios y a la vez permitir el ingreso de nuevos, esto implica una funcionalidad mezclada en una misma vista:

```
./app/views/hilos/index.html.erb
<h1><%=h @hilo.texto %></h1>

<table>
  <tr>
    <th>Usuario</th>
    <th>Texto</th>
  </tr>
<% @hilo.comentarios.each do |comentario| %>
  <tr>
    <td><%= link_to (comentario.usuario.nombre rescue “Sin usuario”), edit_comentario_path(comentario) %></td>
    <td><%=h comentario.texto %></td>
    <td><%= link_to image_tag('delete.png'), comentario, :confirm => '¿Está seguro?', :method => :delete %></td>
  </tr>
<% end %>

<% form_for(@hilo.comentarios.new) do |f| %>
  <%= f.error_messages %>
  <%= f.hidden_field :hilo_id %>
  <td> <%= f.collection_select :usuario_id,Usuario.all,:id,:nombre %></td>
  <td><%= f.text_area :texto, :rows=>2,:cols=>40 %></td>
</table>
<%= f.submit 'Grabar' %>
<% end %>
```

El truco para este tipo de vistas compuestas es copiar y pegar el código ya existente de las otras vistas, y solo se modifica lo que sea necesario.

Aquí aparecen todos los viejos conocidos, sin embargo hay algunos nuevos, nótese la agilidad con la que se crea la tabla donde se mezclan los comentarios y la forma para ingresar nuevos. @hilo.comentarios.each es simplemente la forma en que llamamos a los comentarios de este hilo gracias al ORM, pero nótese también otra cosa: @hilo.comentarios.new genera un comentario en blanco para este hilo, pero se hace menester el uso del hidden_field, ya que sin él, el controlador no sabrá a que hilo pertenece al momento de la grabación.

Véase el uso de rescue en la cláusula comentario.usuario.nombre, esto nos asegurará que el comentario tenga algo que mostrar aún si el usuario_id es nulo.

Aún hay un problema y luce grave, al momento de grabar nos debería mostrar el cambio y quedarse en esta misma vista, para eso necesitaremos modificar el controlador de grabación del comentario.

```
./app/controllers/comentarios_controller.rb (Solo parte)
  def create
    @comentario = Comentario.new(params[:comentario])

    respond_to do |format|
      if @comentario.save
        format.html { redirect_to :controller=>"hilos", :action=>"show", :id=>@comentario.hilo_id}
        format.xml  { render :xml => @comentario, :status => :created, :location => @comentario }
      else
        format.html { render :action => "new" }
        format.xml  { render :xml => @comentario.errors, :status => :unprocessable_entity }
      end
    end
  end
```

En el momento en que el comentario ha sido grabado exitosamente se invoca el comando redirect_to con toda la descripción del controlador y la acción, pero show requiere de un id, que extraemos fácilmente de la llave foránea del comentario hacia el hilo, @comentario.hilo_id.

Ahora tenemos un problema similar cuando borramos un comentario.

```
./app/controllers/comentarios_controller.rb (Solo parte)
  def destroy
    @comentario = Comentario.find(params[:id])
    hilo_id=@comentario.hilo_id
    @comentario.destroy

    respond_to do |format|
      format.html { redirect_to :controller=>"hilos", :action=>"show", :id=>hilo_id }
      format.xml  { head :ok }
    end
  end
```

Las circunstancias en este caso no son iguales ya que el comentario será borrado por lo tanto astutamente hemos sacado una copia de hilo_id antes que eso suceda y es ahora nuestro id para la instrucción redirect_to.

Con estos cambios se hacen innecesarias las vistas creadas por el scaffold para los comentarios, pero ya cumplieron su cometido y están ahí perfectamente disponibles y funcionales para en la etapa de desarrollo, sin embargo habrá que eliminarlas en cuanto se pase a la etapa de producción, ya que es código que simplemente no será usado, y por lo mismo no nos conviene tener una fuente de confusión a la hora del mantenimiento y soporte de la aplicación.

Ahora tendremos que castellanizar las vistas del usuario, esto asumiré que el lector puede realizarlo ya que conlleva exactamente los mismos pasos que las vistas de hilo, por favor no haga un parcial en este momento para edit y new, y modificaremos la vista show ya que será especial, nos mostrará todos los hilos y los comentarios del usuario así como también sus generales.

La vista show de usuario quedaría algo así:

```
./app/views/usuarios/show.html.erb
<h1><%=h @usuario.nombre %></h1>
Hilos creados por <%=h @usuario.nombre %>
<table>
<% @usuario.hilos.each do |hilo| %>
  <tr>
    <td><%= link_to hilo.texto, hilo %></td>
    <td><%=
            "Activo" if hilo.estado==1
            "Inactivo" if hilo.estado==2
        %></td>
  </tr>
<% end %>
</table>

<br>
Comentarios hechos por <%=h @usuario.nombre %>
<table>

<% @usuario.comentarios.each do |comentario| %>
  <tr>
    <td><%= link_to comentario.texto, comentario.hilo %></td>
  </tr>
<% end %>
</table>
```

Acá vuelven a aparecer los sospechosos de siempre, sin embargo en el link_to de los comentarios hay algo interesante, mostramos el texto del comentario, pero el link_to apunta al hilo del comentario, con comentario.hilo, es decir que el path puede también ser relacionado, aunque en realidad lo que sucede es que simplemente analiza si es uno o varios para invocar la vista show o index del modelo.

## Capítulo 16: El maquillaje

Al finalizar el desarrollo, siempre se vuelve menester hacer cambios en la visual, pero estos cambios en general no deben realizarse en las vistas es preferible un buen archivo CSS, como ya se había dicho el stylesheet que es llamado en ./app/views/layouts/application.html.erb es el ./public/stylesheets/scaffold.css, sin embargo lo cambiaremos por algo un poco mas agradable a la vista.

```
./public/stylesheets/scaffold.css
body {
       margin: 30;
       background: #7A287A;
       font-family:    "Trebuchet    MS",
Arial, Helvetica, sans-serif;
       font-size: 13px;
       color: #C752C7;
}

h1, h2, h3 {
       margin: 0;
       font-family:  Georgia,  "Times  New
Roman", Times, serif;
       font-weight: normal;
       color: #649632;
}

h1 { font-size: 44px; }

h2 { font-size: 20px; }

p, ul, ol {
       margin-top: 0;
       line-height: 240%;
       text-align: justify;
       font-family:    "Trebuchet    MS",
Arial, Helvetica, sans-serif;
       font-size: 11px;
}

a { color: #FFFFFF; }

th {
       background: #792879;
       color: #fff;
}

#errorExplanation {
  width: 400px;
  border: 2px solid #792879
  padding: 7px;
  padding-bottom: 12px;
  margin-bottom: 20px;
  background-color: #f0f0f0;
}

#errorExplanation h2 {
  text-align: left;
  font-weight: bold;
  padding: 5px 5px 5px 15px;
  font-size: 12px;
  margin: -7px;
  background-color: #792879;
  color: #fff;
}

#errorExplanation p {
  color: #333;
  margin-bottom: 0;
  padding: 5px;
}

#errorExplanation ul li {
  font-size: 12px;
  list-style: square;
}

.fieldWithErrors {
  padding: 2px;
  background-color: #ff54ff;
  color: #000;
  display: table;
}
```

No mejoró, pero es innegable que cambió, el CSS es muy potente y se pueden realizar excelentes trabajos pero eso excedería la intensión de este texto. Lo más sencillo en este caso es buscar un maquetador o conseguir un template ya armado y simplemente adaptarlo.


# Capítulo 17: Errores

Cuando se habla de errores de una aplicación es común pensar que fue mal programada, sin embargo existe una miríada de casos en los que una aplicación adecuadamente programada puede enviar un error, sin que esto signifique que algo anda mal, en capítulos anteriores hemos visto que una llave foránea nula puede llevarnos a una falla catastrófica por que simplemente el ORM no puede resolver los métodos del modelo asociado, de ahí el uso de rescue, aunque esto nos ayuda increíblemente, este es tan solo un tipo de error.

