

Apuntes de RUBY

Casiano R. León ¹

¹DEIOC Universidad de La Laguna

A Juana

*For it is in teaching that we learn
And it is in understanding that we are understood*

Agradecimientos/Acknowledgments

I'd like to thank

A mis alumnos de Lenguajes y Paradigmas de Programación del Grado de Informática de la Escuela Superior de Informática en la Universidad de La Laguna

Parte I

LENGUAJES Y PARADIGMAS DE PROGRAMACION

Capítulo 1

Introducción a los Lenguajes y Paradigmas de Programación

Los primeros computadores costaban millones de dólares de 1940, ocupaban una o varias habitaciones y consumían tanta electricidad como una fábrica de tamaño mediano.

El Colossus fué el primer computador electrónico digital completamente programable (1943-1944).

Los primeros computadores se programaban en lenguaje máquina: una secuencia de bits que controla directamente al procesador. Los programadores usaban *lenguaje máquina* entre otras cosas porque estaban convencidos de que el tiempo del computador era mucho mas valioso que el tiempo del programador.

El lenguaje ensamblador fue inventado para permitir que las operaciones fueran expresadas mediante abreviaciones nemómicas.

Al principio la correspondencia era uno-uno entre nemómicos e instrucciones de la máquina. Luego se les añadieron *expansiones de macros*, que permiten la abreviación de secuencias parametrizadas de secuencias de instrucciones que aparecen a menudo.

Cuando los computadores empezaron a evolucionar y a tener distintos diseños, comenzó también la frustración de tener que reescribir los programas para cada nueva máquina. Empezó a ser clara la necesidad de disponer de un lenguaje que fuera independiente de la máquina en el cual fuera posible expresar usando una notación matemática los cálculos. En la década de los 50 aparece el lenguaje Fortran. El lenguaje Lisp especificado en 1958 por John McCarthy es sólo un año mas joven. También de esa época es el lenguaje Algol.

Los *compiladores* son los programas del sistema que se encargan de la traducción desde un lenguaje de alto nivel al lenguaje ensamblador o al lenguaje máquina.

Al comienzo los programadores podían escribir código en ensamblador que era mas eficiente que el que el compilador producía. Hoy en día, en parte debido a la complejidad del hardware (pipelining, multiples unidades funcionales, etc.) un compilador puede producir mejor código que el escrito por un humano. Además el coste del trabajador hoy en día sobrepasa con mucho al costo del hardware. Se trata por tanto de economizar en la construcción de programas y su mantenimiento.

1.1. Modelos de Programación

Programming models bridge the gap between the underlying hardware architecture and the supporting layers of software available to applications.

Programming models are different from both programming languages and application programming interfaces (APIs).

Specifically, a *programming model* is an abstraction of the underlying computer system that allows for the expression of both algorithms and data structures.

In comparison, languages and APIs provide implementations of these abstractions and allow the algorithms and data structures to be put into practice – a programming model exists independently of the choice of both the programming language and the supporting APIs.

1.2. Paradigmas de Programación

A *programming paradigm* is a fundamental style of computer programming, a way of building the structure and elements of computer programs.

Capabilities and styles of various programming languages are defined by their supported programming paradigms; some programming languages are designed to follow only one paradigm, while others support multiple paradigms. Programming paradigms that are often distinguished include

- imperative (Fortran, C)
- declarative (Prolog, SQL)
- functional (Haskell)
- object-oriented (Smalltalk, Ruby)
- logic (Prolog)

Programming paradigms can be compared with programming models that are abstractions of computer systems.

For example, the *von Neumann model* is a programming model used in traditional sequential computers. For parallel computing, there are many possible models typically reflecting different ways processors can be interconnected. The most common are based on shared memory, distributed memory with message passing, or a hybrid of the two.

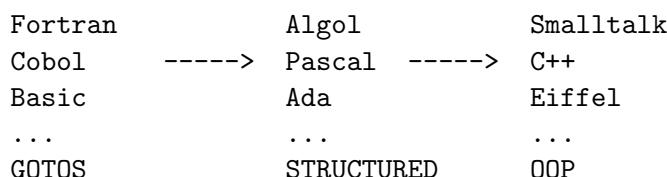
Some programming language researchers criticise the notion of paradigms as a classification of programming languages. They argue that many programming languages cannot be strictly classified into one paradigm, but rather include features from several paradigms.

- *Imperative programming* is focused on describing how a program operates. It describes computation in terms of **statements** that change a **program state**.
- *Object-oriented programming* integrates code and data using the concept of an “object”. An object has both state (data) and behavior (code).
- *Declarative programming* is a style of programming in which programs describe their desired results without explicitly listing commands or steps that must be performed.
- *Logic programming* is a programming paradigm based on formal logic

1.3. El Arte del Diseño de Lenguajes

¿Por que hay tantos lenguajes de programación?

1. Evolución



2. Propósito específico

Lisp	Computación simbólica
Prolog	Relaciones lógicas
Snobol, Icon	Manipulación de texto
Perl	Manipulación de texto y programación de sistemas

3. Preferencias personales

Brevedad de C
Recursión versus iteración
Punteros

¿Qué hace que un lenguaje tenga éxito?

1. Expresividad

Basic (primeras versiones) versus Common Lisp

2. Facilidad de aprendizaje

Basic
Logo, Scratch,
Pascal
Java

3. Fácil de implementar

Basic
Pascal

4. Open Source

C

5. Calidad de los compiladores

Fortran

6. Razones económicas, Patronazgo, Inercia

Cobol, PL/I	<--- IBM
Ada	<--- Departamento de Defensa USA
C#	<--- Microsoft
Java	<--- Sun, Oracle, Google

1.4. El Espectro de los Lenguajes de Programación

Los lenguajes de programación pueden ser clasificados en familias de acuerdo con su modelo de computación [?].

declarativo	
funcional	Lisp, Scheme, Clojure, ML, Haskell, Scala
flujo de datos (dataflow)	Id, Val
lógicos, basados-en-restricciones	Prolog
basados en templates	XSLT
imperativos	
von Neuman	C, Ada, Fortran
scripting	Perl, Python, PHP
orientados a objetos	Smalltalk, Ruby, Eiffel, C++, Scala

1. Los *lenguajes funcionales* usan un modelo computacional basado en la definición de funciones recursivas. Se inspiran en el lambda cálculo (Lisp, ML, Haskell, Erlang, Elixir)
2. Los *lenguajes dataflow* realizan su cálculo según el flujo de información entre nodos funcionales. Son ejemplos de Dataflow los lenguajes Id y Val. Las hojas de cálculo también pueden considerarse un ejemplo, ya que el cambio de una variable fuerza el re-cálculo de todas las celdas que dependen de ella
3. Los *lenguajes lógicos* y los *lenguajes de restricciones* se inspiran en la lógica de predicados
4. Los *lenguajes de von Neumann* se basan en la ejecución de las sentencias de un programa almacenado en memoria y que cambian los valores de las variables almacenadas en la memoria
5. Los *lenguajes de scripting* o *lenguajes dinámicos* facilitan la combinación de componentes que fueron desarrolladas como programas independientes. Son lenguajes de scripting bash, csh, awk, PHP, JavaScript, Perl, Python, Ruby, ...
6. Los *Lenguajes orientados a objetos* describen la solución en base a objetos que interactúan entre sí. Los objetos disponen de su propio estado interno y un conjunto de subrutinas (denominadas métodos) que modifican dicho estado. El lenguaje orientado a objetos por excelencia es Smalltalk

En cuanto a la *concurrencia*, la más de las veces es implementada mediante librerías, paquetes y compiladores extendiendo las capacidades de un lenguaje secuencial como C o Fortran. Algunos lenguajes, entre ellos Java, C#, Ada, Modula-3, Erlang y Elixir contienen características explícitamente concurrentes.

Para ver como diferentes lenguajes resuelven el mismo problema puede consultarse:

- Rosetta Code: A site with solutions to the same task in as many different languages as possible, to demonstrate how languages are similar and different

1.5. ¿Porqué estudiar lenguajes?

1. Los lenguajes de propósito/dominio específico permiten expresar de forma más sencilla la solución de un determinado problema:
 - a) HTML para escribir páginas web
 - b) CSS para darles estilo
 - c) Make o Rake para describir el workflow de tareas
 - d) LATEX para escribir documentos
 - e) SQL para la manipulación de datos
 - f) DOT (GraphViz) para dibujar grafos

la mayor parte de los programadores nunca implementarán un lenguaje de programación pero si que posiblemente tendrán que escribir programas que transformen de un formato a otro o crear su propio lenguaje de configuración para una cierta aplicación

2. Simular características útiles de un lenguaje en otro
3. Elegir entre varias alternativas para expresar una misma cosa una alternativa óptima

Capítulo 2

Programación Imperativa

Imperative programming is focused on describing how a program operates.

In computer science terminologies, imperative programming is a programming paradigm that **describes computation in terms of statements that change a program state**.

The term is often used in contrast to *declarative programming*, which focuses on **what the program should accomplish without prescribing how to do it in terms of sequences of actions to be taken**. Véase la sección Imperative Programming de la Wikipedia.

2.1. Estructuras de Datos

Véase la sección Data structure en la Wikipedia

2.1.1. Tipos de Datos Abstractos/Abstract Data Types

In computer science, an *abstract data type* (ADT) is a mathematical model for a certain class of data structures that have similar behavior; or for certain data types of one or more programming languages that have similar semantics.

An abstract data type is defined indirectly, only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

For example, an abstract stack could be defined by three operations: push, that inserts some data item onto the structure, pop, that extracts an item from it (with the constraint that each pop always returns the most recently pushed item that has not been popped yet), and peek, that allows data on top of the structure to be examined without removal.

When analyzing the efficiency of algorithms that use stacks, one may also specify that all operations take the same time no matter how many items have been pushed into the stack, and that the stack uses a constant amount of storage for each element.

Abstract data types are purely theoretical entities, used (among other things) to simplify the description of abstract algorithms, to classify and evaluate data structures, and to formally describe the type systems of programming languages.

However, an ADT may be implemented by specific data types or data structures, in many ways and in many programming languages; or described in a formal specification language.

ADTs are often implemented as modules: the module's interface declares procedures that correspond to the ADT operations, sometimes with comments that describe the constraints.

This information hiding strategy allows the implementation of the module to be changed without disturbing the client programs.

2.2. Abstracción y Encapsulamiento

Abstraction is the process of identifying common patterns that have systematic variations; an abstraction represents the common pattern and provides a means for specifying which variation to use

Encapsulation is hiding the implementation details which may or may not be for generic or specialized behavior(s).

2.3. Subprogramas

Véase la sección Subrutinas en la Wikipedia.

Véase la sección Call stack en la Wikipedia

2.4. Programación Estructurada

2.4.1. Go To Statement Considered Harmful

Goto (`goto`, `GOTO`, `GO TO` or other case combinations, depending on the programming language) is a statement found in many computer programming languages. It performs a one-way transfer of control to another line of code; in contrast a function call normally returns control. The jumped-to locations are usually identified using labels, though some languages use line numbers. At the machine code level, a `goto` is a form of branch or jump statement. Many languages support the `goto` statement, and many do not; see language support for discussion.

The structured program theorem proved that the `goto` statement is not necessary to write programs; some combination of the three programming constructs of sequence, selection/choice, and repetition/iteration are sufficient for any computation that can be performed by a Turing machine, with the caveat that code duplication and additional variables may need to be introduced.

At machine code level, `goto` is used to implement the structured programming constructs.

In the past there was considerable debate in academia and industry on the merits of the use of `goto` statements.

Use of `goto` was formerly common, but since the advent of structured programming in the 1960s and 1970s its use has declined significantly.

The primary criticism is that code that uses `goto` statements is harder to understand than alternative constructions.

`Goto` remains in use in certain common usage patterns, but alternatives are generally used if available.

Debates over its (more limited) uses continue in academia and software industry circles.

Probably the most famous criticism of `GOTO` is a 1968 letter by Edsger Dijkstra called Go To Statement Considered Harmful.

In that letter Dijkstra argued that unrestricted `GOTO` statements should be abolished from higher-level languages because they complicated the task of analyzing and verifying the correctness of programs (particularly those involving loops).

The letter itself sparked a debate, including a '`GOTO Considered Harmful`' *Considered Harmful* letter sent to Communications of the ACM (CACM) in March 1987, as well as further replies by other people, including Dijkstra's On a Somewhat Disappointing Correspondence.

2.4.2. El teorema del programa estructurado

El teorema del programa estructurado es un resultado en la teoría de lenguajes de programación. Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- **Secuencia:** ejecución de una instrucción tras otra.
- **Selección:** ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- **Iteración:** ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

Este teorema demuestra que la instrucción **GOTO** no es estrictamente necesaria y que para todo programa que la utilice existe otro equivalente que no hace uso de dicha instrucción.

Los científicos de la computación usualmente acreditan el teorema a un artículo de 1966 escrito por Corrado Böhm y Giuseppe Jacopini. Sin embargo, David Harel rastreó sus orígenes hasta la descripción de 1946 de la arquitectura de von Neumann y el teorema de la forma normal de Kleenex.

La demostración de Böhm-Jacopini describe cómo construir diagramas de flujo estructurados a partir de cualquier diagrama de flujo, usando los bits de una variable entera extra para dar seguimiento a la información que el programa original representa mediante puntos de entrada en el código. Esta construcción estuvo basada en el lenguaje de programación P'' de Böhm.

La demostración de Böhm-Jacopini no esclareció la cuestión sobre cuándo convendría usar programación estructurada para el desarrollo de software, en parte porque la construcción ofuscaba el código del programa en lugar de mejorarlo.

Por otro lado, fue el punto de partida para iniciar el debate. Edsger Dijkstra escribió una importante carta titulada "La sentencia Go To considerada dañina."^{en} en el año 1968.

Posteriores estudios agregaron aproximaciones más prácticas a la demostración de Böhm-Jacopini, que mantenían o mejoraban la claridad del programa original.

Véase Teorema del programa estructurado en la Wikipedia.

Capítulo 3

Erlang

Código Fuente

```
casiano@exthost2:~/src/LPP/170912/erlang (master)$ ls
factorial.beam factorial.erl README

casiano@exthost2:~/src/LPP/170912/erlang (master)$ cat factorial.erl
-module(factorial).
-export([fac/1]).

fac(0) -> 1;
fac(N) -> N * fac(N-1).

casiano@exthost2:~/src/LPP/170912/erlang (master)$ erl
Erlang R13B03 (erts-5.7.4) [source] [smp:2:2] [rq:2] [async-threads:0]
[hipe] [kernel-poll:false]

Eshell V5.7.4 (abort with ^G)
1> c(factorial).
{ok,factorial}
2> factorial:fac(4).
24
3> halt().
```

Quicksort en Erlang

```
-module(quicksort).

-export([qsort/1]).

qsort([]) -> [];
qsort([X|Xs]) ->
    qsort([Y || Y <- Xs, Y < X]) ++ [X] ++ qsort([Y || Y <- Xs, Y >= X]).
```

Capítulo 4

Elixir

Elixir is a functional, concurrent, general-purpose programming language built atop the Erlang Virtual Machine (BEAM). Elixir builds on top of Erlang to provide distributed, fault-tolerant, soft real-time, non-stop applications but also extends it to support meta-programming with macros and polymorphism via protocols

- LoneStarRuby Conf 2013 - Elixir: Power of Erlang, Joy of Ruby by Dave Thomas en YouTube
- Elixir: Getting started guide
- elixir home page
- Nine Minutes of Elixir
- Introduction to Elixir
- Jose Valim - Let's talk concurrency (Railsberry 2012)

4.0.1. Ejemplo: Suma de los elementos de una lista

```
▪ [~/src_elixir]$ pwd -P  
/Users/casiano/Google Drive/src/elixir  
  
▪ gist  
  
[~/Google Drive/src/elixir]$ cat sum.elixir  
defmodule MyList do  
  def sum([]), do: 0  
  def sum([ head | tail ]), do: head + sum(tail)  
end  
IO.puts MyList.sum [1, 2, 3, 4]  
  
[~/Google Drive/src/elixir]$ elixir sum.elixir  
/Users/casiano/Google Drive/src/elixir/sum.elixir:1: redefining module MyList  
10  
  
[~/src_elixir]$ iex  
Erlang R16B (erts-5.10.1) [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:f  
  
Interactive Elixir (0.12.0) - press Ctrl+C to exit (type h() ENTER for help)  
iex(1)> c("sum.elixir")  
sum.elixir:1: redefining module MyList  
10  
[MyList]
```

```
iex(2)> MyList.sum [1, 2, 3, 4, 5]
15
iex(3)>
```

Es posible escribir un módulo en la consola:

```
iex(3)> defmodule MyModule do
...>   def hello do
...>     IO.puts "Another Hello"
...>   end
...> end
{:module, MyModule,
<<70, 79, 82, 49, 0, 0, 7, 104, 66, 69, 65, 77, 65, 116, 111, 109, 0, 0, 0, 124, 0, 0, 0, 13,
{:hello, 0}}
iex(4)> MyModule.hello
Another Hello
:ok
```

4.0.2. Quicksort

```
defmodule Quicksort do
  def qsort([]) do
    []
  end

  def qsort([pivot | tail]) do
    qsort( lc x inlist tail, x < pivot, do: x ) ++ [pivot] ++ qsort( lc x inlist tail,x >= pivot )
  end

end

IO.puts inspect(Quicksort.qsort([10,9,1,3,5,6,4,2,8,7]))

defmodule QuickSort do
  def sort([], do: [])
  def sort([head|tail]) do
    {lesser, greater} = Enum.partition(tail, &(&1 < head))
    sort(lesser) ++ [head] ++ sort(greater)
  end
end

IO.inspect QuickSort.sort([1,6,3,4,2,5])
```

Capítulo 5

Programación Lógica

Código fuente

```
casiano@exthost2:~/src/LPP/170912/prolog (master)$ ls
fact.pl  Rakefile  README

casiano@exthost2:~/src/LPP/170912/prolog (master)$ cat fact.pl
fact(X, 1) :- X<2.
fact(X, F) :- Y is X-1, fact(Y,Z), F is Z*X.

casiano@exthost2:~/src/LPP/170912/prolog (master)$ prolog
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.8.0)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [fact].
% fact compiled 0.00 sec, 1,076 bytes
true.

?- fact(4,X).
X = 24 .

?- fact(4, 24).
true .

?- halt.
casiano@exthost2:~/src/LPP/170912/prolog (master)$
```

Capítulo 6

Paralelismo y Computación de Alto Rendimiento en Ruby

1. parallel-examples en GitHub. Algunos ejemplos de uso de la gema parallel
2. Code Safari: Forks, pipes, and the Parallel gem en SitePoint por Xavier Shay Published June 16, 2011
3. La gema parallel en GitHub
4. Easy Ruby Paralellism With the 'Parallel' Gem Sep 27th, 2011
5. peach: parallel each en GitHub
6. Basic Benchmarking in Ruby por Joshua Ballanco
7. Measure Twice, Code Once por Joshua Ballanco
8. Benchmarking parallel gem
 - Charles Nutter - High Performance Ruby

Capítulo 7

Go

- Go for Rubyists, I. Dhaivat Pandya. Published Decemeber 12, 2013
- Go for Rubyists, II. Dhaivat Pandya. Published January 6, 2014

Parte II

**EL LENGUAJE DE
PROGRAMACIÓN RUBY**

Capítulo 8

Introducción

8.1. Primeros Pasos

Puedes usar una interfaz web del intérprete Ruby (se trata de un bucle REPL similar al programa `irb`)¹ en:

<http://tryruby.org/>.

8.1.1. irb

- Instale interactive_editor para `irb`. Siga las instrucciones en el enlace.
- Instale wirble. Siga las instrucciones en el enlace.
- Tab completion in `irb`
- Estudie Ruby en 20 minutos. Versión inglesa en: Ruby in Twenty Minutes
- Opcionalmente, lea este enlace What's Your Favourite IRB Trick? en StackOverflow, en el que se explican algunos trucos con `irb`.

8.1.2. pry

- `pry`
- A basic introduction to using Pry, an IRB alternative for Ruby de Banister Fiend
- Ruby Debugging with Pry de Jason Madsen
- Jason R. Clark - Spelunking in Ruby. Baruco 2014
- Ruby Conf 2013 - REPL driven development with Pry by Conrad Irwin

8.2. Donde Encontrar Ruby y Como Instalarlo

El fuente Ruby lo puede encontrar en la página Web:

<http://www.ruby-lang.org/en/downloads/>.

Se encuentran disponibles los fuentes y los binarios para distintos sistemas.
Comprueba que Ruby existe en tu ordenador:

```
[~/rubystesting/pry]$ ruby -v
ruby 2.1.2p95 (2014-05-08 revision 45877) [x86_64-darwin13.0]
```

¹Interactive RuBy

8.2.1. Donde esta Ruby en Nuestra Máquina

Comprueba en que directorio se guarda el ejecutable:

```
lpp@nereida:~/LPPbook$ which ruby  
[~/rubytesting/pry]$ which ruby  
/Users/casiano/.rvm/rubies/ruby-2.1.2/bin/ruby
```

8.2.2. RVM

Para tener instaladas diferentes versiones de Ruby y poder usarlas es conveniente instalar RVM.
Lea la sección 81 sobre RVM (Ruby Version Manager).

8.2.3. Ejecución de un Programa Ruby

Edita y guarda el fichero `hola.rb`. Cambia los permisos de ejecución:

```
~/rubytesting$ vi hello.rb  
  
... salimos y salvamos ...  
  
~/rubytesting$ cat -n hello.rb  
1#!/usr/bin/env ruby  
2 puts 'Hello World!'  
~/rubytesting$  
  
~/rubytesting$ chmod a+x hello.rb
```

Ahora ya puedes ejecutar tu primer programa Ruby:

```
~/rubytesting$ ./hello.rb  
Hello World!
```

También podríamos haber escrito:

```
~/rubytesting$ ruby hello.rb  
Hello World!
```

8.3. Ejercicio: Calcule el Factorial de un Número

Ejercicio 8.3.1. Escriba un programa que calcule el factorial de un número, pasado en la línea de comandos. Puede encontrar una solución (no la consulte ahora, inténtelo) en <http://www.rubyist.net/slayer/ruby/examples.html>. Pruebela con números grandes `fact(50)`.

8.4. Lints in Ruby: rubocop, reek, etc.

- rubocop
- pelusa: Pelusa is a static analysis tool and framework to inspect your code style and notify you about possible red flags or missing best practices.
- reek

Rubocop

RuboCop is a Ruby static code analyzer. Out of the box it will enforce many of the guidelines outlined in the community Ruby Style Guide.

```
[~/rubytesting/rubocop]$ cat test.rb
def badName
  if something
    test
  end
end
```

```
[~/rubytesting/rubocop]$ rubocop --version
0.26.1
[~/rubytesting/rubocop]$ date
martes, 7 de octubre de 2014, 10:22:07 WEST
```

```
[~/rubytesting/rubocop]$ rubocop test.rb
Inspecting 1 file
W
```

Offenses:

```
test.rb:1:5: C: Use snake_case for methods.
def badName
  ~~~~~
test.rb:2:3: C: Use a guard clause instead of wrapping the code inside a conditional expression
  if something
  ^^
test.rb:2:3: C: Favor modifier if usage when having a single-line body. Another good alternative would be to use a guard clause.
  if something
  ^^
test.rb:4:5: W: end at 4, 4 is not aligned with if at 2, 2
  end
  ^^^

1 file inspected, 4 offenses detected
```

```
[~/rubytesting/rubocop]$ cat test2.rb
def good_name
  test if something
end

[~/rubytesting/rubocop]$ rubocop test2.rb
Inspecting 1 file
.

1 file inspected, no offenses detected
```

Code Smells: reek

reek is a tool that examines Ruby classes, modules and methods and reports any Code Smells it finds.

```
) [~/rubytesting/reek]$ ls
Gemfile      Gemfile.lock Rakefile      dirty.rb
[~/rubytesting/reek]$ cat dirty.rb
class Dirty
```

```

# This method smells of :reek:NestedIterators but ignores them
def awful(x, y, offset = 0, log = false)
  puts @screen.title
  @screen = widgets.map { |w| w.each { |key| key += 3}}
  puts @screen.contents
end
end

[~/rubytesting/reek]$ reek dirty.rb
dirty.rb -- 8 warnings:
[1]:Dirty has no descriptive comment (IrresponsibleModule)
[3]:Dirty#awful has 4 parameters (LongParameterList)
[3]:Dirty#awful has boolean parameter 'log' (BooleanParameter)
[5]:Dirty#awful has the variable name 'w' (UncommunicativeVariableName)
[3]:Dirty#awful has unused parameter 'log' (UnusedParameters)
[3]:Dirty#awful has unused parameter 'offset' (UnusedParameters)
[3]:Dirty#awful has unused parameter 'x' (UnusedParameters)
[3]:Dirty#awful has unused parameter 'y' (UnusedParameters)

```

8.5. Ejecución de un Programa con el Depurador

También podemos usar el depurador para ejecutar el programa paso a paso. El depurador de Ruby se activa usando `-rdebug`:

```

~/rubytesting$ cat -n polar.rb
1 #!/usr/bin/env ruby
2
3 def c2p(x, y)
4   theta = Math.atan2(y, x)
5   r     = Math.hypot(x, y)
6   [r, theta]
7 end
8 x, y = 1, 1
9 x, y = ARGV.map { |t| t.to_f } if ARGV.length == 2
10 r, theta = c2p(x, y)
11
12 puts "r = #{r} theta = #{theta}\n"

```

Para llamar al debugger:

```

~/rubytesting$ ruby -rdebug polar.rb 1 1
Debug.rb
Emacs support available.

```

`polar.rb:3:def c2p(x, y)`

Para pedir ayuda usamos `h`:

```

(rdb:1) help
Debugger help v.-0.002b
Commands
b[reak] [file:|class:]<line|method>
b[reak] [class.]<line|method>
                               set breakpoint to some position
wat[ch] <expression>      set watchpoint to some expression
cat[ch] (<exception>|off)  set catchpoint to an exception

```

```

b[reak]                      list breakpoints
cat[ch]                       show catchpoint
del[ete] [ nnn]                delete some or all breakpoints
disp[lay] <expression>        add expression into display expression list
undisp[lay] [ nnn]              delete one particular or all display expressions
c[ont]                          run until program ends or hit breakpoint
s[tep] [ nnn]                  step (into methods) one line or till line nnn
n[ext] [ nnn]                  go over one line or till line nnn
w[here]                         display frames
f[rame]                         alias for where
l[ist] [ (-|nn-mm)]           list program, - lists backwards
                               nn-mm lists given lines
up[ nn]                         move to higher frame
down[ nn]                        move to lower frame
fin[ish]                         return to outer frame
tr[ace] (on|off)                set trace mode of current thread
tr[ace] (on|off) all            set trace mode of all threads
q[uit]                           exit from debugger
v[ar] g[lobal]                  show global variables
v[ar] l[ocal]                   show local variables
v[ar] i[nstance] <object>      show instance variables of object
v[ar] c[onst] <object>          show constants of object
m[ethod] i[nstance] <obj>       show methods of object
m[ethod] <class|module>        show instance methods of class or module
th[read] l[ist]                  list all threads
th[read] c[ur|rent]             show current thread
th[read] [sw[itch]] <nnn>        switch thread context to nnn
th[read] stop <nnn>             stop thread nnn
th[read] resume <nnn>           resume thread nnn
p expression                     evaluate expression and print its value
h[elp]                           print this help
<everything else>              evaluate
(rdb:1)

```

Para evaluar una expresión:

```
(rdb:1) p 2*5
10
(rdb:1)
```

Para listar el programa:

```
(rdb:1) l
[-2, 7] in polar.rb
 1
 2
=> 3  def c2p(x, y)
    4    theta = Math.atan2(y, x)
    5    r      = Math.hypot(x, y)
    6    [r, theta]
    7  end
(rdb:1) l
[8, 17] in polar.rb
 8  x, y = 1, 1
 9  x, y = ARGV.map { |t| t.to_f }  if ARGV.length == 2
```

```

10 r, theta = c2p(x, y)
11
12 puts "r = #{r} theta = #{theta}\n"
(rdb:1)

```

Para ejecutar una sentencia:

```

(rdb:1) n
polar.rb:8:x, y = 1, 1
(rdb:1)
n
polar.rb:9:x, y = ARGV.map { |t| t.to_f } if ARGV.length == 2
(rdb:1) x
1
(rdb:1) y
1
(rdb:1)

```

Para examinar una expresión:

```

(rdb:1) ARGV
["1", "1"]
(rdb:1) ARGV.length
2
(rdb:1) p "43".to_f
43.0

```

Para probar e investigar:

```

(rdb:1) puts "hello" if ARGV.length == 2
hello
nil

```

Para investigar el funcionamiento de `map`:

```

(rdb:1) [1,8, 9].map { |x| x*x }
[1, 64, 81]
(rdb:1)

```

La sentencia con el `map` es en realidad un bucle:

```

(rdb:1) n
polar.rb:9:x, y = ARGV.map { |t| t.to_f } if ARGV.length == 2
(rdb:1) n
polar.rb:9:x, y = ARGV.map { |t| t.to_f } if ARGV.length == 2
(rdb:1) n
polar.rb:9:x, y = ARGV.map { |t| t.to_f } if ARGV.length == 2
(rdb:1) n
polar.rb:10:r, theta = c2p(x, y)

```

Para entrar al método usamos `s`:

```

(rdb:1) s
polar.rb:4: theta = Math.atan2(y, x)
(rdb:1) n
polar.rb:5: r      = Math.hypot(x, y)
(rdb:1) p theta
0.785398163397448

```

Para salir de la rutina usamos `finish` que retorna al stackframe que envuelve al actual:

```
(rdb:1) fin
polar.rb:12:puts "r = #{r} theta = #{theta}\n"
```

Para continuar la ejecución hasta el siguiente punto de break usamos `c`:

```
(rdb:1) c
r = 1.4142135623731 theta = 0.785398163397448
```

Con `b` podemos poner un punto de break:

```
[~/rubystesting]$ ruby -rdebug polar.rb
Debug.rb
Emacs support available.

polar.rb:3:def c2p(x, y)
(rdb:1) l
[-2, 7] in polar.rb
 1  #!
 2
=> 3  def c2p(x, y)
 4    theta = Math.atan2(y, x)
 5    r      = Math.hypot(x, y)
 6    [r, theta]
 7  end
(rdb:1) l
[8, 17] in polar.rb
 8  x, y = 1, 1
 9  x, y = ARGV.map { |t| t.to_f }  if ARGV.length == 2
10  r, theta = c2p(x, y)
11
12 puts "r = #{r} theta = #{theta}\n"
(rdb:1) b 4
Set breakpoint 1 at polar.rb:4
(rdb:1) c
Breakpoint 1, c2p at polar.rb:4
polar.rb:4:  theta = Math.atan2(y, x)
```

Con `w` podemos ver en que lugar estamos:

```
(rdb:1) w
--> #1 polar.rb:4:in `c2p'
     #2 polar.rb:10
(rdb:1)
```

Véase Para saber mas sobre el debugger consulta:

1. La sección *Debugging, Depurando* 86 de estos apuntes
2. La sección When Trouble Strikes del libro Programming Ruby
3. <http://www.ibm.com/developerworksopensource/tutorials/os-distruby/section3.html>
4. <http://ruby.about.com/od/advancedruby/a/debugging.htm>

8.6. Averiguando que tiempo hace

- `http://openweathermap.org/`: Call current weather data for one location
- OpenURI is an easy-to-use wrapper for Net::HTTP, Net::HTTPS and Net::FTP.
Véase la Documentación
- JSON is a lightweight data-interchange format. It is easy for us humans to read and write. Plus, equally simple for machines to generate or parse. JSON is completely language agnostic, making it the ideal interchange format.
- Psych is a YAML parser and emitter. Psych leverages libyaml for its YAML parsing and emitting capabilities. In addition to wrapping libyaml, Psych also knows how to serialize and de-serialize most Ruby objects to and from the YAML format.

```
[~/rubytesting]$ cat wheater.rb
require 'open-uri'
require 'json'
require 'psych'

city = 'san cristobal de la laguna,spain'.gsub(/\s/, '%20')
request = "http://api.openweathermap.org/data/2.5/weather?q=#{city}&units=metric"
response = open(request).readlines.join
print Psych.dump( JSON.parse(response) )

[~/rubytesting]$ pry
[1] pry(main)> require 'open-uri'
=> true
[2] pry(main)> require 'json'
=> true
[3] pry(main)> require 'psych'
=> true

[4] pry(main)> city = 'san cristobal de la laguna,spain'.gsub(/\s/, '%20')
=> "san%20cristobal%20de%20la%20laguna,spain"

[5] pry(main)> request = "http://api.openweathermap.org/data/2.5/weather?q=#{city}&units=metric"
=> "http://api.openweathermap.org/data/2.5/weather?q=san%20cristobal%20de%20la%20laguna,spain&units=metric"

[6] pry(main)> f = open(request)
=> #<StringIO:0x007fdee29fc4a0
@base_uri=
#<URI::HTTP:0x007fdee29fc720 URL:http://api.openweathermap.org/data/2.5/weather?q=san%20cristobal%20de%20la%20laguna,spain&units=metric
@meta=
{"server"=>"nginx",
 "date"=>"Tue, 07 Oct 2014 10:30:15 GMT",
 "content-type"=>"application/json; charset=utf-8",
 "transfer-encoding"=>"chunked",
 "connection"=>"keep-alive",
 "x-source"=>"redis",
 "access-control-allow-origin"=>"*",
 "access-control-allow-credentials"=>"true",
 "access-control-allow-methods"=>"GET, POST"},

@metas=
{"server"=>["nginx"],
```

```

"date"=>["Tue, 07 Oct 2014 10:30:15 GMT"],
"content-type"=>["application/json; charset=utf-8"],
"transfer-encoding"=>["chunked"],
"connection"=>["keep-alive"],
"x-source"=>["redis"],
"access-control-allow-origin"=>["*"],
"access-control-allow-credentials"=>["true"],
"access-control-allow-methods"=>["GET, POST"]},
@status=["200", "OK"]>

[7] pry(main)> a = f.readlines
=> [{"coord": {"lon": -16.31, "lat": 28.49}, "sys": {"type": 3, "id": 5518, "message": 0.1,
clouds: 0, "icon": "04d"}, "base": "cmc
stations", "main": {"temp": 24, "pressure": 1018, "humidity": 73, "temp_min": 24, "temp_max": 32}},
{"coord": {"lon": -16.31, "lat": 28.49}, "sys": {"type": 3, "id": 5518, "message": 0.1,
clouds: 0, "icon": "04d"}, "base": "cmc
stations", "main": {"temp": 24, "pressure": 1018, "humidity": 73, "temp_min": 24, "temp_max": 32}}]

```

8.7. Ejercicios: Expresiones Regulares. Un programa que convierte de Celsius a Farenheit

Ejercicio 8.7.1. *El siguiente programa convierte de Celsius a Farenheit y viceversa:*

```
MacBookdeCasiano:rubystesting casiano$ ruby f2c.rb 32F
celsius = 0.0 farenheit = 32.0
```

Ejecútelo con y sin la opción -d

```
[~/rubystesting/f2c]$ ruby -d f2c.rb 32F
Exception 'LoadError' at /Users/casiano/.rvm/rubies/ruby-1.9.3-p545/lib/ruby/site_ruby/1.9.1/r
Exception 'LoadError' at /Users/casiano/.rvm/rubies/ruby-1.9.3-p545/lib/ruby/site_ruby/1.9.1/r
32F
Matches num = 32 kind= F
celsius = 0.0 farenheit = 32.0
[~/rubystesting/f2c]$ ruby -d f2c.rb 32X
Exception 'LoadError' at /Users/casiano/.rvm/rubies/ruby-1.9.3-p545/lib/ruby/site_ruby/1.9.1/r
Exception 'LoadError' at /Users/casiano/.rvm/rubies/ruby-1.9.3-p545/lib/ruby/site_ruby/1.9.1/r
32X
does not match
[~/rubystesting/f2c]$ ruby -v
ruby 1.9.3p545 (2014-02-24 revision 45159) [x86_64-darwin13.1.0]
```

e intente averiguar el significado de cada una de las sentencias.

```
$ cat -n f2c.rb
 1  #!/usr/bin/env ruby
 2
 3  t = ARGV[0] || '0C'
 4  puts t if $DEBUG
 5
 6  if t =~ /\s*(\d+(:\.\d+)?)([CF])\s*$/i
 7    puts "Matches num = #{$1} kind= #{$2}" if $DEBUG
 8    num, type = $1.to_f, $2
 9
10  if (type == 'C')
11    celsius = num
12    farenheit = (celsius*9/5)+32
```

```

13     else
14         farenheit = num
15         celsius = (farenheit-32)*5/9
16     end
17     puts "celsius = #{celsius} farenheit = #{farenheit}"
18 else
19     puts "does not match" if $DEBUG
20 end

```

para entenderlo un poco consulte las siguientes secciones de la guía de usuario:

- *Strings*
- *Expresiones Regulares*

Debugging con pry

Pry can be invoked in the middle of a running program. It opens a Pry session at the point it's called and makes all program state at that point available.

It can be invoked on any object using the `my_object.pry` syntax or on the current `binding` (or any binding) using `binding.pry`.

The Pry session will then begin within the scope of the object (or binding).

When the session ends the program continues with any modifications you made to it.

```

[~/rubytesting/f2c]$ cat f2c2.rb
#!/usr/bin/env ruby
require 'pry'

t = ARGV[0] || '0C'

if t =~ /^(\d+\.?\d+)([CF])$/i
    num, type = $1.to_f, $2

    binding.pry
    if (type == 'C')
        celsius = num
        farenheit = (celsius*9/5)+32
    else
        farenheit = num
        celsius = (farenheit-32)*5/9
    end
    puts "celsius = #{celsius} farenheit = #{farenheit}"
else
    $stderr.puts "not expected input"
end

[~/rubytesting/f2c]$ cat Gemfile
source 'https://rubygems.org'

gem 'pry'
gem 'pry-debugger' if RUBY_VERSION == "1.9.3"
gem 'pry-byebug' if RUBY_VERSION == "2.1.2"

[~/rubytesting/f2c]$ ruby -v
ruby 2.1.2p95 (2014-05-08 revision 45877) [x86_64-darwin13.0]
[~/rubytesting/f2c]$ bundle

```

```

Using columnize 0.8.9
Using debugger-linecache 1.2.0
Using slop 3.6.0
Using byebug 3.5.1
Using coderay 1.1.0
Using method_source 0.8.2
Using pry 0.10.1
Using pry-byebug 2.0.0
Using bundler 1.6.2
Your bundle is complete!
Use 'bundle show [gemname]' to see where a bundled gem is installed.

```

```

[~/rubytesting/f2c]$ ruby -v
ruby 1.9.3p545 (2014-02-24 revision 45159) [x86_64-darwin13.1.0]
[~/rubytesting/f2c]$ bundle
Resolving dependencies...
Using coderay 1.1.0
Using columnize 0.8.9
Using debugger-linecache 1.2.0
Using debugger-ruby_core_source 1.3.5
Using debugger 1.6.8
Using method_source 0.8.2
Using slop 3.6.0
Using pry 0.10.1
Using pry-debugger 0.2.3
Using bundler 1.6.0
Your bundle is complete!
Use 'bundle show [gemname]' to see where a bundled gem is installed.

```

```
[~/rubytesting/f2c]$ ruby f2c2.rb
```

```
From: /Users/casiano/local/src/ruby/rubytesting/f2c/f2c2.rb @ line 10 :
```

```

5:
6: if t =~ /^(\d+(:\.\d+)?)([CF])\s*$/i
7:   num, type = $1.to_f, $2
8:
9:   binding.pry
=> 10:  if (type == 'C')
11:    celsius = num
12:    farenheit = (celsius*9/5)+32
13:  else
14:    farenheit = num
15:    celsius = (farenheit-32)*5/9

```

```

[1] pry(main)> p num
0.0
=> 0.0
[2] pry(main)> continue
celsius = 0.0 farenheit = 32.0

```

- pry-debugger
- pry-byebug

8.8. Ejercicio: Excepciones y Expresiones Regulares. Pasar de Hexadecimal a Decimal

Ejercicio 8.8.1. Escriba un programa que reciba como argumentos una lista de cadenas describiendo números hexadecimales y devuelva los numeros en notación decimal.

```
~/rubytesting$ ./h2d.rb 0x1F 0x2A  
31  
42
```

En caso de que alguno de los argumentos no contenga un número hexadecimal deberá producir una excepción:

```
~/rubytesting$ ./h2d.rb 0x1F 0x2Z3 0x2A  
31  
.h2d.rb:4:in 'h2d': Not an hex number: 0x2Z3 (SyntaxError)  
from ./h2d.rb:8  
from ./h2d.rb:8:in 'each'  
from ./h2d.rb:8
```

Notas de ayuda

- La función hex puede ayudarle:

```
>> "0x1f".hex  
=> 31
```

Si tiene la documentación instalada (si no es el caso, puede encontrar la documentación de Ruby en <http://ruby-doc.org/>), puede escribir:

```
$ ri -T String#hex  
----- String#hex  
str.hex => integer  
-----  
Treats leading characters from _str_ as a string of hexadecimal  
digits (with an optional sign and an optional +0x+) and returns the  
corresponding number. Zero is returned on error.  
  
"0x0a".hex      #=> 10  
"-1234".hex     #=> -4660  
"0".hex          #=> 0  
"wombat".hex    #=> 0
```

o bien en pry:

```
[18] pry(main)> show-doc "x".hex
```

```
From: string.c (C Method):  
Owner: String  
Visibility: public  
Signature: hex()  
Number of lines: 8
```

Treats leading characters from str as a string of hexadecimal digits
(with an optional sign and an optional 0x) and returns the

corresponding number. Zero is returned on error.

```
"0xa".hex      #=> 10
"-1234".hex    #=> -4660
"0".hex        #=> 0
"wombat".hex   #=> 0
```

```
[11] pry(main)> show-method "x".hex
```

From: string.c (C Method):

Owner: String

Visibility: public

Number of lines: 10

```
static VALUE
rb_str_hex(VALUE str)
{
    rb_encoding *enc = rb_enc_get(str);

    if (!rb_enc_asciicompat(enc)) {
        rb_raise(rb_eEncCompatError, "ASCII incompatible encoding: %s", rb_enc_name(enc));
    }
    return rb_str_to_inum(str, 16, FALSE);
}
```

- *Lea la sección exceptions en rubylearning.com: Helping Ruby Programmers become Awesome! para saber mas sobre como producir/controlar una excepción.*

```
raise SyntaxError, "Not an hex number: #{h}" if h !~ /^\s*0x[\dA-F]+\s*\$/i
```

*La condición h !~ /^\s*0x[\dA-F]+\s*\\$/i se lee*

*h no casa con la expresión regular /^\s*0x[\dA-F]+\s*\\$/i.*

Esta expresión regular casa con las cadenas que comienzan por blancos, seguidos de 0x y dígitos y letras entre A y F.

```
[12] pry(main)> h = "0x1f"
=> "0x1f"
[13] pry(main)> h =~ /^\s*0x[\dA-F]+\s*\$/i
=> 0
[14] pry(main)> h = "0x0x"; h =~ /^\s*0x[\dA-F]+\s*\$/i
=> nil
[15] pry(main)> h = "0x0x4d"; h =~ /^\s*0x[\dA-F]+\s*\$/i
=> 2
[16] pry(main)> h = "0x0x4d"; h =~ /\s*0x[A-F]+\s*\$/i
=> nil
[17] pry(main)> h = "0x0xd"; h =~ /\s*0x[A-F]+\s*\$/i
=> 2
```

- *Para recorrer el array de argumentos en línea de comando podemos usar el método each:*

```
ARGV.each { |h| puts(h2d(h)) }
```

8.9. Bloques/Blocks

Ruby *Code blocks* (called *closures* in other languages) are chunks of code between braces or between `do..end` that you can associate with [method invocations](#), almost as if they were parameters.

A Ruby block is a way of grouping statements, and may appear only in the source adjacent to a method call;

the block is written starting on the same line as the method call's last parameter (or the closing parenthesis of the parameter list).

- The code in the block is not executed at the time it is encountered.
- Instead, Ruby remembers the context in which the block appears (the local variables, the current object, and so on) and then enters the method.

The Ruby standard is to use braces for single-line blocks and `do..end` for multi-line blocks.

yield

Any method can be called with a block as an implicit argument. Inside the method, you can call the block using the `yield` keyword with a value.

Blocks can have their own arguments. There are many methods in Ruby that iterate over a range of values.

```
[1] pry(main)> a = (1..8).to_a
=> [1, 2, 3, 4, 5, 6, 7, 8]
[2] pry(main)> a.each { |x| puts x*x };
1
4
9
16
25
36
49
64
```

Most of these iterators are written in such a way as to be able to take a code block as part of their calling syntax.

The method can then `yield` control to the code block (i.e. execute the block) during execution as many times as is necessary for the iteration to complete (e.g. if we are iterating over array values, we can execute the block as many times as there are array values etc.).

Once you have created a block, you can associate it with a call to a method. Usually [the code blocks passed into methods are anonymous objects](#), created on the spot.

If you provide a code block when you call a method, then inside the method, you can `yield` control to that code block - suspend execution of the method; execute the code in the block; and return control to the method body, right after the call to `yield`.

```
[~/local/src/ruby/rubytesting]$ cat block_simple.rb
def call_block
  puts "Start of method"
  yield
  yield
  puts "End of method"
end

[3] pry(main)> require_relative 'block_simple'
=> true
```

```
[4] pry(main)> call_block do puts "hello!" end
Start of method
hello!
hello!
End of method
=> nil
```

If no code block is passed, Ruby raises an exception

```
~/rubytesting]$ pry
[1] pry(main)> require_relative 'block_simple'
=> true
[2] pry(main)> call_block
Start of method
LocalJumpError: no block given (yield)
from /Users/casiano/local/src/ruby/rubytesting/block_simple.rb:3:in `call_block'
```

You can provide parameters to the call to `yield`: these will be passed to the block. Within the block, you list the names of the arguments to receive the parameters between vertical bars (`|`).

```
[~/local/src/ruby/rubytesting]$ pry
[1] pry(main)> def tutu(x)
[1] pry(main)*   "hello #{ yield x }"
[1] pry(main)* end
=> nil
[2] pry(main)> tutu('Juan') { |y| y+'a' }
=> "hello Juana"
[3] pry(main)>
```

Note that the code in the block is not executed at the time it is encountered by the Ruby interpreter. Instead, Ruby remembers the context in which the block appears and then enters the method.

```
[~/rubytesting]$ cat block_simple2.rb
y = 4

def call_block(a)
  puts "Start of method"
  puts "Inside the method a = #{a}"
  yield 4
  yield 5
  puts "End of method"
end

call_block 8 do |x|
  puts x+y
end
```

```
[~/rubytesting]$ ruby block_simple2.rb
Start of method
Inside the method a = 8
8
9
End of method
```

block_given?

`block_given?` returns `true` if `yield` would execute a block in the current context.

```
[5] pry(main)> show-source tutu
```

```
From: (pry) @ line 3:  
Owner: Object  
Visibility: public  
Number of lines: 4  
  
def tutu  
  return yield 4 if block_given?  
  "no block was given!"  
end  
[6] pry(main)> tutu { |x| x*x }  
=> 16  
[7] pry(main)> tutu  
=> "no block was given!"
```

Ambito/Scope

Let us see what happens in the following example when a variable outside a block is `x` and a block parameter is also named `x`.

```
[~/rubytesting]$ cat block_scope1.rb  
x = 10  
5.times do |x|  
  puts "x inside the block: #{x}"  
end  
  
puts "x outside the block: #{x}"  
[~/rubytesting]$ ruby block_scope1.rb  
x inside the block: 0  
x inside the block: 1  
x inside the block: 2  
x inside the block: 3  
x inside the block: 4  
x outside the block: 10
```

You will observe that after the block has executed, `x` outside the block is the original `x`. Hence the block parameter `x` was local to the block.

Since `x` is not a block parameter here, the variable `x` is the same inside and outside the block.

```
[~/rubytesting]$ cat block_scope2.rb  
x = 10  
5.times do |y|  
  x = y  
  puts "x inside the block: #{x}"  
end  
  
puts "x outside the block: #{x}"  
[~/rubytesting]$ ruby block_scope2.rb  
x inside the block: 0  
x inside the block: 1  
x inside the block: 2  
x inside the block: 3  
x inside the block: 4  
x outside the block: 4
```

In Ruby 1.9, blocks introduce their own scope for the block parameters only. This is illustrated by the following example:

```
[~/rubytesting]$ cat block_scope3.rb
x = 10
5.times do |y, x|
  x = y
  puts "x inside the block: #{x}"
end
puts "x outside the block: #{x}"
[~/rubytesting]$ ruby -v
ruby 2.1.2p95 (2014-05-08 revision 45877) [x86_64-darwin13.0]
[~/rubytesting]$ ruby block_scope3.rb
x inside the block: 0
x inside the block: 1
x inside the block: 2
x inside the block: 3
x inside the block: 4
x outside the block: 10
[~/rubytesting]$ rvm use 1.8.7-p352
Using /Users/casiano/.rvm/gems/ruby-1.8.7-p352
[~/rubytesting]$ ruby -v
ruby 1.8.7 (2011-06-30 patchlevel 352) [i686-darwin11.3.0]
[~/rubytesting]$ ruby block_scope3.rb
x inside the block: 0
x inside the block: 1
x inside the block: 2
x inside the block: 3
x inside the block: 4
x outside the block: 4
```

Ambiguedad

Keep in mind that the braces syntax has a higher precedence than the `do..end` syntax.

Braces have a high precedence; `do` has a low precedence. If the method invocation has parameters that are not enclosed in parentheses, the brace form of a block will bind to the last parameter, not to the overall invocation.

```
7] pry(main)> a = [1,2,3]
=> [1, 2, 3]
[8] pry(main)> p a.map { |n| n*2 }
[2, 4, 6]
=> [2, 4, 6]
[9] pry(main)> p a.map do |n| n*2 end
#<Enumerator: [1, 2, 3]:map>
```

En el caso `p a.map do |n| n*2 end` la sentencia se interpreta como `(p a.map) (do |n| n*2 end)`.
The `do` form will bind to the invocation.

El bloque se asocia con su método mas cercano

```
[10] pry(main)> show-source tutu
```

```
From: (pry) @ line 12:
Owner: Object
```

```

Visibility: public
Number of lines: 4

def tutu(s)
  puts "tutu"
  yield 4
end
[11] pry(main)> show-source titi

From: (pry) @ line 5:
Owner: Object
Visibility: public
Number of lines: 4

def titi
  puts "titi"
  yield 5
end
[12] pry(main)> tutu titi { |x| puts x } { |z| puts "2nd bloc #{z}" }
titi
5
tutu
2nd bloc 4
=> nil

```

Véase

- Ruby Blocks en rubylearning.com por Satish Talim
- La sección *Argumentos Bloque* en 13.4.6 dentro de la sección *Argumentos de un Método* 13.4
- La sección *Bindings (encarpetados) y eval* 15.2.1

8.10. El método gets

Si se desea leer la entrada desde STDIN podemos usar el método `gets`.
 El método `gets` puede ser utilizado para obtener una cadena desde STDIN:

```

>> z = gets
hello world!
=> "hello world!\n"

```

Observe que el retorno de carro forma parte de la cadena leída.

El método chomp Si se desea se puede eliminar el retorno de carro final de una cadena mediante el método `chomp`:

```

=> "hello world!\n"
>> z.chomp!
=> "hello world!"
>> z
=> "hello world!"

```

El operador % de formateo

```
[5] pry(main)> puts "%10s %-10s %3.2f" % ["Perez", "Hernandez", 4.5789]
Perez Hernandez 4.58
```

que es básicamente lo mismo que:

```
6] pry(main)> printf("%10s %-10s %3.2f", "Perez", "Hernandez", 4.5789)
Perez Hernandez 4.58
```

8.11. Práctica: Producto y Suma de Matrices

Se trata de desarrollar un programa que permita realizar la suma y el producto de dos matrices cuadradas. Se pide no usar la librería `Matrix`. Puede consultar una implementación en <http://rosettacode.org/wiki/Matrix>

Suponemos que las dos matrices de entrada están en un fichero. Suponemos además que cada fila ocupa una línea y que las dos matrices estan separadas entre sí por una o mas líneas en blanco.

```
[~/local/src/ruby/LPP/matrix(master)]$ cat datos.dat
1 2 3
4 5 6
```

```
1 2
3 4
5 6
```

8.11.1. Sugerencias

1. Repositorio privado en GitHub

Lectura de las matrices

Supongamos este fichero de entrada:

```
[~/local/src/ruby/LPP/matrix(master)]$ cat datos.dat
1 2 3
4 5 6
```

```
1 2
3 4
5 6
```

En una primera aproximación podríamos leer el fichero siguiendo una secuencia de comandos como esta:

```
[~/local/src/ruby/LPP/matrix(master)]$ pry
[1] pry(main)> data = File.open('datos.dat').read
=> "1 2 3\n4 5 6\n\n1 2\n3 4\n5 6\n"
[2] pry(main)> puts data
1 2 3
4 5 6

1 2
3 4
5 6
=> nil
```

La clase `File` provee los métodos para trabajar con ficheros. El método `read` esta definido en la clase `I/O` de la cual `File` hereda.

```
[5] pry(main)> File.ancestors.include? IO
=> true
```

Pero claro, tenemos que a partir de la cadena `data` obtener las dos matrices. Podemos usar `split` para obtener las dos cadenas que contienen las respectivas matrices:

```
[4] pry(main)> a, b = data.split(/\n\n+/)
=> ["1 2 3\n4 5 6", "1 2\n3 4\n5 6\n"]
[5] pry(main)> puts a
1 2 3
4 5 6
=> nil
[6] pry(main)> puts b
1 2
3 4
5 6
=> nil
```

Pero aun queda por resolver el problema de pasar de la cadena a la matriz.

Para hacerlo, primero escribimos una función `mapmap` que es una extensión de `map` para matrices:

```
def mapmap(a)
  a.map { |r|
    r.map { |e|
      yield e
    }
  }
end
```

Podemos usar `mapmap` así:

```
[~/local/src/ruby/LPP/matrix(master)]$ pry
[1] pry(main)> require './matrix'
=> true
[2] pry(main)> a = [[1,2],[3,4]]
=> [[1, 2], [3, 4]]
[3] pry(main)> mapmap(a) { |x| x*x }
=> [[1, 4], [9, 16]]
[4] pry(main)> mapmap(a) { |x| x-1 }
=> [[0, 1], [2, 3]]
[5] pry(main)>
```

Ahora podemos escribir un método `to_m` que convierte la cadena con los datos en una matriz:

```
def to_m(a)
  a = a.split(/\n/)
  a = a.map { |r| r.split(/\s+/) }
  a = mapmap(a) { |x| x.to_f }
end
```

Usando `to_m` podemos construir las matrices a partir de los datos de entrada:

```
[~/local/src/ruby/LPP/matrix(master)]$ pry
[1] pry(main)> require './matrix'
=> true
[2] pry(main)> data = File.open('datos.dat').read
=> "1 2 3\n4 5 6\n\n1 2\n3 4\n5 6\n"
```

```
[3] pry(main)> a, b = data.split(/\n\n+/)
=> ["1 2 3\n4 5 6", "1 2\n3 4\n5 6\n"]
[4] pry(main)> a = to_m(a)
=> [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]
[5] pry(main)> a[0][0].class
=> Float
```

Combinando todo lo visto, podemos escribir el método `read_matrices(fn)` que lee el fichero de datos y retorna las dos matrices:

```
def read_matrices(fn)
  text = File.open(fn).read

  a, b = text.split(/\n\n+/)
  a = to_m(a)
  b = to_m(b)

  [a, b]
end
```

Producto

Podemos ahora leer las matrices con `read_matrices`:

```
[~/local/src/ruby/LPP/matrix(master)]$ pry
[1] pry(main)> require './matrix'
=> true
[2] pry(main)> a, b = read_matrices('datos.dat')
=> [[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]], [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]]
[11] pry(main)> a
=> [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]
[12] pry(main)> b
=> [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
```

Podemos detectar el número de filas y columnas de la matriz así:

```
[3] pry(main)> a.length
=> 2
```

Supuesto que todas las filas tienen la misma longitud, el número de columnas viene dado por la longitud de cualquier fila:

```
[4] pry(main)> a[0].length
=> 3
```

Para multiplicar los elementos de la primera fila de `a` con los elementos de la primera columna de `b` podemos hacer:

```
[6] pry(main)> z = a[0].map.with_index { |x, i| x*b[i][0] }
=> [1.0, 6.0, 15.0]
```

El método `transpose` de la clase `Array` da la traspuesta de una matrix:

```
[6] pry(main)> bt = b.transpose
=> [[1.0, 3.0, 5.0], [2.0, 4.0, 6.0]]
```

Para multiplicar los elementos de la primera fila de `a` con los elementos de la primera columna de `b` podemos también hacer:

```
[8] pry(main)> z = a[0].map.with_index { |x, i| x*bt[0][i] }
=> [1.0, 6.0, 15.0]
```

Para multiplicar los elementos de la primera fila de `a` con los elementos de la segunda columna de `b` podemos hacer:

```
[9] pry(main)> z = a[0].map.with_index { |x, i| x*bt[1][i] }
=> [2.0, 8.0, 18.0]
```

Podemos usar el método `reduce` (tambien conocido como `inject`) para sumar los elementos de `z`:

```
[10] pry(main)> c01 = z.reduce(0) { |s, x| s+x }
=> 28.0
```

Por supuesto también podemos usar bucles sobre rangos como se ilustra en este ejemplo en el que multiplicamos la 2^a fila por la 2^a columna:

```
[10] pry(main)> a
=> [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]
[11] pry(main)> b
=> [[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
[12] pry(main)> s = 0
=> 0
[17] pry(main)> for k in (0...3) do s += a[1][k]*b[k][1] end
=> 0...3
[18] pry(main)> s
=> 64.0
```

8.12. Práctica: Evaluar una Expresión en Postfijo

Escriba un programa que lea una línea conteniendo una expresión en postfijo e imprima su valor.

Ejecución La entrada se lee desde línea de comandos.

```
[/rubystesting/postfix]$ ruby postfix.rb '2 3 + 4 *'
20
```

Terminales Para simplificar, asumiremos que los terminales en la expresión están separados por blancos.

Por ejemplo, ante la entrada:

```
4 5 -2 + *
```

El programa deberá imprimir 12 (esto es, se interpreta como $4*(5+(-2))$).

Excepciones Si la entrada es ilegal deberá producirse una excepción:

```
~/src/ruby/rubystesting$ ruby postfix.rb '4 3 + x2 *'
postfix.rb:12: Error. found x2. Expected number or operator (SyntaxError)
  from postfix.rb:3:in `each'
  from postfix.rb:3
```

8.12.1. Pistas

A continuación veamos algunos conocimientos que hacen falta para abordar un programa como este:

Lectura de la línea de comandos

```
[~/rubytesting/postfix]$ irb
1.9.3-p392 :001 > ARGV = ["4 3 +", "2 *"]
(irb):1: warning: already initialized constant ARGV
=> ["4 3 +", "2 *"]
1.9.3-p392 :002 > expr = ARGV.join(" ")
=> "4 3 + 2 *"
1.9.3-p392 :003 > expr
=> "4 3 + 2 *"
```

Obtener un array con los terminales: El método split

```
>> x = "2 3 4 + *".split(/\s+/)
=> ["2", "3", "4", "+", "*"]      # un array con las diferentes partes de la cadena
```

Algoritmo de evaluación de una cadena en postfijo Una vez que tenemos un array con los terminales de la cadena postfija el algoritmo es sencillo. Véase:

1. Postfix Evaluation
2. Una simulacion en YouTube

La idea es ir retirando terminales del array de terminales. Si es un operando se empuja en la pila. Si es un operador se retiran tantos operandos como aridad tiene el operador y se evalúa el operador sobre los dos operandos:

```
[~/rubytesting/postfix]$ pry
[1] pry(main)> z = ['4', '3', '+', '2', '*']
=> ["4", "3", "+", "2", "*"]
[2] pry(main)> stack = []
=> []
[3] pry(main)> d = z.shift
=> "4"
[4] pry(main)> stack.push d
=> ["4"]
[5] pry(main)> z
=> ["3", "+", "2", "*"]
[6] pry(main)> stack
=> ["4"]
[7] pry(main)> d = z.shift
=> "3"
[8] pry(main)> stack.push d
=> ["4", "3"]
[9] pry(main)> d = z.shift
=> "+"
[10] pry(main)> op2 = stack.pop
=> "3"
[11] pry(main)> op1 = stack.pop
=> "4"
[12] pry(main)> stack.push eval "#{op1} #{d} #{op2}"
=> [7]
[13] pry(main)>
```

El método eval

En la sesión `irb` que sigue puede ver como se usa el método `eval`:

```
>> a = 2
=> 2
>> eval "4+#{a}"
=> 6
```

La Sentencia case

Puede usar una sentencia `case` para analizar cada uno de los terminales. La sintáxis es:

```
~/rubytesting$ cat -n case.rb
1  #!/usr/bin/env ruby
2  def check(z)
3    case z
4      when 0 .. 2
5        "baby"
6      when *[3,4,5,6]  # La estrella '*' expande el array a una lista de argumentos 3,4,
7        "little child"
8      when 7,8,9,10,11,12
9        "child"
10     when /^1\d$/
11       "youth"
12     else
13       "adult"
14   end
15 end
16
17 age = 1
18 puts check(age)
19
20 age = 5
21 puts check(age)
22
23 age = 12
24 puts check(age)
25
26 age = "14" # para que la regexp de la linea 10 funcione, deberá ser una cadena
27 puts check(age)
```

Este programa produce como salida:

```
~/rubytesting$ ruby case.rb
baby
little child
child
youth
```

Las comparaciones en un case

Dentro de un `case` las comparaciones se hacen con el operador `==`. Así pues:

```
case expr0
when expr1, expr2
  stmt1
when expr3, expr4
```

```

stmt2
else
  stmt3
end

```

es equivalente a:

```

_tmp = expr0
if expr1 === _tmp || expr2 === _tmp
  stmt1
elsif expr3 === _tmp || expr4 === _tmp
  stmt2
else
  stmt3
end

```

```

[13] pry(main)> (1..10) === 5
=> true
[14] pry(main)> (1..10) === 12
=> false
[15] pry(main)> /^ab*$/ === 'abb'
=> true
[16] pry(main)> /^ab*$/ === 'acb'
=> false
[19] pry(main)> String === "hello"
=> true
[20] pry(main)> String === 4
=> false
[22] pry(main)> 1 === 1
=> true
[23] pry(main)> Fixnum === Fixnum
=> false
[28] pry(main)> 4.is_a? Numeric
=> true
[30] pry(main)> 4.is_a? Fixnum
=> true

```

El operador prefijo *

El operador prefijo * sobre un array expande el array en una lista de argumentos:

```

irb(main)> a = "hello"
=> "hello"
irb(main)> a[1,3] # emepezando en la posicion 1 tomar 3
=> "ell"
irb(main)> a[*[1,3]]
=> "ell"
irb(main)> def c(x,y) x+y end
=> nil
irb(main)> c(1,2)
=> 3
irb(main)> c(*[1,2])
=> 3

```

La clase Math Si quiere extender la práctica para añadir el seno, el coseno, etc. Tendrás que hacer uso del módulo Math:

```
MacBookdeCasiano:programmingRuby casiano$ irb
>> Math::PI
=> 3.14159265358979
>> x = Math.sin(Math::PI/2)
=> 1.0
>> x = Math.cos(Math::PI)
=> -1.0
```

Para tener los nombres exportados en nuestro espacio de nombres usaremos `include`:

```
>> PI
NameError: uninitialized constant PI
      from (irb):4
>> include Math
=> Object
>> PI
=> 3.14159265358979
>> x = sin(PI/2)
=> 1.0
```

8.13. Práctica: Traducción de notación infija a postfijo

Como variante de la práctica anterior puede intentar una modificación del programa que produzca como salida una expresión en infijo equivalente a la expresión postfija dada como entrada.

```
[~/rubytesting/postfix]$ ruby postfix2infix.rb '2 3 + 4 *'
((2 + 3) * 4)
```

8.14. Ejercicios

1. ¿Qué indican los siguientes prefijos/sufijos?

- \$
- @
- @@
- ?
- ! (Como en `x.sort!`)
- = (Por ejemplo en `def x= ...`)

2. ¿Qué contiene `__FILE__`?

3. ¿Qué hacen `=begin`, `=end` cuando aparecen al comienzo de una linea?

4. ¿Qué números son 010, 0x1F, 0b1111?

5. ¿Es correcto escribir subguiones en un número(p. ej. 1_000_000)?

6. Que diferencia hay entre "`\t\n`" y '`\t\n`'?

7. ¿Cómo funciona `%q`? ¿Qué es `%q{hello world\n}`? ¿Qué es `%q{'a' 'b' 'c'}`?

8. ¿Cómo funciona `%Q`? ¿Qué es `%Q{hello world\n}`? ¿Qué es `%Q{"a" "b" "c"}`?

9. ¿Que queda en c?

```
irb(main):001:0> a = 4
=> 4
irb(main):002:0> b =2
=> 2
irb(main):003:0> c = <<HERE
irb(main):004:0" --#{a}--
irb(main):005:0" --#{b}--
irb(main):006:0" HERE
```

10. ¿Que queda en c?

```
irb(main):001:0> a = 4
=> 4
irb(main):002:0> b =2
=> 2
irb(main):008:0> c = <<'HERE'
irb(main):009:0' --#{a}--
irb(main):010:0' --#{b}--
irb(main):011:0' HERE
```

11. a = 'ls'. ¿Que queda en a?

12. a = %x{ls}. ¿Que queda en a?

13. s = "hello"

- s[0,2]
- s[-1,1]
- s[0,0]
- s[0,10]
- s[s.length,1]
- s[s.length+1,1]
- s[0,-1]

14. ¿Que queda en g?

```
>> g = "hello"
=> "hello"
>> g << " world"
```

15. ¿Que queda en e?

```
>> e = '.'*3
```

16. ¿Cual es el resultado?

```
>> a = 0
=> 0
>> "#{a=a+1} "*3
```

17. ¿Cuantos elementos tiene este array? [-10..0, 0..10,]

18. ¿Que es esto? %w[this is a test]

19. ¿Que es esto? %w[\t \n]

20. ¿Que es esto? %W[\t \n]

21. ¿Que contiene **nils**?

```
nils = Array.new(3)
```

22. ¿Que contiene **zeros**?

```
zeros = Array.new(3, 0)
```

23. ¿Que queda en **b**?

```
>> x = [[1,2],[3,4]]  
=> [[1, 2], [3, 4]]  
>> b = Array.new(x)
```

24. ¿Que queda en **c**?

```
>> c = Array.new(3) { |i| 2*i }
```

25. ¿Cual es el resultado de cada una de estas operaciones?

```
>> a = ('a'...'e').to_a  
=> ["a", "b", "c", "d", "e"]  
>> a[0,0]  
=>  
>> a[1,1]  
=>  
>> a[-2,2]  
=>  
>> a[0..2]  
=>  
>> a[0...1]  
=>  
>> a[-2..-1]  
=>
```

26. ¿Cual es el resultado de cada una de estas operaciones?

```
>> a  
=> ["a", "b", "c", "d", "e"]  
>> a[0,2] = %w{A B}  
=> ["A", "B"]  
>> a  
=>  
>> a[2..5] = %w{C D E}  
=> ["C", "D", "E"]  
>> a  
=>  
>> a[0,0] = [1,2,3]  
=> [1, 2, 3]
```

```

>> a
=>
>> a[0,2] = []
=> []
>> a
=>
>> a[-1,1] = [ 'Z' ]
=> ["Z"]
>> a
=>
>> a[-2,2] = nil
=> nil
>> a
=>

```

27. ¿Cual es el resultado de cada una de estas operaciones?

```

>> a = (1...4).to_a
=>
>> a = a + [4, 5]
=>
>> a += [[6, 7, 8]]
=>
>> a = a + 9

```

28. ¿Cual es el resultado de cada una de estas operaciones?

```

>> a = []
=> []
>> a << 1
=>
>> a << 2 << 3
=>
>> a << [4, 5, 6]
=>
>> a.concat [7, 8]
=>

```

29. ¿Cual es el resultado de cada una de estas operaciones?

```

>> x = %w{a b c b a}
=>
>> x = x - %w{b c d}
=>

```

30. ¿Cual es el resultado de cada una de estas operaciones?

```

>> z = [0]*8
=>

```

31. ¿Cual es el resultado de cada una de estas operaciones?

```

>> a = [1, 1, 2, 2, 3, 3, 4]
=> [1, 1, 2, 2, 3, 3, 4]
>> b = [5, 5, 4, 4, 3, 3, 2]
=> [5, 5, 4, 4, 3, 3, 2]
>> c = a | b
=>
>> d = b | a
=>
>> e = a & b
=>
>> f = b & a
=>

```

8.15. Ejemplo en Ruby: Accediendo a Twitter

Donde Puede encontrar una copia de este proyecto/ejercicio en:

- [~/src/ruby/rubytesting/twitter/twitter-test(master)]\$ git remote -v
bb ssh://git@bitbucket.org/casiano/twitter-test.git (fetch)
bb ssh://git@bitbucket.org/casiano/twitter-test.git (push)
origin git@github.com:crguezl/twitter-test.git (fetch)
origin git@github.com:crguezl/twitter-test.git (push)
- https://github.com/crguezl/twitter-test
- https://bitbucket.org/casiano/twitter-test (privado)
- [~/src/ruby/rubytesting/twitter/twitter-test(master)]\$ pwd -P
/Users/casiano/local/src/ruby/rubytesting/twitter/twitter-test
[~/src/ruby/rubytesting/twitter/twitter-test(master)]\$ date
martes, 2 de septiembre de 2014, 12:44:00 WEST

README.md

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ cat README.md
Testing twitter
=====
```

See:

- * http://sferik.github.io/twitter/
- * Register your application in twitter: <https://dev.twitter.com/apps/new>
- * <http://www.vogella.com/articles/Git/article.html>

Once you have registered your application in twitter fill
‘configure.rb.template’ with your [Oauth] (<http://blog.varonis.com/introduction-to-oauth/>) data
and rename it to ‘configure.rb’

Para saber más del formato Markdown:

1. Markdown: Basics
2. Markdown: Syntax
3. Markdown Cheatsheet

Ejecución con rvm Usaremos Ruby Version Manager (rvm) para gestionar las diferentes versiones del intérprete ruby que mantendremos instaladas.

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ rvm list
```

```
rvm rubies
```

```
jruby-1.7.3 [ x86_64 ]
opal [ x86_64 ]
rbx-head [ x86_64 ]
ruby-1.8.7-p352 [ i686 ]
ruby-1.9.3-head [ x86_64 ]
ruby-1.9.3-p392 [ x86_64 ]
ruby-1.9.3-p545 [ x86_64 ]
ruby-2.0.0-p353 [ x86_64 ]
ruby-2.0.0-p451 [ x86_64 ]
ruby-2.1.1 [ x86_64 ]
*= ruby-2.1.2 [ x86_64 ]
```

```
# => - current
# *= - current && default
# * - default
```

Vea la sección 81 para saber mas de rvm .

El intérprete Ruby que estoy usando es:

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ ruby -v
ruby 2.1.2p95 (2014-05-08 revision 45877) [x86_64-darwin13.0]
```

En el directorio actual existe un fichero .ruby-version que controla la versión de Ruby con la que trabajaré en este directorio:

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ ls -la
total 72
drwxr-xr-x 12 casiano staff 408 2 sep 13:29 .
drwxr-xr-x  5 casiano staff 170 18 sep 2013 ..
drwxr-xr-x 14 casiano staff 476 2 sep 13:11 .git
-rw-r--r--  1 casiano staff  26 23 sep 2013 .gitignore
-rw-r--r--  1 casiano staff   6 2 sep 13:03 .ruby-version
-rw-r--r--  1 casiano staff  45 23 sep 2013 Gemfile
-rw-r--r--  1 casiano staff 319 17 sep 2013 Gemfile.lock
-rw-r--r--  1 casiano staff 398 2 sep 13:10 README.md
-rw-r--r--  1 casiano staff 213 25 sep 2013 Rakefile
-rw-r--r--  1 casiano staff 318 25 sep 2013 configure.rb
-rw-r--r--  1 casiano staff 250 17 sep 2013 configure.rb.template
-rw-r--r--  1 casiano staff 558 25 sep 2013 twitt.rb
```

El contenido de dicho fichero es la versión de Ruby que será usada cada vez que se entra en ese directorio:

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ cat .ruby-version
2.1.2
```

Antes de ejecutar nuestro programa instalamos las librerías que son necesarias para su funcionamiento. Las librerías en Ruby se denominan gemas:

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ bundle install
Fetching gem metadata from https://rubygems.org/.....
Installing multipart-post 1.2.0
Installing faraday 0.8.8
Installing multi_json 1.8.0
Installing simple_oauth 0.2.0
Installing twitter 4.8.1
Using bundler 1.6.2
Your bundle is complete!
Use 'bundle show [gemname]' to see where a bundled gem is installed.
```

Ahora podemos proceder a ejecutar el programa:

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ ruby twitt.rb
Username : timoreilly
Name      : Tim O'Reilly
Followers : 1791686
Friends   : 1238
Location  : Sebastopol, CA
URL       : http://t.co/5086iX7oyT
Verified  : true
Tweet text : @matunos @jamesoreilly @TheEconomist True enough.
Tweet time : 2014-09-02 04:39:14 +0100
Tweet ID   : 506647513076482049
```

.ruby-version Con este fichero determinamos con que versión del intérprete Ruby se trabaja en este proyecto (Véase la sección 81.7):

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ cat .ruby-version
2.1.2
```

Gemfile El fichero Gemfile es usado en conjunción con Bundler para informar de que librerías o gemas depende nuestra aplicación (véase 89):

```
[~/local/src/ruby/rubytesting/twitter(master)]$ cat Gemfile
source 'https://rubygems.org'

gem 'faraday', '0.8.7'
gem 'twitter'
```

Bundle El comando `bundle install` instala las gemas que figuran en el Gemfile y sus dependencias:

```
[~/local/src/ruby/rubytesting/twitter(master)]$ bundle install
ERROR: Gem bundler is not installed, run 'gem install bundler' first.
```

Si Bundler no esta instalado como en este ejemplo, procederemos a instalarlo mediante el comando `gem`. (Véase la sección 83 para saber mas):

```
[~/local/src/ruby/rubytesting/twitter(master)]$ gem install bundler
Fetching: bundler-1.3.5.gem (100%)
Successfully installed bundler-1.3.5
1 gem installed
```

Ahora si, instalamos las gemas con `bundle install` (véase 89):

```
[~/local/src/ruby/rubytesting/twitter(master)]$ bundle install
Fetching gem metadata from https://rubygems.org/.....
Fetching gem metadata from https://rubygems.org/..
Installing multipart-post (1.2.0)
Installing faraday (0.8.7)
Installing multi_json (1.8.0)
Installing simple_oauth (0.2.0)
Installing twitter (4.8.1)
Using bundler (1.3.5)
Your bundle is complete!
Use 'bundle show [gemname]' to see where a bundled gem is installed.
[~/local/src/ruby/rubytesting/twitter(master)]$
```

La opción `-a` de `gem search` muestra todas las versiones que existen de una gema:

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ gem search '^farad.y$' -r -a
*** REMOTE GEMS ***

faraday (0.9.0, 0.8.9, 0.8.8, 0.8.7, 0.8.6, 0.8.5, 0.8.4, 0.8.2, 0.8.1, 0.8.0, 0.7.6, 0.7.5, 0
```

Véase También Véase

1. La sección 112 en estos apuntes que describe la gema `twitter`
2. <https://github.com/crguezl/twitter-test> los fuentes de este ejemplo en GitHub
3. `twitter` gem documentación de la gema `twitter`
4. En este enlace registramos nuestra aplicación en Twitter: Register your application in twitter
5. En la rama `sinatra` de este repo `crguezl/rack-last-twits` se encuentra una versión webapp de este ejemplo, usando `sinatra`.
6. En la rama `master` de este repo `crguezl/rack-last-twits` se encuentra una versión webapp de este ejemplo, usando `rack`.
7. Un artículo sobre como usar la gema `twitter`: Data mining with Ruby and Twitter The interesting side to a Twitter API por Tim Jones en IBM developerWorks
8. Hay una versión de esta gema que provee un programa llamado `t` que nos da una interfaz de línea de comandos para acceder a `twitter`.
9. Una gema relacionada: `TweetStream` es una gema que proporciona acceso a la API de Twitter para streaming

Estructura de la Aplicación

```
[~/local/src/ruby/rubytesting/twitter(master)]$ tree
.
|--- Gemfile
|--- Gemfile.lock
|--- README.md
|--- configure.rb
`--- twitt.rb

0 directories, 6 files
```

twitt.rb

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ cat twitt.rb
require 'twitter'
require './configure'

screen_name = ARGV[0] || 'timoreilly'

client = my_twitter_client()
a_user = client.user(screen_name)

puts <<"EOS"
Username : #{a_user.screen_name}
Name     : #{a_user.name}
Followers : #{a_user.followers_count}
Friends   : #{a_user.friends_count}
Location  : #{a_user.location}
URL      : #{a_user.url ? a_user.url : ""}
Verified : #{a_user.verified}
EOS

tweet = client.user_timeline(screen_name).first

if tweet
  puts "Tweet text : #{tweet.text}"
  puts "Tweet time : #{tweet.created_at}"
  puts "Tweet ID   : #{tweet.id}"
end
```

Autentificación con OAuth. dev.twitter.com/apps Algunos métodos de la API de Twitter requieren que estemos autenticados (log in), mientras que otros están disponibles sin login. Existen dos formas de autenticarse con la API de Twitter:

1. Basic HTTP authentication
2. OAuth

Cuando se usa Basic HTTP authentication tenemos que preguntar al usuario por su username y password en Twitter. Entonces usamos el password del usuario para ejecutar nuestra aplicación. Esto es inadecuado. Es probable que el usuario desconfíe de nuestra aplicación y no esté dispuesto a darle su password a la aplicación.

OAuth es un protocolo que no requiere que el usuario le de a las aplicaciones sus passwords (de Twitter o del proveedor del servicio).

Mediante OAuth el usuario le da a tu aplicación permiso para interactuar con su cuenta de Twitter. Twitter nos da un token con el que autenticarnos y nos evita tener que preguntar o que manejar las passwords de los usuarios.

Mas formalmente, *Oauth* es un estandard abierto para autorización. OAuth proporciona un método para que los clientes accedan a los recursos del servidor en nombre de un propietario del recurso (como un usuario final).

También proporciona un proceso para los usuarios finales para autorizar acceso a terceras partes a sus recursos en el servidor sin que haya necesidad de compartir credenciales (el username y password típicos), utilizando redirecciones mediante agentes.

En otras palabras, OAuth permite a un usuario del sitio A compartir su información en el sitio A (proveedor de servicio) con el sitio B (llamado consumidor) sin compartir toda su identidad.

Una aplicación puede solicitar acceso de lectura o escritura a nuestra cuenta Twitter. Cuando le damos permiso a una aplicación, la aplicación es listada en nuestro Twitter en <https://twitter.com/account/connections>. En esta página tenemos la posibilidad de revocar el acceso a las aplicaciones a las que le hemos dado permiso en el pasado.

Para usar OAuth, es necesario registrar nuestra aplicación en twitter visitando el enlace <https://dev.twitter.com>.

Una vez llenado el formulario (deja vacío el campo *Callback URL*. Véase por ejemplo OAuth 1.0 for Web Applications en Google) Twitter nos asigna:

1. Una clave de consumidor (consumer key)
2. Un secreto de consumidor (consumer secret)

Estas claves son únicas a nuestra aplicación.

Otra cosa es que una vez registrada nuestra aplicación en Twitter desde una cuenta autorizada, el tweet contiene el nombre de nuestra aplicación y el enlace a nuestro sitio Web.

A continuación generamos el token y con ello tenemos los datos para llenar en el bloque:

```
client = Twitter::REST::Client.new do |config|
  config.consumer_key         = YOUR_CONSUMER_KEY
  config.consumer_secret     = YOUR_CONSUMER_SECRET
  config.access_token         = YOUR_OAUTH_TOKEN
  config.access_token_secret = YOUR_OAUTH_TOKEN_SECRET
end
```

Véase la sección 93 OAuth en estos apuntes para saber mas sobre OAuth .

configure.rb Copia y pega tus datos OAuth :

```
[~/src/ruby/rubytesting/twitter/twitter-test(master)]$ cat configure.rb
def my_twitter_client
  Twitter::REST::Client.new do |config|
    config.consumer_key         = '*****'
    config.consumer_secret     = '*****'
    config.access_token         = '*****'
    config.access_token_secret = '*****'
  end
end
```

8.16. Práctica: Contar la Popularidad de Nuestros Amigos en Twitter

Escriba una aplicación que reciba como argumento un usuario de Twitter y muestre como resultado la lista de amigos de ese usuario (esto es, a quienes sigue) ordenada según su popularidad (esto es, según el número de seguidores que tiene). Del mas popular al menos popular.

Sigue un ejemplo de ejecución:

```
~/local/src/ruby/rubytesting/twitter/popular_friend(master)]$ ruby popular.rb crondinosaurio 3
kytrinyx          3252
geraldine_ms      436
frangarciar       375
elcocutw          70
```

8.17. Véase También

- Dave Thomas - RubyConf AU 2013 Closing Keynote (Vimeo)
- Escuche la entrevista con Matz *Interview between Yukihiko Matsumoto, creator of the Ruby programming language and Thomas Frey.* (Vimeo)

8.18. Resolviendo Sudokus

Véase:

- <http://websudoku.com/>

```
~/rubytesting$ cat -n useSudoku.rb
1  #!/usr/bin/env ruby
2  # http://www.websudoku.com/
3  $LOAD_PATH.unshift 'RPLExamples'
4  require "Sudoku"
5  puts Sudoku.solve Sudoku::Puzzle.new ARGF.readlines
```

```
~/rubytesting$ cat -n RPLExamples/sudoku.txt
1  3.4..5.61
2  ....9.8.3
3  .1..8...2
4  4...37.5.
5  .6.9.8.2.
6  .7.54...9
7  1...7...9.
8  7.8.6.....
9  29.4..3.7
```

```
~/rubytesting$ ./useSudoku.rb RPLExamples/sudoku.txt
384725961
652194873
917386542
429637158
563918724
871542639
145873296
738269415
296451387
```

```
~/rubytesting$ cat -n RPLExamples/Sudoku.rb
1  # http://www.websudoku.com/
2  module Sudoku
3
4    class Puzzle
5
6      ASCII = ".123456789"
7      BIN = "\000\001\002\003\004\005\006\007\010\011"
8
9      def initialize(lines)
10        if (lines.respond_to? :join)
11          s = lines.join
12        else
```

```

13     s = lines.dup
14   end
15
16   s.gsub!(/\s/, "")
17
18   raise Invalid, "Grid is the wrong size" unless s.size == 81
19
20   raise Invalid, "Illegal character #{s[i,1]} in puzzle" if i = s.index(/[^123456789])
21
22   s.tr!(ASCII, BIN)      # Translate ASCII characters into bytes
23   @grid = s.unpack('c*') # Now unpack the bytes into an array of numbers
24
25   raise Invalid, "Initial puzzle has duplicates" if has_duplicates?
26 end
27
28 def to_s
29   (0..8).collect{|r| @grid[r*9..(r+1)*9].pack('c9')}.join("\n").tr(BIN,ASCII)
30 end
31
32 def dup
33   copy = super          # Make a shallow copy by calling Object.dup
34   @grid = @grid.dup    # Make a new copy of the internal data
35   copy                 # Return the copied object
36 end
37
38 def [](row, col)
39   @grid[row*9 + col]
40 end
41
42 def []=(row, col, newvalue)
43   raise Invalid, "illegal cell value" unless (0..9).include? newvalue
44   @grid[row*9 + col] = newvalue
45 end
46
47 BoxOfIndex = [
48   0,0,0,1,1,1,2,2,2,
49   0,0,0,1,1,1,2,2,2,
50   0,0,0,1,1,1,2,2,2,
51   3,3,3,4,4,4,5,5,5,
52   3,3,3,4,4,4,5,5,5,
53   3,3,3,4,4,4,5,5,5,
54   6,6,6,7,7,7,8,8,8,
55   6,6,6,7,7,7,8,8,8,
56   6,6,6,7,7,7,8,8,8
57 ].freeze
58
59 def each_unknown
60   0.upto 8 do |row|           # For each row
61     0.upto 8 do |col|         # For each column
62       index = row*9+col      # Cell index for (row,col)
63       next if @grid[index] != 0 # Move on if we know the cell's value
64       box = BoxOfIndex[index] # Figure out the box for this cell
65       yield row, col, box     # Invoke the associated block

```

```

66         end
67     end
68 end
69
70 def has_duplicates?
71   0.upto(8) { |row| return true if rowdigits(row).uniq! }
72   0.upto(8) { |col| return true if coldigits(col).uniq! }
73   0.upto(8) { |box| return true if boxdigits(box).uniq! }
74
75   false # If all the tests have passed, then the board has no duplicates
76 end
77
78 AllDigits = [1, 2, 3, 4, 5, 6, 7, 8, 9].freeze
79
80 def possible(row, col, box)
81   AllDigits - (rowdigits(row) + coldigits(col) + boxdigits(box))
82 end
83
84
85 def rowdigits(row)
86   @grid[row*9..row*9+8] - [0]
87 end
88
89 def coldigits(col)
90   result = [] # Start with an empty array
91   col.step(80, 9) { |i| # Loop from col by nines up to 80
92     v = @grid[i] # Get value of cell at that index
93     result << v if (v != 0) # Add it to the array if non-zero
94   }
95   result # Return the array
96 end
97
98 BoxToIndex = [0, 3, 6, 27, 30, 33, 54, 57, 60].freeze
99
100 def boxdigits(b)
101   i = BoxToIndex[b]
102   [
103     @grid[i],    @grid[i+1],    @grid[i+2],
104     @grid[i+9],  @grid[i+10],   @grid[i+11],
105     @grid[i+18], @grid[i+19],   @grid[i+20]
106   ] - [0]
107 end
108 end # This is the end of the Puzzle class
109
110 class Invalid < StandardError
111 end
112
113 class Impossible < StandardError
114 end
115
116 def Sudoku.scan(puzzle)
117   unchanged = false # This is our loop variable
118

```

```

119     until unchanged
120         unchanged = true      # Assume no cells will be changed this time
121         rmin,cmin,pmin = nil # Track cell with minimal possible set
122         min = 10            # More than the maximal number of possibilities
123
124     puzzle.each_unknown do |row, col, box|
125         p = puzzle.possible(row, col, box)
126
127         case p.size
128         when 0 # No possible values means the puzzle is over-constrained
129             raise Impossible
130         when 1 # We've found a unique value, so set it in the grid
131             puzzle[row,col] = p[0] # Set that position on the grid to the value
132             unchanged = false    # Note that we've made a change
133         else    # For any other number of possibilities
134             if unchanged && p.size < min
135                 min = p.size          # Current smallest size
136                 rmin, cmin, pmin = row, col, p # Note parallel assignment
137             end
138         end
139     end
140 end
141
142     return rmin, cmin, pmin
143 end
144
145 def Sudoku.solve(puzzle)
146     puzzle = puzzle.dup
147
148     r,c,p = scan(puzzle)
149
150     return puzzle if r == nil
151
152     p.each do |guess|      # For each value in the set of possible values
153         puzzle[r,c] = guess # Guess the value
154
155         begin
156             return solve(puzzle) # If it returns, we just return the solution
157             rescue Impossible
158                 next           # If it raises an exception, try the next guess
159             end
160         end
161
162         raise Impossible
163     end
164 end

```

Capítulo 9

La Estructura y Ejecución de los Programas Ruby

9.1. Estructura Léxica

9.2. Estructura Sintáctica

9.3. Estructura de Fichero

9.4. Codificación de un Programa

9.5. Ejecución de un Programa

9.6. Práctica: Descarga Páginas de la Wikipedia

Implemente el código descrito en la sección Downloading Wikipedia pages del libro *The Bastards Book of Ruby, A Programming Primer for Counting and Other Unconventional Tasks* [1].

9.7. Práctica: Tweet Fetching

Implemente el código descrito en el capítulo Tweet fetching sección Tweet Fecthing del libro *The Bastards Book of Ruby, A Programming Primer for Counting and Other Unconventional Tasks* [1].

9.8. Práctica: SQL An introduction to relational databases and their query languages

Implemente el código descrito en la sección SQL An introduction to relational databases and their query languages del libro *The Bastards Book of Ruby, A Programming Primer for Counting and Other Unconventional Tasks* [1].

Capítulo 10

Tipos de Datos y Objetos

10.1. Números

10.1.1. Constantes Racionales en Ruby 2.1

You might be aware that floats are far from ideal when doing calculations with fixed decimal points:

```
irb(main):001:0> 0.1 * 3  
=> 0.3000000000000004
```

A lot of Ruby developers fall back to using integers and only fixing it when displaying the result.

However, this only works well if you have indeed have fixed decimal points.

If not, you have to use rationals. Which isn't too bad, except there was no nice syntax for them (short of changing the return value of `Integer#%`).

Ruby 2.1 introduces the `r` suffix for decimal/rational literals to fix this:

```
irb(main):001:0> 0.1r  
=> (1/10)  
irb(main):002:0> 0.1r * 3  
=> (3/10)
```

10.2. Texto

10.3. Arrays

10.4. Hashes

10.5. Rangos

10.6. Símbolos

10.7. Booleanos y Nil

10.7.1. Véase También

- Null Objects and Falsiness by Avdi Grimm
- A quick intro to Naught por Avdi Grimm
- NullAndVoid
- Null Object: Something for Nothing (PDF) by Kevlin Henny

- The Null Object Pattern (PS) por Bobby Woolf
- NullObject en WikiWiki
- Wikipedia Null Object pattern

10.8. Objetos

10.8.1. Referencias a Objetos

Cuando asignamos un valor a una variable no estamos copiando un objeto en esa variable, sólo estamos almacenando una referencia al objeto en esa variable:

```
>> person1 = "Tim"
=> "Tim"
>> person1.class
=> String
>> person1.object_id
=> 2150445000

>> person2 = person1
=> "Tim"
>> person1[0] = "J"
=> "J"
>> person2
=> "Jim"
```

El método `dup` puede ser usado para duplicar un objeto:

```
>> person2 = person1.dup
=> "Jim"
>> person1[0] = "T"
=> "T"
>> person2
=> "Jim"
>> person1
=> "Tim"
```

Alternativamente podemos usar el método `clone`:

```
>> person2 = person1.clone
=> "Tim"
>> person1[0] = "W"
=> "W"
>> person2
=> "Tim"
>> person1
=> "Wim"
```

Valores Inmediatos

Los objetos `Fixnum` son valores inmediatos, no son referencias. Son objetos inmutables.

```
[16:51] [~/Dropbox/Public/LPP/1213(master)]$ irb
ruby-1.9.2-p290 :001 > a = 2
=> 2
ruby-1.9.2-p290 :002 > b = a
```

```
=> 2
ruby-1.9.2-p290 :003 > b = 5
=> 5
ruby-1.9.2-p290 :004 > a
=> 2
ruby-1.9.2-p290 :005 > b
=> 5
```

Los objetos `Symbol` son valores inmediatos, no son referencias. Son objetos inmutables.

```
ruby-1.9.2-p290 :006 > c = :z
=> :z
ruby-1.9.2-p290 :007 > d = c
=> :z
ruby-1.9.2-p290 :008 > d = :w
=> :w
ruby-1.9.2-p290 :009 > c
=> :z
ruby-1.9.2-p290 :010 > d
=> :w
```

10.8.2. Vida de un Objeto

Objects of most classes need to be explicitly created, and this is most often done with a method named `new`:

```
myObject = myClass.new
```

`new` is a method of the `Class` class.

1. It allocates memory to hold the new object, then it initializes the state of that newly allocated *empty* object by invoking its `initialize` method.
2. The arguments to `new` are passed directly on to `initialize`. Most classes define an `initialize` method to perform whatever initialization is necessary for instances.
3. The `new` and `initialize` methods provide the default technique for creating new objects, but classes may also define other methods, known as *factory methods* that return instances.
4. Ruby objects never need to be explicitly deallocated, as they do in languages like C and C++. Ruby uses a technique called *garbage collection* to automatically destroy objects that are no longer needed.
5. An object becomes a candidate for garbage collection when it is unreachable: when there are no remaining references to the object except from other unreachable objects.
6. Garbage collection does not mean that memory leaks are impossible: any code that creates long-lived references to objects that would otherwise be short-lived can be a source of memory leaks.

10.8.3. Identidad de un Objeto

Every object has an object identifier, a `Fixnum`, that you can obtain with the `object_id` method. The value returned by this method is constant and unique for the lifetime of the object. While the object is accessible, it will always have the same ID, and no other object will share that ID.

```

ruby-1.9.2-p290 :003 > a = 4
=> 4
ruby-1.9.2-p290 :004 > a.__id__
=> 9
ruby-1.9.2-p290 :005 > a.object_id
=> 9
ruby-1.9.2-p290 :006 > x = "hello"
=> "hello"
ruby-1.9.2-p290 :007 > x.__id__
=> 70103467703860
ruby-1.9.2-p290 :008 > x.object_id
=> 70103467703860

```

Es único durante la vida del objeto.

10.8.4. Clase y Tipo de un Objeto. Polimorfismo

To determine the class of an object in Ruby, simply ask for it:

```

o = "test" # This is a value
o.class      # Returns an object representing the String class

```

We can ask any class what its superclass is:

```

o.class                      # String: o is a String object
o.class.superclass            # Object: superclass of String is Object
o.class.superclass.superclass # nil: Object has no superclass

```

In Ruby 1.9 and later, Object is no longer the root of the class hierarchy:

```

Object.superclass          # BasicObject: Object has a superclass in 1.9
BasicObject.superclass     # nil: BasicObject has no superclass

```

So a particularly straightforward way to check the class of an object is by direct comparison:

```

o.class == String      # true if o is a String

```

The `instance_of?` method does the same thing and is a little more elegant:

```

o.instance_of? String    # true if o is a String

```

Usually when we test the class of an object, we would also like to know if the object is an instance of any subclass of that class. To test this, use the `is_a?` method, or its synonym `kind_of?`:

```

x = 1                      # This is the value we're working with
x.instance_of? Fixnum        # true: is an instance of Fixnum
x.instance_of? Numeric       # false: instance_of? doesn't check inheritance
x.is_a? Fixnum              # true: x is a Fixnum
x.is_a? Integer             # true: x is an Integer
x.is_a? Numeric             # true: x is a Numeric
x.is_a? Comparable          # true: works with mixin modules, too
x.is_a? Object               # true for any value of x

```

The `Class` class defines the `==` operator in such a way that it can be used in place of `is_a?`:

```

Numeric === x                # true: x is_a Numeric

```

Clase y Vida de un Objeto

Every object has a well-defined class in Ruby, and that class never changes during the lifetime of the object.

Que es un Tipo When we talk about the *type* of an object, we mean *the set of behaviors that characterize the object*

Another way to put it is that *the type of an object is the set of methods it can respond to*

(This definition becomes recursive because it is not just the name of the methods that matter, but also the types of arguments that those methods can accept.)

¿Puedo llamar a este método sobre este objeto?

In Ruby programming, we often don't care about the class of an object, we just want to know whether we can invoke some method on it.

1. Consider, for example, the << operator.

Arrays, strings, files, and other I/O-related classes define this as an append operator.

If we are writing a method that produces textual output, we might write it generically to use this operator.

2. Then our method can be invoked with any argument that implements <<. We don't care about the class of the argument, just that we can append to it. We can test for this with the `respond_to?` method:

```
o.respond_to? :"<<" # true if o has an << operator
```

3. Este método funciona tanto con cadenas como con arrays, listas, etc. es un método polimorfo.
Polymorphism is the provision of a single interface to entities of different types.

The shortcoming of this approach is that it only checks the name of a method, not the arguments for that method:

1. For example, `Fixnum` and `Bignum` implement << as a left-shift operator and expect the argument to be a number instead of a string.
2. Integer objects appear to be *appendable* when we use a `respond_to?` test, but they produce an error when our code appends a string.

There is no general solution to this problem, but an ad-hoc remedy, in this case, is to explicitly rule out `Numeric` objects with the `is_a?` method:

```
o.respond_to? :"<<" and not o.is_a? Numeric
```

Another example of the *type-versus-class* distinction is the `StringIO` class (from Ruby's standard library).

1. `StringIO` enables reading from and writing to string objects as if they were `IO` objects.
2. `StringIO` mimics the `IO` API—`StringIO` objects define the same methods that `IO` objects do.
3. But `StringIO` is not a subclass of `IO`.
4. If you write a method that expects a stream argument, and test the class of the argument with `is_a? IO`, then your method won't work with `StringIO` arguments.

Focusing on types rather than classes leads to a programming style known in Ruby as "duck typing."

Repasa la sección *Comprobación de Tipos y Tipado Pato (Duck Typing)* 14.1.8.

El tipo de un objeto es el conjunto de conductas que caracterizan al objeto. Es el conjunto de métodos a los que puede responder.

10.8.5. Igualdad de Objetos

El método equal?

Usado para comprobar cuando se refieren al mismo objeto.

The `equal?` method is defined by `Object` to test whether two values refer to exactly the same object. For any two distinct objects, this method always returns false:

```
ruby-1.9.2-p290 :009 > a = "hello"
=> "hello"
ruby-1.9.2-p290 :010 > b = c = "hello"
=> "hello"
ruby-1.9.2-p290 :011 > a.equal? b
=> false
ruby-1.9.2-p290 :012 > b.equal? c
=> true
```

By convention, subclasses never override the `equal?` method.

Another way to determine if two objects are, in fact, the same object is to check their `object_id`:

```
a.object_id == b.object_id    # Works like a.equal?(b)
```

El operador ==

The `==` operator is the most common way to test for equality. In the `Object` class, it is simply a synonym for `equal?`, and it tests whether two object references are identical.

Most classes redefine this operator to allow distinct instances to be tested for equality:

```
ruby-1.9.2-p290 :013 > a = "hello"
=> "hello"
ruby-1.9.2-p290 :014 > b = c = "hello"
=> "hello"
ruby-1.9.2-p290 :015 > a == b
=> true
```

El método eql?

The `eql?` method is defined by `Object` as a synonym for `equal?`. Classes that override it typically use it as a strict version of `==` that does no type conversion.

```
ruby-1.9.2-p290 :019 > 1 == 1.0
=> true
ruby-1.9.2-p290 :020 > 1.eql? 1.0
=> false
```

The `Hash` class uses `eql?` to check whether two hash keys are equal.

If two objects are `eql?`, their `hash` methods must also return the same value.

El operador ===

The `==` operator is commonly called the *case equality* operator and is used to test whether the target value of a `case` statement matches any of the `when` clauses of that statement.

Range defines `==` to test whether a value falls within the range.

```
ruby-1.9.2-p290 :021 > (1..10) === 5
=> true
```

Regexp defines `==` to test whether a string matches the regular expression.

```
ruby-1.9.2-p290 :022 > /\d+/ === "a432b"  
=> true  
ruby-1.9.2-p290 :023 > /\d+/ === "acb"  
=> false
```

And Class defines `==` to test whether an object is an instance of that class.

```
ruby-1.9.2-p290 :024 > String === 's'  
=> true  
ruby-1.9.2-p290 :025 > String === 32  
=> false
```

El operador `=~`

The `=~` operator is defined by String and Regexp (and Symbol in Ruby 1.9) to perform pattern matching.

10.8.6. Orden en Objetos

10.8.7. Conversión de Objetos

10.8.8. Copia de Objetos

10.8.9. Marshalling

Serializar un objeto es convertirlo en una cadena de bytes con la idea de almacenarlo para su uso posterior o para su uso por otro programa o proceso. Podemos utilizar el módulo Marshal para ello.

Podemos usar marshalling para guardar el estado de un objeto `o`: en un fichero `objmsh`

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n marshal2.rb  
1  require "Klass"  
2  
3  o = Klass.new("hello", { :a => 1, :b => 2} )  
4  File.open("objmsh", "wb") do |f|  
5    data = Marshal.dump(o, f)  
6  end
```

La clase `Klass` esta definida así:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n Klass.rb  
1  class Klass  
2    def initialize(str, hash)  
3      @str, @hash = str, hash  
4    end  
5  
6    def to_s  
7      h = @hash.keys.map { |x| "#{x} => #{@hash[x]}"} .join(" ")  
8      "str = #{@str} hash = #{@hash}"  
9    end  
10   end
```

Podemos cargar posteriormente el objeto en la ejecución de un script usando `Marshal.load`. De este modo es posible guardar el estado de un programa entre ejecuciones:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n marshal3.rb
1  require "Klass"
2
3  obj = nil
4  File.open("objmsh","rb") {|f| obj = Marshal.load(f)}
5  puts obj.inspect
6  puts obj
```

La ejecución de `marshal2.rb` serializa el objeto y lo guarda en un fichero:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby marshal2.rb
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ls -ltr | tail -1
-rw-r--r-- 1 casianorodriguezleon staff 43 13 oct 21:29 objmsh
```

La ejecución de `marshal3.rb` reconstruye el objeto usando `Marshal.load`:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby marshal3.rb
#<Klass:0x10016a020 @hash={:a=>1, :b=>2}, @str="hello">
str = 'hello' hash = {a => 1 b => 2}
```

Marshalling Datos entre Procesos que se Comunican

```
MacBookdeCasiano:distributedRuby casiano$ cat -n fork.rb
1  #!/usr/bin/env ruby -w
2  def do_in_child
3    read, write = IO.pipe
4
5    pid = fork do
6      read.close
7      result = yield
8      write.puts [Marshal.dump(result)].pack("m")
9      exit!
10   end
11
12   write.close
13   puts "In father. PID = #{Process.pid}"
14   result = read.read
15   Process.wait2(pid)
16   r = Marshal.load(result.unpack("m")[0])
17   puts "#{Process.pid}: #{r.inspect}"
18 end
19
20 do_in_child do
21   puts "In child. PID = #{Process.pid}"
22   { :a => 1, :b => 2 }
23 end
```

- El método `pipe` de `IO` crea un canal y retorna los extremos del mismo. Es necesario que los dos procesos que usan la pareja para comunicarse cierren el extremo del canal que no usen. El extremo de lectura de un canal no generará un final de fichero si hay escritores que tienen el canal abierto. En el caso del padre, el `read` no terminaría nunca si no cierra primero el canal de escritura.
- En la línea 5, la llamada al método `fork` de la clase `Process` hace que se cree un proceso hijo. El `fork` retorna al proceso padre el PID del proceso hijo. El proceso hijo recibe un `nil`. La sintaxis de `fork` es:

```
fork [{ block }] → fixnum or nil
```

Si se especifica un bloque - como ocurre en el ejemplo - es el código de ese bloque el que se ejecuta como subprocesso. En tal caso el fin del bloque indica el final del proceso que por defecto termina con un estatus 0. Por tanto, en este caso, la llamada a `exit!` es un exceso y podría ser eliminada, pero es la forma de hacer explícita la terminación del proceso. El proceso padre deberá esperar por la terminación de los procesos hijos usando un `wait` o bien no esperar usando un `detach` para hacer explícito su desinterés en el estatus del hijo. La no realización de este protocolo puede conducir a la proliferación de zombies.

Se puede ver si `fork` está disponible consultando `Process.respond_to?(:fork)`.

- El método `pack` de la clase `Array` tiene la sintaxis:

```
arr.pack ( aTemplateString ) -> aBinaryString
```

La llamada empaqueta los contenidos de `arr` en una secuencia binaria de acuerdo con las directivas especificadas en `aTemplateString`.

Así en la línea:

```
write.puts [Marshal.dump(result)].pack("m")
```

El formato "`m`" en el parámetro de template indica que la cadena binaria producida por `Marshal.dump(result)` debe ser codificada en Base64.

Base64 denota un grupo de esquemas de codificación que representan datos binarios en una cadena en formato ASCII. Estos esquemas se usan cuando hay necesidad de codificar datos binarios que deben ser almacenados y transferidos sobre un medio que fué diseñado para tratar con datos de tipo texto. De esta forma se asegura que los datos permanecen intactos durante el transporte. Entre otras aplicaciones, es habitual ver el uso de Base64 en el uso de email via MIME y en el almacenamiento de datos complejos en XML.

En este ejemplo el uso de `pack` y `unpack` parece innecesario ya que el canal está ya en un modo de transmisión binario de manera que caracteres como el retorno de carro o el tabulador no se interpreten. Se puede eliminar el uso de `pack` y `unpack` si se usa `syswrite`:

```
8     write.syswrite Marshal.dump(result)
```

y ahora la lectura y reconstrucción de la estructura queda simplificada:

```
14   result = read.read
15   Process.wait2(pid)
16   r = Marshal.load(result)
```

- El método `read` de la clase `IO`

```
read([length [, buffer]]) => string, buffer, or nil
```

lee a lo mas `length` bytes del stream de I/O, o hasta el final de fichero si se omite o es `nil`. Si se especifica el `buffer`, lo leído se guarda en el mismo. Cuando el método `read` alcanza el final de fichero retorna `nil` o `""`.

- La llamada al método `exit!` de la clase `Process` termina inmediatamente el proceso hijo.
- La llamada a `wait2` hace que el proceso padre espere a la salida del proceso hijo. El método `wait2` retorna un array que contiene el PID del proceso hijo y su estatus de salida (que es un objeto `Process::Status`).

Cuando se ejecuta el programa anterior produce una salida parecida a esta:

```
MacBookdeCasiano:distributedRuby casiano$ ./fork.rb
In father. PID = 6583
In child. PID = 6584
6583: {:a=>1, :b=>2}
```

Ejercicios

1. ¿Que contiene la variable global \$\$?
2. ¿Que contiene la variable global \$\$? ¿Cual será la salida? (Véase Process::Status).

```
>> 'ls noexiste'
ls: noexiste: No such file or directory
=> ""
>> $?
=> #<Process::Status: pid=1717,exited(1)>
>> $? .exitstatus
=>
>> $? .pid
=>
>> $? .signaled?
=> false
>> $? .success?
=>
>> $? .termsig
=> nil
```

3. ¿Cual será la salida de este programa?

```
if (pid = fork) then puts $$ else puts $$ end
```

4. ¿Cual será la salida de este programa?

```
1  a = 4
2  if (pid = fork)
3    puts "#{$$}: #{a}"
4  else
5    puts "#{$$}: #{a}"
6    a = 5
7    puts "#{$$}: #{a}"
8    exit!
9  end
10 Process.wait2
11 puts "#{$$}: #{a}"
```

5. El programa unix bc implementa una calculadora. He aqui un ejemplo de sesión

```
~/rubytesting/distributedRuby$ bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty'.
```

```

a = 2*-4
a
-8
b = 9+a
b
1
^D
~/rubytesting/distributedRuby$
```

¿Cuál es la salida del siguiente programa? Véase la documentación de `popen`.

```

~/rubytesting/distributedRuby$ cat -n bc.rb
 1 IO.popen("bc", "r+") do |pipe|
 2   pipe.puts("a = 2*-4")
 3   pipe.puts("a")
 4   output = pipe.gets
 5   puts output
 6
 7   pipe.puts("b = 9+a")
 8   pipe.puts("b")
 9   pipe.close_write
10   output = pipe.gets
11   puts output
12 end
```

Marshalling con YAML

La librería YAML nos permite cierta interoperabilidad entre lenguajes.

La interoperabilidad entre lenguajes es la posibilidad de que el código interactúe con código escrito en un lenguaje de programación diferente. La interoperabilidad entre lenguajes puede ayudar a maximizar la reutilización de código y, por tanto, puede mejorar la eficacia del proceso de programación.

Aquí tenemos un ejemplo:

```

~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n marshallingWithYAML
 1 require 'YAML'
 2
 3 fred = {}
 4 fred['name'] = "Fred Astair"
 5 fred['age'] = 54
 6
 7 laura = {}
 8 laura['name'] = "Laura Higgins"
 9 laura['age'] = 45
10
11 test_data = [ fred, laura ]
12
13 puts "The array dumped by Ruby in YAML format:\n"+YAML::dump(test_data)
14
15 IO.popen('perl perlscript.pl', "w+") do |pipe|
16   pipe.puts YAML::dump(test_data)
17   pipe.close_write
18   output = pipe.read
19   puts "The perl script produced this output:\n<<\n#{output}>>"
```

Este es el código del script Perl arrancado desde Ruby:

```
/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n perlscript.pl
1  use strict;
2  use warnings;
3  $| = 1;
4
5  use YAML;
6
7  $/ = undef;
8  my $rubypersons = <STDIN>;
9  my $perlpersons = Load($rubypersons);
10 for (@$perlpersons) {
11   print($_->{name}, "\n");
12 }
```

Este es el resultado de la ejecución:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby marshallWithYAML.r
The array dumped by Ruby in YAML format:
---
- name: Fred Astair
  age: 54
- name: Laura Higgins
  age: 45
The perl script produced this output:
<<
Fred Astair
Laura Higgins
>>
```

Marshalling con PStore

La librería PStore nos permite almacenar y recuperar estructuras de datos en un fichero.

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n trivialclass.rb
1  class Person
2    attr_accessor :name, :job, :gender, :age
3  end

~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n marshalWithPStore.r
1  require "trivialclass"
2  require "pstore"
3
4  fred = Person.new
5  fred.name = 'Fred Bloggs'
6  fred.age = 54
7
8  laura = Person.new
9  laura.name = "Laura Higgins"
10 laura.age = 45
11
12 store = PStore.new("persistentobjects")
13 store.transaction do
14   store[:people] ||= []
15   store[:people] << fred << laura
16 end
```

El método `transaction(read_only=false)` abre una nueva transacción del almacenamiento de datos. El código que se pasa como bloque puede leer y escribir datos en el fichero. El final del bloque produce un commit automático de los cambios. Es posible producir explícitamente el final de la transacción llamando a `PStore::commit`:

```
require "pstore"

store = PStore.new("data_file.pstore")
store.transaction do # begin transaction
  # load some data into the store...
  store[:one] = 1
  store[:two] = 2

  store.commit      # end transaction here, committing changes

  store[:three] = 3 # this change is never reached
end
```

también es posible finalizar la transacción llamando a `abort`:

```
require "pstore"

store = PStore.new("data_file.pstore")
store.transaction do # begin transaction
  store[:one] = 1      # this change is not applied, see below...
  store[:two] = 2      # this change is not applied, see below...

  store.abort         # end transaction here, discard all changes

  store[:three] = 3   # this change is never reached
end
```

Si se genera una excepción dentro del bloque se produce una llamada a `abort`.

Si el argumento `read_only` es `true`, sólo se podrá acceder para lectura al almacen de datos durante la transacción y cualquier intento de cambiarlo producirá una excepción `PStore::Error`.

Nótese que `PStore` no soporta transacciones anidadas.

El siguiente script recupera los datos almacenados:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n useMarshalWithPStor
1  require "trivialclass"
2  require "pstore"
3  store = PStore.new("persistentobjects")
4  people = []
5  store.transaction do
6    people = store[:people]
7  end
8  people.each { |o|
9    puts o.inspect
10 }
```

Sigue un ejemplo de ejecución:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby marshalWithPStore.rb
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ls -ltr | tail -1
-rw-r--r-- 1 casianorodriguezleon staff    77 16 oct 12:48 persistentobjects
```

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby useMarshalWithPStore.
#<Person:0x10036ee48 @name="Fred Bloggs", @age=54>
#<Person:0x10036ed80 @name="Laura Higgins", @age=45>
```

Ejercicio 10.8.1. Si se ejecuta por segunda vez:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby marshalWithPStore.rb
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ls -ltr | tail -1
-rw-r--r-- 1 casianorodriguezleon staff 125 16 oct 12:49 persistentobjects
```

Obtenemos:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby useMarshalWithPStore.
#<Person:0x10036ee48 @name="Fred Bloggs", @age=54>
#<Person:0x10036ed80 @name="Laura Higgins", @age=45>
#<Person:0x10036ed08 @name="Fred Bloggs", @age=54>
#<Person:0x10036ec90 @name="Laura Higgins", @age=45>
```

¿Cuál es la razón?

Marshalling Lambdas y Procs

Los métodos `dump` y `load` no funcionan con objetos de la clase `Proc`. Cuando ejecutamos este programa:

```
~/rubytesting/distributedRuby/serializeProc$ cat -n marshal2.rb
1 o = Proc.new { |a,b| a+b }
2 File.open("objmsh", "wb") do |f|
3   data = Marshal.dump(o, f)
4 end
```

Obtenemos un mensaje de error:

```
~/rubytesting/distributedRuby/serializeProc$ ruby marshal2.rb
marshal2.rb:3:in `dump': no marshal_dump is defined for class Proc (TypeError)
 from marshal2.rb:3
 from marshal2.rb:2:in `open'
 from marshal2.rb:2
```

Algunos objetos no pueden ser volcados. Si nuestra clase tiene necesidades especiales debemos implementar nuestra propia estrategia de serialización definiendo dos métodos:

1. `_dump`

El método de instancia `_dump` debe retornar una cadena `String` que contenga la información necesaria para la reconstrucción del objeto hasta la profundidad indicada por un parámetro entero. Un valor de `-1` de dicho parámetro indica que desactivamos la comprobación de profundidad.

2. `_load`

El método de clase `_load` toma una `String` y devuelve el objeto reconstruido.

Veamos un ejemplo de uso de `_dump` y `_load`. La clase `SerializableProc` permite serializar Procs:

```
~/rubytesting/distributedRuby/serializeProc$ cat -n SerializableProc.rb
1 class SerializableProc
2
3   def initialize( block )
4     @block = block
```

```

5      @block.sub!(/~/,'lambda ') unless @block =~/^\s*(?:lambda|proc)/
6      # Test if block is valid.
7      @func = eval "#{@block}"
8  end
9
10
11  def call(* args)
12      @func.call(* args)
13  end
14
15  def arity
16      @func.arity
17  end
18
19  def _dump(limit)
20      @block
21  end
22
23  def self._load(s)
24      self.new(s)
25  end
26
27 end

```

Este código hace uso de la clase:

```

~/rubytesting/distributedRuby/serializeProc$ cat -n marshalproc2.rb
1
2  require "SerializableProc"
3
4
5  if $0 == __FILE__
6
7      code = SerializableProc.new %q{ { |a,b| a+b } }
8
9      # Marshal
10     File.open('proc.marshalled', 'w') { |file| Marshal.dump(code, file) }
11     code = File.open('proc.marshalled') { |file| Marshal.load(file) }
12
13     p code.call( 1, 2 )
14
15     p code.arity
16
17 end

```

Sigue una ejecución:

```

~/rubytesting/distributedRuby/serializeProc$ ruby marshalproc2.rb
3
2

```

Este otro código muestra otra solución que no hace uso de `_dump` y `_load`:

```

~/rubytesting/distributedRuby/serializeProc$ cat -n serializableProc2.rb
1  class SerializableProc
2

```

```

3   @@cache = {}
4   def initialize( block )
5     @block = block
6     @block.sub!(/^/, 'lambda ') unless @block =~/^\\s*(?:lambda|proc)/
7     # Test if block is valid.
8     @@cache[@block] = eval "#{@block}"
9   end
10
11
12  def call(* args)
13    @@cache[@block].call(* args)
14  end
15
16  def arity
17    @@cache[@block].arity
18  end
19
20 end
21
22
23 if $0 == __FILE__
24
25   require 'yaml'
26   require 'pstore'
27
28   code = SerializableProc.new %q{ { |a,b| a+b } }
29
30   # Marshal
31   File.open('proc.marshalled', 'w') { |file| Marshal.dump(code, file) }
32   code = File.open('proc.marshalled') { |file| Marshal.load(file) }
33
34   p code.call( 1, 2 )
35
36   # PStore
37   store = PStore.new('proc.pstore')
38   store.transaction do
39     store['proc'] = code
40   end
41   store.transaction do
42     code = store['proc']
43   end
44
45   p code.call( 1, 2 )
46
47   # YAML
48   File.open('proc.yaml', 'w') { |file| YAML.dump(code, file) }
49   code = File.open('proc.yaml') { |file| YAML.load(file) }
50
51   p code.call( 1, 2 )
52
53   p code.arity
54
55 end

```

La solución ”memoiza.” en un hash que es un atributo de la clase la relación entre el fuente y el código. Cuando se ejecuta el código anterior se obtiene:

```
~/rubytesting/distributedRuby$ ruby serializeProc2.rb
3
3
3
2
```

Práctica: Procesos Concurrentes

Generalize el ejemplo visto en la sección ?? escribiendo un método `prun` que permita lanzar un número dado de procesos:

```
res = prun(
  :one      => lambda { |n| "#{n} is 1" },
  :three    => lambda { |n| "#{n} is 3" },
  'two'     => lambda { |n| n*4 }
)

puts res.inspect

res = prun(
  [2, 3, 7] => lambda { |n| n[0]+n[1]+n[2] },
  [4, 5]     => lambda { |n| n[0]*n[1] }
)

puts res.inspect
```

Este programa debería producir una salida similar a esta:

```
~/rubytesting/distributedRuby$ ruby parfork2.rb
{"two"=>"twotwotwotwo", :one=>"one is 1", :three=>"three is 3"}
{[2, 3, 7]=>12, [4, 5]=>20}
```

- Cada lambda es ejecutada en un proceso distinto.
- Las lambdas reciben un único argumento que es la clave asociada en el hash.
- El resultado `res` devuelto por `prun` será un hash cuyas claves son las mismas que se han pasado como argumentos y cuyos valores contienen los resultados devueltos por las correspondientes lambdas.
- Puede ser útil usar un canal de comunicaciones distinto para cada hijo
- El proceso padre deberá sincronizarse con cada uno de los procesos hijos
- El valor retornado por `wait2` contiene el PID del hijo. Esta información puede ser usada para que el proceso padre determine qué canal usar para la lectura

10.8.10. Freezing: Congelando Objetos

Any object may be frozen by calling its `freeze` method.

A frozen object becomes immutable: none of its internal state may be changed, and an attempt to call any of its mutator methods fails:

```

s = "ice"      # Strings are mutable objects
s.freeze      # Make this string immutable
s.frozen?     # true: it has been frozen
s.upcase!     # TypeError: can't modify frozen string
s[0] = "ni"   # TypeError: can't modify frozen string

```

Freezing a class object prevents the addition of any methods to that class.

You can check whether an object is frozen with the `frozen?` method.

Once frozen, there is no way to `thaw` an object.

If you copy a frozen object with `clone`, the copy will also be frozen.

If you copy a frozen object with `dup`, however, the copy will not be frozen.

10.8.11. Objetos Manchados: Tainting

Web applications must often keep track of data derived from untrusted user input to avoid SQL injection attacks and similar security risks.

Taint checking is a feature in some computer programming languages, such as Perl and Ruby, designed to increase security by preventing malicious users from executing commands on a host computer.

Taint checks highlight specific security risks primarily associated with web sites which are attacked using techniques such as SQL injection or buffer overflow attack approaches.

Ruby provides a simple solution to this problem: any object may be marked as tainted by calling its `taint` method.

Once an object is tainted, any objects derived from it will also be tainted.

The taint of an object can be tested with the `tainted?` method.

```

1] pry(main)> s = "untrusted"
=> "untrusted"
[3] pry(main)> s.taint
=> "untrusted"
[4] pry(main)> s.tainted?
=> true
[5] pry(main)> s.upcase.tainted?
=> true
[6] pry(main)> s[3,4].tainted?
=> true

```

Véase un ejemplo en Perl. y también ejemplos en Rails

The object tainting mechanism of Ruby is most powerful when used with the global variable `$SAFE`.

When this variable is set to a value greater than zero, Ruby restricts various built-in methods so that they will not work with tainted data.

When `$SAFE` is 1,

- You can't evaluate a string of code if the string is tainted;
- you can't require a library if the library name is tainted;
- you can't open a named file if the filename is tainted; and
- you can't connect to a network host if the hostname is tainted. Programs, especially networked servers,

When `$SAFE` is 2,3 or 4 this restrictions become harder.

10.8.12. Congelación: freezing

```
>> person2 = person1
=> "Wim"
>> person1.freeze
=> "Wim"
>> person2[0] = "R"
TypeError: can't modify frozen string
  from (irb):18:in '[]='
  from (irb):18

~/rubytesting$ cat -n freeze.rb
1 begin
2   a = [1,2,3].freeze
3   puts a
4   a << [4,7] # this raises an exception
5   puts a
6 rescue
7   puts "Exception <#${!.class}> raised <#${!}>"
8 end
~/rubytesting$ ruby freeze.rb
1
2
3
Exception <TypeError> raised <can't modify frozen array>

pp@nereida:~/LPPbook$ irb
irb(main):001:0>
irb(main):002:0*
irb(main):003:0* quit
lpp@nereida:~/LPPbook$ irb
irb(main):001:0> q = "hello"
=> "hello"
irb(main):002:0> q.freeze
=> "hello"
irb(main):003:0> q.object_id
=> 70233414821660
irb(main):004:0> q += " world"
=> "hello world"
irb(main):005:0> q.object_id
=> 70233414771300
irb(main):006:0> q.freeze
=> "hello world"
irb(main):007:0> q << " world"
TypeError: can't modify frozen string
  from (irb):7:in '<<'
  from (irb):7
  from :0
```

No hay manera de descongelar un objeto sobre el que ha sido llamado `freeze`.

10.9. Ejercicios

1. Explique porque es ineficiente esta forma de ordenar:

```

>> system 'ls -l c*.rb'
-rw-r--r-- 1 casianorodriguezleon staff 296 17 oct 22:12 calif.rb
-rw-r--r-- 1 casianorodriguezleon staff 320 26 sep 08:12 case.rb
-rw-r--r-- 1 casianorodriguezleon staff 526 10 oct 12:51 chuchu.rb
-rw-r--r-- 1 casianorodriguezleon staff 95 7 oct 16:27 closure.rb
=> true
>> files = Dir['c*.rb']
=> ["calif.rb", "case.rb", "chuchu.rb", "closure.rb"]
>> fs = files.sort { |a,b| File.new(b).mtime <= File.new(a).mtime }
=> ["calif.rb", "chuchu.rb", "closure.rb", "case.rb"]

```

2. El método `stat` de la clase `File` nos permite consultar el estado de un fichero:

```

>> File.stat("calif.rb")
=> #<File::Stat dev=0xe000002, ino=29688777, mode=0100644, nlink=1, uid=501, gid=20, rdev=

```

Por ejemplo, el método `mtime` permite ver la fecha de la última modificación de un fichero:

```

>> File.stat("calif.rb").mtime
=> Mon Oct 17 22:12:56 0100 2011
>> File.stat("calif.rb").mtime.class
=> Time
>> File.stat("calif.rb").mtime.to_i
=> 1318885976
>> File.stat("case.rb").mtime
=> Mon Sep 26 08:12:29 0100 2011
>> File.stat("case.rb").mtime.to_i
=> 1317021149
>> File.stat("case.rb").mtime < File.stat("calif.rb").mtime
=> true

```

¿Cuales son las salidas de los siguientes comandos?

```

>> files = Dir['c*.rb']
=> ["calif.rb", "case.rb", "chuchu.rb", "closure.rb"]
>> sf = files.collect { |f| [File.stat(f).mtime, f] }
=> [[Mon Oct 17 22:12:56 0100 2011, "calif.rb"], [Mon Sep 26 08:12:29 0100 2011, "case.rb"]]
>> tf = sf.sort
=>
>> tf.collect { |f| f[1] }
=>

```

Esta forma de ordenar un `Array` según ciertos valores, denominada *Schwartzian Transform*, consiste en tres pasos:

- Crear un array de pares manteniendo la correspondencia entre cada elemento del array y el valor asociado
- Ordenar el array de pares de acuerdo a la primera componente. El orden de arrays es el lexicográfico:

```

>> [2,"a"] <=> [2,"b"]
=> -1
>> [2,"b"] <=> [2,"a"]
=> 1
>> [2,"b"] <=> [2,"b"]
=> 0

```

- c) Proyectar el array ordenado en la segunda componente
3. El método `sort_by` ordena utilizando un conjunto de claves generadas a partir de los valores usando un bloque dado:

```
>> %w{apple pear fig}.sort_by { |w| -w.length }
=> ["apple", "pear", "fig"]
```

Para hacerlo utiliza el algoritmo *Schwartzian Transform*.

¿Cuál es la salida de esta sentencia?

```
>> Dir['c*.rb'].sort_by { |f| File.stat(f).mtime }
=>
```

4. ¿Cuáles son las salidas de los siguientes comandos?

```
>> h = { :a => 3, :b => 2, :c => 1 }
=> { :b=>2, :c=>1, :a=>3}
>> h.sort_by { |x| x[1] }
=> [[:c, 1], [:b, 2], [:a, 3]]
>> h.sort_by { |x| x[1] }.collect { |z| z[0] }
=>
```

10.10. Ejercicios

1. ¿Que resultados se obtienen en las siguientes operaciones?

```
>> h = { :one => 1, :two => 2}
=> { :one=>1, :two=>2}
>> h.index 1
=>
>> h.index 2
=>
>> h.key?(:a)
=>
>> h.key?(:one)
=>
>> h.has_key?(:one)
=>
>> h.include?(:one)
=>
>> h.member?(:one)
=>
>> h.value?(1)
=>
>> h.value?(3)
=>
>> h.has_value?(1)
=>
```

2. Es posible especificar un valor por defecto para un Hash. El método `default` retorna el valor por defecto.

¿Que resultados se obtienen en las siguientes operaciones?

```

>> h = Hash.new("Go Fish")
=> {}
>> h.default
=>
>> h["a"] = 100
=> 100
>> h["b"] = 200
=>
>> h["a"]
=>
>> h["c"]
=>
>> h["c"].upcase!
=> "GO FISH"
>> h["c"]
=>
>> h["d"]
=>
>> h.keys
=>

```

3. Se puede pasar al constructor de la clase `Hash` un bloque de código que se encarga de computar el valor por defecto. Al bloque se le pasa el hash y la clave.

¿Que resultados se obtienen en las siguientes operaciones?

```

>> h = Hash.new { |hash, key| hash[key] = "Go Fish: #{key}" }
=> {}
>> h["c"]
=>
>> h["d"]
=>
>> h.keys
=>

```

4. Este ejercicio muestra como un hash puede ser utilizado en ruby para memoizar un cálculo.

¿Que resultados se obtienen en las siguientes operaciones?

```

>> fact = Hash.new { |h,k| if k > 1 then h[k] = k*h[k-1] else h[1] = 1 end }
=> {}
>> fact[4]
=>
>> fact
=>

```

5. ¿Que resultados se obtienen en las siguientes operaciones (véase la entrada en la Wikipedia para Números de Fibonacci)?

```

>> fib = Hash.new { |h,k| if k > 1 then h[k] = h[k-1]+h[k-2] end }
=> {}
>> fib[0], fib[1] = 0, 1
=> [0, 1]
>> fib[7]
=> 13

```

```

>> fib
=>
>> fib.sort
=>
>> fib[12]
=> 144
>> fib.sort
=>
>> fib.keys.sort { |a,b| a.to_s <=> b.to_s }.each { |k| puts "#{k} => #{fib[k]}" }
=>

```

6. ¿Que resultados se obtienen en las siguientes operaciones?

```

>> h = {}
=> {}
>> h[:a] = 1
=> 1
>> h.store(:b, 2)
=>
>> h
=>
>> h.replace({1 => :a, 2 => :b})
=> {1=>:a, 2=>:b}
>> h
=>
>> h.replace({1 => :a, 2 => :b, 3 => :c})
=>

```

7. ¿Que resultados se obtienen en las siguientes operaciones?

```

>>
?>   class Hash
>>     def []= (*keys)
>>       value = keys.pop
>>       keys.each{|key| store(key, value) }
>>     end
>>   end
=> nil
>>   hsh = {}
=> {}
>>   hsh[:a, :b, :c] = 42

```

```

=> 42
>>     hsh
=>
>>     hsh[:a, :b] = "foo"
=> "foo"
>>     hsh[:a].upcase!
=>
>>     hsh[:b]
=>

```

8. ¿Que resultados se obtienen en las siguientes operaciones?

```

>> h = { :a => 1, :b => 2 }
=> {:b=>2, :a=>1}
>> h.fetch(:a)
=> 1
>> h.fetch(:c)
IndexError: key not found
from (irb):18:in `fetch'
from (irb):18
from :0
>> h[:c]
=>
>> h.fetch(:c, 33)
=>
>> h
=> {:b=>2, :a=>1}
>> h.fetch(:c) { |k| "#{k} oppps!" }
=>

```

9. ¿Que resultados se obtienen en las siguientes operaciones?

```

>> h = { :a => 1, :b => 2, :c => 3 }
=> {:b=>2, :c=>3, :a=>1}
>> h.values_at(:a, :b)
=> [1, 2]
>> h.values_at(:d, :d, :a)
=>
>> h.values_at(:c)
=>

```

10. ¿Que resultados se obtienen en las siguientes operaciones?

```

>> h = { :a => 1, :b => 2, :c => 3 }
=> {:b=>2, :c=>3, :a=>1}
>> h.select { |k,v| v % 2 == 0 }
=>
>> h = { :a => 1, :b => 2}
=> {:b=>2, :a=>1}
>> h.merge({:a => 3, :c => 3})
=>
>> h
=>
>> h.merge!({:a => 3, :c => 3})

```

```
=>  
>> h  
=>
```

11. ¿Que resultados se obtienen en las siguientes operaciones?

```
>> h = { :a => 1, :b => 2}  
=> {:b=>2, :a=>1}  
>> j = { :a => 3, :c => 3}  
=> {:c=>3, :a=>3}  
>> h.merge(j) { |k,a,b| a}  
=>  
>> h.merge(j) { |k,a,b| (a+b)/2}  
=>
```

12. La clase Hash dispone del método Hash[[key =>|, value]*] que crea un nuevo hash con los objetos especificados. Es equivalente a { key, value, ... }. Debe haber un número par de argumentos.

¿Que resultados se obtienen en las siguientes operaciones?

```
>> a = [:a, 1, :b, 2, :c, 3]  
=> [:a, 1, :b, 2, :c, 3]  
>> Hash[*a]  
=>  
>> b = [[:a, 1], [:b, 2], [:c, 3]]  
=> [[:a, 1], [:b, 2], [:c, 3]]  
>> b.flatten  
=>  
>> Hash[*b.flatten]  
=>  
>> x = ["a" => 100, "b" => 200]  
=> [{"a"=>100, "b"=>200}]  
>> h = Hash[*x]  
=>
```

13. ¿Que resultados se obtienen en las siguientes operaciones?

```
>> h = { :a => 1, :b => 2}  
=> {:b=>2, :a=>1}  
>> Array[*h]  
=>  
>> Array[*h].flatten  
=>
```

14. ¿Que resultados se obtienen en las siguientes operaciones?

```
>> a = ('a'..'z').to_a  
=>  
>> a.map { |x| [ x, nil ] }  
=>  
>> a.map { |x| [ x, nil ] }.flatten  
=>  
>> Hash[* a.map { |x| [ x, nil ] }.flatten]  
=>
```

15. ¿Que resultados se obtienen en las siguientes operaciones?

```
>> h = { :a => 1, :b => 2, :c => 3 }
=> { :b => 2, :c => 3, :a => 1 }
>> h.invert
=>
>> h.to_s
=>
>> h.inspect
=>
```

10.11. Práctica: Ordenar por Calificaciones

Se tiene un fichero de entrada con calificaciones de alumnos como el que sigue:

```
Ferrer Pérez, Eduardo & 9'6\\
García García, Laura & 7'5 \\
García Medina, Anai & 6'5\\
García Rodríguez, Pedro & 7'2\\
González del Castillo, Jorge & 5'0\\
Hernández Pérez, Daniel & 5'0\\
Marrero Díaz, Jose Antonio & 8'0\\
Marrero Piñero, Sergio & 7'2\\
Padrón Padrón, Adolfo & 5'0\\
Pedrín Ruiz, Ana & 9'6\\
Pedrínez Pérez, Ignacio & 7'5\\
Piñero Piñero, Antonio & 5'3\\
Pérez García, Adolfo & 9'5\\
Pérez González, Germán & 5'0\\
Pérez Pedrínez, María & 5'0\\
Ramos Ramos, Ignacio & 5'0\\
Rodríguez González, Andrés & 10'0\\
Rodríguez Rodríguez, Jorge & 9'2\\
```

Se pide escribir un programa que muestre el listado de alumnos ordenados en orden decreciente de calificaciones.

Sugerencias:

1. Puede leer el fichero completo en un array usando `readlines`:

```
>> f = File.open('notas')
=> #<File: notas>
>> x = f.readlines
=> ["Ferrer P\303\251rez, Eduardo & 9'6\\\\\\n", ... , "Rodr\303\255uez Rodr\303\255uez,
>> puts x[-1]
Rodríguez Rodríguez, Jorge & 9'2\\
=> nil
```

2. Ahora deberá construir un hash indexado en los nombres de los alumnos y cuyos valores sean las calificaciones:

```
>> y = x.map { |alu| alu =~ /(.* )\s+\&\s+(\d+\.\d*)/; [ $1, $2 ] }
=> [["Ferrer P\303\251rez, Eduardo", "9'6"], ... , ["Rodr\303\255uez Rodr\303\255uez,
>> puts y[-1]
```

```

Rodríguez Rodríguez, Jorge
9'2
>> h = Hash[*y.flatten]
=> {"Pedr\303\255nez P\303\251rez, Ignacio"=>"7'5", ... , "Marrero Pi\303\261ero, Sergio"=gt;
>> h.keys
=> ["Pedr\303\255nez P\303\251rez, Ignacio", ... , "Marrero Pi\303\261ero, Sergio"]
>> h["Pedr\303\255nez P\303\251rez, Ignacio"]
=> "7'5"

```

3. Ordene el hash en orden decreciente de calificaciones. Observe que las números se han escrito según la convención española de usar apóstrofes en vez del punto decimal. Una posibilidad es que utilices `gsub` para hacer sustituciones y convertir los apóstrofes en puntos:

```

>> "7'5".gsub(/(\d+)'(\d*)/, '\1.\2')
=> "7.5"
>> "7".gsub(/(\d+)'(\d*)/, '\1.\2')
=> "7"

```

4. Es posible ordenar los elementos de un hash según sus claves:

```

irb(main):005:0> h = {'a' => 3, 'b' => 2, 'c' => 1}
=> {"a"=>3, "b"=>2, "c"=>1}
irb(main):008:0> h.sort
=> [["a", 3], ["b", 2], ["c", 1]]

```

Se puede pasar un bloque a `sort`. El bloque recibe los dos elementos a comparar. Los elementos son parejas clave-valor:

```

irb(main):006:0> h.sort { |a,b| a[0] <=> b[0] }
=> [["a", 3], ["b", 2], ["c", 1]]
irb(main):007:0> h.sort { |a,b| a[1] <=> b[1] }
=> [["c", 1], ["b", 2], ["a", 3]]

```

10.12. Ejercicios

1. ¿Qué resultados dan las siguientes operaciones?

```

>> a = 1..10
=> 1..10
>> a.class
=> Range
>> a.to_a
=>
>> b = 1...10
=> 1....10
>> b.to_a
=>
>> b.include? 10
=>
>> b.include? 8
=>
>> b.step(2) {|x| print "#{x} "}
=>
>> 1..3.to_a

```

2. ¿Que resultados dan las siguientes operaciones?

```
>> r = 0...100  
=> 0....100  
>> r.member? 50  
=>  
>> r.include? 99.9  
=>  
>> r.member? 99.9  
=>
```

3. ¿Que resultados dan las siguientes operaciones?

```
>> true.class  
=>  
>> false.class  
=>  
>> puts "hello" if 0  
  
>> puts "hello" if nil  
  
>> puts "hello" if ""  
  
>>
```

4. ¿Que resultados dan las siguientes operaciones?

```
>> x = :sym  
=> :sym  
>> x.class  
=>  
>> x == 'sym'  
=>  
>> x == :sym  
=>  
>> z = :'a long symbol'  
=> :"a long symbol"  
>> z.class  
=>  
>> x == 'sym'.to_sym  
=>  
>> x.to_s == 'sym'  
=>
```

10.13. Ejercicios

1. ¿Que resultados se dan?

```
>> s = "Ruby"  
=> "Ruby"  
>> t = s  
=> "Ruby"  
>> t[-1] = ""
```

```
=> ""
>> print s

>> t = "Java"
=> "Java"
>> print s, t
```

2. ¿Que resultados se dan?

```
>> x = "hello"
=> "hello"
>> x.id
(irb):97: warning: Object#id will be deprecated; use Object#object_id
=> 2149253820
>> x.object_id
=>
>> x.__id__
=>
```

3. ¿Que retorna el método `hash` proveído por la clase `Object`?

Capítulo 11

Expresiones y Operadores

11.1. Literales y Palabras Reservadas

11.2. Variables

11.2.1. Variables No Inicializadas

11.3. Constantes

11.4. Invocación de Métodos

11.5. Asignaciones

11.5.1. Asignación a Variables

11.5.2. Asignación a Constantes

11.5.3. Asignación a Atributos y a Elementos de Arrays

11.5.4. Asignaciones Abreviadas

11.5.5. Asignaciones Paralelas

11.6. Operadores

11.6.1. + y - Unario

11.6.2. Exponenciación

11.6.3. Operadores Binarios

11.6.4. Shift y Append

11.6.5. Complemento, Unión e Intersección

11.6.6. Comparación

11.6.7. Igualdad

11.6.8. Operadores Lógicos

11.6.9. Rangos y Flip-Flops

Flip-Flops Lógicos

```
ruby-1.9.2-head :002 > (1..10).each { |x| puts x if x==3..x==5 }
```

3

4

5

=> 1..10

11.6.10. Condicional ?

11.6.11. Operadores de Asignación

11.6.12. El Operador defined?

11.6.13. Modificadores de Sentencias (Sufijos)

11.6.14. Símbolos de Puntuación que no son Operadores

Capítulo 12

Sentencias y Estructuras de Control

12.1. Condicionales

```
if conditional [then]
    code...
[elsif conditional [then]
    code...]...
[else
    code...]
end

case value
when expression1
    #do something
when expression2
    #do something
...
when expressionN
    #do something
else
    #do default case
end

honda = ['honda', 'acura', 'civic', 'element', 'fit', ...]
toyota = ['toyota', 'lexus', 'tercel', 'rx', 'yaris', ...]
...

if include_concept_cars:
    honda += ['ev-ster', 'concept c', 'concept s', ...]
    ...

case car
when *toyota
    # Do something for Toyota cars
when *honda
    # Do something for Honda cars
...
end
```

12.2. Bucles

12.3. Iteradores y Objetos Enumerables

12.3.1. Enumeradores

Iteradores Externos

La clase `Enumerator` permite la iteración tanto interna como externa.

- When the iterator controls the iteration, the iterator is an *internal iterator*.
- Clients that use an *external iterator* must advance the traversal and request the `next` element explicitly from the iterator.

Una manera de crear objetos `Enumerator` es mediante el método `to_enum` (en la clase `Object`):

```
[1] pry(main)> a = [1, 2, "cat" ]
=> [1, 2, "cat"]
[3] pry(main)> ea = a.to_enum
=> #<Enumerator: ...>
[4] pry(main)> ea.next
=> 1
[5] pry(main)> ea.next
=> 2
[6] pry(main)> ea.next
=> "cat"
[7] pry(main)> ea.next
StopIteration: iteration reached an end
from (pry):7:in `next'
[8] pry(main)> ea.rewind
=> #<Enumerator: ...>
[9] pry(main)> ea.next
=> 1
[10] pry(main)> ea.next
=> 2
[11] pry(main)> ea.peek
=> "cat"
```

También puede usarse con hashes:

```
28] pry(main)> h = { a: "hello", b: "world" }
=> {:a=>"hello", :b=>"world"}
[29] pry(main)>
[30] pry(main)> eh = h.to_enum
=> #<Enumerator: ...>
[31] pry(main)> eh.size
=> nil
[32] pry(main)> eh.next
=> [:a, "hello"]
[33] pry(main)> eh.next_values
=> [[:b, "world"]]
[34] pry(main)> eh.rewind
=> #<Enumerator: ...>
```

El método `to_enum` está definido en la clase `Object`.

The call `obj.to_enum(method = :each, *args)` creates a new `Enumerator` which will enumerate by calling method `:each` on `obj`, passing `args` if any.

```

[2] pry(main)> class Tutu
[2] pry(main)*   def chuchu(x)
[2] pry(main)*     if block_given?
[2] pry(main)*       yield x+1
[2] pry(main)*       yield x+2
[2] pry(main)*       yield x+3
[2] pry(main)*     else
[2] pry(main)*       to_enum(:chuchu, x) { 3 }
[2] pry(main)*     end
[2] pry(main)*   end
[2] pry(main)* end
=> :chuchu
[3] pry(main)> t = Tutu.new
=> #<Tutu:0x007f9e6cb5af60>
[5] pry(main)> t.chuchu(10) { |x| puts x }
11
12
13
=> nil
[6] pry(main)> e = t.chuchu(10)
=> #<Enumerator: ...>
[7] pry(main)> e.next
=> 11
[8] pry(main)> e.next
=> 12
[9] pry(main)> e.size
=> 3

```

with_index El método `with_index(offset = 0) {|(*args), idx| ... }` (o bien `with_index(offset = 0)`) iterates the given block for each element with an index, which starts from offset. If no block is given, returns a new Enumerator that includes the index, starting from offset

```

[40] pry(main)> h = { a: "hello", b: "world" }
=> {:a=>"hello", :b=>"world"}
[41] pry(main)> eh = h.to_enum
=> #<Enumerator: ...>
[42] pry(main)> eh.with_index.next
=> [[:a, "hello"], 0]
[44] pry(main)> eh.with_index { |x, i| puts "#{x.inspect}, #{i}" }
[:a, "hello"], 0
[:b, "world"], 1
=> {:a=>"hello", :b=>"world"}

```

Construcción de un Enumerador

1. As well as creating `Enumerators` from existing collections, you can create an explicit enumerator, passing it a block
2. The code in the block will be used when the enumerator object needs to supply a fresh value to your program
3. However, the block isn't simply executed from top to bottom

4. Instead, the block is executed as a *coroutine*¹ with the rest of your program's code
5. Execution starts at the top and pauses when the block **yields** a value to your code
6. When the code needs the **next** value, execution resumes at the statement following the **yields**
7. This lets you write enumerators that generate *infinite sequences*:

```
[~/ruby/PROGRAMMINGRUBYDAVETHOMAS]$ cat fib_enum.rb
@fib = Enumerator.new do |y|
  a, b = 0, 1
  loop do
    a, b = b, a + b
    y.yield a
  end
end
```

`Enumerator.new` receives a block that is used to define the iteration. A **yielder** object `y` is given to the block as a parameter and can be used to **yield** a value by calling the **yield** method (aliased as `<<`).

```
[~/ruby/PROGRAMMINGRUBYDAVETHOMAS]$ pry
[1] pry(main)> require './fib_enum'
=> true
[2] pry(main)> @fib.next
=> 1
[3] pry(main)> @fib.next
=> 1
[4] pry(main)> @fib.next
=> 2
[6] pry(main)> @fib.first
=> 1
[7] pry(main)> @fib.take(5)
=> [1, 1, 2, 3, 5]
```

La clase `Enumerator` incluye el módulo `Enumerable` de donde toma, entre otros, los métodos `first` y `take`.

```
[25] pry(main)> Enumerator.ancestors
=> [Enumerator, Enumerable, Object, PP::ObjectMixin, Kernel, BasicObject]
```

Enumeradores Infinitos

1. You have to be slightly careful with `Enumerators` that can generate infinite sequences
2. Some of the regular `Enumerable` methods such as `count` and `select` will happily try to read the whole enumeration before returning a result
3. If you want a version of `select` that works with infinite sequences, you'll need to write it yourself
4. Here's a version that gets passed an `Enumerator` and a block and returns a new `Enumerator` containing values from the original for which the block returns `true`

¹ Coroutines are computer program components that generalize subroutines for nonpreemptive multitasking, by allowing multiple entry points for suspending and resuming execution at certain locations.

Coroutines are well-suited for implementing more familiar program components such as cooperative tasks, exceptions, event loop, iterators, infinite lists and pipes.

```

[1] pry(main)> require './fib_enum'
=> true
[2] pry(main)> require './grep8'
=> true
[3] pry(main)> e = @fib.grep8 { |x| x % 3 == 0}
=> #<Enumerator: ...>
[4] pry(main)> e.next
=> 3
[5] pry(main)> e.next
=> 21
[6] pry(main)> e.next
=> 144
[7] pry(main)> e.first(5)
=> [3, 21, 144, 987, 6765]

[9] pry(main)> .cat grep8.rb
class Enumerator
  def grep8(&block)
    Enumerator.new do |y|
      self.each do |x|
        y.yield x if block[x]
      end
    end
  end
end

```

Enumeradores Perezosos Otra forma de crear iteradores infinitos en Ruby 2.0 es mediante la clase `Enumerator::Lazy`:

```

[16] pry(main)> e = (1..Float::INFINITY).lazy.map { |x| x*x }
=> #<Enumerator::Lazy: ...>
[17] pry(main)> e.next
=> 1
[18] pry(main)> e.next
=> 4
[19] pry(main)> e.next
=> 9
[20] pry(main)> e.next
=> 16
[21] pry(main)> e.take(4).force
=> [1, 4, 9, 16]
[22] pry(main)> e.take(4)
=> #<Enumerator::Lazy: ...>
[23] pry(main)> e.first(4)
=> [1, 4, 9, 16]

```

En este otro ejemplo creamos un enumerador perezoso `filter_map` que devuelve los elementos procesados por el bloque cuyo valor no es `nil`. Su funcionamiento sería así::

```

[1] pry(main)> require './lazy_enum'
=> true
[2] pry(main)> [1,2,3].map { |x| x*x if x % 2 != 0 }
=> [1, nil, 9]
[3] pry(main)> [1,2,3].filter_map { |x| x*x if x % 2 != 0 }
=> [1, 9]          # se elimina el nil

```

Para usarlo con rangos infinitos:

```
[4] pry(main)> (1..Float::INFINITY).lazy.filter_map{|i| i*i if i.even?}.first(5)
=> [4, 16, 36, 64, 100]
```

El método `lazy` se encuentra en el módulo `Enumerable` y retorna un enumerador perezoso `Enumerator::Lazy` que enumera/construye el valor cuando se necesita.

```
[~/ruby/PROGRAMMINGRUBYDAVETHOMAS]$ cat lazy_enum.rb
module Enumerable
  def filter_map(&block) # Para explicar la semantica
    map(&block).compact
  end
end

class Enumerator::Lazy
  def filter_map
    Lazy.new(self) do |yielder, *values|
      result = yield *values
      yielder << result if result # << es un alias de yield
    end
  end
end
```

The call `new(self) { |yielder, *values| ... }` creates a new `Lazy` enumerator using `self` as the coupled enumerator.

When the enumerator is actually enumerated (e.g. by calling `force` or `next`), `self` will be enumerated and each value passed to the given block.

The block then `yield` values back using `yielder.yield` or `yielder << ...`.

Ejemplo: Números primos

```
[~/rubystesting]$ cat primes.rb
def primes
  (1..Float::INFINITY).lazy.select do |n|
    ((2..n**0.5)).all? { |f| n % f > 0 }
  end
end

p primes.first(10)
```

```
[~/rubystesting]$ ruby primes.rb
[1, 2, 3, 5, 7, 11, 13, 17, 19, 23]
```

Véase

1. Ruby 2.0 Works Hard So You Can Be Lazy por Pat Shaughnessy. April 3rd 2013

12.4. Bloques

Ruby *Code blocks* (called *closures* in other languages) are chunks of code between braces or between `do..end` that you can associate with `method invocations`, almost as if they were parameters.

A Ruby block is a way of grouping statements, and may appear only in the source adjacent to a method call;

the block is written starting on the same line as the method call's last parameter (or the closing parenthesis of the parameter list).

- The code in the block is not executed at the time it is encountered.
- Instead, Ruby remembers the context in which the block appears (the local variables, the current object, and so on) and then enters the method.

The Ruby standard is to use braces for single-line blocks and `do...end` for multi-line blocks.

yield

Any method can be called with a block as an implicit argument. Inside the method, you can call the block using the `yield` keyword with a value.

Blocks can have their own arguments. There are many methods in Ruby that iterate over a range of values.

```
[1] pry(main)> a = (1..8).to_a
=> [1, 2, 3, 4, 5, 6, 7, 8]
[2] pry(main)> a.each { |x| puts x*x };
1
4
9
16
25
36
49
64
```

Most of these iterators are written in such a way as to be able to take a code block as part of their calling syntax.

The method can then `yield` control to the code block (i.e. execute the block) during execution as many times as is necessary for the iteration to complete (e.g. if we are iterating over array values, we can execute the block as many times as there are array values etc.).

Once you have created a block, you can associate it with a call to a method. Usually [the code blocks passed into methods are anonymous objects](#), created on the spot.

If you provide a code block when you call a method, then inside the method, you can `yield` control to that code block - suspend execution of the method; execute the code in the block; and return control to the method body, right after the call to `yield`.

```
[~/local/src/ruby/rubytesting]$ cat block_simple.rb
def call_block
  puts "Start of method"
  yield
  yield
  puts "End of method"
end

[3] pry(main)> require_relative 'block_simple'
=> true
[4] pry(main)> call_block do puts "hello!" end
Start of method
hello!
hello!
End of method
=> nil
```

If no code block is passed, Ruby raises an exception

```

~/rubytesting]$ pry
[1] pry(main)> require_relative 'block_simple'
=> true
[2] pry(main)> call_block
Start of method
LocalJumpError: no block given (yield)
from /Users/casiano/local/src/ruby/rubytesting/block_simple.rb:3:in `call_block'
```

You can provide parameters to the call to `yield`: these will be passed to the block. Within the block, you list the names of the arguments to receive the parameters between vertical bars (`|`).

```

[~/local/src/ruby/rubytesting]$ pry
[1] pry(main)> def tutu(x)
[1] pry(main)*   "hello #{ yield x }"
[1] pry(main)* end
=> nil
[2] pry(main)> tutu('Juan') { |y| y+'a' }
=> "hello Juana"
[3] pry(main)>
```

Note that the code in the block is not executed at the time it is encountered by the Ruby interpreter. Instead, Ruby remembers the context in which the block appears and then enters the method.

```

[~/rubytesting]$ cat block_simple2.rb
y = 4

def call_block(a)
  puts "Start of method"
  puts "Inside the method a = #{a}"
  yield 4
  yield 5
  puts "End of method"
end

call_block 8 do |x|
  puts x+y
end
```

```
[~/rubytesting]$ ruby block_simple2.rb
Start of method
Inside the method a = 8
8
9
End of method
```

block_given?

`block_given?` returns `true` if `yield` would execute a block in the current context.

```
[5] pry(main)> show-source tutu
```

```

From: (pry) @ line 3:
Owner: Object
Visibility: public
Number of lines: 4
```

```
def tutu
```

```

return yield 4 if block_given?
"no block was given!"
end
[6] pry(main)> tutu { |x| x*x }
=> 16
[7] pry(main)> tutu
=> "no block was given!"

```

Ambito/Scope

Let us see what happens in the following example when a variable outside a block is `x` and a block parameter is also named `x`.

```
[~/rubytesting]$ cat block_scope1.rb
x = 10
5.times do |x|
  puts "x inside the block: #{x}"
end

puts "x outside the block: #{x}"
[~/rubytesting]$ ruby block_scope1.rb
x inside the block: 0
x inside the block: 1
x inside the block: 2
x inside the block: 3
x inside the block: 4
x outside the block: 10
```

You will observe that after the block has executed, `x` outside the block is the original `x`. Hence the block parameter `x` was local to the block.

Since `x` is not a block parameter here, the variable `x` is the same inside and outside the block.

```
[~/rubytesting]$ cat block_scope2.rb
x = 10
5.times do |y|
  x = y
  puts "x inside the block: #{x}"
end

puts "x outside the block: #{x}"
[~/rubytesting]$ ruby block_scope2.rb
x inside the block: 0
x inside the block: 1
x inside the block: 2
x inside the block: 3
x inside the block: 4
x outside the block: 4
```

In Ruby 1.9, blocks introduce their own scope for the block parameters only. This is illustrated by the following example:

```
[~/rubytesting]$ cat block_scope3.rb
x = 10
5.times do |y, x|
  x = y
  puts "x inside the block: #{x}"
```

```

end
puts "x outside the block: #{x}"
[~/rubytesting]$ ruby -v
ruby 2.1.2p95 (2014-05-08 revision 45877) [x86_64-darwin13.0]
[~/rubytesting]$ ruby block_scope3.rb
x inside the block: 0
x inside the block: 1
x inside the block: 2
x inside the block: 3
x inside the block: 4
x outside the block: 10
[~/rubytesting]$ rvm use 1.8.7-p352
Using /Users/casiano/.rvm/gems/ruby-1.8.7-p352
[~/rubytesting]$ ruby -v
ruby 1.8.7 (2011-06-30 patchlevel 352) [i686-darwin11.3.0]
[~/rubytesting]$ ruby block_scope3.rb
x inside the block: 0
x inside the block: 1
x inside the block: 2
x inside the block: 3
x inside the block: 4
x outside the block: 4

```

Ambiguedad

Keep in mind that the braces syntax has a higher precedence than the `do..end` syntax.

Braces have a high precedence; `do` has a low precedence. If the method invocation has parameters that are not enclosed in parentheses, the brace form of a block will bind to the last parameter, not to the overall invocation.

```

7] pry(main)> a = [1,2,3]
=> [1, 2, 3]
[8] pry(main)> p a.map { |n| n*2 }
[2, 4, 6]
=> [2, 4, 6]
[9] pry(main)> p a.map do |n| n*2 end
#<Enumerator: [1, 2, 3]:map>

```

En el caso `p a.map do |n| n*2 end` la sentencia se interpreta como `(p a.map) (do |n| n*2 end)`.
The `do` form will bind to the invocation.

El bloque se asocia con su método mas cercano

```

[10] pry(main)> show-source tutu

From: (pry) @ line 12:
Owner: Object
Visibility: public
Number of lines: 4

def tutu(s)
  puts "tutu"
  yield 4
end
[11] pry(main)> show-source titi

```

```

From: (pry) @ line 5:
Owner: Object
Visibility: public
Number of lines: 4

def titi
  puts "titi"
  yield 5
end
[12] pry(main)> tutu titi { |x| puts x } { |z| puts "2nd bloc #{z}" }
titi
5
tutu
2nd bloc 4
=> nil

```

Véase

- Ruby Blocks en rubylearning.com por Satish Talim
- La sección *Argumentos Bloque* en 13.4.6 dentro de la sección *Argumentos de un Método* 13.4
- La sección *Bindings (encarpetados) y eval* 15.2.1

12.5. Alterando el Flujo de Control

12.5.1. throw y catch son Sentencias de Control

`throw` and `catch` are `Kernel` methods that define a control structure that can be thought of as a multilevel `break`.

`throw` doesn't just `break` out of the current loop or block but can actually transfer out any number of levels, causing the block defined with a `catch` to exit.

The `catch` need not even be in the same method as the `throw`. It can be in the calling method, or somewhere even further up the call stack.

Languages like Java and JavaScript allow loops to be named or labeled with an arbitrary prefix. When this is done, a control structure known as a *labeled break* causes the named loop to exit.

Ruby's `catch` method defines a labeled block of code, and Ruby's `throw` method causes that block to exit.

But `throw` and `catch` are much more general than a labeled `break`. For one, it can be used with any kind of statement and is not restricted to loops.

More profoundly, a `throw` can propagate up the call stack to cause a block in an invoking method to exit.

If you are familiar with languages like Java and JavaScript, then you probably recognize `throw` and `catch` as the keywords those languages use for raising and handling exceptions.

Ruby does exceptions differently, using `raise` and `rescue`, which we'll learn about later in this chapter.

But the parallel to exceptions is intentional. Calling `throw` is very much like raising an exception.

And the way a `throw` propagates out through the lexical scope and then up the call stack is very much the same as the way an exception propagates out and up.

Despite the similarity to exceptions, it is best to consider `throw` and `catch` as a general-purpose (if perhaps infrequently used) control structure rather than an exception mechanism.

If you want to signal an error or exceptional condition, use `raise` instead of `throw`.

The following code demonstrates how `throw` and `catch` can be used to `break` out of nested loops:

```

for matrix in data do      # Process a deeply nested data structure.
  catch :missing_data do  # Label this statement so we can break out.
    for row in matrix do
      for value in row do
        throw :missing_data unless value # Break out of two loops at once.
        # Otherwise, do some actual data processing here. end
      end
    end
  end
# We end up here after the nested loops finish processing each matrix.
# We also get here if :missing_data is thrown.
end

```

Note that the `catch` method takes a symbol argument and a block.

It executes the block and returns when the block exits or when the specified symbol is thrown.

`throw` also expects a symbol as its argument and causes the corresponding `catch` invocation to return.

If no `catch` call matches the symbol passed to `throw`, then a `NameError` exception is `raised`.

12.6. Manejo de Excepciones

An exception is an object that represents some kind of exceptional condition; it indicates that something has gone wrong. This could be a programming error—attempting to divide by zero, attempting to invoke a method on an object that does not define the method, or passing an invalid argument to a method. Or it could be the result from some kind of external condition—making a network request when the network is down, or trying to create an object when the system is out of memory.

When one of these errors or conditions occurs, an exception is `raised` (or `thrown`). By default, Ruby programs terminate when an exception occurs. But it is possible to declare exception handlers. An exception handler is a block of code that is executed if an exception occurs during the execution of some other block of code. In this sense, exceptions are a kind of control statement. Raising an exception transfers the flow-of-control to exception handling code. This is like using the `break` statement to exit from a loop. As we'll see, though, exceptions are quite different from the `break` statement; they may transfer control out of many enclosing blocks and even up the call stack in order to reach the exception handler.

Ruby uses the Kernel method `raise` to `raise` exceptions, and uses a `rescue` clause to handle exceptions. Exceptions `raised` by `raise` are instances of the `Exception` class or one of its many subclasses. The `throw` and `catch` methods described earlier in this chapter are not intended to signal and handle exceptions, but a symbol thrown by `throw` propagates in the same way that an exception `raised` by `raise` does. Exception objects, exception propagation, the `raise` method, and the `rescue` clause are described in detail in the subsections that follow.

- Ruby Exceptions en rubylearning.com

12.7. BEGIN y END

12.8. Threads, Fibras y Continuaciones

12.8.1. Fibras

Véase

1. <https://github.com/crguezl/rack-fiber-pool>

12.8.2. Threads

Véase *Threads* 25.

12.9. Práctica: Reto Dropbox. El Problema de la Dieta

En el sitio web de Dropbox en <http://www.dropbox.com/jobs/challenges> pueden encontrarse algunos retos de programación.

We get a lot of interest in working at Dropbox but it's not always easy to tell how a person's brain works from a resume. If you're like us, you love a good puzzle. That's why this page is here! It's a great way to share why we love working at Dropbox: innovative thinking, a bit of elbow grease, and some good fun. Who knows? If you knock these puzzles out of the park, we'll have something to talk about when you come in.

Send all submissions to challenges@dropbox.com. The subject line should match the puzzle name. Please send your code (and any associated makefiles), not the executable.

En esta práctica se trata de resolver *el problema de la dieta*. En el sitio de Dropbox se describe como sigue:

The Dropbox Diet

Of the boatload of perks Dropbox offers, the ones most threatening to our engineers' waistlines are the daily lunches, the fully-stocked drink fridge, and a full-length bar covered with every snack you could want. All of those calories add up. Luckily, the office is also well-equipped with ping-pong, a DDR machine, and a subsidized gym right across the street that can burn those calories right back off. Although we often don't, Dropboxers should choose the food they eat to counterbalance the activities they perform so that they don't end up with caloric deficit or excess.

Help us keep our caloric intake in check. You'll be given a list of activities and their caloric impact. Write a program that outputs the names of activities a Dropboxer should choose to partake in so that the sum of their caloric impact is zero. Once an activity is selected, it cannot be chosen again.

- *Input*

Your program reads an integer N (1 <= N <= 50) from STDIN representing the number of list items in the test input. The list is comprised of activities or food items and its respective calorie impact separated by a space, one pair per line. Activity names will use only lowercase ASCII letters and the dash (-) character.

- *Output*

Output should be sent to stdout, one activity name per line, alphabetized. If there is no possible solution, the output should be no solution. If there are multiple solutions, your program can output any one of them. Solutions should be non-trivial, so don't send us cat > /dev/null, you smart aleck.

- *Examples:*

	<i>Sample Input</i>	<i>Sample Output</i>
1.	2 red-bull 140 coke 110	no solution

<i>Sample Input</i>	<i>Sample Output</i>
<pre> 12 free-lunch 802 mixed-nuts 421 orange-juice 143 heavy-ddr-session -302 cheese-snacks 137 2. cookies 316 mexican-coke 150 dropballers-basketball -611 coding-six-hours -466 riding-scooter -42 rock-band -195 playing-drums -295 </pre>	<pre> coding-six-hours cookies mexican-coke </pre>

- Puede encontrar dos soluciones al problema escritas en Ruby en el blog de Alan Skorkin. La primera es una solución exhaustiva. La segunda hace uso de Programación Dinámica. Asegúrese que entiende ambas soluciones e intente adaptarlas a su estilo y mejorarlas.

Capítulo 13

Métodos, Procs, Lambdas y Clausuras

La mayoría de las secciones en este capítulo de los apuntes están sacadas del libro [2].

13.1. Definiendo Métodos Simples

13.1.1. Valor Retornado por un Método

13.1.2. Métodos y Manejo de Excepciones

A `def` statement that defines a method may include exception-handling code in the form of `rescue`, `else`, and `ensure` clauses, just as a `begin` statement can.

These exception-handling clauses go after the `end` of the method body but before the end of the `def` statement.

In short methods, it can be particularly tidy to associate your `rescue` clauses with the `def` statement.

This also means you don't have to use a `begin` statement and the extra level of indentation that comes with it.

```
def method_name(x)
  # The body of the method goes here.
  # Usually, the method body runs to completion without exceptions
  # and returns to its caller normally.

rescue
  # Exception-handling code goes here.
  # If an exception is raised within the body of the method, or if
  # one of the methods it calls raises an exception, then control
  # jumps to this block.

else
  # If no exceptions occur in the body of the method
  # then the code in this clause is executed.

ensure
  # The code in this clause is executed no matter what happens in the
  # body of the method. It is run if the method runs to completion, if
  # it throws an exception, or if it executes a return statement.

end
```

13.1.3. Invocando un Método en un Objeto

13.1.4. Definiendo Métodos Singleton

Si `def` va seguido de una expresión que determina un objeto, seguida de `.nombre_de_metodo` el método definido es sólo para el objeto específico.

```

o = "message"      # A string is an object
def o.printme      # Define a singleton method for this object
  puts self
end
o.printme          # Invoke the singleton

```

A un método de este tipo se le denomina *método singleton*.

Los métodos de clase como `Socket.tcp` y `File.delete` son, de hecho, métodos singleton.

Ruby implementations typically treat `Fixnum` and `Symbol` values as *immediate values* rather than as true object references.

Debido a ello los objetos `Fixnum` y `Symbol` no pueden tener métodos singleton.

For consistency, singletons are also prohibited on other Numeric objects.

13.1.5. Borrando (Undefining) Métodos

Methods are defined with the `def` statement and may be undefined with the `undef` statement:

```

def sum(x,y); x+y; end      # Define a method
puts sum(1,2)                # Use it
undef sum                     # And undefine it

```

In this code, the `def` statement defines a global method, and `undef` undefines it. `undef` also works within classes to undefine the instance methods of the class. Interestingly, `undef` can be used to undefine inherited methods, without affecting the definition of the method in the class from which it is inherited. Suppose class `A` defines a method `m`, and class `B` is a subclass of `A` and therefore inherits `m`. If you don't want to allow instances of class `B` to be able to invoke `m`, you can use `undef m` within the body of the subclass.

`undef` is not a commonly used statement. In practice, it is much more common to redefine a method with a new `def` statement than it is to undefine or delete the method.

Note that the `undef` statement must be followed by a single identifier that specifies the method name. It cannot be used to undefine a singleton method in the way that `def` can be used to define such a method.

Within a class or module, you can also use `undef_method` (a private method of `Module`) to undefine methods. Pass a symbol representing the name of the method to be undefined.

13.2. Nombres de Métodos

By convention, method names begin with a lowercase letter. (Method names can begin with a capital letter, but that makes them look like constants.) When a method name is longer than one word, the usual convention is to separate the words with underscores `like_this` rather than using mixed case `likeThis`.

Method names may (but are not required to) end with an equals sign, a question mark, or an exclamation point.

An equals sign suffix signifies that the method is a setter that can be invoked using assignment syntax.

The question mark and exclamation point suffixes have no special meaning to the Ruby interpreter, but they are allowed because they enable two extraordinarily useful naming conventions.

The first convention is that any method whose name ends with a question mark returns a value that answers the question posed by the method invocation.

Methods like these are called predicates and. Predicates typically return one of the Boolean values `true` or `false`, but this is not required, as any value

The second convention is that any method whose name ends with an exclamation mark should be used with caution.

13.2.1. Métodos Operadores

Many of Ruby's operators, such as `+`, `*`, and even the array index operator `[]`, are implemented with methods that you can define in your own classes.

You define an operator by defining a method with the same “name” as the operator. (The only exceptions are the unary plus and minus operators, which use method names `+@` and `-@`.)

13.2.2. Alias de Métodos

It is not uncommon for methods in Ruby to have more than one name. The language has a keyword `alias` that serves to define a new name for an existing method. Use it like this:

```
alias aka also_known_as # alias new_name existing_name
```

13.3. Métodos y Paréntesis

Ruby allows parentheses to be omitted from most method invocations. In simple cases, this results in clean-looking code. In complex cases, however, it causes syntactic ambiguities and confusing corner cases.

13.4. Argumentos de un Método

13.4.1. Parámetros por Defecto

When you define a method, you can specify default values for some or all of the parameters. If you do this, then your method may be invoked with fewer argument values than the declared number of parameters. If arguments are omitted, then the default value of the parameter is used in its place. Specify a default value by following the parameter name with an equals sign and a value:

```
def prefix(s, len=1)
  s[0,len]
end

prefix("Ruby", 3)      # => "Rub"
prefix("Ruby")         # => "R"
```

Los valores por defecto pueden ser expresiones arbitrarias en las que aparezcan parámetros previos:

```
# Return the last character of s or the substring from index to the end
def suffix(s, index=s.size-1)
  s[index, s.size-index]
end
```

Los parámetros por defecto son evaluados cuando se invoca al método, no cuando es compilado.

En este ejemplo el valor por defecto `a = []` produce un nuevo array en cada invocación en vez de reutilizar un único array que fuera creado cuando se define el método:

```
# Append the value x to the array a, return a.
# If no array is specified, start with an empty one.
def append(x, a[])
  a << x
end
```

13.4.2. Listas de Argumentos de Longitud Variable y Arrays

Sometimes we want to write methods that can accept an arbitrary number of arguments. To do this, we put an * before one of the method's parameters. Within the body of the method, this parameter will refer to an array that contains the zero or more arguments passed at that position. For example:

```
# Return the largest of the one or more arguments passed
def max(first, *rest)
  # Assume that the required first argument is the largest
  max = first
  # Now loop through each of the optional arguments looking for bigger ones
  rest.each {|x| max = x if x > max }
  # Return the largest one we found
  max
end

max(1)      # first=1, rest=[]
max(1,2)    # first=1, rest=[2]
max(1,2,3)  # first=1, rest=[2,3]
```

Pasando Arrays a Métodos que Esperan una Lista de Argumentos

```
# Return the largest of the one or more arguments passed
def max(first, *rest)
  ...
end

data = [3, 2, 1]
m = max(*data)  # first = 3, rest=[2,1] => 3

A veces nos referimos a este uso de * como el operador de splat

m = max(data)  # first = [3,2,1], rest=[] => [3,2,1]

# Convert the Cartesian point (x,y) to polar (magnitude, angle) coordinates
def polar(x,y)
  return [ Math.hypot(y,x), Math.atan2(y,x) ]
end

# Convert polar coordinates to Cartesian coordinates
def cartesian(magnitude, angle)
  [magnitude*Math.cos(angle), magnitude*Math.sin(angle)]
end

x,y = cartesian(*polar(x, y))
```

Los Enumeradores son Splattables

Los enumeradores `Enumerator` en son objetos con métodos como `each`, `each_with_index` y `next`:

```
ruby-1.9.2-head :015 > z = "hello".each_char
=> #<Enumerator: "hello":each_char>
ruby-1.9.2-head :016 > z.next
=> "h"
ruby-1.9.2-head :017 > z.next
=> "e"
```

```
ruby-1.9.2-head :022 > z.class.instance_methods(false)
=> [:each, :each_with_index, :each_with_object, :with_index,
:with_object, :next_values, :peek_values, :next, :peek, :feed,
:rewind, :inspect]
```

Los enumeradores en Ruby 1.9 son objetos splattables:

```
max(*"hello world".each_char) # => 'w'
```

13.4.3. Asignando Argumentos a los Parámetros

Supongamos que tenemos un método que es declarado con

- o parámetros ordinarios,
- d parámetros con valores por defecto
- un parámetro array prefijado con *

Supongamos que este método es invocado con a argumentos.

- Si a < o se genera una excepción `ArgumentError`
- Si a >= o y a <= o+d entonces los a-o parámetros con defecto mas a la izquierda tendrán argumentos asignados. Los restantes o+d-a tendrán los valores por defecto
- Si a > o + d entonces el array de parámetros cuyo nombre se prefijo con * tendrá a-o-d argumentos. En otro caso estará vacío

13.4.4. Hashes para Argumentos con Nombre

Es difícil recordar el orden de los argumentos de un método que requiere mas de tres argumentos. Cuando el número de argumentos es grande, haremos que los argumentos se empaqueten en un hash. Este es un ejemplo:

```
~/rubystesting$ cat -n shell.rb
 1
 2 def ls(args)
 3   hidden    = args[:hidden]?    '-a' : ''
 4   dir       = args[:dir]?      '-d' : ''
 5   reverse   = args[:reverse]? '-r' : ''
 6   time      = args[:time]?    '-t' : ''
 7   recursive = args[:recursive]? '-R' : ''
 8   regexp    = args[:regexp] || Regexp.new('.')
 9
10   files = %x{ls #{hidden} #{reverse} #{dir} #{recursive} #{time} 2>&1}.split(/\n/)
11   files.select { |f| f =~ regexp }
12 end
13
14 puts  ls(:reverse =>1, :time => 1, :regexp => /~m/)
15 puts  '*****'
16 puts  ls(:hidden =>1,  :regexp => /.ms/)
```

Podríamos haber llamado a ls así

```
ls({:reverse =>1, :time => 1, :regexp => /~m/})
```

Pero ruby nos permite omitir las llavitas cuando el último argumento del método es el hash (o si es el único argumento que le sigue es un argumento bloque prefijado de &). Un hash sin llaves se denomina *bare hash*.

```
ls(:time => 1, :reverse =>1, :regexp => /~m/)
```

Es posible también omitir los paréntesis en la llamada:

```
puts ls :hidden =>1, :regexp => /~.[ms]/
```

Si se omiten los paréntesis se *deberán también omitir las llaves* ya que *si unas llaves siguen a un nombre de método Ruby lo interpreta como que estamos pasando un argumento de tipo bloque*.

Ejecución:

```
~/rubytesting$ ruby shell.rb
myutf.rb
methodmethod.rb
memoize
map2funcWithLambdas.rb
map2func.rb
*****
.myutfstring.rb.swp
.shell.rb.swp
.svn
useGreeter.rb
useSudoku.rb
```

13.4.5. Parámetros con Nombre. Named Parameters in Ruby 2.0

In computer programming, named parameters or keyword arguments refer to a computer language's support for function calls that clearly state the name of each parameter within the function call itself.

Ruby 2.0 admite parámetros con nombre.

```
def foo(bar: '12345', baz: 'abcdef')
  [bar, baz]
end

foo                      # => ["12345", "abcdefg"]
foo(bar: 'bar')          # => ["bar", "abcdef"]
foo(bar: 'bar', baz: 'baz') # => ["bar", "baz"]

[2] pry(main)> require './named_parameters2'
["12345", "abcdef"]
["bar", "abcdef"]
["bar", "baz"]
=> true
```

Si se llama con un nombre no declarado se produce un `ArgumentError`:

```
[3] pry(main)> foo(chuchu: 'chuchu')
ArgumentError: unknown keyword: chuchu
from (pry):3:in '__pry__'
```

Also, you can combine them with regular parameters.

```

def foo(value, bar: '12345', baz: 'abcdef')
  [value, bar, baz]
end

foo('VALUE')          # => ["VALUE", "12345", "abcdefg"]
foo('VALUE', bar: 'bar') # => ["VALUE", "bar", "abcdef"]
foo('VALUE', bar: 'bar', baz: 'baz') # => ["VALUE", "bar", "baz"]

```

Se pueden combinar con splats:

```
[~/rubytesting/TheRubyProgrammingLanguage/Chapter6MethodsProcsLambdasAndClosures]$ cat named_p
require 'pp'
```

```

def cycle(first_value, *values, name: 'default')
  return [first_value, values, name]
end

```

```
pp cycle(1, 2, 3, 4, name: 'Pedro') # [1, [2, 3, 4], "Pedro"]
```

You can stop an `ArgumentError` being raised and deal with arbitrary keyword arguments by using a double-splat (double asterisk) to capture all keywords not already assigned and puts them in a hash:

```
[~/rubytesting/TheRubyProgrammingLanguage/Chapter6MethodsProcsLambdasAndClosures]$ cat named_p
require 'pp'
```

```

def foo(bar: '12345', baz: 'abcdef', **rest)
  [bar, baz, rest]
end

```

```
pp foo(bar: 'bar', chuchu: 'chuchu', chachi: 'chachi') # ["bar", "abcdef", {:chuchu=>"chuchu", :chachi=>"chachi"}]
```

Skipping the default value. Required keyword arguments in Ruby 2.1

Ruby 2.1 introduces required keyword arguments. You can use required argument by skipping the default value.

You'll get a `ArgumentError` if you don't supply the argument:

```

def exclaim(text, exclamation: '!', number:)
  text + exclamation * number
end

exclaim('Yo', number: 5) # => 'Yo!!!!'
exclaim('Yo') # raises: ArgumentError: missing keyword: number

```

Véase

1. Véase Ruby 2.0 - Getting Started and Named Parameters (YouTube)
2. Keyword Arguments in Ruby 2.0 por Chris Zetter

13.4.6. Argumentos Bloque

- Un *bloque* es un trozo de código Ruby asociado con la invocación de un método.
- Los bloques, a diferencia de los métodos, son anónimos
- Un *iterador* es un método que espera un bloque

- En Ruby, cualquier invocación de un método puede ir seguida de un bloque y
- Cualquier método puede invocar el código en el bloque asociado que le sigue mediante la sentencia `yield`.

Una de las características de los bloques es su carácter anónimo. Tampoco son pasados al método en la forma tradicional: *no figuran en la lista de argumentos*. En vez de eso, son invocados por medio de `yield`. Por ejemplo:

```
~/rubystesting$ cat -n iter.rb
1 def seq(n,m,c)
2   i=0
3   while (i < n)
4     yield i*m+c
5     i += 1
6   end
7 end
8
9 seq(5,2,2) { |x| puts x }
```

Que usamos así:

```
~/rubystesting$ ruby iter.rb
2
4
6
8
10
```

Si se prefiere tener un mayor control sobre el bloque (por ejemplo, se desea pasar el bloque como argumento a otro método) lo añadiremos como argumento final a nuestro método, prefijado por un ampersand `&`. Dentro del método, el argumento asociado con el bloque pertenecerá a la clase `Proc` y se podrá invocar usando el método `call` del objeto `Proc` en vez de `yield`:

```
~/rubystesting$ cat -n iter2.rb
1 def seq(n,m,c, &b)
2   i=0
3   while (i < n)
4     b.call(i*m+c)
5     i += 1
6   end
7 end
8
9 seq(5,2,2) { |x| puts x }
```

Nótese que el asunto afecta a la definición del método pero que la forma de la llamada (línea 9) no cambia.

Ejecución:

```
~/rubystesting$ ruby iter2.rb
2
4
6
8
10
```

LLamada con Paso de Objetos Proc de Forma Explícita

También podemos construir un objeto `Proc` y pasarlo explícitamente al método. En este caso se trata de un objeto *ordinario* y ni usamos el `&` ni la llamada tiene la sintaxis *con bloque*:

```
# This version expects an explicitly-created Proc object, not a block
def sequence4(n, m, c, b) # No ampersand used for argument b
  i = 0
  while(i < n)
    b.call(i*m + c)      # Proc is called explicitly
    i += 1
  end
end

p = Proc.new {|x| puts x} # Explicitly create a Proc object
sequence4(5, 2, 2, p)    # And pass it as an ordinary argument
```

Los Argumentos Bloque Prefijados con & Deben Ser los Últimos

Los siguientes dos métodos son legales:

```
def sequence5(args, &b) # Pass arguments as a hash and follow with a block
  n, m, c = args[:n], args[:m], args[:c]
  i = 0
  while(i < n)
    b.call(i*m + c)
    i += 1
  end
end

# Expects one or more arguments, followed by a block
def max(first, *rest, &block)
  max = first
  rest.each {|x| max = x if x > max }
  block.call(max)
  max
end
```

1. Vale la pena resaltar que la sentencia `yield` también funciona sobre un parámetro definido con `&`.
2. Podemos invocarlo con `yield` incluso si el parámetro ha sido convertido a un objeto `Proc` y pasado como un argumento.

Utilizando & en la Invocación de Métodos

& como Operador que convierte un Proc en un Bloque

- `&` can be used in both method definitions and method invocations.
- `&` prefijando un parámetro en una definición de método nos permite explicitar el bloque asociado con la invocación y usarlo como un objeto de la clase `Proc` dentro del cuerpo del método
- Cuando se usa `&` en una invocación/llamada prefijando un objeto `Proc` trata al `Proc` como si fuera un bloque que sigue la invocación

```

a, b = [1,2,3], [4,5]                      # Start with some data.
sum = a.inject(0) { |total,x| total+x }    # => 6. Sum elements of a.
sum = b.inject(sum) { |total,x| total+x }   # => 15. Add the elements of b in.

```

Al reescribirlo como sigue evitamos que el intérprete Ruby parsee dos veces el mismo código:

```

a, b = [1,2,3], [4,5]                      # Start with some data.
summation = Proc.new { |total,x| total+x } # A Proc object for summations.
sum = a.inject(0, &summation)            # => 6  Llamada como argumento
sum = b.inject(sum, &summation)          # => 15 ordinario

```

Sintáxis de la llamada El Proc construido a partir del bloque se pasa como el último argumento en la llamada al método separándolo por comas del resto de los argumentos:

```

ruby-1.9.2-head :008 > def f(a,b); yield a+b; end
=> nil
ruby-1.9.2-head :009 > f(1,2) { |x| puts x }
3
=> nil

```

No podemos llamarlo como si fuera el bloque que sigue:

```

ruby-1.9.2-head :010 > qq = lambda { |x| puts x }
=> #<Proc:0x007f93e387d3d0@(irb):10 (lambda)>
ruby-1.9.2-head :011 > f(1,2) &qq
LocalJumpError: no block given (yield)

```

Así tampoco:

```

ruby-1.9.2-head :012 > f(1,2, qq)
ArgumentError: wrong number of arguments (3 for 2)

```

Es necesario hacer la conversión con `&` y pasarlo como último argumento:

```

ruby-1.9.2-head :013 > f(1,2, &qq)
3
=> nil

```

Usando `&` sobre el Nombre de un Método

Realmente, la invocación con `&` puede usarse con cualquier objeto que disponga de un método `to_proc`.

En Ruby 1.9 la clase `Symbol` dispone de un método `to_proc`, lo que permite a los símbolos ser prefijados con `&` y ser pasados a los iteradores:

```

pry(main)> words = ['and', 'but', 'car']      # An array of words
=> ["and", "but", "car"]
pry(main)> uppercase = words.map &:upcase      # Convert to uppercase with String.upcase
=> ["AND", "BUT", "CAR"]

```

Este código es equivalente a:

```

upper = words.map { |w| w.upcase } # This is the equivalent code with a block

```

13.5. Procs y Lambdas

- Los *bloques* son estructuras sintácticas en Ruby;
- Los bloques *no son objetos y no pueden ser manipulados como objetos.*
- Es sin embargo posible crear objetos que representan a un bloque.
- Dependiendo de como se crea el objeto se le denomina *proc* o *lambda*
- Tanto las lambdas como los proc son objetos de la clase Proc

13.5.1. Creacion de Procs

Asociando un bloque con argumento prefijado por &

```
# This method creates a proc from a block
def makeproc(&p)  # Convert associated block to a Proc and store in p
  p               # Return the Proc object
end
```

Como retorna un Proc podemos usarlo como una suerte de constructor:

```
adder = makeproc {|x,y| x+y }
```

```
sum = adder.call(2,2)  # => 4
```

Proc.new

```
p = Proc.new {|x,y| x+y }
```

Si Proc.new es invocado sin bloque desde el interior de un método que tiene asociado un bloque entonces retorna un Proc representando al bloque asociado.

Así pues, estos dos métodos son equivalentes:

```
def invoke(&b)      def invoke
  b.call            Proc.new.call
end                end
```

Kernel.lambda

```
is_positive = lambda {|x| x > 0 }
```

Kernel.proc

En Ruby 1.9 proc es un sinónimo de Proc.new

```
ruby-1.9.2-head :003 > f = proc { |x| x + 1}
=> #<Proc:0x007ff03c236b60@(irb):3>
ruby-1.9.2-head :004 > f[2]
=> 3
```

Lambda Literals

En el estilo clásico:

```
succ = lambda { |x| x+1}
```

En Ruby 1.9:

- Reemplazar `lambda` por `->`
- a continuación la lista de parámetros
- usando paréntesis como delimitador de los parámetros

```
succ = ->(x){ x+1 }
```

```
succ.call(2)      # => 3
```

La lista de argumentos puede incluir la declaración de variables locales que se garantiza no sobre-escribirán a las variables con el mismo nombre en el ámbito que lo contiene:

```
# This lambda takes 2 args and declares 3 local vars
f = ->(x,y; i,j,k) { ... }
```

Es posible dar valores por defecto a los argumentos:

```
zoom = ->(x,y,factor=2) { [x*factor, y*factor] }
```

Como en las declaraciones de métodos, los paréntesis son opcionales:

```
succ = ->x { x+1 }
f = -> x,y; i,j,k { ... }
zoom = ->x,y,factor=2 { [x*factor, y*factor] }
```

La lambda minimal que no toma argumentos y retorna `nil` vendría dada por:

```
->{}
```

Un ejemplo de expresividad:

```
[12] pry(main)> class Proc
[12] pry(main)*   def *(other)
[12] pry(main)*     ->(*args) { self.call(other.call(*args)) }
[12] pry(main)*   end
[12] pry(main)* end
=> :*
[13] pry(main)> succ = ->x{x+1}
=> #<Proc:0x007fc7a2a2f460@(pry):28 (lambda)>
[14] pry(main)> square = ->x{x*x}
=> #<Proc:0x007fc7a34df838@(pry):29 (lambda)>
[15] pry(main)> ss = succ*square
=> #<Proc:0x007fc7a349f4b8@(pry):25 (lambda)>
[17] pry(main)> ss[2]
=> 5
[18] pry(main)> ss = ->x{x+1} * ->x{x*x}
=> #<Proc:0x007fc7a3415100@(pry):25 (lambda)>
[19] pry(main)> ss[2]
=> 5
```

Si queremos pasar una lambda a un método que espera un bloque es necesario prefijarla de &

```
[21] pry(main)> data = [3,2,1,4,2]
=> [3, 2, 1, 4, 2]
[22] pry(main)> data.sort {|a,b| b-a }      # The block version
=> [4, 3, 2, 1]
[23] pry(main)> data.sort &->(a,b){ b-a } # The lambda literal version
=> [4, 3, 2, 1]
```

13.5.2. Invocación de Procs y Lambdas

1. Procs y lambdas no son métodos: son objetos
2. Si `f` es un Proc no podemos invocar `f` como si fuera un método pero si que podemos invocar métodos (como `call`) en `f`

```
f = Proc.new {|x,y| 1.0/(1.0/x + 1.0/y) }
z = f.call(x,y)
```

3. Se redefine el operador de acceso a arrays para que funcione como `call`

```
z = f[x,y]
```

4. En Ruby 1.9 se pueden prefijar los paréntesis con un punto:

```
z = f.(x,y)
```

No se trata de un operador, sino de [syntactic sugar](#)¹. Puede utilizarse sobre cualquier objeto que disponga de un método `call`

13.5.3. La Aridad de un Proc

1. La *aridad* de un proc o lambda es el número de argumentos que espera.
2. Los objetos `Proc` disponen de un método `arity` que retorna el número de argumentos que espera

```
lambda{||}.arity          # => 0. No arguments expected
lambda{|x| x}.arity       # => 1. One argument expected
lambda{|x,y| x+y}.arity  # => 2. Two arguments expected
```

3. Si el Proc acepta un número arbitrario de argumentos mediante un argumento final prefijado con `*`, el método `arity` retorna un número negativo de la forma `-n-1`. Esto indica que requiere `n` argumentos pero que puede tener también argumentos adicionales.

```
lambda { |*args| }.arity        # => -1. ~-1 = -(-1)-1 = 0 arguments required
lambda { |first, *rest| }.arity # => -2. ~-2 = -(-2)-1 = 1 argument required
```

13.5.4. Igualdad de Procs

La clase `Proc` define un método `==` que devuelve `true` si, y sólo si un `Proc` es el `clone` o el duplicado del otro.

```
p = lambda { |x| x*x }
q = p.dup
p == q                      # => true: the two procs are equal
p.object_id == q.object_id   # => false: they are not the same object
lambda { |x| x*x } == lambda { |x| x*x } # => false
```

¹ In computer science, syntactic sugar is syntax within a programming language that is designed to make things easier to read or to express. It makes the language "sweeter" for human use: things can be expressed more clearly, more concisely, or in an alternative style that some may prefer.

13.5.5. En que Forma las Lambdas Difieren de los Procs

1. Un proc es la representación como objeto de un bloque y se comporta como un bloque
2. Una lambda se comporta mas como un método que como un bloque
3. LLamar una lambda es como invocar un método
4. LLamar un proc es como yielding un bloque
5. El predicado `lambda?` determina si un objeto `Proc` es una lambda o un proc

Return en bloques, procs y lambdas

1. `return` nos saca del método en el que estamos, no del *bloque* en el que estamos:

```
def test
  puts "entering method"
  1.times { puts "entering block"; return } # Makes test method return
  puts "exiting method" # This line is never executed
end
test
```

2. `return` nos saca del método en el que estamos, no del *proc* en el que estamos:

```
def test
  puts "entering method"
  p = Proc.new { puts "entering proc"; return }
  p.call # Invoking the proc makes method return
  puts "exiting method" # This line is never executed
end
test
```

3. Un `return` dentro de un proc puede sorprender un poco:

```
def procBuilder(message) # Create and return a proc
  Proc.new { puts message; return } # return returns from procBuilder
end

def test
  puts "entering method"
  p = procBuilder("entering proc")
  p.call # Prints "entering proc" and raises LocalJumpError!
  puts "exiting method" # This line is never executed
end
test
```

Cuando lo ejecutamos obtenemos:

```
$ ruby procjump.rb
entering method
entering proc
procjump.rb:2:in 'block in procBuilder': unexpected return (LocalJumpError)
```

Al convertir el bloque en un objeto podemos pasar el bloque y utilizarlo fuera de su contexto.

4. Un `return` dentro de una lambda retorna desde la lambda no desde el método que contiene a la lambda

```
def test
  puts "entering method"
  p = lambda { puts "entering lambda"; return }
  p.call          # Invoking the lambda does not make the method return
  puts "exiting method" # This line *is* executed now
end
test
```

5. Eso significa que no tenemos que preocuparnos por que un `return` en una lambda caiga fuera de contexto

```
def lambdaBuilder(message)      # Create and return a lambda
  lambda { puts message; return } # return returns from the lambda
end

def test
  puts "entering method"
  l = lambdaBuilder("entering lambda")
  l.call          # Prints "entering lambda"
  puts "exiting method" # This line is executed
end
test
```

Break en bloques, procs y lambdas

1. Cuando se usa dentro de un bucle `break` transfiere el control a la primera sentencia que sigue al bucle:

```
while(line = gets.chomp)      # A loop starts here
  break if line == "quit"    # If this break statement is executed...
  puts eval(line)
end
puts "Good bye"           # ...then control is transferred here
```

2. Cuando se usa dentro de un bloque `break` transfiere el control fuera del bloque, fuera del iterador hasta la primera expresión que sigue a la invocación del iterador:

```
f.each do |line|          # Iterate over the lines in file f
  break if line == "quit\n" # If this break statement is executed...
  puts eval(line)
end
puts "Good bye"           # ...then control is transferred here
```

3. Cuando creamos un proc con `Proc.new`, el iterador desde el que se sale es `Proc.new`. Eso significa que si el proc con el `break` es llamado fuera de contexto obtendremos un error:

```
$ cat procbreak.rb
def test
  puts "entering test method"
  proc = Proc.new { puts "entering proc"; break }
  proc.call          # LocalJumpError: iterator has already returned
```

```

    puts "exiting test method"
end
test

$ ruby procbreak.rb
entering test method
entering proc
procbreak.rb:3:in `block in test': break from proc-closure (LocalJumpError) `
```

4. Si creamos el proc como argumento del iterador:

```

$ cat procbreakwithampersand.rb
def iterator(&proc)
  puts "entering iterator"
  proc.call # invoke the proc
  puts "exiting iterator" # Never executed if the proc breaks
end

def test
  iterator { puts "entering proc"; break }
end
test
```

la cosa funciona bien porque el proc está en contexto:

```

$ ruby procbreakwithampersand.rb
entering iterator
entering proc
```

5. Dentro de una lambda `break` funciona igual que dentro de un método:

```

$ cat lambdabreak.rb
def test
  puts "entering test method"
  lambda = lambda { puts "entering lambda"; break; puts "exiting lambda" }
  lambda.call
  puts "exiting test method"
end
test
```

de hecho el `break` actúa como un `return` y nos saca de la lambda:

```

$ ruby lambdabreak.rb
entering test method
entering lambda
exiting test method
```

Otras sentencias de control en bloques, procs y lambdas

1. Un `next` al nivel mas alto produce el mismo efecto en un bloque, proc o lambda: hace que se retorne del `yield` o del `call` que invoca el bloque, proc o lambda
2. Si `next` va seguido de una expresión es usada como valor de retorno del bloque, proc o lambda
3. `redo` transfiere el control al comienzo del bloque o lambda

4. El uso de `retry` no esta permitido ni en procs ni en lambdas

5. `raise`

6. `rescue`

Paso de argumentos en procs y lambdas

1. Los valores de los argumentos que siguen un `yield` son asignados a los parámetros de un bloque siguiendo reglas que son mas próximas a las reglas de asignación de variables que a las reglas de invocación de métodos

2. Un código como:

<code>yield k,v</code>	<code>do key, value </code>
	<code>...</code>
	<code>end</code>

funciona como

```
key, value = k, v
```

3. `def two; yield 1,2; end # An iterator that yields two values`
`two {|x| p x } # Ruby 1.8: warns and prints [1,2],`
`two {|x| p x } # Ruby 1.9: prints 1, no warning`
`two {||x| p x } # Either version: prints [1,2]; no warning`
`two {||x,| p x } # Either version: prints 1; no warning`

4. En Ruby 1.9 el argumento splat no tiene por que ser el último:

```
def five; yield 1,2,3,4,5; end      # Yield 5 values
five do |head, *body, tail|        # Extra values go into body array
  print head, body, tail          # Prints "1[2,3,4]5"
end
```

5. La sentencia `yield` permite bare hashes:

```
def hashiter; yield :a=>1, :b=>2; end # Note no curly braces
hashiter {|hash| puts hash[:a]}       # Prints 1
```

6. Si el parámetro final es un bloque puede ser prefijado con `&` para indicar que será utilizado para recibir el bloque asociado con la invocación del bloque:

```
# This Proc expects a block
printer = lambda {&|&b| puts b.call} # Print value returned by b
printer.call { "hi" }                # Pass a block to the block!
```

7. Es legal dar valores por defecto a los parámetros de un bloque:

```
ruby-1.9.2-head :001 > [1,2,3].each { |x,y=2| puts x*y }
2
4
6
=> [1, 2, 3]
```

o también:

```
ruby-1.9.2-head :007 > [1,2,3].each &gt;(x, y=2) { puts x*y }
2
4
6
=> [1, 2, 3]
```

8. La sentencia `yield` usa *semántica yield* mientras que la invocación de un método usa *semántica de invocación*

9. Invocar un proc sigue *semántica yield* mientras que invocar una lambda sigue *semántica de invocación*

```
p = Proc.new {|x,y| print x,y }
p.call(1)      # x,y=1:    nil used for missing rvalue: Prints 1nil
p.call(1,2)    # x,y=1,2:   2 lvalues, 2 rvalues:           Prints 12
p.call(1,2,3)  # x,y=1,2,3: extra rvalue discarded:       Prints 12
p.call([1,2])  # x,y=[1,2]: array automatically unpacked: Prints 12
```

10. Las Lambdas no son tan flexibles; al igual que los métodos, son mas estrictas con el número de argumentos pasados:

```
l = lambda {|x,y| print x,y }
l.call(1,2)    # This works
l.call(1)      # Wrong number of arguments
l.call(1,2,3)  # Wrong number of arguments
l.call([1,2])  # Wrong number of arguments
l.call(*[1,2]) # Works: explicit splat to unpack the array
```

13.5.6. Enumeradores

Iteradores Externos

La clase `Enumerator` permite la iteración tanto interna como externa.

- When the iterator controls the iteration, the iterator is an *internal iterator*.
- Clients that use an *external iterator* must advance the traversal and request the `next` element explicitly from the iterator.

Una manera de crear objetos `Enumerator` es mediante el método `to_enum` (en la clase `Object`):

```
[1] pry(main)> a = [1, 2, "cat" ]
=> [1, 2, "cat"]
[3] pry(main)> ea = a.to_enum
=> #<Enumerator: ...>
[4] pry(main)> ea.next
=> 1
[5] pry(main)> ea.next
=> 2
[6] pry(main)> ea.next
=> "cat"
[7] pry(main)> ea.next
StopIteration: iteration reached an end
from (pry):7:in 'next'
```

```

[8] pry(main)> ea.rewind
=> #<Enumerator: ...>
[9] pry(main)> ea.next
=> 1
[10] pry(main)> ea.next
=> 2
[11] pry(main)> ea.peek
=> "cat"

```

También puede usarse con hashes:

```

28] pry(main)> h = { a: "hello", b: "world" }
=> {:a=>"hello", :b=>"world"}
[29] pry(main)>
[30] pry(main)> eh = h.to_enum
=> #<Enumerator: ...>
[31] pry(main)> eh.size
=> nil
[32] pry(main)> eh.next
=> [:a, "hello"]
[33] pry(main)> eh.next_values
=> [[:b, "world"]]
[34] pry(main)> eh.rewind
=> #<Enumerator: ...>

```

El método `to_enum` está definido en la clase `Object`.

The call `obj.to_enum(method = :each, *args)` creates a new `Enumerator` which will enumerate by calling method `:each` on `obj`, passing `args` if any.

```

[2] pry(main)> class Tutu
[2] pry(main)*   def chuchu(x)
[2] pry(main)*     if block_given?
[2] pry(main)*       yield x+1
[2] pry(main)*       yield x+2
[2] pry(main)*       yield x+3
[2] pry(main)*     else
[2] pry(main)*       to_enum(:chuchu, x) { 3 }
[2] pry(main)*     end
[2] pry(main)*   end
[2] pry(main)* end
=> :chuchu
[3] pry(main)> t = Tutu.new
=> #<Tutu:0x007f9e6cb5af60>
[5] pry(main)> t.chuchu(10) { |x| puts x }
11
12
13
=> nil
[6] pry(main)> e = t.chuchu(10)
=> #<Enumerator: ...>
[7] pry(main)> e.next
=> 11
[8] pry(main)> e.next
=> 12
[9] pry(main)> e.size
=> 3

```

with_index El método `with_index(offset = 0) {|(*args), idx| ...}` (o bien `with_index(offset = 0)`) iterates the given block for each element with an index, which starts from offset. If no block is given, returns a new Enumerator that includes the index, starting from offset

```
[40] pry(main)> h = { a: "hello", b: "world" }
=> {:a=>"hello", :b=>"world"}
[41] pry(main)> eh = h.to_enum
=> #<Enumerator: ...>
[42] pry(main)> eh.with_index.next
=> [[:a, "hello"], 0]
[44] pry(main)> eh.with_index { |x, i| puts "#{x.inspect}, #{i}" }
[:a, "hello"], 0
[:b, "world"], 1
=> {:a=>"hello", :b=>"world"}
```

Construcción de un Enumerador

1. As well as creating `Enumerators` from existing collections, you can create an explicit enumerator, passing it a block
2. The code in the block will be used when the enumerator object needs to supply a fresh value to your program
3. However, the block isn't simply executed from top to bottom
4. Instead, **the block is executed as a *coroutine***² with the rest of your program's code
5. Execution starts at the top and pauses when the block `yields` a value to your code
6. When the code needs the `next` value, execution resumes at the statement following the `yields`
7. This lets you write enumerators that generate *infinite sequences*:

```
[~/ruby/PROGRAMMINGRUBYDAVETHOMAS]$ cat fib_enum.rb
@fib = Enumerator.new do |y|
  a, b = 0, 1
  loop do
    a, b = b, a + b
    y.yield a
  end
end
```

`Enumerator.new` receives a block that is used to define the iteration. A `yielder` object `y` is given to the block as a parameter and can be used to `yield` a value by calling the `yield` method (aliased as `<<`).

```
[~/ruby/PROGRAMMINGRUBYDAVETHOMAS]$ pry
[1] pry(main)> require './fib_enum'
=> true
[2] pry(main)> @fib.next
=> 1
[3] pry(main)> @fib.next
=> 1
```

² Coroutines are computer program components that generalize subroutines for nonpreemptive multitasking, by allowing multiple entry points for suspending and resuming execution at certain locations.

Coroutines are well-suited for implementing more familiar program components such as cooperative tasks, exceptions, event loop, iterators, infinite lists and pipes.

```
[4] pry(main)> @fib.next
=> 2
[6] pry(main)> @fib.first
=> 1
[7] pry(main)> @fib.take(5)
=> [1, 1, 2, 3, 5]
```

La clase `Enumerator` incluye el módulo `Enumerable` de donde toma, entre otros, los métodos `first` y `take`.

```
[25] pry(main)> Enumerator.ancestors
=> [Enumerator, Enumerable, Object, PP::ObjectMixin, Kernel, BasicObject]
```

Enumeradores Infinitos

1. You have to be slightly careful with `Enumerators` that can generate infinite sequences
2. Some of the regular `Enumerable` methods such as `count` and `select` will happily try to read the whole enumeration before returning a result
3. If you want a version of `select` that works with infinite sequences, you'll need to write it yourself
4. Here's a version that gets passed an `Enumerator` and a block and returns a new `Enumerator` containing values from the original for which the block returns `true`

```
[1] pry(main)> require './fib_enum'
=> true
[2] pry(main)> require './grep8'
=> true
[3] pry(main)> e = @fib.grep8 { |x| x % 3 == 0}
=> #<Enumerator: ...>
[4] pry(main)> e.next
=> 3
[5] pry(main)> e.next
=> 21
[6] pry(main)> e.next
=> 144
[7] pry(main)> e.first(5)
=> [3, 21, 144, 987, 6765]
```

```
[9] pry(main)> .cat grep8.rb
class Enumerator
  def grep8(&block)
    Enumerator.new do |y|
      self.each do |x|
        y.yield x if block[x]
      end
    end
  end
end
```

Enumeradores Perezosos

Otra forma de crear iteradores infinitos en Ruby 2.0 es mediante la clase `Enumerator::Lazy`:

```

[16] pry(main)> e = (1..Float::INFINITY).lazy.map { |x| x*x }
=> #<Enumerator::Lazy: ...
[17] pry(main)> e.next
=> 1
[18] pry(main)> e.next
=> 4
[19] pry(main)> e.next
=> 9
[20] pry(main)> e.next
=> 16
[21] pry(main)> e.take(4).force
=> [1, 4, 9, 16]
[22] pry(main)> e.take(4)
=> #<Enumerator::Lazy: ...
[23] pry(main)> e.first(4)
=> [1, 4, 9, 16]

```

En este otro ejemplo creamos un enumerador perezoso `filter_map` que devuelve los elementos procesados por el bloque cuyo valor no es `nil`. Su funcionamiento sería así:

```

[1] pry(main)> require './lazy_enum'
=> true
[2] pry(main)> [1,2,3].map { |x| x*x if x % 2 != 0 }
=> [1, nil, 9]
[3] pry(main)> [1,2,3].filter_map { |x| x*x if x % 2 != 0 }
=> [1, 9]           # se elimina el nil

```

Para usarlo con rangos infinitos:

```

[4] pry(main)> (1..Float::INFINITY).lazy.filter_map{|i| i*i if i.even?}.first(5)
=> [4, 16, 36, 64, 100]

```

El método `lazy` se encuentra en el módulo `Enumerable` y retorna un enumerador perezoso `Enumerator::Lazy` que enumera/construye el valor cuando se necesita.

```

[~/ruby/PROGRAMMINGRUBYDAVETHOMAS]$ cat lazy_enum.rb
module Enumerable
  def filter_map(&block) # Para explicar la semantica
    map(&block).compact
  end
end

class Enumerator::Lazy
  def filter_map
    Lazy.new(self) do |yielder, *values|
      result = yield *values
      yielder << result if result # << es un alias de yield
    end
  end
end

```

The call `new(self) { |yielder, *values| ... }` creates a new Lazy enumerator using `self` as the coupled enumerator.

When the enumerator is actually enumerated (e.g. by calling `force` or `next`), `self` will be enumerated and each value passed to the given block.

The block then `yield` values back using `yielder.yield` or `yielder << ...`.

Ejemplo: Números primos

```
[~/rubytesting]$ cat primes.rb
def primes
  (1..Float::INFINITY).lazy.select do |n|
    ((2..n**0.5)).all? { |f| n % f > 0 }
  end
end

p primes.first(10)
[~/rubytesting]$ ruby primes.rb
[1, 2, 3, 5, 7, 11, 13, 17, 19, 23]
```

Véase

1. Ruby 2.0 Works Hard So You Can Be Lazy por Pat Shaughnessy. April 3rd 2013

13.5.7. Bloques para las Transacciones

You can use blocks to define a chunk of code that must be run under some kind of transactional control.

For example, you'll often open a file, do something with its contents, and then want to ensure that the file is closed when you finish.

A naive implementation (ignoring error handling) could look something like the following:

```
[~/ruby/PROGRAMMINGRUBYDAVETHOMAS]$ cat block_for_transaction.rb
class File
  def self.my_open(*args)
    result = file = File.new(*args)
    if block_given?
      result = yield file
      file.close
    end
    return result
  end
end

File.my_open("testfile", "r") do |file|
  while line = file.gets
    print line
  end
end
```

The responsibility for closing an open file has been shifted from the users of file objects back to the file objects themselves.

```
[~/ruby/PROGRAMMINGRUBYDAVETHOMAS]$ ruby block_for_transaction.rb
1
2
3
fin
```

If we wanted to implement this method properly, we'd need to ensure that we closed a file even if the code processing that file somehow aborted.

1. As a similar example, we could extend this `my_open` method to support some sort of rolling back transaction, making a backup copy of the file being open, and then restoring it if an exception is produced or deleting it if everything went fine.

Véase

1. Ruby Best Practices. Code Blocks: Ruby's Swiss Army Knife. 2009-07-07 written by Gregory Brown

13.5.8. Ejemplo: La Clase Filter

En este ejemplo - modificado de uno en el libro [3] - creamos una clase `Filter` que proveee un pequeño DSL para especificar restricciones que puedan ser usados sobre objetos `Enumerable` para filtrar los elementos producidos por el objeto `Enumerable`:

```
19 f = Filter.new
20 f.constraint { |x| x > 10 } .and { |x| x.even? } .and { |x| x % 3 == 0 }
21
22 puts (8..24).select(&f)
```

Del uso del objeto `Filter` deducimos que los objetos `Filter` debe tener métodos

1. `constraint` (línea 20)
2. `and` (línea 20)
3. Además por la forma de uso (línea 20) podemos ver que `constraint` devuelve un objeto `Filter`, lo mismo que `and`
4. `to_proc` (línea 22)

La ejecución resulta en:

```
[~/srcLPP/rubytesting/ruby_best_practices/chapter5_functional_programming_techniques]$ ruby fi
12
18
24
```

Este es el código de la clase:

```
[~/srcLPP/rubytesting/ruby_best_practices/chapter5_functional_programming_techniques]$ cat -n
1 class Filter
2   def initialize
3     @constraints = []
4   end
5
6   def constraint(&block)
7     @constraints << block
8     self
9   end
10  alias and constraint
11
12  def to_proc
13    ->(e) { @constraints.all? { |fn| fn[e] } }
14  end
15
16 end
```

1. *Method chaining*, is a common syntax for invoking multiple method calls in object-oriented programming languages.

```
person.name("Peter").age(21).introduce
```

2. Each method returns an object, allowing the calls to be chained together in a single statement.
3. Chaining is *syntactic sugar* which eliminates the need for intermediate variables.
4. A similar syntax is *method cascading*, where after the method call the expression evaluates to the current object, not the natural return value of the method.
5. Cascading can be implemented using method chaining by having the method return the current object itself (`self`).
6. Cascading is a key technique in fluent interfaces, and since chaining is widely implemented in object-oriented languages while cascading isn't, this form of *cascading-by-chaining* by returning `self` is often referred to simply as *chaining*.
7. Both chaining and cascading come from the Smalltalk language.
8. The call `@constraints.all? { |fn| fn[e] }` passes each element of the collection `@constraints` to the given block. The method returns `true` if the block never returns `false` or `nil`.

Véase

1. FluentInterface. Martin Fowler. 20 December 2005

13.6. Clausuras

13.6.1. Clausuras y Variables Compartidas

1. El término *clausura* se refiere a un objeto que es a la vez invocable como una función y que dispone de un binding o entorno de ejecución para esa función.
2. El objeto `Proc` resultante de la creación de un proc o una lambda no sólo nos proporciona un bloque ejecutable: también nos proporciona bindings para todas las variables que son usadas por el bloque
3. El bloque asociado con la invocación de `collect` usa `n`:

```
# multiply each element of the data array by n
def multiply(data, n)
  data.collect{|x| x*n }
end

puts multiply([1,2,3], 2)    # Prints 2,4,6
```

4. Si en vez de un proc usamos una lambda, el acceso a `n` permanece mas allá de `multiplier`. Es esta característica la que hace que la lambda `doubler` sea una clausura:

```
# Return a lambda that retains or "closes over" the argument n
def multiplier(n)
  lambda {|data| data.collect{|x| x*n } }
end

doubler = multiplier(2)      # Get a lambda that knows how to double
puts doubler.call([1,2,3])  # Prints 2,4,6
```

`doubler` encapsula o clausura con la lambda el binding de `n`

5. Return a pair of lambdas that share access to a local variable.

```

def accessor_pair(initialValue=nil)
  value = initialValue # A local variable shared by the returned lambdas.
  getter = lambda { value } # Return value of local variable.
  setter = lambda {|x| value = x } # Change value of local variable.
  return getter,setter # Return pair of lambdas to caller.
end

getX, setX = accessor_pair(0) # Create accessor lambdas for initial value 0.
puts getX[] # Prints 0. Note square brackets instead of call.
setX[10] # Change the value through one closure.
puts getX[] # Prints 10. The change is visible through the other.

```

6. Este código retorna un array de lambdas que son multiplicadores:

```

$ cat closure_shared_variables.rb
# Return an array of lambdas that multiply by the arguments
def multipliers(*args)
# x = nil
  args.map {|x| lambda {|y| x*y } }
end

double,triple = multipliers(2,3)
puts double.call(2)

$ rvm use 1.8.7
Using /Users/casiano/.rvm/gems/ruby-1.8.7-p352
$ ruby closure_shared_variables.rb
4

```

Veamos que ocurre cuando descomentamos la línea `# x = nil`:

```

$ cat closure_shared_variables.rb

# Return an array of lambdas that multiply by the arguments
def multipliers(*args)
  x = nil
  args.map {|x| lambda {|y| x*y } } # En 1.8 la |x| es la misma x
end # local

double,triple = multipliers(2,3)
puts double.call(2) # Prints 6 in Ruby 1.8

```

En Ruby 1.8 la `|x|` del argumento del bloque no es local al bloque sino que es la `x` de `multipliers` definida en `x = nil`. Obsérvese que la llamada `multipliers(2, 3)` retorna un array con dos lambdas. Obtenemos así un comportamiento inesperado:

```

$ rvm use 1.8.7
Using /Users/casiano/.rvm/gems/ruby-1.8.7-p352
$ rvm list

rvm rubies

=> ruby-1.8.7-p352 [ i686 ]

```

```

ruby-1.9.2-head [ x86_64 ]
ruby-1.9.2-p290 [ x86_64 ]
ruby-1.9.3-head [ x86_64 ]

$ ruby closure_shared_variables.rb
6

```

Esto es así porque la única `x` contiene el último valor asignado a ella: 3.

En este código el problema desaparece con versiones posteriores a la 1.9 porque los argumentos de un bloque son siempre locales:

```

$ rvm use 1.9.2
Using /Users/casiano/.rvm/gems/ruby-1.9.2-p290
$ ruby closure_shared_variables.rb
4

```

13.6.2. Clausuras y Bindings

La clase `Proc` define un método denominado `binding`. Cuando se llama este método en un `proc` o en una `lambda` retorna un objeto `Binding` que representa las ligaduras en efecto para esa clausura.

Los bindings conllevan mas que variables: guardan toda la información necesaria para poder ejecutar el método incluyendo el valor de self y - si existe - el bloque que será ejecutado con yield

```

# Return a lambda that retains or "closes over" the argument n
def multiplier(n)
  lambda {|data| data.collect{|x| x*n} }
end
doubler = multiplier(2)      # Get a lambda that knows how to double
puts doubler.call([1,2,3])  # Prints 2,4,6

```

El Módulo `Kernel` proporciona la función global `eval` que admite como segundo argumento un objeto `Binding` que provee el contexto para evaluar la cadena de código Ruby:

```

eval("n=3", doubler.binding) # Or doubler.binding.eval("n=3") in Ruby 1.9
puts doubler.call([1,2,3])  # Now this prints 3,6,9!

```

Como atajo, el método `eval` nos permite pasar directamente un objeto `Proc` en vez de pasar el `Binding` del objeto `Proc`:

```
eval("n=3", doubler)
```

El método `binding` de `Kernel` retorna un objeto `Binding` que representa los bindings en efecto en el punto de la llamada.