Algunos tipos de errores comunes y su manejador:

- Registro no encontrado, se maneja con ActiveRecord::RecordNotFound
- Recurso no encontrado, se maneja con ActiveResource::ResourceNotFound
- Error en plantilla (vista), se maneja con ActionView::TemplateError
- Error de rutas, se maneja con ActionController::RoutingError
- Controlador desconocido, se maneja con ActionController::UnknownController
- Método no permitido, se maneja con ActionController::MethodNotAllowed
- Autenticidad no válida, se maneja con ActionController::InvalidAuthenticityToken
- Acción desconocida, se maneja con ActionController::UnknownAction

```
./app/controllers/applicaction_controller.rb
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time
  protect_from_forgery # See ActionController::RequestForgeryProtection for details

  rescue_from ActiveRecord::RecordNotFound, :with => :no_encontrado #400
  rescue_from ActiveResource::ResourceNotFound, :with => :no_encontrado #404
  rescue_from ActionView::TemplateError, :with => :no_encontrado #500
  rescue_from ActionController::UnknownController, :with => :no_encontrado #404
  rescue_from ActionController::MethodNotAllowed, :with => :no_encontrado #405
  rescue_from ActionController::InvalidAuthenticityToken, :with => :no_encontrado #405
  rescue_from ActionController::UnknownAction, :with => :no_encontrado #501

  def rescue_action(exception)
    if ::ActionController::RoutingError === exception
         @exception=exception
         render :controller => 'application', :action=>"error"
    else
         super
    end
  end

  protected
  def no_encontrado
    render :text => "Error 404", :status => 404
  end

end
```

Aquí ya tenemos un muy decente manejo de errores, el maquillaje puede personalizarse a gusto, y deberá proveer la información del error, pero no deberá vender la idea de que algo malo sucedió, el símbolo with indica el método que manejará el error, en este caso todas son el mismo por motivos ilustrativos.

# Capítulo 18: La seguridad y la sesión

No es muy correcto que cuando escribamos un comentario escojamos el usuario, de esta forma es muy simple que otro usuario haga un comentario a nuestro nombre, lo que se requiere es un manejador de sesión que es hermano del manejador de seguridad, uno se encarga de interrogarnos sobre nuestro usuario al iniciar la sesión, y el otro se encargará de pedirnos nuestra clave, de esa forma aseguramos que nuestros comentarios solo son nuestros y facilitamos la labor de ingreso puesto que en toda la sesión siempre seremos conocidos por nuestro nombre de usuario.

```
./app/controllers/applicaction_controller.rb
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time
  protect_from_forgery # See ActionController::RequestForgeryProtection for details

  rescue_from ActiveRecord::RecordNotFound, :with => :no_encontrado #400
  rescue_from ActiveResource::ResourceNotFound, :with => :no_encontrado #404
  rescue_from ActionView::TemplateError, :with => :no_encontrado #500
  rescue_from ActionController::UnknownController, :with => :no_encontrado #404
  rescue_from ActionController::MethodNotAllowed, :with => :no_encontrado #405
  rescue_from ActionController::InvalidAuthenticityToken, :with => :no_encontrado #405
  rescue_from ActionController::UnknownAction, :with => :no_encontrado #501

  def rescue_action(exception)
    if ::ActionController::RoutingError === exception
         @exception=exception
         render :controller => 'application', :action=>"error"
    else
         super
    end
  end

  protected
  def no_encontrado
    render :text => "Error 404", :status => 404
  end

  before_filter :autenticar

  def salir
    session[:usuario]=nil
    session[:usuario_id]=nil
  end

  def autenticar
     authenticate_or_request_with_http_basic do |usuario,clave|
        recuperado=Usuario.find(:first,:conditions=>["nombre=?",usuario])
      if clave.crypt("ml")==recuperado.clave
        session[:usuario]=usuario
         session[:usuario_id]=recuperado.id
        true
      else
        false
      end
    end
  end
```

Toda aplicación para la web tiene una sección pública y una privada, la pública será para registro de nuevos usuarios, y el foro solo será visible hasta que nos autentiquemos. La función authenticate_or_request_with_http_basic provee una forma simplista aunque en muchos casos adecuada para obtener un usuario y una clave, al retornar la colección los distribuye en las variables usuario y clave, se busca al usuario, y luego se compara su clave, el método crypt, encripta la clave obtenida del usuario para que coincida con la guardada en la base de datos, véase que el parámetro debe ser igual al usado al momento de la grabación, esto se discutirá en detalle mas adelante cuando se modifique el modelo. session es muy importante por que mantiene su valor durante toda la sesión, de ahí que hacemos uso del método salir para desasignar session. El método before_filter es el primero en ser ejecutado cuando se inicia la sesión.

Pero hay un problema grave, no tenemos el atributo clave en nuestra tabla usuarios, habrá que añadirlo, simplemente invocamos la consola de la base de datos:

```
usuario@host:~/rails/prueba1$ ./script/dbconsole
```

```
SQLite version 3.6.12
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Aquí tenemos control completo de la base de datos, así que modificaremos la tabla, no lo hacemos a través de las migraciones por que eso conllevaría sacar una copia de la tabla, eliminar la tabla, modificar la migración, correr rake y recuperar los datos, de ahí que si lo hubiéramos planeado desde el principio entonces nada de esto sería necesario, por eso siempre debe haber un diseño muy bueno antes de iniciar cualquier desarrollo, por motivos didácticos hemos omitido este campo deliberadamente.

```
sqlite> alter table usuario add clave string;
sqlite> .exit
```

Ahora ya tenemos donde alojar nuestra clave, sería muy estúpido de nuestra parte guardar las claves tal como se digitan, puesto que se vuelve posible averiguar cual es la clave de un usuario a nivel de programación y a nivel de DBA, entonces las guardaremos encriptadas.

Modificaremos el modelo del usuario y usaremos un método muy común que se denomina salar, de condimentar, es decir que tenemos un algoritmo clásico de encriptamiento pero le agregamos un ingrediente secreto más, para que aunque alguien sepa el algoritmo principal no pueda averiguar la totalidad de la receta. Rails maneja esto con mucha inteligencia, el método crypt recibe un parámetro de 2 caracteres, que representa el ingrediente extra, puede ser cualquier string que solo nosotros conozcamos para que genere una salida personalizada,

```
./app/models/usuario.rb
class Usuario < ActiveRecord::Base
	has_many :hilos
	has_many :comentarios

	validates_presence_of :nombre, :clave

	def after_validation_on_create()
		self.clave=self.clave.strip.crypt("ml")
	end
end
```

after_validation_on_create correrá en cada creación, pero no correrá en la edición por que no es inteligente volver a encriptar lo que ya estaba encriptado.

Mucho ojo con el uso de strip en el after_validation_on_create este método elimina tabuladores, espacios y enters al final y al principio de la clave.

Ahora agregaremos el campo clave a la vista new de usuario que por supuesto es un password_field.

Con este cambio podremos capturar la clave al momento de crear el usuario, pero habrá que hacer una forma especializada para el cambio de la clave, ya que no podrá ser la vista edit, por que este siempre intentará modificar la clave ya existente, además es una buena idea que la persona digite 2 veces su clave para estar seguros, también es importante que la clave sea ininteligible, es decir el clásico formato de asteriscos o puntos para enmascararla y no sea descubierta por curiosos.