```
[~/Chapter6MethodsProcsLambdasAndClosures]$ cat binding.rb
def tutu(p)
  x = 4
  x = 2*p
  return [p, binding]
end
p, b = tutu(8)
p eval("p", b) # 8
p eval("x", b) # 16
```

Véase

1. An Introduction to Procs, Lambdas and Closures in Ruby (CooperPress) (YouTube)

13.7. Repaso

1. ¿Qué es Proc?

2. ¿Qué queda en z?

```
z = ->(x){ x*x }[4]
```

3. ¿Qué valor contiene @fn al final de la ejecución?

```
def tutu(p)
  @fn = 3; p[]
end
fn = -> {@fn = 88 }
tutu(fn)
fn[]
```

4. ¿Cuál es la salida?

```
class Array
  def each!(code)
    self.each_with_index do |n,i|
      self[i] = code[n]
    end
  end
  puts [1,2,3].each!(lambda { |n| n+1 })
```

5. ¿Cuál es la salida?

```
[15] pry(main)> { 2+3 }
```

6. ¿Cuál es la salida?

```
def tutu
  z = proc { return }
  z.call
  puts "In a_method"
end
tutu()
```

Véase 13.5.5

7. ¿Cuál es la salida?

```
def tutu
  z = lambda { return }
  z.call
  puts "In a_method"
end
tutu()
```

8. ¿Qué es un iterador?

9. ¿Qué valor contiene la variable v definida en la línea 1 al final del programa?

```
v = 34 # linea 1
def tutu
    yield 3.14159
end
tutu { |v| puts v }
```

10. ¿Qué es un bloque?

13.8. Objetos Method

- Los métodos Ruby son constructos ejecutables, pero no son objetos
- Procs y Lambdas son objetos que representan a los bloques
- Las instancias de la clase Method representan a los métodos
- La clase Object define un método con nombre `method` que retorna el objeto asociado con un método (también puede usarse `public_method`: ignora a los métodos protegidos y privados)

```
class Demo
    def initialize(n)
        @iv = n
    end
    def hello()
        "Hello, @iv = #{@iv}"
    end
end

k = Demo.new(99)
m = k.method(:hello)
m.call    #=> "Hello, @iv = 99"

l = Demo.new('Fred')
m = l.method("hello")
m.call    #=> "Hello, @iv = Fred"
```

- Cuando necesitamos un `Proc` en vez de un `Method` podemos convertir el `Method` a `Proc` vía el método `to_proc`

```
def square(x); x*x; end
puts (1..10).map(&method(:square))
```

- Definiendo Métodos con `Proc`, `Method` o bloques

El método `define_method` de `Module` espera un símbolo como argumento y crea un método con ese nombre utilizando el bloque asociado como cuerpo del método.

Es posible también pasar a `define_method` un `Proc` o un `Method` en vez del bloque como segundo argumento.

```
$ cat define_method.rb
class Chazam
  def initialize(data)
    @data = data
  end

  %w(user email food).each do |meth|
    define_method(meth) { @data[meth.to_sym] }
  end
end

p = Chazam.new(:user => 'Juan', :food => 'Cucumber')
puts p.user
puts p.food

$ ruby define_method.rb
Juan
Cucumber
```

- La clase `Method` define múltiples métodos

1. `name`

Returns the name of the method.

2. `owner`

Returns the class or module that defines the method.

3. `parameters`

Returns the parameter information of this method.

4. `receiver`

Returns the bound receiver of the method object.

5. `source_location`

Returns the Ruby source filename and line number containing this method or nil if this method was not defined in Ruby (i.e. native)

13.8.1. Objetos Method no Ligados (Unbound Method Objects)

Ruby soporta métodos *unbound*. Métodos que no están asociados con un objeto particular. Cuando se usa `instance_method` se obtiene un método *unbound*. Los métodos *unbound* sólo pueden ser llamados después que son ligados a algún objeto.

Los objetos de la clase `UnboundMethod` representan métodos pero sin el objeto `Binding` que define su entorno de ejecución.

```
ruby-1.9.2-head :001 > unbound_plus = Fixnum.instance_method("+")
=> #<UnboundMethod: Fixnum#+>
ruby-1.9.2-head :002 > plus_5 = unbound_plus.bind(5)
=> #<Method: Fixnum#+>
ruby-1.9.2-head :003 > plus_5[4]
=> 9
```

```
ruby-1.9.2-head :004 > plus_5.class
=> Method
```

También es posible construir objetos `UnboundMethod` usando el método `unbind`:

```
ruby-1.9.2-head :014 > x = 4.method('+')
=> #<Method: Fixnum#+>
ruby-1.9.2-head :015 > u = x.unbind
=> #<UnboundMethod: Fixnum#+>
ruby-1.9.2-head :016 > w = u.bind(3)
=> #<Method: Fixnum#+>
ruby-1.9.2-head :017 > w[5]
=> 8
```

```
~/rubytesting$ cat -n instance_method.rb
1 class Interpreter
2   def do_a() print "there, "; end
3   def do_d() print "Hello "; end
4   def do_e() print "!\n"; end
5   def do_v() print "Dave"; end
6   Dispatcher = {
7     ?a => instance_method(:do_a),
8     ?d => instance_method(:do_d),
9     ?e => instance_method(:do_e),
10    ?v => instance_method(:do_v)
11  }
12  def interpret(string)
13    string.each_byte{|b| Dispatcher[b].bind(self).call }
14  end
15 end
16
17 interpreter = Interpreter.new
18 interpreter.interpret('dave')
```

Al ejecutar, se obtiene:

```
~/rubytesting$ ruby instance_method.rb
Hello there, Dave!
```

El operador de interrogación nos dá el código de un carácter:

```
>> ?a
=> 97
>> 'a'[0]
=> 97
>> 97.chr
=> "a"
```

Veamos otro ejemplo:

```
MacBook-Air-de-casiano:rubytesting casiano$ ruby -r debug unbound_method.rb
unbound_method.rb:1:class Square
(rdb:1) l 1,100
[1, 100] in unbound_method.rb
=> 1 class Square
  2   def area
```

```

3      @side * @side
4    end
5    def initialize(side)
6      @side = side
7    end
8  end
9
10 area_un = Square.instance_method(:area)
11
12 s = Square.new(12)
13 area = area_un.bind(s)
14 area.call  #=> 144

(rdb:1) b 14
Set breakpoint 1 at unbound_method.rb:14
(rdb:1) c
Breakpoint 1, toplevel at unbound_method.rb:14
unbound_method.rb:14:area.call  #=> 144
(rdb:1) area.call
144

```

13.9. Programación Funcional

Ruby no es un lenguaje de programación funcional, pero la existencia de los bloques, procs y lambdas permite programar utilizando un estilo funcional. Cada vez que usamos bloques con `map` o `inject` estamos programando con estilo funcional:

```

# Compute the average and standard deviation of an array of numbers
mean = a.inject{|x,y| x+y } / a.size
sumOfSquares = a.map{|x| (x-mean)**2 }.inject{|x,y| x+y }
standardDeviation = Math.sqrt(sumOfSquares/(a.size-1))

```

13.9.1. Aplicando una Función a un Enumerable

En el ejemplo que sigue escribimos un módulo `Functional` que usamos para extender las clases `Proc` y `Method` con dos operaciones.

1. La operación `f <= a` toma una operación binaria implantada via una lambda, proc o method `f` como primer operando y un enumerable `a` como segundo. Retorna el resultado de operar todos los elementos de `a` con `f` (esto es hace una reducción):

```

1] pry(main)> require './functional'
=> true
[2] pry(main)> a = 1..5
=> 1..5
[3] pry(main)> sum = lambda { |x,y| x+y }
=> #<Proc:0x007f94ef3b8018@(pry):3 (lambda)>
[4] pry(main)> sum <= a
=> 15

```

2. La operación `f | a` aplica el Proc `f` a cada uno de los elementos del enumerable `a` retornando un enumerable:

```

14] pry(main)> a = 1..5
=> 1..5

```

```
[15] pry(main)> h = lambda { |x| 2*x }
=> #<Proc:0x007fd19ca482b8@(pry):14 (lambda)>
[16] pry(main)> h | a
=> [2, 4, 6, 8, 10]
```

Sigue un ejemplo de uso mas complicado:

```
if __FILE__ == $0
  a = (1..5).to_a
  sum = lambda { |x,y| x+y }          # A function to add two numbers
  mean = (sum<=a)/a.size.to_f        # Or sum.reduce(a) or a.inject(&sum)
  puts mean                          # 3.0
  deviation = lambda { |x| x-mean }   # Function to compute difference from mean
  print "#{deviation|a}\n"            # [-2, -1, 0, 1, 2]
  square = lambda { |x| x*x }         # Function to square a number
  puts (sum<=square|(deviation|a)) # 10 = 4 + 1 + 4 + 1
  standardDeviation = Math.sqrt((sum<=square|(deviation|a)).to_f/(a.size-1))
  puts standardDeviation           # sqrt(10/4) = 1.5811388300841898
end
```

Esta es la solución a los requisitos especificados:

```
[~/Chapter6MethodsProcsLambdasAndClosures]$ cat -n functional.rb | head -16
 1 module Functional
 2
 3   def apply(enum)
 4     enum.map &self
 5   end
 6   alias | apply
 7
 8   def reduce(enum)
 9     enum.inject &self
10  end
11  alias <= reduce
12 end
13
14 # Add these functional programming methods to Proc and Method classes.
15 class Proc; include Functional; end
16 class Method; include Functional; end
```

13.9.2. Composición de Funciones

En esta sección ampliamos el módulo `Functional` con el operador `*` para denotar la composición de procs, lambdas y methods. De esta forma, el anterior ejemplo puede reescribirse de este modo:

```
if __FILE__ == $0
  def polar(x,y)
    [Math.hypot(y,x), Math.atan2(y,x)]
  end
  def cartesian(magnitude, angle)
    [magnitude*Math.cos(angle), magnitude*Math.sin(angle)]
  end

  f = lambda { |x| x*x }
  g = lambda { |x| x+1 }
```

```

puts (f*g)[2]    # => 9
puts (g*f)[2]    # => 5

p,c = method(:polar), method(:cartesian)
puts (c*p)[3,4]  # => [3,4]

a = (1..5).to_a
sum = lambda {|x,y| x+y }          # A function to add two numbers
mean = (sum<=a)/a.size            # Or sum.reduce(a) or a.inject(&sum)
deviation = lambda {|x| x-mean }   # Function to compute difference from mean
square = lambda {|x| x*x }         # Function to square a number
puts sum<=square*deviation|a     # 10 = 4 + 1 + 4 + 1
standardDeviation = Math.sqrt((sum<=square*deviation|a)/(a.size-1).to_f)
puts standardDeviation           # sqrt(10/4) = 1.5811388300841898
end

~/TheRubyProgrammingLanguage/Chapter6MethodsProcsLambdasAndClosures]$ cat functional_compose.r
require './functional'

module Functional
  def compose(f)
    if self.respond_to?(:arity) && self.arity == 1
      lambda{|*args| self[f[*args]] }
    else
      lambda{|*args| self[*f[*args]] }
    end
  end

  alias * compose
end

```

13.9.3. Aplicación Parcial de Funciones

Se denomina *aplicación parcial* al proceso de tomar una función y un fijar un subconjunto de sus argumentos a ciertos valores produciendo una función que es equivalente a la anterior sobre el conjunto de argumentos libres.

```

ruby-1.9.2-head :013 > product = lambda { |x, y| x * y } # A function of two arguments
=> #<Proc:0x007fbf53991668@(irb):13 (lambda)>
ruby-1.9.2-head :014 > double = lambda { |y| product[2, y] } # Fix one argument, vary the other
=> #<Proc:0x007fbf530a9de8@(irb):14 (lambda)>
ruby-1.9.2-head :015 > double[4]
=> 8

```

En esta sección extendemos el módulo `Functional` con dos operaciones

- `lambda >> expresion` retorna una `lambda` en la que se ha fijado el primer argumento a `expression`
- `lambda << expresion` retorna una `lambda` en la que se ha fijado el último argumento a `expression`

Ejemplo de uso:

```

if __FILE__ == $0
  p = lambda {|x,y,z| x*y-z}
  d = p >> 2 >> 3          # x = 2 y = 3
  puts d[1]                  # 5 = 2*3-1

```

```

f = p << 4 << 5          # z = 4 y = 5
puts f[2]                   # 6 = 2*5-4
end

```

Implementación:

```
[~/TheRubyProgrammingLanguage/Chapter6MethodsProcsLambdasAndClosures]$ cat functional_partial.
require "./functional_compose"
module Functional
  def apply_head(*first)
    lambda {[*rest| self[*first.concat(rest)]]}
  end

  def apply_tail(*last)
    lambda {[*rest| self[*rest.concat(last)]]}
  end

  alias >> apply_head      # g = f >> 2 -- set first arg to 2
  alias << apply_tail      # g = f << 2 -- set last arg to 2
end
```

En Ruby 1.9 la clase `Proc` dispone del método `curry`. El método retorna un *curried proc*. Un curried proc si no recibe suficientes argumentos retorna un proc parcial sobre el resto de los argumentos:

```

ruby-1.9.2-head :006 > b = proc {|x, y, z| x+2*y-z }
=> #<Proc:0x007fac98842cd0@(irb):6>
ruby-1.9.2-head :007 > c = b.curry
=> #<Proc:0x007fac88828278>
ruby-1.9.2-head :009 > c[1,2,3]
=> 2
ruby-1.9.2-head :010 > c[1,2]
=> #<Proc:0x007fac88913688>
ruby-1.9.2-head :011 > c[1,2][3]
=> 2

```

Práctica: Aplicación Parcial

Extienda el módulo `Functional` de manera que el operador `**` admita como segundo argumento un objeto hash cuyas claves son las posiciones de los argumentos que se desean fijar y cuyos valores indican las expresiones a las que deben ser fijadas:

```

p = lambda {|x,y,z,w=0| x*y-z+w }
d = p ** {0 => 1, 2 => 3}  # x = 1 z = 3
puts d[4]                   # 1 = 1*4-3+0

```

13.9.4. Memoización

La *memoizacion* es una técnica que se usa para acelerar el cómputo de una función cuando:

1. La función retorna siempre el mismo valor cuando se le pasa la misma lista de argumentos (es *pura*, no depende de efectos laterales)
2. Tenemos razones para creer que la función será llamada con los mismos argumentos muchas veces
3. La función tarda bastante en computar el resultado

En esta sección aumentamos el módulo `Functional` con el operador unario `+lambda` que retorna una lambda equivalente a su argumento pero que cachea la relación entre argumentos y el valor retornado.

Veamos un ejemplo de uso. Las circustancias enumeradas anteriormente se dan en el cálculo de la función de *Fibonacci*.

```
require './functional_partial'
require 'benchmark'

...
if $0 == __FILE__
  fib = lambda { |n| n < 2 ? n : fib[n-1] + fib[n-2] }
  fibm = +fib

  n = 35
  Benchmark.benchmark(CAPTION, 7, FORMAT) do |x|
    tf = x.report("computed:") { fib[n] }
    tt = x.report("memoized:") { fibm[n] }
  end
end
```

La ejecución compara las dos versiones, con y sin memoizar:

```
[~/TheRubyProgrammingLanguage/Chapter6MethodsProcsLambdasAndClosures]$ ruby functional_memoize
      user      system      total      real
computed:  6.500000  0.010000  6.510000 ( 6.520144)
memoized:  6.140000  0.000000  6.140000 ( 6.153518)
```

Este es el código de `memoize`:

```
[~/TheRubyProgrammingLanguage/Chapter6MethodsProcsLambdasAndClosures]$ cat functional_memoize.
require './functional_partial'
require 'benchmark'
include Benchmark

module Functional
  #
  # Return a new lambda that caches the results of this function and
  # only calls the function when new arguments are supplied.
  #
  def memoize
    cache = {} # An empty cache. The lambda captures this in its closure.
    lambda {[*args]
      # notice that the hash key is the entire array of arguments!
      cache[args] = self[*args] unless cache.has_key?(args)
      cache[args]
    }
  end

  alias +@ memoize      # cached_f = +f
end
```

El método `benchmark`

1. `benchmark(caption = "", label_width = nil, format = nil, *labels)`

2. Invokes the block with a `Benchmark::Report` object, which may be used to collect and report on the results of individual benchmark tests.
3. Reserves `label_width` leading spaces for labels on each line.
4. Prints `caption` at the top of the report, and uses `format` to format each line.

13.9.5. Símbolos, Métodos y Procs

`to_proc`

1. El método `method` recibe un `Symbol` que sea el nombre de un método y devuelve un objeto `Method`
2. Ruby 1.9 añade el método `to_proc` a la clase `Symbol`
3. Este método `to_proc` permite que un símbolo prefijado con `&` pueda pasarse como bloque a un iterador

```
# Increment an array of integers with the Fixnum.succ method
[1,2,3].map(&:succ) # => [2,3,4]
```

que es equivalente a:

```
[1,2,3].map { |n| n.succ }
```

El método `to_proc` podría implementarse así:

```
class Symbol
  def to_proc
    lambda {|receiver, *args| receiver.send(self, *args)}
  end
end
```

- El método `send` es un método de la clase `Object`:

```
ruby-1.9.2-head :008 > Object.instance_methods.select { |x| x =~ /send/ }
=> [:send, :public_send, :__send__]
```

- `__send__` es un sinónimo de `send`.
- `public_send` sólo invoca métodos públicos, no privados ni protegidos.

También podríamos implementar `to_proc` usando el método `method` de la clase `Object`:

```
class Symbol
  def to_proc
    lambda {|receiver, *args| receiver.method(self)[*args]}
  end
end
```

Accediendo a una Clase como una Colección de Métodos

En el siguiente ejemplo, definiremos `[]` en la clase `Module` para que podamos escribir:

```
String[:reverse].bind("hello").call # olleh
```

Para ello basta con hacer un alias `[] instance_method`

```
[~/srcLPP/rubytesting/TheRubyProgrammingLanguage/Chapter6MethodsProcsLambdasAndClosures]$ cat
1 class Module
2   alias [] instance_method
3 end
4
5 puts String[:reverse].bind("hello").call # olleh
```

El método retornado por `instance_method` es un `UnboundMethod`. Puesto que el método retornado no tiene `Binding`, es necesario llamar al método `bind` sobre un objeto para ligarlo.

Es posible llevar esta analogía, usando el `UnboundMethod` a su vez como hash usando este alias:

```
7 class UnboundMethod
8   alias [] bind
9 end
10
11 puts String[:reverse]["hello"][] # olleh
```

Resulta tentador definir también el operador *asignación a elementos de una colección* `[]=` para definir métodos:

```
[~/srcLPP/rubytesting/TheRubyProgrammingLanguage/Chapter6MethodsProcsLambdasAndClosures]$ cat
1 class Module
2
3   def []=(symbol, code)
4     define_method(symbol, code)
5   end
6
7 end
8
9 String[:chuchu] = lambda { |x| puts "Chuchu#{x}!"}
10
11 "hello".chuchu("Cham") # ChuchuCham!
```

El método `define_method` es un método privado de `Module`:

```
ruby-1.9.2-head :009 > Module.private_methods.select { |x| x =~ /define/ }
=> [:method_undefined, :define_method, :singleton_method_undefined]
```

En la línea 4 `self` es el objeto módulo sobre el que estamos definiendo el método (por ejemplo `String` en la llamada de la línea 9: recuerde que toda clase es un módulo).

Con esta definición de `[]=` podemos escribir:

```
Enumerable[:average] = lambda do
  sum, n = 0.0, 0
  self.each { |x| sum += x; n += 1 }
  if n == 0
    nil
  else
    sum/n
  end
end
```

Viendo los (Nombres de) Métodos como Colecciones de Objetos

Visto lo anterior surge la idea de hacer algo similar para los métodos singleton de los objetos. La primera idea sería definir `[]` y `[]=` en la clase `Object`. El problema es que muchísimas clases redefinen `[]` y `[]=` y, por tanto, nuestra definición se perdería.

En vez de eso podemos hacerlo al revés: ver a los nombres de los métodos como colecciones y a los objetos como índices. No queda tan natural, pero tiene la ventaja de que la clase `Symbol` no es tan heredada como la clase `Object`.

Para hacerlo abrimos la clase `Symbol` y definimos ahí los operadores:

```
[~/Chapter6MethodsProcsLambdasAndClosures]$ cat -n singleton_index.rb
1 class Symbol
2   def [](obj)
3     obj.method(self)
4   end
5
6   def []=(obj,f)
7     sym = self
8     eigenclass = class << obj; self end
9     eigenclass.instance_eval do
10       define_method(sym, f)
11     end
12   end
13 end
14
15 puts :length["Jane"] [] # 4
16
17 x = "hello"
18 :tutu[x] = lambda { |z| puts "#{x} #{z}"}
19 x.tutu("Jane") # hello Jane
end
```

Repase la sección 14.12 para recordar que para abrir la *eigenclass* del objeto `obj` usamos la sintaxis `class << obj`. Así, una forma de obtener la *eigenclass* es:

```
eigenclass = class << o; self; end
```

13.9.6. Véase También

1. Pat Shaughnessy - Functional Programming In Ruby YouTube. Trasparencias en <https://speakerdeck.com/pa-programming-and-ruby>
2. An Introduction to Procs, Lambdas and Closures in Ruby (CooperPress)
3. Ruby 2.0 - Getting Started and Named Parameters
4. Programming with Nothing por Tom Stuart (Rubymanor)
5. Impossible Programs.^a talk by Tom Stuart, author of “Understanding Computation” (O'Reilly)
6. Ruby Conf 12 - Y Not- Adventures in Functional Programming by Jim Weirich YouTube

13.10. Práctica: La Calculadora

Se trata de extender la calculadora en postfijo realizada en la práctica 8.12.

La nueva calculadora deberá estar implantada mediante una clase `PostfixCalc` y dispone de dos operadores `=` y `!` que permiten respectivamente asignar valores a variables y obtener el valor ligado a una variable.

Los objetos `PostfixCalc` disponen de un atributo `stack` (de sólo lectura), de un atributo `st` que contiene la tabla de símbolos.

El objeto dispone también de un hash privado que mantiene las lambdas (véase la sección 13.11) que implantan las operaciones de suma, resta, multiplicación, división, etc.

```

@f = {
  '+' => lambda { |a,b| a+b},
  '*' => lambda { |a,b| a*b},
  ...
}

```

El siguiente ejemplo muestra un código que usa la clase:

```

43 if __FILE__ == $0
44   include Math
45
46   expr = ARGV.join(" ")
47   c = PostfixCalc.new({'sin' => lambda { |a| sin(a)}})
48
49   c.eval(expr)
50
51   puts "----stack-----"
52   c.stack.each { |m| puts m }
53   puts "----symbol table-----"
54   c.st.each { |k,v| puts "#{k} => #{v}" }
55 end

```

Siguen varias ejecuciones:

```

~/rubytesting$ ruby postfixwithhashfunctions3.rb '3.14159 2 / sin'
----stack-----
0.9999999999912

```

Por defecto la calculadora contiene un conjunto básico de operaciones. El programa cliente puede añadir nuevas operaciones como ocurre en el ejemplo:

```
c = PostfixCalc.new({'sin' => lambda { |a| sin(a)}})
```

El siguiente ejemplo muestra el uso del operador de asignación = y del operador de carga (fetch) !:

```

----symbol table-----
~/rubytesting$ ruby postfixwithhashfunctions3.rb '2 a = 3 b = a ! b ! *'
----stack-----
6.0
----symbol table-----
a => 2.0
b => 3.0

```

La expresión `2 a =` retira el 2 y la `a` de la pila y almacena en la entrada `a` de la tabla de símbolos de la calculadora un 2. No añade nada al top de la pila.

La expresión `a !` empuja en el top de la pila el valor asociado con la entrada `a` de la tabla de símbolos.

La función `p` imprime el elemento en el top de la pila:

```

~/rubytesting$ ruby postfixwithhashfunctions3.rb '2 a = 3 a ! 4 + p'
6.0
----stack-----
3.0
6.0
----symbol table-----
a => 2.0

```

- Consulte el método `merge!` de la clase Hash
- Generalize el programa para que los operadores `=` y `!` puedan ir pegados al identificador:

```
~/rubytesting$ ruby postfixwithhashfunctions3.rb '2 a= 3 a! 4 + p'
6.0
----stack-----
3.0
6.0
----symbol table-----
a => 2.0
```

Hay varias soluciones. Una es usar `scan`:

```
>> "12 -3+ 4- 5*".scan(/-?\d+|[-+*/]/)
=> ["12", "-3", "+", "4", "-", "5", "*"]
>>
```

Otra es usar `gsub`. Consulte el método `gsub` de la clase `String`, que permite sustituciones globales:

```
expr.gsub!(/([a-zA-Z_]\w)*([!=])/,'\\1 \\2')
```

13.11. Ejercicios

1. ¿Que resultados dan las siguientes operaciones?

```
>> x = { :a => 1, :b => 2 }
=> {:b=>2, :a=>1}
>> x.keys
=>
>> x.values
=>
>> x[:c] = 3
=> 3
>> x
=>
>> x.delete('a')
=> nil
>> x
=>
>> x.delete(:a)
=> 1
>> x
=>
>> x = { :a => 1, :b => 2, :c => 4 }
=> {:b=>2, :c=>4, :a=>1}
>> x.delete_if { |k,v| v % 2 == 0 }
=>
>> x
=>
>> z = { :a => [1, { :b => { :c => 2 }}], :e => lambda { |x| x*x } }
=> {:e=>#<Proc:0x00000001012c5108@(irb):18>, :a=>[1, { :b=>{:c=>2}}]}
>> z[:a][1][:b][:c]
```

```

=>
>> z[:e]
=> #<Proc:0x00000001012c5108@(irb):18>
>> z[:e].class
=> Proc
>> Proc.instance_methods(false)
=> ["dup", "[]", "==", "to_s", "binding", "clone", "arity", "to_proc", "call"]
>> z[:e].call(4)
=>
>> z[:e][9]
=>
>> z[:e][4]
=>

```

2. ¿Que resultado da la siguiente expresión?

```

>> f = lambda { |x,y| x * y }
=> #<Proc:0x000000010128a4b8@(irb):17>
>> f.arity
=>

```

3. ¿Que retorna la función mult?

```

~/rubytesting$ cat -n closure.rb
1 def mult(n)
2   lambda { |v| v.map { |x| x * n } }
3 end
4
5 doubler = mult(2)
6 puts doubler[[1,2,3]]

```

¿Cuál es la salida?

4. ¿Cual es la salida de este programa?

```

~/rubytesting$ cat -n sharedvarandclosure.rb
1 def accessor_pair(initialValue = nil)
2   value = initialValue
3
4   getter = lambda { value }
5   setter = lambda { |x| value = x }
6   return getter, setter
7 end
8
9
10 getX, setX = accessor_pair(9)
11 puts getX[]
12
13 setX[-1]
14 puts getX[]

```

¿Que retorna `accessor_pair`? Cómo es que la variable `value` no es destruída por el garbage collector de Ruby cuando se termina el ámbito de `accessor_pair`?

5. ¿Qué resultados se dan?

```
>> a = (1..5).to_a  
=>  
>> a = a.collect { |x| x*x }  
=>  
>> b = a.select { |y| (4..16).include? y }  
=>
```

6. ¿Que queda en c?

```
>> a = (1..5).to_a  
=>  
>> c = a.inject(1) { |x,y| x *= y }  
=>
```

7. ¿Cual es el resultado?

```
>> "%d %s" % [3, "rubies"]  
=>
```

8. ¿Cuales son los resultados?

```
>> x, y = 4, 5  
=>  
>> z = x > y ? x : y  
=>  
>> x,y,z = [1,2,3]  
=>
```

9. ¿Cuales son los resultados?

```
>> z = %x{ls}.split(/\s+/).select { |w| w =~ /ruby/i }  
=>  
>> /ruby/i === "Ruby"  
=>
```

10. Escriba una expresión regular que describa los números en punto flotante (1.5, -2.3, -4.7e-1, etc.)

11. Explique el resultado:

```
>> str = "Hello 'name'"  
=> "Hello 'name'"  
>> name = "John"  
=> "John"  
>> str.gsub(/'([^']*')/) { eval($1) }  
=> "Hello John"
```

12. Explique los resultados:

```
>> "hello".index /e/  
=> 1  
>> "hello".index /h/  
=> 0  
>> "hello".index /n/  
=> nil  
>> "hello".index /o/  
=> 4
```

13. ¿Que hace `attr_accessor`? ¿ Que argumentos lleva?
14. ¿Que hace `attr_reader`? ¿ Que argumentos lleva?
15. ¿Que hace `attr_writer`? ¿ Que argumentos lleva?
16. ¿Cómo hago que los objetos de una clase puedan sumarse? ¿Como hago para que puedan ser indexados como si fueran arrays?
17. ¿Cómo se llama la función que hay que definir para el menos unario (opuesto)?
18. ¿Que hace `yield 4, 5`?
19. ¿Cómo se define una constante en Ruby?
20. ¿Como se denota una variable que guarda un atributo de clase en Ruby?
21. ¿Puede un método privado de una clase ser llamado desde una subclase?
22. ¿Puede un método privado `p` de una clase ser llamado `x.p()` desde un objeto `x` de una subclase?
23. ¿Puede un método protected `p` de una clase ser llamado `x.p()` desde un objeto `x` de una subclase?

13.12. Práctica: Un Motor para las Expresiones Regulares en Pocas Líneas

Objetivo

El objetivo es crear nuestro propio motor de expresiones regulares ([4]). Para ello, una expresión regular como `c(a|d)+r` debe ser expresada mediante una lambda. Una regexp como `c(a|d)+r` es el resultado de la composición de lambdas. Por ejemplo, la regexp `c(a|d)+r` se denotará por:

```
seq(char('c'), seq(plus(alt(char('a'), char('d'))), char('r'))))
```

Como se hace

La idea es que `char`, `seq`, `plus`, etc. son métodos que retornan funciones (lambdas). La lambda retornada por `char('d')` reconoce el lenguaje '`d`', la lambda retornada por `star(char('c'))` reconoce el lenguaje `c*` y la lambda retornada por `seq(char('c'), seq(plus(alt(char('a'), char('d'))), char('r'))))` reconoce el lenguaje `c(a|d)+r`.

Mas en concreto, *la lambda que representa la expresión regular r*

1. recibe una cadena `x` y
2. devuelve `false` si no hay un prefijo de `x` que case con `r`
3. y devuelve el resto no casado de la cadena `x` si hubo matching

Por ejemplo, el método `char` recibe una cadena `c` y retorna una lambda que recibe una cadena `x` y

1. devuelve `false` si `c` no es un prefijo de `x`.
2. En caso contrario retorna el resto de la cadena

Esta implementación de `char` usa circuito corto:

```
def char(c)
  lambda { |x| x[0,c.length] == c and x[c.length..-1] }
end
```

Ejemplos de uso del Módulo

```
ruby-1.9.2-p290 :001 > load "regexpcalc1213.rb"
=> true
ruby-1.9.2-p290 :002 > include ULL::ETSII::AluXXX::LambdaRegexp
=> Object
ruby-1.9.2-p290 :003 > char('c')
=> #<Proc:0x007f998b027408@/Users/casiano/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLang
ruby-1.9.2-p290 :004 > char('c')['c'] # 'c' casa con /c/ el resto es vacío ""
=> ""
ruby-1.9.2-p290 :005 > char('c')['d'] # 'd' no casa con /c/. Se retorna false
=> false
ruby-1.9.2-p290 :006 > char('c')['cd'] # El prefijo 'c' casa con 'cd'. Resto = 'd'
=> "d"
ruby-1.9.2-p290 :007 > star(char('c'))['cccef'] # El prefijo 'ccc' casa con /c*/. Resto = 'ef'
=> "ef"
ruby-1.9.2-p290 :008 > star(alt(char('c'), char('e')))['cccef'] # El prefijo 'e' casa con /(c|
=> "f"
ruby-1.9.2-p290 :009 >
```

La siguiente secuencia de llamadas:

```
e = seq(char('c'), char('d'))
s = 'cde'
remaining = e[s]      # 'cde' Matched. Remaining = 'e'
puts "'#{s}' Matched. Remaining = '#{remaining}'" if remaining

e = seq(char('c'), alt(star(char('d')), char('r')))

s = 'cdddf'          #
remaining = e[s]      # 'cdddf' Matched. Remaining = 'f'
puts "'#{s}' Matched. Remaining = '#{remaining}'" if remaining

s = 'crddf'          # 'crddf' Matched. Remaining = 'rddf'
remaining = e[s]
puts "'#{s}' Matched. Remaining = '#{remaining}'" if remaining
```

Debería producir una salida parecida a esta:

```
MacBookdeCasiano:rubytesting casiano$ ruby regexpcalc.rb
'cde' Matched. Remaining = 'e'
'cdddf' Matched. Remaining = 'f'
'crddf' Matched. Remaining = 'rddf'
```

Definiendo Operadores

Es posible usar overriding de operadores para mejorar la expresividad de la notación. He aquí un ejemplo que utiliza una versión en la que se han definido los operadores:

- - es `seq`
- | es `alt`
- + unario es `plus`
- ~ es `star`
- El método `re` es una versión de `char`

```

e = 'cd'.re
s = 'cde'
remaining = e[s] # 'cde' Matched. Remaining = 'e'
puts "/cd/ '#{s}' Matched. Remaining = '#{remaining}'" if remaining

e = 'd'.re | 'r'.re
s = 'rdddf'
remaining = e[s] # 'rdddf' Matched. Remaining = 'ddf'
puts "/d|r/ '#{s}' Matched. Remaining = '#{remaining}'" if remaining

e = 'c'.re - ('d'.re | 'r'.re)
s = 'crddf'
remaining = e[s] # 'crddf' Matched. Remaining = 'ddf'
puts "/c(d|r)/ '#{s}' Matched. Remaining = '#{remaining}'" if remaining

e = 'c'.re - +('d'.re | 'r'.re)
s = 'crddf'
remaining = e[s]
puts "/c(d|r)+/ '#{s}' Matched. Remaining = '#{remaining}'" if remaining

e = 'c'.re - ~( 'd'.re | 'r'.re)
s = 'cdrdrf'
remaining = e[s]
puts "/c(d|r)*/ '#{s}' Matched. Remaining = '#{remaining}'" if remaining

e = 'c'.re - ~( 'd'.re | 'r'.re)
s = 'cff'
remaining = e[s]
puts "/c(d|r)*/ '#{s}' Matched. Remaining = '#{remaining}'" if remaining

e = 'c'.re - ~( 'd'.re | 'r'.re)
s = 'cdrd'
remaining = e[s]
puts "/c(d|r)*/ '#{s}' Matched. Remaining = '#{remaining}'" if remaining

e = 'c'.re - +('d'.re | 'r'.re)
s = 'cff'
remaining = e[s] # 'cff' didn't match. Remaining = 'false'
puts "/c(d|r)+/ '#{s}' didn't match. Remaining = '#{remaining}'" unless remaining

```

Cuando se ejecuta, este programa produce:

```

~/rubytesting$ ruby regexpcalc2.rb
/cd/ 'cde' Matched. Remaining = 'e'
/d|r/ 'rdddf' Matched. Remaining = 'ddf'
/c(d|r)/ 'crddf' Matched. Remaining = 'ddf'
/c(d|r)+/ 'crddf' Matched. Remaining = 'f'
/c(d|r)*/ 'cdrdrf' Matched. Remaining = 'f'
/c(d|r)*/ 'cff' Matched. Remaining = 'fff'
/c(d|r)*/ 'cdrd' Matched. Remaining = ''
/c(d|r)+/ 'cff' didn't match. Remaining = 'false'

```

Tareas Escriba un módulo que implementa este motor de expresiones regulares. Use `module_function` para que el espacio de nombres sea incluible (véase la sección 14.7.4 de los apuntes):

```

module ULL
  module ETSII
    module AluXXX
      module LambdaRegexp

        module_function

        def epsilon
          lambda {|x| x }
        end

        .....

      end # Lambda
    end # AluXXX
  end # ETSII
end # ULL

```

1. Use TDD con RSpec
2. Use Unit Testing
3. Use Continuous Integration (Travis)
4. Use Continuous Testing (Guard)
5. Documente su gema (véase *RDOC::Markup* o *RDOC* o *YARD*).
6. Véa un ejemplo ilustrativo de como debería quedar la documentación del módulo creado en module ULL::ETSII::AluXXX::LambdaRegexp
7. Cree una gema `ull-etsii-aluXX-lambdaregexp`
8. Publique la gema en RubyGems
9. Indique la URL de su repositorio en GitHub

Enlaces Relacionados

- Casiano lambda-regexp project (bitbucket)

Capítulo 14

Clases y Módulos

Véase el capítulo Classes, Objects, and Variables del libro *Programming Ruby. The Pragmatic Programmer's Guide*

14.1. Definición de una Clase Simple

14.1.1. Creando una Clase

```
class Point  
end
```

14.1.2. Instanciando un Punto

```
p = Point.new  
  
p.class      # => Point  
p.is_a? Point # => true
```

14.1.3. Inicializando un Punto

El método `initialize` es privado:

```
$ cat -n calling_initialize.rb  
1  class Tutu  
2    def initialize(a,b)  
3      @a, @b = a,b  
4    end  
5  
6    def titi  
7      initialize(2,3)  
8    end  
9  end  
10 if __FILE__ == $0  
11   x = Tutu.new(4,5)  
12   z = x.titi()  
13   puts z.class  
14   puts z  
15   x.initialize(4,2)  
16 end  
  
casiano@exthost:~/LPPsrc/151012$ ruby calling_initialize.rb  
Array  
2
```

```
3  
calling_initialize.rb:15:in '<main>': private method 'initialize' called for #<Tutu:0x8bb576c  
casiano@exthost:~/LPPsrc/151012$
```

The instance variables of an object can only be accessed by the instance methods of that object. Code that is not inside an instance method cannot read or set the value of an instance variable

Aviso a los programadores Java:

```
casiano@exthost:~/LPPsrc/151012$ cat -n instance_variables_of_a_class.rb  
1  class Tutu  
2    @x = 4  
3    @y = 9  
4  
5    def initialize(x,y)  
6      @x, @y = x, y  
7    end  
8    def self.x  
9      @x  
10   end  
11  
12   def self.y  
13     @y  
14   end  
15  
16   def x  
17     @x  
18   end  
19  
20   def y  
21     @y  
22   end  
23 end  
24  
25 if __FILE__ == $0  
26   puts Tutu.x      # 4  
27   puts Tutu.y      # 9  
28   z = Tutu.new(5,7)  
29   puts z.x         # 5  
30   puts z.y         # 7  
31 end
```

```
casiano@exthost:~/LPPsrc/151012$ ruby instance_variables_of_a_class.rb  
4  
9  
5  
7
```

1. This code does not do at all what a Java programmer expects. Instance variables are always resolved in the context of **self**.
2. When the **initialize** method is invoked, **self** holds an instance of the **Tutu** class.
3. But the code outside of that method is executed as part of the definition of the **Tutu** class.

4. When those first two assignments are executed, `self` refers to the `Tutu` class itself, not to an instance of the class.
5. The `@x` and `@y` variables inside the `initialize` method are completely different from those outside it.

14.1.4. Definiendo un método `to_s`

Just about any class you define should have a `to_s` instance method to return a string representation of the object.

```
class Point
  def initialize(x,y)
    @x, @y = x, y
  end

  def to_s      # Return a String that represents this point
    "#{@x},#{@y}" # Just interpolate the instance variables into a string
  end
end
```

14.1.5. Acceso a los Atributos

Our `Point` class uses two instance variables. As we've noted, however, the value of these variables are only accessible to other instance methods. If we want users of the `Point` class to be able to use the X and Y coordinates of a point, we've got to provide accessor methods that return the value of the variables:

```
class Point
  def initialize(x,y)
    @x, @y = x, y
  end

  def x          # The accessor (or getter) method for @x
    @x
  end

  def y          # The accessor method for @y
    @y
  end
end

p = Point.new(1,2)
q = Point.new(p.x*2, p.y*3)

class MutablePoint
  def initialize(x,y); @x, @y = x, y; end

  def x; @x; end      # The getter method for @x
  def y; @y; end      # The getter method for @y

  def x=(value)       # The setter method for @x
    @x = value
  end

  def y=(value)       # The setter method for @y
```

```

    @y = value
  end
end

p = Point.new(1,1)
p.x = 0
p.y = 0

```

This combination of instance variable with trivial getter and setter methods is so common that Ruby provides a way to automate it. The `attr_reader` and `attr_accessor` methods are defined by the `Module` class. All classes are modules, (the `Class` class is a subclass of `Module`) so you can invoke these method inside any class definition. Both methods take any number of symbols naming attributes. `attr_reader` creates trivial getter methods for the instance variables with the same name. `attr_accessor` creates getter and setter methods. Thus, if we were defining a mutable `Point` class, we could write:

```

class Point
  attr_accessor :x, :y # Define accessor methods for our instance variables
end

class Point
  attr_reader :x, :y # Define reader methods for our instance variables
end

attr_reader "x", "y"

attr :x          # Define a trivial getter method x for @x
attr :y, true    # Define getter and setter methods for @y. Optional boolean is Obsolete

```

The `attr`, `attr_reader`, and `attr_accessor` methods create instance methods for us. This is an example of metaprogramming.

Utilización de los Setters Dentro de una Clase Una vez que se ha definido un setter como `x=` podemos sentirnos tentados de utilizarlo dentro de otro método de instancia de la clase.

That is, instead of writing `@x=2`, you might write `x=2`, intending to invoke `x=(2)` implicitly on `self`.

It doesn't work; `x=2` creates a new local variable.

2. The rule is that assignment expressions *will only invoke a setter method when invoked through an object*
3. If you want to use a setter from within the class that defines it, invoke it explicitly through `self`.
4. For example: `self.x=2`.

```

casiano@exthost:~/LPPsrc/221012$ cat -n setters.rb
 1  class Tutu
 2    attr_accessor :n
 3
 4    def initialize(n)
 5      @n = n
 6    end
 7
 8    def to_s
 9      "#{n*2}"

```

```

10    end
11
12    def square
13      n = n * n # n here is a local variable!
14    end
15
16    def square2
17      self.n = n * n
18    end
19  end
20
21 if __FILE__ == $0
22   x = Tutu.new(ARGV.shift || 4)
23   puts x                      # to_s gets called: 8
24   begin
25     x.square
26   rescue
27     puts "Raised exception '#{$!.class}'. Message: <#{$!}>"
28   end
29   x.square2                  # 32 = 16 *2
30   puts x
31 end

```

```

~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby setters.rb
8
Raised exception 'NoMethodError'. Message: <undefined method '*' for nil:NilClass>
32

```

Los Accessors como attr_accessor se Invocan Fuera de las Definiciones de Métodos attr_reader et al. son proveídos por la clase Module.

Todas las clases son módulos

```

ruby-1.9.2-head :004 > Class.class
=> Class
ruby-1.9.2-head :005 > Class.superclass
=> Module
ruby-1.9.2-head :006 > Class.superclass.superclass
=> Object
ruby-1.9.2-head :007 > Class.superclass.superclass.superclass
=> BasicObject
ruby-1.9.2-head :008 > Class.ancestors
=> [Class, Module, Object, InteractiveEditor::Editors,
  Wirble::Shortcuts,
  PP::ObjectMixin,
  Kernel, BasicObject]

```

Ejercicio 14.1.1. ¿attr_accessor es un método de instancia o de clase?

14.1.6. Definiendo Operadores (La Clase Point)

1. We'd like the + operator to perform vector addition of two Point objects,
2. the * operator to multiply a Point by a scalar,

3. and the unary `{` operator to do the equivalent of multiplying by `-1`.

Method-based operators such as `+` are simply methods with punctuation for names.

Because there are unary and binary forms of the `{` operator, Ruby uses the method name `{@` for unary minus.

Here is a version of the Point class with mathematical operators defined:

```
[~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/point]$ cat -n point.rb
 1 class Point
 2   attr_reader :x, :y    # Define accessor methods for our instance variables
 3
 4   include Comparable
 5   include Enumerable
 6
 7   def initialize(x,y)
 8     @x,@y = x, y
 9   end
10
11  def to_s      # Return a String that represents this point
12    "#{@x},#{@y}"  # Just interpolate the instance variables into a string
13  end
14
15  def chuchu
16    "Point: chuchu"
17  end
18  private :chuchu
19
20  def +(other)      # Define + to do vector addition
21    Point.new(@x + other.x, @y + other.y)
22  end
23
24  def -@          # Define unary minus to negate both coordinates
25    Point.new(-@x, -@y)
26  end
27
28  def *(scalar)    # Define * to perform scalar multiplication
29    return Point.new(@x*scalar,   @y*scalar) if scalar.is_a? Numeric
30    return Point.new(@x*scalar.x, @y*scalar.y) if scalar.is_a? Point
31  end
32
33  def coerce(other)
34    [self, other]
35  end
36
37  def [](index)
38    case index
39    when 0, -2 then @x      # Index 0 (or -2) is the X coordinate
40    when 1, -1 then @y      # Index 1 (or -1) is the Y coordinate
41
42    when :x, "x" then @x    # Hash keys as symbol or string for X
43    when :y, "y" then @y    # Hash keys as symbol or string for Y
44    else nil                # Arrays and hashes just return nil on bad indexes
45  end
46end
```

```

47
48  def hash
49    code = 17
50    code = 37*code + @x.hash
51    code = 37*code + @y.hash
52    code
53  end
54
55  def eql?(o)
56    return @x.eql?(o.x) && @y.eql?(o.y) if o.instance_of? Point
57    false
58  end
59
60  def ==(o)
61    return @x == o.x && @y == o.y if o.instance_of? Point
62    false
63  end
64
65  def <=>(o)
66    return nil unless o.instance_of? Point
67    @x*@x+@y*@y <=> o.x*o.x+o.y*o.y
68  end
69
70 end
71
72 if __FILE__ == $0
73   p = Point.new(1,2)
74   q = Point.new(1,1)
75   puts p
76   puts -p
77   puts p+q
78   puts p*q
79   puts p*p
80   puts 2*p
81
82   puts "#{p[0]}, #{p[1]}"
83   puts "#{p[-2]}, #{p[-1]}"
84   puts "#{p[:x]}, #{p[:y]}"
85   puts "#{p['x']}, #{p['y']}"
86
87   m = { p => 1, q => 2 }
88   puts m[p]
89   puts m[q]
90
91
92 # It is not a total order relation
93 r = Point.new(2, 1)
94 puts p > r
95 puts r > p
96 puts r == p
97 puts q < p
98 puts "[#{p}, #{q}, #{r}].sort"
99 puts [p, q, r].min

```

```

100   puts [p, q, r, Point.new(4,3)].max
101
102
103 end

```

Enlaces Relacionados

1. Comparable
2. Enumerable
3. Variables de Instancia
4. Métodos de Acceso
5. Variables de Clase
6. Métodos de Clase
7. Visibilidad de los métodos
8. Los métodos *private*, *public* y *protected* son métodos de instancia privados de
9. Variables de Instancia de Clase Module

14.1.7. Tabla de Operadores en Ruby

- Tabla de Operadores en Ruby

The following table provides a reference when you need to know the operator precedence used by Ruby. The table lists all operators from highest precedence to lowest.

14.1.8. Polimorfismo, Comprobación de Tipos y Tipado Pato (Duck Typing)

Repase la sección *Clase y Tipo de un Objeto. Polimorfismo* 10.8.4.

Red or yellow, black or white, they are all ducks in the VM's sight.

En programación orientada a objetos, el *polimorfismo* se refiere a la propiedad por la que es posible enviar mensajes (llamar a métodos) con el mismo nombre a objetos de clases distintas.

El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

```
[~/srcLPPRuby/TheRubyProgrammingLanguage/Chapter3DatatypesAndObjects]$ cat type_versus_class_c
# page 75
require "stringio"
class Tutu
  # Appends " chazam!" to its argument
  def chazam_check(x) # duck typing style
    raise "Error in chazam_check" unless x.respond_to? :"<<"
    x << " chazam!"
  end
end

if __FILE__ == $0
  t = Tutu.new
```

```

t = Tutu.new
puts "procesing a String"
p t.chazam_check("ban ban")  # "ban ban chazam!\""
puts "procesing an Array"
p t.chazam_check([1,2,3])    # [1, 2, 3, " chazam!]"
begin
  puts "Processing a Fixnum"
  p t.chazam_check(6)          # can't convert String into Integer
rescue
  puts "t.chazam_check(6) fails:\n\tException type: <#{$!.class}>\n\tmessage: <#{$!}>" 
end
end

[~/srcLPPRuby/TheRubyProgrammingLanguage/Chapter3DatatypesAndObjects]$ ruby type_versus_class_
procesing a String
"ban ban chazam!"
procesing an Array
[1, 2, 3, " chazam!"]
Processing a Fixnum
t.chazam_check(6) fails:
  Exception type: <TypeError>
  message: <can't convert String into Integer>

[~/srcLPPRuby/TheRubyProgrammingLanguage/Chapter3DatatypesAndObjects]$ cat type_versus_class_c
# page 75
require "stringio"
class Tutu
  # Print the lengths of the lines in the IO object
  def show_lengths(x)
    raise "Error in show_lengths" unless x.is_a? IO
    #raise "Error in show_lengths" unless x.respond_to? "each_line"
    x.each_line do |line|
      print "#{line.length} "
    end
    puts ""
  end
end

if __FILE__ == $0
  t = Tutu.new

#####
  t.show_lengths(IO.open(File.open($0).to_i)) # 10 19 ...
begin
  t.show_lengths(StringIO.open("Hello World!\nHello Mars!")) # Error in show_lengths
rescue
  puts "t.show_lengths fails with StringIO:\n\tException type: <#{$!.class}>\n\tmessage: <#{$!}>" 
end
end

[~/srcLPPRuby/TheRubyProgrammingLanguage/Chapter3DatatypesAndObjects]$ ruby type_versus_class_
10 19 11 52 22 52 68 26 30 8 12 6 4 1 18 15 1 58 58 8 87 9 100 6 4
t.show_lengths fails with StringIO:
  Exception type: <RuntimeError>
  message: <Error in show_lengths>

```

Aunque quizá sea larga y requiera para su comprensión completa mas conocimientos de los que tenemos en este punto, escuchar la charla *Less: The Path to a Better Design* por Sandi Metz en Vimeo puede resultar útil. Metz muestra como el uso de *Tipado Pato (Duck Typing)* puede ayudar a desacoplar nuestras clases, mejorando así nuestros diseños y haciendo que nuestro software sea mas adaptable a los cambios que necesariamente habrán de sobrevenir durante su vida útil.

Véanse algunos comentarios de Nathan Youngman sobre la lectura del libro "Practical Object Oriented Design in Ruby: An Agile Primer" de Sandi Metz:

Procedural programs operate by performing a series of steps, whereas object-oriented programs are about sending messages.

When one object tells another to perform a series of steps, it knows too much! To minimize dependencies, an object needs to ask for what it wants, and trust that the receiver takes the necessary steps. It doesn't need to know the details.

The distinction between a message that asks for what the sender wants and a message that tells the receiver how to behave may seem subtle but the consequences are significant.

Rather than depending on a concrete Class, consider relying on a duck type that implements the necessary message. *Depending on a "duck-like" thing is much more flexible.*

14.1.9. Coerción

Numeric types define a conversion method named `coerce`.

Numeric operators are written so that if they don't know the type of the righthand operand, they invoke the `coerce` method of the righthand operand, passing the lefthand operand as an argument.

Consideremos este código:

```
$ cat coerce.rb
require 'point'

p = Point.new(2,3)
q = 2*p
puts q
```

Puesto que `Fixnum` no sabe como multiplicarse por un `Point`, invoca al método `coerce` de su argumento (véase el código de `point.rb` en la sección 14.1.6).

```
33  def coerce(other)
34    [self, other]
35  end
36

28  def *(scalar)      # Define * to perform scalar multiplication
29    return Point.new(@x*scalar,   @y*scalar) if scalar.is_a? Numeric
30    return Point.new(@x*scalar.x, @y*scalar.y) if scalar.is_a? Point
31  end
```

Transformando la llamada `2*p = 2.*(p)` en `p*2 = p.%(2)`.

Veamos una sesión paso a paso con el depurador:

```
[09:01]$ ruby -rdebug -I. coerce.rb
coerce.rb:1:require 'point'
(rdb:1) n
coerce.rb:3:p = Point.new(2,3)
(rdb:1)
n
```

```

coerce.rb:4:q = 2*p
(rdb:1) p p
(2,3)

```

En vez de saltar sobre la llamada `n(ext)` entremos en el método usando `s(tep)`:

```

(rdb:1) s
point.rb:34:      [self, other]

```

¡Aja! estamos en el código de `coerce`:

```

(rdb:1) where
--> #1 point.rb:34:in `coerce'
#2 coerce.rb:4
(rdb:1) self
(2,3)
(rdb:1) other
2
(rdb:1) s
point.rb:29:  return Point.new(@x*scalar,    @y*scalar) if scalar.is_a? Numeric
(rdb:1) n
coerce.rb:5:puts q
(rdb:1)
n
(4,6)

```

The intent of the `coerce` method is to convert the argument to the same type as the object on which the method is invoked, or to convert both objects to some more general compatible type.

```

[13] pry(main)> 2.coerce(1.1)
=> [1.1, 2.0]
[14] pry(main)> 2+1.1
=> 3.1
[8] pry(main)> require 'rational'
=> true
[19] pry(main)> z = Rational(1,3)
=> (1/3)
[20] pry(main)> 2.coerce(z)
=> [0.3333333333333333, 2.0]
[21] pry(main)> z.coerce(2)
=> [(2/1), (1/3)]
[22] pry(main)> 2+z
=> (7/3)

```

14.1.10. Acceso a Arrays y a Hashes

Véase el código en la sección 14.1.6

```

# Define [] method to allow a Point to look like an array or
# a hash with keys :x and :y
def [](index)
  case index
  when 0, -2: @x          # Index 0 (or -2) is the X coordinate
  when 1, -1: @y          # Index 1 (or -1) is the Y coordinate
  when :x, "x": @x        # Hash keys as symbol or string for X
  when :y, "y": @y        # Hash keys as symbol or string for Y

```

```

else nil                      # Arrays and hashes just return nil on bad indexes
end
end

```

Nota Sobre el Case Ruby 1.9 introduces an incompatible syntax change for conditional statements such as `if` and `case/when`. Previously a colon could be used as a shorthand for a `then` statement; this is perhaps most useful with multiple `when` statements on one line.

The following is legitimate ruby in 1.8:

```

case x
when Regexp : puts 'a regex'
when Hash   : puts 'a regex'
when Numeric: puts 'a number'
when String  : puts 'a string'
end

```

But not in ruby 1.9; now an explicit 'then' statement must be used:

```

case x
when Regexp then puts 'a regex'
...

```

14.1.11. Enumeración de Coordenadas

```

09:37] [~/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/poi
[09:38] [~/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/poi
:.

[09:39] [~/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/poi
ruby-1.9.2-p290 :001 > require 'point'
=> true
ruby-1.9.2-p290 :002 > class Point
ruby-1.9.2-p290 :003?>   def each
ruby-1.9.2-p290 :004?>     yield @x
ruby-1.9.2-p290 :005?>     yield @y
ruby-1.9.2-p290 :006?>   end
ruby-1.9.2-p290 :007?>   end
=> nil
ruby-1.9.2-p290 :008 > p = Point.new(2,1)
=> (2,1)
ruby-1.9.2-p290 :009 > p.each do |x| puts "coord = #{x}" end
coord = 2
coord = 1
=> nil

```

14.1.12. Igualdad de Puntos

Repáse la sección *Igualdad de Objetos* 10.8.5

```

def ==(o)                  # Is self == o?
  if o.is_a? Point        # If o is a Point object
    @x==o.x && @y==o.y   # then compare the fields.
  elsif                   # If o is not a Point
    false                 # then, by definition, self != o.
  end
end

```

```

def ==(o)                      # Is self == o?
  @x == o.x && @y == o.y    # Assume o has proper x and y methods
rescue                         # If that assumption fails
  false                          # Then self != o
end

class Point
  alias eql? ==
end

def eql?(o)
  if o.instance_of? Point
    @x.eql?(o.x) && @y.eql?(o.y)
  elsif
    false
  end
end

```

eql? y las Claves de un hash

El predicado `eql?` es usado por Ruby para comparar las claves de un hash. Por ello, si redefinimos `eql?` en nuestra clase, es conveniente que también definamos el método `hash` que es usado por Ruby para calcular el *hashcode* del objeto.

Se debe cumplir siempre que si dos objetos son iguales según `eql?` entonces sus hashcodes deben ser iguales.

No parece que sea una buena idea usar objetos mutables como índices de hashes. Si es el caso, es mejor congelar el objeto (véase la sección 10.8.12).

Véase el código en 14.1.6

```

def hash
  @x.hash + @y.hash
end

def hash
  code = 17
  code = 37*code + @x.hash
  code = 37*code + @y.hash
  # Add lines like this for each significant instance variable
  code # Return the resulting code
end

```

14.1.13. Ordenando Puntos

Véase el código en 14.1.6

- Comparable
- Enumerable

```
include Comparable # Mix in methods from the Comparable module.
```

```
# Define an ordering for points based on their distance from the origin.
# This method is required by the Comparable module.
def <=>(other)
```

```

    return nil unless other.instance_of? Point
    @x**2 + @y**2 <=> other.x**2 + other.y**2
end

p,q = Point.new(1,0), Point.new(0,1)
p == q      # => false: p is not equal to q
p < q       # => false: p is not less than q
p > q       # => false: p is not greater than q

```

14.1.14. Un Punto Mutable

```
[~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/point]$ cat -n mutable_point.rb
1  #encoding : utf-8
2  class MutablePoint
3      def initialize(x,y)
4          @x, @y = x, y
5      end
6
7      def x; @x; end           # The getter method for @x
8      def y; @y; end           # The getter method for @y
9
10     def x=(value)           # The setter method for @x
11         @x = value
12     end
13
14     def y=(value)           # The setter method for @y
15         @y = value
16     end
17
18     def add!(p)              # Add p to self, return modified self
19         @x += p.x
20         @y += p.y
21         self
22     end
23 # -----
24     def add(p)               # A nonmutating version of add!
25         q = self.dup          # Make a copy of self
26         q.add!(p)             # Invoke the mutating method on the copy
27     end
28
29 end
30
31 if __FILE__ == $0
32     z = MutablePoint.new(1,2)
33     p z
34     t = MutablePoint.new(2,1)
35     p t
36     t.x = 4
37     p t
38     z.add!(t)
39     p z
40 end
```

14.1.15. Creando Clases con Struct

```
$ cat -n struct_point.rb
1 Point = Struct.new(:x, :y)    # Creates new class, assigns to Point
2
3 p = Point.new(1,2)    # => #<struct Point x=1, y=2>
4 p.p.x                # => 1
5 p.p.y                # => 2
6 p.x = 3               # => 3
7 p.p.x                # => 3
8 -----
9 p[:x] = 7
10 p.p[:x]             # => 7: same as p.x
11 p.p[1]               # => 2: same as p.y
12 p.each{|c| print c} # prints "72"
13 puts
14 p.each_pair{|n,c| print n,c; puts }
15 puts
16 -----
17 p = Point.new(4,2)
18 q = Point.new(4,2)
19 p.q == p              # => true
20 h = {q => 1}  # Create a hash using q as a key
21 p.h[p]                # => 1: extract value using p as key
22 p.q.to_s              # => "#<struct Point x=4, y=2>"
23
24 require 'extending_struct_point'
25
26 puts "sorting!"
27 puts [p, q].sort
```

Extendiendo clases creadas con Struct Por supuesto podemos abrir la clase creada con Struct y extenderla:

```
$ cat -n extending_struct_point.rb
1 class Point                  # Open Point class for new methods
2   def add!(other)            # Define an add! method
3     self.x += other.x
4     self.y += other.y
5     self
6   end
7
8   include Comparable          # Include a module for the class
9   def <=>(other)            # Define the <=> operator
10    return nil unless other.instance_of? Point
11    self.x**2 + self.y**2 <=> other.x**2 + other.y**2
12  end
13 end

$ ruby -I. struct_point.rb
1
2
3
```

```

7
2
72
x7
y2

false
nil
"#<struct Point x=4, y=2>"
sorting!
#<struct Point x=4, y=2>
#<struct Point x=7, y=2>

```

haciendo Inmutable una clase creada con Struct

Las clases creadas por Struct son mutables. Si la queremos hacer inmutable usamos `undef`:

```

$ cat -n making_struct_inmutable.rb
1 Point = Struct.new(:x, :y) # Define mutable class
2 class Point               # Open the class
3   undef x=, y=, []=         # Undefine mutator methods
4 end
5
6 z = Point.new(2,3)
7
8 z.x = 4

```

```
[~/point]$ ruby making_struct_inmutable.rb
making_struct_inmutable.rb:8:in '<main>':
undefined method 'x=' for #<struct Point x=2, y=3> (NoMethodError)
```

14.1.16. Un Método de Clase

Let's take another approach to adding Point objects together. Instead of invoking an instance method of one point and passing another point to that method, let's write a method named `sum` that takes any number of `Point` objects, adds them together, and returns a new `Point`. This method is not an instance method invoked on a `Point` object. Rather, it is a class method, invoked through the `Point` class itself. We might invoke the `sum` method like this:

```
total = Point.sum(p1, p2, p3) # p1, p2 and p3 are Point objects
```

Keep in mind that the expression `Point` refers to a Class object that represents our point class. To define a class method for the `Point` class, what we are really doing is defining a *singleton method* of the `Point` object. To define a singleton method, use the `def` statement as usual, but specify the object on which the method is to be defined as well as the name of the method. Our class method `sum` is defined like this:

```
[~/point]$ cat -n point_with_sum.rb
1 require 'point'
2 class Point
3   attr_reader :x, :y      # Define accessor methods for our instance variables
4
5   def Point.sum(*points) # Return the sum of an arbitrary number of points
6     x = y = 0
7     points.each { |p| x += p.x; y += p.y }
```

```

8     Point.new(x,y)
9   end
10 end
11
12 if __FILE__ == $0
13   z = [[1,1],[2,4],[3,2]].map { |p| Point.new(*p) }
14   p z
15   w = Point.sum(*z)
16   p w
17   r = Point.sum(z[0], z[1])
18   p r
19 end
20

```

This definition of the class method names the class explicitly, and mirrors the syntax used to invoke the method. Class methods can also be defined using `self` instead of the class name. Thus, this method could also be written like this:

```

def self.sum(*points) # Return the sum of an arbitrary number of points
  x = y = 0
  points.each { |p| x += p.x; y += p.y }
  Point.new(x,y)
end

```

Using `self` instead of `Point` makes the code slightly less clear, but it's an application of the *DRY* (Don't Repeat Yourself) principle. If you use `self` instead of the class name, you can change the name of a class without having to edit the definition of its class methods.

There is yet another technique for defining class methods. Though it is less clear than the previously shown technique, it can be handy when defining multiple class methods, and you are likely to see it used in existing code:

```

# Open up the Point object so we can add methods to it
class << Point      # Syntax for adding methods to a single object
  def sum(*points) # This is the class method Point.sum
    x = y = 0
    points.each { |p| x += p.x; y += p.y }
    Point.new(x,y)
  end

  # Other class methods can be defined here
end

```

This technique can also be used inside the class definition, where we can use `self` instead of repeating the class name:

```

class Point
  # Instance methods go here

  class << self
    # Class methods go here
  end
end

```

También podemos poner `self.sum` o bien `class << self ... end`

```
[~/point]$ ruby point_with_sum.rb
[(1,1), (2,4), (3,2)]
(6,7)
(3,5)
```

14.1.17. Constantes

```
1 class Point
2   attr_reader :x, :y    # Define accessor methods for our instance variables
3
4   include Comparable
5   include Enumerable
6
7   def initialize(x,y)
8     @x,@y = x, y
9   end
10
11 ORIGIN = Point.new(0, 0)
12 E_1    = Point.new(1, 0)
13 E_2    = Point.new(0, 1)
```

Inside the class definition, these constants can be referred to by their unqualified names. Outside the definition, they must be prefixed by the name of the class:

```
>> load "Point2.rb"
=> true
>> Point::E_1
=> #<Point:0x100589f20 @y=0, @x=1>
>> Point::E_1.x
=> 1
>> E_1
NameError: uninitialized constant E_1
  from (irb):4
>>
```

Puesto que las constantes crean instancias de la clase, no podemos definirlas antes de tener definida la versión correcta del método `initialize`:

```
[~/rubytesting]$ cat constant_before_initialize.rb
class Tutu

  C = Tutu.new(5)

  def initialize(x)
    @x = x
  end
end
```

Al ejecutar produce un error:

```
[~/rubytesting]$ ruby constant_before_initialize.rb
constant_before_initialize.rb:3:in `initialize': wrong number of arguments(1 for 0) (ArgumentE
  from constant_before_initialize.rb:3:in `new'
  from constant_before_initialize.rb:3:in `<class:Tutu>'
  from constant_before_initialize.rb:1:in `<main>'
```

Es posible definir constantes de una clase desde fuera de la misma:

```
>> Point::E_3 = Point.new 1, 1
=> #<Point:0x100565df0 @y=1, @x=1>
```

14.1.18. Variables de Clase. Atributos de la Clase

1. Class variables are visible to, and shared by, the class methods and the instance methods of a class, and also by the class definition itself.
2. Like instance variables, class variables are encapsulated; they can be used by the implementation of a class, but they are not visible to the users of a class.
3. Class variables have names that begin with @@.

```
class Point
  # Initialize our class variables in the class definition itself
  @@n = 0          # How many points have been created
  @@totalX = 0     # The sum of all X coordinates
  @@totalY = 0     # The sum of all Y coordinates

  def initialize(x,y) # Initialize method
    @x,@y = x, y      # Sets initial values for instance variables

    # Use the class variables in this instance method to collect data
    @@n += 1          # Keep track of how many Points have been created
    @@totalX += x     # Add these coordinates to the totals
    @@totalY += y
  end

  # A class method to report the data we collected
  def self.report
    # Here we use the class variables in a class method
    puts "Number of points created: #@n"
    puts "Average X coordinate: #{@totalX.to_f/@@n}"
    puts "Average Y coordinate: #{@totalY.to_f/@@n}"
  end
end
```

14.1.19. Variables de Instancia de Clase

1. Classes are objects and can have instance variables just as other objects can
2. The instance variables of a class—often called class instance variables—are not the same as class variables
3. They are similar enough that they can often be used instead of class variables

```
class Point
  # Initialize our class instance variables in the class definition itself
  @n = 0          # How many points have been created
  @totalX = 0     # The sum of all X coordinates
  @totalY = 0     # The sum of all Y coordinates

  def initialize(x,y) # Initialize method
    @x,@y = x, y      # Sets initial values for instance variables
```

```

end

def self.new(x,y)    # Class method to create new Point objects
  # Use the class instance variables in this class method to collect data
  @n += 1            # Keep track of how many Points have been created
  @totalX += x       # Add these coordinates to the totals
  @totalY += y

  super              # Invoke the real definition of new to create a Point
  # More about super later in the chapter
end

# A class method to report the data we collected
def self.report
  # Here we use the class instance variables in a class method
  puts "Number of points created: #{@n}"
  puts "Average X coordinate: #{@totalX.to_f/@n}"
  puts "Average Y coordinate: #{@totalY.to_f/@n}"
end
end

```

1. Because class instance variables are just instance variables of class objects, we can use `attr`, `attr_reader`, and `attr_accessor` to create accessor methods for them
2. The trick, however, is to invoke these metaprogramming methods in the right context
3. Recall that one way to define class methods uses the syntax `class << self`
4. This same syntax allows us to define attribute accessor methods for class instance variables:

```

class << self
  attr_accessor :n, :totalX, :totalY
end

```

El siguiente ejemplo muestra como crear accessors a las variables de instancia de clase:

```

$ cat -n creating_class_accessors.rb
 1  class Tutu
 2
 3  class << self
 4      attr_accessor :n
 5  end
 6
 7 end
 8
 9 Tutu.n = 4
10 p Tutu.n
11
12
[~/point]$ ruby creating_class_accessors.rb
4

```

1. Una desventaja de las variables de instancia de clase es que no pueden ser utilizadas [directamente](#) dentro de los métodos de instancia, mientras que las variables de clase si son accesibles.

2. Otra desventaja es el riesgo potencial de confundirlas con variables de instancia ordinarias
3. Aún así su uso es mas recomendable que el de las variables de clase

14.1.20. Práctica: La Clase Punto del Plano

- Defina una clase **Point** que represente puntos (x, y) del plano cartesiano.
- Defina el método **to_s**
- Defina métodos para el acceso a los atributos
- Defina la suma de puntos $(x_1, y_1) + (x_2, y_2) = (x_1 + y_1, x_2 + y_2)$. Compruebe que el objeto sumando hereda de la clase **Point** usando el predicado **is_a?**.
- Defina el producto de un vector/punto por un escalar $(x, y) * \lambda = (\lambda * x, \lambda * y)$
- Para que un entero se pueda multiplicar por un punto $2 * p$ es necesario definir un método **coerce** en la clase **Point**:

```
def coerce(other)
  [self, other]
end
```

Los tipos numéricos definen un método de conversión denominado **coerce**. Cuando se llama al método ***** de 2: $2 * (p)$ este llamará primero al método **coerce** de **Point** y después intentará multiplicar los elementos del array retornado.

```
>> 1.1.coerce(1)
=> [1.0, 1.1]
>> r = Rational(1,3)
=> Rational1, 3
>> r.coerce(2)
=> [Rational2, 1, Rational1, 3]
```

defina **coerce** de manera que pueda multiplicarse un escalar por un **Point**.

- Defina el opuesto de un punto $-(x, y) = (-x, -y)$. El método para definir el menos unario se llama **-@**:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n unaryminus.rb
 1  class Tutu
 2    def initialize(s)
 3      @s = s
 4    end
 5
 6    def -@
 7      @s.reverse
 8    end
 9
10   def to_s
11     @s
12   end
13 end
14
15 if __FILE__ == $0
16   a = Tutu.new("dlrow olleH")
17   puts -a
18 end
```

Al ejecutarse produce:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby unaryminus.rb
Hello world
```

- Redefine el método [] de manera que pueda accederse a un punto como si fuera un array de sólo lectura. Permite el uso de índices negativos. También debería poder accederse usando una notación hash. Por ejemplo, el código:

```
p = Point.new(1,2)
puts "#{p[0]}, #{p[1]}"
puts "#{p[-2]}, #{p[-1]}"
puts "#{p[:x]}, #{p[:y]}"
puts "#{p['x']}, #{p['y']}"
```

Debería producir como salida:

```
$ ruby Point.rb
1, 2
1, 2
1, 2
1, 2
```

Un hash es una estructura de datos que asocia a cada uno de los elementos de un conjunto de objetos denominados *claves* un *valor*:

```
>> n = { :one => 1, :two => 2, :three => 3}
=> {:one=>1, :two=>2, :three=>3}
>> n[:four] = 4
=> 4
>> n
=> {:one=>1, :two=>2, :three=>3, :four=>4}
>> b = n[:one]+n[:four]
=> 5
>> n[:five]
=> nil
>> n.keys
=> [:one, :two, :three, :four]
>> n.values
=> n.each { |k,v| puts "value for #{k} is #{v}" }
value for one is 1
value for two is 2
value for three is 3
value for four is 4
=> [1, 2, 3, 4]
```

- Si queremos que los objetos de una clase puedan ser usados como claves de un hash es necesario que dispongan de un método `hash` que retorne un código hash de tipo Fixnum. La clase Hash compara las claves usando el predicado `eql?`. Este predicado es similar a `==` pero es mas restrictivo. Por ejemplo, en las subclases de `Numeric` `==` permite la conversión de tipos mientras que `eql?` no:

```
>> 1 == 1.0
=> true
>> 1.eql?(1.0)
=> false
```

Permita que los objetos Point puedan ser usados como índices de un hash. El siguiente código:

```
p = Point.new(1,2)
q = Point.new(1,1)
m = { p => 1, q => 2 }
puts m[p]
puts m[q]
```

deberá producir como salida:

```
1
2
```

Para eso deberá definir:

- El método `hash` de un `Point`: Puede usar con algoritmo de hashCode el descrito en Effective Java (Addison-Wesley 2001) por Joshua Bloch
- El predicado `eql?`. El predicado deberá retornar `true` si el otro objeto es exactamente de la clase `Point`. Véase el método `instance_of?`:

```
>> 4.class
=> Fixnum
>> 4.class.superclass
=> Integer
>> 4.class.superclass.superclass
=> Numeric
>> 4.class.superclass.superclass.superclass
=> Object
>> 4.class.superclass.superclass.superclass.superclass
=> nil
>> 4.instance_of? Fixnum
=> true
>> 4.instance_of? Integer
=> false
>> 4.instance_of? Numeric
=> false
>> 4.is_a? Numeric
=> true
```

- Defina un método `each` que produzca primero la primera coordenada y después la segunda. Incluya el módulo `Enumerable` y compruebe que puede usar algunos de los métodos proveídos por este como `all?` y `any?`.
- Redefina el método `==` para que dos puntos sean iguales si sus coordenadas son iguales. Siguen dos posibles implementaciones:

```
def ==(o)
  if o.is_a? Point
    @x==o.x && @y==o.y
  else
    false
  end
end
```

```
def ==(o)
  @x==o.x && @y==o.y
rescue
  false
end
```

¿Sabría

decir cual es la diferencia entre ambas?

- Incluya el módulo Comparable. Defina el método `<=>` guerra de las galaxias de manera que dos Point se comparan según su distancia al origen de coordenadas. Si `self` es menor que el objeto, `<=>` deberá retornar -1, si son iguales 0 y 1 si es mayor. Compruebe que puede usar los métodos proveídos por Comparable.

Este programa:

```
p = Point.new(1,2)
q = Point.new(1,1)
r = Point.new(2,1)

# It is not a total order relation
puts p > r
puts p < r
puts p == r
puts p > q
```

Deberá producir una salida como esta:

```
false
false
false
true
```

- Incluya el módulo enumerable y compruebe que funcionan los métodos `sort`, `min` y `max`
- Defina una clase `Point3D` que reutilice y herede de `Point` y que tenga las mismas funcionalidades.

14.2. Visibilidad de los métodos: Público, Privado, Protegido

- *Encapsulation* means that the internal representation of an object is generally hidden from view outside of the object's definition. Typically, only the object's own methods can directly inspect or manipulate its fields. Smalltalk and Ruby only allow access via object methods,
- *The Uniform Access Principle* was put forth by Bertrand Meyer. It states *All services offered by a module should be available through a uniform notation, which does not betray whether they are implemented through storage or through computation.*

This principle applies generally to the syntax of object-oriented programming languages. In simpler form, it states that there should be no difference between working with an attribute, precomputed property, or method/query:

1. [~/srcDaveThomasProgrammingRuby]\$ cat virtual_attributes.rb

```
class BookInStock
  attr_reader :isbn
  attr_accessor :price
  def initialize(isbn, price)
    @isbn = isbn
    @price = Float(price)
  end
  def price_in_cents
    Integer(price*100 + 0.5)
  end
  # ...
end
book = BookInStock.new("isbn1", 33.80)
```

```
puts "Price = #{book.price}"
puts "Price in cents = #{book.price_in_cents}"
```

Here we've used attribute methods to create a virtual instance variable. To the outside world, `price_in_cents` seems to be an attribute like any other. Internally, though, it has no corresponding instance variable.

```
2. [~/srcDaveThomasProgrammingRuby]$ ruby virtual_attributes.rb
Price = 33.8
Price in cents = 3380
```

- Los métodos son normalmente `public` a menos que sean explícitamente declarados como `private` o `protected`.
- Una excepción es `initialize` que es implícitamente privado.
- Otra excepción son los métodos declarados fuera de cualquier definición de clase: son declarados como métodos de instancia privados de la clase `Object`

```
[~/rubytesting]$ cat privado_main2.rb
require 'pry'
```

```
def tutu
  "tut "
end
```

```
class Titi
  def toto
    tutu()
  end
end
```

```
binding.pry()
```

```
[~/rubytesting]$ ruby privado_main2.rb
From: /Users/casiano/local/src/ruby/rubytesting/privado_main2.rb @ line 15 :
 10:   end
 11: end
 12:
 13: binding.pry()
 14:
=> 15: z = Titi.new
 16: puts z.toto()
```

```
[1] pry(main)> self
=> main
[2] pry(main)> self.class
=> Object
[4] pry(main)> self.private_methods(false).select {|x| x =~ /tut/ }
=> [:tutu]
[5] pry(main)> self.tutu
NoMethodError: private method `tutu' called for main:Object
from (pry):5:in `<main>'
[6] pry(main)> tutu
=> "tut "
[7] pry(main)> Titi.new.tutu
```

```
NoMethodError: private method `tutu' called for #<Titi:0x007ff03612e700>
from (pry):7:in `<main>'
[8] pry(main)> Titi.new.toto
=> "tut "
[9] pry(main)>
```

- Un método público puede ser llamado desde cualquier lugar
- Si `m` es un método privado, deberá ser invocado implícitamente en `self`, esto es, como *función*, `m()`, y no con receptor explícito: (`obj.m` o bien `self.m`).
- Esto es, un método privado es siempre invocado implícitamente en `self` y no debe ser invocado explícitamente en un objeto
- Las subclases pueden invocar y sobreescribir los métodos `private`
- Al igual que los métodos privados, los métodos `protected` pueden ser invocados desde la clase o sus subclases
- A diferencia de los métodos privados, los métodos `protected` pueden ser invocados en cualquier objeto de la clase y no requieren la invocación implícita via `self`
- La visibilidad de los métodos se declara mediante tres *métodos*: `public`, `private` y `protected`.
- Estos tres métodos son métodos de instancia de la clase `Module`.
- Como todas las clases son módulos y dentro de una definición de clase (pero fuera de las definiciones de métodos) `self` se refiere a la clase, podemos usar estos métodos `public`, `private` y `protected` como si de palabras reservadas se tratara.
- Una clase puede usarlos así:

```
class Point
  # public methods go here

  # The following methods are protected
  protected

  # protected methods go here

  # The following methods are private
  private

  # private methods go here
end
```

- También pueden ser invocados con los nombres de uno o mas métodos (como símbolos o como cadenas). En tal caso la declaración de visibilidad viene después de la definición del método y altera la visibilidad del método con ese nombre.
- Una práctica es declarar todos los métodos protegidos y privados de golpe, al final de la clase
- Otra práctica es declarar la visibilidad justo después de la definición del método:

```
class Widget
  def x                                # Accessor method for @x
    @x
```

```

end
protected :x           # Make it protected

def utility_method      # Define a method
nil
end
private :utility_method # And make it private
end

```

- Si se desea declarar un método de clase como privado se usa el *método private_class_method*:

```
private_class_method :new
```

- Si se desea hacer público de nuevo un método de clase privado se usa el método *public_class_method*
- Las capacidades de metaprogramación de Ruby hacen posible invocar métodos privados y protegidos e incluso el acceso a variables encapsuladas:

```
$ cat -n accesing_private_method.rb
1 class Widget
2   def initialize()
3     @x = 4
4   end
5
6   def x
7     @x
8   end
9   protected :x           # Make it protected
10
11  def utility_method      # Define a method
12    puts "Inside utility_method"
13  end
14  private :utility_method # And make it private
15 end
16
17 w = Widget.new          # Create a Widget
18 w.send :utility_method  # Invoke private method!
19 w.instance_eval { utility_method } # Another way to invoke it
20
21 w.instance_eval { puts @x }      # Read instance variable of w
```

Cuando ejecutamos, obtenemos:

```
[~/point]$ ruby accesing_private_method.rb
Inside utility_method
Inside utility_method
4
```

Utilice *public_send* en vez de *send* si no se quiere que, por error, se llame inadvertidamente a un método privado.

14.2.1. Ejemplo de Visibilidad de Métodos Protegidos

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n SubClass.rb
 1 class Point
 2
 3   def chuchu
 4     "Point: chuchu"
 5   end
 6
 7   def chachi
 8     "Point: chachi"
 9   end
10
11  private  :chuchu
12  protected :chachi
13 end
14
15 class SubClass < Point
16
17   def titi1
18     chuchu
19   end
20
21   def titi2
22     self.chuchu
23   end
24
25   def titi3
26     chachi
27   end
28
29   def titi4
30     self.chachi
31   end
32 end
33
34 if __FILE__ == $0
35   x = SubClass.new
36   puts x.titi1
37   begin
38     puts x.titi2
39     rescue
40       puts "Can't call private meth. via 'self'. Exception type: <#{$!.class}> message"
41   end
42
43   puts x.titi3
44
45   puts x.titi4
46
47   begin
48     puts x.chachi
49     rescue
50       puts "Can't call protected meth. from outside. Exception type: <#{$!.class}> message"
51   end
```

```
52 end
```

La ejecución del programa produce la salida:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby SubClass.rb
Point: chuchu
Can't call private meth. via 'self'. Exception type: <NoMethodError> message: <private method
Point: chachi
Point: chachi
Can't call protected meth. from outside. Exception type: <NoMethodError> message: <protected m
```

The rule about when a **protected** method can be invoked can be described as follows:

A **protected** method defined by a class V (**Visibility**) may be invoked on an object c by a method in an object m (**main** en el ejemplo) if and only **if the classes of c and m are both subclasses of the class V.**

14.2.2. Ejemplo de Visibilidad

```
~/rubytesting/programmingRuby$ !cat
cat -n Visibility.rb
1 class Visibility
2   protected
3     def tutu_prot
4       puts "Prot"
5     end
6
7   private
8     def tutu_priv
9       puts "Priv"
10    end
11
12  public
13    def tutu_pub
14      puts "Pub"
15    end
16 end
17
18
19 class Child < Visibility
20
21   def chachi
22     puts "in chachi"
23     self.tutu_prot
24   end
25
26   def chuchu
27     puts "in chuchu"
28     self.tutu_priv
29   end
30 end
31
32 if $0 == __FILE__
33   b = Child.new
34   b.tutu_pub      # <- Pub
35   begin
```

```

36     b.tutu_priv    # Exception <NoMethodError> raised <private method 'tutu_priv' called
37 rescue NoMethodError
38     puts "Exception <#${!.class}> raised <#${!}>""
39 end
40 b.chachi  # <- in chachi
41     #      Prot
42 begin
43     b.chuchu # <- Exception <NoMethodError> raised <private method 'tutu_priv' called fo
44 rescue
45     puts "Exception <#${!.class}> raised <#${!}>""
46 end
47
48 begin
49     c = Visibility.new
50     c.tutu_pub  # <- Pub
51     c.tutu_prot # <- Exception <NoMethodError> raised <protected method 'tutu_prot' call
52 rescue NoMethodError
53     puts "Exception <#${!.class}> raised <#${!}>""
54 end
55 end

```

14.3. Subclases y Herencia

- Si cuando se define una clase no se especifica una superclase entonces nuestra clase hereda implícitamente de `Object`.
- The fact that *classes may have multiple subclasses but only a single superclass (single inheritance) means that they can be arranged in a tree structure*, which we call the Ruby class hierarchy.
- The `BasicObject` class is the root of this hierarchy, and every class inherits directly or indirectly from `Object`.
- The *descendants* of a class are the subclasses of the class plus the subclasses of the subclasses, and so on recursively.
- The *ancestors* of a class are the superclass, plus the superclass of the superclass, and so on up to `Object`.
- The syntax for extending a class is simple. Just add a `<` character and the name of the superclass to your class statement.

We then create a subclass using class `Child < Parent`. The `<` notation means we're creating a subclass of the thing on the right; the fact that we use less-than presumably signals that the child class is supposed to be a specialization of the parent.

```
class Point3D < Point    # Define class Point3D as a subclass of Point
end
```

- Es posible heredar de una subclase definida mediante `Struct`:

```
[~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ cat subclassing_a_s
class Punto < Struct.new("Point3D", :x, :y, :z)
  # Superclass struct give us accessor methods, ==, to_s, etc.
  def modulo
    Math.sqrt(x*x+y*y+z*z)
  end
```

```

end
if $0 == __FILE__
  p = Punto.new(1,2,3)
  puts p.x
  puts p.modulo
end

```

Ejecución:

```
[~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ ruby subclassing_a_s
1
3.7416573867739413
```

14.3.1. Heredando Métodos

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n Point3D.rb
1  require "Point"
2  class Point3D < Point
3  end

~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ irb
>> require "Point3D"
=> true
>> p1 = Point3D.new(1,2)
=> #<Point3D:0x10057d5e0 @y=2, @x=1>
>> p2 = Point.new(1,2)
=> #<Point:0x10056a440 @y=2, @x=1>
>> p1.class
=> Point3D
>> p2.class
=> Point
>> p1.to_s
=> "(1,2)"
```

14.3.2. super

- GIST: Ruby OOP example

```
class Person
  def initialize name, dob #dob = date of birth
    @name = name
    @dob = dob
  end

  def age
    (Time.now - @dob) / (365 * 24 * 60 * 60)
  end

  def introduce_yoself
    print "Hey, I'm #{@name} and I was born #{age} years ago.\n"
  end
end
```

```

maxxx = Person.new("Max", Time.new(1988, 9, 11))
maxxx.introduce_yoself
# which prints
# "Hey, I'm Max and I was born 23.986193869358075 years ago."

class CoolDude < Person
  def initialize name, dob, nickname
    super name, dob
    @nickname = nickname
  end

  def introduce_yoself
    print "Yo, they call me '#{@nickname}'."
  end
end

waltr0x = CoolDude.new("Walter", Time.new(1987, 9, 29), "The Laser")
# we've still got this function
print waltr0x.age
# 24.939627905041938
waltr0x.introduce_yoself
# "Yo, they call me 'The Laser'." 
```

1. The Ruby-reserved word `super` is a special kind of method invocation expression. This keyword is used when creating a subclass of another class.
2. By itself, `super` passes the arguments of the current method to the method with the same name in the superclass.
3. If you use `super` as a bare keyword—with no arguments and no parentheses—then all of the arguments that were passed to the current method are passed to the superclass method.
4. As with normal method invocations, the parentheses around `super` arguments are optional. Because a bare `super` has special meaning, however, you must explicitly use a pair of empty parentheses if you want to pass zero arguments from a method that itself has one or more arguments.

Ejemplo: Heredando de GServer

`GServer` implements a [generic server](#), featuring thread pool management, simple logging, and multi-server management.

Any kind of application-level server can be implemented using this class. It accepts multiple simultaneous connections from clients, up to an optional maximum number.

Several services (i.e. one service per TCP port) can be run simultaneously, and stopped at any time through the class method `GServer.stop(port)`.

All the threading issues are handled, saving you the effort. (See Rediscovering Ruby: GServer)

```
[~/rubytesting/inheritance]$ cat gserver_example.rb
# http://www.aimred.com/news/developers/2010/01/16/rediscovering_ruby_gserver/
require 'gserver'

class TimeServer < GServer
  def initialize(port=10001, host='0.0.0.0', *args)
    super
  end 
```

```

def serve(io)
  # line = io.readline
  io.puts(Time.now.to_s)
end

server = TimeServer.new
server.stdlog = $stderr
server.audit = true
server.debug = true
server.start

server.join

[~/rubytesting/inheritance]$ cat gserver_client.rb
require 'socket'

echo = TCPSocket.open( 'localhost', 10001 )
#echo.puts( "Hello\nWorld" )
puts( echo.readline )

[~/rubytesting/inheritance]$ ruby gserver_example.rb
[Mon Oct 27 11:32:23 2014] TimeServer 0.0.0.0:10001 start
[Mon Oct 27 11:32:34 2014] TimeServer 0.0.0.0:10001 client:52492 127.0.0.1<127.0.0.1> connect
[Mon Oct 27 11:32:34 2014] TimeServer 0.0.0.0:10001 client:52492 disconnect

[~/rubytesting/inheritance]$ ruby gserver_client.rb
2014-10-27 11:32:34 +0000

```

14.3.3. Predominancia/Invalidación de Métodos (overriding)

Se entiende por overriding la posibilidad de reemplazar la conducta de un método en la superclase.

```

[19:28][~/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$
class WorldGreeter
  def greet                      # Display a greeting
    puts "#{greeting} #{who}"
  end

  def greeting                   # What greeting to use
    "Hello"
  end

  def who                        # Who to greet
    "World"
  end
end

# Greet the world in Spanish
class SpanishWorldGreeter < WorldGreeter
  def greeting                  # Override the greeting
    "Hola"
  end
end

```

```
# We call a method defined in WorldGreeter, which calls the overridden
# version of greeting in SpanishWorldGreeter, and prints "Hola World"
SpanishWorldGreeter.new.greet
```

Ejecución:

```
[19:28] [/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ Hola World
```

- Nótese que es posible definir una clase *Abstracta* que invoca ciertos *métodos abstractos* cuya definición se delega a las subclases.
- Una clase que extiende una clase *abstracta* es *concreta* si define todos los métodos abstractos de sus ancestros.

```
[19:34] [/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ # This class is abstract; it doesn't define greeting or who
# No special syntax is required: any class that invokes methods that are
# intended for a subclass to implement is abstract.
class AbstractGreeter
  def greet
    puts "#{greeting} #{who}"
  end
end

# A concrete subclass
class WorldGreeter < AbstractGreeter
  def greeting; "Hello"; end
  def who; "World"; end
end

WorldGreeter.new.greet # Displays "Hello World"
```

Ejecución:

```
[19:34] [/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ Hello World
```

Predominancia de Métodos Privados

Los métodos privados no pueden ser invocados con un receptor explícito pero son heredados por las subclases. Si ocurre que

1. Nuestra clase hereda de otra
2. Definimos un método (cualquiera que sea su visibilidad) en nuestra subclase que tiene el mismo nombre que un método privado en la superclase
3. Entonces sobreescribimos el método privado.
4. Si lo hemos hecho por equivocación y el método privado así llamado era usado como un helper por la superclase *es muy posible que esta colisión produzca una conducta indeseable*.
5. Cuando herede procure estar familiarizado con la implementación de la superclase.

El siguiente ejemplo ilustra el problema:

```
[~/srcLPP/Chapter7ClassesAndModules/overriding_methods]$ cat overriding_private_methods.rb
class MotherFoo
  def titi
    foo
  end

  private
  def foo
    "mother foo"
  end
end

class ChildFoo < MotherFoo
  def foo
    "child foo"
  end
end

c = ChildFoo.new
puts c.titi # child foo

print "Private methods of mother class: "
class MotherFoo
  puts MotherFoo.private_instance_methods(false)
end
```

La llamada a `titi` hace que se llame al método `foo` de la clase hija en vez de al método `foo` de la clase madre que era lo que el programador pretendía.

Una manera de paliar este problema consiste en documentar los métodos privados de nuestra clase en una sección para desarrolladores.

Como muestra la ejecución del anterior ejemplo siempre es posible averiguar que métodos privados tiene la superclase llamando a `MotherFoo.private_instance_methods`. De esta forma sabemos que nombres evitar en nuestra clase hija:

```
[~/srcLPP/Chapter7ClassesAndModules/overriding_methods]$ ruby overriding_private_methods.rb
child foo
Private methods of mother class: foo
[~/srcLPP/Chapter7ClassesAndModules/overriding_methods]$
```

Otra forma de aliviar el problema es:

1. crear un método anónimo de clase `Proc` o una lambda
2. usar una variable de instancia de la clase para guardarlo.
3. crear un accessor de lectura para la variable de instancia de la clase como objeto

Esto hace mas difícil que las clases que hereden puedan colisionar con nuestro método `privado`.

```
11:10] [~/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ cat overriding_private_methods2.rb
class MotherFoo
  @ofoo = Proc.new do |x|
    "mother foo #{x}"
  end

  def self.foo(x)
```

```

@ofoo.call(x)
end

def titi(y)
  MotherFoo.foo(y)
end
end

class ChildFoo < MotherFoo
  # puts @ofoo.call # generates an exception: undefined method 'call' for nil:NilClass (NoMethodError)
  def foo(z)
    "child foo #{z}"
  end
  puts MotherFoo.foo(9)
end

c = ChildFoo.new
puts c.titi(4)
puts c.foo(3)

```

Como se vé, es todavía posible - pero ahora explicitándolo con `MotherFoo.foo` - acceder al método "privado" evitando de este modo la colisión casual.

Ejecución:

```
[11:13] [~/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ 
mother foo 9
mother foo 4
child foo 3
```

14.3.4. Aumentando la Conducta Mediante Encadenamiento

Cuando queremos aumentar la funcionalidad de un método de la clase de la que se hereda es necesario invocar el método sobreescrito desde la nueva versión del método. Esto se consigue mediante la palabra reservada `super`.

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n Point3D.rb
 1  require "Point"
 2  class Point3D < Point
 3    def initialize(x, y, z)
 4      super(x, y)
 5      @z = z
 6    end
 7  end
 8  if __FILE__ == $0
 9    p = Point3D.new(1,2,3)
10    puts p.inspect
11  end
```

Ejecución:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby Point3D.rb
#<Point3D:0x1001622f8 @y=2, @x=1, @z=3>
```

- Si se usa `super` sin argumentos y sin paréntesis entonces se le pasarán todos los argumentos con los que ha sido llamado el método actual.

- `super()` lo llamará sin argumentos
- Si el método modifica los valores de los argumentos antes de la llamada a `super` entonces son los valores modificados los que son pasados en la invocación del correspondiente método de la superclase:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n Point3DSuper.rb
 1 class Point
 2   attr_reader :x, :y
 3
 4   def initialize(x,y)
 5     @x,@y = x, y
 6   end
 7 end
 8
 9 class Point3D < Point
10   attr_reader :z
11   def initialize(x, y, z)
12     x, y = x+1, y+1
13     super(x, y)
14     @z = z
15   end
16 end
17 if __FILE__ == $0
18   p = Point3D.new(1,1,1)
19   puts "#{p.x} #{p.y} #{p.z}"
20 end
```

Ejecución:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby Point3DSuper.rb
2 2 1
```

14.3.5. Herencia y Métodos de Clase

Si nuestra clase `Point3D` hereda de `Point` y `Point` define un método de clase `sum` entonces la llamada `Point3D.sum` invocará a `Point.sum` en el caso de que la clase `Point3D` no defina su propio método `sum`.

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n Point3DSuper.rb
 1 class Point
 2   attr_reader :x, :y
 3
 4   def initialize(x,y)
 5     @x,@y = x, y
 6   end
 7
 8   def self.sum(* points)
 9     x, y = 0, 0
10     points.each { |p| x += p.x; y += p.y }
11     Point.new(x, y)
12   end
13
14   def to_s
15     "#@x #@y"
```

```

16   end
17 end
18
19 class Point3D < Point
20   attr_reader :z
21
22   def initialize(x, y, z)
23     super(x, y)
24     @z = z
25   end
26
27   def self.sum(* points)
28     p = super
29     z = points.inject(0) { |a,r| a += r.z }
30     Point3D.new(p.x, p.y, z)
31   end
32
33   def to_s
34     super+" #@z"
35   end
36 end
37
38 if __FILE__ == $0
39   q, r, s = Point3D.new(1,1,1), Point3D.new(2,2,2), Point3D.new(3,3,3)
40
41   puts Point3D.sum(q, r, s)
42 end

```

Como se ve, podemos usar `super` con métodos de clase con el mismo significado que en los métodos de objeto.

La línea 28 es equivalente a `p = super(* points)` y también equivalente a `p = Point.sum(* points)`

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby Point3DSuper.rb
6 6 6
```

14.3.6. Herencia y Variables de la Instancia

Considere el siguiente ejemplo:

```
[~/srcLPP/inheritance]$ cat VariableInheritance.rb
class Point
  def initialize(x,y)
    @x,@y = x, y
  end

  def to_s
    "#@x #@y"
  end
end

class Point3D < Point
  def initialize(x, y, z)
    super(x, y)
    @z = z
  end

```

```

def to_s
  "#@x #@y #@z"
end
end

if __FILE__ == $0
  q, r, s = Point3D.new(1,1,1), Point3D.new(2,2,2), Point3D.new(3,3,3)
  puts q, r, s

  print "The instance variables of point 'q' are: "
  puts q.instance_variables.join(',')
end

```

Observe como en la línea 18 accedemos a las referencias `@x` y `@y` de la superclase `Point`. Este código funciona como cabría esperar:

```
[~/srcLPP/inheritance]$ ruby VariableInheritance.rb
1 1 1
2 2 2
3 3 3
The instance variables of point 'q' are: @x,@y,@z
[~/srcLPP/inheritance]$
```

- Una variable de un objeto/instancia se crea en el momento en el que se le asigna un valor
- Las variables de la instancia no son definidas por la clase y no están relacionadas con los mecanismos de herencia
- Si una subclase utiliza el mismo nombre para una variable de instancia que alguno de sus ancestros la variable será *sobreescrita*

Puesto que las variables de instancia no están relacionadas con la herencia, se deduce que una variable de instancia usada por una subclase no puede *ocultar* una variable de instancia definida usada en la superclase. Si una subclase usa una variable de instancia con el mismo nombre que una variable utilizada por uno de sus ancestros, sobreescibirá el valor de la variable de su ancestro. *Esto puede ser que se haga con intencionalidad, para cambiar la conducta del ancestro o puede que ocurra por accidente. En este último caso seguramente dará lugar a bugs.*

1. Observe que, como en el ejemplo, podemos siempre averiguar cuales son las variables de instancia de un objeto `q` llamando a `q.instance_variables`
2. En general, el mecanismo de herencia debe ser usado para extender clases en Ruby sólo cuando estamos familiarizados y tenemos control de la superclase.
3. Compruebe que nombres usa la superclase para sus variables de instancia y evítelos
4. Documente sus variables de instancia en la sección para desarrolladores
5. Prefije los nombres de sus variables de instancia con el nombre de la clase.

14.3.7. Herencia y Variables de Clase y Variables de Instancia de una Clase

Las variables de clase (como `@@count`) son compartidas por una clase y todas sus subclases.

Si una variable `@@a` es definida por una clase `A` entonces cualquier subclase `B` de `A` puede usar dicha variable.

Si una subclase asigna un valor a una variable de clase en uso por una superclase no es que cree una copia de dicha variable sino que altera el valor de la variable de la superclase. Ese será el valor que vean todas las otras subclases de la superclase.

Este ejemplo ilustra el comportamiento:

```
[~/srcLPP/inheritance]$ cat InheritanceAndClassVariables.rb
class A
  @@value = 1
  def A.value
    @@value
  end
end

puts A.value # 1

class B < A
  @@value = 2
end

puts A.value # 2
puts B.value # 2

class C < A; @@value = 3; end

puts B.value # 3
puts C.value # 3

puts "A.class_variables = #{A.class_variables.inspect}" # [:@@value]
puts "B.class_variables = #{B.class_variables.inspect}" # []
puts "C.class_variables = #{C.class_variables.inspect}" # []
```

Al ejecutar tenemos:

```
[~/srcLPP/inheritance]$ ruby InheritanceAndClassVariables.rb
1
2
2
3
3
A.class_variables = [:@@value]
B.class_variables = []
C.class_variables = []
```

Por esta razón es preferible usar variables de instancia de la clase en vez de variables de clase. Nótese que siempre podemos averiguar las variables de clase de una clase mediante el método `class_variables`.

Herencia y Variables de Instancia del Objeto Clase

Observa la diferencia cuando se usan variables de instancia de la clase:

```
13:11] [~/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ c
class A
  @value = 1
  def self.value
    @value
  end
```

```

end

puts A.value # 1

class B < A
  @value = 2
end

puts A.value # 1
puts B.value # 2

class C < A; @value = 3; end

puts B.value # 2
puts C.value # 3

[13:12] [~/Dropbox/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ 1
1
2
2
3

```

1. Véase el excelente artículo Ruby, Smalltalk and Class Variables by Pat Shaughnessy

14.3.8. Herencia y Constantes

Las constantes son heredadas y pueden ser sobreescritas en la misma forma que los métodos.

```

~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n InheritanceAndCons
 1 class Point
 2   def initialize(x,y)
 3     @x,@y = x, y
 4   end
 5
 6   ORIGIN = Point.new(0,0)
 7
 8   def to_s; "#@x #@y" end
 9
10  def chuchu
11    puts ORIGIN
12  end
13 end
14
15 class Point3D < Point
16   def initialize(x, y, z)
17     super(x, y); @z = z
18   end
19   def to_s; super+" #@z" end
20
21   puts ORIGIN          # 0 0
22   ORIGIN = Point3D.new(0, 0, 0)
23   puts ORIGIN          # 0 0 0
24   puts Point::ORIGIN  # 0 0

```

```

25
26 end
27
28 x = Point3D.new(1, 2, 3)
29 x.chuchu          # 0 0

```

Obsérvese como la llamada al método `chuchu` de la línea 29 muestra `Point::ORIGIN` y no `Point3D::ORIGIN`:

```

~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby InheritanceAndConstan
0 0          # linea 21
0 0 0        #      23
0 0          #      24
0 0          #      29

```

La diferencia entre constantes y métodos es que las constantes se buscan en el ámbito léxico del lugar en el que son usadas antes de mirar en la jerarquía de herencia. Esto significa que si `Point3D` hereda de `Point` métodos que usan la constante `ORIGIN`, la conducta de dichos métodos no cambia cuando `Point3D` define su propia constante `ORIGIN`.

14.3.9. Averiguando los Descendientes de una Clase (descendants)

Véase:

- The `ObjectSpace` module contains a number of routines that interact with the garbage collection facility and allow you to traverse all living objects with an iterator.
- `each_object([module]) {|obj| ... }` Calls the block once for each living, nonimmediate object in this Ruby process.

If module is specified, calls the block for only those classes or modules that match (or are a subclass of) module.

Returns the number of objects found. Immediate objects (`Fixnums`, `Symbols` `true`, `false`, and `nil`) are never returned.

```

[~/TheRubyProgrammingLanguageDrive/Chapter7ClassesAndModules]$ cat descendantsofaclass.rb
require 'pp'

class Parent
  def self.descendants
    ObjectSpace.each_object(Class).select { |klass| klass < self }
  end
end

class Child < Parent
end

class GrandChild < Child
end

pp Parent.descendants # [GrandChild, Child]
pp Child.descendants # [GrandChild]

[~/TheRubyProgrammingLanguageDrive/Chapter7ClassesAndModules]$ ruby descendantsofaclass.rb
[GrandChild, Child]
[GrandChild]

```

14.3.10. Delegación

La delegación es un patrón de diseño que consiste en que ciertas tareas, en vez de ser realizadas por el objeto son delegadas en ciertos objetos asistentes.

Delegación usando DelegateClass Una forma de delegar en Ruby consiste en que la clase sea una subclase de `DelegateClass`, a la que se le pasa el nombre de la clase en la que se delega como parámetro (`Flintstone` en el ejemplo).

```
class Fred < DelegateClass(Flintstone)
  def initialize
    # ...
    super(Flintstone.new(...))
  end
  # ...
end
```

Luego, en el `initialize` llamamos al de la superclase, pasandole como argumento el objeto en el que se delega.

Por ejemplo, podríamos reescribir la práctica de los conjuntos en la sección 14.6 delegando en la clase `Array` los métodos `length` e `include?` (Puede encontrar este fichero en <https://gist.github.com/crguezl/>):

```
~/rubystesting/sets$ cat -n setsdelegate.rb
 1  require 'delegate'
 2  class Set2 < DelegateClass(Array)
 3
 4    attr_accessor :sep
 5    attr_reader :a
 6    protected :a
 7
 8    def initialize(a)
 9      @sep = ', '
10      @a = a.uniq.sort
11      super(@a)
12    end
13
14    def to_s
15      @a.join(@sep)
16    end
17
18    # delegated
19    #def length
20    #  @a.length
21    #end
22
23    alias cardinal length
24
25    # delegated
26    #def include?(x)
27    #  @a.include? x
28    #end
29
30  end
31
32  if __FILE__ == $0
```

```

33     x = Set2.new [1, 2, 3, 3, 4]
34     puts "x = #{x}"
35     puts "x.length = #{x.length}"           # x.length = 4
36     puts "x.cardinal = #{x.cardinal}"       # x.cardinal = 4
37
38     puts "x includes 2 = #{x.include? 2}"    # x includes 2 = true
39     puts "x includes 8 = #{x.include? 8}"    # x includes 8 = false
40 end

```

En realidad `DelegateClass(Array)` es una llamada al método `DelegateClass` del package `delegate`.

- Esta llamada crea y retorna una nueva clase, la cual define un `method_missing`
- Este `method_missing` redirecciona las llamadas al objeto en el que se delega.

```

~/rubystesting/sets$ ruby setsdelegate.rb
x = 1, 2, 3, 4
x.length = 4
x.cardinal = 4
x includes 2 = true
x includes 8 = false

```

Otra forma de delegar es usar el método `delegate` de Rails. Podemos instalar `rails` usando `gem`:

```

~$ sudo gem install rails
Password:
Successfully installed rails-3.1.1
1 gem installed
Installing ri documentation for rails-3.1.1...
file 'lib' not found
Installing RDoc documentation for rails-3.1.1...
file 'lib' not found
~$

```

Delegación usando Rails

El método `delegate` recibe la lista de nombres de los métodos (especificados como símbolos o strings) a delegar y el nombre del objeto atributo en el que se delega, especificado mediante la clave `:to`. Véase <https://gist.github.com/crguezl/>.

```

~/rubystesting/TheRubyProgrammingLanguage/chapter8ReflectionandMetaprogramming$ cat -n delegati
1  require "rails"
2  module I
3    def f
4      puts "#{self.class}: doing f()"
5    end
6
7    def g
8      puts "#{self.class}: doing g()"
9    end
10 end
11
13 class A
14   include I
15 end
16

```

```

18 class B
19   include I
20 end
22
23 class C
24   attr_accessor :i
25   delegate :f, :g, :to => :i
26
27 def initialize(klass = A)
28   self.i = klass.new
29 end
30 end
32
33 c = C.new
34 c.f          # output: A: doing f()
35 c.g          # output: A: doing g()
36 c = C.new(B)
37 c.f          # output: B: doing f()
38 c.g          # output: B: doing g()

```

Véase

1. Delegation patterns in Ruby. Reducing object bloat begins with doing less by Samuel Mullen

14.4. Creación e Inicialización de Objetos

14.4.1. new, allocate e initialize

Todas las clases heredan el método de clase `new`. Este método tiene dos tareas:

1. debe alojar al objeto y
2. debe inicializarlo.

Para estas tareas, delega en los métodos. Este es el seudocódigo de `new: allocate e initialize`:

```

def new(* args)
  o = self.allocate      # crear el objeto
  o.initialize(* args)  # inicializarlo
  o                      # retornar el objeto
end

```

- El método `allocate` es un método de instancia de la clase `Class`.
- Es heredado por todos los objetos clase.
- Ruby invoca al método directamente. No puede ser sobreescrito.
- `initialize` es un método de instancia y privado
- La mayor parte de las clases deberían usar `super` para encadenar con el `initialize` de la superclase.
- La tarea habitual de `initialize` es crear e inicializar las variables de instancia del objeto, cuyos valores normalmente se derivan de los argumentos pasados a `new`.
- El valor returnedo por `initialize` es ignorado.

- En realidad, `Class` define dos métodos con nombre `new`:
 - Uno `Class#new` es el método de instancia descrito aquí.
 - El otro `Class::new` es un método de clase y se usa para crear nuevas clases.

14.4.2. Métodos Factoría

A veces se necesita mas de una forma de inicializar las instancias de una clase. Una manera simple de hacerlo es dando valores por defecto a los parámetros de `initialize`:

```
class Point
  def initialize(x, y, z = nil)
    @x, @y, @z = x, y, z
  end
end
```

Pero a menudo no es suficiente con esto. Necesitamos *factory methods* o *métodos factoría* para crear los objetos de nuestra clase.

Supongamos que queremos crear objetos `Point` bien en coordenadas cartesianas bien en coordenadas polares:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n CartAndPolar.rb
 1  include Math
 2  class Point
 3    attr_accessor :x, :y
 4
 5    def initialize(x, y)
 6      @x, @y = x, y
 7    end
 8
 9    private_class_method :new
10
11    def Point.cartesian(x,y)
12      new(x,y)
13    end
14
15    def Point.polar(r,theta)
16      new(r*cos(theta), r*sin(theta))
17    end
18
19    def r
20      sqrt(@x*@x+@y*@y)
21    end
22
23    def theta
24      atan2(@y, @x)
25    end
26  end
27  if __FILE__ == $0
28    p = Point.polar(1, PI/4)
29    puts p.r, p.theta      # 1.0 0.785398
30    q = Point.cartesian(1, 1) # 1 1
31    puts q.x, q.y
32  end
```

Al hacer `new` privado lo hacemos inaccesible a las clases cliente, de manera que las clases usuario de `Point` deberán de usar uno de los dos métodos factoría proveídos para construir un objeto `Point`.

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby CartAndPolar.rb
1.0
0.785398163397448
1
1
```

14.4.3. `dup`, `clone` e `initialize_copy`

Otra forma de construir objetos es via los métodos `dup` y `clone`. Estos métodos crean un objeto que es una copia del del objeto que recibe el mensaje:

```
>> a = "hello"
=> "hello"
>> b = a.dup
=> "hello"
>> c = a.clone
=> "hello"
>> [a.__id__, b.__id__, c.__id__]
=> [2150357800, 2150329860, 2150304120]
```

`clone` hace una copia mas parecida que `dup`. Por ejemplo, conserva el estatus de `freeze` del original, cosa que no hace `dup`:

```
>> a = "hello"
=> "hello"
>> a.freeze
=> "hello"
>> a << " world"
TypeError: can't modify frozen string
>> b = a.clone
=> "hello"
>> b << " world"
TypeError: can't modify frozen string
>> c = a.dup
=> "hello"
>> c << " world"
=> "hello world"
```

`clone` incluso preserva los *singleton methods*, esto es, preserva los métodos individuales del objeto. Sin embargo, `dup` no lo hace.

Métodos Singleton

Un *singleton method* es un método `x` que se añade a un objeto `obj` específico. A partir del momento en el que se añade, ese objeto `obj` responde el mensaje `x`, cosa que no hacen los otros métodos de su clase. Sigue un ejemplo:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n singletonmethandclo
1  obj = "some object"
2  def obj.printme
3    puts "****<#{self}>***"
4  end
5
```

```

6  obj.printme
7
8  b = obj.dup
9  c = obj.clone
10
11 c.printme
12 b.printme

```

En la línea 2 añadimos el método `printme` únicamente al objeto `obj`. El objeto clonado `c` también puede hacer `printme`. No así la copia `b` obtenida por duplicación:

```

~/rubystesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby singletonmethandclone
***<some object>***
***<some object>***
singletonmethandclone.rb:12: undefined method `printme' for "some object":String (NoMethodError)

```

Copia Superficial Cuando `clone` y `dup` copian las variables de instancia / atributos del objeto original al objeto final *sólo copian las referencias a los valores de esas variables no copian los valores reales: sólo hacen una copia superficial*. El siguiente ejemplo ilustra este punto:

```

~/rubystesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n dupclone1.rb
1  class Point
2    attr_accessor :coords
3    def initialize(* coords)
4      @coords = coords
5    end
6  end
7
8  if __FILE__ == $0
9    puts "dup"
10   p = Point.new(1, 2, 3)
11   q = p.dup
12   puts p.inspect
13   puts q.inspect
14
15   q.coords[1] = 4
16   puts p.inspect
17   puts q.inspect
18
19   puts "clone"
20   p = Point.new(1, 2, 3)
21   q = p.clone
22   puts p.inspect
23   puts q.inspect
24
25   q.coords[1] = 4
26   puts p.inspect
27   puts q.inspect
28 end

```

La ejecución nos muestra que la copia es superficial:

```

~/rubystesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby dupclone1.rb
dup
#<Point:0x100169008 @coords=[1, 2, 3]>

```

```
#<Point:0x100168fe0 @coords=[1, 2, 3]>
#<Point:0x100169008 @coords=[1, 4, 3]>
#<Point:0x100168fe0 @coords=[1, 4, 3]>
clone
#<Point:0x100168c20 @coords=[1, 2, 3]>
#<Point:0x100168bf8 @coords=[1, 2, 3]>
#<Point:0x100168c20 @coords=[1, 4, 3]>
#<Point:0x100168bf8 @coords=[1, 4, 3]>
```

Lo que se necesita es una copia profunda.

initialize_copy Si una clase define un método `initialize_copy`, los métodos `clone` y `dup` lo invocarán en el *objeto copia*. Al método `initialize_copy` se le pasa el objeto original como argumento. El valor retornado por `initialize_copy` es ignorado:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n dupclone2.rb
 1  class Point
 2    attr_accessor :coords
 3    def initialize(* coords)
 4      @coords = coords
 5    end
 6
 7    def initialize_copy(orig)
 8      @coords = orig.coords.dup
 9      # or better
10      # @coords = @coords.dup
11    end
12  end
13
14 if __FILE__ == $0
15   puts "dup"
16   p = Point.new(1, 2, 3)
17   q = p.dup
18   puts p.inspect
19   puts "q = p.dup = "+q.inspect
20
21   q.coords[1] = 4
22   puts "Changed q[1] to 4 and p has "+p.inspect
23   puts "Changed q[1] to 4 and q has "+q.inspect
24
25   puts "clone"
26   p = Point.new(1, 2, 3)
27   q = p.clone
28   puts p.inspect
29   puts "q = p.clone = "+q.inspect
30
31   q.coords[1] = 4
32   puts "Changed q[1] to 4 and p has "+p.inspect
33   puts "Changed q[1] to 4 and q has "+q.inspect
34 end
```

La ejecución produce la siguiente salida:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby dupclone2.rb
dup
```

```

#<Point:0x100168900 @coords=[1, 2, 3]>
q = p.dup = #<Point:0x1001688d8 @coords=[1, 2, 3]>
Changed q[1] to 4 and p has #<Point:0x100168900 @coords=[1, 2, 3]>
Changed q[1] to 4 and q has #<Point:0x1001688d8 @coords=[1, 4, 3]>
clone
#<Point:0x1001683d8 @coords=[1, 2, 3]>
q = p.clone = #<Point:0x1001683b0 @coords=[1, 2, 3]>
Changed q[1] to 4 and p has #<Point:0x1001683d8 @coords=[1, 2, 3]>
Changed q[1] to 4 and q has #<Point:0x1001683b0 @coords=[1, 4, 3]>

```

14.4.4. Limitando el Número de Instancias de una Clase

Some classes, such as those that define enumerated types, may want to strictly limit the number of instances that exist.

Classes like these need to make their new method private and also probably want to prevent copies from being made.

The following code demonstrates one way to do that:

```

~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat -n LimitingTheNumberOf
1  class Season
2    NAMES = %w{ Spring Summer Autumn Winter }
3    INSTANCES = []
4
5    def initialize(n)
6      @n = n
7    end
8
9    def to_s
10      NAMES[@n]
11    end
12
13   NAMES.each_with_index { |n,i|
14     season = new(i)
15     INSTANCES[i] = season
16     const_set n, season
17   }
18
19   private_class_method :allocate, :new
20   private :dup, :clone
21 end
22
23 if __FILE__ == $0
24   puts Season::Spring
25   puts Season::Summer
26   puts Season::Autumn
27   puts Season::Winter
28 end

```

El método const_set de la clase Module establece el valor asociado con el nombre pasado como primer argumento al objeto dado como segundo argumento:

```

>> Math.const_set("HIGH SCHOOL_PI", 22.0/7.0)
=> 3.14285714285714
>> Math::HIGH SCHOOL_PI
=> 3.14285714285714

```

Cuando se ejecuta el programa anterior produce esta salida:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ ruby LimitingTheNumberOfIn  
Spring  
Summer  
Autumn  
Winter
```

14.5. Métodos y Clases: Construyendo un Iterador

A class is a collection of related methods that operate on the state of an object.

An object's state is held by its instance variables: variables whose names begin with @ and whose values are specific to that particular object.

This example class demonstrates how to write iterator methods and define operators.

```
lpp@nereida:~/src/rubytesting/TheRubyProgrammingLanguage$ cat -n Sequence.rb  
1  # The Ruby programming Language Flanagan & Matsumoto  
2  # page 8  
3  class Sequence  
4      include Enumerable  
5      def initialize(from, to, by)  
6          @from, @to, @by = from, to, by  
7      end  
8  
9      def each  
10         x = @from  
11         while x <= @to  
12             yield x  
13             x += @by  
14         end  
15     end  
16  
17     def length  
18         return 0 if @from > @to  
19         Integer((@to-@from)/@by)+1  
20     end  
21  
22     alias size length  
23  
24     # override the array-access operator to give random access to the sequence  
25  
26     def [](index)  
27         return nil if index < 0  
28         v = @from+index*@by  
29         return v if v <= @to  
30         return nil  
31     end  
32  
33     def *(factor)  
34         Sequence.new(@from*factor, @to*factor, @by*factor)  
35     end  
36  
37     def +(offset)  
38         Sequence.new(@from+offset, @to+offset, @by)
```

```
39     end  
40 end
```

Here is some code that uses this `Sequence` class:

```
lpp@nereida:~/src/rubytesting/TheRubyProgrammingLanguage$ cat -n useSequence.rb  
1  #!/usr/bin/env ruby  
2  require "Sequence.rb"  
3  
4  a = Sequence.new(1,10,2)  
5  a.each { |x| print x,"\\n" } # 1 3 5 7 9  
6  
7  puts "Length = #{a.length}" # 5  
8  puts "Size = #{a.size}"      # 5  
9  
10 (0..(a.size-1)).each { |i|    # Podemos indexar  
11   puts "a[#{i}] = '#{a[i]}'" # un objeto Sequence  
12 }  
13  
14 puts ""  
15  
16 b = a * 2 # b is a new Sequence object  
17 b.each { |x| print x,"\\n" } # 2 6 10 14 18  
18  
19 puts ""  
20  
21 b = a + 2 # b is a new Sequence object  
22 b.each { |x| print x,"\\n" } # 3 5 7 9 11
```

```
lpp@nereida:~/src/rubytesting/TheRubyProgrammingLanguage$ ./useSequence.rb
```

```
1  
3  
5  
7  
9  
Length = 5  
Size = 5  
a[0] = '1'  
a[1] = '3'  
a[2] = '5'  
a[3] = '7'  
a[4] = '9'
```

```
2  
6  
10  
14  
18
```

```
3  
5  
7  
9  
11
```

- The Enumerable Module
- Modules: Programming Ruby The Pragmatic Programmer's Guide
- Vea la sección Containers, Blocks, and Iterators
- alias

14.6. Práctica: Conjuntos

Defina una clase Conjunto que disponga de métodos para:

- Calcular el cardinal de un conjunto
- Determinar las relaciones pertenencia de un elemento y subconjunto
- La operación de unión:
- La operación de intersección
- Diferencia de conjuntos
- Convertir un conjunto en una cadena

Debe funcionar de manera similar a la clase Set proveída con Ruby:

```
~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ irb
>> require 'set'
=> true
>> s1 = Set.new [1, 2]
=> #<Set: {1, 2}>
>> s2 = Set.new [2, 4]
=> #<Set: {2, 4}>
>> s1+s2
=> #<Set: {1, 2, 4}>
>> s1^s2
=> #<Set: {1, 4}>
>> s2-s1
=> #<Set: {4}>
>> s1.include? 1
=> true
>> s1.include? 6
=> false
```

Puede usar **Arrays** para implementar los conjuntos. Asuma que los elementos del conjunto universal admiten un orden total. Esta sesión con **irb** sugiere una posible implementación:

```
>> a = [6, 4, 3, 4, 2, 1].uniq.sort
=> [1, 2, 3, 4, 6]
>> b = [2, 3, 2, 4, 5].uniq.sort
=> [2, 3, 4, 5]
>> (a | b).uniq.sort
=> [1, 2, 3, 4, 5, 6]
>> (a & b).uniq.sort
=> [2, 3, 4]
```

Esto es, puede guardar el conjunto como un **Array** ordenado en el que se han suprimido los elementos repetidos.

La relación de orden entre conjuntos es parcial: dados dos conjuntos A y B no necesariamente ocurre que A < B o que B < A.

14.7. Módulos

Modules are a way of grouping together methods, classes, and constants.

- Al igual que una clase, un *módulo* es una colección de métodos, constantes y variables de clase.
- A diferencia de una clase, un módulo no puede ser instanciado y no puede ser heredado (aunque puede ser incluído),
- Los módulos pueden agrupar varias clases.
- Los módulos tienen dos usos: como espacios de nombres y como mixins
- Los módulos son objetos de la clase `Module`
- `Class` es una subclase de `Module`

14.7.1. Los Módulos como Espacios de Nombres

Al definir un módulo se crea una constante con el mismo nombre del módulo. El valor de dicha constante es el objeto `Module` que representa al módulo.

Ejemplo Simple

Say you write a set of the trigonometry functions `sin`, `cos`, and so on. You stuff them all into a file, `trig.rb`, for future generations to enjoy. Meanwhile, Sally is working on a simulation of good and evil, and she codes a set of her own useful routines, including `be_good` and `sin`, and sticks them into `moral.rb`. Joe, who wants to write a program to find out how many angels can dance on the head of a pin, needs to load both `trig.rb` and `moral.rb` into his program. But both define a method called `sin`. Bad news.

The answer is the module mechanism. Modules define a namespace, a sandbox in which your methods and constants can play without having to worry about being stepped on by other methods and constants. The trig functions can go into one module:

Dave Thomas

```
module Trig
  PI = 3.141592654
  def Trig.sin(x)
    # ...
  end
  def Trig.cos(x)
    # ...
  end
end

# Sample code from Programming Ruby, page 111
module Moral
  VERY_BAD = 0
  BAD      = 1
  def Moral.sin(badness)
    # ...
  end
end
```

If a third program wants to use these modules, it can simply load the two files (using the Ruby require statement. In order to reference the name sin unambiguously, our code can then qualify the name using the name of the module containing the implementation we want, followed by `::`, the scope resolution operator:

```
$: << '..'
require 'trig'
require 'moral'

y = Trig.sin(Trig::PI/4)
wrongdoing = Moral.sin(Moral::VERY_BAD)
```

Ejemplo: Codificar y Decodificar Datos Usando Base64

Supongamos que estamos un método para codificar y decodificar datos binarios utilizando la codificación Base64. Puesto que `encode` y `decode` son dos nombres muy comunes, para prever colisiones de nombres definimos nuestros dos métodos dentro de un módulo:

```
casiano@exthost:~/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules$ cat base64.rb
1  module Base64
2
3  puts self # Base64
4
5  DIGITS = 'ABCDEFHIJKLMNOPQRSTUVWXYZ' +
6      'abcdefghijklmnopqrstuvwxyz' +
7      '0123456789+/'
8  @x = "hello"
9
10 def self.encode
11     "self.encode #{@x}"
12 end
13
14 def self.decode
15     "self.decode #{@x}"
16 end
17 end
18 if $0 == __FILE__
19     puts Base64::DIGITS
20     puts Base64.encode
21     puts Base64.decode
22 end
```

1. Note that we define our methods with a `self.prefix`, which makes them “class methods” of the module.
2. Outside the `Base64` module, this constant can be referred to as `Base64::DIGITS`.
3. Inside the module, our encode and decode methods can refer to it by its simple name `DIGITS`.
4. If the two methods had some need to share nonconstant data, they could use a class variable (with a `@@` prefix), just as they could if they were defined in a class.

```
[~/local/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ cat base64_mod_self.rb
require 'base64_mod_self'

class A
```

```

def self.tutu
  Base64::encode
end
end

if $0 == __FILE__
  puts A.tutu
  puts Base64::DIGITS
end

[~/local/src/ruby/rubytesting/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/modules]$ ruby Base64
self.encode hello
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

```

Espacios de Nombres Anidados

- Los módulos y las clases pueden anidarse creando espacios de nombres anidados.
- Una clase o módulo anidado dentro de otro no tiene acceso especial al módulo o clase en la que se anida.

```

module Base64
  DIGITS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'

  class Encoder
    def encode
    end
  end

  class Decoder
    def decode
    end
  end

  # A utility function for use by both classes
  def Base64.helper
  end
end

```

1. Tenemos dos clases: `Base64::encoder` y `Base64::Decoder`. Dentro del módulo las clases pueden referirse la una a la otra y a la constante `DIGITS` por sus nombres simples, sin cualificar.
2. Sin embargo, para llamar a `Base64.helper` deben referirse a él por su nombre completo: `Base64.helper`.
3. Las clases pueden también anidarse. Anidar una clase dentro de otra no le da a la clase ningún privilegio especial de acceso a los métodos y variables en la clase mas externa. Su único efecto es en el espacio de nombres de la clase interna.
4. Cuando escribimos una gema es conveniente anidar las clases y módulos - especialmente aquellas clases que no forman parte de la API pública - para no contaminar los espacios de nombres públicos.

14.7.2. Los Módulos como Mixins

1. Modules have another, wonderful use.

At a stroke, they pretty much eliminate the need for inheritance, providing a facility called a mixin.

2. *What happens if I define instance methods within a module?*

Good question

3. A module can't have instances, because a module isn't a class.

4. However, you can `include` a module within a class definition.

When this happens, all the module's instance methods are suddenly available as methods in the class as well.

5. They get mixed in.

6. In fact, mixed-in modules effectively behave as superclasses.

Dave Thomas

Si un módulo define métodos de instancia en vez de métodos de clase, esos métodos de instancia pueden ser combinados/mezclados (*mixin*) en otras clases.

Para mezclar un módulo en una clase llamamos al método *include*:

```
class Point
  include Enumerable
end
```

1. Thanks to the magic of mixins and module `Enumerable`. All you have to do is write an iterator called `each`, which returns the elements of your collection in turn. Mix in `Enumerable`, and suddenly your class supports things such as `map`, `include?`, and `find_all?`.
2. If the objects in your collection implement meaningful ordering semantics using the `<=>` method, you'll also get methods such as `min`, `max`, and `sort`.

`include` es un método privado de instancia de `Module`:

```
ruby-1.9.2-head :004 > Module.private_instance_methods.select{|x| x =~ /inc/ }
=> [:included, :include]
```

Es legal incluir un módulo en una clase o en un módulo. Al hacerlo los métodos de instancia del módulo incluido se convierten en métodos de instancia del incluyente.

```
module D
  def initialize(name)
    @name = name
  end
  def to_s
    @name
  end
end

module Debug
include D
  def who_am_i?
    "#{self.class.name} (\#\{self.__id__}): #{self.to_s}"
  end
end
```

```

end

class Phonograph
  include Debug
  # ...
end

class EightTrack
  include Debug
  # ...
end

ph = Phonograph.new("West End Blues")      # "Phonograph (#70270868282220): West End Blues"
et = EightTrack.new("Surrealistic Pillow") # "EightTrack (#70270868320420): Surrealistic Pill"

ph.who_am_i?
et.who_am_i?

```

We'll make a couple of points about the `include` statement before we go on.

First, it has nothing to do with files.

2. C programmers use a preprocessor directive called `#include` to insert the contents of one file into another during compilation. The Ruby `include` statement simply makes a reference to a module.
3. If that module is in a separate file, you must use `require` (or its less commonly used cousin, `load`) to drag that file in before using `include`.
4. Second, a Ruby `include` does not simply copy the module's instance methods into the class. Instead, it makes a reference from the class to the included module.
5. If multiple classes `include` that module, they'll all point to the same thing.
6. If you change the definition of a method within a module, even while your program is running, all classes that `include` that module will exhibit the new behavior.

Dave Thomas

Variables de Instancia en Mixins

1. Remember how instance variables work in Ruby: the first mention of an `@`-prefixed variable creates the instance variable in the current object, `self`.
2. For a mixin, this means that the module you mix into your client class may create instance variables in the client object and may use `attr_reader` and friends to define accessors for these instance variables.
3. However, this behavior exposes us to a risk. A mixin's instance variables can clash with those of the host class or with those of other mixins.

The example that follows shows a class that uses our `Observer` module but that unluckily also uses an instance variable called `@observer_list`.

```

module Observable
  def observers
    @observer_list ||= []
  end

  def add_observer(obj)

```

```

    observers << obj
end

def notify_observers
  observers.each { |o| o.update }
end
end

require 'code/observer_impl'
class TelescopeScheduler

  # other classes can register to get notifications
  # when the schedule changes
  include Observable

  def initialize
    @observer_list = [] # folks with telescope time
  end

  def add_viewer(viewer)
    @observer_list << viewer
  end

  # ...
end

```

At runtime, this program will go wrong in some hard-to-diagnose ways.

1. For the most part, *mixin modules don't use instance variables directly—they use accessors to retrieve data from the client object*
2. But if you need to create a mixin that has to have its own state, ensure that the *instance variables have unique names to distinguish them from any other mixins in the system* (perhaps by using the module's name as part of the variable name).
3. Alternatively, *the module could use a module-level hash, indexed by the current object ID, to store instance-specific data without using Ruby instance variables*. A downside of this approach is that the data associated with a particular object will not get automatically deleted if the object is deleted.

Dave Thomas

```
[~/local/src/ruby/rubytesting/programmingRuby/instance_variables_in_mixins]$ cat private_state
require 'pp'

module Test
  State = {}
  def state=(value)
    State[__id__] = value
  end
  def state
    State[__id__]
  end

  def delete_state

```

```

    State.delete(__id__)
  end
end

class Client
  include Test
end

c1 = Client.new
c2 = Client.new
c1.state = 'cat'
c2.state = 'dog'

pp Test::State.inspect # {70259441214220=>"cat", 70259441214140=>"dog"}

pp c1.state
c1.delete_state

pp Test::State.inspect # {70259441214140=>"dog"}

pp c2.state

[~/local/src/ruby/rubytesting/programmingRuby/instance_variables_in_mixins]$ ruby private_stat
"70278990371120=>\"cat\", 70278990371100=>\"dog\""
"cat"
"70278990371100=>\"dog\""
"dog"

```

Colisiones

One of the other questions folks ask about mixins is, how is method lookup handled? In particular, what happens if methods with the same name are defined in a class, in that class's parent class, and in a mixin included into the class?

1. The answer is that Ruby looks first in the immediate class of an object,
2. then in the mixins included into that class,
3. and then in superclasses and their mixins.
4. If a class has multiple modules mixed in, the last one included is searched first.

Dave Thomas

¿Que ocurre si se incluyen dos módulos en los que existen métodos con el mismo nombre?

```
[~/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/modules]$ cat module_collision.rb
module M1
  def foo
    puts "M1"
  end
end

module M2
  def foo
    puts "M2"
  end
end
```

Cuando ambos módulos son incluídos en la clase C, la última definición de `foo` es la que domina:

```
[~/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/modules]$ cat class_require_module_col
require 'module_collision'
class C
  include M1
  include M2
end

C.new.foo # 'M2'
puts C.ancestors.inspect # => [C, M2, M1, Object, Kernel, BasicObject]
```

Ahora bien, si en la clase existe un método `foo` ese será encontrado primero:

```
[~/TheRubyProgrammingLanguage/Chapter7ClassesAndModules/modules]$ cat -n module_class_collision
1 module M
2   def foo
3     puts "M"
4   end
5 end
6
7 class C
8   def foo
9     puts "C"
10  end
11  include M
12 end
13 C.new.foo          # => "C"
14 puts C.ancestors.inspect # => [C, M, Object, Kernel, BasicObject]
```

Composición frente a Herencia/Composition versus Inheritance

Inheritance and mixins both allow you to write code in one place and effectively inject that code into multiple classes.

So, when do you use each?

1. First, let's look at subclassing. Classes in Ruby are related to the idea of types.
2. When we create our own classes, you can think of it as adding new types to the language.
3. And when we subclass either a built-in class or our own class, we're creating a subtype.
4. Now, a lot of research has been done on type theories. One of the more famous results is the Liskov Substitution Principle. Formally, this states:

Let $q(x)$ be a property provable about objects x of type T . Then $q(y)$ should be true for objects y of type S where S is a subtype of T

5. What this means is that you should be able to substitute an object of a child class wherever you use an object of the parent class—the child should honor the parent's contract.

We're used to saying this in English: a car is a vehicle, a cat is an animal, and so on.

This means that a cat should, at the very least, be capable of doing everything we say that an animal can do.

6. So, when you're looking for subclassing relationships while designing your application, be on the lookout for these **is-a** relationships.

7. But...here's the bad news. In the real world, there really aren't that many true `is a` relationships. Instead, it's far more common to have `has a` or `uses a` relationships between things.

Dave Thomas

1. Think of containment as a `has a` relationship. A car `has an` engine, a person `has a` name, etc.
2. Think of inheritance as an `is a` relationship. A car `is a` vehicle, a person `is a` mammal, etc.

extend

Aunque `Module.include` es la forma normal de mezclar un módulo también es posible mezclar con `Object.extend`. Este método hace que los métodos de instancia del módulo especificado se incorporen como métodos singleton del objeto receptor. Si el objeto receptor es una clase, esto es, es un objeto de la clase `Class`, entonces los métodos son métodos de clase de dicha clase receptora.

```
[~/srcLPP/Chapter7ClassesAndModules/modules]$ cat module_mixin_with_extend.rb
module Chuchu
  def tutu
    "method tutu. self: #{self.inspect}"
  end
end

class ChanChan
end

p = "hello "
q = "world"
p.extend Chuchu

puts p.tutu

ChanChan.extend Chuchu
puts ChanChan.tutu

puts q.tutu # exception
```

Ejecución:

```
[~/srcLPP/Chapter7ClassesAndModules/modules]$ ruby module_mixin_with_extend.rb
method tutu. self: "hello "
method tutu. self: ChanChan
module_mixin_with_extend.rb:19:in '<main>': undefined method 'tutu' for "world":String (NoMethodError)
```

14.7.3. Ejemplo

1. GIST: Happy OOP examples using Ruby

14.7.4. Módulos Espacio de Nombres Incluibles

```
[~/rubystuff/TheRubyProgrammingLanguage/Chapter7ClassesAndModules]$ irb
ruby-1.9.2-head :001 > Math.sin(Math::PI/2)
=> 1.0
ruby-1.9.2-head :002 > include Math
=> Object
```

```
ruby-1.9.2-head :003 > sin(PI/2)
=> 1.0
```

- Para crear un módulo que funcione como *Math* define tus métodos como métodos de instancia dentro del módulo
- Usa la función `module_function`, un método de instancia privado de `Module` para convertir en *module functions* a los métodos cuyos nombres le pases como argumentos
- El resultado de llamar a `module_function` es que:
 - construye métodos de clase equivalentes para cada uno de los métodos de instancia especificados
 - Hace privados a los métodos de instancia implicados

```
~/srcLPP/Chapter7ClassesAndModules/modules]$ cat module_function.rb
module Space0dissey

  THE_UNIVERSE = 1

  def tutu
    "inside tutu"
  end

  module_function :tutu
end

  puts Space0dissey::THE_UNIVERSE
  puts Space0dissey::tutu

  include Space0dissey
  puts THE_UNIVERSE
  puts tutu
```

Ejecución:

```
[~/srcLPP/Chapter7ClassesAndModules/modules]$ ruby module_function.rb
1
inside tutu
1
inside tutu
```

Cuando se define una función-módulo se debe evitar hacer uso de `self` ya que el valor de `self` hará que dependa de como es invocada.

14.8. Prepend

What `Module#prepend` allows us to do is insert a module in-front of the class it was prepended to:

```
module Before
end

class Example
  prepend Before
end
```

```
Example.ancestors
# => [Before, Example, Object, Kernel, BasicObject]
```

Ejemplo: Memoización

```
module Memoize

  def memoize(method)
    memoizer = Module.new do

      # Define a method in the module with the same name
      define_method method do
        @_memoized_results ||= {}

        if @_memoized_results.include? method
          @_memoized_results[method]
        else
          @_memoized_results[method] = super()
        end
      end
    end

    end

    prepend memoizer
  end
end
```

14.9. Carga y Solicitud de Módulos

14.9.1. El Camino de Búsqueda

14.9.2. Ejecutando Código Cargado

- *load* y *require* ejecutan el código en el fichero especificado inmediatamente.
- La ejecución ocurre en un nuevo ámbito al nivel mas alto, que es diferente del ámbito en el que se invocó el *load* o el *require*
- El fichero cargado puede
 - Ver las variables globales
 - Ver las constantes
 - No puede ver las variables locales

Esto conlleva:

- Las variables locales creadas por el fichero cargado se descartan una vez que se termina la carga
- Al comienzo de la carga *self* es *main* tal como ocurre cuando el intérprete Ruby comienza la ejecución.
- Invocar *load* o *require* no propaga el objeto receptor al fichero cargado.
- El anidamiento actual de módulos es ignorado. Esto es: No podemos , por ejemplo, abrir una clase y luego cargar un fichero con definiciones de métodos y esperar que estén en la clase. El fichero será procesado al nivel mas alto de ámbito, no dentro de la clase o módulo.

Envolviendo una Carga

Si al método `load` se le pasa un segundo argumento que sea distinto de `nil` y `false` entonces carga el fichero especificado en un módulo anónimo. De esta forma el fichero cargado no puede afectar al espacio de nombres global. Todas las constantes, clases, módulos que define quedan atrapados dentro del módulo anónimo. Sin embargo puede establecer variables globales que quedan visibles en el código que lo carga.

```
[~/srcLPP/Chapter7ClassesAndModules/modules]$ cat -n wrapped_module.rb
1 # Modules do not change the names of instance variables
2 # or private methods
3 module Space
4   class Chazam
5     def tutu
6       "inside tutu"
7     end
8   end
9 end
10 $Space = Space

[~/srcLPP/Chapter7ClassesAndModules/modules]$ cat -n wrapped_loads.rb
1 load 'wrapped_module.rb', true
2 puts $Space.inspect
3
4 q = $Space :: Chazam.new # :: is an operator!
5 puts q.tutu

#<Module:0x007fd9c284e790>::Space
inside tutu
[~/srcLPP/Chapter7ClassesAndModules/modules]$
```

14.9.3. Autoloading de Módulos

14.10. Práctica: Matrices

Considere el desarrollo de una clase Ruby para representar Matrices.

1. Cree una clase para representar Matrices usando desarrollo dirigido por pruebas (Test Driven Development - TDD) con la herramienta Rspec.
2. Se ha de seguir la metodología de Integración continua (Continuous Integration) usando la herramienta Travis.
3. Además se ha de comprobar el funcionamiento de la aplicación con la herramienta Guard de comprobación continua (Continuous testing) de manera que permita la ejecución de las pruebas definidas con rspec cuando se modifiquen. Puede ver un ejemplo en: [git@github.com:coromoto/CT-Point.git](https://github.com/coromoto/CT-Point.git)
4. Indique la URL (HTTP) del repositorio que ha desarrollado.

14.11. Práctica: Matrices Dispersas

Considere el desarrollo de una aplicación que contemple la creación de Matrices. Se ha de utilizar una representación para matrices densas y otra diferente para matrices dispersas.

Una matriz se considerará dispersa si tiene más de un 60 % de ceros. Se ha de establecer una relación de herencia entre las clases que se utilicen para su representación. Las operaciones básicas con matrices son diferentes en cada caso.

1. El entregable de esta práctica tiene que ser una Gema Ruby. Véase *Creando Gemas* 90
2. Partiendo de la implementación que se ha realizado en prácticas anteriores para Fracciones y Matrices reorganice el código.
3. Escribir un conjunto de pruebas unitarias (Unit Testing) `test/tc_ordenacion.rb`. Véase *Test/Unit* 16.1
4. Escribir un fichero de expectativas de comportamiento (TDD) `spec/ordenacion_spec.rb`. Véase *Test Driven Development (TDD) y Rspec* 17
5. Comprobar el correcto funcionamiento en distintas plataformas (Continuous Integration) `travis`. Véase *Integración Continua: Travis* 20 y *Custom Bundler arguments and Gemfile locations* 20.1
6. Realizar un seguimiento continuo de las pruebas (Continuous Testing) `guard`. Véase *Guard* 21

Sugerencias:

```
[~/srcLPPRuby/matrices_dispersas]$ cat sparse_matrix.rb
require 'pp'

class SparseVector
  attr_reader :vector

  def initialize(h = {})
    @vector = Hash.new(0)
    @vector = @vector.merge!(h)
  end

  def [](i)
    @vector[i]
  end

  def to_s
    @vector.to_s
  end
end

class SparseMatrix
  attr_reader :matrix

  def initialize(h = {})
    @matrix = Hash.new({})
    for k in h.keys do
      @matrix[k] = if h[k].is_a? SparseVector
                    h[k]
                  else
                    @matrix[k] = SparseVector.new(h[k])
                  end
    end
  end

  def [](i)
    @matrix[i]
  end
end
```

```

def col(j)
  c = {}
  for r in @matrix.keys do
    c[r] = @matrix[r].vector[j] if @matrix[r].vector.keys.include? j
  end
  SparseVector.new c
end

if __FILE__ == $0
  z = SparseMatrix.new 1000 => { 100 => 1, 500 => 200}, 20000 => { 1000 => 3, 9000 => 200}
  puts "z[1000] = #{z[1000]}"
  puts "z[1000][1] = #{z[1000][1]}"
  puts "z[1000][500] = #{z[1000][500]}"
  puts "z.col(500) = #{z.col(500)}"
  puts "z.col(4000) = #{z.col(4000)}"

[~/srcLPPRuby/matrices_dispersas]$ ruby sparse_matrix.rb
z[1000] = {100=>1, 500=>200}
z[1000][1] = 0
z[1000][500] = 200
z.col(500) = {1000=>200}
z.col(4000) = {}

```

14.12. Los Métodos Singleton y la Singleton Class o Eigenclass

Vimos en la sección 13.1.4 que es posible definir *métodos singleton*, *métodos que están definidos para un único objeto mas que para toda una clase de objetos*.

```

def Point.sum
  # Method body goes here
end

```

Los métodos de clase de una clase no son mas que métodos singleton de la instancia de la clase Class que representa a la clase:

- Los métodos singleton de un objeto no estan definidos en la clase del objeto
- Dichos métodos son métodos de una clase anónima asociada con el objeto. Dicha clase se conoce como *Eigenclass*, *Singleton Class* o *Metaclass*
- Ruby define una sintaxis alternativa para abrir la *singleton class* o *eigenclass* de un objeto y añadirle métodos. Para abrir la singleton class o eigenclass del objeto obj usamos **class << obj**. Por ejemplo:

```

class << Point
  def class_method1      # This is an instance method of the eigenclass.
    end                  # It is also a class method of Point.

  def class_method2
    end
end

```

- Si abrimos la singleton class o eigenclass de un objeto clase dentro de la definición de la misma clase podemos usar **self** en vez del nombre de la clase:

```

class Point
  # instance methods go here

  class << self
    # class methods go here as instance methods of the eigenclass
  end
end

```

- Obsérvese las diferencias entre estas 3 líneas:

```

class Point          # Create or open the class Point
class Point3D < Point # Create a subclass of Point
class << Point      # Open the eigenclass of the object Point

```

- Una forma de obtener la singleton class o eigenclass es:

```
eigenclass = class << o; self; end
```

- Pero es mas fácil llamar al método `singleton_class` definido en la clase `Object`

```

[13] pry(main)> w = String.singleton_class
=> #<Class:String>
[14] pry(main)> r = class << String; self; end
=> #<Class:String>
[15] pry(main)> r == w
=> true

```

- El método `singleton_methods` definido en la clase `Object` devuelve un `Array` con la lista de métodos singleton del objeto:

```

[1] pry(main)> z = [1,2,3]
=> [1, 2, 3]
[2] pry(main)> z.singleton_class
=> #<Class:#<Array:0x007ff93a647a90>>
[3] pry(main)> z.singleton_methods
=> []
[4] pry(main)> def z.tutu
[4] pry(main)*   puts "one two three"
[4] pry(main)* end
=> nil
[5] pry(main)> z.singleton_methods
=> [:tutu]
[6] pry(main)> z.tutu
one two three
=> nil

```

14.13. La Búsqueda por Métodos de Instancia

Cuando Ruby evalúa una expresión de invocación de un método (este proceso es conocido como *method lookup* o *method resolution*), por ejemplo `o.m` ocurren los siguientes pasos:

1. Primero comprueba en la *singleton class* o *eigenclass* de `o` la existencia de un método `m`
2. Si no, busca en la clase de `o` por un método `m`

3. Si no, busca en los módulos incluídos por la clase de `o`. Si la clase incluye mas de un módulo se buscan en orden inverso al orden en el que fueron incluídos
 4. Si no, se mueve hacia arriba en la clase de herencia a la superclase. Los pasos 2 y 3 se repiten para cada clase en la jerarquía de herencia hasta que todos los antecesores y módulos incluídos han sido buscados.
 5. Si no, se invoca a un método con nombre `method_missing`. Para encontrar dicho método se empieza en el paso 1.
- El módulo `Kernel` provee una implementación por defecto de `method_missing` por lo que queda garantizado el éxito de esta segunda pasada.

Let's consider a concrete example of this algorithm. Suppose we have the following code:

```
message = "hello"
message.world
```

We want to invoke a method named `world` on the `String` instance "`hello`".

```
[1] pry(main)> String.ancestors
=> [String, Comparable, Object, PP::ObjectMixin, Kernel, BasicObject]
```

Name resolution proceeds as follows:

1. Check the eigenclass for singleton methods. There aren't any in this case.
2. Check the `String` class. There is no instance method named `world`.
3. Check the `Comparable` module of the `String` class for an instance method named `world`.
4. Check the superclass of `String`, which is `Object`. The `Object` class does not define a method named `world`, either.
5. Check the `Kernel` module included by `Object`. The `world` method is not found here either, so we now switch to looking for a method named `method_missing`.
6. Look for `method_missing` in each of the spots above:
 - a) the eigenclass of the `String` object,
 - b) the `String` class,
 - c) the `Comparable` module,
 - d) the `Object` class, and the `Kernel` module

The first definition of `method_missing` we find is in the `Kernel` module, so this is the method we invoke. What it does is raise an exception:

```
NoMethodError: undefined method 'world' for "hello":String
```

This may seem like it requires Ruby to perform an exhaustive search every time it invokes a method. In typical implementations, however, successful method lookups will be cached so that subsequent lookups of the same name (with no intervening method definitions) will be very quick.

14.14. Búsqueda de Métodos de Clase

El algoritmo de resolución para los métodos de clase es el mismo que para los métodos de instancia, pero con un cambio importante.

Supongamos esta clase C sin métodos:

```
class C  
end
```

Supongamos la llamada:

```
c = C.new
```

1. Ruby busca en la eigenclass de C, no se encuentra ya que en C no se han declarado métodos de clase
2. Se busca a continuación en la clase de C: esta clase es la clase **Class**.
3. En **Class** ruby encuentra el método de instancia new y lo invoca

En Ruby cualquier invocación a un método implica un objeto receptor y un nombre de método. Si nuestro objeto es una instancia de la clase **Class** entonces nuestro objeto es una clase.

Para explicar en qué consiste el cambio importante en la búsqueda de métodos de clase mencionado antes, consideremos este otro ejemplo en el que definimos un método de clase **Integer.parse**:

```
def Integer.parse(text)  
  text.to_i  
end
```

Puesto que **Fixnum** es una subclase de **Integer** deberíamos poder invocarlo así:

```
n = Fixnum.parse("1")  
  
[3] pry(main)> Class.ancestors  
=> [Class, Module, Object, PP::ObjectMixin, Kernel, BasicObject]  
[4] pry(main)> Fixnum.ancestors  
=> [Fixnum,  
 Integer,  
 Numeric,  
 Comparable,  
 Object,  
 PP::ObjectMixin,  
 Kernel,  
 BasicObject]
```

Si aplicamos el method lookup para métodos de instancia 14.13 tendríamos:

1. Se busca en la eigenclass de **Fixnum**: no se encuentra
2. Se busca en la clase de **Fixnum**: es **Class**. No se encuentra
3. Se busca en **Module**, **Object**, **Kernel** y **BasicObject** infructuosamente

Falla. ¿Dónde se encuentra alojado el método **parse**? Sabemos que **parse** es un *método singleton del objeto Integer*. Por tanto está en la eigenclass/singleton class de **Integer**.

1. Los objetos de la clase **Class** son especiales: tienen *superclases*.
2. Las singleton clases o eigenclases de los objetos de la clase **Class** son también especiales: tienen *superclases*.

3. La eigenclass de un objeto ordinario no tiene superclase
4. Si la clase A hereda de la clase B (esto es $A < B$) y A' denota la eigenclass de A y B' denota la eigenclass de B, entonces la superclase de A' es B'
5. Ruby para los métodos singleton busca en la eigenclass del objeto y en todas las superclases de la eigenclass

Si aplicamos el nuevo lookup a la búsqueda de `Fixnum.parse` en:

```
n = Fixnum.parse("1")
```

tendríamos:

1. Se busca en la eigenclass de `Fixnum`: no se encuentra
2. Busca en la superclase de la eigenclass de `Fixnum` esta es la eigenclass de `Integer` y allí encuentra el método `parse`
3. Si no lo hubiera encontrado hubiera probado con las subsiguientes eigenclasses de `Numeric` y `Object`. Es decir, sigue buscando por métodos singleton
4. Después probaría con `Class`, `Module`, `Object` y `Kernel`

14.15. La Búsqueda de Constantes

Cuando se referencia una constante sin cualificar su espacio de nombres el intérprete Ruby busca por la definición apropiada de la constante.

1. Intenta resolver la constante en el ámbito local en el que ocurre la referencia
 2. Si no, busca en el siguiente módulo o clase que anida en su ámbito la referencia a la constante
 3. Esto continúa hasta que no hay mas clases o módulos que encierran a la referencia
 4. Si no se encuentra la constante se pasa a buscar en la jerarquía de herencia
 5. Si no se encuentra se buscan en las constantes globales
 6. Si no se encuentra se llama al método `const_missing`
1. El método de clase `Module.nesting` retorna la lista de clases y módulos que serán buscados en el orden de búsqueda
 2. El método `ancestors` (definido en la clase `Module`) de una clase o módulo retorna la lista de clases y módulos que serán buscados después que la búsqueda por ámbito fracase

```
module Kernel
  # Constants defined in Kernel
  A = B = C = D = E = F = "defined in kernel"
end

# Top-level or "global" constants defined in Object
A = B = C = D = E = "defined at toplevel"

class Super
  # Constants defined in a superclass
  A = B = C = D = "defined in superclass"
```

```

end

module Included
  # Constants defined in an included module
  A = B = C = "defined in included module"
end

module Enclosing
  # Constants defined in an enclosing module
  A = B = "defined in enclosing module"

  class Local < Super
    include Included

    # Locally defined constant
    A = "defined locally"

    # The list of modules searched, in the order searched
    # [Enclosing::Local, Enclosing, Included, Super, Object, Kernel]
    search = (Module.nesting + self.ancestors + Object.ancestors).uniq

    puts A # Prints "defined locally"
    puts B # Prints "defined in enclosing module"
    puts C # Prints "defined in included module"
    puts D # Prints "defined in superclass"
    puts E # Prints "defined at toplevel"
    puts F # Prints "defined in kernel"
  end
end

```

- Las constantes del nivel mas alto, definidas fuera de cualquier clase o módulo son definidas implícitamente en `Object`. Cuando una constante del nivel mas alto es referenciada dentro de una clase, es resuelta durante el paso de búsqueda en la jerarquía de herencia
- El módulo `Kernel` es un ancestro de `Object`:

```
[11] pry(main)> Object.ancestors
=> [Object, PP::ObjectMixin, Kernel, BasicObject]
```

Las constantes definidas en `Kernel` pueden ser sobreescritas por constantes al nivel mas alto, ya que estan se definen implícitamente en `Object`

14.16. Véase

1. Message in a Bottle by Konstantin Haase

14.17. Jugador de TicTacToe (Tres en Raya)

14.17.1. Ejemplo de Partida

Comencemos viendo un ejemplo de partida:

```
[~/srcLPP/tictactoe(master)]$ ruby lpp_solution.rb
```

```
+---+---+---+
a |   |   |
+---+---+---+
b |   | X |   |
+---+---+---+
c |   |   |
+---+---+---+
    1   2   3
```

Your move? (format: b3) c1

```
+---+---+---+
a |   |   | X |
+---+---+---+
b |   | X |   |
+---+---+---+
c | O |   |
+---+---+---+
    1   2   3
```

Your move? (format: b3) a1

```
+---+---+---+
a | O |   | X |
+---+---+---+
b | X | X |   |
+---+---+---+
c | O |   |
+---+---+---+
    1   2   3
```

Your move? (format: b3) b3

```
+---+---+---+
a | O |   | X |
+---+---+---+
b | X | X | O |
+---+---+---+
c | O |   | X |
+---+---+---+
    1   2   3
```

Your move? (format: b3) a2

```
+---+---+---+
a | O | O | X |
+---+---+---+
b | X | X | O |
+---+---+---+
c | O | X | X |
+---+---+---+
```

```
1 2 3
```

```
Tie game.
```

```
[~/srcLPP/tictactoe(master)]$
```

14.17.2. Programa Principal

Este es el programa principal del *TicTacToe* o *Tres en Raya*. Simplemente se llama al constructor de la clase `TicTacToe::Game` el cual crea el objeto partida. Después llamamos al método `play` del objeto para que se juegue la partida:

```
if __FILE__ == $0
  if ARGV.size > 0 and ARGV[0] == "-d"
    game = TicTacToe::Game.new TicTacToe::HumanPlayer,
                                TicTacToe::DumbPlayer
  else
    game = TicTacToe::Game.new TicTacToe::HumanPlayer,
                                TicTacToe::SmartPlayer
  end
  game.play
end
```

14.17.3. La Clase Game

La clase `Game` define los objetos que describen una *partida*. El constructor recibe como argumentos las clases que describen a los jugadores:

```
module TicTacToe
  class Game
    def initialize( player1, player2, random = true )
      if random and rand(2) == 1 # decidimos quien juega con "X" y quien con "O"
        @x_player = player2.new("X")
        @o_player = player1.new("O")
      else
        @x_player = player1.new("X")
        @o_player = player2.new("O")
      end

      @board = Board.new([" "] * 9) # Usamos la clase "Board"
    end

    attr_reader :x_player, :o_player

    def play
      until @board.won?
        @board[@x_player.move(@board)] = @x_player.mark
        break if @board.won?

        @board[@o_player.move(@board)] = @o_player.mark
      end

      @o_player.finish @board
      @x_player.finish @board
    end
  end
end
```

```
    end
end
```

14.17.4. La Clase Player: Un Ejemplo de *Strategy Pattern*

El concepto de jugador es definido mediante una clase abstracta:

```
module TicTacToe
  class Player
    def initialize( mark )
      @mark = mark # "X" or "O" or " "
    end
    ...
    attr_reader :mark
    ...
    def move( board )
      raise NotImplementedError, "Player subclasses must define move()."
    end
    ...
    def finish( final_board )..
```

end

de la que heredan los jugadores concretos.

Cada jugador es una clase. Por ejemplo, el jugador humano:

```
module TicTacToe
  class HumanPlayer < Player
    def move( board )
      print board
      ....
    end

    def finish( final_board )
      ....
    end
  end
end
```

Un par de jugadores mas, uno tonto y otro listo:

```
module TicTacToe
  class DumbPlayer < Player
    def move( board )
      moves = board.moves
      moves[rand(moves.size)]
    end
  end

  class SmartPlayer < Player
    def move( board )
      ....
    end
  end
end
```

La Banda de los Cuatro (*The Gang of Four* o *GoF*) llama a esta idea de *implanta el algoritmo en clases separadas* el *strategy pattern* o *patrón estrategia* [5], [6]. Esta estrategia puede aplicarse cuando ocurre que en medio de nuestra aplicación tenemos una parte que varía (el tipo de jugador). A veces en esa parte queremos hacer una cosa (usar el jugador **SmartPlayer**) y a veces otra (el jugador **HumanPlayer**). Es mas, es seguro que en el futuro se querrá otra cosa (¿que tal un jugador **MiniMaxPlayer**?).

1. La idea clave es definir una familia de clases, las *strategy* o *estrategias* que hacen de distintas formas la misma cosa: en nuestro ejemplo son los distintos tipos de **Player** que lo que hacen es decidir la próxima jugada mediante **move**.
2. No sólo realizan la misma tarea sino que comparten la misma interfaz definida por la clase **Player**.
3. Dado que todos los objetos *strategy* tiene la misma pinta vistos desde fuera, pueden ser usados por el usuario de la estrategia - Al que la GoF denomina *context* o *contexto* - como partes intercambiables. En nuestro caso el context es la clase **Game**, que usa los **Players** como objetos intercambiables
4. Al separar la clase cliente **Game** - el contexto - de las clases estrategia **Players** es necesario habilitar un mecanismo de comunicación entre el contexto y la estrategia. En este ejemplo se ha optado por pasar esa información - el tablero - como argumento:

```
@board[@x_player.move(@board)] = @x_player.mark # dentro del método play de la clase Game
```

La estrategia - el jugador **@x_player** - recibe el tablero **@board** y retorna su decisión - la jugada **@x_player.move(@board)** (algo como "b2")

5. Un ejemplo de uso práctico del patrón Strategy puede verse en el código de **rdoc**, la herramienta que se usa en Ruby para extraer documentación de los programas: por un lado hay una variación en los parsers que soporta: C, Ruby y FORTRAN. Por otro lado la salida puede hacerse en diversos formatos: XML, distintas versiones de HTML, CHM, **ri**, etc. En **rdoc** cada uno de los formatos de salida es manejado mediante una estrategia.

14.17.5. La Clase Board

La clase **TicTacToe::Game** confía en la clase **Board** que describe las operaciones que se pueden efectuar sobre el tablero. Por ejemplo, el método **play** hace uso de operaciones sobre el tablero:

```
def play
  until @board.won?
    @board[@x_player.move(@board)] = @x_player.mark # @board["b2"] = "X"
    break if @board.won?

    @board[@o_player.move(@board)] = @o_player.mark
  end

  @o_player.finish @board
  @x_player.finish @board
end
```

Eso significa que tenemos que definir en la clase **Board** predicados como **won?** y operaciones como **[]=(indice, value)** además del **initialize**:

```
class Board
  class Row
    ...
  end
```

```

MOVES = %w{a1    a2    a3    b1    b2    b3    c1    c2    c3}
# Define constant INDICES
INDICES = Hash.new { |h, k| h[k] = MOVES.find_index(k) }

def self.name_to_index( name ) # Receives "b2" and returns 4
  INDICES[name]
end

def self.index_to_name( index ) # Receives the index, like 4 and returns "b2"
  MOVES[index]
end

def initialize( squares )
  @squares = squares # An array of Strings: [ " ", " ", " ", " ", " ", "X", " ", " ", " ", "0"]
end

include SquaresContainer

def []( *indices )
  if indices.size == 2                      # board[1,2] is @squares[7]
    super indices[0] + indices[1] * 3        # calls SquaresContainer [] method
  elsif indices[0].is_a? Fixnum               # board[7]
    super indices[0]
  else                                      # board["b2"]
    super Board.name_to_index(indices[0].to_s)
  end
end

def []=(indice, value)                      # board["b2"] = "X"
  m = Board.name_to_index(indice)
  @squares[m] = value
end

.....

def moves
  moves = []
  @squares.each_with_index do |s, i|
    moves << Board.index_to_name(i) if s == " "
  end
  moves # returns the set of feasible moves [ "b3", "c2", ... ]
end

.....

BOARD =<<EOS

+---+---+---+
a | 0 | 1 | 2 |
+---+---+---+
b | 3 | 4 | 5 |
+---+---+---+

```

```

c | 6 | 7 | 8 |
+---+---+---+
 1   2   3

EOS
def to_s
  BOARD.gsub(/(\d)(?= \|)/) { |i| @squares[i.to_i] }
end

end

```

El predicado `won?` recorre todas las posibles filas con `each_row` comprobando si hay tres alineadas:

```

HORIZONTALS = [ [0, 1, 2], [3, 4, 5], [6, 7, 8] ]
COLUMNS     = [ [0, 3, 6], [1, 4, 7], [2, 5, 8] ]
DIAGONALS   = [ [0, 4, 8], [2, 4, 6] ]
ROWS = HORIZONTALS + COLUMNS + DIAGONALS

def each_row
  ROWS.each do |e|
    yield Row.new(@squares.values_at(*e), e)
  end
end

def won?
  each_row do |row|
    return "X" if row.xs == 3 # "X" wins
    return "O" if row.os == 3 # "O" wins
  end
  return " " if blanks == 0 # tie
  false
end

```

14.17.6. La Clase Row

Como se ve, hemos delegado en la clase `Row` la tarea de contar el número de marcas. La clase `Row` representa a un objeto fila del tablero, incluyendo las diagonales. Sólo tiene dos métodos porque los demás los obtiene del módulo `SquaresContainer`:

```

class Board
  class Row
    def initialize( squares, names )
      @squares = squares
      @names   = names
    end

    include SquaresContainer

    def to_board_name( index )
      Board.index_to_name(@names[index])
    end
  end
  ...
end

```

El método auxiliar `to_board_name` recibe un índice de la fila, columna o diagonal que representa el objeto `Row` (supongamos que por ejemplo `@squares = ["X", "O", " "]`, `@names = [2, 4, 6]` y `index = 2` y devuelve la notación jugada ("c2") que define a esa casilla.

En realidad todos los métodos de conteo `xs`, `os`, `blanks` son proveídos por el módulo `SquaresContainer`:

```
module TicTacToe
  module SquaresContainer
    def []( index ) @squares[index] end

    def blanks() @squares.find_all { |s| s == " " }.size end
    def os()      @squares.find_all { |s| s == "O" }.size end
    def xs()      @squares.find_all { |s| s == "X" }.size end
  end
  ....
end
```

Este mixin es incluído por `Rows` para proveer los contadores para las filas, columnas y diagonales. Pero es también incluído/reutilizado por la clase `Board` para comprobar cuando no hay casillas en blanco y ha habido un empate:

```
def won?
  each_row do |row|
    return "X" if row.xs == 3 # "X" wins
    return "O" if row.os == 3 # "O" wins
  end
  return " " if blanks == 0 # Aqui self es el objeto Board.
  false                      # Reutilizamos blanks
end
```

14.17.7. Código Completo del TicTacToe

```
#!/usr/bin/env ruby
#---
# Visit http://www.pragmaticprogrammer.com/titles/fr_quiz
#---
```

```
module TicTacToe
  module SquaresContainer
    def []( index ) @squares[index] end

    def blanks() @squares.find_all { |s| s == " " }.size end
    def os()      @squares.find_all { |s| s == "O" }.size end
    def xs()      @squares.find_all { |s| s == "X" }.size end
  end

  class Board
    class Row
      def initialize( squares, names )
        @squares = squares
        @names   = names
      end

      include SquaresContainer
```

```

    def to_board_name( index )
      Board.index_to_name(@names[index])
    end
  end

MOVES = %w{a1     a2     a3     b1     b2     b3     c1     c2     c3}
# Define constant INDICES
INDICES = Hash.new { |h, k| h[k] = MOVES.find_index(k) }

def self.name_to_index( name )# Receives "b2" and returns 4
  INDICES[name]
end

def self.index_to_name( index ) # Receives the index, like 4 and returns "b2"
  MOVES[index]
end

def initialize( squares )
  @squares = squares # An array of Strings: [ " ", " ", " ", " ", " ", "X", " ", " ", " ", " ", "0"]
end

include SquaresContainer

def []( *indices )
  if indices.size == 2                      # board[1,2] is @squares[7]
    super indices[0] + indices[1] * 3        # calls SquaresContainer [] method
  elsif indices[0].is_a? Fixnum               # board[7]
    super indices[0]
  else                                      # board["b2"]
    super Board.name_to_index(indices[0].to_s)
  end
end

def []=(indice, value)                      # board["b2"] = "X"
  m = Board.name_to_index(indice)
  @squares[m] = value
end

HORIZONTALS = [ [0, 1, 2], [3, 4, 5], [6, 7, 8] ]
COLUMNS      = [ [0, 3, 6], [1, 4, 7], [2, 5, 8] ]
DIAGONALS    = [ [0, 4, 8], [2, 4, 6] ]
ROWS = HORIZONTALS + COLUMNS + DIAGONALS

def each_row
  ROWS.each do |e|
    yield Row.new(@squares.values_at(*e), e)
  end
end

def moves
  moves = [ ]
  @squares.each_with_index do |s, i|
    moves << Board.index_to_name(i) if s == " "

```

```

    end
    moves # returns the set of feasible moves [ "b3", "c2", ... ]
end

def won?
  each_row do |row|
    return "X" if row.xs == 3 # "X" wins
    return "O" if row.os == 3 # "O" wins
  end
  return " " if blanks == 0   # tie
  false
end

BOARD = <<EOS

+---+---+---+
a | 0 | 1 | 2 |
+---+---+---+
b | 3 | 4 | 5 |
+---+---+---+
c | 6 | 7 | 8 |
+---+---+---+
      1   2   3

EOS

def to_s
  BOARD.gsub(/(\d)(?= \|)/) { |i| @squares[i.to_i] }
end

end

module TicTacToe
  class Player
    def initialize( mark )
      @mark = mark # "X" or "O" or " "
    end

    attr_reader :mark

    def move( board )
      raise NotImplementedError, "Player subclasses must define move()."
    end

    def finish( final_board )
    end
  end
end

module TicTacToe
  class HumanPlayer < Player
    def move( board )
      print board
    end
  end
end

```

```

moves = board.moves
print "Your move? (format: b3) "
move = $stdin.gets
until moves.include?(move.chomp.downcase)
  print "Invalid move. Try again. "
  move = $stdin.gets
end
move.chomp
end

def finish( final_board )
  print final_board

  if final_board.won? == @mark
    print "Congratulations, you win.\n\n"
  elsif final_board.won? == " "
    print "Tie game.\n\n"
  else
    print "You lost tic-tac-toe?! \n\n"
  end
end

end
end

module TicTacToe
  class DumbPlayer < Player
    def move( board )
      moves = board.moves
      moves[rand(moves.size)]
    end
  end

  class SmartPlayer < Player
    def move( board )
      moves = board.moves

      # If I have a win, take it. If he is threatening to win, stop it.
      board.each_row do |row|
        if row.blanks == 1 and (row.xs == 2 or row.os == 2)
          (0..2).each do |e|
            return row.to_board_name(e) if row[e] == " "
          end
        end
      end

      # Take the center if open.
      return "b2" if moves.include? "b2"

      # Defend opposite corners.
      if board[0] != @mark and board[0] != " " and board[8] == " "
        return "c3"
      end
    end
  end
end

```

```

        elsif board[8] != @mark and board[8] != " " and board[0] == " "
          return "a1"
        elsif board[2] != @mark and board[2] != " " and board[6] == " "
          return "c1"
        elsif board[6] != @mark and board[6] != " " and board[2] == " "
          return "a3"
      end

      # Defend against the special case XOX on a diagonal.
      if board.xs == 2 and board.os == 1 and board[4] == "0" and
        (board[0] == "X" and board[8] == "X") or
        (board[2] == "X" and board[6] == "X")
        return %w{a2 b1 b3 c2}[rand(4)]
      end

      # Or make a random move.
      moves[rand(moves.size)]
    end
  end
end

module TicTacToe
  class Game
    def initialize( player1, player2, random = true )
      if random and rand(2) == 1
        @x_player = player2.new("X")
        @o_player = player1.new("O")
      else
        @x_player = player1.new("X")
        @o_player = player2.new("O")
      end

      @board = Board.new(" " * 9)
    end

    attr_reader :x_player, :o_player

    def play
      until @board.won?
        @board[@x_player.move(@board)] = @x_player.mark
        break if @board.won?

        @board[@o_player.move(@board)] = @o_player.mark
      end

      @o_player.finish @board
      @x_player.finish @board
    end
  end
end

if __FILE__ == $0

```

```
if ARGV.size > 0 and ARGV[0] == "-d"
  game = TicTacToe::Game.new TicTacToe::HumanPlayer,
                            TicTacToe::DumbPlayer
else
  game = TicTacToe::Game.new TicTacToe::HumanPlayer,
                            TicTacToe::SmartPlayer
end
game.play
end
```

Capítulo 15

Reflexión y Metaprogramación

Mucho del material de este capítulo es tomado de [7], [2], [8] y de [9].

- La *metaprogramación* consiste en escribir programas y frameworks que nos ayudan a escribir programas
- La Wikipedia dice:

Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data

- The language in which the metaprogram is written is called the *metalanguage*
- The language of the programs that are manipulated is called the *object language*
- The ability of a programming language to be its own metalanguage is called *reflection* or *reflexivity*
- Metaprogramación es un conjunto de técnicas que permiten extender la sintaxis de Ruby para hacer mas fácil la programación
- La metaprogramación esta fuertemente relacionada con la idea de escribir lenguajes de dominio específico (DSL)s (piensa en Rake, RSpec, Gemfile, etc.)

15.1. Tipos, Clases y Módulos

Los *métodos introspectivos* mas usados son los que determinan el tipo de un objeto.

```
>> obj = [1, {:a => 2}]
=> [1, {:a=>2}]
>> obj.class                      # retorna la clase de un objeto
=> Array
>> obj.class.superclass           # retorna la superclase de un objeto
=> Object
>> obj.instance_of? Object         # determina si obj.class == Object
=> false
>> obj.instance_of? Array          # determina si obj es de una subclase de Object
=> true
>> obj.is_a? Object               # determina si obj es de una subclase de Object
=> true
>> obj.is_a? Array                # kind_of? es un sinónimo de is_a?
=> true
>> obj.kind_of? Object            # kind_of? es un sinónimo de is_a?
```

```

=> true
>> Array === obj                         # equivalente a obj.is_a? Array
=> true
>> Object === obj
=> true
>> obj.respond_to? 'each'                 # si tiene un método público o protected llamado 'each'
=> true                                    # si se le pasa true como segundo argumento se comprueban
                                            # también los privados
>> Array.instance_methods(false)
=> ["insert"
  "sort" "include?" "size" "&" "to_ary" "clear" "yaml_initialize" "shuffle"
  "replace" "pack" "zip" "flatten!" "to_s" "pop" "pretty_print_cycle" "hash"
  "cycle" "*" "indices" "nitems" "index" "collect" "+" "compact!"
  "last" "rassoc" "count" "drop" "delete" "delete_at" "combination" "collect!"
  "select" "each_index" "--" "flatten" "eql?" "fill" "length" "uniq!"
  "at" "choice" "reject!" "[]" "take" "inspect" "shift" "compact"
  "pretty_print" "[]=" "|" "find_index" "slice!" "each" "empty?" "transpose"
  "<<" "frozen?" "rindex" "map" "reverse_each" "reverse!" "to_a" "push"
  "uniq" "delete_if" "first" "product" "drop_while" "concat" "reject"
  "map!" "join" "slice" "indexes" "taguri" "<=>" "assoc" "fetch" "to_yaml"
  "==" "values_at" "permutation" "take_while" "unshift" "reverse"
  "sort!" "shuffle!" "taguri="]

```

15.1.1. Antepasados y Módulos

Los siguientes métodos sirven para determinar que módulos han sido incluídos y cuales son los ancestros de una clase o módulo:

```
[~/chapter8ReflectionandMetaprogramming]$ cat ancestryAndModules.rb
module A; end

module B; include A; end

class C; include B; end

if __FILE__ == $0
  puts C < B                      # => true: C includes B
  puts B < A                      # => true: B includes A
  puts C < A                      # => true
  puts Fixnum < Integer          # => true: all fixnums are integers
  puts Integer < Comparable        # => true: integers are comparable
  puts Integer < Fixnum           # => false: not all integers are fixnums
  puts (String < Numeric).inspect # => nil: strings are not numbers

  puts A.ancestors.inspect         # => [A]
  puts B.ancestors.inspect         # => [B, A]
  puts C.ancestors.inspect         # => [C, B, A, Object, Kernel, BasicObject]
  puts String.ancestors.inspect   # => [String, Comparable, Object, Kernel, BasicObject]

  puts C.include?(B)              # => true
  puts C.include?(A)              # => true
  puts B.include?(A)              # => true
  puts A.include?(A)              # => false
  puts A.include?(B)              # => false

```

```

puts A.included_modules.inspect # => []
puts B.included_modules.inspect # => [A]
puts C.included_modules.inspect # => [B, A, Kernel]
end

```

- El método `include?` es un método de instancia público definido en la clase `Module`
- El método `include` es un método privado de `Module`.

```

ruby-1.9.2-head :002 > Module.private_methods.select { |x| x=~ /include/ }
=> [:included, :include]

```

(Para mas información sobre `include`, véase la sección 14.7.2)

El método `extend` de la clase `Object`

Ejercicio 15.1.1. ¿Es posible incorporar los métodos de clase definidos en un módulo dentro de una clase? ¿Cuál es el resultado de la llamada de la línea 15 en el siguiente programa?

```

[~/chapter8ReflectionandMetaprogramming]$ cat -n including_class_methods1.rb
1 module Tutu
2   def self.tutu; "tutu"; end
3 end
4
5 class Chazam
6   include Tutu
7 end
8
9 puts Chazam.ancestors.inspect # [Chazam, Tutu, Object, Kernel, BasicObject]
10
11 chazam_eigenclass = class << Chazam; self; end
12
13 puts chazam_eigenclass.ancestors.inspect # [Class, Module, Object, Kernel, BasicObject]
14
15 Chazam.tutu

```

La línea:

```
chazam_eigenclass = class << Chazam; self; end
```

obtiene una referencia al objeto que representa la *singleton class*, *eigenclass* o *metaclass* de `Chazam`. Es posible obtener mas directamente la eigenclass de una clase usando el método `singleton_class` de `Kernel`:

```

ruby-1.9.2-head :001 > class Tutu; end
=> nil
ruby-1.9.2-head :002 > Tutu.singleton_class
=> #<Class:Tutu>

```

Si tiene dudas para responder al ejercicio, repase las secciones *Búsqueda de Métodos de Clase* 14.14 y *La Búsqueda por Métodos* 14.13.

Ejercicio 15.1.2. ¿Que hacen las líneas 6-8 en el siguiente código? ¿Dónde está siendo incluido el módulo `Tutu`? ¿Cuál es el resultado de la llamada de la línea 17 en el siguiente código?

```
[~/chapter8ReflectionandMetaprogramming]$ cat -n including_class_methods2.rb
1 module Tutu
2   def tutu; puts "tutu"; end
3 end
4
5 class Chazam
6   class << self
7     include Tutu
8   end
9 end
10
11 puts Chazam.ancestors.inspect # [Chazam, Tutu, Object, Kernel, BasicObject]
12
13 chazam_eigenclass = class << Chazam; self; end
14
15 puts chazam_eigenclass.ancestors.inspect # [ Tutu, Class, Module, Object, Kernel, BasicObject]
16
17 Chazam.tutu
```

- El método `extend` de la clase `Object` hace que todos los métodos de los módulos especificados se conviertan en *métodos singleton* (Véase la sección 14.12) del objeto sobre el que se llama `extend`:

```
module Greeter; def hi; "hello"; end; end # A silly module
s = "string object"
s.extend(Greeter)          # Add hi as a singleton method to s
s.hi                      # => "hello"
```

- Si `extend` se llama sobre un objeto `Class` los métodos se añaden como métodos de clase:

```
String.extend(Greeter)  # Add hi as a class method of String
String.hi                # => "hello"
```

- El método `extend` de la clase `Object` es simplemente una abreviación que incluye el módulo en la *singleton class* o *eigenclass* del objeto receptor

El Método `nesting` de la clase `Module`

- El método `nesting` de la clase `Module` retorna un array especificando el anidamiento de módulos en el punto de llamada:

```
module M
  class C
    Module.nesting  # => [M::C, M]
  end
end
```

15.1.2. Definiendo Clases y Módulos

Las clases y los módulos son instancias de las clases `Class` y `Module`. Por tanto, es posible crearlos dinámicamente:

```
M = Module.new      # Define a new module M
C = Class.new       # Define a new class C
D = Class.new(C) { # Define a subclass of C
  include M        # that includes module M
}
D.to_s             # => "D": class gets constant name by magic
```

Cuando un módulo o clase dinámicamente creado es asignado a una constante el nombre de esa constante es usado como nombre del módulo o clase.

```
~/Chapter8ReflectionandMetaprogramming$ cat -n dynamicclassesandmodules.rb
 1  Fred = Module.new do
 2    def meth1
 3      "hello #{args.inspect}"
 4    end
 5    def meth2
 6      "bye #{args.inspect}"
 7    end
 8  end
 9
10 Klass = Class.new do
11   include Fred
12   attr_accessor 'args'
13
14   def initialize(*args)
15     @args = args
16   end
17 end
18
19 puts Fred.to_s      # Fred
20 puts Klass.name     # Klass
21
22 a = Klass.new(4, 5)
23
24 puts a.meth1       # hello [4, 5]
25 puts a.meth2       # bye [4, 5]
```

- Cuando una clase o módulo es creado dinámicamente y asignado a una constante, el nombre de la constante es utilizado como nombre de la clase
- Si la clase o módulo fuera asignado a una variable (como en la línea 22), el método `to_s` devolvería algo del estilo `#<Class:0x10016a868>` y `name` devolvería `nil`.

Ambitos y los métodos `Class.new`, `Module.new`, `define_method`

1. El método `local_variables` del módulo `Kernel` nos permite acceder a las variables locales visibles en un punto.
2. Algunos lenguajes permiten a un *ámbito* interno ver las variables de un *ámbito* mas externo. Este tipo de visibilidad no ocurre en general en Ruby
3. Hay tres puntos en los que un programa abandona el *ámbito* previo y abre un nuevo *ámbito* (*scope gates* en al terminología del libro *Metaprogramming Ruby* [8]):
 - a) Definiciones de clase
 - b) Definiciones de módulo
 - c) Definiciones de método
4. Las variables globales son visibles desde cualquier *ámbito*
5. Las variables de instancia son visibles desde cualquier punto en el que `self` se refiere al mismo objeto

```
[~/chapter8ReflectionandMetaprogramming]$ cat -n scopegates1.rb
1     v1 = 1
2     puts "%2d" % __LINE__.to_s+' '+local_variables.inspect      # [:v1, :obj]
3
4     class Tutu
5         v2 = 2
6         puts "%2d" % (__LINE__.to_s)+' '+local_variables.inspect # [:v2]
7
8         def my_method
9             v3 = 3
10            puts "%2d" % __LINE__.to_s+' '+local_variables.inspect # [:v3]
11        end
12
13        puts "%2d" % __LINE__.to_s+' '+local_variables.inspect    # [:v2]
14    end
15
16    obj = Tutu.new
17    obj.my_method
18    obj.my_method
```

Puesto que `Class.new`, `Module.new` y `define_method` usan el bloque que les sigue y los bloques no abandonan el *ámbito* previo (aunque crean uno nuevo), podemos jugar con estos tres métodos para crear clausuras en las que se comparten variables locales entre clases, métodos y ámbitos. Observe como en esta reescritura del programa anterior cambia la visibilidad:

```
v1 = 1

Tutu = Class.new do
  v2 = 2
  puts "%2d" % (__LINE__.to_s)+' '+local_variables.inspect # [:v2, :v1, :obj]

  define_method :my_method do
    v3 = 3
    puts "%2d" % __LINE__.to_s+' '+local_variables.inspect    # [:v3, :v2, :v1,
  end

  puts "%2d" % __LINE__.to_s+' '+local_variables.inspect      # [:v2, :v1, :obj]
end

obj = Tutu.new
obj.my_method
obj.my_method
puts "%2d" % __LINE__.to_s+' '+local_variables.inspect      # [:v1, :obj]
```

15.2. Evaluando Strings y Bloques

El método `eval` de `Kernel` permite evaluar el código contenido en un objeto `String`:

```
>> x = 1
=> 1
>> eval "x+1"
=> 2
```

en general, se desaconseja el uso generalizado de `eval` cuando se trabaja en una aplicación distribuída en la que las cadenas a evaluar pueden venir de fuentes externas. Es conveniente comprobar primero la estructura de dichas cadenas.

In this example an attacker can control all or part of an input string that is fed into an `eval()` function call

```
myvar = "varname";
x = params['arg'];
eval("myvar = x");
```

The argument of `eval` will be processed as Ruby, so additional commands can be appended. For example, if `arg` is set to

```
"10; system %q{/bin/echo uh-oh}
```

additional code is run which executes a program on the server, in this case `/bin/echo`.

15.2.1. Bindings (encarpetados) y eval

1. Un `Binding` es un objeto que representa el estado de los bindings definidos por las variables en un instante dado.
2. El método `binding` retorna un objeto `Binding` describiendo los bindings en efecto en el punto de la llamada.
3. Es posible pasarle a `eval` un segundo argumento de la clase `Binding` que especifique el contexto en que es evaluada la cadena:

```
~/Chapter8ReflectionandMetaprogramming$ cat -n bindingsAndEval.rb
1  class Demo
2    def initialize(n)
3      @secret = n
4    end
5    def getBinding
6      return binding()
7    end
8  end
9
10 k1 = Demo.new(99)
11 b1 = k1.getBinding
12 k2 = Demo.new(-3)
13 b2 = k2.getBinding
14
15 puts eval("@secret", b1)    #=> 99
16 puts eval("@secret", b2)    #=> -3
17 puts eval("@secret")        #=> nil
```

Ejecución:

```
~/Chapter8ReflectionandMetaprogramming$ ruby bindingsAndEval.rb
99
-3
nil
```

Procs y Bindings

1. La clase `Proc` define un método `binding`. Cuando se llama dentro de un proc o de una lambda retorna un objeto `Binding` que representa los bindings en efecto en la clausura de ese `Proc`.
2. Además, Ruby ≥ 1.9 define un método `eval` para los objetos `Binding`.

```
def fred(param)
  proc {}
end

def multiplier(n)
  lambda do |*arr|
    arr.collect { |i| i*n }
  end
end

b = fred(99)
puts eval("param", b.binding)      #=> 99

puts *****
doubler = multiplier(2)
puts doubler[1, 2, 3] # 2 4 6

puts *****
eval("n = 3", doubler.binding)
puts doubler.call(1, 2, 3) # 3 6 9

puts *****
eval("n = 5", doubler)
puts doubler.call(1, 2, 3) # 5 10 15
```

ejecución:

```
~/Chapter8ReflectionandMetaprogramming$ ruby procsAndBindings.rb
99
*****
2
4
6
*****
3
6
9
*****
5
10
15
```

15.2.2. `instance_eval` y `class_eval`

- La clase `BasicObject` define el método `instance_eval`.

```
ruby-1.9.2-head :004 > BasicObject.public_methods.select { |x| x =~ /instance_e/ }
=> [:instance_eval, :instance_exec]
```

- La clase `Module` define el método `class_eval`.
- El método `module_eval` es un sinónimo de `class_eval`.

Estos dos métodos evalúan de manera similar a `eval` pero con estas consideraciones:

1. Evalúan el código en el contexto del objeto o de la clase o módulo especificados
2. El objeto o módulo es el valor de `self` durante la evaluación
3. `instance_eval` define **métodos singleton del objeto** (y, por tanto define métodos de clase cuando el objeto es una clase)

```
# Use instance_eval to define class method String.empty
# Note that quotes within quotes get a little tricky...
String.instance_eval("def empty; ''; end")
```

4. `class_eval` define métodos de instancia

```
# Define an instance method len of String to return string length
String.class_eval("def len; size; end")
```

5. `instance_eval` y `class_eval` aceptan un bloque de código como argumento para la evaluación

```
String.class_eval {
  def len
    size
  end
}
String.class_eval { alias len size }
String.instance_eval { def empty; ""; end }
```

6. La sintaxis de `instance_eval` es:

```
instance_eval(string [, filename [, lineno]] )
instance_eval {| | block }
```

En el caso de la `String`, los dos últimos argumentos son utilizados para mejorar los mensajes de error

7. El método `class_eval` tiene la sintaxis:

```
class_eval(string [, filename [, lineno]])
module_eval {| | block }
```

Veamos un ejemplo:

```
class Klass
  def initialize
    @secret = 99
  end
end
k = Klass.new
k2 = Klass.new
puts k.instance_eval { @secret }    #=> 99
```

```

k.instance_eval " def tutu; puts self.inspect; %q{hello}; end"
puts k.tutu      # #<Klass:0x10016a9f8 @secret=99>
                  # hello
begin
  puts k2.tutu    # "tutu" is a singleton method of "k"
rescue NoMethodError
  puts $!  # undefined method 'tutu' for #<Klass:0x10016a958 @secret=99>
end

Klass.instance_eval {
  def chachi      # A singleton method of a class is a class method!
    self.inspect
  end
}

puts Klass.chachi # Klass

```

Los dos últimos argumentos `class_eval(string [, filename [, lineno]])` son utilizados para mejorar los mensajes de error:

```

~/Chapter8ReflectionandMetaprogramming$ cat -n class_eval.rb
1  class Thing
2  end
3  a = %q{def hello() "Hello there!" end}
4  Thing.module_eval(a)
5  puts Thing.new.hello()
6
7  Thing.module_eval("invalid code",
8                      "ficherito.rb",   # Fichero del que se leyó el código
9                      123            # nº de linea
10                     )

```

Ejecución:

```

~/Chapter8ReflectionandMetaprogramming$ ruby class_eval.rb
Hello there!
ficherito.rb:123: undefined local variable or method 'code' for Thing:Class (NameError)

```

Creando accessors con `class_eval`

Sabemos que podemos crear accessors a los atributos de una clase usando `attr_accessor`, `attr_reader` y `attr_writer`. Estas no son palabras reservadas de Ruby sino que se trata de métodos. Veamos como podemos emular su construcción usando `class_eval`:

```

MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano$ cat -n myAccessors.rb
1  class Module
2    def my_attr_reader(*syms) # método de instancia de Module
3      syms.each { |s|         # esto es, ¡un método de clase!
4        class_eval %{
          # aquí self es el objeto Module
5          def #{s} # estamos definiendo un método de
6              #@#{s} #instancia en la clase "self"
7            end
8        }
9      }

```

```

10    end
11 end
12
13 class Chchu
14   my_attr_reader :a, :b
15
16   def initialize(a,b)
17     @a, @b = a, b
18   end
19 end
20
21 x = Chchu.new(4, "z")
22 puts x.a    # 4
23 puts x.b    # z

```

15.2.3. `instance_exec` y `class_exec`

Los métodos `instance_exec` (`BasicObject`) y `class_exec` (y su alias `module_exec`, ambos en `Module`) evalúan un bloque (pero no una `String`) de la misma forma que sus homólogos `instance_eval` y `class_eval`. La diferencia es que los métodos `exec` aceptan argumentos que pasan al bloque evaluado.

Véase Understanding instance exec in ruby by Neeraj Singh (12 Mar 2013).

15.3. Variables y Constantes

Existen métodos para listar los nombres (como cadenas) de

- las variables globales que estén definidas (`global_variables`, definido en `Kernel`)
- de las variables locales que estén actualmente definidas (`local_variables`, definido en `Kernel`)
- de las variables de instancia de un objeto (`instance_variables`, definido en `Object`)
- de las variables de clase o módulo (`class_variables`, definido en `Class`)
- y de las constantes de una clase o módulo (`constants`, definido en `Module`)

```

~/chapter8ReflectionandMetaprogramming$ cat -n variablesAndConstants.rb
1 class Tutu
2   def initialize(x,y); @x, @y = x, y; end
3   @@classvar = 1
4   ORIGIN = Tutu.new(0, 0)
5 end
6
7 puts "Globals variables with 'D': "
8 puts(" "+global_variables.select { |w| w =~ /D/ }.inspect) # [:$KCODE, :$LOAD_PATH, :$LOA
9 x = 1
10 puts "Local variables:"
11 puts " #{local_variables.inspect}" # [:x]
12
13 puts "Instance variable names: "
14 Tutu::ORIGIN.instance_variables.each { |v|
15   puts " #{v}"           # @x @y
16 }
17 puts "Class variables names: "

```

```

18 Tutu.class_variables.each { |v|
19   puts "  #{v}"                      # @@classvar
20 }
21 puts "Constant names: "
22 Tutu.constants.each { |c|
23   puts "  #{c}"                      # ORIGIN
24 }

```

Ejecución:

```

~/chapter8ReflectionandMetaprogramming$ ruby variablesAndConstants.rb
Globals variables with 'D':
["$KCODE", "$LOAD_PATH", "$DEBUG", "$LOADED_FEATURES"]
Local variables:
["x"]
Instance variable names:
@x
@y
Class variables names:
@@classvar
Constant names:
ORIGIN

```

15.3.1. Buscando, Dando Valores y Suprimiendo Variables y Constantes

No existen métodos especiales para indagar o dar valores a las variables locales y globales pero siempre es posible usar `eval` para eso:

```

x = 1
varname = "x"
eval(varname)           # => 1
eval("varname = '$g'") # Set varname to "$g"
eval("#{varname} = x") # Set $g to 1
eval(varname)           # => 1

```

Cualesquieras variables que se definen dentro de un `eval` son locales al `eval` y dejan de existir cuando este retorna.

Es posible preguntar por el valor, poner el valor y comprobar la existencia de variables de instancia de cualquier objeto y de variables y constantes de clase (o módulo):

```

o = Object.new
o.instance_variable_set(:@x, 0)    # Note required @ prefix
o.instance_variable_get(:@x)       # => 0
o.instance_variable_defined?(:@x) # => true

Object.class_variable_set(:@@x, 1)  # Private in Ruby 1.8
Object.class_variable_get(:@@x)    # Private in Ruby 1.8
Object.class_variable_defined?(:@@x) # => true; Ruby 1.9 and later

Math.const_set(:EPI, Math::E*Math::PI)
Math.const_get(:EPI)              # => 8.53973422267357
Math.const_defined? :EPI          # => true

String.class_eval { class_variable_set(:@@x, 1) } # Set @@x in String
String.class_eval { class_variable_get(:@@x) }     # => 1

```

```

o.instance_eval { remove_instance_variable :@x }
String.class_eval { remove_class_variable(:@@x) }
Math.send :remove_const, :PI # Use send to invoke private method

def Symbol.const_missing(name)
  name # Return the constant name as a symbol
end
Symbol::Test # => :Test: undefined constant evaluates to a Symbol

```

15.4. Métodos

15.4.1. Listando y Comprobando Métodos

15.4.2. Obteniendo los Métodos de Objetos

15.4.3. Llamando a los Métodos Dinámicamente

Cuando llamamos a un método, usamos la notación *punto*. Sin embargo, también podemos usar el método `send` de la clase `Object`:

```
MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano$ cat -n dynamiccall.rb
1 class Tutu
2   def tutu(arg)
3     a = arg*2
4     yield a if block_given?
5   end
6 end
7
8 obj = Tutu.new
9 obj.tutu(4) { |m| puts "In block: #{m}" }
10 print "Action?: "
11 n = gets.chomp!.to_sym # Leer desde stdin, suprimir '\n' y convertirº
12 obj.send(n, 5) { |m| puts "In block: #{m}" }
```

Ejecución:

```
MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano$ ruby dynamiccall.rb
In block: 8
Action?: tutu
In block: 10
```

send: Rendimiento/Benchmark

El módulo `Benchmark` nos permite comparar el rendimiento de distintos códigos:

```
~/rubytesting$ cat -n benchcall.rb
1 require 'benchmark'
2 include Benchmark
3 test = "Comparing Calls"
4 m = test.method(:length)
5 n = 1000000
6 bm(12) { |x|
7   x.report("call") { n.times { m.call } }
8   x.report("send") { n.times { test.send(:length) } }
9   x.report("eval") { n.times { eval "test.length" } }
10 }
```

```
~/rubystesting$ ruby benchcall.rb
              user      system      total      real
call        0.210000  0.000000  0.210000 ( 0.209029)
send        0.240000  0.000000  0.240000 ( 0.247114)
eval        1.950000  0.000000  1.950000 ( 1.951107)
```

15.4.4. Definiendo, Suprimiendo y Haciendo Alias de Métodos

Definiendo Métodos con `define_method`

En Ruby tenemos dos formas de generar métodos dinámicamente: `define_method` y `eval + def`.

<pre>define_method(symbol, method) define_method(symbol) { block }</pre>	<pre>class_eval <<-"END" def #{name} # body end END</pre>	<p><code>define_method</code> es un método de clase privado y por tanto debe ser invocado sin especificar el receptor: el destinatario es siempre la clase que está en contexto. Los parámetros son:</p> <ul style="list-style-type: none"> ■ <code>symbol</code> especifica el nombre del método. ■ <code>method</code> es una <code>lambda</code> o <code>Proc</code> o un bloque.
--	---	--

do de clase privado y por tanto debe ser invocado sin especificar el receptor: el destinatario es siempre la clase que está en contexto. Los parámetros son:

- `symbol` especifica el nombre del método.
- `method` es una `lambda` o `Proc` o un bloque.

```
~/chapter8ReflectionandMetaprogramming$ cat -n hashtoclass.rb
```

```

1  class MyClass
2      def self.h; @@h end
3
4      print "Write a ruby hash: "
5      @@h = eval gets()
6
7      @@h.keys.each { |k|
8          define_method k do
9              @@h[k]
10         end
11     }
12     define_method "#{k}=" do |v|
13         @@h[k] = v
14     end
15   }
16 end
17
18 x = MyClass.new
19 print "Action: "; action = gets().chomp
20 if action =~ /=/
21     print "Value: "; value = eval gets().chomp
22     puts x.send(action, value)
23 else
24     puts x.send(action)
25 end
26 puts MyClass.h.inspect
```

```
~/chapter8ReflectionandMetaprogramming$ ruby hashtoclass.rb
```

```
Write a ruby hash: { :a => 1, :b => 2}
```

```
Action: b=
Value: 45
45
{:a=>1, :b=>45}
```

`define_method` nos permite definir métodos cuyos nombres no se atienden a las reglas convencionales:

```
>> class Titi
>> define_method 'chu$|chu' do
?> puts "in chu$|chu"
>> end
>> end
=> #<Proc:0x000000010046d8f8@(irb):33>
>> z = Titi.new
=> #<Titi:0x10046ac20>
>> z.send 'chu$|chu'
in chu$|chu
=> nil
```

15.4.5. Manejando Métodos No Definidos: `method_missing`

```
MacBookdeCasiano:rubystesting casiano$ cat -n searchmethod.rb
```

```
1  class A
2    def tutu
3      puts "A::tutu"
4    end
5    def method_missing(name, *args)
6      puts "In A::method missing"
7    end
8  end
9
10 class B < A
11   def method_missing(name, *args)
12     puts "In B::method missing"
13   end
14 end
15
16 x = B.new
17 x.tutu()    # A::tutu
18
19 y = A.new
20 y.titi()    # In A::method missing
21
22 z = B.new    # In B::method missing
23 z.tete()
```

```
MacBookdeCasiano:rubystesting casiano$ ruby searchmethod.rb
```

```
A::tutu
In A::method missing
In B::method missing
```

```
~/chapter8ReflectionandMetaprogramming$ cat -n method_missing.rb
```

```
1  class Shell
2    def method_missing(method, *args)
```

```

3      r = %x{#{method} #{args.join(' ')} 2>&1}
4      return r unless block_given?
5      yield r
6    end
7  end
8
9  s = Shell.new
10 puts s.ls('-l', 'r*.rb') { |x|
11   puts "inside block"
12   x.gsub(/\^/, '** ')
13 }
14 puts s.uname

~/chapter8ReflectionandMetaprogramming$ ruby method_missing.rb
inside block
** -rw-r--r-- 1 casianorodriguezleon staff 1195 1 nov 10:05 recipes.rb
** -rw-r--r-- 1 casianorodriguezleon staff 1194 1 nov 10:05 recipes2.rb
** -rw-r--r-- 1 casianorodriguezleon staff 483 12 nov 11:08 ruport_example.rb
Darwin

~/chapter8ReflectionandMetaprogramming$ cat -n method_missing2.rb
1 def method_missing(method, *args)
2   r = %x{#{method} #{args.join(' ')} 2>&1}
3   return r unless block_given?
4   yield r
5 end
6
7 puts ls('-l', 'r*.rb') { |x|
8   puts "inside block"
9   x.gsub(/\^/, '** ')
10 }
11 puts uname
~/chapter8ReflectionandMetaprogramming$ ruby method_missing2.rb
inside block
** -rw-r--r-- 1 casianorodriguezleon staff 1195 1 nov 10:05 recipes.rb
** -rw-r--r-- 1 casianorodriguezleon staff 1194 1 nov 10:05 recipes2.rb
** -rw-r--r-- 1 casianorodriguezleon staff 483 12 nov 11:08 ruport_example.rb
Darwin

```

Véase También

1. Ruby for Newbies: Missing Methods

15.4.6. Ejercicios

El siguiente ejemplo tomado del libro [8] es un programa que realiza un sorteo: genera un número para cada miembro de un equipo. Para cada persona genera tres números aleatoriamente y le asigna el último. El que saca el menor número pierde y tiene que ir a por el café al bar de la esquina.

```

~/chapter8ReflectionandMetaprogramming$ cat -n roulette.rb
1 class Roulette
2   def method_missing(name, *args)
3     person = name.to_s.capitalize

```

```

4   3.times do
5     number = rand(10) + 1
6     puts "#{number}..."
7   end
8   "#{person} got a #{number}"
9 end
10 end
11
12 number_of = Roulette.new
13 puts number_of.bob
14 puts number_of.frank

```

El programa contiene un bug. Cuando se ejecuta produce la siguiente salida:

```

~/chapter8ReflectionandMetaprogramming$ ruby roulette.rb 2> err 1> tutu
~/chapter8ReflectionandMetaprogramming$ cat -n err
1  roulette.rb:6:in `method_missing': stack level too deep (SystemStackError)
2    from roulette.rb:4:in `times'
3    from roulette.rb:4:in `method_missing'
4    from roulette.rb:8:in `method_missing'
5    from roulette.rb:13
~/chapter8ReflectionandMetaprogramming$ tail tutu
4...
1...
10...
8...
6...
4...
8...
7...
10...
9...

```

¿Sabrías decir cual es el error? ¿Sabrías como arreglarlo?

15.5. Hooks (Ganchos)

`Module`, `Class` y `Object` proveen varios métodos callback o *hooks*. Por defecto estos métodos no están definidos, pero si los definimos serán invocados cuando ocurra un cierto evento.

15.5.1. El Hook `inherited`

Cuando se define una nueva clase, Ruby invoca al método de clase `inherited` de la superclase de la nueva clase pasándole como argumento el objeto que representa a la nueva clase.

15.5.2. El Hook `included`

Cuando un módulo es incluído en una clase, el método de clase `included` es invocado con argumento el objeto clase o módulo en el cual es incluído.

Veamos un ejemplo de uso de `included`.

Creando declaraciones de métodos abstractos

En Ruby no existe una palabra `abstract` que indique que un método es abstracto. Creemos una:

```
[~/Chapter7ClassesAndModules/inheritance]$ cat abstract.rb
module Abstract
  def self.included(base)
    base.extend(ClassMethods)
  end

  module ClassMethods
    def abstract_methods(*args)
      args.each do |name|
        class_eval(<<-CHUCHU)
        def #{name}(*args)
          raise NotImplementedError.new("You must implement #{name}.")
        end
      CHUCHU
    end # args.each
  end # def abstract_methods
end # module ClassMethods
end

class Tutu
  include Abstract

  abstract_methods :foo, :bar
end

[~/Chapter7ClassesAndModules/inheritance]$ pry
[1] pry(main)> require './abstract'
=> true
[2] pry(main)> z = Tutu.new
=> #<Tutu:0x007f9bde11ac70>
[3] pry(main)> z.bar
NotImplementedError: You must implement bar.
from (eval):2:in `bar'
[4] pry(main)>
```

Incluyendo Métodos de Instancia y de Clase de un Módulo

El Problema

- Ejercicio 15.5.1.**
- Cuando hacemos `include` de un módulo para mezclarlo en un módulo o en una clase, los métodos pasan a estar disponibles al nivel de instancia.
 - Si usamos `extend` los tenemos disponibles como métodos de clase.
- Encuentre alguna forma de hacer que un módulo pueda proveer métodos de clase y métodos de instancia

Primeras Solución

```
[~/src/ruby/ruby_best_practices/chapter3_mastering_the_dynamic_toolkit/tracking_mixins(master)
module MyFeatures

  module ClassMethods
    def say_hello
      "Hello"
    end
  end
```

```

def say_goodbye
  "Goodbye"
end
end

def say_hello
  "Hello from #{self}!"
end

def say_goodbye
  "Goodbye from #{self}"
end
end

class A
  include MyFeatures
  extend MyFeatures::ClassMethods
end

# List instance methods
puts A.public_instance_methods.select { |m| m =~/say/ }.inspect # [:say_hello, :say_goodbye]
# List class methods
puts A.methods.select { |m| m =~/say/ }.inspect                 # [:say_hello, :say_goodbye]

puts A.say_hello      # Hello
puts A.new.say_hello # Hello from #<A:0x007fb8c38880f0>!

```

Segunda Solución: Usando el Gancho included

```
[~/src/ruby/ruby_best_practices/chapter3_mastering_the_dynamic_toolkit/tracking_mixins(master)
module MyFeatures

  module ClassMethods
    def say_hello
      "Hello"
    end

    def say_goodbye
      "Goodbye"
    end
  end

  def self.included(base) # set the hook
    base.extend(ClassMethods)
  end

  def say_hello
    "Hello from #{self}!"
  end

  def say_goodbye
    "Goodbye from #{self}"
  end

```

```

end

class A
  include MyFeatures # Now only one include is needed
end

# List instance methods
puts A.public_instance_methods.select { |m| m =~/say/ }.inspect # [:say_hello, :say_goodbye]
# List class methods
puts A.methods.select { |m| m =~/say/ }.inspect                 # [:say_hello, :say_goodbye]

puts A.say_hello      # Hello
puts A.new.say_hello # Hello from #<A:0x007fb8c38880f0>!

```

15.5.3. Ganchos/Hooks: Sustituyendo (y delegando en) un método existente

El ejemplo muestra como sustituir el método `system` del módulo `Kernel`:

```

~/rubystesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ cat -n ex0678.rb
 1 module Kernel
 2   alias_method :old_system, :system
 3   def system(*args)
 4     result = old_system(*args)
 5     puts "system(#{args.join(' ', '')}) returned #{result}"
 6     result
 7   end
 8 end
 9
10 system("date")
11 system("kangaroo", "-hop 10", "skippy")

~/rubystesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ ruby ex0678.rb
domingo, 20 de noviembre de 2011, 11:21:19 GMT
system(date) returned true
system(kangaroo, -hop 10, skippy) returned false

```

15.5.4. Ganchos/Hooks: Interviniendo en el Momento en que un Objeto es Creado

En esta sección se muestra una forma de intervenir en el momento en que un objeto es creado. El ejemplo consiste en que queremos añadirle a todo objeto un atributo `timestamp` que indica su fecha de creación. Se hace abriendo la clase `Object` y añadiéndole el atributo:

```

class Object
  attr_accessor :timestamp
end

```

Ahora bien, el método `new` es un método de clase, (`Tutu.new`) es un metodo de instancia de la clase `Class`:

```

>> Class.instance_methods.select { |x| x =~/new/ }
=> ["new"]

```

Así que lo que haremos es abrir la clase `Class` y reescribir `new`. Repetimos el truco usado en `system` en la sección 15.5.3. La técnica no es completa, debido a que ciertos objetos preconstruidos como pueda ser el caso de las `String` se construyen sin que se produzca una llamada a `new`.

```

~/rubytesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ ruby -rdebug ex0681.rb
Debug.rb
Emacs support available.

ex0681.rb:2: class Object
(rdb:1) l 1,50
[1, 50] in ex0681.rb
  1
=> 2   class Object
  3     attr_accessor :timestamp
  4   end
  5   class Class
  6     alias_method :old_new,  :new
  7     def new(*args)
  8       result = old_new(*args)
  9       result.timestamp = Time.now
 10      result
 11    end
 12  end
 13  class Test
 14  end
 15
 16  obj1 = Test.new
 17  sleep(0.002)  #!sh!
 18  obj2 = Test.new
 19
 20  obj1.timestamp.to_f
 21  obj2.timestamp.to_f
(rdb:1) b 18
Set breakpoint 1 at ex0681.rb:18
(rdb:1) c
Breakpoint 1, toplevel at ex0681.rb:18
ex0681.rb:18: obj2 = Test.new
(rdb:1) p obj1
#<Test:0x100361c70 @timestamp=Sun Nov 20 11:54:46 +0000 2011>
(rdb:1) l
[13, 22] in ex0681.rb
  13   class Test
  14   end
  15
  16   obj1 = Test.new
  17   sleep(0.002)  #!sh!
=> 18   obj2 = Test.new
  19
  20   obj1.timestamp.to_f
  21   obj2.timestamp.to_f
(rdb:1) n
ex0681.rb:20: obj1.timestamp.to_f
(rdb:1) obj2
#<Test:0x10035f628 @timestamp=Sun Nov 20 11:55:15 +0000 2011>
(rdb:1) p obj1.timestamp.to_f
1321790086.82083
(rdb:1) p obj2.timestamp.to_f

```

```
1321790115.59523
(rdb:1)
```

15.6. Traza

15.6.1. set_trace_func

Es posible monitorizar la ejecución de un programa usando `set_trace_func`. `set_trace_func` tiene la sintaxis:

```
set_trace_func( proc ) → proc
set_trace_func( nil ) → nil
```

Establece `proc` como un manejador para crear una traza de la ejecución. Si es `nil` desactivamos la traza.

El `proc` toma 6 parámetros:

- Un nombre de evento,
- Un nombre de fichero,
- Un número de línea
- Un ID de objeto
- Un binding, y
- Un nombre de una clase

`proc` es invocado cada vez que ocurre un evento. Los eventos son:

- c-call
- c-return
- call
- class
- end
- line
- raise
- return

La traza se desactiva en el contexto de `proc`.

```
~/rubystesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ cat -n ex0682.rb
1  class Test
2    def initialize
3      puts "In initialize"
4    end
5    def test
6      a = 1
7      b = 2
8    end
9  end
10
```

```

11  set_trace_func proc {|event, file, line, id, binding, classname|
12    printf "%8s %s:-%2d %10s %8s\n", event, file, line, id, classname
13  }
14  t = Test.new
15  t.test

~/rubytesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ ruby ex0682.rb
  line ex0682.rb:14          false
  c-call ex0682.rb:14        new   Class
    call ex0682.rb:2  initialize  Test
    line ex0682.rb:3  initialize  Test
  c-call ex0682.rb:3        puts   Kernel
  c-call ex0682.rb:3        write   IO
In initialize
c-return ex0682.rb:3        write   IO
  c-call ex0682.rb:3        write   IO

c-return ex0682.rb:3        write   IO
c-return ex0682.rb:3        puts   Kernel
  return ex0682.rb:3  initialize  Test
c-return ex0682.rb:14        new   Class
  line ex0682.rb:15          false
    call ex0682.rb:5        test   Test
    line ex0682.rb:6        test   Test
    line ex0682.rb:7        test   Test
  return ex0682.rb:7        test   Test

```

Otro método que puede ser usado para obtener mas información sobre la traza es `trace_var`

```

~/rubytesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ cat -n trace_var.rb
1  trace_var :$_, proc{|v| puts "$_ is now '#{v}'" }
2  $_ = "hello"
3  $_ = ' there'

~/rubytesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ ruby trace_var.rb
$_ is now 'hello'
$_ is now ' there'

```

15.6.2. caller

La llamada:

```
caller(skip = 1)
```

retorna un array de cadenas con el formato

```
\file:line" o \file:line: in 'method'
```

El parámetro opcional indica el número de pilas iniciales a saltarse.

Retorna `nil` si el numero de las que te saltas es mayor que el número de llamadas en la pila.