```
./app/views/usuarios/new.html.erb
<h1>Nuevo usuario</h1>

<% form_for(@usuario) do |f| %>
  <%= f.error_messages %>

  <p>
    <%= f.label :nombre %><br />
    <%= f.text_field :nombre %>
  </p>
  <p>
    <%= f.label :clave %><br />
    <%= f.password_field :clave %>
  </p>
  <p>
    <%= f.submit 'Grabar' %>
  </p>
<% end %>
```

# Capítulo 19: Regionalización

Capitulo obsoleto funciona en Rails 2 pero no en Rails 3. La regionalización es sumamente simple, en el archivo /config/environment.rb hay que modificar la linea de locales

/config/locales/es.yml

```
es:
  number:
    format:
      separator: ","
      delimiter: "."
      precision: 3
    currency:
      format:
        format: "%n %u"
        unit: "€"
        separator: ","
        delimiter: "."
        precision: 2
    percentage:
      format:
        delimiter: ""
    precision:
      format:
        delimiter: ""
    human:
      format:
        delimiter: ""
        precision: 1
      storage_units:
        format: "%n %u"
        units:
          byte:
            one:   "Byte"
            other: "Bytes"
          kb: "KB"
          mb: "MB"
          gb: "GB"
          tb: "TB"
  datetime:
    distance_in_words:
      half_a_minute: "medio minuto"
      less_than_x_seconds:
        one:  "menos de 1 segundo"
        other: "menos de {{count}}
segundos"
      x_seconds:
        one:  "1 segundo"
        other: "{{count}} segundos"
      less_than_x_minutes:
        one:  "menos de 1 minuto"
        other: "menos de {{count}}
minutos"
      x_minutes:
        one:  "1 minuto"
        other: "{{count}} minutos"
      about_x_hours:
        one:  "alrededor de 1 hora"
        other: "alrededor de {{count}}
horas"
      x_days:
        one:  "1 día"
        other: "{{count}} días"
      about_x_months:
        one:  "alrededor de 1 mes"
        other: "alrededor de {{count}}
meses"
      x_months:
        one:  "1 mes"
        other: "{{count}} meses"
      about_x_years:
        one:  "alrededor de 1 año"
        other: "alrededor de {{count}}
años"
      over_x_years:
        one:  "más de 1 año"
        other: "más de {{count}} años"
      almost_x_years:
        one: "casi 1 año"
        other: "casi {{count}} años"
    prompts:
      year:   "Año"
      month:  "Mes"
      day:    "Día"
      hour:   "Hora"
      minute: "Minutos"
      second: "Segundos"
  activerecord:
    errors:
      template:
        header:
          one:   "No se pudo guardar
este {{model}} porque se encontró un
error"
          other:  "No se pudo guardar
este {{model}} porque se encontraron
{{count}} errores"
        body: "Se encontraron problemas
con los siguientes campos:"
      messages:
        inclusion: "no está incluido en
la lista"
        exclusion: "está reservado"
        invalid: "no es válido"
        confirmation: "no coincide con
la confirmación"
        accepted: "debe ser aceptado"
        empty: "no puede estar vacío"
        blank: "no puede estar en
blanco"
        too_long: "es demasiado largo
({{count}} caracteres máximo)"
        too_short: "es demasiado corto
({{count}} caracteres mínimo)"
        wrong_length: "no tiene la
longitud correcta ({{count}} caracteres
exactos)"
        taken: "ya está en uso"
        not_a_number: "no es un número"
        greater_than: "debe ser mayor
que {{count}}"
        greater_than_or_equal_to: "debe
ser mayor que o igual a {{count}}"
        equal_to: "debe ser igual a
{{count}}"
        less_than: "debe ser menor que
{{count}}"
        less_than_or_equal_to: "debe
ser menor que o igual a {{count}}"
        odd: "debe ser impar"
        even: "debe ser par"
        record_invalid: "La validación
falló: {{errors}}"
      models:
      attributes:
  date:
    formats:
      default: "%e/%m/%Y"
      short: "%e de %b"
      long: "%e de %B de %Y"

    day_names: [Domingo, Lunes, Martes,
Miércoles, Jueves, Viernes, Sábado]
    abbr_day_names: [Dom, Lun, Mar,
Mie, Jue, Vie, Sab]
    month_names: [~, Enero, Febrero,
Marzo, Abril, Mayo, Junio, Julio,
Agosto, Septiembre, Octubre, Noviembre,
Diciembre]
    abbr_month_names: [~, Ene, Feb,
Mar, Abr, May, Jun, Jul, Ago, Sep, Oct,
Nov, Dic]
    order: [ :day, :month, :year ]
  time:
    formats:
      default: "%A, %e de %B de %Y %H:
%M:%S %z"
      short: "%e de %b %H:%M"
      long: "%e de %B de %Y %H:%M"
    am: "am"
    pm: "pm"
  support:
    select:
      prompt: "Por favor seleccione"
    array:
      words_connector: ", "
      two_words_connector: " y "
      last_word_connector: " y "
```

config.i18n.default_locale = :es y agregar el archivo es.yml a /config/locales, y eso es todo.

# ADENDAS

# Adenda LAMP

## Inicio y reinicio del servidor Linux

En general para reiniciar un servidor Linux podremos correr el comando

```
usuario@host:~$ sudo init 6
```

Ahora bien existe la opción de apagar el servidor completamente y es muy probable que se requiera de interacción humana para poder iniciarlo, si lo que queremos es apagar definitivamente entonces corremos:

```
usuario@host:~$ sudo init 0
```

## Terminales

Tenemos múltiples tipos de terminales y emuladores de terminal, el Linux Debian típico viene con gnome-terminal para el ambiente gráfico y tenemos las CTRL-F1 a F10 como terminales tty estandard, sin embargo existen terminales muy útiles a la hora del desarrollo como podría ser Terminator que puede dividir en multiples secciones una misma pantalla y Guake que puede aparacer y desaparecer al presionar una secuencia de teclas, para instalarlas basta con :

```
usuario@host:~$ sudo apt-get install terminator guake
```

## Compresores

En general se usarán compresores de archivos para copias de respaldo o simplemente para que algo ocupe menos espacio para su transporte o almacenamiento, el estandard Linux diríamos que es tar.gz que esencialmente la unión de 2 utilerías, tar que une varios archivos y gzip que los comprime, también debemos conocer zip que es muy popular, pero no muy eficiente, rar igualmente popular y mucho mas eficiente que zip y 7zip que es un muy buen compresor pero ciertamente no es común, existen algunos otros pero estos por ser los principales los comentaremos.

### tar.gz

```
Comprimir          tar -cvzf directorio.tar.gz directorio
Descomprmir        tar -xvzf directorio.tar.gz
Excluir            tar -cvzf directorio.tar.gz directorio --exclude directorio/excluido/*
```

### tar

```
Comprimir          tar -cvf directorio.tar directorio
Descomprmir        tar -xvf directorio.tar
Excluir            tar -cvf directorio.tar directorio --exclude directorio/excluido/*
```

### zip

```
Comprimir          zip -r directorio.zip directorio
Descomprmir        unzip directorio.zip
Excluir            zip -r directorio.zip directorio -x directorio/excluido/*
```

rar

```
Comprimir           rar a -r directorio.rar directorio
Descomprmir         rar x directorio.zip
Comprimir           rar a -r directorio.rar directorio -xdirectorio/excluido
```

7z

```
Comprimir           p7zip directorio
Descomprmir         p7zip -d directorio.7z
```

Para lograr esto necesitamos instalar las utilerías con el siguiente comando:

```
usuario@host:~$ sudo apt-get install zip rar p7zip
```

## Demonios SSH y FTP

## SSH

El demonio ssh nos permitirá acceder a la línea de comando desde una computadora remota, a través del comando ssh usuario@servidor en algunos casos nos pedirá permiso para agregar el servidor a nuestra base de conexiones confiables y posteriormente nos pedirá la clave de acceso al servidor y estaremos virtualmente frente a la máquina remota donde podremos correr la totalidad de comandos y efectuar cualquier actividad que podríamos hacer si estuviéramos físicamente frente a la computadora remota.

Con un poco de conocimiento podemos efectuar labores desatendidas o programar un robot local para que efectúe acciones remotas a través de este comando, también con un poco de hackeo del comando en sí podemos prescindir del demonio FTP.

No se debe confundir el cliente ssh con el servidor ssh, el cliente ssh es un comando ubicuo que se invoca a través de ssh y el servidor ssh es un demonio que puede ser sshd, openSSH u otros.

En conclusión el demonio ssh una de las armas mas poderosas dentro del arsenal del informático.

Para instalar el servidor y permitir que un cliente ssh se conecte con este servidor:

```
usuario@host:~$ sudo apt-get install ssh
```

Para bajar el servicio:

```
usuario@host:~$ sudo service ssh stop
```

Para subir el servicio:

```
usuario@host:~$ sudo service ssh start
```

Para reiniciar el servicio:

```
usuario@host:~$ sudo service ssh restart
```

Para enviar archivos a un servidor remoto a través del cliente ssh

```
usuario@host:~$ ssh usuario@servidor “cat > archremoto” < archlocal
```

Para enviar extraer archivos de un servidor remoto con un cliente ssh

```
usuario@host:~$ ssh usuario@servidor “cat archremoto” > archlocal
```

## FTP

El demonio FTP ofrece capacidad de transferencia de archivos entre 2 computadoras, esta transferencia es segura y de alta velocidad, existe una gran cantidad de clientes FTP los distintos sistemas operativos existentes esto nos permite tener una manera estándar de trasiego de archivos.

El cliente estándar de invoca como ftp servidor donde nos pedirá usuario y luego la clave, los comando son muy similares al bash, las acciones típicas son enviar, recibir, crear directorios, borrar y renombrar archivos entre otros.

No se debe confundir el cliente ftp con el servidor ftp, el cliente ftp es un comando ubicuo que se invoca a través de ftp y el servidor ftp es un demonio que puede ser ftpd u otros.

Para instalar el servicio FTP

```
usuario@host:~$ sudo apt-get install vsftpd
```

Para bajar el servicio:

```
usuario@host:~$ sudo service vsftpd stop
```

Para subir el servicio:

```
usuario@host:~$ sudo service vsftpd start
```

Para reiniciar el servicio:

```
usuario@host:~$ sudo service vsftpd restart
```

Para conectar un cliente ftp usamos:

```
usuario@host:~$ ftp usuario@servidor
```

Nos pedirá la clave si aplica y usaremos los siguientes comandos para navegar entre los diferentes directorios tanto los locales como los remotos:

| | |
|---|---|
| Cambiar directorio remoto | `cd dirremoto` |
| Cambiar directorio local | `lcd dirlocal` |
| Enviar archivo local al servidor | `put archivolocal` |
| Recuperar archivo del servidor | `get archivoremoto` |
| Ver contenido del directorio remoto | `ls` |
| Ver ruta remota | `pwd` |
| Ver ruta local | `lpwd` |
| Salir | `bye` |

FTP es un protocolo muy rápido de transferencia de archivos pero también es común que esté bloqueado por el firewall de ahí que el uso de SSH para transferencia de archivos de uso cotidiano.

## Sudoers

Los *sudoers* no son mas que usuarios que tienen el privilegió de poder ejecutar el comando *sudo*, y así poder efectuar labores administrativas, para poder declarar a un usuario como sudoer, debemos ser sudoers o root y ejecutar el siguiente comando:

```
usuario@host:~$ sudo visudo
```

Y simplemente agregamos una linea en cualquier parte del archivo que contenga algo así:

```
usuario   ALL=(ALL:ALL) ALL
```

Ahora el usuario puede ejecutar sudo antes de cualquier comando que requiera privilegios de superusuario y la clave que pedirá es la clave misma del usuario.

## Claves

Si un usuario necesita cambiar su clave basta con invocar el comando passwd, sin embargo en la mayoría de los casos tendremos que cambiar la clave de un usuario por que simplemente nos es desconocida.

Para esto es necesario correr:

```
usuario@host:~$ sudo passwd usuario
```

Una vez ingresada la clave 2 veces ya podemos hacer un login con este usuario y la clave recién ingresada.

# VI

vi es por definición el editor de UNIX y ahora lo es de GNU/BSD, su principal competencia por llamarle de alguna forma es emacs pero en realidad no se acerca ni por poco a la capacidad de vi para trabajar con archivos enormes y facilidad de uso, realmente era un editor muy adelantado a su época y por si fuera poco trabaja muy bien casi en cualquier condición.

Sin embargo cualquiera que haya trabajado por primera vez con vi se dará cuenta que por alguna razón los programadores como que estuvieron de acuerdo en complicar mas allá de lo ridículamente complejo, ya que vi es el rey de la criptografía de comandos y con esto me refiero a que no hay una sola labor que sea de realización intuitiva, por supuesto lo mejor es que el lector lo invoque con el siguiente comando y juegue con los comandos hasta que se sienta un tanto en confianza.

```
usuario@host:~$ vi archivodeprueba
```

La principal bondad de vi es que es ubicuo y robusto, trabaja idéntico en cualquier servidor, es muy ligero y no le importa si el archivo que va a trabajar ocupa 16 Gigas, es muy rapido para buscar y reemplazar y permite operaciones complejas de copiar y pegar, claro con su dosis de jalones de pelo de parte del novato lector.

Aqui unos comandos esenciales todos deben ir precedidos de [ESC], es decir se presiona Escape antes de introducir el comando:

| | |
|---|---|
| ***Salir*** | |
| Salir y grabar | :x |
| Salir si no hay cambios | :q |
| Salir y grabar si hay cambios | ZZ |
| Salir y no grabar | :q! |
| | |
| ***Insertar texto*** | |
| Insertar antes del cursor | i |
| Insertar antes de la línea | I |
| Añadir después del cursor | a |
| Añadir después de la línea | A |
| Añadir línea nueva después de la actual | o |
| Añadir línea nueva antes de la actual | O |
| Reemplazar un caracter | r |
| Reemplazar múltiples caracteres | R |
| | |
| ***Movimiento del cursor*** | |
| Izquierda | h |
| Abajo | j |

| | |
|---|---|
| Arriba | k |
| Derecha | l |
| Siguiente palabra | w |
| Siguiente espacio antes de palabra | W |
| Principio de la palabra | b |
| Espacio al principio de la palabra | B |
| Fin de la palabra | e |
| Espacio al final de la palabra | E |
| Una oración atrás | ( |
| Una oración adelante | ) |
| Un párrafo atrás | { |
| Un párrafo adelante | } |
| Principio de la línea | 0 (cero) |
| Final de la línea | $ |
| Primera línea del archivo | 1G |
| Última línea del archivo | G |
| n.ava línea del archivo | nG |
| n.ava línea del archivo | :n |
| Moverse a c | fc |
| Regresar a c | Fc |
| Principio de la pantalla | H |
| Mitad de la pantalla | M |
| Final de la pantalla | L |
| Mover en función de (),{}.[] | % |
| | |
| ***Borrar texto*** | |
| Borrar un caracter a la derecha del cursor | x |
| Borrar un caracter a la izquierda del cursor | X |
| Borrar hasta el final de la línea | D |
| Borrar la línea actual | dd |
| Borrar la línea actual | :d |
| Borrar n lineas | ndd |
| | |
| ***Copiar texto*** | |
| Copiar línea actual | yy |
| Copiar línea actual | :y |
| Copiar n líneas | nyy |
| | |
| ***Cambiar texto*** | |
| Cambiar hasta el final de la línea | C |
| Cambiar la línea actual | cc |
| | |
| ***Pegar texto*** | |
| Pegar texto después de la línea actual | p |
| Pegar texto antes de la línea actual | P |
| | |
| ***Buscar*** | |

| | |
|---|---|
| Buscar cadena hacia adelante | /cadena |
| Buscar cadena hacia atrás | ?cadena |
| Buscar siguiente hacia adelante | n |
| Buscar siguiente hacia atrás | N |
| | |
| ***Reemplazar texto*** | |
| Reemplaza el P con S de acuerdo a banderas | :s/P/S/banderas |
| Bandera reemplace todas las ocurrencias | g |
| Bandera Confirme cada reemplazo | c |
| Repita el último :s/P/S/banderas | & |
| | |
| ***Archivos*** | |
| Grabar archivo | :w |
| Insertar archivo después de la línea actual | :r |
| Siguiente archivo | :n |
| Archivo anterior | :p |
| Editar archivo | :e |
| Reemlace la línea actual con la salida del programa | !!programa |
| | |
| ***Otros*** | |
| Intercambiar entre mayúsculas y minúsculas | ~ |
| Unir líneas | J |
| Repetir el último comando cambiar texto | . |
| Deshacer el ultimo cambio | u |
| Deshacer el ultimo cambio a la línea | U |

## Adenda Apache2 (httpd)

Es una realidad que muchos servidores web deben correr aplicaciones PHP, incluyendo los que correrán Rails, de ahí que resulta mas fácil hacer que Apache 2 corra Ruby a que Mongrel o Webrick corran phpmyadmin, añadido a que, cualquier hosting que se respete, manejará las aplicaciones Rails de forma implícita y no deberemos invocar explícitamente el servicio.

Iniciamos instalando las bibliotecas adicionales de Apache 2, con:

```
$ sudo apt-get install libapache2-mod-fcgid libfcgi-ruby1.8
```

Inmediatamente después activaremos algunos módulos del Apache 2 y lo reiniciaremos:

```
$ sudo a2enmod ssl
$ sudo a2enmod rewrite
$ sudo a2enmod suexec
$ sudo a2enmod include
$ sudo service apache2 force-reload
```

Ahora modificaremos el archivo de sitios disponibles, en nuestro caso reemplazaremos el servicio en el puerto 80 y todo lo que eso implica, por favor saque una copia de respaldo, el archivo es /etc/apache2/sites-available/default

Para el ejemplo nuestra aplicación estará alojada en el directorio /var/rails/prueba1, se espera que miap.misitio.com apunte hacia el servidor.