Ejemplo:

```

~/rubytesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ cat -n ex0683.rb
 1 def cat_a
 2   puts caller.join("\n")
 3 end
 4 def cat_b
 5   cat_a
 6 end
 7 def cat_c
 8   cat_b
 9 end
10 cat_c

~/rubytesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ ruby ex0683.rb
ex0683.rb:5:in `cat_b'
ex0683.rb:8:in `cat_c'
ex0683.rb:10

```

Veamos un ejemplo mas completo:

```

~/rubytesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ cat -n caller.rb
 1 def a(skip)
 2   caller(skip)
 3 end
 4 def b(skip)
 5   a(skip)
 6 end
 7 def c(skip)
 8   b(skip)
 9 end
10 puts c(0).inspect
11 puts c(1).inspect
12 puts c(2).inspect
13 puts c(3).inspect
14 puts c(4).inspect
15 puts c(5).inspect

~/rubytesting/programmingRuby/src_of_programming_ruby_by_dave_thomas$ ruby caller.rb
["caller.rb:2:in `a'", "caller.rb:5:in `b'", "caller.rb:8:in `c'", "caller.rb:10"]
["caller.rb:5:in `b'", "caller.rb:8:in `c'", "caller.rb:11"]
["caller.rb:8:in `c'", "caller.rb:12"]
["caller.rb:13"]
[]
nil

```

15.7. Los Módulos ObjectSpace y GC

El módulo `ObjectSpace` contiene métodos que permiten atravesar los objetos existentes::

```

>> ObjectSpace.each_object(Numeric) { |x| p x }
2.71828182845905
3.14159265358979
9223372036854775807
95166617196965824296817888526506730120332022876053991446234245250621584887778
2.22044604925031e-16

```

```

1.79769313486232e+308
2.2250738585072e-308
0
0
95.1
102.5
=> 11

>> Float::MAX
=> 1.79769313486232e+308
>> Float::EPSILON
=> 2.22044604925031e-16
>> Math::PI
=> 3.14159265358979
>> Math::E
=> 2.71828182845905

```

El módulo `ObjectSpace` no accede a los objetos con valores inmediatos: `Fixnum`, `Symbol`, `true`, `false` y `nil`:

```

>> a = 102
=> 102
>> a.class
=> Fixnum
>> b = 95
=> 95
>> ObjectSpace.each_object(Numeric) { |x| p x }
2.71828182845905
3.14159265358979
9223372036854775807
97997521621253453151246418904344427109054468668812181388822276467821973581887
2.22044604925031e-16
1.79769313486232e+308
2.2250738585072e-308
0
0
=> 9
>>

```

Obsérvese que `ObjectSpace.each_object(Class)` nos pasa el objeto, no el nombre del objeto:

```

>> ObjectSpace.each_object(Numeric) { |x| p x.class }
Float
Float
Bignum
Bignum
Float
Float
Float
Float
Bignum
Bignum
=> 9
>>

```

Podemos inspeccionar los nombres de los métodos de un objeto usando `methods`:

```
>> r = 1..10
=> 1..10
>> x = r.methods
=> ["find", "inspect", ..., "include?", "is_a?", "all?"]
>> x.length
=> 94

>> x.index("first")
=> 53
>> w = x[53]
=> "first"

>> m = r.method w
=> #<Method: Range#first>
>> m.arity
=> 0
>> r.send(w)
=> 1

>> m.call
=> 1
>> m.call()
=> 1
>> m[]
=> 1
>>
```

15.8. Estructuras de Control a la Carta

15.8.1. Creando una *Palabra Clave*

Supongamos que estamos escribiendo un programa C# que conecta con un servidor remoto y que tenemos un objeto que representa dicha conexión:

```
RemoteConnection conn = new RemoteConnection("my_server");
String stuff = conn.readStuff();
conn.dispose(); // close the connection to avoid a leak
```

Este código libera los recursos asociados a la conexión después de usarla. Sin embargo, no considera la presencia de excepciones. Si `readStuff()` lanza una excepción, `conn.dispose()` no se ejecutará. C# proporciona una palabra clave `using` que simplifica el manejo de la liberación de recursos:

```
RemoteConnection conn = new RemoteConnection("some_remote_server");
using (conn) {
    conn.readSomeData();
    doSomeMoreStuff();
}
```

La palabra `using` espera que el objeto `conn` tenga un método con nombre `dispose()`. Este método es llamado automáticamente después del código entre llaves, tanto si se genera una excepción como si no.

Supongamos que se nos pide como ejercicio que extendamos Ruby con una *palabra clave using* que funcione de manera similar al `using` de C#.

Para comprobar que lo hagamos correctamente se nos da el siguiente programa de test (véase Unit Testing en la wikipedia y la documentación de la librería `Test::Unit Test::Unit::Assertions`):

```

~/rubytesting$ cat -n using_test.rb
 1  require 'test/unit'
 2
 3  class TestUsing < Test::Unit::TestCase
 4    class Resource
 5      def dispose
 6        @disposed = true
 7      end
 8
 9      def disposed?
10        @disposed
11      end
12    end
13
14    def test_disposes_of_resources
15      r = Resource.new
16      using(r) {}
17      assert r.disposed?
18    end
19
20    def test_disposes_of_resources_in_case_of_exception
21      r = Resource.new
22      assert_raises(Exception) {
23        using(r) {
24          raise Exception
25        }
26      }
27      assert r.disposed?
28    end
29  end

```

Las clases que representan tests deben ser subclases de `Test::Unit::TestCase`. Los métodos que contienen `assertions` deben empezar por la palabra `test`. `Test::Unit` utiliza introspección para decidir para encontrar que métodos debe ejecutar.

Por supuesto, si se ejecutan ahora, nuestras pruebas fallan:

```

~/rubytesting$ ruby using_test.rb
Loaded suite using_test
Started
EF
Finished in 0.005613 seconds.

1) Error:
test_disposes_of_resources(TestUsing):
NoMethodError: undefined method `using' for #<TestUsing:0x100348608>
  using_test.rb:16:in `test_disposes_of_resources'

2) Failure:
test_disposes_of_resources_in_case_of_exception(TestUsing) [using_test.rb:22]:
<Exception> exception expected but was
Class: <NoMethodError>
Message: <"undefined method `using' for #<TestUsing:0x1003485e0>">
---Backtrace---
using_test.rb:23:in `test_disposes_of_resources_in_case_of_exception'

```

```
using_test.rb:22:in `test_disposes_of_resources_in_case_of_exception'  
-----
```

```
2 tests, 1 assertions, 1 failures, 1 errors
```

Idea: No podemos definir `using` como una palabra clave, por supuesto, pero podemos producir un efecto parecido definiendo `using` como un método de `Kernel`:

```
~/rubytesting$ cat -n using1.rb  
1 module Kernel  
2   def using(resource)      # resource: el recurso a liberar con dispose  
3     begin                  # llamamos al bloque  
4       yield  
5       resource.dispose    # liberamos  
6     end  
7   end  
8 end
```

Sin embargo, esta versión no está a prueba de excepciones. Cuando ejecutamos las pruebas obtenemos:

```
~/rubytesting$ ruby -rusing1 using_test.rb  
Loaded suite using_test  
Started  
.F  
Finished in 0.005043 seconds.
```

```
1) Failure:  
test_disposes_of_resources_in_case_of_exception(TestUsing) [using_test.rb:27]:  
<nil> is not true.
```

```
2 tests, 3 assertions, 1 failures, 0 errors
```

El mensaje nos indica la causa del fallo:

```
test_disposes_of_resources_in_case_of_exception(TestUsing) [using_test.rb:27]: <nil> is not tr
```

Es posible también ejecutar un test específico:

```
~/rubytesting$ ruby -rusing1 -w using_test.rb --name test_disposes_of_resources  
Loaded suite using_test  
Started  
.F  
Finished in 0.000196 seconds.
```

```
1 tests, 1 assertions, 0 failures, 0 errors
```

```
~/rubytesting$
```

(Para saber mas sobre Unit Testing en Ruby vea Ruby Programming/Unit testing en Wikibooks).

Volviendo a nuestro fallo, recordemos que la cláusula `rescue` es usada cuando queremos que un código se ejecute si se produce una excepción:

```
begin  
  file = open("/tmp/some_file", "w")  
  # ... write to the file ...
```

```

file.close
rescue
  file.close
  fail # raise an exception
end

```

Pero en este caso es mejor usar la cláusula `ensure`. El código bajo un `ensure` se ejecuta, tanto si se ha producido excepción como si no:

```

begin
  file = open("/tmp/some_file", "w")
  # ... write to the file ...
rescue
  # ... handle the exceptions ...
ensure
  file.close # ... and this always happens.
end

```

Es posible utilizar `ensure` sin la cláusula `rescue`, y viceversa, pero si aparecen ambas en el mismo bloque `begin...end` entonces `rescue` debe preceder a `ensure`.

Reescribimos nuestro `using` usando `ensure`:

```

~/rubytesting$ cat -n using.rb
1 module Kernel
2   def using(resource)
3     begin
4       yield
5     ensure
6       resource.dispose
7     end
8   end
9 end

```

Ahora los dos tests tienen éxito:

```

~/rubytesting$ ruby -rusing using_test.rb
Loaded suite using_test
Started
..
Finished in 0.000346 seconds.

2 tests, 3 assertions, 0 failures, 0 errors

```

Lo normal durante el desarrollo es guardar las pruebas en una carpeta con nombre `test` o `t`:

```

project
|
`-lib/
|   |
|   '-using.rb
|   |
|   '-otros ficheros ...
`-test/
    |
    '-using_test
    |
    '-otros tests ...

```

El problema con esta estructura es que hay que decirle a Ruby donde debe encontrar la librería cuando se ejecutan las pruebas.

- Añadir el camino en el que están las librerías a `$LOAD_PATH`
- Ejecutar los tests siempre desde el directorio raíz e incluir en los tests una línea como esta:

```
$:.unshift File.join(File.dirname(__FILE__), "..", "lib")
require ...
```

La variable `$:` es una array conteniendo el PATH de búsqueda de librerías.

Lo habitual, cuando un proyecto crece es que los tests se acumulen. Lo normal es clasificarlos por afinidades en diferentes ficheros. Los podemos agrupar en suites. Para ello creamos un fichero con una lista de `requires`

```
require 'test/unit'
require 'tc_smoke'
require 'tc_client_requirements'
require 'tc_extreme'
```

Ahora, simplemente ejecutando este fichero ejecutaremos todos los casos en la suite.

15.9. Métodos Missing y Constantes Missing

15.9.1. Creando Dinámicamente los Métodos dentro de `method_missing`

Podemos usar `define_method` dentro de `missing_method` para crear el método fantasma. `define_method` define un método de instancia en el objeto receptor. La sintaxis es:

```
define_method(symbol, method)      => new_method
define_method(symbol) { block }     => proc
```

El parámetro `method` puede ser un `Proc` o un `Method`. Si se usa un bloque como parámetro, es usado como cuerpo del método. El bloque es evaluado utilizando `instance_eval`. Puesto que `define_method` es privado y de clase, no podemos llamarlo directamente dentro de `method_missing`. Al ser privado no podemos llamarlo explicitamente en la clase. Dentro de `method_missing` la variable `self` refiere al objeto, y `define_method` no es un método de instancia. Una solución es llamarlo via `Kernel.send`.

```
MacBook-Air-de-casiano:rubystesting casiano$ cat -n chamcham3.rb
1  class A
2    def method_missing(n, *a)
3      super unless %{"aa bb cc dd"}.include? n.to_s
4
5      m = Kernel.send :define_method, n do
6        puts "Inside #{n}"
7      end
8      m[]
9      puts "Inside method_missing, building #{n}"
10     end
11   end
12
13 a = A.new()
14 a.aa()
15 a.aa()
16 a.aa()
17 a.ee()
```

```
MacBook-Air-de-casiano:rubystesting casiano$ ruby chamcham3.rb
Inside aa
Inside method_missing, building aa
Inside aa
Inside aa
chamcham3.rb:3:in `method_missing': undefined method `ee' for #<A:0x10513dcf0> (NoMethodError)
  from chamcham3.rb:17
```

Otra forma de hacerlo es usar `class_eval`:

```
MacBook-Air-de-casiano:rubystesting casiano$ cat -n chamcham5.rb
1  class A
2
3    def method_missing(n, *a)
4      super unless %{aa bb cc dd}.include? n.to_s
5      A.class_eval {
6        m = define_method(n) do
7          puts "Inside #{n} #{self.class.ancestors.inspect}"
8        end
9        m[]
10      }
11      puts "Inside method_missing, building #{n}"
12    end
13  end
14
15 a = A.new()
16 a.aa()
17 a.aa()
18 a.aa()
```

```
MacBook-Air-de-casiano:rubystesting casiano$ ruby chamcham5.rb
Inside aa [Class, Module, Object, Kernel]
Inside method_missing, building aa
Inside aa [A, Object, Kernel]
Inside aa [A, Object, Kernel]
```

O bien llamar a un método de clase desde el cual se llame a `defined_method`:

```
MacBook-Air-de-casiano:rubystesting casiano$ cat -n chamcham4.rb
1  class A
2    def self.build_method(n)
3      m = define_method(n) do
4        puts "Inside #{n} #{self.class.ancestors.inspect}"
5      end
6      m[]
7    end
8
9    def method_missing(n, *a)
10      super unless %{aa bb cc dd}.include? n.to_s
11      self.class.build_method(n)
12      puts "Inside method_missing, building #{n}"
13    end
14  end
15
16 a = A.new()
```

```

17 a.aa()
18 a.aa()
19 a.aa()

MacBook-Air-de-casiano:rubystesting casiano$ ruby chamcham4.rb
Inside aa [Class, Module, Object, Kernel]
Inside method_missing, building aa
Inside aa [A, Object, Kernel]
Inside aa [A, Object, Kernel]

```

15.10. Creando Métodos Dinámicamente

15.10.1. Definiendo Métodos con `class_eval`

15.10.2. Definiendo Métodos con `define_methods`

15.11. Encadenamiento de Alias

15.12. Lenguajes de Dominio Específico. Domain Specific Languages. DSL

15.12.1. Un Lenguaje de Dominio Específico para Describir Recetas de Cocina

Supongamos que queremos crear una Clase `Recipe` cuyo constructor entienda un Lenguaje de Dominio Específico (Domain Specific Language, DSL) para describir recetas de cocina. Queremos que sea posible especificar con facilidad los ingredientes y los pasos de la forma mas simple posible. La API debería ser parecida a esta:

```

puts Recipe.new("Noodles and Cheese") {
  ingredient "Water",    amount => "2 cups"
  ingredient "Noodles",   amount => "1 cup"
  ingredient "Cheese",    amount => "1/2 cup"

  step "Heat water to boiling.",      during => "5 minutes"
  step "Add noodles to boiling water.", during => "6 minutes"
  step "Drain water."
  step "Mix cheese in with noodles."
}

```

y el resultado de ejecutar el código anterior debería ser similar a este:

```

~/Chapter8ReflectionandMetaprogramming$ ruby recipes.rb
Noodles and Cheese
=====

```

Ingredients: Water (2 cups), Noodles (1 cup), Cheese (1/2 cup)

- 1) Heat water to boiling. (5 minutes)
- 2) Add noodles to boiling water. (6 minutes)
- 3) Drain water.
- 4) Mix cheese in with noodles.

Como se ve en el ejemplo de uso, el constructor de la clase recibe un bloque que describe la receta. Una solución es usar `yield` para llamar al bloque:

```

~/Chapter8ReflectionandMetaprogramming$ cat -n recipes2.rb
 1 class Recipe2
 2   attr_accessor :name, :ingredients, :instructions
 3
 4   def initialize(name)
 5     self.name = name
 6     self.ingredients = []
 7     self.instructions = []
 8
 9     yield self
10   end
11
12   def to_s
13     output = name
14     output << "\n#{'=' * name.size}\n\n"
15     output << "Ingredients: #{ingredients.join(', ')}\n\n"
16
17     instructions.each_with_index do |instruction, index|
18       output << "#{index + 1}) #{instruction}\n"
19     end
20
21     output
22   end
23
24   def ingredient(name, options = {})
25     ingredient = name
26     ingredient << " (#{options[:amount]})" if options[:amount]
27
28     ingredients << ingredient
29   end
30
31   def step(text, options = {})
32     instruction = text
33     instruction << " (#{options[:during]})" if options[:during]
34
35     instructions << instruction
36   end
37 end

```

Los métodos `ingredient` y `step` nos ayudan en el proceso de construcción de la cadena de presentación.

Sin embargo, esta solución no es satisfactoria, ya que debemos pasar al bloque de definición el objeto receta que está siendo construido:

```

39   noodles_and_cheese = Recipe2.new("Noodles and Cheese") do |r|
40     r.ingredient "Water",    :amount => "2 cups"
41     r.ingredient "Noodles", :amount => "1 cup"
42     r.ingredient "Cheese",  :amount => "1/2 cup"
43
44     r.step "Heat water to boiling.",      :during => "5 minutes"
45     r.step "Add noodles to boiling water.", :during => "6 minutes"
46     r.step "Drain water."
47     r.step "Mix cheese in with noodles."
48   end
49

```

```
50 puts noodles_and_cheese
```

Peor aún, debemos repetir constantemente el objeto `r.ingredient` y `r.step`. Sabemos que `instance_eval` evalúa el bloque que se le pasa en el contexto del objeto en el que es llamado (`self` en la línea 9 abajo):

```
~/Chapter8ReflectionandMetaprogramming$ cat -n recipes.rb
1 class Recipe
2   attr_accessor :name, :ingredients, :instructions
3
4   def initialize(name, &block)
5     self.name = name
6     self.ingredients = []
7     self.instructions = []
8
9     instance_eval &block
10    end
11
12   def to_s
13     output = <<"EORECIP"
14   #{name}
15   #{'=' * name.size}
16
17   Ingredients: #{ingredients.join(', ')}
18
19   #{
20     out = ""
21     instructions.each_with_index do |instruction, index|
22       out << "#{index + 1}) #{instruction}\n"
23     end
24     out
25   }
26 EORECIP
27   end
28
29   def ingredient(name, options = {})
30     ingredient = name
31     ingredient << " (#{$options[:amount]})" if options[:amount]
32
33     ingredients << ingredient
34   end
35
36   def step(text, options = {})
37     instruction = text
38     instruction << " (#{$options[:for]})" if options[:for]
39
40     instructions << instruction
41   end
42
43   def amount
44     :amount
45   end
46
47   def during
48     :for
```

```

49     end
50   end
51
52   puts Recipe.new("Noodles and Cheese") {
53     ingredient "Water",    amount => "2 cups"
54     ingredient "Noodles", amount => "1 cup"
55     ingredient "Cheese",   amount => "1/2 cup"
56
57     step "Heat water to boiling.",           during => "5 minutes"
58     step "Add noodles to boiling water.",  during => "6 minutes"
59     step "Drain water."
60     step "Mix cheese in with noodles."
61   }

```

15.12.2. Un DSL para Procesar Documentos XML usando `instance_eval`

1. Ejemplo modificado de la sección *Dealing with XML* del libro [9].
2. Véase también <https://github.com/cocoa/eloquent-ruby>.

XML

Un ejemplo de XML. Contenidos del fichero `fellowship.xml`:

```

<?xml version="1.0" encoding="UTF-8" ?>
<document>
  <title>The Fellowship of the Ring</title>
  <author>J. R. R. Tolken</author>
  <published>1954</published>

  <chapter>
    <title>A Long Expected Party</title>
    <content>When Mr. Bilbo Bagins from Bag End ...</content>
  </chapter>

  <chapter>
    <title>A Shadow Of The Past</title>
    <content>The talk did not die down ...</content>
  </chapter>

  <!-- etc -->
</document>

```

REXML

Véanse:

- REXML Tutorial
- XmlSimple - XML made easy
- Veamos un ejemplo de uso de `rexml/document`:

```

#!/usr/bin/env ruby

require 'rexml/document'
filename = ARGV.shift || 'fellowship.xml'

```

```

File.open(filename) do |f|
  doc = REXML::Document.new(f)
  author = REXML::XPath.first(doc, '/document/author')
  print "Author: "
  puts author.text      # J. R. R. Tolken

  puts "Chapters:"
  REXML::XPath.each(doc, '/document/chapter/title') do |title|
    puts "+title.text # A Long Expected Party"
  end                      # A Shadow Of The Past
end

```

Ejecución:

```

$ ruby find_author.rb
Author: J. R. R. Tolken
Chapters:
  A Long Expected Party
  A Shadow Of The Past

```

- También podemos arreglar la errata en el nombre del autor:

```

#!/usr/bin/env ruby

require 'rexml/document'
filename = ARGV.shift || 'fellowship.xml'
File.open(filename) do |f|
  doc = REXML::Document.new(f)
  REXML::XPath.each(doc, '/document/author') do |author|
    author.text = 'J.R.R. Tolkien'
  end
  puts doc
end

```

Un DSL para XML: Primera Versión. Modo de Uso

Aunque REXML es fácil de usar, nos gustaría disponer de un poco mas de expresividad de manera que podamos expresar un conjunto de acciones que se ejecuten cuando se produce cierto matching. El siguiente fragmento de código muestra una primera versión de nuestro DSL:

```

if $0 == __FILE__
  ripper = XmlRipper.new do |r|
    puts "Compiling ..."
    r.on_path '/document/title' do |t|
      puts "Title: "+t.text
    end
    r.on_path '/document/author' do |a|
      puts "Author: "+a.text
    end
    r.action { puts "Chapters: " }
    r.on_path '/document/chapter/title' do |ct|
      puts "+ct.text"
    end
  end
end

```

```

filename = ARGV.shift || 'fellowship.xml'
ripper.run filename
# Compiling ...
# Title: The Fellowship of the Ring
# Author: J. R. R. Tolken
# Chapters:
#   A Long Expected Party
#   A Shadow Of The Past
end

```

La Clase XmlRipper: Primera Aproximación

```

require 'rexml/document'

class XmlRipper
  def initialize()
    @before_action = proc { } # do nothing
    @path_action  = []         # Array: to preserve the order
    @after_action = proc { }
    yield self if block_given?
  end

  def on_path(path, &block)
    @path_action << [ path, block ]
  end

  def action(&block)
    @path_action << [ '', block ]
  end

  def before(&block)
    @before_action = block
  end

  def after(&block)
    @after_action = block
  end

  def run_path_actions(doc)
    @path_action.each do |path_action|
      path, action = path_action
      REXML::XPath.each(doc, path) do |element|
        action[element]
      end
    end
  end

  def run(file_name)
    File.open(file_name) do |f|
      doc = REXML::Document.new(f)
      @before_action[doc]
      run_path_actions(doc)
      @after_action[doc]
    end
  end
end

```

```
end  
end
```

Un DSL para XML: Segunda Versión

Queremos evitar esas cacofonías de uso del objeto `r` en la versión anterior (sección 15.12.2) y poder usarlo así:

```
[~/chapter8ReflectionandMetaprogramming/DSL/xmlripper]$ cat sample.xr  
puts "Compiling ..."  
on_path '/document/title' do |t|  
  puts "Title: "+t.text  
end  
on_path '/document/author' do |a|  
  a.text = "J.R.R. Tolkien"  
  puts "Author: "+a.text  
end  
action { puts "Chapters: " }  
on_path '/document/chapter/title' do |ct|  
  puts " "+ct.text  
end
```

Nueva versión de initialize

Casi lo único que hay que cambiar en la versión anterior es el método `initialize`:

```
def initialize( path = nil, &block )  
  @before_action = proc { } # do nothing  
  @path_action = [] # Array: to preserve the order  
  @after_action = proc { }  
  
  if path then  
    instance_eval( File.read( path ) )  
  else  
    instance_eval( &block ) if block_given?  
  end  
end
```

Si queremos que nuestro DSL pueda ser llamado especificando el contexto para que el bloque se ejecute podemos usar esta versión ligeramente mas complicada:

```
def initialize( path = nil, &block )  
  @before_action = proc { } # do nothing  
  @path_action = [] # Array: to preserve the order  
  @after_action = proc { }  
  
  if path then  
    instance_eval( File.read( path ) )  
  elsif block_given?  
    if block.arity < 1  
      instance_eval( &block )  
    else  
      block.call(self)  
    end  
  end  
end
```

Un Ejecutable para Interpretar Guiones en nuestro DSL *XRipper*

En la misma forma en que trabajan `rake` y `rspec` podemos crear un ejecutable que recibe como entrada un fichero escrito en el DSL *XRipper* y ejecuta las acciones asociadas:

```
[~/DSL/xmlripper]$ cat xripper
#!/usr/bin/env ruby

require 'xml_ripper_v2'

scriptname = ARGV.shift
xmlfile = ARGV.shift

if scriptname and xmlfile and File.exists? scriptname and File.exists? xmlfile
  XmlRipper.new(scriptname).run(xmlfile)
else
  puts "Usage:\n\t#$0 xripper_script file.xml"
end
```

Un Guión Escrito en Nuestro DSL *XRipper* para Nuestro Intérprete `xripper`

Ahora lo único que hay que hacer es ejecutar nuestro intérprete `xripper` sobre un fichero escrito en el DSL:

```
[~/chapter8ReflectionandMetaprogramming/DSL/xmlripper]$ cat sample.xr
puts "Compiling ..."
on_path '/document/title' do |t|
  puts "Title: "+t.text
end
on_path '/document/author' do |a|
  a.text = "J.R.R. Tolkien"
  puts "Author: "+a.text
end
action { puts "Chapters: " }
on_path '/document/chapter/title' do |ct|
  puts "  "+ct.text
end
```

Ejecutando el Intérprete *XRipper*

Para llevar a cabo la ejecución escribimos un `Rakefile`:

```
[~/DSL/xmlripper]$ cat Rakefile
task :default => :run

desc "run xripper"
task :run do
  sh "ruby -I. xripper sample.xr fellowship.xml"
end
```

La ejecución produce esta salida:

```
[~/chapter8ReflectionandMetaprogramming/DSL/xmlripper]$ rake
ruby -I. xripper sample.xr fellowship.xml
Compiling ...
Title: The Fellowship of the Ring
Author: J.R.R. Tolkien
Chapters:
  A Long Expected Party
  A Shadow Of The Past
```

15.12.3. Dos DSLs: Generando XML con Validación via Generación de Métodos

Primer DSL: Lenguaje para la Generación de XML

```
HTMLForm.generate(STDOUT) do
  comment "This is a simple HTML form"
  form :name => "registration",
    :action => "http://www.example.com/register.cgi" do
      content "Name:"
      br
      input :name => "name"
      br
      content "Post:"
      br
      textarea :name => "address", :rows=>6, :cols=>40 do
        "Please enter your post here"          # Este bloque retorna contenido
      end
      br
      button { "Submit" }                   # Este bloque retorna contenido
    end
  end
end
```

XML Generado

```
[xmlgenerator]$ ruby xml_generator.rb > ejemplo.html
[xmlgenerator]$ cat ejemplo.html
<!-- This is a simple HTML form -->
<form name='registration' action='http://www.example.com/register.cgi' method='GET' enctype='a
<input name='name' type='text'/>
<br/>
Post:<br/>
<textarea name='address' rows='6' cols='40'>Please enter your post here</textarea>
<br/>
<button type='submit'>Submit</button>
</form>
```

Visualización del HTML Generado

Segundo DSL: Lenguaje para la Generación del DSL Anterior y su Validación

Este DSL permite describir una gramática XML: que tags son permitidos y para cada tag que atributos son legales y de que tipo son. Se usaría así:

```
class HTMLForm < XMLGrammar
  element :form, :action => REQ,           # atributo requerido
          :method => "GET",            # cadena: valor por defecto
          :enctype => "application/x-www-form-urlencoded",
          :name => OPT              # opcional
  element :input, :type => "text", :name => OPT, :value => OPT,
          :maxlength => OPT, :size => OPT, :src => OPT,
          :checked => BOOL, :disabled => BOOL, :readonly => BOOL
  element :textarea, :rows => REQ, :cols => REQ, :name => OPT,
          :disabled => BOOL, :readonly => BOOL
  element :button, :name => OPT, :value => OPT,
          :type => "submit", :disabled => OPT
```

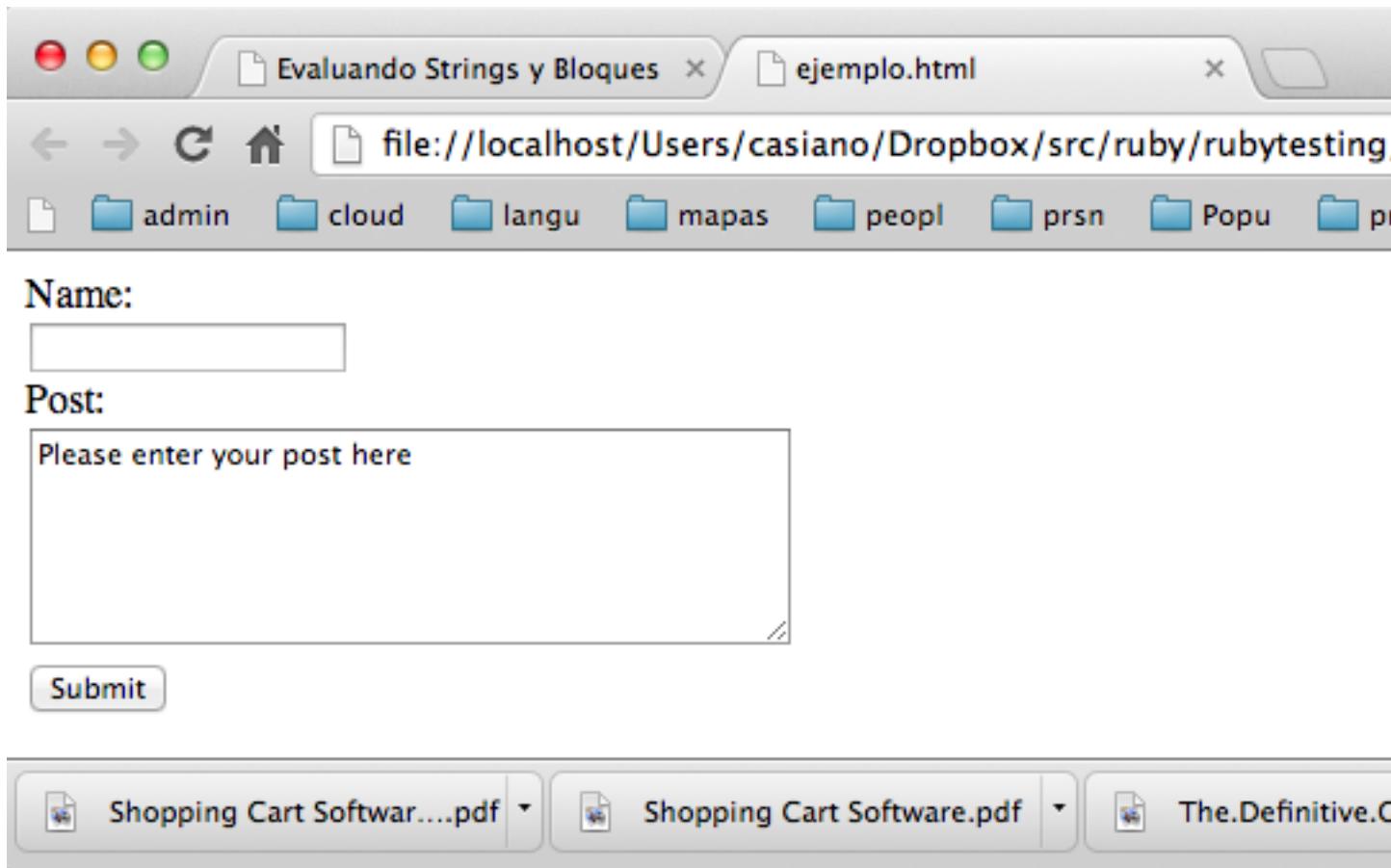


Figura 15.1: Formulario generado

```
element :br
end
```

El método `element` construye un método de instancia con el nombre especificado (por ejemplo, `:form`) como primer argumento en la subclase (`HTMLForm` en el ejemplo). Como segundo argumento opcional recibe un hash especificando los atributos legales del elemento y de qué tipo son (`REQ` por requerido, `OPT` por opcional, una `String` como en `:method => "GET"` indica valor por defecto y `BOOL` para atributos cuyo valor es su propio nombre).

En el código anterior se crean métodos `form`, `input`, `textarea`, `button` y `br` en la clase `HTMLForm`.

La Clase XMLGrammar

```
class XMLGrammar
  # Create an instance of this class, specifying a stream or object to
  # hold the output. This can be any object that responds to <<(String).
  def initialize(out)
    @out = out # Remember where to send our output
  end

  # Invoke the block in an instance that outputs to the specified stream.
  def self.generate(out, &block)
    new(out).instance_eval(&block)
  end
```

```

# Define an allowed element (or tag) in the grammar.
# This class method is the grammar-specification DSL
# and defines the methods that constitute the XML-output DSL.
def self.element(tagname, attributes={})
  @allowed_attributes ||= {}
  @allowed_attributes[tagname] = attributes

  class_eval %Q{
    def #{tagname}(attributes={}, &block)
      tag(:#{tagname}, attributes,&block)
    end
  }
end

# These are constants used when defining attribute values.
OPT = :opt      # for optional attributes
REQ = :req      # for required attributes
BOOL = :bool    # for attributes whose value is their own name

def self.allowed_attributes
  @allowed_attributes
end

# Output the specified object as CDATA, return nil.
def content(text)
  @out << text.to_s
  nil
end

# Output the specified object as a comment, return nil.
def comment(text)
  @out << "<!-- #{text} -->\n"
  nil
end

# Output a tag with the specified name and attribute.
# If there is a block, invoke it to output or return content.
# Return nil.
def tag(tagname, attributes={})
  # Output the tag name
  @out << "<#{tagname}>

  # Get the allowed attributes for this tag.
  allowed = self.class.allowed_attributes[tagname]

  # First, make sure that each of the attributes is allowed.
  # Assuming they are allowed, output all of the specified ones.
  attributes.each_pair do |key,value|
    raise "unknown attribute: #{key}" unless allowed.include?(key)
    @out << " #{key}=#{value}"
  end

  # Now look through the allowed attributes, checking for

```

```

# required attributes that were omitted and for attributes with
# default values that we can output.
allowed.each_pair do |key,value|
  # If this attribute was already output, do nothing.
  next if attributes.has_key? key
  if (value == REQ)
    raise "required attribute '#{key}' missing in <#{tagname}>"
  elsif value.is_a? String
    @out << " #{key}='#{value}'"
  end
end

if block_given?
  # This block has content
  @out << '>'          # End the opening tag
  content = yield        # Invoke the block to output or return content
  if content            # If any content returned
    @out << content.to_s # Output it as a string
  end
  @out << "</#{tagname}>\n" # Close the tag
else
  # Otherwise, this is an empty tag, so just close it.
  @out << "/>\n"
end
nil # Tags output themselves, so they don't return any content.
end
end

```

Código Completo

```

class XMLGrammar
  # Create an instance of this class, specifying a stream or object to
  # hold the output. This can be any object that responds to <<(String).
  def initialize(out)
    @out = out # Remember where to send our output
  end

  # Invoke the block in an instance that outputs to the specified stream.
  def self.generate(out, &block)
    new(out).instance_eval(&block)
  end

  # Define an allowed element (or tag) in the grammar.
  # This class method is the grammar-specification DSL
  # and defines the methods that constitute the XML-output DSL.
  def self.element(tagname, attributes={})
    @allowed_attributes ||= {}
    @allowed_attributes[tagname] = attributes

    class_eval %Q{
      def #{tagname}(attributes={}, &block)
        tag(:#{tagname}, attributes, &block)
      end
    }
  end
end

```

```

end

# These are constants used when defining attribute values.
OPT = :opt      # for optional attributes
REQ = :req      # for required attributes
BOOL = :bool    # for attributes whose value is their own name

def self.allowed_attributes
  @allowed_attributes
end

# Output the specified object as CDATA, return nil.
def content(text)
  @out << text.to_s
  nil
end

# Output the specified object as a comment, return nil.
def comment(text)
  @out << "<!-- #{text} -->\n"
  nil
end

# Output a tag with the specified name and attribute.
# If there is a block, invoke it to output or return content.
# Return nil.
def tag(tagname, attributes={})
  # Output the tag name
  @out << "<#{tagname}""

  # Get the allowed attributes for this tag.
  allowed = self.class.allowed_attributes[tagname]

  # First, make sure that each of the attributes is allowed.
  # Assuming they are allowed, output all of the specified ones.
  attributes.each_pair do |key,value|
    raise "unknown attribute: #{key}" unless allowed.include?(key)
    @out << " #{key}='#{value}'"
  end

  # Now look through the allowed attributes, checking for
  # required attributes that were omitted and for attributes with
  # default values that we can output.
  allowed.each_pair do |key,value|
    # If this attribute was already output, do nothing.
    next if attributes.has_key? key
    if (value == REQ)
      raise "required attribute '#{key}' missing in <#{tagname}>"
    elsif value.is_a? String
      @out << " #{key}='#{value}'"
    end
  end
end

```

```

if block_given?
  # This block has content
  @out << '>'          # End the opening tag
  content = yield        # Invoke the block to output or return content
  if content            # If any content returned
    @out << content.to_s # Output it as a string
  end
  @out << "</#{tagname}>\n" # Close the tag
else
  # Otherwise, this is an empty tag, so just close it.
  @out << "/>\n"
end
nil # Tags output themselves, so they don't return any content.
end
end

class HTMLForm < XMLGrammar
  element :form, :action => REQ,
           :method => "GET",
           :enctype => "application/x-www-form-urlencoded",
           :name => OPT
  element :input, :type => "text", :name => OPT, :value => OPT,
           :maxlength => OPT, :size => OPT, :src => OPT,
           :checked => BOOL, :disabled => BOOL, :readonly => BOOL
  element :textarea, :rows => REQ, :cols => REQ, :name => OPT,
             :disabled => BOOL, :readonly => BOOL
  element :button, :name => OPT, :value => OPT,
             :type => "submit", :disabled => OPT
  element :br
end

HTMLForm.generate(STDOUT) do
  comment "This is a simple HTML form"
  form :name => "registration",
        :action => "http://www.example.com/register.cgi" do
    content "Name:"
    br
    input :name => "name"
    br
    content "Post:"
    br
    textarea :name => "address", :rows=>6, :cols=>40 do
      "Please enter your post here"
    end
    br
    button { "Submit" }
  end
end

```

15.12.4. Creando un ORM

1. Ruby Metaprogramming Tutorial - Part 1 - send method por José Mota (YouTube)
2. Ruby Metaprogramming Tutorial - Part 2 - define_method por José Mota (YouTube)

3. Ruby Metaprogramming Tutorial - Part 3 - ORM example por José Mota (YouTube)
4. GitHub repo source code for the project creator exercise in the "Metaprogramming in Ruby" course.

15.13. Práctica: DSL: Redacción de Cuestionarios I (Sin Contexto)

Se trata de escribir un programa que redacte cuestionarios. En principio, sólo soportaremos preguntas del tipo selección múltiple:

1. ¿En qué año Cristóbal Colón descubrió América?

- 1 - 1942
- 2 - 1492
- 3 - 1808
- 4 - 1914

Su respuesta:

Debe definir una clase `Quiz` que soporte un pequeño lenguaje en el que las preguntas puedan ser especificadas. El constructor de `Quiz` va seguido de un bloque al que pasa como argumento el objeto `e` que representa al examen:

```
quiz = Quiz.new("Cuestionario de PFS 10/12/2011") do |e|
  e.question '¿En qué año Cristóbal Colón descubrió América?',
    e.wrong =>'1942',
    e.right =>'1492',
    e.wrong =>'1808',
    e.wrong =>'1914'

  a = rand(10)
  b = rand(10)
  e.question "#{a}+#{b} = ",
    e.wrong =>"44",
    e.wrong =>"#{a + b + 2}",
    e.right =>"#{a + b}",
    e.wrong =>"#{a + b - 2}"
end

quiz.run
```

El programa anterior podría producir una salida parecida a esta:

```
MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano$ ruby Quiz.rb
Cuestionario de PFS 10/12/2011
```

¿En qué año Cristóbal Colón descubrió América?

- 1 - 1942
- 2 - 1492
- 3 - 1808
- 4 - 1914

Su respuesta: 3

0+8 =

1 - 44

2 - 10
3 - 8
4 - 6

Su respuesta: 1

0 respuestas correctas de un total de 2.

MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano\$

- Los cuestionarios deberían tener un método `to_s` que devuelve una `String` conteniendo el examen en texto plano. Así al imprimir el objeto `quiz` del ejemplo anterior:

```
puts quiz
puts "*****"
```

obtendríamos como salida:

Cuestionario de PFS 10/12/2011

¿En que año Cristobal Colón descubrió América?

1 - 1942
2 - 1492
3 - 1808
4 - 1914

0+8 =

1 - 44
2 - 10
3 - 8
4 - 6

- Los cuestionarios deberían tener un método `run` que formulará cada una de las preguntas del cuestionario y mostrara el porcentaje de aciertos
- Puede que le interese crear tres clases, una para modelar las respuestas (`Answer`), otra para modelar las preguntas (`Question`) y una tercera para el cuestionario (`Quiz`)
- El método `question` recibe dos argumentos. El primero es el título del examen, el segundo es un hash:

```
e.question '¿En que año Cristobal Colón descubrió América?',
e.wrong =>'1942',
e.right =>'1492',
e.wrong =>'1808',
e.wrong =>'1914'
```

de modo que la llamada, realmente es equivalente a:

```
e.question('¿En que año Cristobal Colón descubrió América?',
{e.wrong =>'1942', e.right =>'1492', e.wrong =>'1808', e.wrong =>'1914'})
```

si el segundo argumento de `question` es un hash y las claves son `:wrong` y `:right` se va a producir una colisión y el último valor sobreescribirá a los anteriores. ¿Como resolverlo? Una posible forma de hacerlo es que los métodos `wrong` y `right` diferencien las ocurrencias de las respuestas usando un contador `@counter`:

```
def wrong
  @counter += 1
  [@counter, WRONG]
end
```

- Escriba un método `to_html` que genere una página describiendo el examen. Use ERB.
- Opcionalmente puede incluir hojas de estilo, javascript, etc. en el HTML generado
- Use TDD con RSpec
- Use Unit Testing
- Use Continuous Integration (Travis)
- Use Continuous Testing (Guard)
- Documente su gema (véase `RDOC::Markup` o `RDOC` o `YARD`).
- Cree una gema `ull-etsii-aluXX-quiz`
- Publique la gema en RubyGems
- Indique la URL de su repositorio en GitHub y la URL en RubyGems

15.14. Práctica: DSL: Redacción de Cuestionarios II (Con Contexto)

Se trata de escribir un programa que redacte cuestionarios. En principio, sólo soportaremos preguntas del tipo selección múltiple:

1. ¿En que año Cristobal Colón descubrió América?

```
1 - 1942
2 - 1492
3 - 1808
4 - 1914
```

Su respuesta:

Debe definir una API que soporte un pequeño lenguaje en el que las preguntas puedan ser especificadas de una forma natural. algo así vale:

```
quiz = Quiz.new("Cuestionario de PFS 10/12/2011") {
  question '¿En que año Cristobal Colón descubrió América?',
    wrong =>'1942',
    right =>'1492',
    wrong =>'1808',
    wrong =>'1914'

  a = rand(10)
  b = rand(10)
  question "#{a}+#{b} = ",
    wrong =>"44",
```

```

wrong => "#{a + b + 2}",
right => "#{a + b}",
wrong => "#{a + b - 2}"
}

puts quiz
puts "*****"
quiz.run

```

El programa anterior podría producir una salida parecida a esta:

```
MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano$ ruby Quiz.rb
Cuestionario de PFS 10/12/2011
```

¿En que año Cristobal Colón descubrió América?

- 1 - 1942
- 2 - 1492
- 3 - 1808
- 4 - 1914

0+8 =

- 1 - 44
- 2 - 10
- 3 - 8
- 4 - 6

```
*****
```

```
Cuestionario de PFS 10/12/2011
```

¿En que año Cristobal Colón descubrió América?

- 1 - 1942
- 2 - 1492
- 3 - 1808
- 4 - 1914

Su respuesta: 3

0+8 =

- 1 - 44
- 2 - 10
- 3 - 8
- 4 - 6

Su respuesta: 1

0 respuestas correctas de un total de 2.

```
MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano$
```

- Los cuestionarios deberían tener un método `to_s`

- Los cuestionarios deberían tener un método `run` que formulara cada una de las preguntas del cuestionario y mostrara el porcentaje de aciertos
- Puede que le interese crear tres clases, una para las respuestas (`Answer`), otra para las preguntas (`Question`) y una para el cuestionario (`Quiz`)
- Hay un problema con la llamada al método `question`:

```
question '¿En que año Cristobal Colón descubrió América?',
  wrong =>'1942',
  right =>'1492',
  wrong =>'1808',
  wrong =>'1914'
```

si el segundo argumento es un hash y las claves son `wrong` y `right` se va a producir una colisión y el último valor sobreescribirá a los anteriores. ¿Se puede resolver?

15.15. Práctica: HTML DSL

En esta práctica deberá escribir una clase que provea una funcionalidad parecida (por supuesto, no completa) a la de la librería markaby. La librería provee métodos `html`, `body`, `h1`, etc. que permiten la escritura de los correspondientes tags HTML:

```
require 'markaby'

mab = Markaby::Builder.new
mab.html do
  head { title "Boats.com" }
  body do
    h1 "Boats.com has great deals"
    ul do
      li "$49 for a canoe"
      li "$39 for a raft"
      li "$29 for a huge boot that floats and can fit 5 people"
    end
  end
end
puts mab.to_s
```

Los métodos, como `b` reciben como primer argumento una cadena que es la que hay que rodear entre las marcas `` y ``. También pueden recibir opcionalmente un segundo argumento que sea un hash especificando los atributos asociados con el tag:

```
a "google", :href => "http://www.google.com", :name => "foo"
```

que debería dar lugar a un texto parecido a este:

```
<a href = "http://www.google.com", name = "foo">google</a>
```

Si en la llamada no se provee la cadena inicial, se deberá proveer un bloque. Dicho bloque construye la cadena que se interpolará entre las marcas. Así:

```
head { title "My wonderful home page" }
```

debería producir la cadena:

```
<head><title>My wonderful home page</title></head>
```

La ejecución de este código

```
~/chapter8ReflectionandMetaprogramming$ cat -n html.rb
 1 class HTML
 2   attr_accessor :page
 3
 4   def initialize.....
 5     .....
 6     .....
 7   end
 8
 9   def build_attr(attributes)
10     .....
11     .....
12   end
13
14   def .....(tag, *args)
15     .....
16     .....
17     .....
18     .....
19     .....
20     .....
21     .....
22     .....
23     .....
24     .....
25     .....
26     .....
27   end
28
29   def to_s
30     .....
31   end
32 end
33
34 if __FILE__ == $0
35   q= HTML.new {
36     html {
37       head(:dir => "chazam", :lang => "spanish") { title "My wonderful home page" }
38       body do
39         h1 "Welcome to my home page!", :class => "chuchu", :lang => "spanish"
40         b "My hobbies:"
41         ul do
42           li "Juggling"
43           li "Knitting"
44           li "Metaprogramming"
45         end #ul
46       end # body
47     }
48   }
49   puts q
50 end
```

debería producir una salida parecida a esta:

```

~/chapter8ReflectionandMetaprogramming$ ruby html.rb
<html>
<head dir = "chazam" lang = "spanish">
<title>
My wonderful home page
</title>
</head>
<body>
<h1 class = "chuchu" lang = "spanish">
Welcome to my home page!
</h1> <b>
My hobbies:
</b> <ul>
<li>
Juggling
</li> <li>
Knitting
</li> <li>
Metaprogramming
</li>
</ul>
</body>
</html>

```

Repase las secciones 15.4.5 y 15.12.1.

Solución

Puede consultar una solución en <https://github.com/crguezl/dsl-quiz-simple>

15.16. Práctica: HTML DSL con Git y Rake

Complete la práctica descrita en la sección 15.15 con los siguientes requisitos:

- Use `git` y `github` para el desarrollo de la práctica. Cree en el repositorio un subproyecto LPP y en el mismo un subproyecto para la práctica.
- Documente el código
- En el directorio `lib` residirá la librería
- Cree un directorio `test` en el que residirán las pruebas
- Escriba las pruebas. Recuerde:
 - Todas las pruebas deben automatizarse
 - Todos los fallos que se detecten deberían quedar traducidos en pruebas
 - La aplicación debería pasar todas las pruebas después de cualquier modificación importante y también al final del día
 - El desarrollo de las pruebas debería preceder el desarrollo del código
 - Todos los requerimientos deben ser expresados en forma de pruebas
 - El tiempo invertido en *Comprobar (testing)* y el tiempo invertido en *Depurar (debugging)* mantiene una relación inversa. *Cuanto menor es el tiempo invertido en la preparación de pruebas mayor será nuestro tiempo de depuración.* Cuanto mejor sea nuestra suite de pruebas menor será el tiempo que necesitemos para fijar errores en nuestros programas.

- Las pruebas no aseguran la corrección de nuestro programa. Las pruebas descubren errores.
Una prueba tiene éxito cuando falla y demuestra la presencia de un error.
- En palabras de Bruce Eckel (*Thinking in Java*):

You can write all the prose, or create all the diagrams you want, describing how a class should behave and what it looks like, but nothing is as real as a set of tests. The former is a wish list, but the tests are a contract that is enforced by the compiler and the running program. It's hard to imagine a more concrete description of a class than the tests

- Cuando escriba las pruebas invierta su escala de valores: *Alégrese cuando una prueba descubre un error*. Es peor que el error esté ahí y no sea descubierto.
- Que comprobar:
 - Convertir cualquier fallo que encontremos en una prueba
 - Comprobar el funcionamiento en mas de una plataforma
 - Dar valores de entrada erróneos o en el límite. Por ejemplo:
 - ◊ Los valores máximo y mínimo posibles
 - ◊ Cadenas vacías
 - ◊ Cadenas multilínea
 - ◊ Entradas con caracteres de control, etc.
 - ◊ Valores como '0', '0E0', listas vacías, hashes vacíos, etc.
 - ◊ Llamar sin argumentos a una rutina que los espera y viceversa
 - ◊ Llamar a una rutina con mas argumentos de los que espera
 - ◊ Llamar a una rutina con los argumentos en orden incorrecto
 - Estudie las interacciones y dependencias entre recursos que se da por sentado que están (pero que a veces pueden no estar. Por ejemplo, un fichero con cuya existencia se cuenta)
 - Estudie la dependencia de versiones del software instalado
 - Compruebe que toda subrutina se prueba al menos una vez. Prepare para cada subrutina importante al menos una prueba que cubra el caso promedio y otra que compruebe su funcionamiento en casos límite.
 - Estudie la escalabilidad de la distribución con valores tendiendo al límite. Ficheros grandes, números grandes, etc.
- Cree un **Rakefile** con diferentes objetivos:
 - **test** para ejecutar las pruebas
 - **dist** para crear un **tar.gz** con todos los ficheros
 - **zip** para crear un **zip** con todos los ficheros
 - **clean** para borrar ficheros temporales

15.17. Repaso

1. ¿Cuanto vale **self** en las líneas 5-8 cuando **my_attr_reader** es llamado desde la línea 14?

```
MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano$ cat -n myAccessors.rb
 1  class Module
 2    def my_attr_reader(*syms)
 3      syms.each { |s|
 4        class_eval %{
 5          def #{s}
 6            @#{s}
```

```

7           end
8       }
9   }
10  end
11 end
12
13 class Chuchu
14   my_attr_reader :a, :b
15
16   def initialize(a,b)
17     @a, @b = a, b
18   end
19 end

```

2. Rellene las partes que faltan en el código de la clase `Quiz` que implementa un DSL para la escritura de cuestionarios. Comente el código.

```

class Quiz
  RIGHT = :right
  WRONG = :wrong

  attr_accessor :name, :-----
  def initialize(name, &block)
    self.name = name
    self.questions = --
    @counter = 0
    ----- &block
  end

  def question(text, answers)
    q = Question.new(text, answers)
    questions << q
    @----- = -
  end

  def to_s
    out = <<"EOQUIZ"
    #{self.name}

    #{self.questions.----("\n")}
    EOQUIZ
  end

  def wrong
    @counter += 1
    [-----]
  end

  def right
    @counter+= 1
    [-----]
  end

```

```

end

def run
  counter=0
  puts self.name+"\n\n"
  self.questions.each { |q| counter += 1 if q.ask }
  puts "#{counter} respuestas correctas de un total de #{@questions.size}."
end

```

3. Rellene las partes que faltan en el código de la clase `Question` que implementa los objetos `question` para el DSL visto en clase para la escritura de cuestionarios. Comente el código.

```

class Question
  ORDER = 0
  KIND = 1
  attr_accessor :text, :answers

  def initialize(text, answers)
    @text = text
    @answers = answers.map { |k, v| ----- }.sort
  end

  def to_s
    output = <<"EORECIPE"
#{-----}

#{{
    out = ""
    @answers.each do |answer|
      out << " #{answer}\n"
    end
    ---
  }
EORECIPE
  end

  def ask
    begin
      puts self
      print "Su respuesta: "
      answerno = gets.to_i - 1
    end while (answerno < 0 or answerno >= @-----)
    @answers[-----].is_right?
  end

end

```

4. Rellene las partes que faltan en el código de la clase `Answer` que implementa las respuestas en el DSL para la escritura de cuestionarios. Comente el código.

```

class Answer
  attr_accessor :-----, :-----, :-----

```

```

def initialize(order, kind, answer)
  @kind, @order, @answer = kind, order, answer
end

def to_s
  "#{@order} - #{@answer}"
end

def is_right?
  -----
end

def ___(other)
  -----
end

```

5. Complete las partes que faltan en el siguiente código que implementa el método `prun` de la práctica *Procesos Concurrentes*. Comente el programa.

```

#!/usr/__/_/env ___. -w

def deb(m)
  puts "#{ $$ }: #{ m }" if $DEBUG
end

def getResult(chan)
  Marshal.___(chan.___)
end

def prun(p)
  pid, name, read = {}, {}, {}
  p.each { |n, code|
    read[n], write = IO.___
    pid[n] = fork do
      read[n]._____
      res = _____(n)
      write.syswrite _____(res)
    end # fork
    write._____
  }
  name = pid.invert
  result = []
  p.length.times {
    i, s = Process._____
  }

```

```

n = ----

result[n] = getResult(read[n])
deb "Finished process '----' pid=---- with status #{s} result = -----"
}

-----
end

res = prun(
  :one     => lambda { |n| "#{n} is 1" },
  :three   => lambda { |n| "#{n} is 3" },
  'two'    => lambda { |n| n*4 }
)

```

6. Complete las partes que faltan en el siguiente código de la práctica *Ordenar por Calificaciones* que implementa la ordenación de un ficheros de notas con el formato:

Ferrer Pérez, Eduardo & 9'6\\

Se ordenan en orden decreciente de calificaciones. Comente el programa.

```

~/rubytesting$ cat -n calif.rb
 1 f = File.open('notas')
 2 x = f.-----
 3
 4 y = x.map { |alu| alu =~ /-----/; ----- }
 5 h = -----[-----]
 6 h.keys.each { |alu| h[alu].gsub!(/-----/, '-----'); h[alu] = h[alu].----- }
 7
 8 s = h.sort { |a,b| ----- <=> ----- }
 9 s.each { |x| puts "-----" }

```

7. Complete las partes que faltan en el siguiente código de la práctica *La Calculadora* que implementa una calculadora extensible de expresiones en postfijo que admiten variables y asignación. Comente el programa.

```

1 class PostfixCalc
2
3   attr_reader :stack
4   attr_accessor :st
5
6   def initialize(f)
7     @f = {
8       '+' => lambda { |a,b| a+b },
9       '*' => lambda { |a,b| a*b },
10      '-' => lambda { |a,b| a-b },
11      '/' => lambda { |a,b| a/b },
12      'p' => lambda { ----- },
13      '!' => lambda { ----- },
14      '=' => lambda { ----- },
15    }
16    @f.----!(f)
17    @st = -----
18    @stack = --

```

```

19    end
20
21  def eval(expr)
22    expr.gsub!(/-----/, '\1 \2') # Se admite a= y b!
23    expr.split(/__/.each do |x|
24      case x
25        when *@f.keys
26          f = -----
27          ops = @stack.pop -----
28          b = f.call(-----)
29          @stack.push __ if __
30        when /^-?\d+(\.\d+)?$/
31          @stack.push -----
32        when /^[a-zA-Z_]\w*$/
33          -----
34        else
35          -----
36      end
37    end # each
38  end
39 end
... ...

```

8. Complete las partes que faltan en el siguiente fragmento de código que implementa la práctica *Conjuntos* usando arrays. Comente el programa.

```

1  class Set2
2
3  attr_accessor :sep
4  attr_reader :a
5  protected :a
6
7  def initialize(a)
8    @sep = ', '
9    @a = a.uniq.-----
10 end
11
12 def to_s
13   -----(@sep)
14 end
15
16 def length
17   -----
18 end
19
20 alias cardinal -----
21
22 def +(o)
23   Set2.new(-----)
24 end
25
26 def ^(o) # intersección
27   Set2.new(-----)
28 end

```

```

29
30     def -(o)
31         Set2.new(-----)
32     end
33
34     def include?(x)
35         -----
36     end
37
38     def <(s)
39         if @a.length < s.a.length
40             return true if @a.all? { |m| ----- }
41         end
42         return false
43     end
44
45
46     def ==(s)
47         -----
48     end
49
50     def <=(s)
51         -----
52     end
53
54
55
56
57
58
59
60

```

9. ¿Cuál es la diferencia entre `dup` y `clone`?
10. ¿Cómo se llama el método que nos permite declarar un método de clase como privado?
11. ¿Qué hace El método `const_set` de la clase `Module`?
12. ¿Qué retorna el método `fork`? ¿Qué argumentos lleva?
13. ¿Qué retorna el método `pipe`? ¿En qué clase está?
14. ¿Qué hace el método `wait2`?
15. ¿Qué hace el método `exit!`?
16. ¿Qué es un proceso zombi?
17. ¿Qué diferencias hay entre `syswrite` y `puts`?
18. ¿Cómo funciona el método `popen`? (véase por ejemplo ??)

```
    popen(cmd_string, mode="r" ) {|io| block } => obj
```

19. ¿Qué es YAML? ¿Para qué sirve? (véase por ejemplo ??)
 20. ¿Cuál es el resultado?
- ```

>> ww ||= 8
=>
>> ww ||= 9
=>

```
21. Complete el código que falta. La clase `SerializableProc` implementa bloques de código serializables (véase ??):

```

class SerializableProc

 def initialize(block)
 @block = block
 @block.sub!(/~/,'lambda ') unless @block =~/^\\s*(?:lambda|proc)/
 @func = eval "#{@block}"
 end

 def call(* args)
 -----(* args)
 end

 def arity
 @func.arity
 end

 def ----(limit)

 end

 def ----.----(s)
 ----- (s)
 end
end

```

22. ¿Cuales son las salidas?

```

>> def tutu(n,m, &b) b.call() end
=> nil
>> tutu 2, { :a =>1, :b =>2} { puts "hello" }


```

```

>> tutu(2,{:a =>1, :b =>2}) { puts "hello" }


```

23. ¿Que hace el método `extend` cuando es llamado en un objeto? ¿Que argumentos recibe? ¿Cual es la salida de estas sentencias? (véase ??)

```

>> module Chuchu
>> def hello; "hello" end
>> end
=> nil
>> s = {}
=> {}
>> s.extend Chuchu
=> {}
>> s.hello
=>

```

24. ¿Cual es el peligro de usar `eval` en un entorno distribuído?

25. Comente el siguiente programa y explica su comportamiento:

```

def multiplier(n)
 lambda do |*arr|
 arr.collect { |i| i*n }
 end
end

doubler = multiplier(2)
puts doubler[1, 2, 3]

eval("n = 3", doubler.binding)
puts doubler.call(1, 2, 3)

eval("n = 5", doubler)
puts doubler.call(1, 2, 3)

```

26. ¿Que diferencias hay entre `instance_eval`, `class_eval` y `eval`?

27. ¿Que es un DSL? ¿Que es un DSL interno o empotrado?

Repase los ejercicios 10.9, 10.8.9, 10.10, y 13.11.

## 15.18. Repaso

- ¿Cuál es la salida de este programa?

```

~/rubytesting$ cat -n closure1.rb
 1 def greetings
 2 x = "Goodbye"
 3 yield
 4 end
 5
 6 x = "Hello"
 7 puts greetings { "#{x} world" }

```

- ¿Cuál es la salida de este programa?

```

~/rubytesting$ cat -n blocks1_8.rb
 1 def chuchu
 2 yield 2
 3 end
 4
 5 x = 1
 6 chuchu { |x| }
 7 puts x
~/rubytesting$ ruby -v
ruby 1.8.7 (2010-01-10 patchlevel 249) [universal-darwin10.0]
~/rubytesting$ ruby blocks1_8.rb

```

- ¿Cuál es la salida de este programa?

```

~/rubytesting$ cat -n blocks1_8.rb
 1 def chuchu
 2 yield 2

```

```

3 end
4
5 x = 1
6 chuchu { |x| }
7 puts x
~/rubytesting$ sudo rvm install 1.9.2
...
Install of ruby-1.9.2-p290 - #complete
~/rubytesting$
~/rubytesting$ rvm list

rvm rubies

ruby-1.9.2-p290 [x86_64]

~/rubytesting$ rvm ruby-1.9.2-p290
~/rubytesting$ ruby -v
ruby 1.9.2p290 (2011-07-09 revision 32553) [x86_64-darwin10.8.0]
~/rubytesting$ ruby blocks1_8.rb

```

RVM is a command-line tool which allows you to easily install, manage, and work with multiple ruby environments from interpreters to sets of gems.

- Dado el programa:

```

~/rubytesting$ cat -n scopes.rb
1 v1 = 1
2
3 class Tutu
4 v2 = 2
5 puts "%2d"%(__LINE__.to_s)+' '+local_variables.inspect
6
7 def my_method
8 v3 = 3
9 puts "%2d"%__LINE__.to_s+' '+local_variables.inspect
10 end
11
12 puts "%2d"%__LINE__.to_s+' '+local_variables.inspect
13 end
14
15 obj = Tutu.new
16 obj.my_method
17 obj.my_method
18 puts "%2d"%__LINE__.to_s+' '+local_variables.inspect

```

¿Cuál es la salida de este programa?

```

~/rubytesting$ ruby scopes.rb
5
12
9
9
18

```

- Dado el programa:

```
~/rubytesting$ cat -n instance_scope.rb
 1 @var = "hello"
 2
 3 class Tutu
 4 puts "Inside Tutu: #{@var}"
 5 end
 6
 7 puts @var
```

¿Cuál es la salida de esta ejecución?

```
~/rubytesting$ ruby -w instance_scope.rb
```

- Dado el programa:

```
~/rubytesting$ cat -n scopes2.rb
 1 v1 = 1
 2
 3 Tutu = Class.new do
 4 v2 = 2
 5 puts "%2d"%(__LINE__.to_s)+ ' +local_variables.inspect
 6
 7 def my_method
 8 v3 = 3
 9 puts "%2d"%__LINE__.to_s+ ' +local_variables.inspect
10 end
11
12 puts "%2d"%__LINE__.to_s+ ' +local_variables.inspect
13 end
14
15 obj = Tutu.new
16 obj.my_method
17 obj.my_method
18 puts "%2d"%__LINE__.to_s+ ' +local_variables.inspect
```

¿Cuál es la salida de este programa?

```
~/rubytesting$ ruby scopes2.rb
5
12
9
9
18
```

- Dado el programa:

```
~/rubytesting$ cat -n scopes3.rb
 1 v1 = 1
 2
 3 Tutu = Class.new do
```

```

4 v2 = 2
5 puts "%2d%(_LINE_.to_s)+' '+local_variables.inspect
6
7 define_method :my_method do
8 v3 = 3
9 puts "%2d%(_LINE_.to_s)+' '+local_variables.inspect
10 end
11
12 puts "%2d%(_LINE_.to_s)+' '+local_variables.inspect
13 end
14
15 obj = Tutu.new
16 obj.my_method
17 obj.my_method
18 puts "%2d%(_LINE_.to_s)+' '+local_variables.inspect

```

¿Cuál es la salida de este programa?

```

~/rubytesting$ ruby scopes3.rb
5
12
9
9
18

```

- ¿En que clase está definido `define_method`?
- ¿Que visibilidad tiene `define_method`? (público, privado, protegido)
- ¿En que clase queda definida el método creado por `define_method`?
- ¿Es posible llamar a `define_method` con un receptor explícito?

`Tutu.define_method`

- Dado el programa:

```

~/rubytesting$ cat -n shared_scope2.rb
1 def define_methods
2 shared = 5
3
4 Kernel.send :define_method, :counter do
5 shared
6 end
7
8 Kernel.send :define_method, :inc do |x|
9 shared += x
10 end
11 end
12
13 define_methods
14
15 puts counter
16 inc(4)
17 puts counter

```

- ¿Cuál es la salida de este programa?
  - ¿Por que no puede llamarse a `define_method` en vez de `Kernel.send define_method`?
  - ¿Dónde (en que clase/objeto) estamos incorporando los métodos `counter` e `inc`?
  - ¿A que objeto/clase esta siendo enviado el mensaje `counter` en la línea 15?
- ¿Cuál es la salida?

```
MacBook-Air-de-casiano:rubystarting casiano$ cat -n define_method.rb
 1 class A
 2 def fred
 3 puts "In Fred"
 4 end
 5 def create_method(name, &block)
 6 self.class.send(:define_method, name, &block)
 7 end
 8 define_method(:wilma) { puts "Charge it!" }
 9 end
10 class B < A
11 define_method(:barney, instance_method(:fred))
12 end
13 a = B.new
14 a.barney
15 a.wilma
16 a.create_method(:betty) { p self }
17 a.betty
```

`instance_method` es definido en `Module`. Retorna un *UnboundMethod*.

## 15.19. Repaso

1. Complete el siguiente programa Ruby que resuelve el reto Dropbox conocido como “El Problema de la Dieta”(partes en subrayado):

```
1 def solve(goal, values)
2 x = values._____
3
4 subsets = lambda do |k|
5 return { _____ } if k < 0
6 old_sums = subsets[____]
7
8 new_sums = ____
9
10 old_sums.keys.each do |p|
11 c = p + _____
12 new_sums[c] = _____ if c <= goal
13 end
14
15 return old_sums._____ (new_sums)
16 end
17
18 z = subsets[values._____] [goal] || []
19 x.values_at(*z)
20 end
```

```

21
22 rdropbox = {
23 'free-lunch' => 802,
24
25 'playing-drums' => -295
26 }.invert
27
28 s = solve(0, rdropbox.keys)
29 puts s.inspect
30 puts rdropbox.values_at(*s).inspect

```

2. Escriba los nombres de los métodos para listar los nombres (como cadenas) de

- las variables globales que estén definidas (\_\_\_\_\_)
- de las variables locales que estén actualmente definidas (\_\_\_\_\_)
- de las variables de instancia de un objeto (\_\_\_\_\_)
- de las variables de clase o módulo (\_\_\_\_\_)
- y de las constantes de una clase o módulo (\_\_\_\_\_)

¿Cuales son métodos de instancia? ¿Cuáles de clase?

3. El código que sigue implementa un motor para las expresiones regulares. Para simplificar las expresiones, se han definido algunos operadores. Siguen algunos ejemplos de uso:

```

e = 'c'.re - +('d'.re | 'r'.re) # /c(d|r)+/
s = 'crddf'
remaining = e[s] # s =~ /c(d|r)+/
puts "/c(d|r)+/" '#{s}' Matched. Remaining = '#{remaining}'" if remaining
 # /c(d|r)+/'crddf' Matched. Remaining = 'f'
e = 'c'.re - ~('d'.re | 'r'.re) # /c(d|r)*/
s = 'cdrdrf'
remaining = e[s] # s =~ /c(d|r)*/
puts "/c(d|r)/*/" '#{s}' Matched. Remaining = '#{remaining}'" if remaining

```

Rellene el código que falta:

```

1 class ----
2
3 def -(right) # concatenacion
4 lambda { |x| _ _ self[x] ___ right[_] }
5 end
6
7 def |(right) # or
8 lambda { |x| self[x] -- -----[x] }
9 end
10
11 def epsilon # palabra vacía
12 lambda {|x| _ }
13 end
14
15 def ~ # cierre de Kleene *
16 lambda { |x| (-----)[x] or self.-----[x] }
17 end
18

```

```

19 def +@ # cierre positivo
20 lambda { |x| (-----)[x] }
21 end
22
23 end # Proc
24
25 class -----
26 def re
27 lambda { |x| x[0,self._____] == _____ and x[self._____...____] }
28 end
29 end

```

4. ¿Que predicado debo de usar para saber si la llamada al método actual es seguida de un bloque?
5. ¿Que hace el método `send`? ¿Que argumentos lleva? ¿que retorna?
6. ¿Cómo puedo llamar a un método privado en Ruby desde fuera de la clase?
7. ¿Que visibilidad tiene `define_method`? ¿Cómo funciona? ¿Que argumentos recibe? ¿Que devuelve? ¿En que contexto espera ser evaluado?
8. ¿Es posible definir en Ruby un método cuyo nombre no case con la expresión regular `^ [a-zA-Z_]+\w*$/i`?
9. ¿Cuándo se ejecuta `method_missing`? ¿Que argumentos espera? ¿Que precauciones hay que tomar cuando se escribe uno?
10. ¿Que se entiende por delegación?

```

1 require 'delegate'
2 class Set < DelegateClass(Array)

```

- ¿Que ocurre en la línea 2? ¿Que debe hacerse en el `initialize` para que el proceso de delegación funcione?
11. ¿Cómo puedo recorrer los objetos (referencia) que estan vivos en un punto dado de la ejecución de un programa?
  12. ¿Como puedo obtener los nombres de los métodos de un objeto?
  13. ¿Como funciona el método de instancia `index` de un objeto de la clase `Array`?
  14. ¿Cómo puedo llamar a un método cuyo nombre esta almacenado en la variable `x`?
  15. ¿Cómo puedo saber cuantos argumentos espera un método cuyo nombre esta almacenado en la variable `x`?
  16. ¿Cuales son los pasos para sobreescribir un método `mm` de una clase o módulo `MM` por nuestro propio `mm`?
  17. Rellene las partes de código que faltan en esta clase que implementa una clase que da una funcionalidad parecida a la de la librería `markby`:

```

class HTML

attr_accessor :page

def initialize(&block)
 @page = -----

```

```

----- &block
end

def build_attr(attributes)
 return '' if attributes.nil? or attributes.____?
 attributes.____(____) { |s,x| s += %{ ----- } }
end

def method_missing(tag, *args)
 if block_given?
 @page.____ -----

 text = @page._____
 else
 text = args._____| | "\n"
 end
 tagattr = build_attr(args.shift)
 text = "-----"
 @page[____].push ----
 text
end

def to_s
 @page.join("\n")
end

```

18. Explique brevemente que hacen los siguientes comandos:

- a) git init
- b) git add
- c) git commit
- d) git clone ssh://..
- e) git status
- f) git diff oldref newref
- g) git push
- h) git pull

- 19.
- a) ¿Que paquete hay que cargar **require** '-----' para elaborar una prueba?
  - b) ¿De que clase debe heredar nuestra clase para que pueda ser usada en la implementación de pruebas **Test::Unit**?
  - c) ¿Que patrón debe seguir el nombre de los métodos de prueba dentro de la clase de prueba?
- 20.
- a) ¿Cómo se declara una tarea en **Rake**?
  - b) ¿Cómo se declara que una tarea tiene dependencias?
  - c) ¿Cómo se definen las acciones asociadas que componen una tarea?
  - d) Cada tarea se define en un lugar del fichero **Rakefile** ¿cierto o falso?
  - e) ¿Cómo se declara una tarea *fichero*? ¿Como se definen sus dependencias? ¿Cómo se declara una tarea *directorio*?
  - f) ¿Que hace el método **sh**?

- g) ¿Como se llama la clase que nos facilita la tarea de ejecutar un conjunto de pruebas? ¿Que hacen los métodos `libs` y `pattern` del objeto `t` en el siguiente código?

```
Rake::TestTask.new do |t|
 t.libs << "test"
 t.test_files = FileList['test/test*.rb']
 t.verbose = true
end
```

## 15.20. Referencias. Véase También

1. Metaprogramming in Ruby en rubylearning.com
2. Ruby for Newbies: Missing Methods en <http://net.tutsplus.com>
3. Muse is a Ruby DSL for making music, it creates a WAV file. GitHub source
4. Simple Academia DSL in Ruby Tutorial. NOV 20 2010 by Shahrier Akram gdakram

# Capítulo 16

## Pruebas Unitarias

### 16.1. Test/Unit

*Unit testing* is a great way to catch errors early in the development process, if you dedicate time to writing appropriate and useful tests. As in other languages, Ruby provides a framework in its standard library for setting up, organizing, and running tests called **Test/Unit**.

**Test/Unit** provides three basic functionalities:

1. A way to define basic pass/fail tests.
2. A way to gather like tests together and run them as a group.
3. Tools for running single tests or whole groups of tests.

#### 16.1.1. Ejemplo Sencillo de uso de test/unit

```
[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ tree
.
|-- Rakefile
|-- lib
| '-- simple_number.rb
`-- test
 '-- tc_simple_number.rb

2 directories, 3 files

[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ cat lib/simple_number.rb
File: simple_number.rb

class SimpleNumber

 def initialize(num)
 raise unless num.is_a?(Numeric)
 @x = num
 end

 def add(y)
 @x + y
 end

 def multiply(y)
 @x * y
 end

```

```
end
```

Véase la documentación en Test/Unit

```
[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ cat test/tc_simple_number.rb
File: tc_simple_number.rb
```

```
require "simple_number"
require "test/unit"

class TestSimpleNumber < Test::Unit::TestCase

 def test_simple
 assert_equal(4, SimpleNumber.new(2).add(2))
 assert_equal(6, SimpleNumber.new(2).multiply(3))
 end
```

```
end
```

```
[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ cat Rakefile
desc "Test class SimpleNumber"
task :test do
 sh "ruby -Ilib -Itest test/tc_simple_number.rb"
end
```

```
[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ rake test
ruby -Ilib -Itest test/tc_simple_number.rb
Run options:
```

```
Running tests:
```

```
.
```

```
Finished tests in 0.000530s, 1886.7925 tests/s, 3773.5849 assertions/s.
```

```
1 tests, 2 assertions, 0 failures, 0 errors, 0 skips
```

Failures are when your test cases fail - i.e. your assertions are incorrect. Errors are unexpected errors that occur while trying to actually run the test - exceptions, etc.

### 16.1.2. Cuando una assertion falla

```
[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ cat test/tc_simple_number2.rb
File: tc_simple_number2.rb
```

```
require "simple_number"
require "test/unit"

class TestSimpleNumber < Test::Unit::TestCase

 def test_simple
 assert_equal(4, SimpleNumber.new(2).add(2))
 assert_equal(4, SimpleNumber.new(2).multiply(2))
 end
```

```

def test_typecheck
 assert_raise(RuntimeError) { SimpleNumber.new('a') }
end

def test_failure
 assert_equal(3, SimpleNumber.new(2).add(2), "Adding doesn't work")
end

end

[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ cat Rakefile
desc "Test class SimpleNumber"
task :test do
 sh "ruby -Ilib -Itest test/tc_simple_number.rb"
end

desc "Test with test/tc_simple_number2.rb. Expect failures"
task :test2 do
 sh "ruby -Ilib -Itest test/tc_simple_number2.rb"
end

[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ rake test2
ruby -Ilib -Itest test/tc_simple_number2.rb
Run options:

Running tests:

F..

Finished tests in 0.001706s, 1758.4994 tests/s, 2344.6659 assertions/s.

1) Failure:
test_failure(TestSimpleNumber) [test/tc_simple_number2.rb:18]:
Adding doesn't work.
<3> expected but was
<4>.

3 tests, 4 assertions, 1 failures, 0 errors, 0 skips
rake aborted!
Command failed with status (1): [ruby -Ilib -Itest test/tc_simple_number2.r...]
/Users/casiano/local/src/ruby/LPP/unit_testing/simple_example/Rakefile:8:in `block in <top (re...
/Users/casiano/.rvm/gems/ruby-1.9.3-p392/bin/ruby_noexec_wrapper:14:in `eval'
/Users/casiano/.rvm/gems/ruby-1.9.3-p392/bin/ruby_noexec_wrapper:14:in `<main>'
Tasks: TOP => test2
(See full trace by running task with --trace)

```

1. Now there are three tests (three member functions) in the class.
2. The function `test_typecheck` uses `assert_raise` to check for an exception.
3. The function `test_failure` is set up to fail, which the Ruby output happily points out, not only telling us which test failed, but how it failed (`expected <3> but was <4>`).
4. On this assertion, we've also added an final parameters which is a custom error message. It's strictly optional but can be helpful for debugging.

5. All of the assertions include their own error messages which are usually sufficient for simple debugging.

### 16.1.3. Organizando las pruebas

1. Tests for a particular unit of code are grouped together into a *test case*, which is a subclass of `Test/Unit/TestCase`.
2. `Assertions` are gathered in *tests*, member functions for the test case whose names start with `test_`.
3. When the test case is executed or required, `Test/Unit` will iterate through all of the tests (finding all of the *member functions* which start with `test_` using reflection) in the test case, and provide the appropriate feedback.
4. Test case classes can be gathered together into test suites which are Ruby files which require other test cases:

```
File: ts_allTheTests.rb
require 'test/unit'
require 'testOne'
require 'testTwo'
require 'testThree'
```

In this way, related test cases can be naturally grouped.

5. Further, test suites can contain other test suites, allowing the construction of a hierarchy of tests.
6. This structure provides relatively fine-grained control over testing.
  - a) Individual tests can be run from a test case
  - b) a full test case can be run stand-alone,
  - c) a test suite containing multiple cases can be run,
  - d) or a suite of suites can run, spanning many test cases.

#### Convenciones con respecto a los Nombres

The author of `Test/Unit`, Nathaniel Talbott, suggests starting the names of test cases with `tc_` and the names of test suites with `ts_`

**Ejecución de pruebas específicas** It's possible to run just one (or more) tests out of a full test case:

```
>> ruby -w tc_simpleNumber2.rb --name test_typecheck
Loaded suite tc_simpleNumber2
Started
.
Finished in 0.003401 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

It is also possible to run all tests whose names match a given pattern:

```
>> ruby -w tc_simpleNumber2.rb --name /test_type*/
Loaded suite tc_simpleNumber2
Started
.
Finished in 0.003401 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

## Opciones en Línea de Comandos

```
unittestingpoint/examples(master)]$ ruby -I. tc_point_1.rb --help
Usage: tc_point_1 [options]
minitest options:
-h, --help Display this help.
-s, --seed SEED Sets random seed
-v, --verbose Verbose. Show progress processing files.
-n, --name PATTERN Filter test names on pattern.
--jobs-status [TYPE] Show status of jobs every file; Disabled when --jobs isn't specified
-j, --jobs N Allow run tests with N jobs at once
--separate Restart job process after one testcase has done
--retry Retry running testcase when --jobs specified
--no-retry Disable --retry
--ruby VAL Path to ruby; It'll have used at -j option
-q, --hide-skip Hide skipped tests
--show-skip Show skipped tests
--color[=WHEN] colorize the output. WHEN defaults to 'always' or can be 'never' or 'auto'.
 force to output tty control. WHEN defaults to 'yes' or can be 'no'.
--tty[=WHEN]
 Base directory of test suites.
 Exclude test files on pattern.
-Idirectory Add library load path
--[no-]gc-stress Set GC.stress as true
```

### 16.1.4. Setup y Teardown

There are many cases where a small bit of code needs to be run before and/or after each test. Test/Unit provides the `setup` and `teardown` *member functions*, which are run before and after every test (member function).

```
[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ cat test/tc_simple_number3.rb
File: tc_simple_number3.rb

require "simple_number"
require "test/unit"

class TestSimpleNumber < Test::Unit::TestCase

 def setup
 @num = SimpleNumber.new(2)
 end

 def teardown
 ## Nothing really
```

```

end

def test_simple
 assert_equal(4, @num.add(2))
end

def test_simple2
 assert_equal(4, @num.multiply(2))
end

end

[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ cat Rakefile
desc "Test class SimpleNumber"
task :test do
 sh "ruby -Ilib -Itest test/tc_simple_number.rb"
end

desc "Test with test/tc_simple_number2.rb. Expect failures"
task :test2 do
 sh "ruby -Ilib -Itest test/tc_simple_number2.rb"
end

desc "Test with test/tc_simple_number3.rb. setup adn teardown"
task :test3 do
 sh "ruby -Ilib -Itest test/tc_simple_number3.rb"
end

[~/local/src/ruby/LPP/unit_testing/simple_example(master)]$ rake test3
ruby -Ilib -Itest test/tc_simple_number3.rb
Run options:

Running tests:

..

```

Finished tests in 0.000988s, 2024.2915 tests/s, 2024.2915 assertions/s.

2 tests, 2 assertions, 0 failures, 0 errors, 0 skips

### 16.1.5. Veáse

1. Pruebas Unitarias para la clase Point vista en la sección *Definiendo Operadores (La Clase Point)* 14.1.6 en GitHub
2. Documentación de Test/Unit
3. Ruby Programming/Unit testing en WikiBooks

## 16.2. Minitest

1. Documentación de MiniTest
2. Documentación de MiniTest/Assertions

## Capítulo 17

# Pruebas. Test Driven Development (TDD) y Rspec

*Test-Driven Development (TDD)* is a practice designed to force developers to think about their code before writing it (Kent Beck).

This is done in small increments in which the developer must identify increasingly more-complex interfaces to other objects as well as build up the functionality of the object under development.

Here's the flow for development with RSpec:

1. You write a test. This test describes the behavior of a small element of your system.
2. You run the test. The test fails because you have not yet built the code for that part of your system. This important step tests your test case, verifying that your test case fails when it should.
3. You write enough code to make the test pass.
4. You run the tests and verify that they pass.

In essence, a RSpec developer turns test cases from *red (failing)* to green (passing) all day.

### Vídeos sobre TDD

- Testing with RSpec course at Code School (recomendado)
- Test-Driven Development in Ruby Video Tutorials
- Test-Driven Development in Ruby - First Steps -Waterfall-vs-Agile

## 17.1. Introducción

Puede encontrar el código de esta sección en Github: <https://github.com/crguezl/rspec-rpcalc>

### 17.1.1. 0: Red

```
/Users/casiano/local/src/ruby/LPP/rspec_examples/rpccalculator/
|~lib/
| ‘~math/
| ‘-rpcalc.rb
|~spec/
| ‘~math/
| ‘-rpcalc_spec.rb
|-Rakefile
‘-README.md
```

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator]$ cat spec/math/rpcalc_spec.rb
module Math
 describe RPCalc do
 end
 end
```

We use the `describe( )` method to define an example group. The class or string we pass to it represents the facet of the system that we want to `describe` (a new account). The block holds the code examples that make up that group.

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator]$ rspec spec/math/rpcalc_spec.rb
/Users/casiano/local/src/ruby/LPP/rspec_examples/rpcalculator/spec/math/rpcalc_spec.rb:2:
in '<module:Math>': uninitialized constant Math::RPCalc (NameError)
```

### 17.1.2. 1: Green

```
/Users/casiano/local/src/ruby/LPP/rspec_examples/rpcalculator/
|~lib/
| |~math/
| | |-rpcalc.rb
| |-rpcalc.rb
|~spec/
| |~math/
| | |-rpcalc_spec.rb
| |-spec_helper.rb
|-Rakefile
`-README.md
```

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(0)]$ cat Rakefile
desc 'run rspec tests'
task :spec do
 sh "rspec -Ilib -Ispec spec/math/rpcalc_spec.rb"
end
...
```

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat spec/math/rpcalc_spec.rb
#require File.join(File.dirname(__FILE__), "../spec_helper")
require "spec_helper"
module Math
 describe RPCalc do
 end
 end
```

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat spec/spec_helper.rb
#$: << File.join(File.dirname(__FILE__), "../../lib")
require "rspec"
require "rpcalc"
```

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat lib/math/rpcalc.rb
module Math
 class RPCalc
 end # class RPCalc
end # Math
```

```
[~/local/src/ruby/LPP/rspec_examples/rpccalculator(master)]$ cat lib/rpcalc.rb
require 'math/rpcalc'
```

```
[~/local/src/ruby/LPP/rspec_examples/rpccalculator(1)]$ rake spec
rspec -Ilib -Ispec spec/math/rpcalc_spec.rb
No examples found.
```

Finished in 0.00005 seconds

0 examples, 0 failures

### 17.1.3. 2: Red

```
[~/local/src/ruby/LPP/rspec_examples/rpccalculator(1)]$ cat spec/math/rpcalc_spec.rb
#require File.join(File.dirname(__FILE__), "../spec_helper")
require "spec_helper"
module Math
 describe RPCalc do
 before :each do
 @c = Math::RPCalc.new
 end

 context "When an erroneous input is given" do
 it "must raise an exception" do
 expect { @c.calc('a') }.to raise_error(SyntaxError)
 expect { @c.calc('a') }.to raise_error("Error. found 'a'. Expected number or operator")
 end
 end
 end
end
```

```
[~/local/src/ruby/LPP/rspec_examples/rpccalculator(1)]$ rake spec
rspec -Ilib -Ispec spec/math/rpcalc_spec.rb
```

Math::RPCalc

```
 When an erroneous input is given
 must raise an exception (FAILED - 1)
```

Failures:

- 1) Math::RPCalc When an erroneous input is given must raise an exception  
Failure/Error: expect { @c.calc('a') }.to raise\_error(SyntaxError)  
expected SyntaxError, got #<NoMethodError: undefined method 'calc' for #<Math::RPCalc:0  
# ./spec/math/rpcalc\_spec.rb:11:in 'block (4 levels) in <module:Math>'  
# ./spec/math/rpcalc\_spec.rb:11:in 'block (3 levels) in <module:Math>'  
# ./spec/math/rpcalc\_spec.rb:11:in 'block (3 levels) in <module:Math>'

Finished in 0.00196 seconds

1 example, 1 failure

Failed examples:

```
rspec ./spec/math/rpcalc_spec.rb:10 # Math::RPCalc When an erroneous input is given must raise
```

1. expect
2. raise\_error

### 17.1.4. 3: Green

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat lib/math/rpcalc.rb
module Math
 class RPCalc
 attr_accessor :stack

 def initialize()
 @stack = []
 end

 def calc(expr)
 @stack = []
 expr.split(/\s+/).each do |x|
 case x
 when '+', '*', '- ', '/'
 #
 when /^-?\s*\d+(\.\d+)?([eE][+-]?\d+)?\s*?/
 #
 else
 raise SyntaxError, "Error. found '#{x}'. Expected number or operator"
 end
 end
 end
 end # class RPCalc
end # Math
```

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ rake spec
rspec -Ilib -Ispec spec/math/rpcalc_spec.rb
```

```
Math::RPCalc
When an erroneous input is given
 must raise an exception
```

```
Finished in 0.0018 seconds
1 example, 0 failures
```

### 17.1.5. 4: refactor

*Refactoring* is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

Its heart is a series of small behavior preserving transformations. Each transformation (called a '*refactoring*') does little, but a sequence of transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is also kept fully working after each small refactoring, reducing the chances that a system can get seriously broken during the restructuring.

Martin Fowler

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat spec/math/rpcalc_spec.rb
#require File.join(File.dirname(__FILE__), "../spec_helper")
```

```

require "spec_helper"
module Math
 describe RPCalc do
 before :each do
 @c = Math::RPCalc.new
 end

 context "When an erroneous input is given" do
 before :each do
 @e = proc { @c.calc('a') }
 end
 it "must raise an exception" do
 expect { @e[] }.to raise_error(SyntaxError)
 expect { @e[] }.to raise_error("Error. found 'a'. Expected number or operator")
 end
 end
 end
end

```

```
[~/local/src/ruby/LPP/rspec_examples/rpccalculator(master)]$ rake spec
rspec -Ilib -Ispec spec/math/rpcalc_spec.rb
```

```
Math::RPCalc
When an erroneous input is given
 must raise an exception
```

```
Finished in 0.00184 seconds
1 example, 0 failures
[~/local/src/ruby/LPP/rspec_examples/rpccalculator(master)]$
```

### Las Cuatro Reglas de Kent Beck para un Diseño Simple

Use *Kent Beck's four rules of simple design* to guide you:

1. Run all the tests
2. Contain no duplicate code
3. Express all the ideas the author wants to express
4. Minimize classes and methods

By re-running the test cases, the developer can be confident that code refactoring is not damaging any existing functionality.

#### 17.1.6. 5: Red: should and should\_not

```
[~/local/src/ruby/LPP/rspec_examples/rpccalculator(master)]$ cat spec/math/rpcalc_spec.rb
#require File.join(File.dirname(__FILE__), "../spec_helper")
require "spec_helper"
module Math
 describe RPCalc do
 before :each do
 @c = Math::RPCalc.new
 end
 end

```

```

context "When an erroneous input is given" do
 before :each do
 @e = proc { @c.calc('a') }
 end
 it "must raise an exception" do
 expect { @e[] }.to raise_error(SyntaxError)
 expect { @e[] }.to raise_error("Error. found 'a'. Expected number or operator")
 end
end

context "When a correct input is given" do
 it "must compute the answer" do
 r = @c.calc('2 3 + 4 *')
 r.should eq 20
 @c.stack.should eq []
 end
end

end # RPCalc
end # Math

```

RSpec achieves a high level of expressiveness and readability by exploiting open classes in Ruby to add the methods `should()` and `should_not()` to every object in the system. Each method accepts either a *matcher* or a Ruby expression using a specific subset of Ruby operators.

A matcher is an object that tries to match against an expected outcome.

Let's take a look at an example using the equal matcher, which you can access through the method `eq`:

```
r.should eq 20
```

When the Ruby interpreter encounters this line, it begins by evaluating `eq 20`.

This is an RSpec method that returns a matcher object configured to match for equality with the value 20. The matcher then becomes the argument to `r.should`.

Behind the scenes, the `should( )` method calls `matcher.matches?`, passing `self` (the result object) as the argument. Because `should( )` is added to every object, it can be any object. Similarly, the matcher can be any object that responds to `matches?(object)`.

If `matches?(self)` returns `true`, then the expectation is met and execution moves on to the next line in the example.

If `matches?(self)` returns `false`, `should( )` asks the matcher for a failure message and raises an `ExpectationNotMetError` with that message.

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ rake spec
rspec -Ilib -Ispec spec/math/rpcalc_spec.rb
```

`Math::RPCalc`

```
When an erroneous input is given
 must raise an exception
When a correct input is given
 must compute the answer (FAILED - 1)
```

Failures:

- 1) `Math::RPCalc` When a correct input is given must compute the answer  
Failure/Error: `r = @c.calc('2 3 + 4 *)'`  
`SyntaxError:`

```

Error. found '2'. Expected number or operator
./lib/math/rpcalc.rb:12:in `block in calc'
./lib/math/rpcalc.rb:11:in `each'
./lib/math/rpcalc.rb:11:in `calc'
./spec/math/rpcalc_spec.rb:21:in `block (3 levels) in <module:Math>'

Finished in 0.00244 seconds
2 examples, 1 failure

```

Failed examples:

```

rspec ./spec/math/rpcalc_spec.rb:20 # Math::RPCalc When a correct input is given must compute
rake aborted!

```

### 17.1.7. 6: Green

```

[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat lib/math/rpcalc.rb
module Math
 class RPCalc
 attr_accessor :stack

 def initialize()
 @stack = []
 end

 def calc(expr)
 @stack = []
 expr.split(/\s+/).each do |x|
 case x
 when '+', '*', '- ', '/'
 op2 = @stack.pop
 op1 = @stack.pop
 @stack.push eval "(#{op1} #{x} #{op2})"
 when /^-?\s*\d+(\.\d+)?([eE][+-]?\d+)?\s*$/
 @stack.push x
 else
 raise SyntaxError, "Error. found '#{x}'. Expected number or operator"
 end
 end
 @stack.pop
 end
 end # class RPCalc
end # Math

```

```

[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat spec/math/rpcalc_spec.rb
#require File.join(File.dirname(__FILE__), "../spec_helper")
require "spec_helper"
module Math
 describe RPCalc do
 before :each do
 @c = Math::RPCalc.new
 end
 end

```

```

context "When an erroneous input is given" do
 before :each do
 @e = proc { @c.calc('a') }
 end
 it "must raise an exception" do
 expect { @e[] }.to raise_error(SyntaxError)
 expect { @e[] }.to raise_error("Error. found 'a'. Expected number or operator")
 end
end

context "When a correct input is given" do
 it "must give the correct answer for integer expressions" do
 r = @c.calc('2 3 + 4 *')
 r.should eq 20
 @c.stack.should eq []
 end

 it "must give the correct answer for float expressions" do
 r = @c.calc('4 3 + 2.5 3.5 +')
 r.should be_within(0.01).of(6.0)
 @c.stack.should eq [7]
 end
end

end # RPCalc
end # Math

```

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ rake spec
rspec -Ilib -Ispec spec/math/rpcalc_spec.rb
```

```

Math::RPCalc
When an erroneous input is given
 must raise an exception
When a correct input is given
 must give the correct answer for integer expressions
 must give the correct answer for float expressions

```

```
Finished in 0.00457 seconds
3 examples, 0 failures
```

### 17.1.8. 7: un poco mas tarde ...

```
[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat lib/math/rpcalc.rb
module Math
 class RPCalc
 attr_accessor :stack

 def initialize()
 @stack = []
 end

 def calc(expr)
 @stack = []
 expr.split(/\s+/).each do |x|

```

```

case x
when '+', '*', '- ', '/'
 op2 = @stack.pop or raise SyntaxError, "No first argument for '#{x}'"
 op1 = @stack.pop or raise SyntaxError, "No second argument for '#{x}'"
 @stack.push eval "(#{op1} #{x} #{op2})"
when /^-?\s*\d+(\.\d+)?([eE][+-]?\d+)?\s*?/
 @stack.push x
else
 raise SyntaxError, "Error. found '#{x}'. Expected number or operator"
end
@stack.pop
end

end # class RPCalc
end # Math

[~/local/src/ruby/LPP/rspec_examples/rpcalculator(master)]$ cat spec/math/rpcalc_spec.rb
#require File.join(File.dirname(__FILE__), "../spec_helper")
require "spec_helper"
module Math
 describe RPCalc do
 before :each do
 @c = Math::RPCalc.new
 end

 context "When an erroneous input is given" do
 context "not numeric input" do
 before :each do
 @e = proc { @c.calc('a') }
 end

 it "must raise an exception for not numeric input" do
 expect { @e[] }.to raise_error(SyntaxError)
 expect { @e[] }.to raise_error("Error. found 'a'. Expected number or operator")
 end
 end
 end

 context "not enough operands" do
 it "must raise an exception when not enough operands" do
 expect { @c.calc('2 3 + *') }.to raise_error(SyntaxError)
 expect { @c.calc('2 3 + *') }.to raise_error("No second argument for '*'")
 end
 end
 end

 context "When a correct input is given" do
 it "must give the correct answer for integer expressions" do
 r = @c.calc('2 3 + 4 *')
 r.should eq 20
 @c.stack.should eq []
 end

 it "must give the correct answer for float expressions" do

```

```

 r = @c.calc('4 3 + 2.5 3.5 +')
 r.should be_within(0.01).of(6.0)
 @c.stack.should eq [7]
 end
end

end # RPCalc
end # Math

```

[~/local/src/ruby/LPP/rspec\_examples/rpccalculator(master)]\$ rake spec  
rspec -Ilib -Ispec spec/math/rpcalc\_spec.rb

```

Math::RPCalc
When an erroneous input is given
 not numeric input
 must raise an exception for not numeric input
 not enough operands
 must raise an exception when not enough operands
When a correct input is given
 must give the correct answer for integer expressions
 must give the correct answer for float expressions

```

Finished in 0.00526 seconds  
4 examples, 0 failures

## 17.2. rspec opciones

```

[~/srcLPPclases/question-simple_choice(develop2)]$ rspec --help
Usage: rspec [options] [files or directories]

-I PATH Specify PATH to add to $LOAD_PATH
 (may be used more than once).
-r, --require PATH Require a file.
-O, --options PATH Specify the path to a custom options file.
--order TYPE[:SEED] Run examples by the specified order type.
 [defined] examples and groups are run
 in the order they are defined
 [rand] randomize the order of
 groups and examples
 [random] alias for rand
 [random:SEED] e.g. --order random:123
 Equivalent of --order rand:SEED.
 Abort the run on first failure.
 Do not abort the run on first failure.
 Override the exit code used when there
 are failing specs.
 Print the formatter output of your
 suite without
 running any examples or hooks
 Run examples via DRb.
 Port to connect to the DRb server.
 Initialize your project with RSpec.

-X, --[no-]drb Run examples via DRb.
 Port to connect to the DRb server.
 Initialize your project with RSpec.

```

\*\*\*\* Output \*\*\*\*

|                                              |                                                                                                                                                                                                                     |
|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-f, --format FORMATTER</code>          | Choose a formatter.<br><code>[p]rogress</code> (default - dots)<br><code>[d]ocumentation</code> (group and example names)<br><code>[h]tml</code><br><code>[j]son</code><br><code>custom formatter class name</code> |
| <code>-o, --out FILE</code>                  | Write output to a file instead of \$stdout.<br>This option applies to the previously specified --format, or the default format if no format is specified.                                                           |
| <code>--deprecation-out FILE</code>          | Write deprecation warnings to a file instead of \$stderr.                                                                                                                                                           |
| <code>-b, --backtrace</code>                 | Enable full backtrace.                                                                                                                                                                                              |
| <code>-c, --[no-]color, --[no-]colour</code> | Enable color in the output.                                                                                                                                                                                         |
| <code>-p, --[no-]profile [COUNT]</code>      | Enable profiling of examples and list the slowest examples (default: 10).                                                                                                                                           |
| <code>-w, --warnings</code>                  | Enable ruby warnings                                                                                                                                                                                                |

#### \*\*\*\* Filtering/tags \*\*\*\*

In addition to the following options for selecting specific files, groups, or examples, you can select a single example by appending the line number to the filename:

```
rspec path/to/a_spec.rb:37
```

|                                        |                                                                                                                                               |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-P, --pattern PATTERN</code>     | Load files matching pattern<br>(default: "spec/**/*_spec.rb").                                                                                |
| <code>--exclude-pattern PATTERN</code> | Load files except those matching pattern.<br>Opposite effect of --pattern.                                                                    |
| <code>-e, --example STRING</code>      | Run examples whose full nested names include STRING (may be used more than once)                                                              |
| <code>-t, --tag TAG[:VALUE]</code>     | Run examples with the specified tag, or exclude examples by adding ~ before the tag.<br>- e.g. ~slow<br>- TAG is always converted to a symbol |
| <code>--default-path PATH</code>       | Set the default path where RSpec looks for examples (can be a path to a file or a directory).                                                 |

#### \*\*\*\* Utility \*\*\*\*

|                            |                       |
|----------------------------|-----------------------|
| <code>-v, --version</code> | Display the version.  |
| <code>-h, --help</code>    | You're looking at it. |

### 17.3. Resumen y Enlaces

En resumen:

1. **Think:** Figure out what test will best move your code towards completion. (Take as much time as you need. This is the hardest step for beginners.)

2. **Red:** Write a very small amount of test code. Only a few lines... usually no more than five. Run the tests and watch the new test fail: the test bar should turn red. (This should only take about 30 seconds.)
3. **Green:** Write a very small amount of production code. Again, usually no more than five lines of code. Don't worry about design purity or conceptual elegance. Sometimes you can just hardcode the answer. This is okay because you'll be refactoring in a moment. Run the tests and watch them pass: the test bar will turn green. (This should only take about 30 seconds, too.)
4. **Refactor:** Now that your tests are passing, you can make changes without worrying about breaking anything.
  - a) Pause for a moment. Take a deep breath if you need to.
  - b) Then look at the code you've written, and ask yourself if you can improve it.
  - c) Look for duplication and other code smells."
  - d) If you see something that doesn't look right, but you're not sure how to fix it, that's okay.
  - e) Take a look at it again after you've gone through the cycle a few more times. (Take as much time as you need on this step.)
  - f) After each little refactoring, run the tests and make sure they still pass.
5. **Repeat:** Do it again. You'll repeat this cycle dozens of times in an hour. Typically, you'll run through several cycles (three to five) very quickly, then find yourself slowing down and spending more time on refactoring. Than you'll speed up again. 20-40 cycles in an hour is not unreasonable.

James Shore, The Art of Agile Programming

## Véase

- Puede encontrar el código de esta sección en Github: <https://github.com/crguezl/rspec-rpcalc>
- **RSpec::Matchers**
- Behavior-driven testing with RSpec. A comprehensive approach to test-driven development Bruce Tate (bruce@rapidred.com), CTO, WellGood LLC IBM developerWorks
- RSpec documentation
- Rspec Matchers
- RSpec Matchers by Charles Max Wood
- An Introduction to Rspec
- Rspec 101 by Jason Noble
- Ruby for Newbies: Testing with Rspec (screencast)
- Empezando con RSpec por jmbeas
- An Introduction To RSpec treehouse blog
- <http://www.slideshare.net/AlanHecht/rspec-and-rails>
- My Journey through Rspec by Carmen
- Testing Tuesday Series

- Testing Tuesday: Tests Make Software
- Testing Tuesday #2: From Test-Driven Development to Behavior-Driven Development
- Testing Tuesday #8: Behavior-Driven Integration and Unit Testing

# Capítulo 18

# Pruebas. Capybara y Cucumber

El material de este capítulo esta sacado en su mayoría del libro de Robbins Mathew [10].

## 18.1. Introducción

Capybara helps you test web applications by simulating how a real user would interact with your app. It is agnostic about the driver running your tests and comes with Rack::Test and Selenium support built in. Webkit is supported through an external gem.

Capybara starts a browser.

- This can be a real browser (Firefox via selenium-webdriver) or
- headless browser (like Phantomjs)

The good thing about it, is that you can use selenium backend while you work on your script, and switch over to Phantomjs when it's done.

This way, you will see what is going on while you do the hacking, and executes without opening any extra windows when you are done and want to use it.

## 18.2. Usando Capybara Directamente

### 18.2.1. Usando la Sesión Directamente

**Donde** Véase [crguezl/selenium-examples](#) en GitHub.

```
[~/sinatra/sinatra-selenium/intro(master)]$ pwd -P
/Users/casiano/local/src/ruby/sinatra/sinatra-selenium/intro
```

#### Ejemplo de Uso Directo de Capybara con Capybara::Session

La forma mas simple de usar Capybara es instanciar un objeto `Capybara::Session`. A partir de ese momento es posible llamar a los métodos del DSL capybara sobre dicho objeto:

```
[~/sinatra/sinatra-selenium/intro(master)]$ cat -n hello_lpp.rb
1 require 'capybara'
2 session = Capybara::Session.new(:selenium)
3 session.visit "http://nereida.deioc.ull.es/~lpp/perlexamples/"
4
5 if session.has_content?("Apuntes de RUBY")
6 puts "All shiny, captain!"
7 else
8 puts ":(" no tagline found, possibly something's broken"
9 exit(-1)
10 end
```

## Capybara::Session

La clase `Session` representa la interacción de un único usuario con el sistema.

La `Session` puede utilizar cualquiera de los drivers subyacentes.

Para inicializar manualmente una sesión podemos hacer algo como esto:

```
session = Capybara::Session.new(:culerity, MyRackApp)
```

La aplicación que se pasa como segundo argumento es opcional.

Cuando se ejecuta Capybara contra una página externa como en el ejemplo deberemos omitirla.

```
session = Capybara::Session.new(:culerity)
Culerity is a gem that integrates Cucumber and Celerity
(Celerity is a headless Java browser with JavaScript support)
in order to test your application's full stack
including Javascript.
session.visit('http://www.google.com')
```

`Session` provee métodos para controlar la navegación en la página como: `visit`, `switch_to_window` etc.

Esto permite la interacción con la página:

```
session.fill_in('q', :with => 'Capybara')
session.click_button('Search')
expect(session).to have_content('Capybara')
```

También delega algunos métodos a la clase `Capybara::Node::Document`, que representa al documento HTML actual.

El código de delegación puede verse en el fichero `lib/capybara/session.rb` de la distribución de Capybara:

```
NODE_METHODS = [
 :all, :first, :attach_file, :text, :check, :choose,
 :click_link_or_button, :click_button, :click_link, ...
]
@api private
DOCUMENT_METHODS = [:title, :assert_title, ...]
...

def document
 @document ||= Capybara::Node::Document.new(self, driver)
end

NODE_METHODS.each do |method|
 define_method method do |*args, &block|
 @touched = true
 current_scope.send(method, *args, &block)
 end
end

DOCUMENT_METHODS.each do |method|
 define_method method do |*args, &block|
 document.send(method, *args, &block)
 end
end
```

```

def inspect
 %(<Capybara::Session>)
end

def current_scope
 scopes.last || document
end
...
def scopes
 @scopes ||= [nil]
end

```

**El método visit** Una llamada `session.visit(url)` navega a la URL dada.  
La URL puede ser relativa o absoluta. La conducta depende del driver.

```

session.visit('/foo')
session.visit('http://google.com')

```

For drivers which can run against an external application, such as the selenium driver giving an absolute URL will navigate to that page.

**Capybara.app\_host** This allows testing applications running on remote servers. For these drivers, setting `Capybara.app_host` will make the remote server the default. For example:

```

Capybara.app_host = 'http://google.com'
session.visit('/') # visits the google homepage

```

### Capybara.always\_include\_port

If `Capybara.always_include_port` is set to true and this session is running against a rack application, then the port that the rack application is running on will automatically be inserted into the URL.

Supposing the app is running on port 4567, doing something like:

```
visit("http://google.com/test")
```

Will actually navigate to `google.com:4567/test`.

**Capybara con Otros Drivers** Capybara viene con dos drivers: Rack::Test y Selenium WebDriver. No obstante Capybara permite añadir otros drivers via plugins.

Entre las alternativas se encuentran:

- Capybara WebKit
- Capybara Poltergeist

### Poltergeist

Poltergeist is a *headless browser* driver for Capybara. You will need to install PhantomJS and make sure that `phantomjs` command is in your path first.

El siguiente Rakefile muestra como Instalar phantomjs en un Mac:

```
[~/sinatra/sinatra-selenium/intro(master)]$ cat Rakefile
desc "install phantomjs in a mac"
task :phantom do
 sh "brew update && brew install phantomjs"
end
```

A headless browser is a web browser without a graphical user interface. In other words it is a browser, a piece of software, that access web pages but doesn't show them to any human being. They're actually used to provide the content of web pages to other programs.

Este es el mismo ejemplo anterior pero usando capybara/poltergeist:

```
[~/sinatra/sinatra-selenium/intro(master)]$ cat -n hello_lpp_phantom.rb
 1 require 'capybara'
 2 require 'capybara/poltergeist'
 3
 4 session = Capybara::Session.new(:poltergeist)
 5
 6 session.visit "http://nereida.deioc.ull.es/~lpp/perlexamples/"
 7
 8 if session.has_content?("Apuntes de RUBY")
 9 puts "All shiny, captain!"
10 else
11 puts ":(" no tagline found, possibly something's broken"
12 exit(-1)
13 end
```

## Gemfile usado en los Ejemplos

```
[~/sinatra/sinatra-selenium/intro(master)]$ cat Gemfile
source 'https://rubygems.org'

gem 'capybara'
gem 'poltergeist'
```

### 18.2.2. Usando Capybara Directamente Incluyendo el Módulo Capybara::DSL

Cuando usamos/incluimos el módulo Capybara::DSL en la librería capybara/dsl, la Session es inicializada automáticamente.

```
[~/sinatra/sinatra-selenium/intro(master)]$ cat hello_lpp_dsl.rb
require 'capybara/dsl'
require 'capybara/poltergeist'
include Capybara::DSL

Capybara.default_driver = :poltergeist

visit "http://nereida.deioc.ull.es/~lpp/perlexamples/"

if has_content?("Apuntes de RUBY")
 puts "All shiny, captain!"
else
 puts ":(" no tagline found, possibly something's broken"
 exit(-1)
end
```

Al ejecutar este programa obtenemos un warning:

```
[~/sinatra/sinatra-selenium/intro(master)]$ ruby hello_lpp_dsl.rb
including Capybara::DSL in the global scope is not recommended!
All shiny, captain!
```

Si queremos silenciar este warning podemos usar la opción -W del intérprete Ruby estableciendo el nivel a 0:

```
[~/sinatra/sinatra-selenium/intro(master)]$ ruby -W0 hello_lpp_dsl.rb
All shiny, captain!
```

### 18.2.3. Aprendiendo Capybara con Ficheros HTML Locales

Para probar Capybara con ficheros HTML locales standalone usaremos el siguiente programa:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat testinglocalhtml.rb
#!/usr/bin/env ruby
This program evaluates a Capybara script against
a local HTML page
Usage:
testinglocalhtml index.html capybara_script.rb [poltergeist]
require "capybara"
require 'capybara/poltergeist'
require 'rspec/expectations'

exit(1) if ARGV.length < 2
html, capybara = ARGV[0,2]
driver = (ARGV[2] || :selenium).to_sym

File.open(html) do |h|
 html = h.read
end

File.open(capybara) do |c|
 capybara = c.read
end

app = proc { |env| [200, { "Content-Type" => "text/html" }, [html]] }

session = Capybara::Session.new(driver, app)

session.instance_eval capybara
end
end
```

Esta es la forma en la que se ejecuta. El primer argumento es la página y el segundo el script capybara en el que se indica que pruebas se van a hacer:

```
$./testinglocalhtml.rb click.html find_and_click.rb
```

Ejemplo del script Capybara que se usa como segundo argumento:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat find_and_click.rb
visit '/'
sleep 2
find('#mydiv').click
sleep 2
accept_alert
sleep 2
```

Tambien puede llamarse así:

```
$./testinglocalhtml.rb index.html script.rb poltergeist
```

## 18.3. Ejemplo de uso de Capybara, RSpec y Cucumber con una Aplicación Web Externa

Vamos a empezar testeando YouTube (no es que creamos que le haga falta :-). Esto es posible porque las pruebas que haremos son de **aceptación**: se refieren al comportamiento de la aplicación y por tanto no es necesario siquiera que tengamos su código fuente. Nos basta con tener la posibilidad de ejecutarla.

Para las pruebas usaremos Cucumber.

Cucumber es una herramienta para escribir *acceptance tests*. En la metodología TDD el stackholder/cliente en vez de pasarse los requisitos al equipo de desarrollo, colabora con los desarrolladores en la escritura de las pruebas que expresan el resultado que el quiere.

Es por eso que a este tipo de pruebas se las denominan *acceptance tests*: intentan capturar lo que el stackholder quiere.

Estas pruebas son diferentes de los tests unitarios cuyo motivo es ayudar a los desarrolladores a comprobar su diseño software.

Se suele decir que **los tests unitarios nos aseguran que construimos la cosa correctamente mientras que los tests de aceptación nos aseguran que construimos la cosa correcta**.

Cuando se usa Behaviour-Driven Development BDD nos preocupamos de escribir los acceptance tests como *ejemplos/examples* que cualquiera pueda leer y entender.

Esto es lo que Cucumber intenta: hacer que la colaboración entre stakeholders y desarrolladores sean fluída.

Este es un ejemplo de test de aceptación Cucumber:

**Feature: Filling a Book Review**

**Scenario: Complete Book Review**

```
Given I am on a book review site
When I submit a book review
Then I should see the saved details confirmed
```

Nótese como las pruebas son especificadas como *examples/ejemplos* de como se debe conducir el sistema en un escenario concreto.

- Cucumber es una herramienta para usar en la línea de comandos.
- Cuando la ejecutamos lee nuestras especificaciones en ficheros denominados *features*, las examina para encontrar los *escenarios* a probar y corre los escenarios que son una lista de pasos/*steps* que Cucumber debe trabajar.

Los ficheros de features están escritos en un lenguaje que se denomina Gherkin (pepinillo).

Cuando se usa cucumber lo primero es crear un fichero **features** en el que se guardan las pruebas.

Este será el *scenario* para testear YouTube en el fichero **features/youtube\_search.features**:

```
$ cat youtube_search.feature
Feature: Search for Videos on YouTube
```

**Scenario: Search for Videos of Large Rodents**

```
Given I am on the YouTube home page
When I search for "capybara"
Then videos of large rodents are returned
```

Esta es nuestra jerarquía de ficheros:

```

features/
|-- youtube_search.feature
|-- step_defs
| '-- steps.rb
 '-- support
 '-- env.rb

```

Supuesto que tenemos un fichero `youtube_search.feature` pero no hemos escrito aún las *step definitions*, si ejecutamos `cucumber` obtenemos:

```

[~/sinatra/sinatra-capybara/youtube]$ cucumber
Feature: Search for Videos on YouTube

 Scenario: Search for Videos of Large Rodents # features/youtube_search.feature:3
 Given I am on the YouTube home page # features/youtube_search.feature:4
 When I search for "capybara" # features/youtube_search.feature:5
 Then videos of large rodents are returned # features/youtube_search.feature:6

1 scenario (1 undefined)
3 steps (3 undefined)
0m0.003s

```

You can implement step definitions for undefined steps with these snippets:

```

Given(/^I am on the YouTube home page$/) do
 pending # express the regexp above with the code you wish you had
end

```

```

When(/^I search for "(.*?)"$/) do |arg1|
 pending # express the regexp above with the code you wish you had
end

```

```

Then(/^videos of large rodents are returned$/) do
 pending # express the regexp above with the code you wish you had
end

```

Copiamos y pegamos los fragmentos de código que nos aconseja Cucumber en nuestro fichero `features/step_defs/`. Ahora ejecutamos `cucumber` de nuevo:

```

~/sinatra/sinatra-capybara/youtube]$ tree
.
'-- features
 |-- step_defs
 | '-- steps.rb
 '-- support
 '-- env.rb
 '-- youtube_search.feature

3 directories, 3 files

```

```

[~/sinatra/sinatra-capybara/youtube]$ cucumber
Feature: Search for Videos on YouTube

 Scenario: Search for Videos of Large Rodents # features/youtube_search.feature:3
 Given I am on the YouTube home page # features/step_defs/steps.rb:1

```

```

TODO (Cucumber::Pending)
features/youtube_search.feature:4:in 'Given I am on the YouTube home page'
When I search for "capybara" # features/step_defs/steps.rb:5
Then videos of large rodents are returned # features/step_defs/steps.rb:9

1 scenario (1 pending)
3 steps (2 skipped, 1 pending)
0m0.003s

```

Cucumber nos informa que los pasos están por implementar.

Ahora editamos `support/env.rb`:

```
[~/sinatra/sinatra-capybara/youtube]$ cat features/support/env.rb
require 'capybara/cucumber'
```

```
Capybara.default_driver = :selenium
```

Estamos usando Selenium WebDriver como herramienta para la automatización del navegador.

Completamos el fichero `features/step_defs/steps.rb`:

```
[~/sinatra/sinatra-capybara/youtube]$ cat features/step_defs/steps.rb
Given(/^I am on the YouTube home page$/) do
 visit 'http://www.youtube.com'
end
```

```
When(/^I search for "(.*?)"$/) do |search_term|
 fill_in 'search_query', :with => search_term
 click_on 'search-btn'
end
```

```
Then(/^videos of large rodents are returned$/) do
 puts page.inspect
 expect(page).to have_content 'Garibaldi'
end
```

La llamada `fill_in 'search_query', :with => search_term` busca por un elemento del DOM que sea una text area o un text field con un atributo `name`, `id` o `label` establecido a `search_query` y lo rellena con el contenido de `search_term` (que es `capybara`).

Mirando el fuente de la página de YouTube encontramos el correspondiente tag `input` con el atributo `name` a `search_query`:

```
<input
 id="masthead-search-term" autocomplete="off" autofocus
 class="search-term yt-uix-form-input-bidi"
 name="search_query" value=""
 type="text" tabindex="1" title="Search">
```

Que tiene al lado el botón de búsqueda con un `id` igual a `search-btn`:

```
<button
 class="yt-uix-button yt-uix-button-size-default yt-uix-button-default search-btn-component s
 type="submit"
 onclick="
 if (_gel('masthead-search-term').value == '')
 return false;
 _gel('masthead-search').submit();"
```

```

 return false;;
 return true;
"
id="search-btn"
tabindex="2"
dir="ltr"
>
Search

```

A partir de la información `search_query` Capybara es capaz de encontrar el elemento `<input>` del DOM y rellenarlo.

Cuando ejecutamos de nuevo `cucumber` se abre Firefox que navega hasta YouTube y busca por Capybara comprobando que la página de resultados contiene la palabra `Garibaldi` y produciendo un informe como este:

```
[~/sinatra/sinatra-capybara/youtube]$ cucumber
Feature: Search for Videos on YouTube

 Scenario: Search for Videos of Large Rodents # features/youtube_search.feature:3
 Given I am on the YouTube home page # features/step_defs/steps.rb:1
 When I search for "capybara" # features/step_defs/steps.rb:5
 Then videos of large rodents are returned # features/step_defs/steps.rb:10
 #<Capybara::Session>

1 scenario (1 passed)
3 steps (3 passed)
0m22.491s
```

### 18.3.1. Poltergeist

Poltergeist es un driver para Capybara.

Nos permite ejecutar pruebas Capybara con un navegador WebKit<sup>1</sup> headless, basado en PhantomJS.

Es posible ejecutar las pruebas del ejemplo anterior usando PhantomJS via Poltergeist cambiando el fichero `features/support/env.rb`:

```
[~/sinatra/sinatra-capybara/youtube(poltergeist)]$ cat features/support/env.rb
require 'capybara/cucumber'
require 'capybara/poltergeist'

#Capybara.default_driver = :selenium
Capybara.register_driver :poltergeist do |app|
 Capybara::Poltergeist::Driver.new(app, :js_errors => false)
end
Capybara.default_driver = :poltergeist
#Capybara.javascript_driver = :webkit
```

Es necesario poner el atributo `:js_errors => false` para evitar que los errores que se producen en JavaScript se propaguen al código Ruby y causen que las pruebas se detengan.

Sigue una ejecución:

---

<sup>1</sup> WebKit is a layout engine software component for rendering web pages in web browsers.

It powers Apple's Safari web browser and a fork of the project is used in Google's Chrome web browser.

By September 2013, WebKit browser market share was larger than that of both the Trident engine used by Internet Explorer and the Gecko engine used by Firefox.

```
[~/sinatra/sinatra-capybara/youtube(poltergeist)]$ rake
cucumber
Feature: Search for Videos on YouTube

 Scenario: Search for Videos of Large Rodents # features/youtube_search.feature:3
 Refused to display document because display forbidden by X-Frame-Options.

 Given I am on the YouTube home page # features/step_defs/steps.rb:1
 Unsafe JavaScript attempt to access frame with URL about:blank from frame with URL file:///Use

 When I search for "capybara" # features/step_defs/steps.rb:5
 Then videos of large rodents are returned # features/step_defs/steps.rb:10
 #<Capybara::Session>

1 scenario (1 passed)
3 steps (3 passed)
0m6.230s
```

## 18.4. Capybara y Rack::Test

Rack::Test es una librería que implementa el protocolo Rack y que hace posible testear la pila completa de nuestra aplicación sin que tengamos que pagar la latencia de las comunicaciones HTTP ni el uso de navegadores reales.

Podemos usar su API para enviar una solicitud/request a nuestra aplicación.

Cuando nuestra aplicación responda, Rack::Test procesa la respuesta y nos permite cuestionarla.

Puede hacerlo porque implementa el protocolo Rack. La forma en la que lo hace es construyendo el hash de request y pasandolo a la aplicación tal y como la haría un servidor web, interpretando el hash de respuesta y dejandonoslo disponible vía su API.

Veamos un ejemplo de una aplicación que usa Rack::Test pero no Capybara para testear una sencilla aplicación Rack del tipo [Hello World](#) (Véase [https://github.com/crguezl/application\\_testing\\_with\\_capybara](https://github.com/crguezl/application_testing_with_capybara)):

```
[~/application_testing_with_capybara/sinatra(master)]$ pwd -P
/Users/casiano/local/src/ruby/testing_with_capybara/application_testing_with_capybara/sinatra
[~/application_testing_with_capybara/sinatra(master)]$ cat test.rb
require 'bundler/setup'
require 'rack'
require "rack/test"
require 'test/unit'

class HelloWorld
 def response
 [200, {'Content-Length' => '11'}, ['Hello World']]
 end
end

class HelloWorldApp
 def self.call(env)
 HelloWorld.new.response
 end
end

class HelloWorldAppTest < Test::Unit::TestCase
```

```

include Rack::Test::Methods

def app
 HelloWorldApp
end

def test_redirect_logged_in_users_to_dashboard
 get "/"

 assert last_response.ok?
 assert_equal last_response.body, 'Hello World'
end

end

```

Una vez instaladas las dependencias podemos ejecutar las pruebas:

```
[~/application_testing_with_capybara/sinatra(master)]$ ruby test.rb
Run options:
```

```
Running tests:
```

```
Finished tests in 0.028666s, 34.8845 tests/s, 69.7691 assertions/s.
1 tests, 2 assertions, 0 failures, 0 errors, 0 skips
```

```
ruby -v: ruby 2.1.2p95 (2014-05-08 revision 45877) [x86_64-darwin13.0]
```

Aquí no ha habido ninguna comunicación externa, todo ocurre vía software.

Rack::Test no tiene una API para hacer click en los elementos, poner radio buttons, manejar forms, etc.

De lo que se ocupa es de relizar la transacción Rack imitando al servidor web.

Para que sirva de ejemplo, este es el código del driver Capybara que traduce el evento click (fichero capybara/rack-test/node.rb) para que sea procesado por Rack::Test:

```

def click
 if tag_name == 'a' && !self[:href].nil?
 method = self["data-method"] if driver.options[:respect_data_method]
 method ||= :get
 driver.follow(method, self[:href].to_s)
 elsif (tag_name == 'input' and %w(submit image).include?(type)) or
 ((tag_name == 'button') and type.nil? or type == "submit")
 associated_form = form
 Capybara::RackTest::Form.new(driver, associated_form).submit(self) if associated_form
 end
end

```

Capybara determina el tipo de elemento que está siendo clickeado y después calcula si Rack::Test debe seguir el enlace o submitir el formulario.

#### 18.4.1. Testeando una Aplicación Sinatra

Consideremos la siguiente aplicación Sinatra que permite submitir una revisión de un libro y que retorna los datos enviados. No se ha implementado persistencia.

## app.rb

```
[~/application_testing_with_capybara/sinatra(master)]$ cat app.rb
require 'bundler/setup'
require 'sinatra'

class BookReview < Sinatra::Base

 get '/form' do
 erb :form
 end

 post '/submit' do
 @name = params[:name]
 @title = params[:title]
 @review = params[:review]
 @age = params[:age]
 erb :result
 end

end

#only run if invoked from command line - otherwise leave to Capybara
BookReview.run! if __FILE__ == $0
```

## Entrada

### Plantilla para el Formulario. Template: form.erb

```
[~/application_testing_with_capybara/sinatra(master)]$ cat views/form.erb
<link rel="stylesheet" href="css/form.css">
<form action="/submit" method="post">
 <header id="header" class="info">
 <h2>Book Reviews</h2>
 <div>Review the last book you purchased...</div>
 </header>

 <label for="name">
 Your Name
 </label>
 <input type="text" id="name" name="name" maxlegth="255">

 <label class="desc" for="age">
 Age Range
 </label>
 <div>
 <select id="age" name="age">
 <option value="-" selected="selected">
 </option>
 <option value="<20" >
 Under 20
 </option>
 </select>
 </div>

</form>
```

```

<option value="20-50" >
20 - 50
</option>
<option value="50+" >
Over 50
</option>
</select>
</div>

<label for="book_title">
 Book Title
</label>
<input type="text" id="book_title" name="title" maxlegth="255">

<label for="review">
 Your Review...
</label>
<textarea id="review" name="review" rows="10" cols="50"></textarea>

<input type="submit" value="Submit"/>

</form>

```

**Plantilla/Template con los resultados: result.erb** Hay otro template ERB utilizado para mostrar la revisión:

```

[~/application_testing_with_capybara/sinatra(master)]$ cat views/result.erb
<div class="saved_review">
<p>You submitted the following on: <%= Time.new.strftime("%Y-%m-%d %H:%M:%S") %> </p>

<p id="name">Name: <%= @name %></p>

<p id="age">Age: <%= @age %></p>

<p id="title">Book Title: <%= @title %></p>

<p id="review">Book Review: <%= @review %></p>

Submit another review...
</div>

```

## Salida

Veamos ahora como hacer las pruebas para esta aplicación usando Capybara.

### 18.4.2. Testing con Rack::Test y Cucumber

Desde la perspectiva de Capybara, la interacción con nuestra aplicación Sinatra es la misma que la interacción con una aplicación remota.

La única diferencia esta en el setup.

Vamos a usar Cucumber.

Cucumber es una herramienta para escribir *acceptance tests*. En la metodología TDD el stackholder/cliente en vez de pasarse los requisitos al equipo de desarrollo, colabora con los desarrolladores en la escritura de las pruebas que expresan el resultado que el quiere.

Es por eso que a este tipo de pruebas se las denominan *acceptance tests*: intentan capturar lo que el stackholder quiere.

Estas pruebas son diferentes de los tests unitarios cuyo motivo es ayudar a los desarrolladores a comprobar su diseño software.

Se suele decir que [los tests unitarios nos aseguran que construimos la cosa correctamente mientras que los tests de aceptación nos aseguran que construimos la cosa correcta](#).

Cuando se usa Behaviour-Driven Development BDD nos preocupamos de escribir los acceptance tests como *ejemplos/examples* que cualquiera pueda leer y entender.

Esto es lo que Cucumber intenta: hacer que la colaboración entre stackholders y desarrolladores sean fluída.

Este es un ejemplo de test de aceptación Cucumber:

**Feature: Filling a Book Review**

```
Scenario: Complete Book Review
 Given I am on a book review site
 When I submit a book review
 Then I should see the saved details confirmed
```

Nótese como las pruebas son especificadas como *examples/ejemplos* de como se debe conducir el sistema en un escenario concreto.

- Cucumber es una herramienta para usar en la línea de comandos.
- Cuando la ejecutamos lee nuestras especificaciones en ficheros denominados *features*, las examina para encontrar los *escenarios* a probar y corre los escenarios que son una lista de pasos/*steps* que Cucumber debe trabajar.

Los ficheros de features están escritos en un lenguaje que se denomina Gherkin (pepinillo).

Cuando se usa cucumber lo primero es crear un fichero **features** en el que se guardan las pruebas. Esta es la estructura de **features/chapter3/**:

```
[~/application_testing_with_capybara(master)]$ tree features/chapter3/
features/chapter3/
|-- sinatra.feature
`-- steps
 '-- sinatra.rb

1 directory, 2 files
```

Cada prueba Cucumber se denomina *scenario* y cada scenario contiene *steps* que le dicen a Cucumber que hay que hacer.

Este es el fichero **sinatra.feature**:

```
[~/application_testing_with_capybara(master)]$ cat features/chapter3/sinatra.feature
Feature: Using Capybara and Rack-Test to Interact with Sinatra App
```

```

Scenario: Complete Book Review
 Given I am on a book review site
 When I submit a book review
 Then I should see the saved details confirmed

```

Las palabras clave Feature , Scenario , Given , When y Then son reservadas. El resto es texto. La sintaxis se denomina *Gherkin*. Un fichero Gherkin suele tener una estructura como esta:

```

1: Feature: Some terse yet descriptive text of what is desired
2: Textual description of the business value of this feature
3: Business rules that govern the scope of the feature
4: Any additional information that will make the feature easier to understand
5:
6: Scenario: Some determinable business situation
7: Given some precondition
8: And some other precondition
9: When some action by the actor
10: And some other action
11: And yet another action
12: Then some testable outcome is achieved
13: And something else we can check happens too
14:
15: Scenario: A different situation
16: ...

```

Es necesario implantar el significado de estos pasos. En nuestro caso los ponemos en `features/chapter3/steps`. Cucumber nos permite ponerle en cualquier subdirectorio siempre que la extensión sea `.rb`.

Estos son los contenidos de `steps/sinatra.rb`:

```

[~/application_testing_with_capybara(master)]$ cat features/chapter3/steps/sinatra.rb
Given(/^I am on a book review site$/) do
 visit('/form')
end

When(/^I submit a book review$/) do
 fill_in 'name', :with => 'Matt'
 fill_in 'title', :with => 'Catch 22'
 fill_in 'review', :with => 'Alright I guess....'
 select '20 - 50', :from => 'age'
 click_on 'Submit'
end

Then(/^I should see the saved details confirmed$/) do
 page.should have_text 'You submitted the following on:'
 expect(find('#name')).to have_text 'Matt'
 expect(find('#age')).to have_text '20-50'
 expect(find('#review')).to have_text 'Alright I guess....'
 expect(find('#title')).to have_text 'Catch 22'
end

```

Cucumber falla un step si su definición genera una excepción (`raise`).

Junto con las features le damos a Cucumber un conjunto de [step definitions](#) como el anterior.

En este fichero establecemos una correspondencia entre el lenguaje orientado al cliente y el código Ruby que debe llevar a cabo las acciones.

Es aquí donde interviene Capybara que nos provee de un DSL para la automatización del navegador.

El fichero `support/env.rb` gobierna la configuración:

```
[~/application_testing_with_capybara(rack_version)]$ cat features/support/env.rb
require 'bundler'
Bundler.require
require 'capybara/cucumber'
require 'rspec/expectations'
require_relative '../../sinatra/app'

Capybara.default_driver = :rack_test
```

```
Capybara.app = BookReview #our Sinatra app
```

- En este fichero establecemos el valor de Capybara.app a la clase de nuestra aplicación Sinatra (BookReview).

En el caso de una aplicación sinatra clásica pondríamos:

```
Capybara.app = Sinatra::Application
```

- También establecemos Capybara.default\_driver a :rack\_test de manera que se use Rack::Test en vez de Selenium. De este modo la ejecución será mucho mas rápida.
- Ademas indicamos que queremos trabajar con capybara, con rspec y cargamos nuestra aplicación para que pueda ser testeada:

```
require 'capybara/cucumber'
require 'rspec/expectations'
require_relative '../../sinatra/app'
```

- La línea Bundler.require:

```
require 'bundler'
Bundler.require
```

itera sobre cada gema en el Gemfile y hace un require de todas las gemas que no hayan sido explícitamente desabilitadas con la opción :require => false.

Es posible pasarle una lista de grupos:

```
Bundler.require(:default, :development)
```

## Rakefile

```
[~/application_testing_with_capybara(master)]$ cat Rakefile
task :default => :rackfeatures
task :rackfeatures do
 sh "cucumber -r features features/chapter3/sinatra.feature"
end
```

## Ejecución de las Pruebas

```
[~/application_testing_with_capybara(master)]$ rake
cucumber -r features features/chapter3/sinatra.feature
Feature: Using Capybara and Rack-Test to Interact with Sinatra App
```

<pre>Scenario: Complete Book Review   Given I am on a book review site</pre>	<pre># features/chapter3/sinatra.feature:3 # features/chapter3/steps/sinatra.rb:1</pre>
------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------

```

When I submit a book review # features/chapter3/steps/sinatra.rb:5
Then I should see the saved details confirmed # features/chapter3/steps/sinatra.rb:13

1 scenario (1 passed)
3 steps (3 passed)
0m0.070s

```

## Desventajas de usar Rack::Test

Hay al menos dos desventajas

- Rack::Test no es un navegador
- Rack::Test no permite ejecutar código en el lado del cliente

## 18.5. La API de Capybara

### 18.5.1. Búsqueda de Elementos HTML

#### Búsqueda de Elementos HTML por Capybara

Capybara usa selectores CSS y consultas XPath para buscar en el árbol DOM<sup>2</sup> por contenido específico dentro de la página web.

Para buscar un elemento cuyo atributo `id` vale `main`:

- XPath:

```
//*[@id='main']
```

- CSS:

```
#main
```

Para buscar un hijo directo o indirecto de cualquier elemento `<div>` con clase `container`:

- XPath:

```
//div//*[@class='container']
```

- CSS:

```
div .container
```

Capybara usa por defecto selectores CSS3<sup>3</sup>:

```
page.find('#maincontent')
```

Si deseamos usar selectores XPath podemos explicitarlo en la llamada al método:

```
page.find(:xpath, //*[@id="maincontent"])
```

O podemos establecerlo globalmente - por ejemplo en el fichero `env.rb` si estamos usando Cucumber:

```

require 'capybara/cucumber'
Capybara.default_selector = :xpath
Capybara.default_driver = :selenium

```

---

<sup>2</sup> El DOM es una estructura en árbol resultante del análisis sintáctico del contenido de la página HTML así como del resultado de la ejecución del JavaScript asociado

<sup>3</sup> Por tanto podemos usamos usar selectores CSS3 como `:nth-child`, `:nth-of-type`

## Chrome: Observando las propiedades de un elemento HTML con inspect

Nos situamos en la página sobre el elemento del que queremos información y pulsamos el botón de contexto. Elegimos inspeccionar elemento:

Esto nos lleva a las herramientas del desarrollador mostrándonos la información sobre el elemento:

## Buscando Elementos en Chrome en el panel Elements

El panel **Elements** es a menudo la mejor forma de visualizar el fuente de una página y navegar en la estructura de la página y del DOM.

Podemos tambien elegir un elemento y se verá resaltado en la página.

La búsqueda mediante expresiones regulares nos permite tambien hacer búsquedas en la página.

## Buscando Elementos en Chrome en la Consola

En Chrome podemos buscar por los elementos que casan con un xpath usando `$x("some_xpath")` y usando `$$("css-selectors")` en la consola (Véase Chrome Tips and Tricks).

### 18.5.2. Clicks en Botones y Enlaces

Estos métodos permiten a la aplicación navegar usando links o botones:

- `click_link_or_button`
- `click_on`
- `click_link`

#### Ejemplo de Uso de `click_link`

Tenemos estos fichero HTML:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat index.html
<!DOCTYPE HTML>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title></title>
</head>
<body>

 simple_link.html

</body>
</html>
```

y este otro:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat simple_link.html
<!DOCTYPE HTML>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title></title>
</head>
<body>
 <div id="main">
 <div class="section">
 Click this Anchor
```

```

 </div>
 </div>
</body>
</html>

```

El siguiente programa visita `index.html` clica en el link y visita la segunda página volcando sus resultados:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat check_click.rb
require 'capybara'
require 'capybara/poltergeist'

Capybara.default_driver = :poltergeist

session = Capybara::Session.new(:poltergeist)
session.visit "http://crguezl.github.io/selenium-examples/index.html"

session.click_link "example1"

puts session.body # dump the contents of the page

session.save_page('example1_download.html')
```

El funcionamiento sería el mismo si usáramos `click_on` o `click_link_or_button`.

**Algoritmo de Búsqueda de Capybara para los Links y los Botones** En el caso de los enlaces y los botones, Capybara mira las siguientes propiedades de los elementos con vistas a localizar donde clicar:

- The `id` attribute of the anchor, button, or input tag
- The `title` attribute of the anchor, button, or input tag
- `Text` within the anchor, button, or input tag
- The `value` attribute of the input element where its type is one of `button`, `reset`, `submit`, or `image`
- The `alt` attribute where an image is used as an anchor or input

### 18.5.3. Envío de Formularios

Consideremos el siguiente fragmento de una vista HTML:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ sed -ne '/@@index/,/^$/p' check_form_server.rb
@@index
<form id="myform" method="post" action="/">
 <label for "Forename">Forename</label>
 <input type="text" name="Forename" value="" />

 <label for "Surname">Surname</label>
 <input type="text" name="Surname" value="" />

 <input type="submit" value="Go" />
</form>
```

`fill_in ... :with ...` Queremos llenar los campos y enviarlos. El siguiente código lo hace:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat check_form_client.rb
require 'capybara'
require 'capybara/poltergeist'

Capybara.default_driver = :poltergeist

session = Capybara::Session.new(:poltergeist)
session.visit "http://localhost:4567"

session.instance_eval do
 fill_in 'Forename', :with => 'Casiano'
 fill_in 'Surname', :with => 'Rodríguez'
 click_on 'Go'

 save_screenshot("formfilled.pdf")
 f = find('#Forename') # f is a Capybara::Element object
 puts "****\nForename = #{f.text}"
 s = find('#Surname')
 puts "****\nSurname = #{s.text}"
end
```

Cuando se ejecuta `ruby check_form_client.rb` produce esta salida:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ rake formclient
ruby check_form_client.rb

Forename = Casiano

Surname = Rodríguez
```

## Guardando un Screenshot

Además, la llamada a `save_screenshot` nos ha guardado un pdf con un screenshot de la página:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ ls -ltr | tail -1
-rw-r--r-- 1 casiano staff 13625 1 dic 13:35 formfilled.pdf
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ open formfilled.pdf
```

Véase el código del servidor sinatra en [crguezl/selenium-examples/blob/gh-pages/check\\_form\\_server.rb](#)

## Algoritmo de Búsqueda de Capybara para `fill_in`

Para localizar campos que aceptan `input`, Capybara utilizará:

1. The `id` attribute of the input element
2. The `name` attribute of the input element
3. A related `label` element

## Checkboxes y Radio Buttons

La API para manipular checkboxes y radio buttons es similar a la de los text inputs.

La siguiente vista contiene algunas checkboxes y radio buttons:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ sed -ne '/@@index/,/^$/p' check_radio_buttons_se
@@index
<form id="myform" method="post" action="/" enctype="multipart/form-data">
 <label for="name1">User Forename</label>
 <input id="name1" type="text" name="Forename" value="" />
 <label for="name2">User Surname</label>
 <input id="name2" type="text" name="Surname" value="" />
 <p>
 <label for="title">Title</label>
 <select name="user_title" id="title">
 <option>Mrs</option>
 <option>Mr</option>
 <option>Miss</option>
 </select>
 </p>
 <p>
 <label for="under_16">Under 16</label>
 <input type="radio" name="age" value="under"
 id="under_16" >
 <label for="over_16">Over 16</label>
 <input type="radio" name="age" value="over"
 id="over_16">
 </p>
 <p>
 <label for="consent">Consent Given?</label>
 <input type="checkbox" value="yes" name="consent_checkbox"
 id="consent"/>
 </p>
 <p>
 <label for="form_image">Image (.png)</label>
 <input type="file" name="image" id="form_image"/>
 </p>
 <input type="submit" value="Go" />
</form>
```

Esta es la forma en la que se ve:

Este es el código para manipular el formulario con menus, radio buttons y checkboxes:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat check_radio_buttons_client.rb
require 'capybara'
require 'capybara/poltergeist'

Capybara.default_driver = :poltergeist

session = Capybara::Session.new(:poltergeist)

session.instance_eval do
 visit "http://localhost:4567"
 fill_in 'Forename', :with => 'Casiano'
 fill_in 'Surname', :with => 'Rodríguez'
 select 'Mr', :from => 'title' # id
 choose "Over 16" # text
 check "consent" # id
 attach_file "form_image", "./bulb.png"
```

```

click_on 'Go'

Save a screenshot of the page and open it for inspection.
save_and_open_screenshot("./formfilled.pdf")

f = find('#Forename') # f is a Capybara::Element object
puts "#{f.text}"
s = find('#Surname')
puts "#{s.text}"
puts "#{s.text}"
puts "#{find('#title').text}"
puts "#{find('#age').text}"
puts "#{find('#consent').text}"
end

```

## Guardando un Screenshot y Abriéndolo

La llamada `save_and_open_screenshot("./formfilled.pdf")` guarda un screenshot de la página y abre el fichero .pdf:

**Menus: seleccionado una opción en un select** Para seleccionar de un menú desplegable  
`<select>`

```

<select name="user_title" id="title">
 <option>Mrs</option>
 <option>Mr</option>
 <option>Miss</option>
</select>

```

usamos el método `select`

```
select 'Mr', :from => 'title' # id
```

Si `:from` esta presente, `select` encuentra la caja y la selecciona. En otro caso encuentra una opción y la selecciona. Si el `select` es múltiple<sup>4</sup>, se le puede llamar varias veces para que seleccione mas de una opción.

Como ocurren con los elementos `input` Capybara mira en las labels y en los atributos `id` y `name`.

El valor a seleccionar (`Mr`) debe ser uno de los textos en la lista de hijos `option`:

```

<option>Mrs</option>
<option>Mr</option>
<option>Miss</option>

```

Capybara dispone asimismo de un método `unselect` que limpia la selección.

**Selección del radio button** Para seleccionar un radio button

```

<label for="under_16">Under 16</label>
<input type="radio" name="age" value="under"
 id="under_16" >
<label for="over_16">Over 16</label>
<input type="radio" name="age" value="over"
 id="over_16">

```

usamos `choose`

---

<sup>4</sup> HTML5 introduce el atributo `multiple` que especifica que se pueden seleccionar múltiples opciones de una vez

```
choose "Over 16" # text
```

La sintáxis es:

```
#choose(locator, options = {}) ⇒ Object
```

encuentra el radio button y lo marca. locator puede ser name, id o el texto de la label.

Este es su código:

```
File 'lib/capybara/node/actions.rb', line 68
def choose(locator, options={})
 find(:radio_button, locator, options).set(true)
end
```

## Checkbox

Para seleccionar la checkbox

```
<label for="consent">Consent Given?</label>
<input type="checkbox" value="yes" name="consent_checkbox"
 id="consent"/>
```

usamos check:

```
check "consent" # id
```

The check box can be found via name, id or label text.

Existe un método inverso uncheck:

```
uncheck(locator, options = {})
```

Find a check box and mark uncheck it.

```
File 'lib/capybara/node/actions.rb', line 94
def uncheck(locator, options={})
 find(:checkbox, locator, options).set(false)
end
```

## Cargando un Fichero /File Upload

Para cargar un fichero

```
<p>
 <label for="form_image">Image (.png)</label>
 <input type="file" name="image" id="form_image"/>
</p>
```

que es procesado en el servidor por esta ruta:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ sed -ne '/^post/,/^$/p' check_radio_buttons_server
post '/' do
 pp params
 File.open("public/image.png", "w") do |f|
 f.write(params['image'][tempfile].read) if f.size < (1 << 16)
 end
 erb :result, :locals => { :params => params }
end
```

La vista result muestra los resultados:

```

@@result

 <li id="Forename"> Forename: <%= params[:Forename] %>
 <li id="Surname"> Surname: <%= params[:Surname] %>
 <li id="title"> User title: <%= params[:user_title] %>
 <li id="age"> Age (under/over 16): <%= params[:age] %>
 <li id="consent"> Consent: <%= params[:consent_checkbox] %>
 <li id="image">

Go back

```

Recuerde que debe establecer el `enctype` en el elemento `form` a `multipart/form-data`, sino obtendríamos el nombre del fichero en vez del objeto:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ grep '<form' check_radio_buttons_server.rb
<form id="myform" method="post" action="/" enctype="multipart/form-data">
```

Para subir el fichero en capybara usamos `attach_file`:

```
attach_file "form_image", "./bulb.png"
```

The file field can be found via its name, id or label text.

```
File 'lib/capybara/node/actions.rb'
def attach_file(locator, path, options={})
 Array(path).each do |p|
 raise Capybara::FileNotFound,
 "cannot attach file, #{p} does not exist" unless
 File.exist?(p.to_s)
 end
 find(:file_field, locator, options).set(path)
end
```

## 18.6. Buscadores/Finders, Ámbito/Scope y Coincidencias/Matches Múltiples

Capybara está construido sobre un conjunto de bloques básicos. Estos bloques básicos consisten en expresiones XPath que se usan para encontrar cosas en la página para a continuación delegar la acción correspondiente en el driver subyacente.

La mayor parte del tiempo usaremos los métodos de alto nivel, pero hay ocasiones en las que debemos recurrir a estos *finders* para lograr nuestro propósito.

Consideremos esta página web:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat click.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Click Examples</title>
 <script>
 window.onload = function() {
 var myanchor = document.getElementById("myanchor");
 var mydiv = document.getElementById("mydiv");
 var mybutton = document.getElementById("mybutton");
 }
 </script>
 </head>
 <body>
 <div id="mydiv">This is a test</div>
 Click me!
 <input type="button" value="I'm a button" id="mybutton"/>
 </body>
</html>
```

```

myanchor.onclick = function() {
 alert("anchor has been clicked");
};

mydiv.onclick = function() {
 alert("div has been clicked");
};

mybutton.onclick = function() {
 alert("button has been clicked");
};

</script>
</head>
<body>
<div id="main">
 <div class="section">
 Click this Anchor
 <div id="mydiv" title="mydivtitle">Click This Div</div>
 <input id="mybutton" type="button" value="Click This Button" title="mybutontitle"/>
 </div>
</div>
</body>
</html>

```

Cuando se cliquea en el div

```
<div id="mydiv" title="mydivtitle">Click This Div</div>
```

se emite un mensaje de alerta.

Para automatizar este click haremos uso del método `find`.

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat find_and_click.rb
visit '/'
find('#mydiv').click
accept_alert
```

`find('#mydiv')` retorna un `Capybara::Node::Element` sobre el cual se invoca la acción.

Un `Element` representa un elemento de la página. Es posible interactuar con los contenidos de este elelemnto en la misma forma que con un documento:

```
session = Capybara::Session.new(:rack_test, my_app)

bar = session.find('#bar') # from Capybara::Node::Finders
bar.select('Baz', :from => 'Quox') # from Capybara::Node::Actions
```

Los `Element` también tienen atributos HTML:

```
bar.value
bar.text
bar[:title]
```

Para probar este HTML vamos a usar nuestro script genérico `testinglocals.rb`:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$./testinglocalhtml.rb click.html find_and_click
```

Casi todo el resto de métodos de búsqueda se construyen a partir de este (Véase la clase Capybara::Node::Finders):

- - **find\_field**: This finder searches for form fields by the related label element, or the name/id attribute

```
File 'lib/capybara/node/finders.rb'
def find_field(locator, options={})
 find(:field, locator, options)
end
```
  - **field\_labeled**: This finder is the same as **find\_field**
  - **find\_link**: This finder finds an anchor or an image link using the text, id, or img alt attribute
  - **find\_button**: This finder finds a button by the id, name, or value attribute
  - **find\_by\_id**: This finder finds any element by the id attribute
  - **first([kind], locator, options)**: Find the first element on the page matching the given selector and options, or nil if no element matches.

```
File 'lib/capybara/node/finders.rb'
def first(*args)
 all(*args).first
end
```

## Múltiples Coincidencias/Multiple Matches

Supongamos que estamos comprobando algunas búsquedas de resultados que son añadidos dinámicamente a la página:

```
[~/sinatra/sinatra-selenium/intro(gh-pages)]$ cat multiple.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Multiple Match Examples</title>
 <style>
 .result {
 height: 30px;
 width: 100px;
 background-color: green;
 margin-bottom: 10px;
 }
 </style>
 </head>
 <body>
 <div id="main">
 <h1>Search Results</h1>
 <ul class="section">
 <li id="res1" class="result">Match 1
 <li id="res2" class="result">Match 2
 <li id="res3" class="result">Match 3
```

```

<li id="res4" class="result">Match 4
<li id="res5" class="result">Match 5

</div>
</body>
</html>

```

Una posible solución es iterar sobre los elementos de la clase `.result` usando el método `all`:

```

$ cat multiple.rb
RSpec.configure do |config| # avoid warning
 config.expect_with :rspec do |c|
 c.syntax = [:should, :expect]
 end
end
visit '/'
all('.result').each_with_index do |elem, idx|
 elem.text.should == "Match #{idx + 1}"
end

```

Cuando se ejecuta

```

$./testinglocalhtml.rb multiple.html multiple.rb p
$

```

no produce fallos.

## Visibilidad

La visibilidad de un elemento afecta a la forma en la que Capybara lo localiza en el DOM. Capybara dispone de un atributo global que se usa para determinar si se prueban o no elementos escondidos (por ejemplo usando propiedades CSS como `display: none` o `visibility: hidden`):

```
Capybara.ignore_hidden_elements = true
```

el valor por defecto es `true`.

## Ambito

Una opción cuando se intenta encontrar contenido que nos fácil de identificar únicamente es restringir la consulta a una sección de la página.

Capybara provee el método `within` que permite consultas con ámbito.

```
#within(*find_args) => Object
#within(a_node) => Object
```

Executes the given block within the context of a node. `within` takes the same options as `find`, as well as a block. For the duration of the block, any command to Capybara will be handled as though it were scoped to the given element.

```
within(:xpath, '//div[@id="delivery-address"]') do
 fill_in('Street', :with => '12 Main Street')
end
```

Just as with `find`, if multiple elements match the selector given to `within`, an error will be raised, and just as with `find`, this behaviour can be controlled through the `:match` and `:exact` options.

It is possible to omit the first parameter, in that case, the selector is assumed to be of the type set in `Capybara.default_selector`.

```
within('div#delivery-address') do
 fill_in('Street', :with => '12 Main Street')
end
```

Note that a lot of uses of `within` can be replaced more succinctly with chaining:

```
find('div#delivery-address').fill_in('Street', :with => '12 Main Street')
```

Raises: (`Capybara::ElementNotFound`) — If the scope can't be found before time expires

## 18.7. Véase

1. Cucumber + Capybara + Poltergeist: Rockstar BDD Combo by Shashikant Jagtap
2. Automate tasks on the web with Ruby and Capybara
3. Testing front-end for a Sinatra app with RSpec and Capybara by Julio Cesar Ody
4. RSpec and Capybara Intro Tutorial at RailsConf 2013 by Brian Sam-Bodden, Founder and CEO at Integrallis Software, LLC
5. NetTuts+ tutorial: Testing Web Apps with Capybara and Cucumber Andrew Burgess on Aug 22nd 2011 with 22 Comments

## Capítulo 19

# Pruebas. Other Testing tools

See SAASBOOK. Questions specific to CS 169.1x/CS 169.2x

### Why are you using Ruby and Rails rather than a more popular language like Java or Python?

We believe it is critical to teach the importance of testing, of communicating with nontechnical customers, and of using all methods at your disposal to create code that is modular and reusable. Our colleagues in the SaaS industry strongly support these goals. The Ruby/Rails ecosystem has by far the most developed tools for unit and functional testing (rspec), integration testing (capybara), acceptance testing and customer communication (cucumber), as well as great tools for code analysis both quantitative (flog, simplecov, saikuro) and qualitative (reek, flay). And Ruby, as a language, includes a number of mechanisms that make it easier to architect your code for reuse (including higher order functions, internal iterators using yield(), mix-ins, and metaprogramming).

- Reek is a tool that examines Ruby classes, modules and methods and reports any code smells it finds.
- Code coverage for Ruby 1.9+ with a powerful configuration library and automatic merging of coverage across test suites
- Code coverage for ruby 1.9.3, 2.0 and 2.1

# Capítulo 20

## Integración Contínua: Travis

*Continuous integration (CI)* is the practice, in software engineering, of merging all developer workspaces with a shared mainline several times a day. It was first named and proposed as part of extreme programming (*XP*).

Its main aim is to prevent integration problems, referred to as [integration hell](#) in early descriptions of *XP*.

CI can be seen as an intensification of practices of periodic integration advocated by earlier published methods of incremental and iterative software development.

1. CI was originally intended to be used in combination with automated unit tests written through the practices of test-driven development.
2. Initially this was conceived of as running all unit tests and verifying they all passed before committing to the mainline.
3. Later elaborations of the concept introduced build servers, which automatically run the unit tests periodically or even after every commit and report the results to the developers.
4. In the same vein the practice of *continuous delivery* further extends CI by [making sure the software checked in on the mainline is always in a state that can be deployed to users](#) and makes the actual deployment process very rapid.

*The use of build servers (not necessarily running unit tests) had already been practised by some teams outside the XP community. Now, many organisations have adopted CI without adopting all of XP.*

### 20.1. Uso de Travis

#### Step one: Sign in

To get started with Travis CI, sign in through GitHub OAuth. Go to Travis CI and follow the [Sign In link at the top](#).

GitHub will ask you to grant read and write access. Travis CI needs write access for setting up service hooks<sup>1</sup> for your repositories when you request it, but it won't touch anything else.

#### Step two: Activate GitHub Service Hook

Once you're signed in go to your profile page. You'll see a list of your repositories.

Flip the on/off switch for each repository that you want to hook up on Travis CI.

---

<sup>1</sup> Like many other Version Control Systems, Git has a way to fire off custom scripts when certain important actions occur. There are two groups of these hooks: client-side and server-side. Client-side hooks are triggered by operations such as committing and merging, while server-side hooks run on network operations such as receiving pushed commits. You can use these hooks for all sorts of reasons. The hooks are all stored in the `hooks` subdirectory of the Git directory

**Step three: Install the travis gem** The travis gem includes both a command line client and a Ruby library to interface with a Travis CI service. Véase *Travis en la Línea de Comandos* 20.6.

### Add .travis.yml file to your repository

In order for Travis to build your project, you need to tell the system a little bit about it. To do so, add `.travis.yml` to the root of your repository.

1. The most important one is the `language` key. It tells Travis what builder to pick.
2. Ruby projects typically use different build tools and practices than Clojure or PHP projects do, so Travis needs to know what to do.
3. If `.travis.yml` is not in the repository, is misspelled or is not valid YAML, travis-ci.org will ignore it, assume Ruby as the language and use default values for everything.

```
[~/srcLPP/coro/frac(master)]$ cat .travis.yml
language: ruby
rvm:
 - 1.9.3
 - jruby-18mode # JRuby in 1.8 mode
 - jruby-19mode # JRuby in 1.9 mode
 - rbx-18mode
 - rbx-19mode
 - 1.8.7
```

**Validate Your .travis.yml** The travis gem includes both a command line client and a Ruby library to interface with a Travis CI service.

Inside a repository with `.travis.yml`

```
[~/sinatra/sinatra-number_cruncher(master)]$ travis lint
Hooray, .travis.yml looks valid :)
```

`travis-lint` will check things like

1. The `.travis.yml` file is valid YAML
2. The `language` key is present
3. The runtime versions (Ruby, PHP, OTP, etc) specified are supported in the Travis CI Environment
4. That you are not using deprecated features or runtime aliases and so on.

### Step four: Trigger Your First Build With a Git Push

Once GitHub hook is set up, push your commit that adds `.travis.yml` to your repository.

That should add a build into one of the queues on Travis CI and your build will start as soon as one worker for your language is available.

To start a build commit and push something to your repository

### Default Test Script

Travis will use `Bundler` to install your project's dependencies and run `rake` by default to execute your tests.

Please note that *you need to add `rake` to your `Gemfile`* (adding it to just `:test` group should be sufficient).

```
group :development, :test do
 gem "rake"
end
```

Groups can be ignored by `bundle` at install-time (using `--without=GROUP1[ GROUP2...]`).

```
[~/srcLPP/coro/frac(master)]$ cat Gemfile
source 'https://rubygems.org'

gem 'rake'
gem 'rspec'
```

Exclude non-essential gems like `ruby-debug` from your `Gemfile`.

**Testing Against Multiple Versions of Gems** Many projects need to be tested against multiple versions of Rack, EventMachine, HAML, Sinatra, Ruby on Rails, you name it. You can do it with Travis CI. See the manual pages.

1. Create a directory in your project's repository root where you will keep gemfiles (`./gemfiles` is a commonly used name)
2. Add one or more gemfiles to it
3. Instruct Travis CI to use those gemfiles using the `gemfile` option in your `.travis.yml`

```
gemfile:
 - Gemfile
 - gemfiles/eventmachine-pre
```

**Custom Bundler arguments and Gemfile locations** You can specify a custom Gemfile name:

```
gemfile: gemfiles/Gemfile.ci
```

Unless specified, the worker will look for a file named `Gemfile` in the root of your project.

You can also set extra arguments to be passed to bundle install:

```
bundler_args: --without=development
```

## Rakefile

```
john@exthost:~/291012/frac$ rake -T
rake spec # Run RSpec code examples
rake specman # Run rspec with --format documentation
rake thtml # Run rspec with format: html

john@exthost:~/291012/frac$ cat Rakefile
$:unshift File.dirname(__FILE__) + 'lib'

require 'rspec/core/rake_task'
RSpec::Core::RakeTask.new
task :default => :spec

desc "Run rspec with --format documentation"
task :specman do
 sh "rspec -Ilib spec/frac_spec.rb --format documentation"
end
```

```

desc "Run rspec with format: html"
task :thtml do
 sh "rspec -Ilib spec/frac_.rb --format html > index.html"
end

```

Véase la documentación de la clase RSpec::Core::RakeTask.

```

john@exthost:~/291012/frac$ rake
/usr/local/ruby/bin/ruby -S rspec ./spec/frac_spec.rb
.....
Finished in 0.00451 seconds
20 examples, 0 failures
john@exthost:~/291012/frac$

```

### Ejemplo de Jerarquía

```

[~/srcLPP/coro/frac(master)]$ tree -A
.
|-- Gemfile
|-- README
|-- Rakefile
|-- lib
| '-- fraction.rb
`-- spec
 '-- frac_spec.rb

2 directories, 5 files

```

### Página de Travis Mostrando los Resultados de unas Pruebas con Rspec

Figura 20.1: Página de Travis Mostrando los Resultados de unas Pruebas con Rspec

## 20.2. Enlaces

- Ejemplo en GitHub
  1. La Clase Point con Integración Contínua (GitHub)
  2. La clase Point con Integración Contínua (Travis)
- URL shortener por Jazer y Javier
- Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis 36.0.2
- *Sample app with Sinatra Rspec Capybara and Selenium 36.0.4*
- *Capybara 18*
- Continuous Integration (Wikipedia)
- Travis

1. Travis
  2. Travis para Ruby
  3. Travis (Proyecto en GitHub)
- Creación de Gemas
    1. How To Build A Ruby Gem With Bundler, Test-Driven Development, Travis CI And Coveralls, Oh My! By Mark McDonnell
    2. Bleigh tutorial to building gems (Proyecto en GitHub)
    3. Polishing Rubies (Part 3): Tools for Testing by Michael Bleigh March 23, 2012

## 20.3. Notificaciones/Notifications

Véase Configuring Build Notifications

## 20.4. Limites de Timepo/Build Timeouts

Because it is very common to see test suites or before scripts to hang up, Travis CI has hard time limits. If a script or test suite takes longer to run, the build will be forcefully terminated and you will see a message about this in your build log.

With our current timeouts, a build will be terminated if it's still running after 50 minutes (respectively 70 on travis-ci.com), or if there hasn't been any log output in 10 minutes.

## 20.5. Databases and other services

See Databases and other services

Some of the services available are:

1. MySQL
2. PostgreSQL
3. MongoDB
4. CouchDB
5. Redis
6. Riak
7. RabbitMQ
8. Memcached
9. Cassandra
10. Neo4J
11. ElasticSearch
12. Kestrel
13. SQLite3

Podemos ver una aplicación Sinatra en la que se usa Travis para CI en:

Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis por Darren Jones, Published December 9, 2013

## 20.6. Travis en la Línea de Comandos

The travis gem includes both a command line client and a Ruby library to interface with a Travis CI service. Both work with travis-ci.org, travis-ci.com or any custom Travis CI setup you might have. Check out the installation instructions to get it running in no time.

### The Travis Client

Véase The Travis Client.

**login** The `login` command will ask you for your GitHub user name and password.

We can also create a GitHub token on our own and supply it via `--github-token`:

```
$ travis login --github-token xxxxxxxxxxxxxxxxxxxxxxxx --pro
Successfully logged in as crguezl!
[~/sinatra/sinatra-number_cruncher(master)]$ travis whoami
You are crguezl (Casiano Rodriguez-Leon)
```

Para obtener el token que usamos como argumento de `--github-token` nos vamos a

GitHub → settings → Applications → Generate new token

Véase el tutorial Creating an access token for command-line use.

### init

When setting up a new project, we can run `travis init` to generate a `.travis.yml` and [enable the project](#):

```
[~/sinatra/sinatra-number_cruncher(master)]$ travis init ruby --rvm 2.1.2 -r crguezl/number_cruncher
.travis.yml file created!
crguezl/number_cruncher: enabled :)
```

A file called `.travis.yml` was created that contains information about our app.

```
[~/sinatra/sinatra-number_cruncher(master)]$ cat .travis.yml
language: ruby
rvm: 2.1.2
```

**Ejecutando las pruebas en Travis** A build on Travis is initialized when we push to our Github account, so do that now:

```
[~/sinatra/sinatra-number_cruncher(master)]$ git add .travis.yml Rakefile
[~/sinatra/sinatra-number_cruncher(master)]$ git commit -am '.travis.yml added'
[master b25b8e8] .travis.yml added
 2 files changed, 7 insertions(+), 1 deletion(-)
[~/sinatra/sinatra-number_cruncher(master)]$ git push origin master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 466 bytes | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)
To git@github.com:crguezl/number_cruncher.git
 5e51ec1..b25b8e8 master -> master
```

**show** To check the status of the build, run the following command in a terminal:

```
[~/sinatra/sinatra-number_cruncher(master)]$ travis show -r crguezl/number_cruncher
no build yet for crguezl/number_cruncher
```

Travis can take a while to run a build, so you might get a message like the one above. Let us try again:

```
[~/sinatra/sinatra-number_cruncher(master)]$ travis show -r crguezl/number_cruncher
Job #1.1: .travis.yml added
State: passed
Type: push
Branch: master
Compare URL: https://github.com/crguezl/number_cruncher/compare/5e51ec1d8a3a...b25b8e8d7086
Duration: 18 sec
Started: 2014-09-24 17:09:38
Finished: 2014-09-24 17:09:56
Allow Failure: false
Config: rvm: 2.1.2
[~/sinatra/sinatra-number_cruncher(master)]$
```

This time we got confirmation that the build has passed successfully.

**logs** Given a job number, `logs` simply prints out that job's logs. By default it will display the first job of the latest build.

```
[~/sinatra/sinatra-number_cruncher(master)]$ travis logs -r crguezl/number_cruncher
displaying logs for crguezl/number_cruncher#1.1
Using worker: worker-linux-6-2.bb.travis-ci.org:travis-linux-2

$ git clone --depth=50 --branch=master git://github.com/crguezl/number_cruncher.git crguezl/nu
Cloning into 'crguezl/number_cruncher'...
remote: Counting objects: 36, done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 36 (delta 11), reused 28 (delta 7)
Receiving objects: 100% (36/36), 4.63 KiB | 0 bytes/s, done.
Resolving deltas: 100% (11/11), done.
Checking connectivity... done.
$ cd crguezl/number_cruncher
$ git checkout -qf b25b8e8d70861b82c1399f87ebab036a180bc653
couchdb stop/waiting
$ rvm use 2.1.2 --install --binary --fuzzy
.....
```

## open

```
travis open -r crguezl/number_cruncher
```

Opens the project view in the Travis CI web interface.

## Status Images

With Travis CI, you can embed little status icons into your project's `README.md` or general documentation. That way, visitors of your projects or site can immediately see its build status.

```



```

The URLs for status images are available on your repositories' page on Travis CI.

You'll find the most current status image in the top right.

Clicking that will reveal a dialog that allows you to copy and paste the URL and ready to use templates for common markup formats, like Markdown or Textile.

Make sure to select the right branch in the dropdown. The default URL, without the branch parameter, will return the status for the latest build, on any branch. Selecting the desired branch makes sure your image only displays the status for the branch whose stability is most relevant.

## Travis console

```

[~/sinatra/sinatra-selenium/capybara-selenium(master)]$ travis console
>> .ls
Gemfile README.md my_sinatra_app.rb
Gemfile.lock Rakefile spec
>> .uname -a
Darwin air.local 13.4.0 Darwin Kernel Version 13.4.0: Sun Aug 17 19:50:11 PDT 2014; root:xnu-2
>> User.current
=> #<User: crguezl>
>> Repository.find('crguezl/sinatra-capybara-selenium')
=> #<Repository: crguezl/sinatra-capybara-selenium>
>> _.last_build
=> #<Travis::Client::Build: crguezl/sinatra-capybara-selenium#13>

```

## 20.7. Coveralls: Análisis de Cubrimiento

Coveralls works with your continuous integration server to give you test coverage history and statistics.

When you log into Coveralls for the first time, it will ask you to select repositories to monitor.

It works like Travis CI, listing all of your repositories for you to enable and disable access. (You can also click a button to resynchronize the repository list, in case you've added a repository since last syncing).

To set up Coveralls, we need to add a file that tells Coverall what to do.

For our project, we need to add a file to the root directory named `.coveralls.yml`, in which we'll include a single line of configuration:

```
service_name: travis-ci
```

This tells Coveralls that we're using Travis CI as our CI server. (If you've signed up for a Pro account, then use `travis-pro` instead.)

We also need to add the Coveralls gem to our `Gemfile`

```

group :development, :test do
 gem 'rspec'
 gem 'rack-test'
 gem 'coveralls', require: false
end

```

Finally, we need to include `coveralls` code in our `spec_helper.rb` file:

```
require 'coveralls'
Coveralls.wear!
```

Notice that we have to load the code before the app code.

If you load Coveralls after the application's code has loaded, then it wouldn't be able to hook into the application properly.

## coveralls en la Línea de Comandos

```
[~/sinatra/sinatra-selenium/capybara-selenium(master)]$ coveralls --help
Commands:
 coveralls help [COMMAND] # Describe available commands or one specific command
 coveralls last # View the last build for this repository on Coveralls.
 coveralls open # View this repository on Coveralls.
 coveralls push # Runs your test suite and pushes the coverage results to Cover...
 coveralls report # Runs your test suite locally and displays coverage statistics.
 coveralls service # View this repository on your CI service's website.
 coveralls version # See version
```

## Running locally: COVERALLS\_REPO\_TOKEN

When you run the client you get:

```
[~/sinatra/sinatra-coverall(master)]$ coveralls open
No repo_token configured.
```

If you're running locally, you must have a `.coveralls.yml` with your `repo_token` in it; or, you must provide a `COVERALLS_REPO_TOKEN` environment-variable on the command-line.

```
[~/sinatra/sinatra-coverall(master)]$ cat .coveralls.yml.bak
service_name: travis-ci
repo_token: HHhhhhhh9saXXXXZZZ4444444444xdWh
```

or

```
export COVERALLS_REPO_TOKEN=HHhhhhhh9saXXXXZZZ4444444444xdWh
```

You can get the token in your coveralls repo page on the right side in the section REPO TOKEN.

## Donde

- [~/sinatra/sinatra-coverall(master)]\$ pwd -P  
`/Users/casiano/local/src/ruby/sinatra/sinatra-coverall`
- `crguezl/sample-ruby-project`

## See

- <https://coveralls.io/>
- CoverAlls documentation
- Ejemplo: Ruby, Heroku and Groonga tutorial 2014
- `crguezl/sample-ruby-project`

# Capítulo 21

## Guard

Guard is a gem that allows you to observe your project's files for change and perform actions based on that change. The first thing you will need to do is add Guard (and its helpers for Bundler and RSpec) to the end of your Gemfile:

```
[~/srcLPPclases/question-simple_choice(master)]$ vi Guardfile

source 'https://rubygems.org'

Specify your gem's dependencies in question-simple_choice.gemspec

group :development, :test do
 gem 'rspec'
 gem 'pry'
 gem 'guard'
 # do not call require when you start Bundler
 gem 'guard-rspec', require: false
end
```

You should now run bundle install in your project root to install Guard if it isn't already.

```
[~/srcLPPclases/question-simple_choice(master)]$ guard --version
Guard version 2.8.0
```

```
[~/srcLPPclases/question-simple_choice(master)]$ guard help
Commands:
 guard help [COMMAND] # Describe available commands or one specific command
 guard init [GUARDS] # Generates a Guardfile at the current directory (if it i...
 guard list # Lists Guard plugins that can be used with init
 guard notifiers # Lists notifiers and its options
 guard show # Show all defined Guard plugins and their options
 guard start # Starts Guard
 guard version # Show the Guard version
```

Now we need to initialize Guard and configure it for our project. Luckily, Guard comes with its own command line helpers:

```
[~/srcLPPclases/question-simple_choice(master)]$ guard help init
Usage:
 guard init [GUARDS]
```

Options:

```
-b, [--bare=Generate a bare Guardfile without
```

```
adding any installed plugin into it], [--no-bare]
```

Generates a Guardfile at the current directory (if it is not already there) and adds all installed Guard plugins or the given GUARDS into it

Si solo queremos generar código para vigilar el Gemfile usando guard-bundler:

```
[~/srcLPPclases/question-simple_choice(master)]$ guard init rspec
16:18:14 - INFO - Writing new Guardfile to
 /Users/casiano/local/src/ruby/LPP/clases/question-simple_choice/Guardfile
16:18:14 - INFO - rspec guard added to Guardfile,
 feel free to edit it

[~/srcLPPclases/question-simple_choice(master)]$ cat Guardfile
A sample Guardfile
More info at https://github.com/guard/guard#readme

Note: The cmd option is now required due to the increasing number of ways
rspec may be run, below are examples of the most common uses.
* bundler: 'bundle exec rspec'
* bundler binstubs: 'bin/rspec'
* spring: 'bin/rsspec' (This will use spring if running and you have
installed the spring binstubs per the docs)
* zeus: 'zeus rspec' (requires the server to be started separately)
* 'just' rspec: 'rspec'
guard :rspec, cmd: 'bundle exec rspec' do
 watch(%r{^spec/.+_spec\.rb$})
 watch(%r{^lib/(.+)\.rb$}) { |m| "spec/lib/#{m[1]}_spec.rb" }
 watch('spec/spec_helper.rb') { "spec" }

 # Rails example
 watch(%r{^app/(+)\.rb$}) { |m| "spec/#{m[1]}_spec.rb" }
 watch(%r{^app/(.*)\.(erb|haml|slim)$}) { |m| "spec/#{$<1>#{m[1]}_spec.rb" }
 watch(%r{^app/controllers/(.+)_controller\.rb$}) { |m| ["spec/routing/#{m[1]}_routing_spec.rb"] }
 watch(%r{^spec/support/(+)\.rb$}) { "spec" }
 watch('config/routes.rb') { "spec/routing" }
 watch('app/controllers/application_controller.rb') { "spec/controllers" }
 watch('spec/rails_helper.rb') { "spec" }

 # Capybara features specs
 watch(%r{^app/views/(.+)/(.*\.(erb|haml|slim)$)}) { |m| "spec/features/#{m[1]}_spec.rb" }

 # Turnip features and steps
 watch(%r{^spec/acceptance/(+)\.feature$})
 watch(%r{^spec/acceptance/steps/(+)_steps\.rb$}) { |m| Dir[File.join("**/#{m[1]}.feature", "*/step_definitions/*.rb")].sort }
end
```

Si solo queremos generar código para vigilar las pruebas usando guard-rspec:

```
guard init rspec
```

1. Guard works similarly to Bundler and Rake by creating a Guardfile in your project's root directory.
2. These commands automatically add example configuration for each guard type to the Guardfile (after guard init creates the file to begin with).

3. While I'm going to tell you explicitly what to put in your Guardfile, the `init` commands can really help jog your memory if you're trying to do it from scratch.

Let's modify our Guardfile to look like this:

```
guard 'bundler' do
 watch('Gemfile')
end

guard 'rspec', :version => 2 do
 watch(%r{^spec/.+_spec\.rb$})
 watch(%r{^lib/(.+)\.rb$}) { |m| "spec/#{$m[1]}_spec.rb" }
 watch('spec/spec_helper.rb') { "spec" }
end
```

*Guard works by watching certain files in your project and then performing actions when those files change.*

1. The first guard

```
guard 'bundler' do
 watch('Gemfile')
end
```

tells us to watch for the `Gemfile` to change, and run `bundle install` when that happens.

2. The second guard tells us to watch all of our RSpec test files, our `spec_helper`, and the corresponding library files for each of our RSpec tests.
3. This means that when you change a file, Guard can automatically re-run the tests for that specific file only.
4. Now that your Guardfile is properly configured, you can just run `bundle exec guard start` from your project's root directory.

```
[~/srcLPPuby/building_gems/bleigh_tutorial_to_building_gems/my-gem(master)]$ guard help
Usage:
 guard start
```

Options:

```
-c, [--clear=Auto clear shell before each action]
-n, [--notify=Notifications feature]
-d, [--debug>Show debug information]
-g, [--group=Run only the passed groups]
-P, [--plugin=Run only the passed plugins]
-w, [--watchdir=Specify the directories to watch]
-G, [--guardfile=Specify a Guardfile]
-i, [--no-interactions=Turn off completely any Guard terminal interactions]
-B, [--no-bundler-warning=Turn off warning when Bundler is not present]
 [--show-deprecations=Turn on deprecation warnings]
-l, [--latency=Overwrite Listen's default latency]
-p, [--force-polling=Force usage of the Listen polling listener]
```

Starts Guard

5. When executing `guard`, it shows a `pry` console, with some Guard specific `pry` commands:

```
[~/srcLPPruby/building_gems/bleighTutorial_to_building_gems/my-gem(master)]$ guard
17:41:40 - INFO - Guard here! It looks like your project has a Gemfile, yet you are running
> [#] 'guard' outside of Bundler. If this is your intent, feel free to ignore this
> [#] message. Otherwise, consider using 'bundle exec guard' to ensure your
> [#] dependencies are loaded correctly.
> [#] (You can run 'guard' with --no-bundler-warning to get rid of this message.)
17:41:40 - WARN - Guard::RSpec DEPRECATION WARNING: The :version option is deprecated. On
17:41:40 - INFO - Guard is using TerminalTitle to send notifications.
17:41:40 - INFO - Bundle already up-to-date
17:41:40 - INFO - Guard::RSpec is running
17:41:40 - INFO - Guard is now watching at '/Users/casiano/local/src/ruby/building_gems/bi
[1] guard(main)>
```

This will get `Guard` up and running and you will now automatically run tests and re-bundle as you build your application.

Si ahora en una terminal modificamos uno de los ficheros vigilados:

```
$ touch spec/spec_helper.rb
```

`Guard` dispara la correspondiente acción, que en este caso es ejecutar `RSpec`:

```
17:50:28 - INFO - Running: spec
Run options: include {:focus=>true}

All examples were filtered out; ignoring {:focus=>true}
*
```

Pending:

```
MyGem requires additional testing
Not yet implemented
./spec/my_gem_spec.rb:5
```

```
Finished in 0.00062 seconds
1 example, 0 failures, 1 pending
```

Randomized with seed 46343

```
[1] guard(main)>
```

Running `Guard` makes the development feedback loop as tight and automatic as possible.

## Enlaces Relacionados

Gemas

1. [crguezl / bleigh-tutorial-to-building-gems en GitHub](#)

Testing

1. [Polishing Rubies \(Part 3\): Tools for Testing](#)

Guard

1. [Guard en GitHub](#)

2. [Guard Wiki](#)

3. [Hire a Guard for Your Project](#)

4. Automate your application's development environment with Guard by Michael van Rooijen
5. Guard Rspec
6. guard-gitpusher
7. Guard and Rack-livereload With Sinatra
8. Guard is Your Best Friend Jeffrey Way NetTuts Screencast

## Capítulo 22

# Programación Orientada a Eventos

### 22.1. Introducción a Programación Orientada a Eventos

De la Wikipedia ():

In computer programming, *event-driven programming (EDP)* or *event-based programming* is a programming paradigm in which the flow of the program is determined by events—e.g., sensor outputs or user actions (mouse clicks, key presses) or messages from other programs or threads.

Event-driven programming can also be defined as an application architecture technique in which the application has a main loop which is clearly divided down to two sections:

1. the first is *event selection* (or event detection)
2. the second is *event handling*.

In *embedded systems* the same may be achieved using interrupts instead of a constantly running main loop; in that case the former portion of the architecture resides completely in computer hardware.

Event-driven programming is similar to *data-driven programming*, in that both are structured as pattern matching and resulting processing, though they are typically applied to different domains.

Event-driven programs can be written in any language, although the task is easier in languages that provide high-level abstractions, such as closures 13.6

De Ruby's EventMachine – Part 1 : Event-based Programming by Phil Whelan on September 14, 2012:

Within an event-based application, everything must be non-blocking to get the real benefit of event-based programming.

The event-loop is running in a single thread within a single process.

Therefore, the first time you do something that is blocking, it causes the whole event loop to stop and wait for you.

That means while you are talking to your database for *just 10 milliseconds*, nothing else happens.

You would be surprised at how many operations you can perform in those 10ms of halting your entire event loop.

In that time, potentially hundreds of connections, that only hit RAM, could have come and gone, but instead they get queued waiting for your blocking action.

## 22.2. Building our own I/O Event Loop

In this article in <https://practicingruby.com> the authors implement an I/O event loop. They capture the core part of EventMachine/Node.js/Twisted in about 150 lines of Ruby.

1. Event loops demystified

## 22.3. EventMachine

Event::Machine is an event-driven I/O and lightweight concurrency library for Ruby. It provides event-driven I/O using the *Reactor pattern*.

The reactor is the core of Event::Machine. All of EM's processing happens in the reactor. Your application will create code that hooks into the reactor, using timers, socket connections or various other means. Typically you'll wrap your entire application inside an EM#run block.

```
EventMachine.run do
 EventMachine.start_server ...
end
```

This block won't complete until EM#stop is called. The reactor will loop infinitely.

**run y stop\_event\_loop**

```
[~/ruby/eventmachine/simple(master)]$ cat timer.rb
require 'rubygems'
require 'eventmachine'
puts "Start"
EM.run do
 puts "Init"
 EM.add_timer(1) do
 puts "Quitting"
 EM.stop_event_loop
 end
end
puts "Done"
```

```
[~/ruby/eventmachine/simple(master)]$ ruby timer.rb
Start
Init
Quitting
Done
```

**reactor\_running?**

```
[~/ruby/eventmachine/simple(master)]$ cat reactor_running.rb
require 'rubygems'
require 'eventmachine'
puts EM.reactor_running? # false
EM.run do
 puts EM.reactor_running? # true
 EM.next_tick do
 puts EM.reactor_running? # true
 EM.stop
 end
end
puts EM.reactor_running? # false
```

1. `reactor_running?` Tells you whether the EventMachine reactor loop is currently running.

Useful when writing libraries that want to run event-driven code, but may be running in programs that are already event-driven.

In such cases, if `reactor_running?` returns `false`, your code can invoke `run` and run your application code inside the block passed to that method. If this method returns true, just execute your event-aware code.

2. `(Object) next_tick(pr = nil, &block)`

Schedules a proc for execution immediately after the next "turn" through the reactor core.

## add\_timer

```
[~/ruby/eventmachine/simple(master)]$ cat add_timer_family.rb
require "eventmachine"
EM.run do
 EM.add_periodic_timer(1) do
 puts "Tick..."
 end
 EM.add_timer(5) do
 puts "BOOM"
 EM.stop_event_loop
 end
end
[~/ruby/eventmachine/simple(master)]$ ruby add_timer_family.rb
Tick...
Tick...
Tick...
Tick...
BOOM
```

1. `(Object) add_timer(*args, &block)`

Adds a one-shot timer to the event loop. Call it with one or two parameters. The first parameter is a delay-time expressed in seconds (not milliseconds). The second parameter, if present, must be an object that responds to `:call`. If 2nd parameter is not given, then you can also simply pass a block to the method call.

This method may be called from the block passed to `run` or from any callback method. It schedules execution of the `proc` or block passed to it, after the passage of an interval of time equal to at least the number of seconds specified in the first parameter to the call.

`add_timer` is a non-blocking method. Callbacks can and will be called during the interval of time that the timer is in effect. There is no built-in limit to the number of timers that can be outstanding at any given time.

2. `(Object) add_periodic_timer(*args, &block)`

Adds a periodic timer to the event loop. It takes the same parameters as the one-shot timer method, `add_timer`. This method schedules execution of the given block repeatedly, at intervals of time at least as great as the number of seconds given in the first parameter to the call.

## next\_tick

```
[~/ruby/eventmachine/simple(master)]$ cat next_tick.rb
require 'eventmachine'
EM.run do
 EM.add_periodic_timer(1) do
```

```

 puts "Hai"
end
EM.add_timer(5) do
 EM.next_tick do
 EM.stop_event_loop
 end
end
end

```

(Object) `next_tick(pr = nil, &block)` Schedules a proc for execution immediately after the next “turn”through the reactor core.

This can be useful for improving memory management and/or application responsiveness, especially when scheduling large amounts of data for writing to a network connection.

This method takes either a single argument (which must be a callable object) or a block.

Parameters:

`pr (#call) (defaults to: nil) | A callable object to run`

```
[~/ruby/eventmachine/simple(master)]$ ruby next_tick.rb
Hai
Hai
Hai
Hai
```

**defer** EM.defer is used to push a otherwise blocking operation on an EM internal queue whose jobs are then processed by a thread pool of 20 threads (by default).

You can use EM.defer with operations which aren’t made to work asynchronously.

```
[~/ruby/eventmachine/simple(master)]$ cat defer.rb
require 'eventmachine'
require 'thread'
EM.run do
 EM.add_timer(2) do
 puts "Main #{Thread.current}"
 EM.stop_event_loop
 end
 EM.defer(proc do
 puts "Defer #{Thread.current}"
 end)
end
```

(Object) `defer(op = nil, callback = nil, &blk)`

1. Event::Machine.defer is used for integrating blocking operations into Event::Machine’s control flow.
2. The action of `defer` is to take the block specified in the first parameter (the “operation”) and schedule it for asynchronous execution on an internal thread pool maintained by Event::Machine.
3. When the operation completes, it will pass the result computed by the block (if any) back to the Event::Machine reactor.
4. Then, Event::Machine calls the block specified in the second parameter to defer (the “callback”), as part of its normal event handling loop.
5. The result computed by the operation block is passed as a parameter to the callback.

6. You may omit the callback parameter if you don't need to execute any code after the operation completes.

```
[~/ruby/eventmachine/simple(master)]$ ruby defer.rb
Defer #<Thread:0x007fd9c3826960>
Main #<Thread:0x007fd9c38bcd98>
```

#### **next\_tick versus defer**

1. The loop runs on a single thread.
2. What this means is that apart from I/O - at any time, only one task/event is processed by the event loop.
3. You can imagine this event loop to be a queue of callbacks that are processed on every tick of the event loop.
4. So, even if you are running on a multi-core machine, you will not get any parallelism in terms of actual processing - all events will be processed only one at a time.
5. For every I/O bound task, you can simply define a callback that will get added to the event queue.
6. The callback will fire when the I/O operation is done, and in the mean time, the application can continue to process other I/O bound requests.
7. Given this model, what `process.nextTick()` actually does is defer the execution of an action till the next pass around the event loop.

#### **22.3.1. Un server**

```
~/ruby/eventmachine/simple_server]$ cat server.rb
require 'eventmachine'

module EchoServer
 def post_init
 puts "-- someone connected to the echo server!"
 end

 def receive_data data
 send_data ">>>you sent: #{data}"
 puts "received #{data}"
 close_connection if data =~ /quit/i
 end

 def unbind
 puts "-- someone disconnected from the echo server!"
 end
end

Note that this will block current thread.
EventMachine.run {
 EventMachine.start_server "127.0.0.1", 8081, EchoServer
}
```

```
[~/ruby/eventmachine/simple_server]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
 sh "ruby server.rb"
end

SERVER = "127.0.0.1 8081"
M = %q{
And now, the end is near
And so I face the final curtain
My friend, I'll say it clear
I'll state my case, of which I'm certain
I've lived a life that's full
I traveled each and ev'ry highway
And more, much more than this, I did it my way
quit
}.split(/\n/)

desc "run a client"
task :client do
 IO.popen("telnet #{SERVER}", "r+",
 :external_encoding=>"utf-8",
 :err=>[:child, :out]) do |chan|
 3.times do
 output = chan.gets
 puts "SERVER says: #{output}"
 end

 M.each do |line|
 puts "client dending #{line}"
 chan.puts line
 output = chan.gets
 puts "SERVER says: #{output}"
 end
 end # popen
end

[~/ruby/eventmachine/simple_server]$ rake
ruby server.rb

[~/ruby/eventmachine/simple_server]$ rake client
SERVER says: Trying 127.0.0.1...
SERVER says: Connected to localhost.
SERVER says: Escape character is '^]'.
client dending
SERVER says: >>>you sent:
client dending And now, the end is near
SERVER says: >>>you sent: And now, the end is near
client dending And so I face the final curtain
SERVER says: >>>you sent: And so I face the final curtain >>>you sent:
client dending My friend, I'll say it clear
SERVER says: >>>you sent: My friend, I'll say it clear
```

```

client dending I'll state my case, of which I'm certain
SERVER says: >>>you sent: I'll state my case, of which I'm certain
client dending I've lived a life that's full
SERVER says: >>>you sent: I've lived a life that's full >>>you sent:
client dending I traveled each and ev'ry highway
SERVER says: >>>you sent: I traveled each and ev'ry highway
client dending And more, much more than this, I did it my way
SERVER says: >>>you sent: And more, much more than this, I did it my way
client dending quit
SERVER says: Connection closed by foreign host.

```

```

[~/ruby/eventmachine/simple_server]$ rake
ruby server.rb
-- someone connected to the echo server!
received
received And now, the end is near
received And so I face the final curtain
received
received My friend, I'll say it clear
received I'll state my case, of which I'm certain
received I've lived a life that's full
received
received I traveled each and ev'ry highway
received And more, much more than this, I did it my way
received quit
-- someone disconnected from the echo server!

```

### callback y errback

```

[~/ruby/eventmachine/simple(master)]$ cat callback_errback.rb
require 'rubygems'
require 'eventmachine'
require 'em-http'

urls = ARGV
if urls.size < 1
 puts "Usage: #{$0} <url> <url> <...>"
 exit
end

pending = urls.size

EM.run do
 urls.each do |url|
 http = EM::HttpRequest.new(url).get
 http.callback {
 puts "#{url}\n#{http.response_header.status} - #{http.response.length} bytes\n"
 puts http.response

 pending -= 1
 EM.stop if pending < 1
 }
 http.errback {
 puts "#{url}\n" + http.error
 }
 end
end

```

```

 pending -= 1
 EM.stop if pending < 1
 }
end
end

[~/ruby/eventmachine/simple(master)]$ ruby callback_errback.rb 'http://www.google.com'
http://www.google.com
302 - 258 bytes
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
here.
</BODY></HTML>
```

### 22.3.2. Deferrable

Event::Machine's Deferrable borrows heavily from the deferred object in Python's Twisted event-handling framework. Here's a minimal example that illustrates Deferrable:

```

require 'eventmachine'

class MyClass
 include EM::Deferrable

 def print_value x
 puts "MyClass instance received #{x}"
 end
end

EM.run {
 df = MyClass.new
 df.callback { |x|
 df.print_value(x)
 EM.stop
 }

 EM::Timer.new(2) {
 df.set_deferred_status :succeeded, 100
 }
}
```

This program will spin for two seconds, print out the string `MyClass instance received 100` and then exit.

The Deferrable pattern allows you to specify any number of Ruby code blocks (callbacks or errbacks) that will be executed at some future time when the status of the Deferrable object changes.

How might that be useful?

1. Well, imagine that you're implementing an HTTP server,
2. but you need to make a call to some other server in order to fulfill a client request.
3. When you receive a request from one of your clients, you can create and return a Deferrable object.

4. Some other section of your program can add a callback to the Deferrable that will cause the client's request to be fulfilled.
5. Simultaneously, you initiate an event-driven or threaded client request to some different server.
6. And then your EM program will continue to process other events and service other client requests.
7. When your client request to the other server completes some time later, you will call the `set_deferred_status` method on the Deferrable object, passing either a `success` or `failure` status, and an arbitrary number of parameters (which might include the data you received from the other server).
8. At that point, the status of the Deferrable object becomes known, and its callback or errback methods are immediately executed.
9. Callbacks and errbacks are code blocks that are attached to Deferrable objects at any time through the methods `callback` and `errback`.
10. The deep beauty of this pattern is that it decouples the disposition of one operation (such as a client request to an outbound server) from the subsequent operations that depend on that disposition (which may include responding to a different client or any other operation).
11. The code which invokes the deferred operation (that will eventually result in a success or failure status together with associated data) is completely separate from the code which depends on that status and data.
12. This achieves one of the primary goals for which threading is typically used in sophisticated applications, with none of the nondeterminacy or debugging difficulties of threads.
13. As soon as the deferred status of a Deferrable becomes known by way of a call to `set_deferred_status`, the Deferrable will IMMEDIATELY execute all of its callbacks or errbacks in the order in which they were added to the Deferrable.
14. Callbacks and errbacks can be added to a Deferrable object at any time, not just when the object is created. They can even be added after the status of the object has been determined! (In this case, they will be executed immediately when they are added.)
15. A call to `Deferrable#set_deferred_status` takes `:succeeded` or `:failed` as its first argument. (This determines whether the object will call its callbacks or its errbacks.) `set_deferred_status` also takes zero or more additional parameters, that will in turn be passed as parameters to the callbacks or errbacks.
16. In general, you can only call `set_deferred_status` ONCE on a Deferrable object. A call to `set_deferred_status` will not return until all of the associated callbacks or errbacks have been called. If you add callbacks or errbacks AFTER making a call to `set_deferred_status`, those additional callbacks or errbacks will execute IMMEDIATELY. Any given callback or errback will be executed AT MOST once.
17. It's possible to call `set_deferred_status` AGAIN, during the execution a callback or errback. This makes it possible to change the parameters which will be sent to the callbacks or errbacks farther down the chain, enabling some extremely elegant use-cases. You can transform the data returned from a deferred operation in arbitrary ways as needed by subsequent users, without changing any of the code that generated the original data.
18. A call to `set_deferred_status` will not return until all of the associated callbacks or errbacks have been called. If you add callbacks or errbacks AFTER making a call to `set_deferred_status`, those additional callbacks or errbacks will execute IMMEDIATELY.

```
[~/ruby/eventmachine/deferrable(master)]$ cat defaultdeferrable.rb
require 'eventmachine'
EM.run do
 df = EM::DefaultDeferrable.new
 df.callback do |x|
 puts "got #{x}"
 end
 df.callback do |x|
 EM.stop
 end
 EM.add_timer(1) do
 df.set_deferred_status :succeeded, "monkeys"
 end
end
```

DefaultDeferrable is an otherwise empty class that includes Deferrable.

This is useful when you just need to return a Deferrable object as a way of communicating deferred status to some other part of a program.

```
[~/ruby/eventmachine/deferrable(master)]$ ruby defaultdeferrable.rb
got monkeys
```

### 22.3.3. Véase

1. Ruby EventMachine - The Speed Demon by Ilya Grigorik on May 27, 2008
2. EventMachine Introductions, 27 Feb 2009

### 22.4. Véase

- Adam Hawkins - Dear God What Am I Doing? Concurrency and Parallel Processing
- Tony Arcieri: Concurrent Programming with Celluloid

## Capítulo 23

# Programación distribuída/Distributed Programming

### 23.1. DRb

*Distributed Ruby* or *DRb* allows Ruby programs to communicate with each other on the same machine or over a network. DRb uses remote method invocation (RMI) to pass commands and data between processes

1. Wikibooks: RubyProgrammin/Standard Library/DRb
2. drb: Ruby Standard Library Documentation

### 23.2. Distributed Ruby and SSH: drbssh

drbssh: A protocol implementation for Distributed Ruby (DRb), supporting SSH-connections.

```
[~/ruby/distributedRuby]$ cat drbssh_example.rb
require 'drbssh'

DRb.start_service 'drbssh://'
remote = DRbObject.new_with_uri("drbssh://banot.etsii.ull.es/")
s = remote.eval(%q{
 {
 :uname => `uname -a`,
 :pwd => `pwd`,
 :version => `ruby -v`
 }
})
puts s.inspect
```

#### Ejecución

```
[~/ruby/distributedRuby]$ ruby drbssh_example.rb
{
 :uname=>"Linux banot.etsii.ull.es 2.6.18-308.24.1.el5 #1 SMP Tue Dec 4 17:42:30 EST 2012 i686
 :pwd=>"/home/casiano\n",
 :version=>"ruby 1.9.2dev (2010-07-11 revision 28618) [i686-linux]\n"
}
```

# Capítulo 24

## Actores

### 24.1. Celluloid

Celluloid provides a simple and natural way to build fault-tolerant concurrent programs in Ruby. With Celluloid, you can build systems out of concurrent objects just as easily as you build sequential programs out of regular objects.

Much of the difficulty with building concurrent programs in Ruby arises because the object-oriented mechanisms for structuring code, such as classes and inheritance, are separate from the concurrency mechanisms, such as threads and locks. Celluloid combines these into a single structure, an active object running within a thread, called an “actor”, or in Celluloid vernacular, a “cell”.

The Actor Model can be summarized as follows:

1. Actors communicate by sending messages
2. Every actor has a mailbox with a unique address.
3. If you have an actor’s address, you can send it messages
4. Actors can create other actors, and when they do they know their address so they can send the newly-created actors messages
5. Actors can pass addresses (or handles) to themselves or other actors in messages

#### 24.1.1. DCell

Objects can message objects transparently that live on other machines over the network, and you don’t have to worry about the networking gunk, and you don’t have to worry about finding them, and you don’t have to worry about anything. It’s just as if you messaged an object that’s right next door.

Steve Jobs. 1995

DCell is a simple and easy way to build distributed applications in Ruby. Somewhat similar to DRb, DCell lets you easily expose Ruby objects as network services, and call them remotely just like you would any other Ruby object. However, unlike DRb all objects in the system are concurrent. You can create and register several available services on a given node, obtain handles to them, and easily pass these handles around the network just like any other objects.

DCell is a distributed extension to Celluloid, which provides concurrent objects for Ruby with many of the features of Erlang, such as the ability to supervise objects and restart them when they crash, and also link to other objects and receive event notifications of when they crash. This makes it easier to build robust, fault-tolerant distributed systems.

DCell uses the 0MQ messaging protocol which provides a robust, fault-tolerant brokerless transport for asynchronous messages sent between nodes. DCell is built on top of the Celluloid::ZMQ library, which provides a Celluloid-oriented wrapper around the underlying ffi-rzmq library.

```
[~/ruby/celluloid]$ cat itchy.rb
#!/usr/bin/env ruby
require 'dcell'

DCell.start :id => "itchy", :addr => "tcp://127.0.0.1:9001"

class Itchy
 include Celluloid

 def initialize
 puts "Ready for mayhem!"
 @n = 0
 end

 def fight
 @n = (@n % 6) + 1
 if @n <= 3
 puts "Bite!"
 else
 puts "Fight!"
 end
 end
end

Itchy.supervise_as :itchy
sleep
```

To configure an individual DCell node, we need to give it a node ID and a 0MQ address to bind to. Node IDs look like domain names, and 0MQ addresses look like URLs that start with `tcp`:

```
DCell.start :id => "itchy", :addr => "tcp://127.0.0.1:9001"

[~/ruby/celluloid]$ cat scratchy.rb
#!/usr/bin/env ruby
require 'dcell'

DCell.start :id => "scratchy", :addr => "tcp://127.0.0.1:9002"
itchy_node = DCell::Node["itchy"]

puts "Fighting itchy! (check itchy's output)"

6.times do
 itchy_node[:itchy].fight
 sleep 1
end
```

To create a cluster, we need to start another Ruby VM and connect it to the first VM.

```
DCell.start :id => "scratchy", :addr => "tcp://127.0.0.1:9002"
itchy_node = DCell::Node["itchy"]
```

Once you have a cluster of multiple nodes, you can bootstrap additional nodes into the cluster by pointing them at any node, and all nodes will gossip about the newly added node.

## 24.2. El Problema de los Filósosofos Comensales / The Dining Philosophers Problem

The *Dining Philosophers problem* was formulated by *Edsger Djisktra* in 1965 to illustrate the kind of issues we can find when multiple processes compete to gain access to exclusive resources.

1. A gentle introduction to actor-based concurrency.
  2. Source code from this article. All of the code from this article is in Practicing Ruby's example repository, but the links below highlight the main points of interest:
    - A solution that leads to deadlocks

- A coordinated mutex-based solution
  - An actor-based solution using Celluloid
  - An actor-based solution using a hand-rolled actor library
  - Minimal implementation of the actor model
  - Chopsticks class definition
  - Table class definition
3. Scaling Ruby with Actors, or How I Learned to Stop Worrying and Love Threads by Mike Perham Rubyconf 2011

### 24.3. Véase

- celluloid
- An Introduction to Celluloid, Part I
- An Introduction to Celluloid, Part III
- An Introduction to Celluloid, Part II
- Concurrent Programming with Celluloid by Tony Arcieri YouTube
- Ruby Conf 12 - The Celluloid Ecosystem by Tony Arcieri
- Evented Ruby vs Node.js by Jerry Cheung
- 2012 Introducing DCell: actor-based distributed objects for Ruby TONY ARCIERI.
- Scaling Ruby with Actors, or How I Learned to Stop Worrying and Love Threads by Mike Perham Rubyconf 2011
- A gentle introduction to actor-based concurrency

# Capítulo 25

## Threads

### 25.1. Enlaces /Véase

- Threads in Ruby. Imran Latif. Published October 14, 2013 SitePoint.
- Ruby Multithreading TutorialsPoint
- Ruby TCP Chat SitePoint

### 25.2. Creación de Threads

#### new

1. Ruby makes it easy to write multi-threaded programs with the `Thread` class
2. To start a new thread, just associate a block with a call to `Thread.new`
3. A new thread will be created to execute the code in the block,
4. the original thread will return from `Thread.new` immediately and resume execution with the next statement

```
arr = []
a, b, c = 1, 2, 3
Thread.new(a,b,c) { |d,e,f| arr << d << e << f }.join
arr #=> [1, 2, 3]
```

A `ThreadError` exception is raised if `::new` is called without a block.

#### join

1. When a thread calls `thr.join` The calling thread will suspend execution and run `thr`.
2. The calling thread does not return until `thr` exits or until `limit` seconds have passed if we have used `thr.join(limit)`.
3. If the time `limit` expires, `nil` will be returned, otherwise `thr` is returned.
4. Any threads not joined will be killed when the main program exits.
5. If `thr` had previously raised an exception and the exception has not yet been processed, it will be processed at this time.

```
[~/ruby/threads(master)]$ cat join_with_timer.rb
require 'thread'

y = Thread.new { 4.times { sleep 0.2; puts 'tick... ' } }
puts "Waiting" until y.join(0.1)

[~/ruby/threads(master)]$ ruby join_with_timer.rb
Waiting
tick...
Waiting
Waiting
tick...
Waiting
Waiting
tick...
Waiting
Waiting
tick...
```

## Localidad y Privacidad

A thread can normally access any variables that are in scope when the thread is created.

Variables local to the block containing the thread code are local to the thread and are not shared.

Class `Thread` features a special facility that allows thread-local variables to be created and accessed by name.

You simply treat the thread object as if it were a `Hash`, writing to elements using `[] =` and reading them back using `[]`.

Veamos un ejemplo:

```
[~/ruby/threads(master)]$ cat sharing.rb
require 'thread'
require 'barrier'

numthreads = 4
lastthread = numthreads-1
th = []
b = Barrier.new(numthreads)
(0..lastthread).each do |i|
 th[i] = Thread.new(i) do
 b.wait
 th[(i+1)%numthreads][:my_x] = i
 end
end
th.each do |t|
 t.join
 print t[:my_x].to_s+"\n"
end
```

Necesitamos la barrera proveida por `Barrier` para asegurarnos de que todas las threads han sido creadas en el momento en el que cada thread escribe en la variable `my_x` de la thread siguiente.

La ejecución resulta en:

```
[~/ruby/threads(master)]$ ruby sharing.rb
3
0
1
2
```

## 25.3. Exclusión Mutua y la Clase Mutex

### Race Condition: definición

De la Wikipedia sobre Race\_condition

A race condition or race hazard is the behavior of an electronic or software system where the output is dependent on the sequence or timing of other uncontrollable events.

It becomes a bug when events do not happen in the order the programmer intended.

The term originates with the idea of two signals racing each other to influence the output first.

### Ejemplo de Race Condition

```
[~/ruby/threads(master)]$ cat race_condition.rb
def inc(n)
 n+1
end
sum = 0
threads = (1..10).map do
 Thread.new do
 10_000.times do
 sum = inc(sum)
 end
 end
end
threads.each(&:join)
```

**Ventajas de Usar JRuby Cuando se Usan Threads** JRuby is built upon the JVM. One can make use of "true" native threads which are able to run concurrently with your parent thread (See Parallelism is a Myth in Ruby by Ilya Grigorik on November 13, 2008).

A *Global Interpreter Lock (GIL)* is a mutual exclusion lock held by a programming language interpreter thread to avoid sharing code that is not *thread-safe* with other threads.

Some language implementations that implement a Global Interpreter Lock are CPython, the most widely used implementation of Python, and Ruby MRI, the reference implementation of Ruby.

JVM-based equivalents of these languages (Jython and JRuby) do not use Global Interpreter Locks. JRuby is a Ruby implementation allowing us to natively scale our Ruby code across multiple cores. IronPython and IronRuby are implemented on top of Microsoft's Dynamic Language Runtime and also avoid using a GIL.

Also, Rubinius implements native operating system threads for concurrency and has no global interpreter lock (GIL).

### Ejecución con JRuby

```
[~/ruby/threads(master)]$ rvm list

rvm rubies

jruby-1.7.3 [x86_64]
*= ruby-2.0.0-p247 [x86_64]
 ruby-2.1.0-preview1 [x86_64]

=> - current
*= - current && default
* - default
```

```
[~/ruby/threads(master)]$ rvm use jruby-1.7.3
Using /Users/casiano/.rvm/gems/jruby-1.7.3
[~/ruby/threads(master)]$ ruby -v
jruby 1.7.3 (1.9.3p385) 2013-04-17 ffffff on Java HotSpot(TM) 64-Bit Server VM 1.7.0_25-b15 [x86_64]
[~/ruby/threads(master)]$ ruby race_condition.rb
28375
[~/ruby/threads(master)]$ ruby race_condition.rb
25464
[~/ruby/threads(master)]$ ruby race_condition.rb
23442
```

## Ejecucion con ruby MRI

```
[~/ruby/threads(master)]$ rvm use 2.1.0-preview1
Using /Users/casiano/.rvm/gems/ruby-2.1.0-preview1
[~/ruby/threads(master)]$ ruby race_condition.rb
100000
[~/ruby/threads(master)]$ ruby race_condition.rb
100000
[~/ruby/threads(master)]$
```

## Mutex

We can use the built-in class `Mutex` to create synchronized regions—areas of code that only one thread may enter at a time.

We resolve problems like these by using a cooperative locking mechanism:

1. Each thread that wants to access shared data must first `lock` that data
2. The `lock` is represented by a `Mutex` (short for "mutual exclusion") object
3. To `lock` a `Mutex`, you call its `lock` method
4. When you're done reading or altering the shared data, you call the `unlock` method of the `Mutex`
5. The `lock` method blocks when called on a `Mutex` that's already locked, and it does not return until the caller has successfully obtained a `lock`
6. If each thread that accesses the shared data locks and `unlocks` the `Mutex` correctly, no thread will see the data in an inconsistent state and we won't have problems like those we've described
7. Instead of using the `lock` and `unlock` methods explicitly, it is more common to use the `synchronize` method and associate a block with it
8. `synchronize` locks the `Mutex`, runs the code in the block, and then `unlocks` the `Mutex` in an `ensure` clause so that exceptions are properly handled

```
[~/ruby/threads(master)]$ cat mutex_synchronize.rb
Wait for all threads (other than the current thread and
main thread) to stop running.
Assumes that no new threads are started while waiting.
def join_all
 main = Thread.main # The main thread
 current = Thread.current # The current thread
 all = Thread.list # All threads still running
 # Now call join on each thread
 all.each { |t| t.join unless t == current or t == main }
```

```

end

def inc(n)
 n+1
end
sum = 0
mutex = Mutex.new
threads = (1..10).map do
 Thread.new do
 10_000.times do
 mutex.synchronize do
 sum = inc(sum)
 end
 end
 end
end
join_all
p sum

```

## 25.4. Un Servidor Multithreaded (and a Client)

An almost canonical use case for threads is for writing servers that can communicate with more than one client at a time.

```
[~/ruby/threads(master)]$ cat simple_server.rb
require 'socket'

This method expects a socket connected to a client.
It reads lines from the client, reverses them and sends them back.
Multiple threads may run this method at the same time.
def handle_client(c)
 while true
 input = c.gets.chomp # Read a line of input from the client
 p "Received <#{input}>"
 break if !input # Exit if no more input
 break if input=="quit" # or if the client asks to.
 yield c, input
 c.flush # Force our output out
 end
 c.close # Close the client socket
end

port = ARGV.shift || 2000
server = TCPServer.open(port)
p "listening in port #{port}"

while true # Servers loop forever
 client = server.accept # Wait for a client to connect
 p "new client: #{client}"
 Thread.start(client) do |c|
 handle_client(c) do |c, input| # And handle the client on that thread
 c.puts(input.reverse)
 end
 end
end
```

```
end
end
```

1. To write Internet servers, we use the `TCPServer` class
2. In essence, a `TCPServer` object is a factory for `TCPSocket` objects
3. Call `TCPServer.open` to specify a port for your service and create a `TCPServer` object
4. Next, call the `accept` method of the returned `TCPServer` object
5. This method waits until a client connects to the port you specified, and then returns a `TCPSocket` object that represents the connection to that client
6. The call `Thread.start(client)` is basically the same as a call to `Thread.new`.
7. Inside the thread we call the `handle_client` method, passing the socket as argument
8. The block that follows the invocation of `handle_client` specifies the behavior of the server.
9. Un `TCPSocket` es un objeto IO:

```
>> TCPSocket.ancestors
=> [TCPSocket, IPSocket, BasicSocket, IO,
 File::Constants, Enumerable, Object, Kernel, BasicObject]
```

## El Cliente

To obtain a `TCPSocket` instance use the `TCPSocket.open` class method, or with its synonym `TCPSocket.new`.

Pass the name of the host to connect to as the first argument and the port as the second argument. The port should be an integer between 1 and 65535, specified as a `Fixnum` or `String` object. Different internet protocols use different ports. Web servers use port 80 by default, for example.

You may also pass the name of an Internet service, such as `http`, as a string, in place of a port number, but this is not well documented and may be system dependent.

```
[~/ruby/threads(master)]$ cat simple_client.rb
require 'socket' # Sockets are in standard library

host, port = 'localhost', '2000'

s = TCPSocket.open(host, port) # Open a socket to host and port
words = ARGV
words = [
 "amore, roma.",
 "a man, a plan, a canal: panama.",
 "no 'x' in 'nixon.'",
 "dábale arroz a la zorra el abad"
] unless words.length > 0
words.each do |x|
 s.puts x
 line = s.gets # Read lines from the socket
 break if line.nil?
 puts line.chop # And print with platform line terminator
end
s.close # Close the socket when done
```

## ejecución del Servidor

```
[~/ruby/threaded(master)]$ ruby simple_server.rb
"listening in port 2000"
"new client: #<TCPSocket:0x007fd8fb8aee50>"
```

"Received <amore, roma.>"  
"Received <a man, a plan, a canal: panama.>"  
"Received <no 'x' in 'nixon.'>"  
"Received <d\xC3\xA1bale arroz a la zorra el abad>"

## Ejecución del Cliente

```
[~/ruby/threaded(master)]$ ruby simple_client.rb
.amor ,eroma
.amanap :lanac a ,nalp a ,nam a
'.noxin' ni 'x' on
```

## 25.5. Colas

### Las Clases Queuey SizedQueue

1. The standard `Thread` library defines the `Queue` and `SizedQueue` data structures specifically for concurrent programming.
2. They implement thread-safe FIFO queues and are intended for a producer/consumer model of programming.
3. Under this model, one thread produces values of some sort and places them on a queue with the `enq` (enqueue) method or its synonym `push`.
4. Another thread “consumes” these values, removing them from the queue with the `deq` (dequeue) method as needed. (The `pop` and `shift` methods are synonyms for `deq`.)
5. The key features of `Queue` that make it suitable for concurrent programming is that the `deq` method blocks if the queue is empty and waits until the producer thread adds a value to the queue.
6. The `Queue` and `SizedQueue` classes implement the same basic API, but the `SizedQueue` variant has a maximum size.
7. If the queue is already at its maximum size, then the method for adding a value to the queue will block until the consumer thread removes a value from the queue.

### Example: A Threaded Map-Reduce

1. We now define a method that combines a concurrent `map` with a concurrent `inject`.
2. It creates a thread for each element of the enumerable collection and uses that thread to apply a mapping `Proc`.
3. The value returned by that `Proc` is enqueued on a `Queue` object.
4. One final thread acts as a consumer; it removes values from the queue and passes them to the injection `Proc` as they become available.

Ejemplo de uso:

```

if $0 == __FILE__
 a = if ARGV.empty?
 [-2,-1,0,1,2]
 else
 ARGV.map &:to_f
 end
 mapper = lambda { |x| x*x } # Compute squares
 injector = lambda { |total,x| total+x } # Compute sum
 result = a.conject(0, mapper, injector) # => 10
 puts result
end

```

Código:

```

[~/ruby/threaded(master)]$ cat conject.rb
require 'thread'

module Enumerable
 # Concurrent inject: expects an initial value and two Procs
 def conject(initial, mapper, injector)
 # Use a Queue to pass values from mapping threads to injector thread
 q = Queue.new
 count = 0 # How many items?
 each do |item| # For each item
 Thread.new do # Create a new thread
 q.enq(mapper[item]) # Map and enqueue mapped value
 end
 count += 1 # Count items
 end

 t = Thread.new do # Create injector thread
 x = initial # Start with specified initial value
 while(count > 0) # Loop once for each item
 x = injector[x,q.deq] # Dequeue value and inject
 count -= 1 # Count down
 end
 x # Thread value is injected value
 end

 t.value # Wait for injector thread and return its value
 end
end

```

## 25.6. El Problema de los Filósofos Comensales / The Dining Philosophers Problem

The Dining Philosophers problem was formulated by Edsger Djisktra in 1965 to illustrate the kind of issues we can find when multiple processes compete to gain access to exclusive resources.

1. A gentle introduction to actor-based concurrency.
2. Source code from this article. All of the code from this article is in Practicing Ruby's example repository, but the links below highlight the main points of interest:
  - A solution that leads to deadlocks

- A coordinated mutex-based solution  
Chopsticks class definition
  - Table class definition
3. Scaling Ruby with Actors, or How I Learned to Stop Worrying and Love Threads by Mike Perham  
Rubyconf 2011

# Capítulo 26

## Juegos con Gosu

### 26.1. Enlaces

1. gosu
2. gosu en github
3. ejemplos usando gosu
4. RubyforKids
5. RubyforKids screencasts
6. RubyforKids en Español
7. Programming with kids Español
8. Juego del Tetris usando Gosu
9. Variante del Tutorial del star fighter con disparos, estrellas cayendo y música

### 26.2. El Juego de la Vida

1. A Ruby implementation of Conway's Game of Life in Gosu

### 26.3. Starfighter

Para entender este código véase el tutorial en Gosu Ruby Tutorial. Las imágenes pueden encontrarse en el directorio examples/media de Gosu.