```
/etc/apache2/sites-available/default
<Virtualhost *:80>
   ServerName miap.misitio.com
   DocumentRoot /var/rails/prueba1/public/

   <Directory /var/rails/prueba1/public>
      Options ExecCGI FollowSymLinks
      AllowOverride all
      Order allow,deny
      Allow from all
   </Directory>
</Virtualhost>
```

Sin embargo aún no tenemos /var/rails, entonces lo deberemos crear, pero también habrá que darle permisos para que Apache 2 puedo acceder a él, de ahí que usaremos el usuario www-data y el grupo www-data que son propios de Apache 2, pero antes reiniciaremos el servidor para que tome los sitios disponibles, todo eso se hace con los siguientes comandos:

```
usuario@host:~$ sudo service apache2 restart
usuario@host:~$ sudo mkdir /var/rails
usuario@host:~$ sudo chown -R www-data:www-data /var/rails
usuario@host:~$ cd /var/rails
usuario@host:/var/rails$ sudo su -m www-data
www-data@host:/var/rails$ rails prueba1
```

Ahora creamos los archivos .htaccess en /var/rails/prueba1/public, y dispatch.fcgi con el cuidado de darle permisos 755 a este último.

```
/public/.htaccess
AddHandler fcgid-script .fcgi
Options +FollowSymLinks +ExecCGI

RewriteEngine On

RewriteRule ^$ index.html [QSA]
RewriteRule ^([^.]+)$ $1.html [QSA]
RewriteCond %{REQUEST_FILENAME} !-f
#RewriteRule ^(.*)$ dispatch.cgi [QSA,L]
RewriteRule ^(.*)$ dispatch.fcgi [QSA,L]

ErrorDocument 500 "<h2>Aplicación con problemas</
h2>Sufrimos de dificultades, intente mas tarde"
```

```
/public/dispatch.fcgi (con permisos 755)
#!/usr/bin/ruby
require File.dirname(__FILE__) + "/../
config/environment"
require 'fcgi_handler'
RailsFCGIHandler.process!
```

Para nuestro ejemplo deberemos retornar a nuestro usuario para dar los permisos ya que www-data no es un sudoer.

```
www-data@host:/var/rails$ exit
usuario@host:/var/rails$ sudo chmod 755 prueba1/public/dispatch.fcgi
```

Ahora podemos probar en nuestro navegador con un URL del tipo http://miap.misitio.com o algo como http://192.168.0.100 y deberemos ver la pantalla de bienvenida de Rails, de ahora en adelante deberemos trabajar exclusivamente con el usuario www-data para poder desarrollar y dar mantenimiento a nuestra aplicación, el ambiente de producción se establecerá en el archivo ./config/environment.rb agregando:

```
ENV['RAILS_ENV'] ||= 'production'
```

Curiosamente no será necesario reiniciar nada, Rails hace esto automágicamente.

## Adenda Ruby

Aquí hay unos pequeños comentarios del lenguaje Ruby y algunos ejemplos típicos, Ruby es un lenguaje interpretado orientado a objetos, se supone que debería ser muy sencillo, pero al principio es un tanto críptico, es una buena idea adquirir conocimientos de Ruby en el aprendizaje de Rails, aunque este último intentará aislarnos lo más posible del lenguaje materno.

Las ventajas inherentes de los lenguajes interpretados son:

- Posibilidad de realizar directamente llamadas al sistema operativo
- Potentes operaciones sobre cadenas de caracteres y expresiones regulares
- Retroalimentación inmediata durante el proceso de desarrollo

Rápido y sencillo:

- Son innecesarias las declaraciones de variables
- Las variables no tienen tipo
- La sintaxis es simple y consistente
- La gestión de la memoria es automática

Programación orientada a objetos:

- Todo es un objeto
- Clases, herencia, métodos, ...
- Métodos tipo singleton
- Mezcla por módulos
- Iteradores y cierres

También:

- Enteros de precisión múltiple
- Modelo de procesamiento de excepciones
- Carga dinámica
- Hilos

Nuestro primer programa desde la línea de comandos

```
usuario@host:~$ ruby -e 'print "hola mundo\n"
hola mundo
```

Este mismo programa podría hacerse desde un archivo, que resulta mucho mas adecuado, usamos cat por simplicidad, pero definitivamente es mucho mas útil un editor mas formal como vi, emacs o nano:

```
usuario@host:~$ cat > hola.rb
print "hola mundo\n"
[CTRL-D]
usuario@host:~$ ruby hola.rb
hola mundo
```

Ahora el típico programa que calcula factoriales donde n factorial denotado por n! es básicamente el producto de lista de factores desde 1 hasta el valor mismo es decir: 5! es igual a 1*2*3*4*5 que es 120, se exceptúa 0 donde el factorial es 1, una función recursiva nos soluciona el problema muy elegantemente, tomaremos el valor del sistema operativo mismo:

```
usuario@host:~$ cat > fact.rb
def fact(n)
      if n == 0
            1
      else
            n * fact(n-1)
      end
end
print fact(ARGV[0].to_i), "\n"
[CTRL-D]
usuario@host:~$ ruby fact.rb 5
120
```

Ruby es un lenguaje muy robusto y preciso, invito al lector a que juegue con el programa y vea la capacidad de Ruby para manejar número tan grandes como factorial de 400 o incluso mayores.

Para poder ejecutar directamente el programa sin necesidad de llamar a ruby en cada momento debemos añadir la línea #!/usr/bin/ruby al inicio de nuestro programa así bash sabrá que debe llamar primero a ruby y luego inducirle el resto de las líneas.

Los comentarios están denotados por un # pueden ir al principio o a media línea y marcaran como comentarios todo lo que esté a la derecha hasta el fin de la línea.

Existen dos instrucciones muy importantes BEGIN y END no hay que confundirlas con principio de estructura y fin de estructura que están en minúsculas, BEGIN contiene el código que se ejecutará primero, ahí deberíamos colocar toda la construcción del ambiente y END la destrucción del ambiente, ni importa en que parte del programa estén Se ejecutan siempre en el orden BEGIN, resto del programa END, un ejemplo de todo lo anterior:

```
usuario@host:~$ cat > orden.rb
#!/usr/bin/ruby
END {
      puts “Principio “
}
      puts “Mi programa ” # puts es similar a print
      #Codigo que se ejecutará al principio
BEGIN {
      puts “Final”
}
[CTRL-D]
usuario@host:~$ chmod 777 orden.rb
usuario@host:~$ ./orden.rb
Principio Mi programa Final
```

Existe una forma de comentarios en bloque con =begin e =end, un ejemplo:

```
=begin
bloque de comentarios
bloque de comentarios
bloque de comentarios
hasta aquí
=end
```

Ruby tiene un modo interactivo llamado Interactive Ruby con su comando irb, al ejecutar:

```
usuario@host:~$ irb
```

Ahora tendremos el prompt >> que nos permitirá ingresar código ruby directamente en la consola, en el caso de los comando que requieran de un `end` para finalizar irb esperará hasta que sea concluido en comando completo para intentar ejecutarlo, para salirse se utiliza el comando `exit`.

Las variables en Ruby pueden iniciar con letras estandard y simbolos de guion mayor _ existen algunos prefijos para definir el alcance de la variable, @@ variable de clase, @ variables de instanacia, y $ variables globales, ejemplo:

```
>> a="hola mundo"
=> "hola mundo"
>> _variable=123
=> 123
```

Ruby maneja tanto cadenas como datos numéricos. Las cadenas pueden estar entre comillas dobles ("...") o comillas simples ('...').

```
>> "abc"
=> "abc"
>> 'abc'
=> "abc"
```

Las comillas simples y dobles a veces tienen efectos diferentes. Una cadena de comillas dobles permite la presencia embebida de caracteres de escape precedidos por una diagonal invertida o backslash y también la expresión de evaluación #{}. Mientras que una cadena de comillas simples no realiza esta evaluación, lo que se ve es lo que se obtiene. como por ejemplo:

```
>> print "a\nb\nc","\n"
a
b
c
=> nil
```

Ahora el mismo comando con comillas simples

```
>> print 'a\nb\nc','\n'
a\nb\nc\n=> nil
```

Podemos inducir operaciones o valores de variables en cadenas con el modificador #{}

```
>> "abcd #{5*3} efg"
=> "abcd 15 efg"
>> var="abc"
=> "abc"
>> "1234#{var}5678"
=> "1234abc5678"
```

Las expresiones matemáticas, lógicas y de asignación se denotan por:

```
=           para asignación
**          potencia
- * / %     para adición, substracción, multiplicación, división y módulo
& |         para conjunción y disyunción a nivel de bits
<< >>       corrimiento de bits a la izquierda y derecha
&& || !     para conjunción, disyunción y negación a nivel lógico
and or not  para conjunción, disyunción y negación a nivel lógico
?:          para if then else ternario
<= == >= != para menor igual, igual, mayor igual y diferente lógico
.. ...      rango incluyente y excluyente
var+=25     equivalente a var=var+25, se puede aplicar en -=, *=, **=, /=, etc..
```

Concatenamos cadenas, e inclusive podemos multiplicar cadenas con + y *

```
>> "uno"+"dos"
=> "unodos"
>> "uno"*3
=> "unounouno"
```

Extracción de caracteres, vea que los caracteres en Ruby son enteros, y aquí inician los problemas de concepto, ya que las comparaciones no son tan intuitivas:

```
>> planta="hierbabuena"
=> "hierbabuena"
>> planta[0]
=> 104
>> planta[0]=="h"
=> false
```

Asi que necesitamos extraer caracteres, para eso tenemos un conjunto de opciones, la extracción se hace a través de 2 parámetros, la posición y la cantidad, la posición si es cero o mayor se cuenta de izquiera a derecha, si es menor que cero entonces de derecha a izquierda, la cadena inicia en cero, la cantidad siempre es hacia la derecha de la posición:

```
>> planta[0,1]
=> "h"
>> planta[0,1]=="h"
=> true
```

Y ahora unos ejemplos de extracción, algunos son un tanto extraños para el novato:

```
>> planta[5,4]
=> "abue"
>> planta[-5,4]
=> "buen"
>> planta[0,5]
=> "hierb"
>> planta[0,20]
=> "hierbabuena"
>> planta[20,2]
=> nil
```

También tenemos intervalos para cadenas es decir, posición inicial y posición final inclusive ambas, de igual forma si el valor es mayor que cero se cuenta de izquierda a derecha y viceversa si es menor:

```
>> planta[0..3]
=> "hier"
>> planta[3..-5]
=> "rbab"
>> planta[0..50]
=> "hierbabuena"
```

Todo en Ruby es un objeto, una cadena al ser un objeto tiene múltiples métodos, estos métodos pueden servir para extraer la longitud, revertirla, mayusculizar, eliminar espacios etc. Unos ejemplos:

```
>> planta.length
=> 11
>> planta.reverse
=> "aneubabreih"
>> planta.upcase
=> "HIERBABUENA"
>> (“    “+planta.capitalize+”    “).strip
=> "Hierbabuena"
```

La combinación de los métodos y los intervalos pueden resolver problemas como extraer los últimos 5 caracteres de una cadena:

```
>> planta.reverse[0..4].reverse
=> "buena"
```

Si desea averiguar los métodos aplicables a un objeto en particular basta con presionar 2 veces la tecla de tabulador inmediatamente después del punto de igual forma como lo hace bash en Linux, otra manera, pero quizás menos interactiva y legible, es preguntar directamente con:

```
>> planta.methods
=> ["upcase!", "zip", "find_index", "between?", "unpack", "enum_slice", "to_f",
"minmax", "lines", "sub", "methods", "send", "replace", "empty?", "group_by", ....
```

En general los métodos que modifican la cadena misma pueden usar el posfijo ! que efectúa la persistencía del modificador, es decir:

```
>> planta
=> "hierbabuena"
>> planta.reverse!
=> "aneubabreih"
>> planta
=> "aneubabreih"
```

Por otro lado tenemos los arreglos, estos pueden ser mixtos, de tal suerte que se pueden mezclar diferentes tipos y por ende diferentes clases de objetos, y como agradable sorpresa podemos tener arreglos de arreglos y en ese momento cambia su nombre a colección, los arreglos al igual que las cadenas inician en cero:

```
>> palabras=["primera",1234,"456789"]
=> ["primera", 1234 "456789"]
>> palabras[0][0..2]
=> "pri"
>> palabras[2][0..2]
=> "456"
>> palabras[1][0..2]
TypeError: can't convert Range into Integer
      from (irb):84:in `[]'
      from (irb):84
```

Este error es por que a un entero no se le puede extraer subcadenas, sin embargo de ser necesario podemos convertir entre los diferentes tipos con los métodos to_ donde to_i es para enteros y to_s para cadenas, existen muchos otros pero para motivos de ejemplo solo usaremos estos:

```
>> palabras[1].to_s[0..2]
=> "123"
>> palabras[2].to_i+1
=> 456790
```

Para agregar elementos a un arreglo se puede usar el operador <<

```
>> palabras=["primera",1234,"456789"]
=> ["primera", 1234 "456789"]
>> palabras << “uno mas”
=> ["primera", 1234 "456789",”uno mas”]
```

Para eliminar un elemento de un arreglo se puede usar el operador -=

```
>> palabras=["primera",1234,"456789"]
=> ["primera", 1234 "456789"]
>> palabras -= [“456789”]
=> ["primera", 1234]
```

Como ejemplo de una colección tenemos:

```
>> coleccion=["Escalar 1",[1,2,3],"Escalar 3"]
=> ["Escalar 1", [1, 2, 3], "Escalar 3"]
>> coleccion[0]
=> "Escalar 1"
>> coleccion[1]
=> [1, 2, 3]
>> coleccion[1][1]
=> 2
```

Los arreglos no son algo para tomarse a la ligera, podemos aunar, unir, diferenciar, interseccionar, complementar etc.. Los operadores son:+ para aunado, | para unión, & para intersección, - para diferencia del primero y el segundo, etc.. Es posible también usar +=, -=, |=, &= para modificar el arreglo original:

```
>> a=[1,2,3,4,5]
=> [1, 2, 3, 4, 5]
>> b=[4,5,6,7,8]
=> [4, 5, 6, 7, 8]
>> a+b
=> [1, 2, 3, 4, 5, 4, 5, 6, 7, 8]
>> a|b
=> [1, 2, 3, 4, 5, 6, 7, 8]
>> a&b
=> [4, 5]
>> a-b
=> [1, 2, 3]
>> b-a
=> [6, 7, 8]
>> (a|b)-(a&b)
=> [1, 2, 3, 6, 7, 8]
```

Uno muy interesante y útil es sort:

```
>> a=[4,5,2,3,7,6,1]
=> [4, 5, 2, 3, 7, 6, 1]
>> a.sort
=> [1, 2, 3, 4, 5, 6, 7]
```

Las expresiones regulares en Ruby están basadas en Perl, de ahí que se dice que son Perlish Regexes y son muy estándares, están definidas mas o menos así, y digo mas o menos por que pueden no estar limitadas a estas:

```
[ ] Cualquier caracter o grupo que este definido aquí
[^ ] Cualquier caracter o grupo que no este definido aqui
. Cualquier caracter excepto CR o LF
\w Letra o dígito; es lo mismo que [0-9A-Za-z]
\W Ni letra, ni dígito
\s Espacio, es lo mismo que [ \t\n\r\f]
\S No espacio
\d Dígito; es lo mismo que [0-9]
\D No dígito
\b Backspace (0x08) (sólo si aparece en una especificación de rango)
\b Límite de palabra (sólo si no aparece en una especificación de rango)
\B No límite de palabra
\A inicio de una cadena
\z fin de la cadena
\Z fin de la cadena o antes de una CR o LF al final
\/ diagonal
* Cero o más repeticiones de lo que precede
*?  Cero o más repeticiones de lo que precede sin cancelar búsqueda
+ Una o más repeticiones de lo que precede
+? Una o más repeticiones de lo que precede sin cancelar búsqueda
? Cero o una repeticiones de lo que precede
{m,n} Al menos m y como máximo n de lo que precede
{m,n}? Al menos m y como máximo n de lo que precede sin cancelar la búsqueda
? Al menos una repetición de lo que precede; es lo mismo que [0,1]
| Puede coincidir con lo que precede o con lo que sigue
() Agrupamiento
^ Inicio de la expresión
$ Final de la expresión
```

Un pequeño ejemplo de como utilizarlas:

```
>> expresion= /[pg]ato|perro/
=> /[pg]ato|perro/
>> "concuerda" if expresion.match("gato")
=> "concuerda"
>> "concuerda" if expresion.match("pato")
=> "concuerda"
>> "concuerda" if expresion.match("chucho")
=> nil
```

Los hashes en Ruby funcionan de igual forma que en otros lenguajes que los soportan, tenemos una llave y un valor, donde ambos están relacionados, deben verse como pequeños diccionarios que nos simplifican la codificación de elementos. Por ejemplo la computadora entiende del número 12 pero el humano entiende del mes de diciembre, el humano entiende de masculino y femenino, pero la maquina solo entiende de 1 y 2, de ahí que ayudan a tablas de una posible base de datos simplificando toda la programación, un ejemplo:

```
>> mihash={"Autorizado"=>1,"Impreso"=>2,"Anulado"=>3}
=> {"Anulado"=>3, "Autorizado"=>1, "Impreso"=>2}
>> mihash[“Autorizado”]
=> 1
>> mihash.length
=> 3
>> mihash.index(1).upcase
=> "AUTORIZADO"
```

Hay un tipo de algo que parece variable, ciertamente no es un objeto y al final lo dejaremos como un simple símbolo, es algo que marca o identifica, pero solo se puede consultar y asignar pero nada más, es decir que no se aplica para expresiones, muy útil para usarlo con los hashes de la siguiente forma:

```
>> h=Hash.new
=> {}
>> h[:nombre]="Juana"
=> "Juana"
>> h[:edad]=20
=> 20
>> h[:direccion]
=> nil
```

Son muy útiles por que podemos simplemente llenarlos y llamarlos por su nombre, sin que esto implique asignación de variables adicionales, solo son banderas que identifican sus contenidos.

Ruby al ser un lenguaje estructurado orientado a objetos posee todas las virtudes de cualquiera lenguaje de tercera generación, sin embargo también es un lenguaje dulcificado y eso implica que algunos de sus elementos son opcionales o intercambiables y en general nos perdonará un buen número de errores de sintaxis.

La forma de las decisiones es estructurada pero podría tomar una forma lineal sí así se vuelve mas legible con el inconveniente de perder la capacidad de realizar un else:

```
a=25
if  a==30
      print  “Es treinta”
else
      print “No es treinta”
end
print “Es veinticinco” if a==25
```

Los operadores de comparación son <=, ==, >= !=, para menor igual, igual, mayor igual y diferente, se puede usar && y and para una conjunción y || u or para una disyunción la negación pude hacerse con el prefijo ! o con el operador not, una expresión típica podría ser:

```
if a>=3 and !(b<4 or c>5) and r!=0
```

Existen métodos que tiene el posfijo ? como nil? o blank?, estos devuelven verdadero o falso en función del resultado, nil? pregunta si la clase es nula, y devuelve verdadero si lo es y falso sino, blank? si está en blanco.

Por supuesto también tenemos al siempre bienvenido operador ternario ? :

```
b = ( var==30 ? 34 : 35 )
```

Donde b se volverá 34 si var es 30, sino entonces 35

Tenemos el operador unless ( a menos que ) que simplemente evalúa la negación la expresión, la sintaxis es similar a la del if:

```
print “A tiempo!” unless hora > 3
```

Esta sería equivalente a un if (si condicional) si la expresión estuviera negada o en su forma lógica inversa.

```
print “A tiempo!” if hora <= 3
```

En el caso de case que es una especie de condición estructurada muy popular entre los lenguajes de tercera generación tenemos un ejemplo:

```
>> i=8
=> 8
>> case i
>>     when 1, 2..5
>>          print "Está entre 1 y 5\n"
>>     when 6..10
>>          print "Está entre 6 y 10\n"
>>     else
>>          print "Está fuera de rango\n"
>> end
Está entre 6 y 10
```

Case también tiene una forma lineal muy interesante, en donde asignara a la variable bis el valor de la última expresión ejecutada en este caso un true:

```
>> a = 2000
=> 2000
>> bis = case
>>          when a % 400 == 0
>>          when a % 100 == 0
>>          else a % 4 == 0
>>        end
>> puts “Es bisiesto” if bis
Es bisiesto
```

While ( mientras) es una estructura bien conocida que hará ciclos mientras la condición se cumpla:

```
>> var = 0
>> while var < 10
>>   puts var.to_s
>>   var += 1
>> end
```

Y por supuesto también existe la forma lineal de while:

```
>> i=0
=> 0
>> print "#{i+=1}\n" while i < 3
1
2
3
```

Until ( hasta ) es idéntico a while a excepción que evaluará la negación de la expresión o el complemento lógico del mismo:

```
>> var = 0
>> until var >= 10
>>   puts var.to_s
>>   var += 1
>> end
```

Until también tiene su forma líneal

```
>> i=0
=> 0
>> print "#{i+=1}\n" until i >= 3
1
2
3
```

Tanto para while y until existe una variación muy interesante que es la evaluación al final de la expresión, y el rompimiento de la estructura a través de los comandos break, retry y redo, break finalizará el ciclo inmediatamente, retry ira a la primera línea de la estructura y volverá a ejecutar todas las instrucciones, y redo ira a la evaluación misma de la estructura para validar su entrada al ciclo o no:

```
>> var = 0
>> begin
>>   puts var.to_s
>>   var += 1
>>   break if var == 9
>> end until var >= 10
```

y para while:

```
>> var = 0
>> begin
>>   puts var.to_s
>>   var += 1
>>   break if var == 9
>> end while var < 10
```

For ( para ) tiene un significado un tanto distinto en Ruby, al igual que algunos lenguajes orientados a objetos, for puede manejar intervalos, arreglos, colecciones y hashes, y existe una forma que se le llama de iterador que simplemente pasa el valor de la variable a un bloque para que pueda ser trabajada, unos ejemplos simples:

```
>> for j in 4..7
>>    puts j
>> end
4
5
6
7
```

Para arreglos

```
>> for j in [“hola”,”mundo”]
>>    puts j
>> end
hola
mundo
```

En el caso del iterador tendríamos algo como

```
>> (4..7).each { |j|
>>    puts j
>> }
4
5
6
7
```

Para arreglos

```
>> [“hola”,”mundo”].each { |j|
>>    puts j
>> }
hola
mundo
```

Se debe entender el operador |j| como un puente que comunica la expresión exterior con la interior utilizando la variable j como conexión. en todos los casos se pueden aplicar las instrucciones break, retry y redo.

Hay una variación del ciclo for gracias a algunos métodos propios de los enteros como times o upto:

```
>> 3.times do |j|
>>    puts “hola ”+j.to_s
>> end
hola 0
hola 1
hola 2

>> 100.upto(200) do |j|
>>    puts “hola ”+j.to_s
>> end
hola 100
hola 101
...
hola 200
```

Aqui vemos el tipo mas simple de estructura la do ... end, en realidad casi todo se puede englobar dentro de estas dos instrucciones y es posible aplicarle retry, redo y break.

Las clases se definen con las instrucción class, e inmediatamente ya se pueden declarar los métodos, los atributos se denotan por variables de instancia con un prefijo @ y no es necesario declararlos solo usarlos, siguiendo las mejores prácticas no es posible acceder a un atributo directamente sino solamente a través de métodos, un ejemplo

```
class Mascota
  attr_accessor :precio
  attr_accessor :tipo
  attr_accessor :edad
end

class Perro<Mascota
  def initialize(nombre="No tiene")
    @color="Indefinido"
    @nombre=nombre
  end
  def color=(color)
    @color=color
  end
  def color
    @color
  end
  attr_accessor :nombre
  def ladrar
        "Guau!"
  end
end

chucho1=Perro.new
puts "Ladrido de chucho1 "+chucho1.ladrar+"\n"
puts "Color de chucho1 "+chucho1.color+"\n"
chucho1.color="Cafe"
puts "Color de chucho1 "+chucho1.color+"\n"
chucho1.nombre="El Pulgas"
puts "Nombre de chucho1 "+chucho1.nombre+"\n"

chucho2=Perro.new("Manchas")
puts "Nombre de chucho2 "+chucho2.nombre+"\n"
chucho2.tipo="Can"
puts "Tipo de chucho2 "+chucho2.tipo+"\n"
```

Grabelo y ejecutelo para ver algo como:

```
Ladrido de chucho1 Guau!
Color de chucho1 Indefinido
Color de chucho1 Cafe
Nombre de chucho1 El Pulgas
Nombre de chucho2 Manchas
Tipo de chucho2 Can
```

Expliquemos que sucede: La clase Mascota tiene attr_accessor para 3 variables de instancia precio, edad y tipo el attr_accessor hace dos métodos exactamente iguales a los métodos color= y color de la clase Perro, estos nos permiten acceder a la variable a través de la asignación y la consulta tal como se ve en el ejemplo. La clase Perro hereda todos los métodos y atributos de la clase Mascota de ahí que es posible acceder al método tipo= y tipo y a su consecuente atributo.

El constructor es initialize que cuando se le envía un parámetro obvia el default, pero si el parámetro esta ausente entonces si lo aplicará. También existen attr_reader y attr_writer para generar los métodos asociados con la variable.

En algunos casos tenemos que crear métodos especiales para clases ya instanciadas, siguiendo el programa anterior, para ejemplificar esto agregaremos las siguientes instrucciones:

```
def chucho2.desmanchador
  @nombre="Blanco"
end
chucho2.desmanchador
puts "Nuevo nombre de chucho2 "+chucho2.nombre+"\n"

...
Nuevo nombre de chucho2 Blanco
```

En este caso la instancia chucho2 tiene un método exclusivo y si intentamos correr desmanchador sobre chucho1 nos dirá que no es posible por que no existe, a estos métodos singulares se les llama Singleton, y nos evita tener que crear clases especiales para instancias especiales.

Ruby tiene una miríada de funciones ya predefinidas llamados módulos, por motivos de simplicidad y mantener bajo el consumo de recursos no vienen implícitas dentro del lenguajes sino que hay que llamarlos directa o indirectamente, ejemplo en el irb:

```
>> Math::sqrt(2)
=> 1.4142135623731
>> Math::PI
=> 3.14159265358979
>> include Math
=> Object
>> sqrt(2)
=> 1.4142135623731
>> PI
=> 3.14159265358979
```

Los módulos son similares a las clases pero con las siguientes limitantes:
- Un módulo no puede tener instancias
- Un módulo no puede tener subclases
- Un módulo se define con module ... end

El operador :: indica al intérprete de Ruby qué módulo debe consultar para obtener el valor de la constante (es concebible, que algún otro módulo a parte de Math interprete PI de otra forma). Si queremos referenciar a los métodos o constantes de un módulo, directamente, sin utilizar ::, podemos incluir ese módulo con include.

En el caso de las fechas Ruby viene con todo el arsenal listo para trabajar, sin embargo hay un conjunto de modulos que componen una biblioteca, en este caso llamaremos a date que está dentro del conjunto estandard:

```
>> require 'date'
=> true
>> Time::now
=> Wed Jul 20 18:57:09 -0600 2011
>> fecha=Date.new(1972,5,25)
=> #<Date: 4882925/2,0,2299161>
>> fecha.year
=> 1972
>> fecha.to_s
=> "1972-05-25"
>> fechahora=DateTime.new(1972,5,25,23,5,10)
=> #<DateTime: 21094244311/8640,0,2299161>
>> fechahora.to_s
=> "1972-05-25T23:05:10+00:00"
>> (fechahora+5).to_s
=> "1972-05-30T23:05:10+00:00"
>> fecha=Date.new(2011,2,30)
ArgumentError: invalid date
      from /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/1.8/
date.rb:752:in `new'
      from (irb):55
```

Hay un tipo muy especial que se le llama objeto procedimiento tiene forma de variable pero se comporta como procedimiento y resulta muy útil por que es posible crear funciones al aire y asignarlas a un objeto procedimiento y ejecutarlas:

```
>> hola = proc { print "hola mundo\n" }
=> #<Proc:0x003298a8@(irb):16>
>> hola.call
hola mundo
```

Cabe mencionar que Ruby puede separar instrucciones con ; si están en una misma linea y proc puede tomar múltiples líneas englobadas entre { y }.

Ruby tiene variables especiales como las globales, denotadas por $, variables de sistema que se presentará un listado a continuación y constantes que al final son variables de solo una vez. Este es el listado de las variables de sistema, es posible que existan algunas otras:

```
$! Último mensaje de error
$@ Posición del error
$_ Última cadena leída con gets
$. Último número de línea leído por el interprete
$& Última cadena que ha coincidido con una expresión regular
$~ Última cadena que ha coincidido con una expresión regular como array de subexpresiones
$n La n-ésima subexpresión regular de la última coincidencia (igual que $~[n])
$= flag para tratar igual las mayúsculas y minúsculas
$/ Separador de registros de entrada
$\ Separador de registros de salida
$0 El nombre del fichero del guión Ruby
$* El comando de la línea de argumentos
$$ El número de identificación del proceso del intérprete Ruby
$? Estado de retorno del último proceso hijo ejecutado
```

De las variables anteriores $_ y $~, tienen ámbito local. Sus nombres sugieren que deberían tener ámbito global, pero son más útiles de esta forma y existen razones históricas para utilizar estos identificadores.

Una constante se denota por una mayúscula inicial, en realidad es posible reasignarla, Ruby se restringirá a un mensaje de aviso y eso es todo:

```
>> Varglobal=1
=> 1
>> Varglobal=2
(irb):19: warning: already initialized constant Varglobal
=> 2
>> Varglobal
=> 2
```

Ruby ofrece una pequeña caja de herramientas para cuando las cosas no salen como es esperado, es el control de excepciones:

```
>> begin
?>    print 23/0
>> rescue
>>    print "Ocurrió un error, pero no se preocupe, vuelva a intentarlo"
>> end
Ocurrió un error, pero no se preocupe, vuelva a intentarlo=> nil
```

O en su forma lineal que siempre ayuda:

```
>> print 23/0 rescue "Error matemático, vuelva a intentarlo"
=> "Error matemático, vuelva a intentarlo"
```

La idea es mantener el error controlado y no presentar algo que venda que nuestra aplicación se acaba de ir al infierno mismo, y de ser posible intentar continuar con la operación, existe la instrucción `ensure` que correrá el conjunto de instrucción siguiente aún si hay error.

# Capítulos propuestos por los descarrilados

AJAX

Los cambios

Las pruebas

GIT

Respaldo

Importación

Informes

Exportación

Publicación

SOAP, XML, JSON, WebServices

Los generadores

JRuby

Hosting