```
require 'rubygems'
require 'gosu'

module ZOrder
 Background, Stars, Player, UI = *0..3
end

class Player
 attr_reader :score

 def initialize(window)
 @image = Gosu::Image.new(window, "media/Starfighter.bmp", false)
```

```

@beep = Gosu::Sample.new(window, "media/Beep.wav")
@x = @y = @vel_x = @vel_y = @angle = 0.0
@score = 0
end

def warp(x, y)
 @x, @y = x, y
end

def turn_left
 @angle -= 4.5
end

def turn_right
 @angle += 4.5
end

def accelerate
 @vel_x += Gosu::offset_x(@angle, 0.5)
 @vel_y += Gosu::offset_y(@angle, 0.5)
end

def move
 @x += @vel_x
 @y += @vel_y
 @x %= 640
 @y %= 480

 @vel_x *= 0.95
 @vel_y *= 0.95
end

def draw
 @image.draw_rot(@x, @y, ZOrder::Player, @angle)
end

def collect_stars(stars)
 stars.reject! do |star|
 if Gosu::distance(@x, @y, star.x, star.y) < 35 then
 @score += 10
 @beep.play
 true
 else
 false
 end
 end
end
end

class Star
 attr_reader :x, :y

 def initialize(animation)

```

```

@animation = animation
@color = Gosu::Color.new(0xff000000)
@color.red = rand(256 - 40) + 40
@color.green = rand(256 - 40) + 40
@color.blue = rand(256 - 40) + 40
@x = rand * 640
@y = rand * 480
end

def draw
 img = @animation[Gosu::milliseconds / 100 % @animation.size]
 img.draw(@x - img.width / 2.0, @y - img.height / 2.0,
 ZOrder::Stars, 1, 1, @color, :add)
end
end

class GameWindow < Gosu::Window
 def initialize
 super(640, 480, false)
 self.caption = "Gosu Tutorial Game"

 @background_image = Gosu::Image.new(self, "media/Space.png", true)

 @player = Player.new(self)
 @player.warp(320, 240)

 @star_anim = Gosu::Image::load_tiles(self, "media/Star.png", 25, 25, false)
 @stars = Array.new

 @font = Gosu::Font.new(self, Gosu::default_font_name, 20)
 end

 def update
 if button_down? Gosu::KbLeft or button_down? Gosu::GpLeft then
 @player.turn_left
 end
 if button_down? Gosu::KbRight or button_down? Gosu::GpRight then
 @player.turn_right
 end
 if button_down? Gosu::KbUp or button_down? Gosu::GpButton0 then
 @player.accelerate
 end
 @player.move
 @player.collect_stars(@stars)

 if rand(100) < 4 and @stars.size < 25 then
 @stars.push(Star.new(@star_anim))
 end
 end

 def draw
 @background_image.draw(0, 0, ZOrder::Background)
 @player.draw
 end
end

```

```

@stars.each { |star| star.draw }
@font.draw("Score: #{@player.score}", 10, 10, ZOrder::UI, 1.0, 1.0, 0xffffffff)
end

def button_down(id)
 if id == Gosu::KbEscape then
 close
 end
end
end

window = GameWindow.new
window.show

```

## 26.4. Random bouncy particles using ruby Gosu games library

Random bouncy particles using ruby Gosu games library

### Donde

```
[~/src/ruby/rubytesting/gosu]$ pwd -P
/Users/casiano/local/src/ruby/rubytesting/gosu
```

### Código

```
[~/src/ruby/rubytesting/gosu]$ cat bouncing.rb
require 'gosu'
$width, $height = 200, 200
$number_of_particles = 200

class Quad
 def initialize
 @pos = {x:rand($width), y:rand($width)}
 @vel = { x:(rand(5)+1)*[1,-1].sample, y:(rand(5)+1)*[1,-1].sample }
 @size = rand(4)*[1,-1].sample
 @color = [Gosu::Color::GRAY, Gosu::Color::WHITE,
 Gosu::Color::AQUA, Gosu::Color::RED,
 Gosu::Color::GREEN, Gosu::Color::BLUE,
 Gosu::Color::YELLOW, Gosu::Color::FUCHSIA,
 Gosu::Color::CYAN].sample
 end
 def update
 @vel[:x] = @vel[:x] * -1 if @pos[:x] <= 0 or @pos[:x] >= $width
 @vel[:y] = @vel[:y] * -1 if @pos[:y] <= 0 or @pos[:y] >= $height
 @pos[:x] += @vel[:x]
 @pos[:y] += @vel[:y]
 end
 def draw win
 win.draw_quad @pos[:x]-@size, @pos[:y]-@size, @color,
 @pos[:x]+@size, @pos[:y]-@size, @color,
 @pos[:x]+@size, @pos[:y]+@size, @color,
 @pos[:x]-@size, @pos[:y]+@size, @color
 end
end
```

```
end

class GameWindow < Gosu::Window
 def initialize
 super $width, $height, false
 self.caption = "Quads"
 @quads = []
 $number_of_particles.times { @quads << Quad.new }
 end
 def update
 @quads.each { |q| q.update }
 end
 def draw
 color = Gosu::Color::WHITE
 draw_quad 0, 0, color, $width, 0, color, $width, $height, color, 0, $height, color
 @quads.each { |q| q.draw self }
 end
end

window = GameWindow.new
window.show
```

## Capítulo 27

# Shoes

- The green\_shoes wiki

## Capítulo 28

# herramientas para el Control de la Calidad

### 28.1. Lints: rubocop y reek

- rubocop
- pelusa: Pelusa is a static analysis tool and framework to inspect your code style and notify you about possible red flags or missing best practices.
- reek

#### Rubocop

RuboCop is a Ruby static code analyzer. Out of the box it will enforce many of the guidelines outlined in the community Ruby Style Guide.

```
[~/rubytesting/rubocop]$ cat test.rb
def badName
 if something
 test
 end
end

[~/rubytesting/rubocop]$ rubocop --version
0.26.1
[~/rubytesting/rubocop]$ date
martes, 7 de octubre de 2014, 10:22:07 WEST
```

```
[~/rubytesting/rubocop]$ rubocop test.rb
Inspecting 1 file
W
```

Offenses:

```
test.rb:1:5: C: Use snake_case for methods.
def badName
^~~~~~
test.rb:2:3: C: Use a guard clause instead of wrapping the code inside a conditional expression
 if something
 ^^
test.rb:2:3: C: Favor modifier if usage when having a single-line body. Another good alternative
 if something
```

```

^
test.rb:4:5: W: end at 4, 4 is not aligned with if at 2, 2
 end
 ^

```

1 file inspected, 4 offenses detected

```
[~/rubytesting/rubocop]$ cat test2.rb
def good_name
 test if something
end
```

```
[~/rubytesting/rubocop]$ rubocop test2.rb
Inspecting 1 file
```

.

1 file inspected, no offenses detected

### Code Smells: reek

reek is a tool that examines Ruby classes, modules and methods and reports any Code Smells it

```
)[~/rubytesting/reek]$ ls
Gemfile Gemfile.lock Rakefile dirty.rb
[~/rubytesting/reek]$ cat dirty.rb
class Dirty
 # This method smells of :reek:NestedIterators but ignores them
 def awful(x, y, offset = 0, log = false)
 puts @screen.title
 @screen = widgets.map { |w| w.each { |key| key += 3}}
 puts @screen.contents
 end
end
[~/rubytesting/reek]$ reek dirty.rb
dirty.rb -- 8 warnings:
[1]:Dirty has no descriptive comment (IrresponsibleModule)
[3]:Dirty#awful has 4 parameters (LongParameterList)
[3]:Dirty#awful has boolean parameter 'log' (BooleanParameter)
[5]:Dirty#awful has the variable name 'w' (UncommunicativeVariableName)
[3]:Dirty#awful has unused parameter 'log' (UnusedParameters)
[3]:Dirty#awful has unused parameter 'offset' (UnusedParameters)
[3]:Dirty#awful has unused parameter 'x' (UnusedParameters)
[3]:Dirty#awful has unused parameter 'y' (UnusedParameters)
```

## 28.2. Véase

- roodi
- Saikuro: Ruby cyclomatic complexity analyzer.
- codeclimate
- *Coveralls: Análisis de Cubrimiento 20.7*
- The Ruby ToolBox: Code Metrics

## Capítulo 29

# La Plataforma Ruby

### 29.1. Expresiones Regulares

- Regular Expressions Are Your Friend by Aaron Kalin, Hashrocket

# Capítulo 30

## El Entorno Ruby

**30.1. Invocando al intérprete Ruby**

**30.2. El Entorno al Nivel mas Alto**

**30.3. Atajos para la Extracción e Informes (Tipo Perl)**

**30.4. Llamando al Sistema Operativo**

**30.5. Seguridad**

**30.6. El Compilador de Ruby**

1. Complex Ruby Concepts Simplified por Matt Aimonetty

# **Parte III**

# **SINATRA**

# Capítulo 31

# Rack, un Webserver Ruby Modular

## 31.1. Introducción

**Que es Rack** rack provides an minimal interface between web servers supporting Ruby and Ruby frameworks.

Ruby on Rails, Ramaze, Sinatra and other Ruby frameworks use it by default to talk to web servers, including Mongrel, Thin or Apache via Passenger.

Lo que hace Rack es que unifica la API de los diferentes web servers envolviendo las peticiones y respuestas HTTP en la forma mas simple posible.

1. Rack includes *handlers* that connect Rack to all these web application servers (WEBrick, Mongrel etc.).
2. Rack includes *adapters* that connect Rack to various web frameworks (Sinatra, Rails etc.).
3. Between the server and the framework, Rack can be customized to your applications needs using *middleware*.

The fundamental idea behind *Rack middleware* is – come between the calling client and the server, process the HTTP request before sending it to the server, and processing the HTTP response before returning it to the client.

**Que es una Aplicación Rack** Una aplicación Rack es un objeto que

1. Debe responder al método `call`.
2. El método `call` será llamado por el servidor y se le pasa como argumento `env` que es un hash que contiene información sobre el entorno CGI.
3. El método `call` debe retornar un array con tres elementos:
  - a) `status`: un entero
  - b) `headers`: un hash
  - c) `body`: un objeto que responde al método `each` y que para cada llamada de `each` retorna una `String`.

## Un Ejemplo Sencillo

*Rack* uses a configuration file with extension `.ru`, that instructs Rack::Builder what middleware should it use and in which order. Let's create one:

```
[~/sinatra/rackup/simple(master)]$ cat myapp.rb
my_app.rb
#
```

```

class MyApp
 def call env
 [200, {"Content-Type" => "text/html"}, ["Hello Rack Participants"]]
 end
end

```

Esta es la aplicación Rack mas simple posible.

```
[~/sinatra/rackup/simple(master)]$ cat config.ru
require './myapp'
run MyApp.new
```

To start your newly created app, you need to use `rackup` command:

```
$ rackup config.ru
```

The application will be available by default on port 9292, so you have to visit <http://localhost:9292> to see it.

Donde se encuentra este ejemplo:

- [~/sinatra/rack/rack-simple(env)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/rack/rackup/simple
- [~/sinatra/rack/rack-simple(env)]\$ git remote -v  
origin git@github.com:crguezl/a-quick-introduction-to-rack.git (fetch)  
origin git@github.com:crguezl/a-quick-introduction-to-rack.git (push)
- Repo: crguezl/a-quick-introduction-to-rack

## Un Ejemplo con la Consola Interactiva de Ruby

Arranquemos la consola interactiva de Ruby. Cargamos `rack`:

```
1] pry(main)> require 'rack'
=> true
```

Comprobemos que handlers tenemos instalados:

```
[2] pry(main)> Rack::Handler::constants
=> [:CGI,
 :FastCGI,
 :Mongrel,
 :EventedMongrel,
 :SwiftipliedMongrel,
 :WEBrick,
 :LSWS,
 :SCGI,
 :Thin]
```

Todos los handlers Rack tienen un método `run` por lo que podemos llamar el método `run` en cualquiera de esos handlers instalados.

Un objeto que tiene un método `call` es cualquier objeto `Proc` y por tanto podemos usar una lambda que se atenga al protocolo de rack para nuestro ejemplo:

```
[3] pry(main)> Rack::Handler::WEBrick.run lambda
 { |env| [200,
 {"Content-Type" => "text/plain"},
 ["Hello. The time is #{Time.now}"]] }
```

```
[2013-09-11 17:59:07] INFO WEBrick 1.3.1
[2013-09-11 17:59:07] INFO ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-11 17:59:07] INFO WEBrick::HTTPServer#start: pid=25123 port=8080
localhost - - [11/Sep/2013:17:59:26 WEST] "GET / HTTP/1.1" 200 44
- -> /
localhost - - [11/Sep/2013:17:59:26 WEST] "GET /favicon.ico HTTP/1.1" 200 44
- -> /favicon.ico
```

El primer argumento de `run` es nuestra aplicación Rack

```
lambda { |env| [200, {"Content-Type" => "text/plain"}, ["Hello. The time is #{Time.now}"]]
```

y el segundo es el conjunto de opciones para nuestro programa.

ahora podemos visitar la aplicación con nuestro navegador en `http://localhost:8080`

## 31.2. Analizando env con pry-debugger

### 31.2.1. Introducción

Tenemos esta sencilla aplicación:

```
~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello.rb
require 'rack'
require 'pry-debugger'

class HelloWorld
 def call env
 binding.pry
 [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
 end
end
```

Arrancamos un servidor:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ pry
[1] pry(main)> require './hello'
=> true
[6] pry(main)> Rack::Handler::WEBrick.run HelloWorld.new
[2013-09-23 12:36:21] INFO WEBrick 1.3.1
[2013-09-23 12:36:21] INFO ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 12:36:21] INFO WEBrick::HTTPServer#start: pid=9458 port=8080
```

En otra ventana arrancamos un cliente:

```
[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ curl -v localhost:8080
* About to connect() to localhost port 8080 (#0)
* Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
```

En la ventana del servidor ahora aparece:

```
From: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello.rb @ line 6 HelloWorld#c
```

```
5: def call env
=> 6: binding.pry
 [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
8: end
```

Ahora podemos inspeccionar las variables:

```
[1] pry(#<HelloWorld>) > env
=> {"GATEWAY_INTERFACE"=>"CGI/1.1",
 "PATH_INFO"=>"/",
 "QUERY_STRING"=>"",
 "REMOTE_ADDR"=>"::1",
 "REMOTE_HOST"=>"localhost",
 "REQUEST_METHOD"=>"GET",
 "REQUEST_URI"=>"http://localhost:8080/",
 "SCRIPT_NAME"=>"",
 "SERVER_NAME"=>"localhost",
 "SERVER_PORT"=>"8080",
 "SERVER_PROTOCOL"=>"HTTP/1.1",
 "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)",
 "HTTP_USER_AGENT"=>
 "curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5",
 "HTTP_HOST"=>"localhost:8080",
 "HTTP_ACCEPT"=>"*/*",
 "rack.version"=>[1, 2],
 "rack.input"=>#<StringIO:0x007fbba40263b0>,
 "rack.errors"=>#<IO:<STDERR>>,
 "rack.multithread"=>true,
 "rack.multiprocess"=>false,
 "rack.run_once"=>false,
 "rack.url_scheme"=>"http",
 "HTTP_VERSION"=>"HTTP/1.1",
 "REQUEST_PATH"=>"/"}
[2] pry(#<HelloWorld>)
```

Hay tres categorías de variables en *env*:

1. Variables CGI
2. Variables específicas de Rack (empiezan por `rack.`)
3. Un tercer tipo de variables son las de la aplicación y/o el servidor. En este ejemplo no aparecen

Véase la especificación Rack.

Le indicamos al servidor que continue:

```
[2] pry(#<HelloWorld>) > co<TABULADOR>
cohen-poem continue
[2] pry(#<HelloWorld>) > continue
localhost - - [23/Sep/2013:12:36:48 WEST] "GET / HTTP/1.1" 200 11
- -> /
```

después de entregar la respuesta el servidor cierra la conexión HTTP. Esto es así porque HTTP es un protocolo sin estado, esto es, no se mantiene información de la conexión entre transacciones. En la ventana del cliente obtenemos la siguiente salida:

```
[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ curl -v localhost:8080
* About to connect() to localhost port 8080 (#0)
* Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Server: WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)
< Date: Mon, 23 Sep 2013 11:45:00 GMT
< Content-Length: 11
< Connection: Keep-Alive
<
* Connection #0 to host localhost left intact
* Closing connection #0
Hello world
```

### 31.2.2. REQUEST\_METHOD, QUERY\_STRING y PATH\_INFO

```
[~/local/src/ruby/sinatra/rack/rack-env]$ cat app.rb
require 'rack'
require 'thin'

cgi_inspector = lambda do |env|
 [200, #status
 { 'Content-Type' => 'text/html' }, #headers
 ["<h1>
 Your request:

 http method is: #{env['REQUEST_METHOD']}
 path is: #{env['PATH_INFO']}
 Query string is: #{env['QUERY_STRING']}

 </h1>
 "
]
]
end
```

Rack::Handler::Thin.run cgi\_inspector, :Port => 3000

Visite la página localhost:3000/camino?var=4.

Esta es la salida:

```
[~/local/src/ruby/sinatra/rack/rack-env]$ curl -v localhost:3000/camino?var=4
* About to connect() to localhost port 3000 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 3000 (#0)
> GET /camino?var=4 HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
```

```

> Host: localhost:3000
> Accept: /*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
<h1>
 Your request:

 http method is: GET
 path is: /camino
 Query string is: var=4

</h1>
* Closing connection #0

```

### 31.3. Detectando el Proceso que está Usando un Puerto

Si intentamos ejecutar una segunda instancia del servidor mientras otra instancia esta ejecutandose obtenemos un error que indica que el puerto está en uso:

```

[~/sinatra/sinatra-simple(master)]$ rackup
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on 0.0.0.0:9292, CTRL+C to stop
/Users/casiano/.rvm/gems/ruby-2.0.0-p247/gems/eventmachine-1.0.3/lib/eventmachine.rb:526:
 in 'start_tcp_server': no acceptor
 (port is in use or requires root privileges) (RuntimeError)
 from /Users/casiano/.rvm/gems/ruby-2.0.0-p247/gems/eventmachine-1.0.3/lib/eventmachine.rb:526:
 in 'start_server'
...

```

Si sabemos en que puerto esta corriendo - como es el caso - podemos hacer algo así para saber el PID del proceso que lo ocupa:

```

[~/sinatra/sinatra-simple(master)]$ lsof -i :9292
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
ruby 52870 casiano 9u IPv4 0x9f3ffc595152af29 0t0 TCP *:armtechdaemon (LISTEN)

```

Si no lo sabemos podemos hacer:

```

[~/sinatra/sinatra-simple(master)]$ ps -fa | egrep ruby
 501 52870 565 0 11:16AM ttys003 0:00.61 ruby /Users/casiano/.rvm/gems/ruby-2.0.0-p247
 501 53230 52950 0 11:35AM ttys006 0:00.00 egrep ruby

```

Si tenemos privilegios suficientes podemos ahora eliminar el proceso:

```
[~/sinatra/sinatra-simple(master)]$ kill -9 52870
```

```

[~/sinatra/sinatra-simple(master)]$ rackup
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on 0.0.0.0:9292, CTRL+C to stop
Killed: 9

```

El comando

```
$ lsof -i | egrep -i 'tcp.*(\d+.)+'
```

Nos da una lista bastante completa de como están nuestras conexiones.

1. **-i [i]** selects the listing of files any of whose Internet address matches the address specified in i. If no address is specified, this option selects the listing of all Internet and x.25 (HP-UX) network files.

### 31.4. Usando PATH\_INFO y erubis para construir una aplicación (Noah Gibbs)

**config.ru**

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ cat config.ru
require "erubis"

use Rack::ContentType

def output(text, options = {})
 [options[:status] || 200,
 {}, [text].flatten]
end

def from_erb(file, vars = {})
 eruby = Erubis::Eruby.new File.read(file)
 output eruby.result vars
end

run proc { |env|
 path = env['PATH_INFO']
 if path =~ %r{/foo}
 from_erb "template.html.erb"
 else
 output "Not found!", :status => 400
 end
}
```

**Template erb**

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ cat template.html.erb
<p> A template! </p>
<% 10.times do -%> <p> Pretty cool! </p> <% end -%>
```

**Arrancando el Servidor**

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop}
```

## Ejecutando un cliente

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ curl -v http://localhost:9292/foochazam
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /foochazam HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.8
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
<p> A template! </p>
<p> Pretty cool! </p> <p> Pretty cool! </p> <p> Pretty cool! </p>
<p> Pretty cool! </p> <p> Pretty cool! </p> <p> Pretty cool! </p>
<p> Pretty cool! </p> <p> Pretty cool! </p> <p> Pretty cool! </p>
<p> Pretty cool! </p>
* Closing connection #0
```

## Logs del servidor

```
[~/local/src/ruby/sinatra/rack/hangout-framework(master)]$ rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
127.0.0.1 - - [20/Oct/2013 12:22:37] "GET /foochazam HTTP/1.1" 200 - 0.0014
```

## Véase

1. erubis
2. Noah Gibbs Demo Rack framework for March 6th, 2013 Ruby Hangout.
3. Ruby Hangout 3-13 Noah Gibbs

## 31.5. HTTP

### 31.5.1. Introducción

1. HTTP es un protocolo sin estado: que no guarda ninguna información sobre conexiones anteriores.
2. El desarrollo de aplicaciones web necesita frecuentemente mantener estado.
3. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente.
4. Esto le permite a las aplicaciones web introducir la noción de *sesión*, y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

5. Una transacción HTTP está formada por un *encabezado* seguido, opcionalmente, por una línea en blanco y algún dato.
6. El encabezado especificará cosas como la acción requerida del servidor, o el tipo de dato retornado, o el código de estado.
7. El uso de campos de encabezados enviados en las transacciones HTTP le da flexibilidad al protocolo. Estos campos permiten que se envíe información descriptiva en la transacción, permitiendo así la autenticación, cifrado e identificación de usuario.
8. Un encabezado es un bloque de datos que precede a la información propiamente dicha, por lo que a veces se hace referencia a él como metadato, porque tiene datos sobre los datos.
9. Si se reciben líneas de encabezado del cliente, el servidor las coloca en las variables de entorno de CGI con el prefijo `HTTP_` seguido del nombre del encabezado. Cualquier carácter guion (-) del nombre del encabezado se convierte a caracteres `"_"`.

Ejemplos de estos encabezados del cliente son `HTTP_ACCEPT` y `HTTP_USER_AGENT`.

- a) `HTTP_ACCEPT`. Los tipos MIME que el cliente aceptará, dados los encabezados HTTP. Los elementos de esta lista deben estar separados por comas
- b) `HTTP_USER_AGENT`. El navegador que utiliza el cliente para realizar la petición. El formato general para esta variable es: software/versión biblioteca/versión.

El servidor envía al cliente:

- a) Un *código de estado* que indica si la petición fue correcta o no. Los códigos de error típicos indican que el archivo solicitado no se encontró, que la petición no se realizó de forma correcta o que se requiere autenticación para acceder al archivo.
- b) La información propiamente dicha. HTTP permite enviar documentos de todo tipo y formato, como gráficos, audio y video.
- c) Información sobre el objeto que se retorna.

### 31.5.2. Sesiones HTTP

1. Una sesión HTTP es una secuencia de transacciones de red de peticiones y respuestas
2. Un cliente HTTP inicia una petición estableciendo una conexión TCP con un puerto particular de un servidor (normalmente el puerto 80)
3. Un servidor que esté escuchando en ese puerto espera por un mensaje de petición de un cliente.
4. El servidor retorna la *línea de estatus*, por ejemplo `HTTP/1.1 200 OK`, y su propio mensaje. El cuerpo de este mensaje suele ser el recurso solicitado, aunque puede que se trate de un mensaje de error u otro tipo de información.

Veamos un ejemplo. Usemos este servidor:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello1.rb
require 'rack'

class HelloWorld
 def call env
 [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
 end
end
```

```
Rack::Handler::WEBrick::run HelloWorld.new
```

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ ruby hello1.rb
[2013-09-23 15:16:58] INFO WEBrick 1.3.1
[2013-09-23 15:16:58] INFO ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 15:16:58] INFO WEBrick::HTTPServer#start: pid=12113 port=8080
```

Arrancamos un cliente con telnet con la salida redirigida:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ telnet localhost 8080 > salida
```

Escribimos esto en la entrada estandard:

```
GET /index.html HTTP/1.1
Host: localhost
Connection: close
```

con una linea en blanco al final. Este texto es enviado al servidor.

El cliente deja su salida en el fichero `salida`:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat salida
Trying ::1...
Connected to localhost.
Escape character is '^]'.
HTTP/1.1 200 OK
Content-Type: text/plain
Server: WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)
Date: Mon, 23 Sep 2013 14:33:16 GMT
Content-Length: 11
Connection: close
```

Hello world

El cliente escribe en la salida estandard:

```
Connection closed by foreign host.
```

### 31.5.3. Métodos de Petición

#### 1. *GET*

Solicita una representación de un recurso especificado. Las peticiones que usen *GET* deberían limitarse a obtener los datos y no tener ningún otro efecto.

#### 2. *HEAD*

Pregunta por la misma respuesta que una petición *GET* pero sin el cuerpo de la respuesta

#### 3. *POST*

Requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, as examples,

- a) an annotation for existing resources;
- b) a message for a bulletin board, newsgroup, mailing list, or comment thread;
- c) a block of data that is the result of submitting a web form to a data-handling process;
- d) or an item to add to a database.

#### 4. *PUT*

Requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.

#### 5. *DELETE*

Deletes the specified resource.

#### 6. *TRACE*

Echoes back the received request so that a client can see what (if any) changes or additions have been made by intermediate servers.

#### 7. *OPTIONS*

Returns the HTTP methods that the server supports for the specified URL. This can be used to check the functionality of a web server by requesting \* instead of a specific resource.

#### 8. *CONNECT*

Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.

#### 9. *PATCH*

Is used to apply partial modifications to a resource. HTTP servers are required to implement at least the GET and HEAD methods and, whenever possible, also the OPTIONS method

### 31.5.4. Véase

1. ArrrrCamp #6 - Konstantin Haase - We don't know HTTP
2. Resources, For Real This Time (with Webmachine) Sean Cribbs Ruby Conference 2011

## 31.6. Rack::Request y Depuración con pry-debugger

### 31.6.1. Conexión sin Parámetros

Partimos del mismo código fuente que en la sección anterior:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello.rb
require 'rack'
require 'pry-debugger'

class HelloWorld
 def call env
 binding.pry
 [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
 end
end
```

Arranquemos un servidor dentro de pry:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ pry
[1] pry(main)> require './hello'
=> true
[2] pry(main)> Rack::Handler::WEBrick::run HelloWorld.new
[2013-09-23 13:10:42] INFO WEBrick 1.3.1
[2013-09-23 13:10:42] INFO ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 13:10:42] INFO WEBrick::HTTPServer#start: pid=10395 port=8080
```

Si visitamos la página:

```
$ curl -v localhost:8080/jkdfkdjg
```

Esto hace que se alcance el break:

```
From: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello.rb @ line 6 HelloWorld#c
```

```
 5: def call env
=> 6: binding.pry
 7: [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
 8: end
```

Ahora creamos un objeto Rack::Request:

```
[3] pry(#<HelloWorld>) > req = Rack::Request.new(env)
=> #<Rack::Request:0x007fbba4ff3298
@env=
{"GATEWAY_INTERFACE"=>"CGI/1.1",
 "PATH_INFO"=>"/jkdfkdjg",
 "QUERY_STRING"=>"",
 "REMOTE_ADDR"=>"::1",
 "REMOTE_HOST"=>"localhost",
 "REQUEST_METHOD"=>"GET",
 "REQUEST_URI"=>"http://localhost:8080/jkdfkdjg",
 "SCRIPT_NAME"=>"",
 "SERVER_NAME"=>"localhost",
 "SERVER_PORT"=>"8080",
 "SERVER_PROTOCOL"=>"HTTP/1.1",
 "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)",
 "HTTP_USER_AGENT"=>
 "curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5",
 "HTTP_HOST"=>"localhost:8080",
 "HTTP_ACCEPT"=>"*/*",
 "rack.version"=>[1, 2],
 "rack.input"=>#<StringIO:0x007fbba4e74980>,
 "rack.errors"=>#<IO:<STDERR>>,
 "rack.multithread"=>true,
 "rack.multiprocess"=>false,
 "rack.run_once"=>false,
 "rack.url_scheme"=>"http",
 "HTTP_VERSION"=>"HTTP/1.1",
 "REQUEST_PATH"=>"/jkdfkdjg"}>
```

Este objeto Rack::Request tiene métodos para informarnos del Rack::Request:

```
[4] pry(#<HelloWorld>) > req.get?
=> true
[5] pry(#<HelloWorld>) > req.post?
=> false
[7] pry(#<HelloWorld>) > req.port
=> 8080
[12] pry(#<HelloWorld>) > req.host()
=> "localhost"
[13] pry(#<HelloWorld>) > req.host_with_port()
=> "localhost:8080"
```

```
[15] pry(#<HelloWorld>) > req.path()
=> "/jkdfkdjg"
[18] pry(#<HelloWorld>) > req.url()
=> "http://localhost:8080/jkdfkdjg"
[19] pry(#<HelloWorld>) > req.user_agent
=> "curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5"
```

### 31.6.2. Conexión con Parámetros

Partimos del mismo código fuente que en la sección anterior:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello.rb
require 'rack'
require 'pry-debugger'
```

```
class HelloWorld
 def call env
 binding.pry
 [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
 end
end
```

Arrancamos el servidor:

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ pry
[1] pry(main)> require './hello'
=> true
[2] pry(main)> Rack::Handler::WEBrick::run HelloWorld.new
[2013-09-23 13:10:42] INFO WEBrick 1.3.1
[2013-09-23 13:10:42] INFO ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 13:10:42] INFO WEBrick::HTTPServer#start: pid=10395 port=8080
```

En el cliente tendríamos:

```
$ curl -v 'localhost:8080?a=1&b=2&c=3'
```

comienza produciendo esta salida:

```
* About to connect() to localhost port 8080 (#0)
* Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> GET /?a=1&b=2&c=3 HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
```

En la ventana del servidor se produce el break:

```
From: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello.rb @ line 6 HelloWorld#c
```

```
5: def call env
=> 6: binding.pry
7: [200, {"Content-Type" => "text/plain"}, ["Hello world"]]
8: end
```

## Rack::Request.new

Creamos un objeto Rack::Request:

```
[1] pry(#<HelloWorld>) > req = Rack::Request.new env
=> #<Rack::Request:0x007fafd27946c0
@env=
{"GATEWAY_INTERFACE"=>"CGI/1.1",
 "PATH_INFO"=>"/",
 "QUERY_STRING"=>"a=1&b=2&c=3",
 "REMOTE_ADDR"=>"::1",
 "REMOTE_HOST"=>"localhost",
 "REQUEST_METHOD"=>"GET",
 "REQUEST_URI"=>"http://localhost:8080/?a=1&b=2&c=3",
 "SCRIPT_NAME"=>"",
 "SERVER_NAME"=>"localhost",
 "SERVER_PORT"=>"8080",
 "SERVER_PROTOCOL"=>"HTTP/1.1",
 "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)",
 "HTTP_USER_AGENT"=>
 "curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.5",
 "HTTP_HOST"=>"localhost:8080",
 "HTTP_ACCEPT"=>"*/*",
 "rack.version"=>[1, 2],
 "rack.input"=>#<StringIO:0x007fafd26bbbe0>,
 "rack.errors"=>#<IO:<STDERR>>,
 "rack.multithread"=>true,
 "rack.multiprocess"=>false,
 "rack.run_once"=>false,
 "rack.url_scheme"=>"http",
 "HTTP_VERSION"=>"HTTP/1.1",
 "REQUEST_PATH"=>"/"}>
```

**req.params** Ahora podemos interrogarle:

```
[2] pry(#<HelloWorld>) > req.params
=> {"a"=>"1", "b"=>"2", "c"=>"3"}
```

## Indexación de los objetos Rack::Request

Recordemos que la URL visitada fué: localhost:8080?a=1&b=2&c=3

```
[3] pry(#<HelloWorld>) > req["a"]
=> "1"
[4] pry(#<HelloWorld>) > req["b"]
=> "2"
[5] pry(#<HelloWorld>) > req["c"]
=> "3"
```

## req.path

```
[6] pry(#<HelloWorld>) > req.path
=> "/"
[7] pry(#<HelloWorld>) > req.fullpath
=> "/?a=1&b=2&c=3"
[9] pry(#<HelloWorld>) > req.path_info
=> "/"
[10] pry(#<HelloWorld>) > req.query_string
=> "a=1&b=2&c=3"
```

```
req.url
```

```
[11] pry(#<HelloWorld>) > req.url
=> "http://localhost:8080/?a=1&b=2&c=3"
```

```
req.values
```

```
[12] pry(#<HelloWorld>) > req.values_at("a")
=> ["1"]
[13] pry(#<HelloWorld>) > req.values_at("a", "b")
=> ["1", "2"]
[14] pry(#<HelloWorld>) > req.values_at("a", "b", "c")
=> ["1", "2", "3"]
```

```
[16] pry(#<HelloWorld>) > continue
localhost - - [23/Sep/2013:13:10:49 WEST] "GET /?a=1&b=2&c=3 HTTP/1.1" 200 11
- -> /?a=1&b=2&c=3
```

```
$ curl -v 'localhost:8080?a=1&b=2&c=3'
* About to connect() to localhost port 8080 (#0)
* Trying ::1... connected
* Connected to localhost (::1) port 8080 (#0)
> GET /?a=1&b=2&c=3 HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/plain
< Server: WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)
< Date: Mon, 23 Sep 2013 12:35:37 GMT
< Content-Length: 11
< Connection: Keep-Alive
<
* Connection #0 to host localhost left intact
* Closing connection #0
Hello world
```

## 31.7. Rack::Response

### 31.7.1. Introducción

Rack::Response provides a convenient interface to create a Rack response.

It allows setting of headers and cookies, and provides useful defaults (a `OK` response containing HTML).

You can use `Response#write` to iteratively generate your response, but note that this is buffered by Rack::Response until you call `finish`.

Alternatively, the method `finish` can take a block inside which calls to write are synchronous with the Rack response.

Your application's call should end returning `Response#finish`.

### 31.7.2. Ejemplo Simple

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat body_bytesize.rb
require 'rack'
require 'thin'

app = lambda do |env|
 req = Rack::Request.new env
 res = Rack::Response.new

 body = "----- Header -----\\n"

 if req.path_info == '/hello'
 body << "hi "
 name = req['name']
 body << name if name
 body << "\\n"
 else
 body << "Instead of #{req.url} visit something like "+
 "http://localhost:8080/hello?name=Casiano\\n"
 end
 res['Content-Type'] = 'text/plain'
 res["Content-Length"] = body.bytesize.to_s
 #res["Content-Length"] = Rack::Utils.bytesize(body).to_s
 res.body = [body]
 res.finish
end

Rack::Handler::Thin.run app
```

### 31.7.3. Ejemplo con POST

```
[~/local/src/ruby/sinatra/rack/rack-debugging]$ cat hello_response.rb
encoding: utf-8
require 'rack'
require 'pry-debugger'

class HelloWorld

 def call env
 req = Rack::Request.new(env)
 res = Rack::Response.new
 binding.pry if ARGV[0]
 res['Content-Type'] = 'text/html'
 name = (req["firstname"] && req["firstname"] != '') ? req["firstname"] : 'World'
 res.write <<--EOS"
 <!DOCTYPE HTML>
 <html>
 <title>Rack::Response</title>
 <body>
 <h1>
 Hello #{name}!
 <form action="/" method="post">
 Your name: <input type="text" name="firstname" autofocus>

```

```

 <input type="submit" value="Submit">
 </form>
</h1>
</body>
</html>
EOS
res.finish
end
end

Rack::Server.start(
 :app => HelloWorld.new,
 :Port => 9292,
 :server => 'thin'
)

```

[~/local/src/ruby/sinatra/rack/rack-debugging]\$ ruby hello\_response.rb debug  
>> Thin web server (v1.5.1 codename Straight Razor)  
>> Maximum connections set to 1024  
>> Listening on 0.0.0.0:9292, CTRL+C to stop

Ahora cuando visitamos la página `http://localhost:9292` el navegador queda a la espera del servidor y el servidor alcanza la línea de break.

From: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello\_response.rb @ line 10 He

```

7: def call env
8: req = Rack::Request.new(env)
9: res = Rack::Response.new
=> 10: binding.pry if ARGV[0]
11: res['Content-Type'] = 'text/html'
12: name = (req["firstname"] && req["firstname"] != '') ? req["firstname"] : 'World'
13: res.write <<-"EOS"
14: <!DOCTYPE HTML>
15: <html>
16: <title>Rack::Response</title>
17: <body>
18: <h1>
19: Hello #{name}!
20: <form action="/" method="post">
21: Your name: <input type="text" name="firstname" autofocus>

22: <input type="submit" value="Submit">
23: </form>
24: </h1>
25: </body>
26: </html>
27: EOS
28: res.finish
29: end

```

[1] pry(#<HelloWorld>)>

Consultemos los contenidos de `res`:

```
[1] pry(#<HelloWorld>) > res
=> #<Rack::Response:0x007fe3fb1e6180
@block=nil,
@body=[],
@chunked=false,
@header={},
@length=0,
@status=200,
@writer=
#<Proc:0x007fe3fb1e5f50@/Users/casiano/.rvm/gems/ruby-1.9.3-p392/gems/rack-1.5.2/lib/rack/re
```

Después de un par de `continue` el servidor se queda a la espera:

```
[3] pry(#<HelloWorld>) > continue
...
[1] pry(#<HelloWorld>) > continue
```

Rellenamos la entrada con un nombre (Pedro) y de nuevo el servidor alcanza el punto de ruptura:

```
[2] pry(#<HelloWorld>) > req.params
=> {"firstname"=>"Pedro"}

[7] pry(#<HelloWorld>) > break 28
Breakpoint 1: /Users/casiano/local/src/ruby/sinatra/rack/rack-debugging/hello_response.rb @ li

26: </html>
27: EOS
=> 28: res.finish
29: end

[8] pry(#<HelloWorld>) > continue
Breakpoint 1. First hit.
...
[9] pry(#<HelloWorld>) > res.headers
=> {"Content-Type"=>"text/html", "Content-Length"=>"370"}
[10] pry(#<HelloWorld>) >
```

## 31.8. Cookies y Rack

Cookies may be used to maintain data related to the user during navigation, possibly across multiple visits.

### Introducción

1. A *cookie*, is a small piece of data sent from a website and stored in a user's web browser while the user is browsing that website.
2. Every time the user loads the website, the browser sends the cookie back to the server to notify the website of the user's previous activity
3. Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items in a shopping cart) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited by the user as far back as months or years ago).

4. A user's *session cookie* (also known as an in-memory cookie or transient cookie) for a website exists in temporary memory only while the user is reading and navigating the website.
5. When an expiry date or validity interval is not set at cookie creation time, a session cookie is created. Web browsers normally delete session cookies when the user closes the browser
6. A *persistent cookie* will outlast user sessions. If a persistent cookie has its `Max-Age` set to 1 year, then, during that year, the initial value set in that cookie would be sent back to the server every time the user visited the server. This could be used to record information such as how the user initially came to this website. For this reason, persistent cookies are also called *tracking cookies*
7. A *secure cookie* has the `secure` attribute enabled and is only used via HTTPS, ensuring that the cookie is always encrypted when transmitting from client to server. This makes the cookie less likely to be exposed to cookie theft via eavesdropping.
8. *First-party cookies* are cookies that belong to the same domain that is shown in the browser's address bar (or that belong to the sub domain of the domain in the address bar).
9. *Third-party cookies* are cookies that belong to domains different from the one shown in the address bar.
  - a) Web pages can feature content from third-party domains (such as banner adverts), which opens up the potential for tracking the user's browsing history.
  - b) Privacy setting options in most modern browsers allow the blocking of third-party tracking cookies.
  - c) As an example, suppose a user visits `www.example1.com`.
  - d) This web site contains an advert from `ad.foxytracking.com`, which, when downloaded, sets a cookie belonging to the advert's domain (`ad.foxytracking.com`).
  - e) Then, the user visits another website, `www.example2.com`, which also contains an advert from `ad.foxytracking.com`, and which also sets a cookie belonging to that domain (`ad.foxytracking.com`).
  - f) Eventually, both of these cookies will be sent to the advertiser when loading their ads or visiting their website.
  - g) The advertiser can then use these cookies to build up a browsing history of the user across all the websites that have ads from this advertiser.

## Propiedades de un cookie

Un cookie tiene los siguientes atributos:

1. nombre
2. valor
3. domain (dominio)
4. path o camino
5. secure / seguridad

Cuando ejecutamos este programa:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat study_cookie1.ru
run lambda { |e|
 [200,
 { 'Content-Type' => 'text/html',
 'Set-Cookie' => "name=John; path=/; secure"
 }
]
}
```

```

 'Set-cookie' => "id=123456\nname=jack\nphone=65452334"
},
['hello world']
]
}

```

y hacemos `www.example.com` un alias de `127.0.0.1`:

```
[~]$ cat /etc/hosts
##
Host Database
#
localhost is used to configure the loopback interface
when the system is booting. Do not change this entry.
##
127.0.0.1 localhost www.example.com
```

al visitar la página `www.example.com:9292` y abrir las herramientas para desarrolladores tenemos:

Observemos que:

1. Como no hemos establecido el tiempo de caducidad ( `expires Max-Age` ), los cookies son de sesión.
2. Como no hemos establecido el dominio, los cookies son de dominio `www.example.com`.

### Estableciendo `expires`

Modifiquemos el ejemplo anterior para establecer una fecha de caducidad:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat study_cookie2.ru
run lambda { |e|
 t = Time.now.gmtime + 3*60
 [200,
 { 'Content-Type' => 'text/html',
 'Set-cookie' => "chuchu=chachi;expires=#{t.strftime("%a, %d-%b-%Y %H:%M:%S GMT")}"
 },
 ['hello world']
]
}
```

Al ejecutar este programa vemos que hemos establecido la caducidad. Obsérvese la diferencia entre GMT y el tiempo de Canarias.

### Estableciendo el atributo `domain` de una cookie

1. Establezcamos `domain` a `example.com`:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat study_cookie3.ru
run lambda { |e|
 t = Time.now.gmtime + 3*60
 [200,
 { 'Content-Type' => 'text/html',
 'Set-cookie' => "chuchu=chachi;expires=#{t.strftime("%a, %d-%b-%Y %H:%M:%S GMT")}"
 ";domain=example.com"
 },
 ['hello world']
]
}
```

2. Manipulamos /etc/hosts:

```
[~]$ cat /etc/hosts
127.0.0.1 localhost www.example.com test.example.com app.test
```

3. Ejecutamos el servidor y lo visitamos con el navegador en www.example.com:9292.

4. A continuación arrancamos este segundo servidor en el puerto 8080:

```
[~/local/src/ruby/sinatra/rack/rack-simple(master)]$ cat config.ru
require './myapp'
run MyApp.new

[~/local/src/ruby/sinatra/rack/rack-simple(master)]$ cat myapp.rb
my_app.rb
#
class MyApp
 def call env
 [200, {"Content-Type" => "text/html"}, ["Hello Rack Participants"]]
 end
end
```

5. y visitamos test.example.com:8080 (que de nuevo es resuelto a localhost)

La figura muestra que el cookie generado por www.example.com:9292 es enviado a test.example.com:8080:

### El atributo path

Si path es / entonces casa con todos las páginas en el dominio. Si path es /foo entonces casa con foobar y /foo/chuchu/toto.html.

### El atributo secure

Si se pone secure el cookie solo se envía si se usa https

**Envío de Cookies** As long as the URL requested is within the same domain and path defined in the cookie (and all of the other restrictions – secure, not expired, etc) hold, the cookie will be sent for every request. The client will include a header field similar to this:

Cookie: name1 = value1 [;name2=value2]

### Establecer un cookie usando Rack::Response

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat hello_cookie.rb
require 'rack'
```

```
class HelloWorld
 def call env
 response = Rack::Response.new("Hello world!")
 response.status = 200
 response.headers['Content-type'] = "text/plain"
 response.set_cookie('asignatura', 'SYTW')
 response.finish
 end
end
```

```
Rack::Handler::WEBrick::run HelloWorld.new
```

## Obtener los valores de los cookies usando Rack::Request

Es posible acceder a los cookies con el objeto Rack::Request mediante el método cookies. Vease la documentación de Rack::Response y Rack::Request.

```
[~/rack/rack-debugging(master)]$ cat hello_cookie.rb
require 'rack'

class HelloWorld
 def call env
 req = Rack::Request.new(env)
 response = Rack::Response.new("Hello world! cookies = #{req.cookies.inspect}\n")
 response.write("asignatura => #{req.cookies['asignatura']}") if req.cookies['asignatura']
 response.status = 200
 response['Content-type'] = "text/plain"
 response.set_cookie('asignatura', 'SYTW')
 response.finish
 end
end
```

```
Rack::Handler::WEBrick::run HelloWorld.new
```

**El código del método cookies** El método cookies retorna un hash:

```
File lib/rack/request.rb, line 290
def cookies
 hash = @env["rack.request.cookie_hash"] ||= {}
 string = @env["HTTP_COOKIE"]

 return hash if string == @env["rack.request.cookie_string"]
 hash.clear

 # According to RFC 2109:
 # If multiple cookies satisfy the criteria above, they are ordered in
 # the Cookie header such that those with more specific Path attributes
 # precede those with less specific. Ordering with respect to other
 # attributes (e.g., Domain) is unspecified.
 cookies = Utils.parse_query(string, ',') { |s| Rack::Utils.unescape(s) rescue s }
 cookies.each { |k,v| hash[k] = Array === v ? v.first : v }
 @env["rack.request.cookie_string"] = string
 hash
end
```

## Código de set\_cookie

```
 # File lib/rack/response.rb, line 57
57: def set_cookie(key, value)
58: Utils.set_cookie_header!(header, key, value)
59: end
```

Aquí value es un hash con claves :domain, :path, :expires, :secure y :httponly

## Código de delete\_cookie

```

File lib/rack/response.rb, line 61
61: def delete_cookie(key, value={})
62: Utils.delete_cookie_header!(header, key, value)
63: end

```

Aquí `value` es un hash con claves :domain, :path, :expires, :secure y :httponly

### **domains, periods, cookies and localhost**

1. By design domain names must have at least two dots otherwise browser will say they are invalid.
2. Only hosts within the specified domain can set a cookie for a domain
3. domains must have at least two (2) or three (3) periods in them to prevent domains of the form: .com, .edu, and va.us.
4. Any domain that falls within one of the seven special top level domains COM, EDU, NET, ORG, GOV, MIL, and INT require two periods.
5. Any other domain requires at least three.
6. On `localhost`, when we set a cookie on server side and specify the domain explicitly as `localhost` (or `.localhost`), the cookie does not seem to be accepted by some browsers.

## **31.9. Gestión de Sesiones**

### **Introducción**

1. Hypertext Transfer Protocol (HTTP) is stateless: a client computer running a web browser must establish a new Transmission Control Protocol (TCP) network connection to the web server with each new HTTP GET or POST request.
2. The web server, therefore, cannot rely on an established TCP network connection for longer than a single HTTP GET or POST operation.
3. *Session management* is the technique used by the web developer to make the stateless HTTP protocol support session state.
4. For example, once a user has been authenticated to the web server, the user's next HTTP request (GET or POST) should not cause the web server to ask for the user's account and password again.
5. The session information is stored on the web server using the *session identifier* generated as a result of the first (sometimes the first authenticated) request from the end user running a web browser.
6. The "storage" of Session IDs and the associated session data (user name, account number, etc.) on the web server is accomplished using a variety of techniques including, but not limited to, local memory, flat files, and databases.
7. A *session token* is a unique identifier that is generated and sent from a server to a client to identify the current interaction session.
8. The client usually stores and sends the token as an HTTP cookie and/or sends it as a parameter in GET or POST queries. The reason to use session tokens is that the client only has to handle the identifier—all session data is stored on the server (usually in a database, to which the client does not have direct access) linked to that identifier.

## Uso de Cookies para el manejo de sesiones

1. Allowing users to log into a website is a frequent use of cookies.
2. A web server typically sends a cookie containing a unique *session identifier*.
3. The web browser will send back that session identifier with each subsequent request and [related items are stored associated with this unique session identifier](#).
4. Typically the web server will first send a cookie containing a unique session identifier. Users then submit their credentials and the web application authenticates the session and allows the user access to services.
5. Applications today usually store the gathered information in a database on the server side, rather than storing them in cookies

### Ejemplo

Rack::Session::Cookie proporciona un sencillo sistema para gestionar sesiones basado en cookies.

1. La `session` es un cookie que contiene un hash almacenado mediante marshalling codificado en base64.
2. Por defecto el nombre del cookie es `rack.session` pero puede ser modificado mediante el atributo `:key`.
3. Dándole un valor a `secret_key` se garantiza que es comprobada la integridad de los datos de la cookie
4. Para acceder dentro de nuestro programa a la sesión accedemos al hash `env["rack.session"]` o bien `env["key-value"]` si hemos especificado el atributo `:key`

Sigue un ejemplo:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat configapp.ru
require 'pp'
require './myapp'

use Rack::Session::Cookie,
 :key => 'rack.session',
 :domain => 'example.com',
 :secret => 'some_secret'

run MyApp.new

[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat myapp.rb
class MyApp

 def set_env(env)
 @env = env
 @session = env['rack.session']
 end

 def some_key
 return @session['some_key'].to_i if @session['some_key']
 @session['some_key'] = 0
 end
```

```

def some_key=(value)
 @session['some_key'] = value
end

def call(env)
 set_env(env)
 res = Rack::Response.new
 req = Rack::Request.new env

 self.some_key = self.some_key + 1 if req.path == '/'

 res.write("some_key = #{@session['some_key']}\n")

 res.finish
end

end

```

Hagamos la prueba conectándonos a `www.example.com`. Para ello editamos `/etc/hosts` para que `localhost` apunte a `www.example.com`:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ cat /etc/hosts
##
Host Database
#
localhost is used to configure the loopback interface
when the system is booting. Do not change this entry.
##
127.0.0.1 localhost www.example.com
...
```

Arrancamos el servidor:

```
[~/local/src/ruby/sinatra/rack/rack-session-cookie(master)]$ rackup configapp.ru
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
```

Y visitamos `www.example.com` con nuestro navegador:

### 31.9.1. Ejercicio: Race Conditions. Concurrencia

Supongamos el siguiente programa rack en el que se incrementa la variable `@some_key` (Véase GitHub: `crguezl/rack-race-conditions`):

```
[~/local/src/ruby/sinatra/rack/rack-appvswebserver(icon)]$ cat configapp.ru
class Persistence

def call(env)

 res = Rack::Response.new
 req = Rack::Request.new env

 @some_key ||= 0
 @some_key = @some_key + 1
```

```

 res.write("@some_key = #{@some_key}\n")

 res.finish
 end

end

```

run Persistence.new

Supongamos que arranco el servidor:

```
[~/local/src/ruby/sinatra/rack/rack-appswebserver(master)]$ rackup configapp.ru >> Thin web s
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
```

Nótese que con thin arrancado desde rack se tienen los valores de env para las claves:

```
rack.multithread => false
rack.multiprocess => false
```

lo que indica que el servidor no está soportando multithreading ni multiproceso.

Responda a estas preguntas:

1. ¿Que valores de @some\_key serán mostrados cuando me conecto a localhost:9292?
2. ¿Y si recargo la página varias veces?
3. ¿Y si abro un nuevo navegador o ventana de incógnito en la misma URL?
4. ¿Y si re-arranco el servidor?
5. ¿Como afectaría a la conducta que el servidor fuera multithreading?

```
[~/local/src/ruby/sinatra/rack/rack-appswebserver(icon)]$ rvm use jruby-1.7.3
Using /Users/casiano/.rvm/gems/jruby-1.7.3
[~/local/src/ruby/sinatra/rack/rack-appswebserver(icon)]$ rackup configapp.ru
Puma 2.6.0 starting...
* Min threads: 0, max threads: 16
* Environment: development
* Listening on tcp://0.0.0.0:9292
rack.multithread => true
rack.multiprocess => false
```

```
[~/local/src/ruby/sinatra/rack/rack-appswebserver(icon)]$ cat Rakefile
desc "run the server"
task :default do
 sh <<--EOS"
 #rvm use jruby-1.7.3 &&
 #ruby -v &&
 rackup -s puma configapp.ru
 EOS
end

desc "run the client"
task :client do
 pids = []
```

```

(0...100).each do
 pids << fork do
 sh %q{curl -v 'http://localhost:9292' >> salida 2>> logs}
 end
end
puts pids
end

desc "remove output and logs"
task :clean do
 sh "rm -f salida logs"
end

```

De acuerdo a una respuesta en StackOverflow a la pregunta: Is Sinatra multi-threaded? I read else where that "sinatra is multi-threaded by default", what does that imply?

The choice is mainly made by the server and middleware you use:

1. Multi-Process, non-preforking: Mongrel, Thin, WEBrick, Zbatery
2. Multi-Process, preforking: Unicorn, Rainbows, Passenger
3. Evented (suited for sinatra-synchrony): Thin, Rainbows, Zbatery
4. Threaded: Net::HTTP::Server, Threaded Mongrel, Puma, Rainbows, Zbatery, Phusion Passenger Enterprise ↳= 4
5. Since Sinatra 1.3.0, Thin will be started in threaded mode, if it is started by Sinatra (i.e. with ruby app.rb, but not with the thin command, nor with rackup).

### 31.10. Ejemplo Simple Combinando Rack::Request, Rack::Response y Middleware (Lobster)

Este código se encuentra en <https://github.com/crguezl/rack-lobster>

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat lobster.rb
require 'rack/request'
require 'rack/response'

module Rack
 class Lobster
 LobsterString = "a lobster"

 def call(env)
 req = Request.new(env)

 req.env.keys.sort.each { |x| puts "#{x} => #{req.env[x]}" }

 if req.GET["flip"] == "left"
 lobster = LobsterString.reverse
 href = "?flip=right"
 elsif req.GET["flip"] == "crash"
 raise "Lobster crashed"
 else
 lobster = LobsterString
 href = "?flip=left"
 end

 [200, {"Content-Type" => "text/html"}, ["<html><body>" + lobster + "</body></html>"]]
 end
 end
end
```

```

 end

 res = Response.new
 res.write <<-"EOS"
 <title>Lobstericious!</title>
 <pre>
 #{lobster}
 </pre>
 <p>flip!</p>
 <p>crash!</p>
 EOS
 res.finish
 end
end
end

if $0 == __FILE__
 require 'rack'
 require 'rack/showexceptions'
 Rack::Server.start(
 :app => Rack::ShowExceptions.new(
 Rack::Lint.new(
 Rack::Lobster.new)),
 :Port => 9292,
 :server => 'thin'
)
end

```

Véase:

1. rack/lib/rack/showexceptions.rb

(Rack::ShowExceptions catches all exceptions raised from the app it wraps. It shows a useful backtrace with the sourcefile and clickable context, the whole Rack environment and the request data.

Be careful when you use this on public-facing sites as it could reveal information helpful to attackers)

2. rack/lib/rack/lint.rb en GitHub

(Rack::Lint validates your application and the requests and responses according to the Rack spec)

Tanto Rack::ShowExceptions como Rack::Lint disponen de un método `call` que recibe una variable `env` describiendo el entorno CGI. Esto es, se trata de aplicaciones que siguen el protocolo Rack. Así este código:

```

Rack::Server.start(
 :app => Rack::ShowExceptions.new(
 Rack::Lint.new(
 Rack::Lobster.new)),
 :Port => 9292,
 :server => 'thin'
)

```

construye una nueva objeto/aplicación Rack que es la composición de los tres Racks.

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat Rakefile
desc "run the server"
task :default do
 sh "ruby lobster.rb"
end

desc "run the client flip left"
task :left do
 sh %q{curl -v 'http://localhost:9292?flip=left'}
end

desc "run the client flip right"
task :right do
 sh %q{curl -v 'http://localhost:9292?flip=right'}
end

desc "run the client. Generate exception"
task :crash do
 sh %q{curl -v 'http://localhost:9292/?flip=crash'}
end
```

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake left
curl -v 'http://localhost:9292?flip=left'
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /?flip=left HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 168
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
< <title>Lobstericious!</title>
< <pre>
 retsbol a
 </pre>
 <p>flip!</p>
 <p>crash!</p>
* Connection #0 to host localhost left intact
* Closing connection #0
```

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake right
curl -v 'http://localhost:9292?flip=right'
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /?flip=right HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
```

```

> Host: localhost:9292
> Accept: /*
>
< HTTP/1.1 200 OK
< Content-Length: 167
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<title>Lobstericious!</title>
<pre>
a lobster
</pre>
<p>flip!</p>
<p>crash!</p>
* Connection #0 to host localhost left intact
* Closing connection #0

```

### 31.11. Práctica: Accediendo a Twitter y Mostrando los últimos twitts en una página

Convierta el programa de ejemplo usado en la sección *Ejemplo en Ruby: Accediendo a Twitter* 8.15 en una aplicación Rack que muestre en su página los últimos twitts de una lista de usuarios obtenidos desde un formulario (puede modificar/diseñar la interfaz como crea conveniente)

### 31.12. Ejemplo: Basic Authentication con Rack::Auth::Basic

Rack::Auth::Basic implements HTTP Basic Authentication, as per RFC 2617.

#### Introducción

1. In the context of an HTTP transaction, basic access authentication is a method for an HTTP user agent to provide a user name and password when making a request.
2. *HTTP Basic authentication (BA)* implementation is the simplest technique for enforcing access controls to web resources because it doesn't require cookies, session identifier and login pages. Rather, HTTP Basic authentication uses static, standard HTTP headers which means that no handshakes have to be done in anticipation.
3. The BA mechanism provides no confidentiality protection for the transmitted credentials. They are merely encoded with BASE64 in transit, but not encrypted or hashed in any way. Basic Authentication is, therefore, typically used over HTTPS.
4. Because BA header has to be sent with each HTTP request, the web browser needs to cache the credentials for a reasonable period to avoid constant prompting user for the username and password. Caching policy differs between browsers.
5. While HTTP does not provide a method for web server to instruct the browser to "log out" the user (forget cached credentials), there are a number of workarounds using specific features in various browsers. One of them is redirecting the user to an URL on the same domain containing credentials that are intentionally incorrect

## Protocolo

1. When the server wants the user agent to authenticate itself towards the server, it can send a request for authentication.
2. This request should be sent using the HTTP 401 Not Authorized response code containing a WWW-Authenticate HTTP header.
3. The WWW-Authenticate header for basic authentication (used most often) is constructed as following:

```
WWW-Authenticate: Basic realm="insert realm"
```

4. When the user agent wants to send the server authentication credentials it may use the Authorization header.
  5. The Authorization header is constructed as follows:
    - a) Username and password are combined into a string `username:password`
    - b) The resulting string literal is then encoded using Base64
    - c) The authorization method and a space i.e. `Basic` is then put before the encoded string.
    - d) For example, if the user agent uses `Aladdin` as the username and `open sesame` as the password then the header is formed as follows:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

## Ejemplo de BA en Rack

Initialize with the Rack application that you want protecting, and a block that checks if a username and password pair are valid.

Puede encontrar el fuente en GitHub

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat protectedlobster.rb
require 'rack'
require './lobster'
require 'yaml'

lobster = Rack::Lobster.new

passwd = YAML.load(File.open('etc/passwd.yml')).read

protected_lobster = Rack::Auth::Basic.new(lobster) do |username, password|
 passwd[username] == password
end

protected_lobster.realm = 'Lobster 2.0'
pretty_protected_lobster = Rack::ShowStatus.new(Rack::ShowExceptions.new(protected_lobster))

Rack::Server.start :app => pretty_protected_lobster, :Port => 9292
```

## lobster.rb

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat lobster.rb
require 'rack/request'
require 'rack/response'

module Rack
 class Lobster
 LobsterString = "a lobster"

 def call(env)
 req = Request.new(env)

 req.env.keys.sort.each { |x| puts "#{x} => #{req.env[x]}" }

 if req.GET["flip"] == "left"
 lobster = LobsterString.reverse
 href = "?flip=right"
 elsif req.GET["flip"] == "crash"
 raise "Lobster crashed"
 else
 lobster = LobsterString
 href = "?flip=left"
 end

 res = Response.new
 res.write <<-"EOS"
 <title>Lobstericious!</title>
 <pre>
 #{lobster}
 </pre>
 <p>flip!</p>
 <p>crash!</p>
 EOS
 res.finish
 end
 end
end

if $0 == __FILE__
 require 'rack'
 require 'rack/showexceptions'
 Rack::Server.start(
 :app => Rack::ShowExceptions.new(
 Rack::Lint.new(
 Rack::Lobster.new)),
 :Port => 9292,
 :server => 'thin'
)
end
```

## etc/passwd.yml

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat etc/passwd.yml
```

```
--- # Indented Block
casiano: tutu
ana: titi
```

## Rakefile

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ cat Rakefile
...
desc "run the server for protectedlobster"
task :protected do
 sh "ruby protectedlobster.rb"
end

desc "run the client with user and password flip left"
task :protectedleft do
 sh %q{curl -v --basic -u casiano:tutu 'http://localhost:9292?flip=left'}
end

...
task :crash do
 sh %q{curl -v 'http://localhost:9292/?flip=crash'}
end
```

## Ejecución

1. Servidor:

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake protected
ruby protectedlobster.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
```

2. Cliente:

```
[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake protectedleft
curl -v --basic -u casiano:tutu 'http://localhost:9292?flip=left'
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
* Server auth using Basic with user 'casiano'
> GET /?flip=left HTTP/1.1
> Authorization: Basic Y2FzaWFubzpZWNyZXRv
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.8
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 168
< Connection: keep-alive
```

```

< Server: thin 1.5.1 codename Straight Razor
<
<title>Lobstericious!</title>
<pre>
retsbol a
</pre>
<p>flip!</p>
<p>crash!</p>
* Connection #0 to host localhost left intact
* Closing connection #0

```

3. Servidor después de la petición:

```

[~/local/src/ruby/sinatra/rack/rack-lobster(master)]$ rake protected
ruby protectedlobster.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
...
HTTP_AUTHORIZATION => Basic Y2FzaWFubzp0dXR1
REMOTE_USER => casiano
...

```

Véase

1. Código de rack-lobster en GitHub
2. Código fuente de Rack::Auth::Basic
3. Documentación en Rack::Auth::Basic
4. La Wikipedia Basic Access Authentication

### 31.13. Redirección

```

[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat redirect.rb
require 'rack'
require 'thin'

app = lambda do |env|
 req = Rack::Request.new env
 res = Rack::Response.new

 if req.path_info == '/redirect'
 res.redirect('https://plus.google.com/u/0/')
 else
 res.write "You did not get redirected"
 end
 res.finish
end

Rack::Server.start(
 :app => app,

```

```
:Port => 9292,
:server => 'thin'
)
```

### 31.14. La Estructura de una Aplicación Rack: Ejemplo de Middleware

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat middlefoo.rb
require 'rack'
```

```
class MiddleFoo
```

```
def initialize(app)
 @app = app
end

def call env
 # Podemos modificar el request (env) aqui
 env['chuchu'] = 'SYTW'
 status, headers, body = @app.call(env)
 # Podemos modificar la respuesta aqui
 newbody = body.map(&:upcase)
 [status, headers, newbody]
end
end
```

```
[~/local/src/ruby/sinatra/rack/rack-debugging(master)]$ cat hello_middle.rb
require 'rack'
require './middlefoo'
```

```
class HelloWorld
```

```
 def call env
 [200, {"Content-Type" => "text/plain"}, ["Hello world\nchuchu=#{env['chuchu']}\n"]]
 end
end
```

```
Rack::Handler::WEBrick::run MiddleFoo.new(HelloWorld.new)
```

Cuando ejecutamos el programa produce la salida:

```
HELLO WORLD
CHUCHU=SYTW
```

### 31.15. rackup

#### Introducción

1. The Rack gem gives you a rackup command which lets you start your app on any supported application server.
2. rackup is a useful tool for running Rack applications, which uses the Rack::Builder DSL to configure middleware and build up applications easily.
3. rackup automatically figures out the environment it is run in, and runs your application as FastCGI, CGI, or standalone with Mongrel or WEBrick, all from the same configuration.

De hecho este es todo el código del ejecutable rackup

```
#!/usr/bin/env ruby
```

```
require "rack"
Rack::Server.start
```

El método `start` starts a new rack server (like running rackup). This will parse ARGV and provide standard ARGV rackup options, defaulting to load `config.ru`.

Providing an option hash will prevent ARGV parsing and will not include any default options.

This method can be used to very easily launch a CGI application, for example:

```
Rack::Server.start(
 :app => lambda do |e|
 [200, {'Content-Type' => 'text/html'}, ['hello world']]
 end,
 :server => 'cgi'
)
```

Further options available here are documented on `Rack::Server#initialize` (véase el código en `Rack::Server`):

```
def self.start(options = nil)
 new(options).start
end
```

como se ve, el código de `Rack::Server` está en Github.

The Options of `start` and `new` may include:

1. `:app` a rack application to run (overrides `:config`)
2. `:config` a rackup configuration file path to load (`.ru`)
3. `:environment` this selects the middleware that will be wrapped around your application. Default options available are:
  - a) development: CommonLogger, ShowExceptions, and Lint
  - b) deployment: CommonLogger
  - c) none: no extra middleware
- note: when the server is a cgi server, CommonLogger is not included.
4. `:server` choose a specific Rack::Handler, e.g. cgi, fcgi, webrick
5. `:daemonize` if true, the server will daemonize itself (fork, detach, etc)
6. `:pid` path to write a pid file after daemonize
7. `:Host` the host address to bind to (used by supporting Rack::Handler)
8. `:Port` the port to bind to (used by supporting Rack::Handler)
9. `:AccessLog` webrick access log options (or supporting Rack::Handler)
10. `:debug` turn on debug output (`$DEBUG = true`)
11. `:warn` turn on warnings (`$-w = true`)
12. `:include` add given paths to `$LOAD_PATH`
13. `:require` require the given libraries

## Ejemplo de uso

Si no se especifica, `rackup` busca un fichero con nombre `config.ru`.

```
[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ cat config.ru
require './myapp'
run MyApp.new
```

Esta es la aplicación:

```
[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ cat myapp.rb
class MyApp
 def call env
 [200, {"Content-Type" => "text/html"}, ["Hello Rack Participants"]]
 end
end

[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
 sh "rackup"
end

desc "run client via curl"
task :client do
 sh "curl -v localhost:9292"
end
```

## Ejecución

```
[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
```

```
[~/local/src/ruby/sinatra/rack/rackup/simple(master)]$ curl -v localhost:9292
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
* Closing connection #0
Hello Rack Participants
```

## Opciones del ejecutable rackup

1. Véase rackup.

```
[~]$ rackup --help
Usage: rackup [ruby options] [rack options] [rackup config]
```

### Ruby options:

-e, --eval LINE	evaluate a LINE of code
-d, --debug	set debugging flags (set \$DEBUG to true)
-w, --warn	turn warnings on for your script
-I, --include PATH	specify \$LOAD_PATH (may be used more than once)
-r, --require LIBRARY	require the library, before executing your script

### Rack options:

-s, --server SERVER	serve using SERVER (webrick/mongrel)
-o, --host HOST	listen on HOST (default: 0.0.0.0)
-p, --port PORT	use PORT (default: 9292)
-O NAME[=VALUE],	pass VALUE to the server as option NAME. If no VALUE, sets it to true. Run 'rackup -s SERVER -h' to get a list of options for SERVER
--option	
-E, --env ENVIRONMENT	use ENVIRONMENT for defaults (default: development)
-D, --daemonize	run daemonized in the background
-P, --pid FILE	file to store PID (default: rack.pid)

### Common options:

-h, -?, --help	Show this message
--version	Show version

**Especificación de Opciones en la primera línea** Si la primera línea de un fichero config.ru empieza por \# es tratada como una línea de opciones permitiendo así que los argumentos de rackup se especifiquen en el fichero de configuración:

```
#\w -p 8765

use Rack::Reloader, 0
use Rack::ContentLength

app = proc do |env|
 [200, {'content-Type' => 'text/plain'}, ['a']]
end

run app
```

## 31.16. Rack::Static

### Véase

1. Documentación de Rack::Static
2. Este ejemplo: rack-static-example en GitHub
3. Código fuente de Rack::Static en GitHub

## Ejemplo

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ tree
```

```
.
```

```
|--- README
```

```
|--- README.md
```

```
|--- Rakefile
```

```
|--- config.ru
```

```
|--- myapp.rb
```

```
'---- public
```

```
 '--- index.html
```

```
1 directory, 6 files
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ cat public/index.html
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD
```

```
<html>
```

```
 <head>
 <title>Hello</title>
 </head>
 <body>
 <h1>Hello World!</h1>
 </body>
</html>
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ cat config.ru
```

```
require './myapp'
```

```
use Rack::Static, :urls => ["/public"]
```

```
run MyApp.new
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ cat Rakefile
```

```
task :default => :server
```

```
desc "run server"
```

```
task :server do
 sh "rakeup"
end
```

```
desc "run client via curl"
```

```
task :client do
 sh "curl -v localhost:9292"
end
```

```
desc "access to static file"
```

```
task :index do
 sh "curl -v localhost:9292/public/index.html"
end
```

```
[~/local/src/ruby/sinatra/rack/rack-static(master)]$ cat myapp.rb
```

```
my_app.rb
```

```
#
```

```
class MyApp
```

```

def call env
 [200, {"Content-Type" => "text/html"}, ["Hello SYTW!"]]
end
end

[~/local/src/ruby/sinatra/rack/rack-static(master)]$ rake client
curl -v localhost:9292
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
* Closing connection #0
Hello SYTW!

[~/local/src/ruby/sinatra/rack/rack-static(master)]$ rake index
curl -v localhost:9292/public/index.html
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /public/index.html HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Last-Modified: Thu, 03 Oct 2013 08:24:43 GMT
< Content-Type: text/html
< Content-Length: 227
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD
<html>
 <head>
 <title>Hello</title>
 </head>
 <body>
 <h1>Hello World!</h1>
 </body>
</html>

* Connection #0 to host localhost left intact

```

```
* Closing connection #0
```

El comando `rackup`

`rackup` converts the supplied rack config file to an instance of `Rack::Builder`. In short, rack config files are evaluated within the context of a `Rack::Builder` object.

`Rackup` also has a `use` method that accepts a *middleware*. Let us use one of Rack's built-in middleware.

```
[~/sinatra/rackup/middleware]$ cat config.ru
require './myapp'
require './myrackmiddleware'
use Rack::Reloader
use MyRackMiddleware
run MyApp.new
```

```
[~/sinatra/rackup/middleware]$ cat myapp.rb
myapp.rb
class MyApp
 def call(env)
 [200, {"Content-Type" => "text/html"}, ["Hello Rack Participants from across the globe"]]
 end
end
```

```
[~/sinatra/rackup/middleware]$ cat myrackmiddleware.rb
class MyRackMiddleware
 def initialize(appl)
 @appl = appl
 end
 def call(env)
 status, headers, body = @appl.call(env)
 append_s = "... greetings from RubyLearning!!!"
 [status, headers, body << append_s]
 end
end
```

Véase

- RailCast #151 Rack Middleware
- Rack Middleware as a General Purpose Abstraction por Mitchell Hashimoto

### 31.17. Un Ejemplo Simple: Piedra, Papel, tijeras

```
[~/rack/rack-rock-paper-scissors(simple)]$ cat -n rps.rb
1 require 'rack/request'
2 require 'rack/response'
3
4 module RockPaperScissors
5 class App
6
7 def initialize(app = nil)
8 @app = app
9 @content_type = :html
10 @defeat = {'rock' => 'scissors', 'paper' => 'rock', 'scissors' => 'paper'}
```

```

11 @throws = @defeat.keys
12 @choose = @throws.map { |x|
13 "%Q{ #{x} }"
14 }.join("\n")
15 @choose = "<p>\n\n#{@choose}\n"
16 end
17
18 def call(env)
19 req = Rack::Request.new(env)
20
21 req.env.keys.sort.each { |x| puts "#{x} => #{req.env[x]}" }
22
23 computer_throw = @throws.sample
24 player_throw = req.GET["choice"]
25 answer = if !@throws.include?(player_throw)
26 "Choose one of the following:"
27 elsif player_throw == computer_throw
28 "You tied with the computer"
29 elsif computer_throw == @defeat[player_throw]
30 "Nicely done; #{player_throw} beats #{computer_throw}"
31 else
32 "Ouch; #{computer_throw} beats #{player_throw}. Better luck next time!"
33 end
34
35 res = Rack::Response.new
36 res.write <<-EOS
37 <html>
38 <title>rps</title>
39 <body>
40 <h1>
41 #{answer}
42 #{@choose}
43 </h1>
44 </body>
45 </html>
46 EOS
47 res.finish
48 end # call
49 end # App
50 end # RockPaperScissors
51
52 if $0 == __FILE__
53 require 'rack'
54 require 'rack/showexceptions'
55 Rack::Server.start(
56 :app => Rack::ShowExceptions.new(
57 Rack::Lint.new(
58 RockPaperScissors::App.new)),
59 :Port => 9292,
60 :server => 'thin'
61)
62 end

```

**El Objeto req** El objeto `req` pertenece a la clase `Rack::Request`. Tiene un único atributo `env`:

```
(rdb:1) req
#<Rack::Request:0x007f8d735b1410
@env={
 "SERVER_SOFTWARE"=>"thin 1.5.1 codename Straight Razor",
 "SERVER_NAME"=>"0.0.0.0",
 "rack.input"=>#<Rack::Lint::InputWrapper:0x007f8d735776c0
 @input=#<StringIO:0x007f8d735426a0>, "rack.version"=>[1, 0],
 "rack.errors"=>#<Rack::Lint::ErrorWrapper:0x007f8d73577620 @error=#<IO:<STDERR>
 >,
 "rack.multithread"=>false,
 "rack.multiprocess"=>false,
 "rack.run_once"=>false,
 "REQUEST_METHOD"=>"GET",
 "REQUEST_PATH"=>"/",
 "PATH_INFO"=>"/",
 "REQUEST_URI"=>"/",
 "HTTP_VERSION"=>"HTTP/1.1",
 "HTTP_HOST"=>"0.0.0.0:9292",
 "HTTP_CONNECTION"=>"keep-alive",
 "HTTP_ACCEPT"=>"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
 "HTTP_USER_AGENT"=>"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.36 (KHTML,
 "HTTP_ACCEPT_ENCODING"=>"gzip,deflate,sdch",
 "HTTP_ACCEPT_LANGUAGE"=>"es-ES,es;q=0.8",
 "GATEWAY_INTERFACE"=>"CGI/1.2",
 "SERVER_PORT"=>"9292",
 "QUERY_STRING"=>"",
 "SERVER_PROTOCOL"=>"HTTP/1.1",
 "rack.url_scheme"=>"http",
 "SCRIPT_NAME"=>"",
 "REMOTE_ADDR"=>"127.0.0.1",
 "async.callback"=>#<Method: Thin::Connection#post_process>,
 "async.close"=>#<EventMachine::DefaultDeferrable:0x007f8d735603f8>}>
```

Cuando llamamos a `GET` para obtener el valor del parámetro `choice`:

```
player_throw = req.GET["choice"]
```

Si visitamos la página `http://0.0.0.0:9292/` el entorno contiene algo como esto:

```
rdb:1) p @env
{"SERVER_SOFTWARE"=>"thin 1.5.1 codename Straight Razor",
 ...
 "QUERY_STRING"=>"",
 "REQUEST_URI"=>"/"
 ...
}
```

el código de `GET` nos da los datos almacenados en `QUERY_STRING`:

```
def GET
 if @env["rack.request.query_string"] == query_string
 @env["rack.request.query_hash"]
 else
 @env["rack.request.query_string"] = query_string
```

```

 @env["rack.request.query_hash"] = parse_query(query_string)
 end
end

def query_string; @env["QUERY_STRING"].to_s end

```

si es la primera vez, `@env["rack.request.query_string"]` está a nil y se ejecuta el `else` inicializando `@env["rack.request.query_string"]` y `@env["rack.request.query_hash"]`

Si por ejemplo visitamos la URL: `http://localhost:9292?choice=rock` entonces env contendrá:

```

rdb:1) p env
{
 ...
 "QUERY_STRING"=>"choice=rock",
 "REQUEST_URI"=>"/?choice=rock",
 ...
}

```

Familiaricemonos con algunos de los métodos de `Rack::Request`:

```

(rdb:1) req.GET
{"choice"=>"paper"}
(rdb:1) req.GET["choice"]
"paper"
(rdb:1) req.POST
{}
(rdb:1) req.params
{"choice"=>"paper"}
(rdb:1) req["choice"]
"paper"
(rdb:1) req[:choice]
"paper"
(rdb:1) req.cookies()
{}
(rdb:1) req.get?
true
(rdb:1) req.post?
false
(rdb:1) req.fullpath
"/?choice=paper"
(rdb:1) req.host
"0.0.0.0"
(rdb:1) req.host_with_port
"0.0.0.0:9292"
(rdb:1) req.body
#<Rack::Lint::InputWrapper:0x007f8d7369b5d8 @input=#<StringIO:0x007f8d73690318>>
(rdb:1) req.cookies()
{}
(rdb:1) req.get?
true
(rdb:1) req.post?
false
(rdb:1) req.fullpath
"/?choice=paper"
(rdb:1) req.host
"0.0.0.0"

```

```
(bdb:1) req.host_with_port
"0.0.0.0:9292"
(bdb:1) req.ip
"127.0.0.1"
(bdb:1) req.params
{"choice"=>"paper"}
(bdb:1) req.path
"/"
(bdb:1) req.path_info
"/"
(bdb:1) req.port
9292
(bdb:1) req.request_method
"GET"
(bdb:1) req.scheme
"http"
(bdb:1) req.url
"http://0.0.0.0:9292/?choice=paper"
(bdb:1) req.user_agent
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/29.0.1547.76 Safari/537.36"
(bdb:1) req.values_at("choice")
["paper"]
```

## Rakefile

```
[~/rack/rack-rock-paper-scissors(simple)]$ cat Rakefile
desc "run the server"
task :default do
 sh "ruby rps.rb"
end

desc "run the client with rock"
task :rock do
 sh %q{curl -v 'http://localhost:9292?choice=rock'}
end

desc "run the client with paper"
task :paper do
 sh %q{curl -v 'http://localhost:9292?choice=paper'}
end

desc "run the client with scissors"
task :scissors do
 sh %q{curl -v 'http://localhost:9292?choice=scissors'}
end

1. curl
```

## Ejecuciones

```
[~/rack/rack-rock-paper-scissors(simple)]$ rake
ruby rps.rb
>> Thin web server (v1.5.1 codename Straight Razor)
```

```

>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop

[~/rack/rack-rock-paper-scissors(simple)]$ rake rock
curl -v 'http://localhost:9292?choice=rock'
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /?choice=rock HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 332
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<html>
 <title>rps</title>
 <body>
 <h1>
 Nicely done; rock beats scissors
 <p>

 rock
 paper
 scissors

 </h1>
 </body>
</html>
* Connection #0 to host localhost left intact
* Closing connection #0

[~/rack/rack-rock-paper-scissors(simple)]$ rake
ruby rps.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
GATEWAY_INTERFACE => CGI/1.2
HTTP_ACCEPT => */
HTTP_HOST => localhost:9292
HTTP_USER_AGENT => curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.8
HTTP_VERSION => HTTP/1.1
PATH_INFO => /
QUERY_STRING => choice=rock
REMOTE_ADDR => 127.0.0.1
REQUEST_METHOD => GET
REQUEST_PATH => /
REQUEST_URI => /?choice=rock
SCRIPT_NAME =>
SERVER_NAME => localhost

```

```

SERVER_PORT => 9292
SERVER_PROTOCOL => HTTP/1.1
SERVER_SOFTWARE => thin 1.5.1 codename Straight Razor
async.callback => #<Method: Thin::Connection#post_process>
async.close => #<EventMachine::DefaultDeferrable:0x007ff4e2bf8e78>
rack.errors => #<Rack::Lint::ErrorWrapper:0x007ff4e2c04b88>
rack.input => #<Rack::Lint::InputWrapper:0x007ff4e2c04c00>
rack.multiprocess => false
rack.multithread => false
rack.run_once => false
rack.url_scheme => http
rack.version => [1, 0]

```

### Véase También

Véase la documentación de las siguientes clases:

1. Rack::Request
2. Rack::Response
3. Rack::Server
4. Rack::ShowExceptions
5. Rack::Lint

#### 31.17.1. Práctica: Rock, Paper, Scissors: Debugging

Implemente el ejemplo anterior Rock, Paper, Scissors y ejecútelo con un depurador. Lea la sección *Depurando una Ejecución con Ruby* 51.1.

Instale la gema `debugger`. Llame al método `debugger` en el punto en el que quiere detener la ejecución para inspeccionar el estado del programa. Arranque el servidor y en el navegador visite la página.

#### 31.17.2. Práctica: Añadir Template Haml a Rock, Paper, Scissors

Use Haml para crear un template `index.haml` en un directorio `views`.

```
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(template)]$ tree
.
|--- README
|--- Rakefile
|--- rps.rb
'--- views
 --- index.haml
```

El template puede ser usado así:

```

require 'rack/request'
require 'rack/response'
require 'haml'

module RockPaperScissors
 class App
 ...
 def call(env)
```

```

...
engine = Haml::Engine.new File.open("views/index.haml").read
res = Rack::Response.new
res.write engine.render({}, {
 :answer => answer,
 :choose => @choose,
 :throws => @throws)
res.finish
end # call
end # App
end # RockPaperScissors

```

Véase:

## 1. Haml::Engine

La sintáxis del método `render` es:

```
(String) render(scope = Object.new, locals = {})
```

También se puede usar como `to_html`. Procesa el template y retorna el resultado como una cadena.

El parámetro `scope` es el contexto en el cual se evalúa el template.

Si es un objeto `Binding` `haml` lo usa como segundo argumento de `Kernel#eval` (Véase la sección *Bindings (encarpetados) y eval* en 15.2.1) en otro caso, `haml` utiliza `#instance_eval`.

Nótese que Haml modifica el contexto de la evaluación (bien el objeto ámbito o el objeto `self` del ámbito del binding). Se extiende `Haml::Helpers` y se establecen diversas variables de instancia (todas ellas prefijadas con `haml_`).

Por ejemplo:

```
s = "foobar"
Haml::Engine.new("%p= upcase").render(s)
```

produce:

```
"<p>FOOBAR</p>"
```

Ahora `s` extiende `Haml::Helpers`:

```
s.respond_to?(:html_attrs) #=> true
```

`Haml::Helpers` contiene un conjunto de métodos/utilidades para facilitar distintas tareas. La idea de que estén disponibles en el contexto es para ayudarnos dentro del template. Por ejemplo el método

- (String) `escape_once(text)`

Escapa las entidades HTML en el texto.

`locals` es un hash de variables locales que se deja disponible dentro del template. Por ejemplo:

```
Haml::Engine.new("%p= foo").render(Object.new, :foo => "Hello, world!")
```

producirá:

```
"<p>Hello, world!</p>"
```

Si se pasa un bloque a `render` el bloque será ejecutado en aquellos puntos en los que se llama a `yield` desde el template.

Debido a algunas peculiaridades de Ruby, si el ámbito es un `Binding` y se proporciona también un bloque, el contexto de la evaluación puede no ser el que el usuario espera.

Parametros:

1. `scope` (`Binding`, `Proc`, `Object`) (por defecto: `Object.new`). El contexto en el que se evalúa el template
2. `locals` (`{Symbol => Object}`) (por defecto: `{}`). Variables locales que se dejan disponibles en el template
3. `block` (`#to_proc`) Un bloque que será llamado desde el template.
4. Retorna una `String` con el template renderizado

### 31.17.3. Práctica: Añada Hojas de Estilo a Piedra Papel Tijeras

Añada hojas de estilo a la práctica anterior (sección 31.17.2).

1. Mostramos una posible estructura de ficheros en la que se incluyen hojas de estilo usando `bootstrap`:

```
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(bootstrap)]$ tree
.
|--- Gemfile
|--- Gemfile.lock
|--- README
|--- Rakefile
|--- TODO
|--- config.ru
|--- lib
| '--- rps.rb
|--- public
| |--- css
| | |--- bootstrap-responsive.css
| | |--- bootstrap-responsive.min.css
| | |--- bootstrap.css
| | '-- bootstrap.min.css
| |--- img
| | |--- glyphicons-halflings-white.png
| | |--- glyphicons-halflings.png
| | '-- programming-languages.jpg
| '-- js
| |--- bootstrap.js
| '-- bootstrap.min.js
|--- rps.rb
'-- views
 '-- index.haml

6 directories, 18 files
```

2. El middleware `Rack::Static` intercepta las peticiones por ficheros estáticos (javascript, imágenes, hojas de estilo, etc.) basandose en los prefijos de las urls pasadas en las opciones y los sirve utilizando un objeto `Rack::File`. Ejemplos:

```
use Rack::Static, :urls => ["/public"]
```

Servirá todas las peticiones que comiencen por `/public` desde la carpeta `public` localizada en el directorio actual (esto es `public/*`).

En nuestro jerarquía pondremos en el programa `rps.rb`:

```

builder = Rack::Builder.new do
 use Rack::Static, :urls => ['/public']
 use Rack::ShowExceptions
 use Rack::Lint

 run RockPaperScissors::App.new
end

Rack::Handler::Thin.run builder

```

y dentro del template `haml` nos referiremos por ejemplo al fichero javascript como

```
%script{:src => "/public/js/bootstrap.js"}
```

Otro ejemplo:

```
use Rack::Static, :urls => ["/css", "/images"], :root => "public"
```

servirá las peticiones comenzando con `/css` o `/images` desde la carpeta `public` en el directorio actual (esto es `public/css/*` y `public/images/*`)

3. Véase el código en GitHub de `Rack::Static`
4. En el template `views/index.haml` deberá enlazar a las hojas de estilo:

```
!!!
%html{:lang => "en"}
 %head
 %meta{:charset => "utf-8"}/
 %title RPS
 %link{:href => "/public/css/bootstrap.css", :rel => "stylesheet"}
 %link{:href => "/public/css/bootstrap.css", :rel => "stylesheet"}
```

y las imágenes como:

```
%img(src="/public/img/programming-languages.jpg" width="40%)
```

5. `Rack::File` es un middleware que sirve los ficheros debajo del directorio dado, de acuerdo con el `path info` de la petición Rack. por ejemplo, cuando se usa `Rack::File.new("/etc")` podremos acceder al fichero `passwd` como `localhost:9292/passwd`.
6. Vease el código en github de `Rack::File`
7. Para saber mas de Bootstrap véase la sección 41

### 31.18. Middleware y la Clase `Rack::Builder`

We mentioned earlier that between the server and the framework, Rack can be customized to your applications needs using middleware.

The fundamental idea behind Rack middleware is

1. come between the calling client and the server,
2. process the HTTP request before sending it to the server, and
3. processing the HTTP response before returning it to the client.

## Motivación para el método `use`

Si tenemos una app Rack `rack_app` y dos middlewares con nombres `MiddleWare1` y `MiddleWare2` que queremos usar, podemos escribir esto:

```
Rack::Handler::Thin.run Middleware1.new(Middleware2.new(rack_app))
```

Si necesitamos pasar opciones en el segundo argumento la llamada quedaría mas o menos como esto:

```
Rack::Handler::Thin.run(
 Middleware1.new(
 Middleware2.new(rack_app, options2),
 options1)
)
```

Si fueran mas de dos middlewares el correspondiente código se volverá aún mas ilegible y hace mas fácil que metamos la pata cuando queramos hacer algo como - por ejemplo - modificar el orden de los middleware.

## La Clase `Rack::Builder`

La clase `Rack::Builder` implementa un pequeño DSL para facilitar la construcción de aplicaciones Rack.

`Rack::Builder` is the thing that glues various Rack middlewares and applications together and convert them into a single entity/rack application.

A good analogy is comparing `Rack::Builder` object with a stack, where at the very bottom is your actual rack application and all middlewares on top of it, and the whole stack itself is a rack application too.

1. El método `use` añade un middleware a la pila
2. El método `run` ejecuta una aplicación
3. El método `map` construye un `Rack::URLMap` en la forma apropiada. It mounts a stack of rack application/middleware on the specified path or URI.

## Conversión de una Aplicación Rack a `Rack::Builder`

Dada la aplicación:

```
infinity = Proc.new {|env| [200, {"Content-Type" => "text/html"}, env.inspect] }
Rack::Handler::Mongrel.run infinity, :Port => 9292
```

Podemos reescribirla:

```
[~/sinatra/rack/rack-builder/map]$ cat app_builder.rb
require 'rack'

infinity = Proc.new {|env| [200, {"Content-Type" => "text/html"}, [env.inspect]]}
builder = Rack::Builder.new
builder.run infinity
Rack::Handler::Thin.run builder, :Port => 9292
```

o bien:

```
[~/sinatra/rack/rack-builder/map]$ cat app_builder2.rb
require 'rack'
```

```
infinity = Proc.new {|env| [200, {"Content-Type" => "text/html"}, [env.inspect]]}
builder = Rack::Builder.new do
 run infinity
end
Rack::Handler::Thin.run builder, :Port => 9292
```

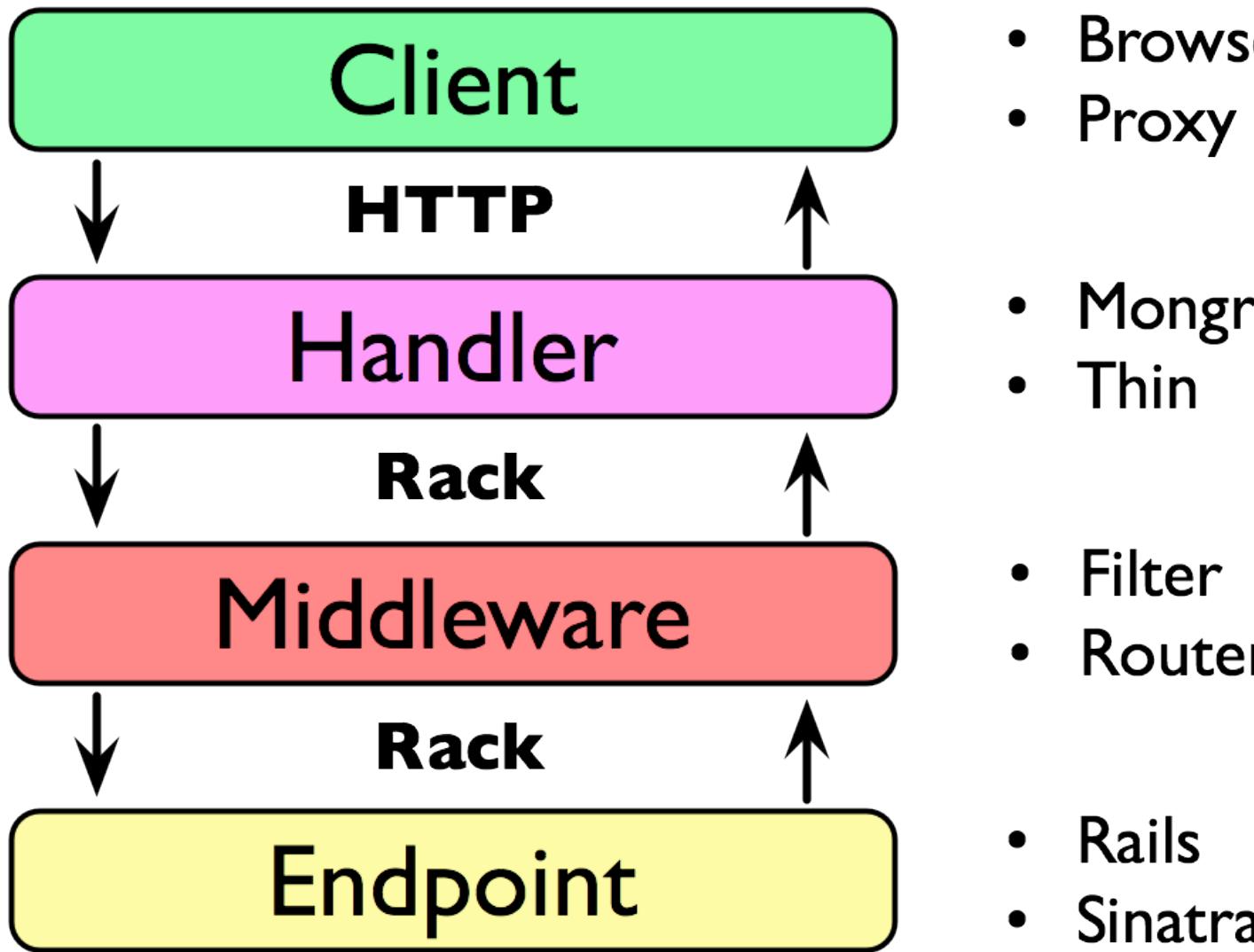


Figura 31.1: La pila Rack

#### Ejemplo Simple de Uso de Rack::Builder

```
[~/local/src/ruby/sinatra/rack/rack-builder/simple1]$ cat app.rb
require 'rack'
require 'rack/server'

app = Rack::Builder.new do
 use Rack::CommonLogger
 use Rack::ShowExceptions
 use Rack::Lint
 map "/chuchu" do
 run lambda { |env| [200, {}, ["hello"]] }
 end
 map "/chachi" do
 run lambda { |env| [200, {}, ["world"]] }
 end
 run lambda { |env| [200, {}, ["everything"]] }
end
```

```
Rack::Server.start :app => app
```

### Ejemplo de Uso de Rack::Builder: Dos Middlewares

```
[~/rack/rack-from-the-beginning(master)]$ cat hello_world.rb
hello_world.rb
require 'rack'
require 'rack/server'

class EnsureJsonResponse
 def initialize(app = nil)
 @app = app
 end

 # Set the 'Accept' header to 'application/json' no matter what.
 # Hopefully the next middleware respects the accept header :)
 def call(env)
 env['HTTP_ACCEPT'] = 'application/json'
 puts "env['HTTP_ACCEPT'] = #{env['HTTP_ACCEPT']}"
 @app.call(env) if @app
 end
end

class Timer
 def initialize(app = nil)
 @app = app
 end

 def call(env)
 before = Time.now
 status, headers, body = @app.call(env) if @app

 headers['X-Timing'] = (Time.now - before).to_i.to_s

 [status, headers, body]
 end
end

class HelloWorldApp

 def initialize(app = nil)
 @app = app
 end

 def self.call(env)
 [200, {}, ['hello world!']]
 end
end

put the timer at the top so it captures everything below it
app = Rack::Builder.new do
 use Timer # put the timer at the top so it captures everything below it
 use EnsureJsonResponse
```

```

 run HelloWorldApp
end

Rack::Server.start :app => app

~/rack/rack-from-the-beginning(master)]$ cat Rakefile
desc "run the server"
task :default do
 sh "rakeup"
end

desc "run the server hello_world.rb"
task :server do
 sh "ruby hello_world.rb"
end

desc "run the client"
task :client do
 sh %q{curl -v 'http://localhost:9292'}
end

desc "run the client for hello_world"
task :client2 do
 sh %q{curl -v 'http://localhost:8080'}
end

[~/rack/rack-from-the-beginning(master)]$ rake server
ruby hello_world.rb
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:8080, CTRL+C to stop

[~/rack/rack-from-the-beginning(master)]$ rake client2
curl -v 'http://localhost:8080'
* About to connect() to localhost port 8080 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Timing: 0
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
* Closing connection #0
hello world!

[~/rack/rack-from-the-beginning(master)]$ rake server
ruby hello_world.rb
>> Thin web server (v1.5.1 codename Straight Razor)

```

```

>> Maximum connections set to 1024
>> Listening on 0.0.0.0:8080, CTRL+C to stop
env['HTTP_ACCEPT'] = application/json

```

### 31.19. Ejemplo de Middleware: Rack::ETag

An ETag or *entity tag*, is part of HTTP, the protocol for the World Wide Web. It is one of several mechanisms that HTTP provides for *web cache validation*, and which allows a client to make conditional requests.

This allows caches to be more efficient, and saves bandwidth, as a web server does not need to send a full response if the content has not changed.

An ETag is an opaque identifier assigned by a web server to a specific version of a resource found at a URL.

If the resource content at that URL ever changes, a new and different ETag is assigned.

Used in this manner ETags are similar to fingerprints, and they can be quickly compared to determine if two versions of a resource are the same or not.

1. Rack::ETag en GitHub
2. Documentación de Rack::ETag

```

require 'digest/md5'

module Rack
 class ETag
 DEFAULT_CACHE_CONTROL = "max-age=0, private, must-revalidate".freeze

 def initialize(app, no_cache_control = nil, cache_control = DEFAULT_CACHE_CONTROL)
 @app = app
 @cache_control = cache_control
 @no_cache_control = no_cache_control
 end

 def call(env)
 status, headers, body = @app.call(env)

 if etag_status?(status) && etag_body?(body) && !skip_caching?(headers)
 digest, body = digest_body(body)
 headers['ETag'] = %("#{digest}"") if digest
 end

 unless headers['Cache-Control']
 if digest
 headers['Cache-Control'] = @cache_control if @cache_control
 else
 headers['Cache-Control'] = @no_cache_control if @no_cache_control
 end
 end

 [status, headers, body]
 end

 private

```

```

def etag_status?(status)
 status == 200 || status == 201
end

def etag_body?(body)
 !body.respond_to?(:to_path)
end

def skip_caching?(headers)
 (headers['Cache-Control'] && headers['Cache-Control'].include?('no-cache')) ||
 headers.key?('ETag') || headers.key?('Last-Modified')
end

def digest_body(body)
 parts = []
 digest = nil

 body.each do |part|
 parts << part
 (digest ||= Digest::MD5.new) << part unless part.empty?
 end

 [digest && digest.hexdigest, parts]
end
end
end

```

## 31.20. Construyendo Nuestro Propio Rack::Builder

Véase:

1. <https://github.com/crguezl/rack-mybuilder>

```
[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ cat mybuilder.rb
module Rack
 class MyBuilder

 def initialize(&block)
 @use = []
 instance_eval(&block) if block_given?
 end

 def use(middleware, *args, &block)
 @use << proc { |app| middleware.new(app, *args, &block) }
 end

 def run(app)
 @run = app
 end

 def to_app
 @use.reverse.inject(@run) { |app, middleware| middleware[app] }
 end
 end
end
```

```

def call(env)
 to_app.call(env)
end

end
end

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ cat decorator.rb
class Decorator

 def initialize(app, *options, &block)
 @app = app
 @options = (options[0] || {})
 end

 def call(env)
 status, headers, body = @app.call(env)

 new_body = ""
 new_body << (@options[:header] || "----Header----
")
 body.each { |str| new_body << str }
 new_body << (@options[:footer] || "
----Footer----")

 [status, headers, [new_body]]
 end
end

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ cat app.rb
require 'rack'
require 'thin'

require 'mybuilder'
require 'decorator'

app = Rack::MyBuilder.new do
 use Decorator, :header => "***** header *****
"

 cheer = ARGV.shift || "<h1>Hello world!</h1>"
 run lambda { |env| [200, { 'Content-Type' => 'text/html' }, ["<h1>#{cheer}</h1>"]] }
end

Rack::Handler::Thin.run app, :Port => 3333, :Host => 'localhost'

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ cat Rakefile
desc "run app server"
task :default => :server

desc "run app server"
task :server, :greet do |t, args|
 cheer = args[:greet] || 'bye, bye!'
 sh "ruby -I. app.rb #{cheer}"
end

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ rake -T

```

```

rake default # run app server
rake server[greet] # run app server

[~/local/src/ruby/sinatra/rack/rack-builder/own(master)]$ rake server[tachaaaAAAAAANNN]
ruby -I. app.rb tachaaaAAAAAANNN
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on localhost:3333, CTRL+C to stop

```

### 31.21. Código de Rack::Builder

Tomado de <https://github.com/rack/rack/blob/master/lib/rack/builder.rb>:

```

module Rack
 # Rack::Builder implements a small DSL to iteratively construct Rack
 # applications.
 #
 # Example:
 #
 # require 'rack/lobster'
 # app = Rack::Builder.new do
 # use Rack::CommonLogger
 # use Rack::ShowExceptions
 # map "/lobster" do
 # use Rack::Lint
 # run Rack::Lobster.new
 # end
 # end
 #
 # run app
 #
 # Or
 #
 # app = Rack::Builder.app do
 # use Rack::CommonLogger
 # run lambda { |env| [200, {'Content-Type' => 'text/plain'}, ['OK']] }
 # end
 #
 # run app
 #
 # +use+ adds middleware to the stack, +run+ dispatches to an application.
 # You can use +map+ to construct a Rack::URLMap in a convenient way.

 class Builder
 def self.parse_file(config, opts = Server::Options.new)
 options = {}
 if config =~ /\.ru$/
 cfgfile = ::File.read(config)
 if cfgfile[/^#\\(.*)/] && opts
 options = opts.parse! $1.split(/\s+/)
 end
 cfgfile.sub!(/^__END__\n.*\Z/m, '')
 app = new_from_string cfgfile, config
 else

```

```

 require config
 app = Object.const_get(::File.basename(config, '.rb')).capitalize)
end
return app, options
end

def self.new_from_string(builder_script, file="(rackup)")
 eval "Rack::Builder.new {\n" + builder_script + "\n}.to_app",
 TOPLEVEL_BINDING, file, 0
end

def initialize(default_app = nil, &block)
 @use, @map, @run = [], nil, default_app
 instance_eval(&block) if block_given?
end

def self.app(default_app = nil, &block)
 self.new(default_app, &block).to_app
end

Specifies middleware to use in a stack.
#
class Middleware
def initialize(app)
@app = app
end
#
def call(env)
env["rack.some_header"] = "setting an example"
@app.call(env)
end
end
#
use Middleware
run lambda { |env| [200, { "Content-Type" => "text/plain" }, ["OK"]]}
#
All requests through to this application will first be processed by the middleware class
The +call+ method in this example sets an additional environment key which then can be
referenced in the application if required.
def use(middleware, *args, &block)
 if @map
 mapping, @map = @map, nil
 @use << proc { |app| generate_map app, mapping }
 end
 @use << proc { |app| middleware.new(app, *args, &block) }
end

Takes an argument that is an object that responds to #call and returns a Rack response.
The simplest form of this is a lambda object:
#
run lambda { |env| [200, { "Content-Type" => "text/plain" }, ["OK"]]}
#
However this could also be a class:

```

```

#
class Heartbeat
def self.call(env)
[200, { "Content-Type" => "text/plain" }, ["OK"]]
end
end
#
run Heartbeat
def run(app)
 @run = app
end

Creates a route within the application.
#
Rack::Builder.app do
map '/' do
run Heartbeat
end
end
#
The +use+ method can also be used here to specify middleware to run under a specific path
#
Rack::Builder.app do
map '/' do
use Middleware
run Heartbeat
end
end
#
This example includes a piece of middleware which will run before requests hit +Heartbeat+
#
def map(path, &block)
 @map ||= {}
 @map[path] = block
end

def to_app
 app = @map ? generate_map(@run, @map) : @run
 fail "missing run or map statement" unless app
 @use.reverse.inject(app) { |a,e| e[a] }
end

def call(env)
 to_app.call(env)
end

private

def generate_map(default_app, mapping)
 mapped = default_app ? {'/' => default_app} : {}
 mapping.each { |r,b| mapped[r] = self.class.new(default_app, &b) }
 URLMap.new(mapped)
end

```

```
 end
end
```

## 31.22. Rack::Cascade

Rack::Cascade tries an request on several apps, and returns the first response that is not 404 (or in a list of configurable status codes).

### Ejemplo

```
[~/local/src/ruby/sinatra/rack/rack-cascade]$ cat cascade2.ru
statuses = [200,404]
apps = [
 lambda {|env|
 status = statuses.sample
 [status, {}, ["I'm the first app. Status = #{status}\n"]]
 },
 lambda {|env|
 status = statuses.sample
 [status, {}, ["I'm the second app. Status = #{status}\n"]]
 },
 lambda {|env|
 status = statuses.sample
 [status, {}, ["I'm the last app. Status = #{status}\n"]]
 }
]
use Rack::ContentLength
use Rack::ContentType
run Rack::Cascade.new(apps)
```

```
[~/local/src/ruby/sinatra/rack/rack-cascade]$ rake client
curl -v 'http://localhost:9292'
* About to connect() to localhost port 9292 (#0)
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 33
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
I'm the second app. Status = 200
* Connection #0 to host localhost left intact
* Closing connection #0
```

```
[~/local/src/ruby/sinatra/rack/rack-cascade]$ rake client
curl -v 'http://localhost:9292'
* About to connect() to localhost port 9292 (#0)
* Trying 127.0.0.1... connected
```

```

* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 31
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
I'm the last app. Status = 200
* Connection #0 to host localhost left intact
* Closing connection #0

[~/local/src/ruby/sinatra/rack/rack-cascade]$ rake client
curl -v 'http://localhost:9292'
* About to connect() to localhost port 9292 (#0)
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Content-Length: 32
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
I'm the first app. Status = 200
* Connection #0 to host localhost left intact
* Closing connection #0]

```

## Código del Constructor

```

File lib/rack/cascade.rb, line 11
11: def initialize(apps, catch=404)
12: @apps = []; @has_app = {}
13: apps.each { |app| add app }
14:
15: @catch = {}
16: [*catch].each { |status| @catch[status] = true }
17: end

```

## Código de call

```

File lib/rack/cascade.rb, line 19
19: def call(env)
20: result = NotFound
21:
22: @apps.each do |app|

```

```

23: result = app.call(env)
24: break unless @catch.include?(result[0].to_i)
25: end
26:
27: result
28: end

```

### 31.23. Rack::Mount

1. A *router* is similar to a Rack middleware.
2. The main difference is that it doesn't wrap a single Rack endpoint, but keeps a list of endpoints, just like Rack::Cascade does.
3. Depending on some criteria, usually the requested path, the router will then decide what endpoint to hand the request to.
4. Most routers differ in the way they decide which endpoint to hand the request to.
5. All routers meant for general usage do offer routing based on the path, but how complex their path matching might be varies.
6. While Rack::URLMap only matches prefixes, most other routers allow simple wildcard matching.
7. Both Rack::Mount, which is used by Rails, and Sinatra allow arbitrary matching logic.
8. However, such flexibility comes at a price: Rack::Mount and Sinatra have a routing complexity of O(n), meaning that in the worst-case scenario an incoming request has to be matched against all the defined routes.
9. Rack::Mount is known to produce fast routing, however its API is not meant to be used directly but rather by other libraries, like the Rails routes DSL.

```

[~/local/src/ruby/sinatra/rack/rack-mount]$ cat config.ru
require 'sinatra/base'
require 'rack/mount'

class Foo < Sinatra::Base
 get('/foo') { 'foo' }
 get('/fou') { 'fou' }
end

class Bar < Sinatra::Base
 get('/bar') { 'bar' }
 get('/ba') { 'ba' }
end

Routes = Rack::Mount::RouteSet.new do |set|
 set.add_route Foo, :path_info => %r{^/fo[ou]$}
 set.add_route Bar, :path_info => %r{^/bar?$}
end

run Routes

```

## 31.24. Rack::URLMap

Rack::URLMap takes a hash mapping urls or paths to apps, and dispatches accordingly. Support for HTTP/1.1 host names exists if the URLs start with http:// or https://.

Rack::URLMap modifies the `SCRIPT_NAME` and `PATH_INFO` such that the part relevant for dispatch is in the `SCRIPT_NAME`, and the rest in the `PATH_INFO`. This should be taken care of when you need to reconstruct the URL in order to create links.

Rack::URLMap dispatches in such a way that the longest paths are tried first, since they are most specific.

```
[~/local/src/ruby/sinatra/rack/rack-urlmap(master)]$ cat config.ru
app1 = lambda { |e| [200, {}, ["one\n"]]}
app2 = lambda { |e| [200, {}, ["two\n"]]}
app3 = lambda { |e| [200, {}, ["one + two = three\n"]]}
app = Rack::URLMap.new "/one" => app1, "/two" => app2, "/one/two" => app3
run app

[~/local/src/ruby/sinatra/rack/rack-urlmap(master)]$ cat Rakefile
desc "run the server"
task :default do
 sh "rackup"
end

desc "run the client with one"
task :one do
 sh %q{curl -v 'http://localhost:9292/one'}
end

desc "run the client with two"
task :two do
 sh %q{curl -v 'http://localhost:9292/two'}
end

desc "run the client with one/two"
task :onetwo do
 sh %q{curl -v 'http://localhost:9292/one/two'}
end

[~/local/src/ruby/sinatra/rack/rack-urlmap]$ rake
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
127.0.0.1 - - [17/Oct/2013 21:24:48] "GET /two HTTP/1.1" 200 - 0.0006

[~/local/src/ruby/sinatra/rack/rack-urlmap(master)]$ rake onetwo
curl -v 'http://localhost:9292/one/two'
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /one/two HTTP/1.1
```

```

> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Transfer-Encoding: chunked
< Connection: close
< Server: thin 1.5.1 codename Straight Razor
<
one + two = three
* Closing connection #0

```

### 31.25. El método run de Rack::Handler::WEBrick

Véa por ejemplo una versión del código de `run` de `Rack::Handler::WEBrick`: (puede encontrarse una en `Rack::Handler::WEBrick`):

```

def self.run(app, options={})
 options[:BindAddress] = options.delete(:Host) if options[:Host]
 options[:Port] ||= 8080
 @server = ::WEBrick::HTTPServer.new(options)
 @server.mount "/", Rack::Handler::WEBrick, app
 yield @server if block_given?
 @server.start
end

```

1. Vemos que `run` espera un objeto `app` que representa la aplicación y un hash de opciones.
2. Si arrancamos un servidor en 127.0.0.1, sólo escucha en localhost; si lo arrancamos en 0.0.0.0, escucha a cualquier IP, en particular en nuestra IP local.

Veamos el siguiente experimento:

```

[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ cat bindaddress0000.rb
require 'rack'

#ENV['RACK-ENV'] = 'production'

app = lambda { |e|
 [200, { 'content-type' => 'text/html'}, ["<h1>hello world!</h1>"]]
}

Rack::Handler::WEBrick.run app, { :Host => '0.0.0.0' }

[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ ifconfig en0 | grep 'inet
inet 192.168.0.103

[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ ruby bindaddress0000.rb
[2013-09-23 12:04:36] INFO WEBrick 1.3.1
[2013-09-23 12:04:36] INFO ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 12:04:36] INFO WEBrick::HTTPServer#start: pid=8720 port=8080

```

```
[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ curl -v 'http://192.168.0.103:8080'
* Trying 192.168.0.103... connected
* Connected to 192.168.0.103 (192.168.0.103) port 8080 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8y zlib/1.2.8
> Host: 192.168.0.103:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Server: WEBrick/1.3.1 (Ruby/1.9.3/2013-02-22)
< Date: Mon, 23 Sep 2013 11:11:40 GMT
< Content-Length: 21
< Connection: Keep-Alive
<
* Connection #0 to host 192.168.0.103 left intact
* Closing connection #0
<h1>hello world!</h1>

[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ cat bindaddress127001.rb
require 'rack'

#ENV['RACK-ENV'] = 'production'

app = lambda { |e|
 [200, { 'content-type' => 'text/html'}, ["<h1>hello world!</h1>"]]
}

Rack::Handler::WEBrick.run app, { :Host => '127.0.0.1' }

[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ ruby bindaddress127001.rb
[2013-09-23 12:13:07] INFO WEBrick 1.3.1
[2013-09-23 12:13:07] INFO ruby 1.9.3 (2013-02-22) [x86_64-darwin11.4.2]
[2013-09-23 12:13:07] INFO WEBrick::HTTPServer#start: pid=8993 port=8080

[~/local/src/ruby/sinatra/rack/rack-testing/bindaddress(master)]$ curl -v 'http://192.168.0.103:8080'
* About to connect() to 192.168.0.103 port 8080 (#0)
* Trying 192.168.0.103... Connection refused
* couldn't connect to host
* Closing connection #0
curl: (7) couldn't connect to host
```

3. Luego se crea un nuevo objeto que representa al servidor con `@server = ::WEBrick::HTTPServer.new(options)`. Esto crea un nuevo objeto WEBrick HTTP server de acuerdo a `options`. Un servidor HTTP tiene los siguientes atributos:

- `AccessLog`: An array of access logs. See `WEBrick::AccessLog`
- `BindAddress`: Local address for the server to bind to
- `DocumentRoot`: Root path to serve files from
- `DocumentRootOptions`: Options for the default `HTTPServlet::FileHandler`
- `HTTPVersion`: The HTTP version of this server
- `Port`: Port to listen on

- g) `RequestCallback`: Called with a request and response before each request is serviced.
  - h) `RequestTimeout`: Maximum time to wait between requests
  - i) `ServerAlias`: Array of alternate names for this server for virtual hosting
  - j) `ServerName`: Name for this server for virtual hosting
4. `mount` recibe un directorio y un servlet. Un servlet es una clase que se usa para extender las capacidades de un servidor. En este caso estamos extendiendo `@server` que es un servidor `WEBrick::HTTPServer` con las capacidades definidas en la clase `Rack::Handler::WEBrick`. La sintaxis de `mount` es:
- ```
mount(dir, servlet, *options)
```
- Las opciones son pasadas al servlet en el momento de la creación del servlet.
5. Observamos que `run` puede ir seguido de un bloque al que se le pasa como argumento el objeto `server`
- ```
yield @server if block_given?
```
- Este bloque puede ser usado como una nueva oportunidad para configurar el server
6. Se arranca el servidor con la llamada al método `start` definido en `webrick/server.rb`

## 31.26. Documentación

- rack documentación

## 31.27. Pruebas/Testing

### 31.27.1. Pruebas Unitarias

1. Los fuentes de este ejemplo están en <https://github.com/crguezl/rack-unit-test>
2. Fuentes en GitHub de Rack::Test: <https://github.com/brynary/rack-test>

```
[~/rack/rack-unit-test(master)]$ cat rack_hello_world.rb
my_app.rb
#
require 'rack'

class MyApp
 def call env
 [200, {"Content-Type" => "text/html"}, ["Hello"]]
 end
end

[~/rack/rack-unit-test(master)]$ cat test_hello_world.rb
require "test/unit"
require "rack/test"
require './rack_hello_world'

class AppTest < Test::Unit::TestCase
 include Rack::Test::Methods
```

```

def app
 Rack::Builder.new do
 run MyApp.new
 end.to_app
end

def test_index
 get "/"
 #puts last_response.inspect
 assert last_response.ok?
end

def test_body
 get "/"
 assert_equal last_response.body, 'Hello', "body must be hello"
end
end

```

1. The `Rack::Test::Methods` module serves as the primary integration point for using `Rack::Test` in a testing environment.

It depends on an `app` method being defined in the same context,

```

def app
 Rack::Builder.new do
 run MyApp.new
 end.to_app
end

```

and provides the `Rack::Test` API methods (see `Rack::Test::Session` for their documentation).

2. The `get` method issue a GET request for the given URI. Stores the issues request object in `#last_request` and the app's response in `#last_response` (whose class is `Rack::MockResponse`)

Yield `#last_response` to a block if given.

```

def test_index
 get "/"
 assert last_response.ok?
end

```

3. Otros métodos que se pueden usar son:

- a) (`Object`) `basic_authorize(username, password)` (`also: #authorize`) Set the username and password for HTTP Basic authorization, to be included in subsequent requests in the `HTTP_AUTHORIZATION` header.
- b) (`Object`) `delete(uri, params = {}, env = {}, &block)` Issue a DELETE request for the given URI.
- c) (`Object`) `digest_authorize(username, password)` Set the username and password for HTTP Digest authorization, to be included in subsequent requests in the `HTTP_AUTHORIZATION` header.
- d) (`Object`) `env(name, value)` Set an env var to be included on all subsequent requests through the session.
- e) (`Object`) `follow_redirect!` Rack::Test will not follow any redirects automatically.

- f) (Object) get(uri, params = {}, env = {}, &block) Issue a GET request for the given URI with the given params and Rack environment.
- g) (Object) head(uri, params = {}, env = {}, &block) Issue a HEAD request for the given URI.
- h) (Object) header(name, value) Set a header to be included on all subsequent requests through the session.
- i) (Session) initialize(mock\_session) constructor Creates a Rack::Test::Session for a given Rack app or Rack::MockSession.
- j) (Object) options(uri, params = {}, env = {}, &block) Issue an OPTIONS request for the given URI.
- k) (Object) patch(uri, params = {}, env = {}, &block) Issue a PATCH request for the given URI.
- l) (Object) post(uri, params = {}, env = {}, &block) Issue a POST request for the given URI.
- m) (Object) put(uri, params = {}, env = {}, &block) Issue a PUT request for the given URI.
- n) (Object) request(uri, env = {}, &block) Issue a request to the Rack app for the given URI and optional Rack environment.

4. The `#last_response` object has methods:

```
=~(other) body() empty?() match(other)
```

and attributes:

<code>errors</code>	[RW]
<code>original_headers</code>	[R]
<code>Headers</code>	

5. Si se usan middleware adicionales es necesario especificarlo en `app`:

```
def app
 Rack::Builder.new do
 use(Rack::Session::Cookie, { :key => 'rack session',
 #:domain => 'localhost',
 #:path => '/',
 #:expire_after => 2592000,
 :secret => 'change_me' })
 run RockPaperScissors::App.new
 end.to_app
end
```

6. El método `last_response.body` returns the last response received in the session. Raises an error if no requests have been sent yet.

```
[~/rack/rack-unit-test(master)]$ cat Rakefile
task :default => :test
desc "run the tests"
task :test do
 sh "ruby test_hello_world.rb"
end
```

```
[~/rack/rack-unit-test(master)]$ cat Gemfile
source 'https://rubygems.org'

gem 'rack'
gem 'rack-test'

[~/rack/rack-unit-test(master)]$ rake
ruby test_hello_world.rb
Run options:

Running tests:
.

Finished tests in 0.015253s, 131.1217 tests/s, 131.1217 assertions/s.

2 tests, 2 assertions, 0 failures, 0 errors, 0 skips
```

### 31.27.2. Rspec con Rack

Véase

1. Los fuentes de este ejemplo están en: <https://github.com/crguezl/rack-rspec>
2. Using RSpec with Rack en Youtube por Mike Bethany
3. Documentación en rubydoc.info del módulo Rack::MockSession <http://rdoc.info/github/brynary/rack-test/master/Rack/MockSession>
4. Código fuente en lib/rack/mock.rb
5. Documentación en rubydoc.info del módulo Rack::Test::Methods: <http://rdoc.info/github/brynary/rack-test/master/Rack/Test/Methods>
6. Documentación de Rack::Test::Session
7. webmock gem
8. Class: Rack::MockRequest documentation
9. How to Test Sinatra-Based Web Services by Harlow Ward, March 17, 2013 Webmock Written by thoughtbot Harlow Ward March 17, 2013

### Jerarquía

```
[~/rack/rack-rspec(master)]$ tree
.
|--- Gemfile
|--- Gemfile.lock
|--- README
|--- Rakefile
|--- lib
| |--- rsack
| | '--- server.rb
| '--- rsack.rb
'--- spec
 |--- rsack
```

```
| '-- server_spec.rb
`--- spec_helper.rb
```

4 directories, 8 files

### lib/rsack.rb

```
[~/rack/rack-rspec(master)]$ cat lib/rsack.rb
require 'rack'
require 'rsack/server'
```

### lib/rsack/server.rb

```
[~/rack/rack-rspec(master)]$ cat lib/rsack/server.rb
module Rsack
 class Server
 def call(env)
 #["200", {}, "hello"]
 response = Rack::Response.new
 response.write("Hello world!")
 response.finish
 end
 end
end
```

### spec/rsack/server\_spec.rb

```
[~/rack/rack-rspec(master)]$ cat spec/rsack/server_spec.rb
require 'spec_helper'

describe Rsack::Server do

 #let(:server) { Rack::MockRequest.new(Rsack::Server.new) }
 def server
 Rack::MockRequest.new(Rsack::Server.new)
 end

 context '/' do
 it "should return a 200 code" do
 response = server.get('/')
 response.status.should == 200
 end
 end
end
```

Rack::MockRequest helps testing your Rack application without actually using HTTP.

```
Rack::MockRequest.new(Rsack::Server.new)
```

After performing a request on a URL `response = server.get('/')` with `get/post/put/patch/delete`, it returns a MockResponse with useful helper methods for effective testing (Véase el código de MockResponse en Github en el fichero lib/rack/mock.rb).

Un objeto MockResponse dispone de los métodos:

```
=~ [] match new
```

y de los atributos:

```
body [R] Body
errors [RW] Errors
headers [R] Headers
original_headers [R] Headers
status [R] Status
```

Si se usan middleware adicionales es necesario especificarlo en `server`. Por ejemplo:

```
Rack::MockRequest.new(Rack::Session::Cookie.new(RockPaperScissors::App.new,
 :secret =>'cookie'))
```

#### spec/spec\_helper.rb

```
[~/rack/rack-rspec(master)]$ cat spec/spec_helper.rb
$:.unshift File.expand_path(File.dirname(__FILE__)+'../lib')
$:.unshift File.dirname(__FILE__)

#puts $:.inspect

require 'rspec'
require 'rack'

require 'rsack'
```

#### Rakefile

```
[~/rack/rack-rspec(master)]$ cat Rakefile
desc "run rspec tests"
task :default do
 sh "rspec spec/rsack/server_spec.rb"
end
```

#### Gemfile

```
[~/rack/rack-rspec(master)]$ cat Gemfile
A sample Gemfile
source "https://rubygems.org"

gem 'rack'

group :development, :test do
 gem 'rspec'
end
```

## 31.28. Práctica: Añada Pruebas a Rock, Paper,Scissors

Complete la practica realizada en la sección *Añada Hojas de Estilo a Piedra Papel Tijeras* 31.17.3 con:

1. Pruebas unitarias (Vea la sección *Pruebas Unitarias* 31.27.1)
2. *Desarrollo Dirigido por las Pruebas TDD* (Vea la sección *Rspec con Rack* 31.27.2)
3. Cree una sesión de manera que la aplicación disponga de contadores que lleven el número de partidas jugadas y el número de partidas ganadas por el jugador (Vea las secciones *Gestión de Sesiones* 31.9 y *Cookies* 31.8)

## 31.29. Prácticas: Centro de Cálculo

1. Modo de trabajo en el sistema de archivos del CC de la ETSII
2. Ubicación de las salas

## 31.30. Despliegue de una Aplicación Web en la ETSII

Para desplegar una aplicación web usaremos `exthost2` (en 2013).

Veamos que puertos están libres usando `netstat`:

```
casiano@exthost2:~$ netstat -an | less
```

o bien usamos `lsof`:

```
lsof -i | less
```

y después - si es necesario - terminamos el proceso que ya estuviera escuchando en el puerto

```
kill -9 PID
```

Veamos una simple aplicación usando `rack`:

```
casiano@exthost2:~/src/ruby/simplewebapp$ cat hello.rb
require 'rack'
```

```
app = lambda { |env| [200, {"Content-Type" => "text/plain"}, ["Hello. The time is #{Time.now}"]]
Rack::Handler::WEBrick.run app, :Port => 4567
```

La ejecutamos:

```
casiano@exthost2:~/src/ruby/simplewebapp$ ruby hello.rb
[2013-10-28 09:58:54] INFO WEBrick 1.3.1
[2013-10-28 09:58:54] INFO ruby 1.9.3 (2011-10-30) [i686-linux]
[2013-10-28 09:58:54] WARN TCPServer Error: Address already in use - bind(2)
[2013-10-28 09:58:54] INFO WEBrick::HTTPServer#start: pid=16597 port=4567
```

Ya tenemos disponible la página en `exthost2` en el puerto correspondiente.

El acceso al servidor está limitado a la red de la ULL.

Véase también

1. *Gemas instaladas en local 99.3*

## 31.31. Práctica: Despliegue en Heroku su Aplicación Rock, Paper,Scissors

Despliegue en Heroku la practica realizada en la sección *Añada Pruebas a Rock, Paper,Scissors*  
31.28. Repase la sección *Despliegue en Heroku* 52

## 31.32. Faking Sinatra with Rack and Middleware

1. Faking Sinatra with Rack and Middleware por Charles Max Wood (Vimeo)
2. crguezl/rack-sinatra-in-5-minutes en GitHub
3. Noah Gibbs Ruby Hangout

### 31.33. Véase También

- A Quick Introduction to Rack
- Writing modular web applications with Rack
- Rackup Wiki
- Rack from the Beginning por Adam Hawkins (github: <https://github.com/crguezl/rack-from-the-beginning>)
- Understanding Rack de Tekpub Productions (Vimeo)
- Media Test: Rack Middleware on Any Framework por Noah Gibbs (YouTube)
- Rails Online Conf: Rack in Rails 3 por Ryan Tomayko (Youtube)
- 32 Rack Resources to Get You Started por Jason Seifer
- The Little Rack Book
- Rack Developer's Notebook
- Rails Conf 2013 You've got a Sinatra on your Rails by José Valim
- The Web Server Gateway Interface is a simple and universal interface between web servers and web applications or frameworks for the Python programming language. Rack is inspired in WSGI

# Capítulo 32

## Primeros Pasos

### 32.1. Introducción

#### 32.1.1. Referencias sobre Sinatra

- Ruby/Sinatra Class Page
- Sinatra introduction de Ben Schwarz
- How to create a Twilio app on Heroku de Morten Baga
- ArrrrCamp #6 - Aleksander Dabrowski - Sinatra autopsy Vimeo

Referencias sobre Rack:

- Rackup Wiki
- Understanding Rack de Tekpub Productions

#### 32.1.2. Ejercicio: Instale la Documentación en sinatra.github.com

Instale la documentación de Sinatra en <https://github.com/sinatra/sinatra.github.com>

A la hora de empezar la jerarquía de una aplicación sinatra se puede seguir la estructura propuesta por Lee Martin en sinatra-stack

# Capítulo 33

## Fundamentos

### 33.1. Ejemplo Simple de uso de Sinatra

#### Donde

- [~/sinatra/sinatra-simple(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-simple
- [~/sinatra/sinatra-simple(master)]\$ git remote -v  
origin git@github.com:crguezl/sinatra-simple.git (fetch)  
origin git@github.com:crguezl/sinatra-simple.git (push)
- <https://github.com/crguezl/sinatra-simple>

#### Código

```
[~/sinatra/sinatra-simple(master)]$ cat hi.rb
require 'sinatra'

get '/hi' do
 "Hello World!"
end
```

#### Opciones de Ejecución

```
[~/sinatra/sinatra-simple(master)]$ ruby hi.rb --help
Usage: hi [options]
 -p port set the port (default is 4567)
 -o addr set the host (default is localhost)
 -e env set the environment (default is development)
 -s server specify rack server/handler (default is thin)
 -x turn on the mutex lock (default is off)
```

Sinatra can be used in threaded environments where more than a single request is processed at a time.

However, not all applications and libraries are thread-safe and may cause intermittent errors or general weirdness.

Enabling the `-x` setting causes all requests to synchronize on a mutex lock, ensuring that only a single request is processed at a time.

The mutex lock setting is disabled by default.

### 33.2. Rutas/Routes

Repase la sección *HTTP 31.5*.

Vease el código de este ejemplo en GitHub

## Aplicacion

```
[~/Dropbox/src/ruby/sinatra/sinatra-simple(master)]$ cat app.rb
require 'sinatra/base'

class App < Sinatra::Base
 get '/' do
 "hello get!"
 end

 post '/' do
 'hello post!'
 end

 put '/' do
 'hello put!'
 end

 delete '/' do
 'hello delete!'
 end

 get '/:name' do |name|
 "hello #{name}!"
 end

 get '/:name/?:apellido1?' do |name, apellido|
 "hello #{apellido}, #{name}!"
 end
end
```

In Sinatra, a route is an HTTP method paired with a URL-matching pattern. Each route is associated with a block

Routes are matched in the order they are defined. The first route that matches the request is invoked.

Route patterns may include named parameters, accessible via the `params` hash:

```
get '/hello/:name' do
 # matches "GET /hello/foo" and "GET /hello/bar"
 # params[:name] is 'foo' or 'bar'
 "Hello #{params[:name]}!"
end
```

You can also access named parameters via block parameters:

```
get '/:name' do |name|
 "hello #{name}!"
end
```

Route patterns may also include splat (or wildcard) parameters, accessible via the `params[:splat]` array:

```
get '/say/*/to/*' do
 # matches /say/hello/to/world
 params[:splat] # => ["hello", "world"]
```

```

 end

 get '/download/*.*' do
 # matches /download/path/to/file.xml
 params[:splat] # => ["path/to/file", "xml"]
 end

```

## config.ru

```
[~/sinatra/sinatra-simple(master)]$ cat config.ru
require './app'
```

run App

## Rakefile

```
[~/sinatra/sinatra-simple(master)]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
 sh "rackup"
end

desc "make a get / request via curl"
task :get do
 sh "curl -v localhost:9292"
end

desc "make a post / request via curl"
task :post do
 sh "curl -X POST -v -d 'ignored data' localhost:9292"
end

desc "make a put / request via curl"
task :put do
 sh "curl -X PUT -v localhost:9292"
end

desc "make a DELETE / request via curl"
task :delete do
 sh "curl -X DELETE -v localhost:9292"
end

desc "make a get /name request via curl"
task :getname, :name do |t,h|
 name = h[:name] or 'pepe'
 sh "curl -v localhost:9292/#{name}"
end

desc "make a get /name/apellido request via curl"
task :getfullname, :name, :apellido do |t,h|
 name = h[:name] or 'pepe'
 apellido = h[:apellido] or 'rodriguez'
```

```

sh "curl -v localhost:9292/#{name}/#{apellido}"
end

task :html do
 sh "kramdown README.md > README.html"
end

```

## Ejecución del servidor

```
[~/sinatra/sinatra-simple(master)]$ rake server
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
127.0.0.1 - - [01/Jul/2013 20:25:16] "GET /juana HTTP/1.1" 200 12 0.0689
```

## Ejecución de los clientes

```
[~/Dropbox/src/ruby/sinatra/sinatra-simple(master)]$ rake getname[juana]
{:name=>"juana"}
curl -v localhost:9292/juana
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /juana HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 12
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
* Connection #0 to host localhost left intact
* Closing connection #0
hello juana!
```

```
[~/Dropbox/src/ruby/sinatra/sinatra-simple(master)]$ rake getfullname[Ana,Hernandez]
curl -v localhost:9292/Ana/Hernandez
* About to connect() to localhost port 9292 (#0)
* Trying ::1... Connection refused
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /Ana/Hernandez HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
```

```

< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 21
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
* Connection #0 to host localhost left intact
* Closing connection #0
hello Hernandez, Ana!

```

### 33.2.1. Verbos HTTP en Sinatra/Base

Método get

```

def get(path, opts = {}, &block)
 conditions = @conditions.dup
 route('GET', path, opts, &block)

 @conditions = conditions
 route('HEAD', path, opts, &block)
end

def put(path, opts = {}, &bk) route 'PUT', path, opts, &bk end
def post(path, opts = {}, &bk) route 'POST', path, opts, &bk end
def delete(path, opts = {}, &bk) route 'DELETE', path, opts, &bk end
def head(path, opts = {}, &bk) route 'HEAD', path, opts, &bk end
def options(path, opts = {}, &bk) route 'OPTIONS', path, opts, &bk end
def patch(path, opts = {}, &bk) route 'PATCH', path, opts, &bk end
def link(path, opts = {}, &bk) route 'LINK', path, opts, &bk end
def unlink(path, opts = {}, &bk) route 'UNLINK', path, opts, &bk end

```

Método route

```

def route(verb, path, options = {}, &block)
 # Because of self.options.host
 host_name(options.delete(:host)) if options.key?(:host)
 enable :empty_path_info if path == "" and empty_path_info.nil?
 signature = compile!(verb, path, block, options)
 (@routes[verb] ||= []) << signature
 invoke_hook(:route_added, verb, path, block)
 signature
end

```

### 33.3. Ficheros Estáticos

1. Static files are served from the ./public directory.
2. You can specify a different location by setting the :public\_folder option:

```
set :public_folder, File.dirname(__FILE__) + '/static'
```

Put this code in a `configure` block

3. Note that the public directory name is not included in the URL.
4. A file `./public/css/style.css` is made available as `http://example.com/css/style.css`.
5. Use the `:static_cache_control` setting to add Cache-Control header info. Use an explicit array when setting multiple values:

```
set :static_cache_control, [:public, :max_age => 300]
```

6. What would be delivered in the event that a defined route conflicts with the name of the static resource?. The answer is the static resource.

### 33.4. Vistas

Writing a program that spits out HTML is often more difficult than you might imagine. Although programming languages are better at creating text than they used to be (some of us remember character handling in Fortran and standard Pascal), creating and concatenating string constructs is still painful. If there isn't much to do, it isn't too bad, but a whole HTML page is a lot of text manipulation.

With static HTML pages - those that don't change from request to request - you can use nice WYSIWYG editors. Even those of us who like raw text editors find it easier to just type in the text and tags rather than fiddle with string concatenation in a programming language.

Of course the issue is with dynamic Web pages - those that take the results of something like database queries and embed them into the HTML. The page looks different with each result, and as a result regular HTML editors aren't up to the job.

The best way to work is to compose the dynamic Web page as you do a static page but put in markers that can be resolved into calls to gather dynamic information. Since the static part of the page acts as a template for the particular response, I call this a *Template View*.

Martin Fowler

Views in Sinatra are *HTML templates* that can optionally contain data passed from the application. There are two ways to work with views in Sinatra: *inline templates* and *external templates*. Véase en GitHub [sinatra-up-and-running/tree/master/chapter2/views](https://github.com/sinatra/sinatra-up-and-running/tree/master/chapter2/views).

#### 33.4.1. Templates Inline

Templates may be defined at the end of the source file. En este ejemplo trabajamos con varios templates inline en diferentes ficheros:

```
[~/sinatra/sinatraupandrunning/chapter2/views(master)]$ cat example2-14.rb
require 'sinatra/base'

class App < Sinatra::Base
 enable :inline_templates
 get '/index' do
 puts "Visiting #{request.url}"
 erb :index
 end
end

require './another'
```

```

__END__
@@index
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Inline template</title>
 </head>
 <body>
 <h1>Worked!</h1>
 </body>
</html>

```

En este fichero tenemos un segundo template inline:

```

[~/sinatra/sinatraupandrunning/chapter2/views(master)]$ cat another.rb
class App
 enable :inline_templates
 get '/' do
 erb :another
 end
end

__END__
@@another
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Separated file</title>
 </head>
 <body>
 <h1>Inside another!</h1>
 </body>
</html>

```

Este es nuestro config.ru:

```
[~/sinatra/sinatraupandrunning/chapter2/views(master)]$ cat config.ru
require './example2-14'
```

```
run App
```

Para simplificar las cosas hemos hecho un Rakefile:

```
[~/sinatra/sinatraupandrunning/chapter2/views(master)]$ cat Rakefile
task :default => :server

desc "run server"
task :server do
 sh "rakeup"
end

desc "make a get / request via curl"
task :root do
```

```

sh "curl -v localhost:9292"
end

desc "make a get /index request via curl"
task :index do
 sh "curl -v localhost:9292/index"
end

```

El resultado de la ejecución es:

```
[~/sinatra/sinatra-up-and-running/chapter2/views/inline_templates(master)]$ curl http://localhost:9292/index
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Inline template</title>
 </head>
 <body>
 <h1>Worked!</h1>
 </body>
</html>
[~/sinatra/sinatra-up-and-running/chapter2/views/inline_templates(master)]$ curl http://localhost:9292/another
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Separated file</title>
 </head>
 <body>
 <h1>Inside another!</h1>
 </body>
</html>
```

### 33.4.2. Named Templates

Templates may also be defined using the top-level `template` method:

```

template :layout do
 "%html\n =yield\n"
end

template :index do
 '%div.title Hello World!'
end

get '/' do
 haml :index
end

```

If a template named `layout` exists, it will be used each time a template is rendered.

You can individually disable layouts by passing `:layout => false` or disable them by default via `set :haml, :layout => false`:

```

get '/' do
 haml :index, :layout => !request.xhr?
end

```

### 33.4.3. Templates Externos

```
$ ls
Rakefile example2-16.rb views
config.ru

$ cat example2-16.rb
require 'sinatra/base'

class App < Sinatra::Base
 get '/index' do
 puts "Visiting #{request.url}"
 erb :index
 end
end

$ cat views/index.erb
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Inline template</title>
 </head>
 <body>
 <h1>Worked!</h1>
 </body>
</html>

$ cat config.ru
require './example2-16'

run App

$ cat Rakefile
task :default => :server

desc "run server"
task :server do
 sh "rakeup"
end

desc "make a get / request via curl"
task :root do
 sh "curl -v localhost:9292"
end

desc "make a get /index request via curl"
task :index do
 sh "curl -v localhost:9292/index"
end

$ rake server
rakeup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
```

```

>> Listening on 0.0.0.0:9292, CTRL+C to stop
Visiting http://localhost:9292/index
127.0.0.1 - - [03/Jul/2013 22:30:16] "GET /index HTTP/1.1" 200 157 0.0774

$ rake index
curl -v localhost:9292/index
* About to connect() to localhost port 9292 (#0)
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /index HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 157
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Inline template</title>
 </head>
 <body>
 <h1>Worked!</h1>
 </body>
</html>

* Connection #0 to host localhost left intact
* Closing connection #0

```

### 33.4.4. Templates Externos en Subcarpetas

Véase en GitHub [sinatra-up-and-running/tree/master/chapter2/views/external\\_view\\_files/external\\_in\\_subfolders](https://github.com/sinatra-up-and-running/tree/master/chapter2/views/external_view_files/external_in_subfolders)

```

$ ls
Rakefile app.rb config.ru views

$ cat app.rb
require 'sinatra/base'

class App < Sinatra::Base
 get '/:user/profile' do |user|
 @user = user
 erb '/user/profile'.to_sym
 end

 get '/:user/help' do |user|

```

```

@user = user
erb :'/user/help'
end
end

$ cat views/user/profile.erb
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Profile Template</title>
</head>
<body>
<h1>Profile of <%= @user %></h1>
<%= params %>
</body>
</html>

$ cat views/user/help.erb
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>HELP Template</title>
</head>
<body>
<h1>Help for user <%= @user %></h1>
<pre>
<%= params %>
</pre>
</body>
</html>

$ cat config.ru
require './app'

run App

$ cat Rakefile
PORT = 9292
task :default => :server

desc "run server"
task :server do
 sh "rakeup"
end

desc "make a get /pepe/profile request via curl"
task :profile, :name do |t, h|
 user = h['name'] || 'pepe'
 sh "curl -v localhost:#{PORT}/#{user}/profile"
end

desc "make a get /pepe/help request via curl"

```

```

task :help do
 sh "curl -v localhost:#{PORT}/pepe/help"
end

$ rake server
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
127.0.0.1 - - [03/Jul/2013 21:40:04] "GET /Pedro/profile HTTP/1.1" 200 227 0.1077

$ rake profile[Pedro]
curl -v localhost:9292/Pedro/profile
* About to connect() to localhost port 9292 (#0)
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /Pedro/profile HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 227
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Profile Template</title>
 </head>
 <body>
 <h1>Profile of Pedro</h1>
 {"splat"=>[], "captures"=>["Pedro"], "user"=>"Pedro"}
 </body>
</html>

* Connection #0 to host localhost left intact
* Closing connection #0

```

### 33.4.5. Variables en las Vistas

**Comunicación vía variables de instancia** Los templates se evalúan en el mismo contexto que los manejadores de las rutas. Las variables de instancia son accesibles directamente en los templates.

```

get '/:id' do
 @foo = Foo.find(params[:id])
 haml '%h1= @foo.name'
end

```

Veamos un ejemplo de comunicación via variables de instancia entre el manejador de la ruta y el template:

```
[~/sinatra/sinatra-views/passing_data_into_views(master)]$ ls
Rakefile config.ru via_instance.rb
```

```
[~/sinatra/sinatra-views/passing_data_into_views(master)]$ cat via_instance.rb
require 'sinatra/base'
```

```
class App < Sinatra::Base
 get '/*' do |name|
 def some_template
 <<-HAMLTEMP
%ol
- @foo.each do |item|
 %li
 %i #{item}
HAMLTEMP
 end

 puts "----**#{name}**----"
 @foo = name.split('/')
 haml some_template
 end
end
```

```
[~/sinatra/sinatra-views/passing_data_into_views(master)]$ cat config.ru
require './via_instance'
```

```
run App
```

```
[~/sinatra/sinatra-views/passing_data_into_views(master)]$ cat Rakefile
task :default => :server
```

```
desc "run server"
task :server do
 sh "rackup"
end
```

```
desc "make a get /juan/leon/hernandez request via curl"
task :client do
 sh "curl -v localhost:9292/juan/leon/hernandez"
end
```

```
[~/sinatra/sinatraupandrunning/chapter2/views/passing_data_into_views(master)]$ rake server
rackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
----**juan/leon/hernandez**----*
127.0.0.1 - - [05/Jul/2013 17:06:05] "GET /juan/leon/hernandez HTTP/1.1" 200 109 0.3502
```

```
[~/sinatra/sinatra-views/passing_data_into_views(master)]$ rake client
curl -v localhost:9292/juan/leon/hernandez
```

```

* About to connect() to localhost port 9292 (#0)
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /juan/leon/hernandez HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 109
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<

 <i>juan</i>

 <i>leon</i>

 <i>hernandez</i>

* Connection #0 to host localhost left intact
* Closing connection #0

```

### 33.4.6. Pasando variables a la vista explícitamente via un hash

También es posible pasar en la llamada un hash especificando las variables locales:

```

get '/:id' do
 foo = Foo.find(params[:id])
 haml '%h1= bar.name', :locals => { :bar => foo }
end

```

This is typically used when rendering templates as partials from within other templates.

Veamos un ejemplo:

```

$ ls
Rakefile config.ru via_hash.rb views

$ cat via_hash.rb
require 'sinatra/base'

class App < Sinatra::Base
 get '/*' do |name|
 def some_template
 <<-ERBTEMP
<% name.each do |item| %>
 <i> <%= item %> </i>

```

```

 <% end %>

ERBTEMP
 end # method some_template

 puts "----**#{name}----**"
 erb some_template, :locals => { :name => name.split('/') }
end
end

$ cat views/layout.erb
<!DOCTYPE html>
<html>
 <head>
 <title>Sinatra</title>
 </head>
 <body>
 <h1>Accesing variables in templates via a parameter hash</h1>
 <%= yield %>
 </body>
</html>

$ cat config.ru
require './via_hash'

run App

$ cat Rakefile task :default => :server

desc "run server"
task :server do
 sh "rackup"
end

desc "make a get /juan/leon/hernandez request via curl"
task :client do
 sh "curl -v localhost:9292/juan/leon/hernandez"
end

$ rake serverrackup
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:9292, CTRL+C to stop
----**juan/leon/hernandez----**
127.0.0.1 - - [05/Jul/2013 17:50:20] "GET /juan/leon/hernandez HTTP/1.1" 200 290 0.0352

$ rake client
curl -v localhost:9292/juan/leon/hernandez
* About to connect() to localhost port 9292 (#0)
* Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9292 (#0)
> GET /juan/leon/hernandez HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4 OpenSSL/0.9.8x zlib/1.2.
> Host: localhost:9292

```

```

> Accept: /*
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 290
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
< Server: thin 1.5.1 codename Straight Razor
<
<!DOCTYPE html>
<html>
 <head>
 <title>Sinatra</title>
 </head>
 <body>
 <h1>Accesing variables in templates via a parameter hash</h1>

 <i> juan </i>

 <i> leon </i>

 <i> hernandez </i>

 </body>
</html>
* Connection #0 to host localhost left intact
* Closing connection #0

```

### 33.4.7. Opciones pasadas a los Métodos de los Templates

Options passed to the render method override options set via `set`.

Available Options:

#### 1. `locals`

List of locals passed to the document. Handy with partials. Example:

```
erb "<%= foo %>", :locals => { :foo => "bar" }
```

#### 2. `default_encoding`

String encoding to use if uncertain. Defaults to

```
settings.default_encoding.
```

#### 3. `views`

Views folder to load templates from. Defaults to `settings.views`.

#### 4. `layout`

Whether to use a layout (true or false), if it's a Symbol, specifies what template to use. Example:

```
erb :index, :layout => !request.xhr?
```

## 5. content\_type

Content-Type the template produces, default depends on template language.

## 6. scope

Scope to render template under.

Defaults to the application instance.

If you change this, instance variables and helper methods will not be available.

## 7. layout\_engine

Template engine to use for rendering the layout.

Useful for languages that do not support layouts otherwise.

Defaults to the engine used for the template. Example:

```
set :rdoc, :layout_engine => :erb
```

## 8. layout\_options

Special options only used for rendering the layout. Example:

```
set :rdoc, :layout_options => { :views => 'views/layouts' }
```

## 9. Templates are assumed to be located directly under the ./views directory.

To use a different views directory:

```
set :views, settings.root + '/templates'
```

## 10. One important thing to remember is that you always have to reference templates with symbols, even if they're in a subdirectory (in this case, use: :subdir/template or 'subdir/template'.to\_sym).

You must use a symbol because otherwise rendering methods will render any strings passed to them directly.

## 33.5. Filtros

**Before Filters** *Before filters* are evaluated before each request within the same context as the routes will be and can modify the request and response.

Instance variables set in filters are accessible by routes and templates:

```
before do
 @note = 'Hi!'
 request.path_info = '/foo/bar/baz'
end

get '/foo/*' do
 @note #=> 'Hi!'
 params[:splat] #=> 'bar/baz'
end
```

**After Filters** *After filters* are evaluated after each request within the same context and can also modify the request and response.

Instance variables set in before filters and routes are accessible by after filters:

```
after do
 puts response.status
end
```

Note: Unless you use the `body` method rather than just returning a String from the routes, the body will not yet be available in the after filter, since it is generated later on.

**Filters can take a Pattern** Filters optionally take a pattern, causing them to be evaluated only if the request path matches that pattern:

```
before '/protected/*' do
 authenticate!
end

after '/create/:slug' do |slug|
 session[:last_slug] = slug
end
```

**Filters can take a Condition** Like routes, filters also take conditions:

```
before :agent => /Songbird/ do
 # ...
end

after '/blog/*', :host_name => 'example.com' do
 # ...
end
```

### 33.6. Manejo de Errores

1. The HTTP specification states that response status in the range 200-299 indicate success in processing a request
2. 500-599 is reserved for server errors
3. Sinatra offers helpers for the 404 (Not Found) and 500 (Internal Server Error) status

**not\_found** When a `Sinatra::NotFound` exception is raised, or the response's status code is 404, the `not_found` handler is invoked:

```
not_found do
 'This is nowhere to be found.'
end
```

**error** The `error` handler is invoked any time an exception is raised from a route block or a filter. The exception object can be obtained from the `sinatra.error` Rack variable:

```
error do
 'Sorry there was a nasty error - ' + env['sinatra.error'].name
end
```

Custom errors:

```
error MyCustomError do
 'So what happened was...' + env['sinatra.error'].message
end
```

Then, if this happens:

```
get '/' do
 raise MyCustomError, 'something bad'
end
```

You get this:

```
So what happened was... something bad
```

Alternatively, you can install an error handler for a status code:

```
error 403 do
 'Access forbidden'
end

get '/secret' do
 403
end
```

Or a range:

```
error 400..510 do
 'Boom'
end
```

Sinatra installs special `not_found` and `error` handlers when running under the development environment to display nice stack traces and additional debugging information in your browser (esto es, en producción estos handlers son mucho más "parcos").

### 33.7. The methods body, status and headers

1. It is possible and recommended to set the status code and response body with the return value of the route block.
2. However, in some scenarios you might want to set the body at an arbitrary point in the execution flow.
3. You can do so with the `body` helper method.
4. If you do so, you can use that method from there on to access the body:

```
get '/foo' do
 body "bar"
end

after do
 puts body
end
```

It is also possible to pass a block to `body`, which will be executed by the Rack handler (this can be used to implement *streaming*).

5. Similar to the `body`, you can also set the `status` code and `headers`:

```
get '/foo' do
 status 418
 headers \
 "Allow" => "BREW, POST, GET, PROPFIND, WHEN",
 "Refresh" => "Refresh: 20; http://www.ietf.org/rfc/rfc2324.txt"
 body "I'm a tea pot!"
end
```

6. Like `body`, `headers` and `status` with no arguments can be used to access their current values.

### 33.8. Acceso al Objeto Request

El objeto que representa la solicitud *the request object* es un hash con información de la solicitud: quien hizo la petición, que versión de HTTP usar, etc.

El objeto que representa la solicitud puede ser accedido desde el nivel de solicitud: filtros, rutas y manejadores de error.

Véase [https://github.com/crguezl/sinatra\\_intro/blob/master/accesing\\_the\\_request\\_object.rb](https://github.com/crguezl/sinatra_intro/blob/master/accesing_the_request_object.rb)

### 33.9. Caching / Caches

Mediante el uso del helper `headers` podemos establecer los headers que queramos para influir sobre la forma en la que ocurre el caching downstream.

1. Caching Tutorial

### 33.10. Sesiones y Cookies en Sinatra

**Introducción** A session is used to keep state during requests. If activated, you have one session hash per user session:

```
enable :sessions

get '/' do
 "value = " << session[:value].inspect
end

get '/:value' do
 session[:value] = params[:value]
end
```

1. Note that `enable :sessions` actually stores all data in a cookie
2. This might not always be what you want (storing lots of data will increase your traffic, for instance)
3. You can use any Rack session middleware: in order to do so, do not call `enable :sessions`, but instead pull in your middleware of choice as you would any other middleware:

```

use Rack::Session::Pool, :expire_after => 2592000

get '/' do
 "value = " << session[:value].inspect
end

get '/:value' do
 session[:value] = params[:value]
end

```

4. To improve security, the session data in the cookie is signed with a session secret
5. A random secret is generated for you by Sinatra
6. However, since this secret will change with every start of your application, you might want to set the secret yourself, so all your application instances share it:

```
set :session_secret, 'super secret'
```

If you want to configure it further, you may also store a hash with options in the sessions setting:

```
set :sessions, :domain => 'foo.com'
```

7. Just use `session.clear` to destroy the session.

```

get '/login' do
 session[:username] = params[:username]
 "logged in as #{session[:username]}"
end

get '/logout' do
 old_user = session[:username]
 session.clear
 "logged out #{old_user}"
end

```

## Cookies

1. According to the Computer Science definition, a *cookie*, which is also known as an *HTTP cookie*, a *tracking cookie*, or a *browser cookie*, is a piece of text, no bigger than 4 kilobytes, which is stored on the user's computer by a web server via a web browser
2. It is a key-value pair structure, which is designed to retain specific information such as
  - user preferences,
  - user authentication,
  - shopping carts,
  - demographics,
  - sessions, or
  - any other data used by a website
3. This mechanism, which was developed by Netscape in the distant 1994, provides a way to receive information from a web server and to send it back from the web browser absolutely unchanged

4. This system complements the stateless nature of the HTTP protocol as it provides enough memory to store pieces of information during HTTP transactions
5. When you try to access a web site, your web browser connects to a web server and it sends a request for the respective page
6. Then the web server replies by sending the requested content and it simultaneously stores a new cookie on your computer
7. Every time the web browser requests web pages from the web server, it always sends the respective cookies back to the web server
8. The process takes place as described, if the web browser supports cookies and the user allows their usage
9. Only the web server can modify one or more of the cookie values
10. Then it sends them to the web browser upon replying to a specific request
11. According to the RFC2965 specification, cookies are case insensitive
12. A set of defined properties is inherent to the cookie structure Those properties include: an expiration date, a path and a domain
13. The first attribute requires a date defined in `Wdy, DD-Mon-YYYY HH:MM:SS GMT` format
14. The rest of the cookie characteristics require a `path` and/or a `domain` defined as a string
15. Let's take a look at this example:

```
Cookie: key0=value0; ...; keyX=valueX; expires=Wed, 23-Sep-2009 23:59:59 GMT; path=/; domai
```

16. When the `expiration` date is defined, your cookie will be *persistent* as it will reoccur in different sessions until the set `expiration` date has been reached
17. If the `expiration` date has not been defined in the cookie, it will occur until the end of your current session or when you close your web browser
18. If the `path` and/or the `domain` attributes have been defined in your cookie, then the web server limits the scope of the cookie to that specific `domain`, `sub-domain` or `path`

## Ejemplo con Sesiones

```
require 'rubygems'
require 'sinatra'
require 'haml'

enable :sessions

get '/' do
 session["user"] ||= nil
 haml :index
end

get '/introduction' do
 haml :introduction
end
```

```

post '/introduction' do
 session["user"] = params[:name]
 redirect '/'
end

get '/bye' do
 session["user"] = nil
 haml :bye
end

```

## Ejemplo con Cookies

1. The last example will demonstrate how to directly manage cookies through the `request` and `response` singletons provided by Sinatra
2. You will see in the following example that the previously described process involving the use of cookies is clearly implemented
3. This technique is recommended when your application requires to use persistent and/or scoped cookies
4. In this example, the application uses two persistent cookies, which expire at the same time, in order to store and manage different configuration data

```

require 'sinatra'
require 'haml'

get '/' do
 @@expiration_date = Time.now + (60 * 2) \
 unless request.cookies.key?('some_options') && request.cookies.key?('other_options')
 haml :index
end

get '/some_options' do
 @some_cookie = request.cookies["some_options"]
 haml :some_options
end

post '/some_options' do
 response.set_cookie('some_options', :value => cookie_values(params), :expires => @@expiration_date)
 redirect '/'
end

get '/other_options' do
 @other_cookie = request.cookies["other_options"]
 haml :other_options
end

post '/other_options' do
 response.set_cookie('other_options', :value => cookie_values(params), :expires => @@expiration_date)
 redirect '/'
end

helpers do
 def cookie_values(parameters)

```

```

values = {}
parameters.each do |key, value|
 case key
 when 'options'
 values[value] = true
 else
 values[key] = true
 end
end
values
end

```

## Problemas

1. I'm not sure why but my session gets wiped out every request?
2. To keep sessions consistent you need to set a session secret, e.g.:

```
set :session_secret, 'super secret'
```

When it's not set sinatra generates random one on application start and shotgun restarts application before every request.

## Véanse

1. Daily Ruby Tips #60 – Simple Use of Sessions in Sinatra May 6, 2013
2. La sección *Cookies* en Rack 31.8.
3. Cookie-based Sessions in Sinatra by JULIO JAVIER CICCHELLI on SEPTEMBER 30, 2009 RubyLearning Blog. El código está en un Gist en GitHub

## 33.11. Downloads / Descargas / Attachments

### Usando attachment

There is a built-in `attachment` method that optionally takes a filename parameter. If the filename has an extension (.jpg, etc.) that extension will be used to determine the Content-Type header for the response.

The evaluation of the route will provide the contents of the attachment.

1. Documentación de attachment
2. Código de attachment en GitHub
3. Upload and download files in Sinatra Random Ruby Thoughts

```
[~/sinatra/sinatra-download(master)]$ cat app.rb
require 'sinatra'

before do
 content_type :txt
end

get '/attachment?' do
 attachment 'file.txt'
 "Here's what will be sent downstream, in an attachment called 'file.txt'."
end
```

Cuando visitamos la página con el navegador se nos abre una ventana para la descarga de un fichero que será guardado (por defecto) como `file.txt`.

Los contenidos de ese fichero serán:

```
[~/sinatra/sinatra-download(master)]$ cat ~/Downloads/file.txt
Here's what will be sent downstream, in an attachment called 'file.txt'.
```

### Usando `send_file`

Véase la documentación del módulo `Sinatra::Streaming`

```
[~/sinatra/sinatra-download(master)]$ cat sending_file.rb
require 'sinatra'
```

```
get '/' do
 send_file 'foo.png',
 :type => 'img/png',
 :disposition => 'attachment',
 :filename => 'tutu.png',
 :stream => false
end
```

The options are:

1. `filename` file name, in response, defaults to the real file name.
2. `last_modified` value for Last-Modified header, defaults to the file's `mtime`.
3. `type` content type to use, guessed from the file extension if missing.
4. `disposition` used for Content-Disposition, possible value: `nil` (default), `:attachment` and `:inline`
5. `length` Content-Length header, defaults to file size.
6. `status` Status code to be send.

Useful when sending a static file as an error page. If supported by the Rack handler, other means than streaming from the Ruby process will be used. If you use this helper method, Sinatra will automatically handle range requests.

## 33.12. Uploads. Subida de Ficheros en Sinatra

### Véase

1. El repositorio `sinatra-upload` en GitHub con el código de este ejemplo
2. FILE UPLOAD WITH SINATRA BY PANDAFOX POSTED IN RUBY, TUTORIALS
3. Multiple file uploads in Sinatra

### Jerarquía de ficheros

```
[~/sinatra/sinatra-upload]$ tree
.
|-- app.rb
|-- uploads
| '-- README
`-- views
 '-- upload.haml

2 directories, 3 files
```

## upload.haml

The important part is not to forget to set the `enctype` in your form element, otherwise you will just get the filename instead of an object:

```
[~/sinatra/sinatra-upload(master)]$ cat views/upload.haml
%html
 %body
 %h1 File uploader!
 %form(method="post" enctype='multipart/form-data')
 %input(type='file' name='myfile')
 %br
 %input(type='submit' value='Upload!')
```

## app.rb

```
[~/sinatra/sinatra-upload(master)]$ cat app.rb
require 'rubygems'
require 'sinatra'
require 'haml'
require 'pp'

Handle GET-request (Show the upload form)
get "/upload?" do
 haml :upload
end

Handle POST-request (Receive and save the uploaded file)
post "/upload" do
 pp params
 File.open('uploads/' + params['myfile'][filename], "w") do |f|
 f.write(params['myfile'][tempfile].read)
 end
 return "The file was successfully uploaded!"
end
[~/sinatra/sinatra-upload(master)]$
```

As you can see, you don't have to write much code to get this to work. The `params`-hash contains our uploaded element with data such as filename, type and the actual datafile, which can be accessed as a Tempfile-object.

We read the contents from this file and store it into a directory called uploads, which you will have to create before running this script.

Here's an example of what the `params`-hash may look like when uploading a picture of a cat:

```
{
 "myfile" => {
 :type => "image/png",
 :head => "Content-Disposition: form-data;
 name=\"myfile\"";
 :filename=>"cat.png"\r\n
 :Content-Type: image/png\r\n",
 :name => "myfile",
 :tempfile => #<File:/var/folders/3n/3asd/-Tmp-/RackMultipart201-1476-nfw2-0>,
 :filename=>"cat.png"
 }
}
```

## **BEWARE!**

This script offers little to no security at all. Clients will be able to overwrite old images, fill up your harddrive and so on. So just use some common sense and do some Ruby magic to patch up the security holes yourself.

### **33.13. halt**

Sometimes we want to stop the program: maybe a critical error has occurred. To immediately stop a request within a filter or route use:

```
halt
```

You can also specify the status when halting:

```
halt 410
```

Or the body:

```
halt 'this will be the body'
```

Or both:

```
halt 401, 'go away!'
```

With headers:

```
halt 402, {'Content-Type' => 'text/plain'}, 'revenge'
```

It is of course possible to combine a template with halt:

```
halt erb(:error)
```

### **33.14. Passing a Request**

When we want to pass processing to the next matching route we use `pass`:

```
get '/guess/:who' do
 pass unless params[:who] == 'Frank'
 'You got me!'
end

get '/guess/*' do
 'You missed!'
end
```

The route block is immediately exited and control continues with the next matching route.

If no matching route is found, a 404 is returned.

### 33.15. Triggering Another Route: calling call

Sometimes `pass` is not what you want, instead you would like to get the result of calling another route. Simply use `call` to achieve this:

```
get '/foo' do
 status, headers, body = call env.merge("PATH_INFO" => '/bar')
 [status, headers, body.map(&:upcase)]
end

get '/bar' do
 "bar"
end
```

Note that in the example above, you would ease testing and increase performance by simply moving `"bar"` into a helper used by both `/foo` and `/bar`.

If you want the request to be sent to the same application instance rather than a duplicate, use `call!` instead of `call`.

Check out the Rack specification if you want to learn more about `call`.

### 33.16. Logging

In the request scope, the `logger` helper exposes a `Logger` instance:

```
get '/' do
 logger.info "loading data"
 #
end
```

1. This `logger` will automatically take your Rack handler's `logging` settings into account
2. If `logging` is disabled, this method will return a dummy object, so you do not have to worry in your routes and filters about it

Note that `logging` is only enabled for `Sinatra::Application` by default, so if you inherit from , you probably want to enable it yourself:

```
class MyApp < Sinatra::Base
 configure :production, :development do
 enable :logging
 end
end
```

1. To avoid any `logging` middleware to be set up, set the `logging` setting to `nil`
2. However, keep in mind that `logger` will in that case return `nil`
3. A common use case is when you want to set your own `logger`. Sinatra will use whatever it will find in `env['rack.logger']`

#### Logging a stdout y a un fichero

Véase `Rack::CommonLogger` en Sinatra Recipes.

Sinatra has logging support, but it's nearly impossible to log to a file and to the `stdout` (like Rails does).

However, there is a little trick you can use to log to `stdout` and to a file:

```

require 'sinatra'

configure do
 # logging is enabled by default in classic style applications,
 # so 'enable :logging' is not needed
 file = File.new("#{settings.root}/log/#{settings.environment}.log", 'a+')
 file.sync = true
 use Rack::CommonLogger, file
end

get '/' do
 'Hello World'
end

```

You can use the same configuration for modular style applications, but you have to `enable :logging` first:

```

require 'sinatra/base'

class SomeApp < Sinatra::Base
 configure do
 enable :logging
 file = File.new("#{settings.root}/log/#{settings.environment}.log", 'a+')
 file.sync = true
 use Rack::CommonLogger, file
 end

 get '/' do
 'Hello World'
 end

 run!
end

```

## Ejecución

```

~/sinatra/sinatra-logging]$ tree
.
|-- app.rb
`-- log

1 directory, 1 file
[~/sinatra/sinatra-logging]$ ruby app.rb
== Sinatra/1.4.4 has taken the stage on 4567 for development with backup from Thin
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on localhost:4567, CTRL+C to stop

```

Consola después de visitar la página:

```
127.0.0.1 - - [19/Nov/2013 14:53:06] "GET / HTTP/1.1" 200 11 0.0041
```

Fichero después de visitar la página:

```
[~/sinatra/sinatra-logging]$ cat log/development.log
127.0.0.1 - - [19/Nov/2013 14:53:06] "GET / HTTP/1.1" 200 11 0.0038
```

## Véase

1. el código en GitHub de Rack::CommonLogger
2. Logging in Sinatra. StackOverflow. Destination is set by changing `env['rack.errors']`. Kons-tantin Haase May 13 '11 at 21:18'

### 33.17. Generating URLs

For generating URLs you should use the `url` helper method, for instance, in Haml:

```
%a{:href => url('/foo')} foo
```

It takes reverse proxies and Rack routers into account, if present.

This method is also aliased to `to`.

### 33.18. Redireccionamientos/Browser Redirect

You can trigger a browser redirect with the `redirect` helper method:

```
get '/foo' do
 redirect to('/bar')
end
```

Any additional parameters are handled like arguments passed to `halt`:

```
redirect to('/bar'), 303
redirect 'http://google.com', 'wrong place, buddy'
```

You can also easily `redirect` back to the page the user came from with `redirect back`:

```
get '/foo' do
 "do something"
end

get '/bar' do
 do_something
 redirect back
end
```

To pass arguments with a `redirect`, either add them to the query:

```
redirect to('/bar?sum=42')
```

Or use a session:

```
enable :sessions

get '/foo' do
 session[:secret] = 'foo'
 redirect to('/bar')
end

get '/bar' do
 session[:secret]
end
```

### 33.19. Configuration / Configuración

Run once, at startup, in any environment:

```
configure do
 # setting one option
 set :option, 'value'

 # setting multiple options
 set :a => 1, :b => 2

 # same as 'set :option, true'
 enable :option

 # same as 'set :option, false'
 disable :option

 # you can also have dynamic settings with blocks
 set(:css_dir) { File.join(views, 'css') }
end
```

Run only when the environment (`RACK_ENV` environment variable) is set to `:production`:

```
configure :production do
 ...
end
```

Run when the environment is set to either `:production` or `:test`:

```
configure :production, :test do
 ...
end
```

You can access those options via settings:

```
configure do
 set :foo, 'bar'
end

get '/' do
 settings.foo? # => true
 settings.foo # => 'bar'
 ...
end
```

### 33.20. Configuring attack protection

Sinatra is using Rack::Protection to defend your application against common, opportunistic attacks.

1. You can easily disable this behavior (which will open up your application to tons of common vulnerabilities):

```
disable :protection
```

2. To skip a single defense layer, set protection to an options hash:

```
set :protection, :except => :path_traversal
```

3. You can also hand in an array in order to disable a list of protections:

```
set :protection, :except => [:path_traversal, :session_hijacking]
```

4. By default, Sinatra will only set up session based protection if :sessions has been enabled.

Sometimes you want to set up sessions on your own, though.

In that case you can get it to set up session based protections by passing the :session option:

```
use Rack::Session::Pool
set :protection, :session => true
```

### 33.21. Settings disponibles/Available Settings

1. **absolute\_redirects** If disabled, Sinatra will allow relative redirects, however, Sinatra will no longer conform with RFC 2616 (HTTP 1.1), which only allows absolute redirects.

Enable if your app is running behind a reverse proxy that has not been set up properly. Note that the `url` helper will still produce absolute URLs, unless you pass in `false` as the second parameter. Disabled by default.

2. **addCharsets** mime types the `content_type` helper will automatically add the charset info to. You should add to it rather than overriding this option:

```
settings.addCharsets << "application/foobar"
```

3. **app\_file** Path to the main application file, used to detect project root, views and public folder and inline templates.

4. **bind** IP address to bind to (default: 0.0.0.0 or localhost if your `environment` is set to `development`). Only used for built-in server.

5. **default\_encoding** encoding to assume if unknown (defaults to "utf-8").

6. **dump\_errors** display errors in the log.

7. **environment** current environment, defaults to ENV['RACK\_ENV'], or `development` if not available.

8. **logging** use the logger.

9. **lock** Places a lock around every request, only running processing on request per Ruby process concurrently. Enabled if your app is not thread-safe. Disabled per default.

10. **method\_override** use \_method magic to allow put/delete forms in browsers that don't support it.

11. **port** Port to listen on. Only used for built-in server.

12. **prefixed\_redirects** Whether or not to insert `request.script_name` into redirects if no absolute path is given. That way redirect '/foo' would behave like redirect to('/foo'). Disabled per default.

13. **protection** Whether or not to enable web attack protections. See protection section above.

14. **public\_dir** Alias for `public_folder`. See below.

15. `public_folder` Path to the folder public files are served from. Only used if static file serving is enabled (see static setting below). Inferred from `app_file` setting if not set.
16. `reload_templates` Whether or not to reload templates between requests. Enabled in development mode.
17. `root` Path to project root folder. Inferred from `app_file` setting if not set.
18. `raise_errors` raise exceptions (will stop application). Enabled by default when environment is set to "test", disabled otherwise.
19. `run` if enabled, Sinatra will handle starting the web server, do not enable if using rackup or other means.
20. `running` is the built-in server running now? do not change this setting!
21. `server` Server or list of servers to use for built-in server. order indicates priority, default depends on Ruby implementation.
22. `sessions` Enable cookie-based sessions support using Rack::Session::Cookie. See 'Using Sessions' section for more information.
23. `show_exceptions` Show a stack trace in the browser when an exception happens. Enabled by default when environment is set to "development", disabled otherwise. Can also be set to `:after_handler` to trigger app-specified error handling before showing a stack trace in the browser.
24. `static` Whether Sinatra should handle serving static files. Disable when using a server able to do this on its own. Disabling will boost performance. Enabled per default in classic style, disabled for modular apps.
25. `static_cache_control` When Sinatra is serving static files, set this to add Cache-Control headers to the responses. Uses the `cache_control` helper. Disabled by default. Use an explicit array when setting multiple values:

```
set :static_cache_control, [:public, :max_age => 300]
```
26. `threaded` If set to true, will tell Thin to use EventMachine.defer for processing the request.
27. `views` Path to the views folder. Inferred from `app_file` setting if not set.
28. `x_cascade` Whether or not to set the X-Cascade header if no route matches. Defaults to true.

## 33.22. Environments

1. There are three predefined environments: `development`, `production` and `test`.
2. Environments can be set through the `RACK_ENV` environment variable.
3. The default value is `development`
4. In the `development` environment all templates are reloaded between requests, and special `not_found` and `error` handlers display stack traces in your browser
5. In the `production` and `test` environments, templates are cached by default
6. To run different environments, set the `RACK_ENV` environment variable:

```
RACK_ENV=production ruby my_app.rb
```

7. You can use predefined methods: `development?`, `test?` and `production?` to check the current environment setting:

```
get '/' do
 if settings.development?
 "development!"
 else
 "not development!"
 end
end
```

### 33.23. Correo

```
[~/srcSTW/sinatra-faq/mail(esau)]$ cat app.rb
require 'sinatra'
require 'pony'
raise "Execute:\n\t#{$0} password email_to email_from" if ARGV.length.zero?
get '/' do
 email = ARGV.shift
 pass = ARGV.shift
 Pony.mail({
 :to => email,
 :body => "Hello Casiano",
 :subject => 'Howdy, Partna!',
 :via => :smtp,
 :via_options => {
 :address => 'smtp.gmail.com',
 :port => '587',
 :enable_starttls_auto => true,
 :user_name => ARGV.shift,
 :password => pass,
 :authentication => :plain, # :plain, :login, :cram_md5, no auth by default
 :domain => "localhost.localdomain" # the HELO domain provided by the c
 }
 })
 "Check your email at #{email}"
end
```

1. Getting started with Sinatra

### 33.24. Ambito

The scope you are currently in determines what methods and variables are available.

#### Ámbito de Clase/Class Scope

1. Every Sinatra application corresponds to a subclass of `Sinatra::Base`
2. If you are using the top-level DSL (`require 'sinatra'`), then this class is `Sinatra::Application`, otherwise it is the subclass you created explicitly
3. At class level you have methods like `get` or `before`, but you cannot access the `request` or `session` objects, as there is only a single application class for all requests

Options created via `set` are methods at class level:

```
class MyApp < Sinatra::Base
 # Hey, I'm in the application scope!
 set :foo, 42
 foo # => 42

 get '/foo' do
 # Hey, I'm no longer in the application scope!
 end
end
```

You have the application scope binding inside:

1. Your application class body
2. Methods defined by extensions
3. The block passed to `helpers`
4. Procs/blocks used as value for `set`
5. The block passed to `Sinatra.new`

You can reach the scope object (the class) like this:

1. Via the object passed to configure blocks (`configure { |c| ... }`)
2. `settings` from within the request scope

## Ámbito de Instancia/Instance Scope

For every incoming request, a new instance of your application class is created and all handler blocks run in that scope

1. From within this scope you can access the `request` and `session` objects or
2. call rendering methods like `erb` or `haml`
3. You can access the application scope from within the `request` scope via the `settings` helper:

```
[~/sinatra/sinatra-scope]$ cat app.rb
require 'sinatra'

class MyApp < Sinatra::Base
 # Hey, I'm in the application scope!
 get '/define_route/:name' do
 # Request scope for '/define_route/:name'
 @value = 42
 puts "Inside /define_route/:name @value = #{@value}"
 puts self.class

 settings.get("/#{params[:name]}") do
 # Request scope for "/#{params[:name]}"
 puts "@value = <#{@value}>"
 "Inside defined route #{params[:name]}"
 end
 end
end
```

```

"Route #{params[:name]} defined!"
end

run! if __FILE__ == $0
end

```

### Ejecución en el servidor

```
[~/sinatra/sinatra-scope]$ ruby app.rb
== Sinatra/1.4.4 has taken the stage on 4567 for development with backup from Thin
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on localhost:4567, CTRL+C to stop
Inside /define_route/:name @value = 42
MyApp
@value = <>
```

### Ejecución en el cliente. Ruta: /define\_route/juan

```
[~/sinatra/sinatra-scope]$ curl -v 'http://localhost:4567/define_route/juan'
* Adding handle: conn: 0x7fbacb004000
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7fbacb004000) send_pipe: 1, recv_pipe: 0
* About to connect() to localhost port 4567 (#0)
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 4567 (#0)
> GET /define_route/juan HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4567
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 19
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
* Server thin 1.6.1 codename Death Proof is not blacklisted
< Server: thin 1.6.1 codename Death Proof
<
* Connection #0 to host localhost left intact
Route juan defined!
[~/sinatra/sinatra-scope]$
```

### Ejecución en el cliente. Ruta: /juan

```
[~/sinatra/sinatra-scope]$ curl -v 'http://localhost:4567/juan'
* Adding handle: conn: 0x7fbdd1800000
* Adding handle: send: 0
* Adding handle: recv: 0
```

```

* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7fbdd1800000) send_pipe: 1, recv_pipe: 0
* About to connect() to localhost port 4567 (#0)
* Trying ::1...
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 4567 (#0)
> GET /juan HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4567
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=utf-8
< Content-Length: 21
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Frame-Options: SAMEORIGIN
< Connection: keep-alive
* Server thin 1.6.1 codename Death Proof is not blacklisted
< Server: thin 1.6.1 codename Death Proof
<
* Connection #0 to host localhost left intact
Inside defined route

```

You have the request scope binding inside:

1. get, head, post, put, delete, options, patch, link, and unlink blocks
2. before and after filters
3. helper methods
4. templates/views

### 33.25. Sinatra Authentication

#### 33.25.1. Autentificación Básica en Sinatra con Rack::Auth::Basic

```
[~/srcSTW/sinatra-faq/authentication/basic(esau)]$ cat app.rb
require 'rubygems'
require 'sinatra'

use Rack::Auth::Basic, "Restricted Area" do |username, password|
 [username, password] == ['admin', 'admin']
end

get '/' do
 "You're welcome"
end

get '/foo' do
 "You're also welcome"
end
```

### 33.25.2. Ejemplo con Warden

Donde

- <https://github.com/crguezl/sinatra-warden-example>
- [~/sinatra/sinatra-warden-example(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-warden-example

### 33.25.3. Referencias

1. Ejemplo de uso de sinatra-authentication
2. Véase la sección *OAuth* 93

## 33.26. Sinatra como Middleware

Not only is Sinatra able to use other Rack middleware, any Sinatra application can in turn be added in front of any Rack endpoint as middleware itself.

This endpoint could be another Sinatra application, or any other Rack-based application (Rails/-Ramaze/Camping/...):

1. When a request comes in, all `before` filters are triggered
2. Then, if a route matches, the corresponding block will be executed
3. If no route matches, the request is handed off to the wrapped application
4. The `after` filters are executed after we've got a response back from the route or the wrapped app

Thus, our Sinatra app is a middleware.

```
[~/sinatra/sinatra-as-middleware]$ cat app.rb
require 'sinatra/base'
require 'haml'
require 'pp'

class LoginScreen < Sinatra::Base
 enable :sessions
 enable :inline_templates

 get('/login') { haml :login }

 post('/login') do
 if params[:name] == 'admin' && params[:password] == 'admin'
 puts "params = "
 pp params
 session['user_name'] = params[:name]
 redirect '/'
 else
 redirect '/login'
 end
 end
end

class MyApp < Sinatra::Base
```

```

enable :inline_templates
middleware will run before filters
use LoginScreen

before do
 unless session['user_name']
 halt haml :denied
 end
end

get('\/') do
 haml :cheer, :locals => { :name => session['user_name'] }
end

run!
end

__END__

@@ layout
!!!
%html
 %head
 %title Sinatra as Middleware
 %body
 %h1 Sinatra as Middleware
 = yield

@@ login
%form{:action=>'/login', :method=>'post'}
 %label{:for=>'name'} Name
 %input#name{:type=>"text", :name=>"name", :autofocus => true }
 %br
 %label{:for=>'password'} Password
 %input#password{:type=>"password", :name=>"password"}
 %br
 %button#go{:type=>"submit", :name=>"submit", :value=>"submit"} Click me!

@@ cheer
%h1
 Hello #{name}
 %br

@@ denied
%h1
 Access denied, please
 %a{:href=>'\/login'}login.

```

### 33.27. Práctica: Aplicación Web con Sinatra: Contar la Popularidad de Nuestros Amigos en Twitter

Escriba una aplicación web en Sinatra similar a la solicitada en la práctica 8.16 y que una vez obtenido el nombre del usuario muestre la lista de los amigos de ese usuario (esto es, a quienes sigue)

ordenada según su [popularidad](#) (esto es, según el número de seguidores que tiene). Del mas popular al menos popular (no mas de 10 usuarios. Evite posibles cortes de Twitter por el uso del ancho de banda. Lea Tips to avoid being Rate Limited). Puede usar la aplicación `rack-last-twits` (rama `sinatra`) en <https://github.com/crguezl/rack-last-twits/tree/sinatra> como base para su código.

La siguiente sesión `pry` muestra como obtener los seguidores de un usuario:

```
[1] pry(main)> require 'twitter'
=> true
[2] pry(main)> require './configure'
=> true
[3] pry(main)> client = my_twitter_client()
=> #<Twitter::REST::Client:0x007fe972075e78
 @access_token="*****",
 @access_token_secret="*****",
 @consumer_key="*****",
 @consumer_secret="*****"
[5] pry(main)> seguidores = client.friend_ids("crguezl").attrs[:ids].take(10)
=> [476049040, 249595722,
 582620116, 546228347, 39973845,
 150456846, 79024273, 2693104332,
 2471387516, 1635283039]
[6] pry(main)> amigos = {}
[7] pry(main)> seguidores.map { amigos[client.user(f).name] = client.user(f).followers_count }
=> [269, 269, 269, 269, 269, 269, 269, 269, 269]
[8] pry(main)> amigos
=> {"@laculturaensc"=>486, "Teatro Guimerá"=>2738,
 "TEA Tenerife"=>5649, "Tenerife Ocio"=>12925,
 "lagenda tenerife"=>7876, "El Chikiplan"=>424,
 "PateaTusMontes"=>6038, "Mastering Vim"=>1170,
 "CocoaConf Podcast"=>94, "Kido Tsubomi"=>269}
[9] pry(main)> amigos.sort_by { |x, y| -y }
=> [["Tenerife Ocio", 12925],
 ["lagenda tenerife", 7876], ["PateaTusMontes", 6038],
 ["TEA Tenerife", 5649], ["Teatro Guimerá", 2738],
 ["Mastering Vim", 1170], ["@laculturaensc", 486],
 ["El Chikiplan", 424], ["Kido Tsubomi", 269],
 ["CocoaConf Podcast", 94]]
```

### 33.28. Práctica: TicTacToe

El código que sigue implanta un jugador de tres-en-rayo.

1. Mejore el estilo actual usando SAAS: utilice variables, extensiones, mixins ...
2. Despliegue su versión en Heroku

### Referencias

1. <http://sytw-tresenraya.herokuapp.com/>
2. <https://github.com/crguezl/tictactoe-1>
3. Sass (Syntactically Awesome StyleSheets): [Sass Basics](#)
4. Un TicTacToe Simple (No una webapp)

## Estructura

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ tree
.
|--- Gemfile
|--- Gemfile.lock
|--- Procfile
|--- Rakefile
|--- Readme.md
|--- app.rb
|--- public
| |--- css
| | |--- app.css
| | '--- style.css
| |--- images
| | |--- blackboard.jpg
| | |--- circle.gif
| | '--- cross.gif
| '--- js
| '--- app.js
'--- views
 |--- final.erb
 |--- final.haml
 |--- game.erb
 |--- game.haml
 |--- layout.erb
 |--- layout.haml
 '--- styles.scss

5 directories, 19 files
```

## Rakefile

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat Rakefile
desc "run server"
task :default do
 sh "bundle exec ruby app.rb"
end

desc "install dependencies"
task :install do
 sh "bundle install"
end

###
desc 'build css'
task :css do
 sh "sass views/styles.scss public/css/style.css"
end
```

## HAML

1. game.haml en GitHub
2. layout.haml

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat views/game.haml
.screen
.gameboard
- HORIZONTALS.each do |row|
.gamerow
- row.each do |p|
%a(href=p)
%div{:id => "#{p}", :class => "cell #{b[p]}"}
.message
%h1= m

[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat views/layout.haml
!!!
%html
%head
tic tac toe
-#%link{:rel=>"stylesheet", :href=>"/css/app.css", :type=>"text/css"}
-# dynamically accessed
-#%link{:rel=>"stylesheet", :href=>"/styles.css", :type=>"text/css"}
-# statically compiled
%link{:rel=>"stylesheet", :href=>"css/style.css", :type=>"text/css"}
%script{:type=>"text/javascript", :src=>"http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js"}
%script{:type=>"text/javascript", :src=>"/js/app.js"}
%body
= yield
```

1. El fuente `styles.scss` puede compilarse *dinámicamente*. Véase el fragmento de código que empieza por `get '/styles.css'` en `app.rb`
2. O puede compilarse estáticamente. Véase el Rakefile

## HTML generado

```
<!DOCTYPE html>
<html>
 <head>
 <title>tic tac toe</title>
 <link href='css/style.css' rel='stylesheet' type='text/css'>
 <script src='http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js' type='text/javascript'></script>
 </head>
 <body>
 <div class='screen'>
 <div class='gameboard'>
 <div class='gamerow'>

 <div class='cell' id='a1'></div>

 <div class='cell' id='a2'></div>

 <div class='cell' id='a3'></div>

```

```

</div>
<div class='gamerow'>

 <div class='cell ' id='b1'></div>

 <div class='cell circle' id='b2'></div>

 <div class='cell ' id='b3'></div>

</div>
<div class='gamerow'>

 <div class='cell ' id='c1'></div>

 <div class='cell ' id='c2'></div>

 <div class='cell cross' id='c3'></div>

</div>
<div class='message'>
 <h1></h1>
</div>
</div>
</div>
</body>
</html>

```

## SASS

1. styles.scss
2. Sass (Syntactically Awesome StyleSheets): Sass Basics
3. SASS documentación
4. sass man page
5. *SASS (Syntactically Awesome StyleSheets) 96*

```

~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat views/styles.scss
$red: #903;
$black: #444;
$white: #fff;
$ull: #9900FF;
$pink: #F9A7B0;

$main-font: Helvetica, Arial, sans-serif;
$message-font: 22px/1;

$board-left: 300px;
$board-margin: 0 auto;

```

```

$board-size: 500px;

$opacity: 0.8;

$cell-width: $board-size/8.5;
$cell-height: $board-size/8.5;
$cell-margin: $cell-width/12;
$cell-padding: $cell-width/1.3;

$background: "/images/blackboard.jpg";
$cross: "/images/cross.gif";
$circle: "/images/circle.gif";

body {
 // background-color: lightgrey;
 font-family: $main-font;
 background: url($background) repeat; background-size: cover;
}

.gameboard { //margin-left: $board-left;
 width: $board-size;
 margin: $board-margin;
 text-align:center;
}

.gamerow { clear: both; }

.cell {
 color: blue;
 background-color: white;
 opacity: $opacity;
 width: $cell-width;
 height: $cell-height;
 margin: $cell-margin;
 padding: $cell-padding;
 &:hover {
 color: black ;
 background-color: $ull;
 }
 float: left;
}

@mixin game-piece($image) {
 background: url($image) no-repeat; background-size: cover;
}

.cross { @include game-piece($cross); }
.circle { @include game-piece($circle); }

.base-font { color: $pink; font: $message-font $main-font; }

.message {
 @extend .base-font;
 display: inline;
 background-color: transparent;
}

```

## Procfile

Procfile en GitHub

In order to declare the processes that make our app, and scale them individually, we need to be able to tell Heroku what these processes are.

The Procfile is a simple YAML file which sits in the root of your application code and is pushed to your application when you deploy. This file contains a definition of every process you require in your application, and how that process should be started.

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat Procfile
#web: bundle exec unicorn -p $PORT -E $RACK_ENV
#web: bundle exec ruby app.rb -p $PORT
web: bundle exec ruby app.rb
#web: bundle exec thin start
```

Véase The Procfile is your friend

1. Heroku, Thin and everything in between en StackOverflow
2. Process Types and the Procfile en Heroku

## Gemfile

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat Gemfile
source "https://rubygems.org"
```

```
gem "sinatra"
gem 'haml'
gem "sass", :require => 'sass'
gem 'thin'
```

## La Aplicación

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ cat app.rb
require 'sinatra'
require 'sass'
require 'pp'

settings.port = ENV['PORT'] || 4567
enable :sessions
#use Rack::Session::Pool, :expire_after => 2592000
#set :session_secret, 'super secret'

#configure :development, :test do
set :sessions, :domain => 'example.com'
#end

#configure :production do
set :sessions, :domain => 'herokuapp.com'
#end

module TicTacToe
 HUMAN = CIRCLE = "circle" # human
 COMPUTER = CROSS = "cross" # computer
 BLANK = ""

```

```

HORIZONTALS = [%w{a1 a2 a3}, %w{b1 b2 b3}, %w{c1 c2 c3}]
COLUMNS = [%w{a1 b1 c1}, %w{a2 b2 c2}, %w{a3 b3 c3}]
DIAGONALS = [%w{a1 b2 c3}, %w{a3 b2 c1}]
ROWS = HORIZONTALS + COLUMNS + DIAGONALS
MOVES = %w{a1 a2 a3 b1 b2 b3 c1 c2 c3}

def number_of(symbol, row)
 row.find_all{ |s| session["bs"][s] == symbol }.size
end

def inicializa
 @board = {}
 MOVES.each do |k|
 @board[k] = BLANK
 end
 @board
end

def board
 session["bs"]
end

def [] key
 board[key]
end

def []= key, value
 board[key] = value
end

def each
 MOVES.each do |move|
 yield move
 end
end

def legal_moves
 m = []
 MOVES.each do |key|
 m << key if board[key] == BLANK
 end
 puts "legal_moves: Tablero: #{board.inspect}"
 puts "legal_moves: m: #{m}"
 m # returns the set of feasible moves ["b3", "c2", ...]
end

def winner
 ROWS.each do |row|
 circles = number_of(CIRCLE, row)
 puts "winner: circles=#{circles}"
 return CIRCLE if circles == 3 # "circle" wins
 crosses = number_of(CROSS, row)
 end
end

```

```

 puts "winner: crosses=#{crosses}"
 return CROSS if crosses == 3
 end
 false
end

def smart_move
 moves = legal_moves

 ROWS.each do |row|
 if (number_of(BLANK, row) == 1) then
 if (number_of(CROSS, row) == 2) then # If I have a win, take it.
 row.each do |e|
 return e if board[e] == BLANK
 end
 end
 end
 end
 ROWS.each do |row|
 if (number_of(BLANK, row) == 1) then
 if (number_of(CIRCLE, row) == 2) then # If he is threatening to win, stop it.
 row.each do |e|
 return e if board[e] == BLANK
 end
 end
 end
 end
 # Take the center if open.
 return "b2" if moves.include? "b2"

 # Defend opposite corners.
 if self["a1"] != COMPUTER and self["a1"] != BLANK and self["c3"] == BLANK
 return "c3"
 elsif self["c3"] != COMPUTER and self["c3"] != BLANK and self["a1"] == BLANK
 return "a1"
 elsif self["a3"] != COMPUTER and self["a3"] != BLANK and self["c1"] == BLANK
 return "c1"
 elsif self["c1"] != COMPUTER and self["c1"] != BLANK and self["a3"] == BLANK
 return "a3"
 end

 # Or make a random move.
 moves[rand(moves.size)]
end

def human_wins?
 winner == HUMAN
end

def computer_wins?
 winner == COMPUTER
end

```

```

end

helpers TicTacToe

get %r{^/([abc] [123])?$} do |human|
 if human then
 puts "You played: #{human}!"
 puts "session: "
 pp session
 if legal_moves.include? human
 board[human] = TicTacToe::CIRCLE
 # computer = board.legal_moves.sample
 computer = smart_move
 redirect to ('/humanwins') if human_wins?
 redirect to ('/') unless computer
 board[computer] = TicTacToe::CROSS
 puts "I played: #{computer}!"
 puts "Tablero: #{board.inspect}"
 redirect to ('/computerwins') if computer_wins?
 end
 else
 session["bs"] = inicializa()
 puts "session = "
 pp session
 end
 haml :game, :locals => { :b => board, :m => '' }
end

get '/humanwins' do
 puts "/humanwins session="
 pp session
begin
 m = if human_wins? then
 'Human wins'
 else
 redirect '/'
 end
 haml :final, :locals => { :b => board, :m => m }
rescue
 redirect '/'
end
end

get '/computerwins' do
 puts "/computerwins"
 pp session
begin
 m = if computer_wins? then
 'Computer wins'
 else
 redirect '/'
 end
 haml :final, :locals => { :b => board, :m => m }

```

```

rescue
 redirect '/'
end
end

not_found do
 puts "not found!!!!!!!!!!"
 session["bs"] = inicializa()
 haml :game, :locals => { :b => board, :m => 'Let us start a new game' }
end

get '/styles.css' do
 scss :styles
end

```

### 33.29. Práctica: TicTacToe usando DataMapper

Añada una base de datos a la práctica del TicTacToe 33.28 de manera que se lleve la cuenta de los usuarios registrados, las partidas jugadas, ganadas y perdidas. Repase la sección *DataMapper y Sinatra* 46.

Mejore las hojas de estilo usando SAAS 96. Deberán mostrarse las celdas pares e impares en distintos colores. También deberá mostrarse una lista de jugadores con sus registros.

Despliegue la aplicación en Heroku.

### 33.30. Práctica: Servicio de Syntax Highlighting

Construya una aplicación que provee syntax higlighting para un código que se vuelca en un formulario. Use la gema syntaxi.

El siguiente ejemplo muestra como funciona la gema syntaxi:

```
[~/rubytesting/syntax_highlighting]$ cat ex_syntaxi.rb
require 'syntaxi'
text = <<"EOF"
[code lang="ruby"]
 def foo
 puts 'bar'
 end
[/code]
EOF
formatted_text = Syntaxi.new(text).process
puts formatted_text
```

Ejecución:

```
[~/rubytesting/syntax_highlighting]$ ruby ex_syntaxi.rb
<pre>
<code>
1 def foo
2 puts
'bar'
3 end
</code>
</pre>
```

La gema `syntaxi` usa la gema `syntax`:

```
[~/rubytesting/syntax_highlighting]$ gem which syntaxi/Users/casiano/.rvm/gems/ruby-1.9.2-head
[~/rubytesting/syntax_highlighting]$ grep "require.*'" /Users/casiano/.rvm/gems/ruby-1.9.2-head
require 'syntax/convertors/html'
```

Es en esta gema que se definen las hojas de estilo:

```
[~/rubytesting/syntax_highlighting]$ gem which syntax
/Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax-1.0.0/lib/syntax.rb
[~/rubytesting/syntax_highlighting]$ tree /Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax
/Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax-1.0.0/
|-- data
| |-- ruby.css
| |-- xml.css
| '-- yaml.css
|-- lib
| |-- syntax
| | |-- common.rb
| | |-- convertors
| | | |-- abstract.rb
| | | '-- html.rb
| | |-- lang
| | | |-- ruby.rb
| | | |-- xml.rb
| | | '-- yaml.rb
| | '-- version.rb
| '-- syntax.rb
`-- test
 |-- ALL-TESTS.rb
 |-- syntax
 | |-- tc_ruby.rb
 | |-- tc_xml.rb
 | |-- tc_yaml.rb
 | '-- tokenizer testcase.rb
 '-- tc_syntax.rb
```

7 directories, 17 files

En el esquema incompleto que sigue se ha hecho para el lenguaje Ruby. Añada que se pueda elegir el lenguaje a colorear (xml, yaml).

```
$ tree -A
. .
|-- Gemfile
|-- Gemfile.lock
|-- toopaste.rb
`-- views
 |-- layout.erb
 |-- new.erb
 '-- show.erb

$ cat Gemfile
source 'https://rubygems.org'
```

```

Specify your gem's dependencies in my-gem.gemspec
gemspec
gem 'guard'
gem 'guard-rspec'
gem 'guard-bundler'
gem 'rb-fsevent', '~> 0.9.1'

gem 'syntaxi'

```

Este es un fragmento de la aplicación:

```

[~/srcSTW/syntax_highlighting(withoutdm)]$ cat toopaste.rb
require 'sinatra'
require 'syntaxi'

class String
 def formatted_body
 source = "[code lang='ruby']"
 #{self}
 [/code]"
 html = Syntaxi.new(source).process
 %Q{
 <div class="syntax syntax_ruby">
 #{html}
 </div>
 }
 end
end

get '/' do
 erb :new
end

post '/' do

end

```

Una versión simple de lo que puede ser `new.erb`:

```

[~/srcSTW/syntax_highlighting(withoutdm)]$ cat views/new.erb
<div class="snippet">
 <form action="/" method="POST">
 <textarea name="body" id="body" rows="20"></textarea>

<input type="submit" value="Save"/>
 </form>
</div>

```

Véase la página HTML generada por el programa para la entrada `a = 5`:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Toopaste!</title>

```

```

<style>
 html {
 background-color: #eee;
 }
 .snippet {
 margin: 5px;
 }
 .snippet textarea, .snippet .sbody {
 border: 5px dotted #eee;
 padding: 5px;
 width: 700px;
 color: #fff;
 background-color: #333;
 }
 .snippet textarea {
 padding: 20px;
 }
 .snippet input, .snippet .sdate {
 margin-top: 5px;
 }

 /* Syntax highlighting */
 #content .syntax_ruby .normal {}

 #content .syntax_ruby .comment { color: #CCC; font-style: italic; border: none; margin: none; }
 #content .syntax_ruby .keyword { color: #C60; font-weight: bold; }
 #content .syntax_ruby .method { color: #9FF; }
 #content .syntax_ruby .class { color: #074; }
 #content .syntax_ruby .module { color: #050; }
 #content .syntax_ruby .punct { color: #0D0; font-weight: bold; }
 #content .syntax_ruby .symbol { color: #099; }
 #content .syntax_ruby .string { color: #C03; }
 #content .syntax_ruby .char { color: #F07; }
 #content .syntax_ruby .ident { color: #0D0; }
 #content .syntax_ruby .constant { color: #07F; }
 #content .syntax_ruby .regex { color: #B66; }
 #content .syntax_ruby .number { color: #FF0; }
 #content .syntax_ruby .attribute { color: #7BB; }
 #content .syntax_ruby .global { color: #7FB; }
 #content .syntax_ruby .expr { color: #909; }
 #content .syntax_ruby .escape { color: #277; }

 #content .syntax {
 background-color: #333;
 padding: 2px;
 margin: 5px;
 margin-left: 1em;
 margin-bottom: 1em;
 }

 #content .syntax .line_number {
 text-align: right;
 font-family: monospace;
 padding-right: 1em;
 color: #999;
 }

```

```

</style>
</head>
<body>
 <div class="snippet">
 <div class="snippet">
 <div class="sbody" id="content">
 <div class="syntax syntax_ruby">
 <pre>
 <code>
 1
 a
 =
 5
 </code>
 </pre>
 </div>
 </div>

New Paste!
 </div>
 </body>
</html>

```

La gema

Una versión resumida de `layout.erb`:

```

[~/srcSTW/syntax_highlighting(withoutdm)]$ cat views/layout.erb
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title><%= @title || 'Toopaste!' %></title>
 <style>

 </style>
</head>
<body>
 <%= yield %>
</body>
</html>

```

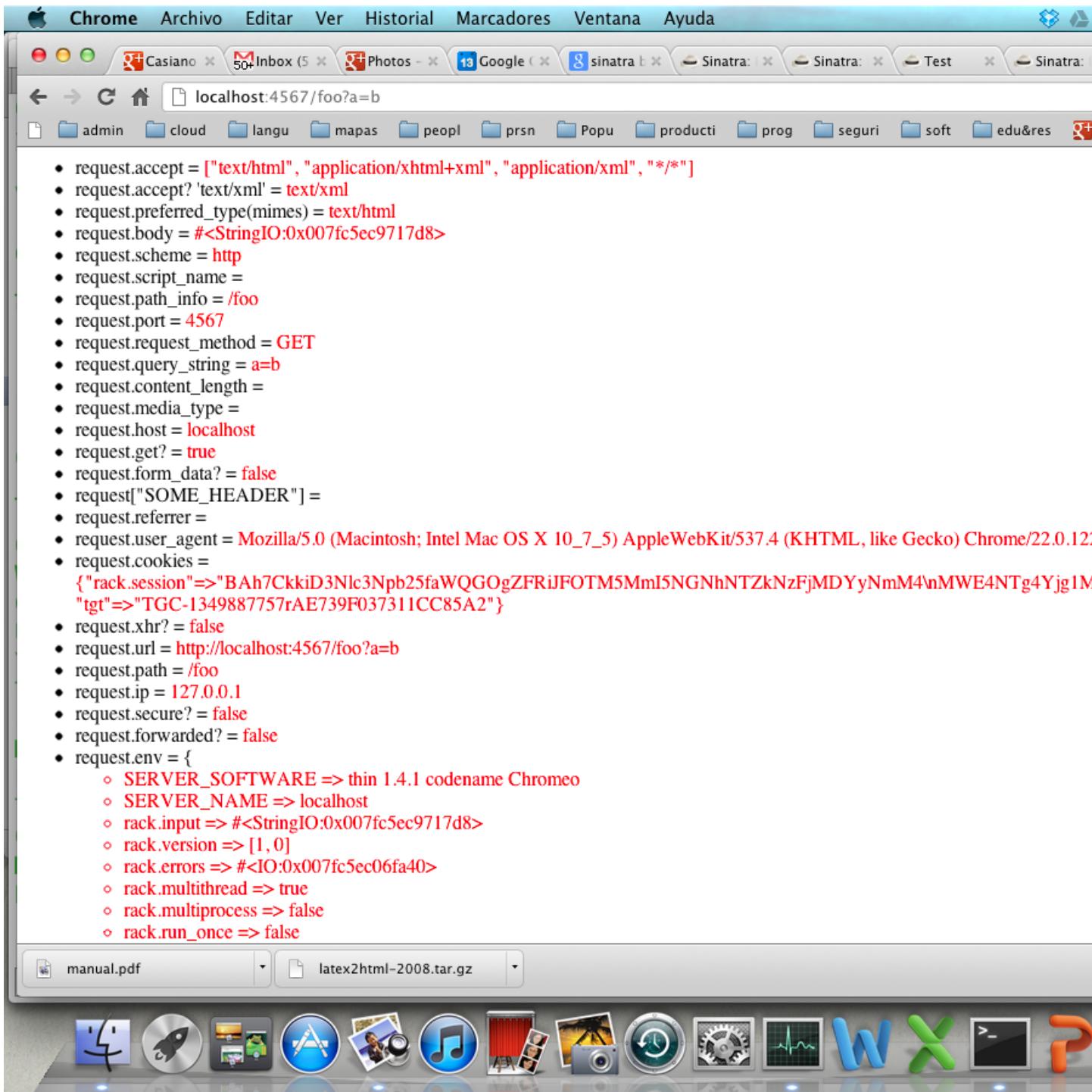
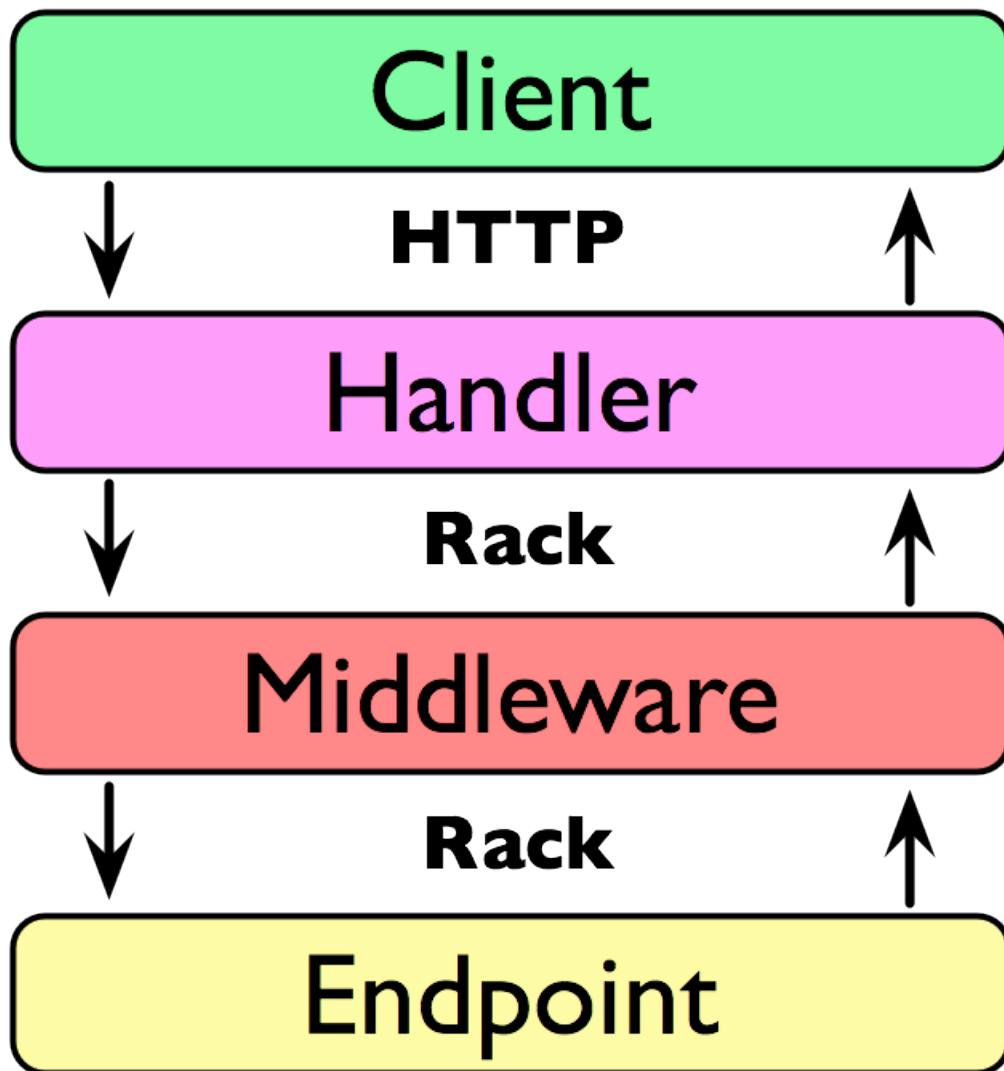


Figura 33.1: El Objeto Request



- Browser
- Proxy
- MongoDB
- Thin
- Filter
- Router
- Rails
- Sinatra

Figura 33.2: La pila Rack

# Capítulo 34

## Sinatra desde Dentro

### 34.1. tux

- tux en GitHub
- Tux: a Sinatra Console
- Tux documentación

### 34.2. Aplicación y Delegación

### 34.3. Helpers y Extensiones

### 34.4. Petición y Respuesta

## Capítulo 35

# Aplicaciones Modulares

Las aplicaciones normales Sinatra se denominan *aplicaciones clásicas sinatra* y viven en Sinatra::Application, que es una subclase de Sinatra::Base.

En las aplicaciones clásicas Sinatra extiende la clase `Object` en el momento de cargarse lo que, en cierto modo, contamina el espacio de nombres global. Eso dificulta que nuestra aplicación pueda ser distribuida como una gema y que se puedan tener varias aplicaciones clásicas en un único proceso.

Una aplicación Sinatra se dice una *aplicación modular sinatra* si no hace uso de Sinatra::Application, renunciando al DSL de alto nivel proveído por Sinatra, sino que hereda de Sinatra::Base.

Podemos combinar una aplicación clásica con una modular, pero sólo puede haber una aplicación clásica por proceso.

```
[~/sinatra/sinatra-simple(master)]$ cat app.rb
require 'sinatra/base'

class App < Sinatra::Base
 get '/' do
 "hello get!"
 end
end

[~/sinatra/sinatra-simple(master)]$ cat config.ru
require './app'

run App
```

# Capítulo 36

## Testing en Sinatra

### 36.0.1. Véase

1. Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis by Darren Jones. Published December 9, 2013 SitePoint
2. Código del artículo anterior [https://github.com/crguezl/number\\_cruncher](https://github.com/crguezl/number_cruncher)
3. Mini MiniTest Tutorial by Tim Millwood
4. sample app with sinatra rspec capybara and selenium by arunyadav4u January 9, 2013
5. NetTuts+ tutorial: Testing Web Apps with Capybara and Cucumber Andrew Burgess on Aug 22nd 2011 with 22 Comments
6. Testing sinatra helpers with rspec por netzfisch's. Enseña como testear un Sinatra helper.

### 36.0.2. Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis

#### Donde

1. Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis by Darren Jones. Published December 9, 2013 SitePoint
2. Código del artículo anterior [https://github.com/crguezl/number\\_cruncher](https://github.com/crguezl/number_cruncher)

#### The Problem

The application to build is a Number Cruncher. It will return information about numbers, such as its factors, whether it is odd or even, and if it's prime.

**Monkey-patch the Integer class** The first thing we want to do is monkey-patch the Integer class to add a method that returns the factors of a number as an array.

```
[~/sinatra/sinatra-number_cruncher(master)]$ cat -n number_cruncher.rb | head -n 10
 1 class Integer
 2 def factors
 3 square_root = self**0.5
 4 (1..square_root).map{ |n| [n,self/n] if self/n*n == self }.compact.flatten.sort
 5 end
 6
 7 def prime?
 8 self.factors.size == 2 ? true : false
 9 end
10 end
```

I also want to add a `prime?` method to test if a number is prime.

Once I've done this, I'll use Sinatra to provide an API that will return information about a number in JSON format.

## See it working

- Number Cruncher is deployed now (September 2014) in Heroku at <http://primefactorization.herokuapp.com/>
- You can see the Continuous Integration tests running on Travis
- The code is on GitHub

## Testing con minitest/autorun y rack/test

Before we write any code, we should write some tests that cover the description above. First of all, create a file called `number_cruncher.rb` and another called `test.rb`.

Then add the following MiniTest setup code to `test.rb`:

```
ENV['RACK_ENV'] = 'test'
require 'minitest/autorun'
require 'rack/test'
require_relative 'number_cruncher.rb'

include Rack::Test::Methods

def app
 Sinatra::Application
end
```

The `Rack::Test::Methods` module includes a variety of helper methods for simulating requests against an application and asserting expectations about the response. It's typically included directly within the test context and makes a few helper methods and attributes available.

This module serves as the primary integration point for using `Rack::Test` in a testing environment.

It depends on an `app` method being defined in the same context, and provides the `Rack::Test` API methods (see `Rack::Test::Session` for their documentation).

If you want to test one Sinatra `app` in isolation, you just return `Sinatra::Application` as shown above.

If you are using Sinatra in the modular style, (Véase la sección 35) replace `Sinatra::Application` above with the `class name` of your `app`.

## Especificando Requisitos

Now, we'll write a spec to test some of the features described above. Add the following code to the bottom of `test.rb`:

```
describe "Number Cruncher" do

 it "should return the factors of 6" do
 6.factors.must_equal [1,2,3,6]
 end

 it "should say that 2 is prime" do
 assert 2.prime?
 end

 it "should say that 10 is not prime" do
 refute 10.prime?
 end
```

```

end

it "should return json" do
 get '/6'
 last_response.headers['Content-Type'].must_match 'application/json'
end

it "should return the correct info about 6 as json" do
 get '/6'
 six_info = { number: 6, factors: 6.factors,
 odd: 6.odd?,
 even: 6.even?, prime: 6.prime? }
 six_info.to_json.must_equal last_response.body
end

end

```

The first test is for the factors method that I plan to add to the Integer class. We check to see that if the integer 6 calls the factors method followed by if the factors of 6 are returned as an array.

Next, test the prime? method that will also be added to the Integer class. First of all, check to see if 2 returns true and then if 10 returns false.

The last two tests are specific to the API. The fourth test checks to see if the app is returning JSON. This is done by performing a get request with 6 as a parameter.

The `last_response.headers` are checked for a content-type of `application/json; charset=utf-8`.

In the last test, we check that the correct info is returned by comparing a JSON string to the `last_response.body` method.

To run our tests, enter the following in a terminal prompt:

```

$ ruby test.rb
This produces the following output:
Running:
EEEEEE
Finished in 0.008729s, 572.8333 runs/s, 0.0000 assertions/s.
5 runs, 0 assertions, 0 failures, 5 errors, 0 skips

```

All five of our tests produce errors, which is expected since we haven't written any code yet. Let's see if we can get those tests passing.

### 36.0.3. Mini MiniTest Tutorial

See Mini MiniTest Tutorial by Tim Millwood.

We'll use MiniTest::Spec, which is a functionally complete spec engine, to create a spec for a Hello World! Sinatra application. Firstly we create a folder 'spec' within the application directory. Within this, two files.

```

ENV['RACK_ENV'] = 'test'

require 'minitest/autorun'
require 'rack/test'
require_relative '../app'

include Rack::Test::Methods

def app
 Sinatra::Application
end

```

The above code is to go into the file `spec_helper.rb`. It sets up the initial setting for the test. We set the `RACK_ENV` environment variable to 'test' then require '`minitest/autorun`' for MiniTest, '`rack/test`' because we are testing a rack based application and our application, in this case `app.rb`. The `Rack::Test::Methods` module is included to add a number of helper methods to allow the testing of rack applications. Finally we define the application as a Sinatra Application. The next file `app_spec.rb` will be our spec.

```
require_relative 'spec_helper'

describe 'Hello World' do

 it 'should have hello world' do
 get '/'
 last_response.must_be :ok?
 last_response.body.must_include "Hello world!"
 end
end
```

Firstly the `spec_helper` file is required, we then create a describe block to describe the 'Hello World' application. Within this a single behaviour. We run a get method against the root route and check the response is ok, and that the page includes the text 'Hello World!'.

### 36.0.4. Sample app with Sinatra Rspec Capybara and Selenium

#### Donde

- <https://github.com/crguezl/sinatra-capybara-selenium>
- [~/sinatra/sinatra-selenium/capybara-selenium(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-selenium/capybara-selenium
- sample app with sinatra rspec capybara and selenium

#### Introducción

Véase la sección *Capybara* 18.

Capybara starts a browser.

#### Gemfile

```
[~/sinatra/sinatra-selenium/capybara-selenium(master)]$ cat Gemfile
source 'https://rubygems.org'

gem "thin"
gem "sinatra"

group :development, :test do
 gem "rspec", ">= 1.2.1"
 gem "capybara", ">= 1.1.2"
 gem "selenium-webdriver"
end
```

#### Rakefile

```
[~/sinatra/sinatra-selenium/capybara-selenium(master)]$ cat Rakefile
task :default => :spec
```

```

desc "run examples"
task :spec do
 sh "rspec -I. spec/my_sinatra_app_specs.rb"
end

task :server do
 sh "ruby my_sinatra_app.rb"
end

Had many problems installing nokogiri 1.6.4 on my mac.
solved this way:

gem install nokogiri -- --use-system-libraries
[--with-xml2-config=/path/to/xml2-config]
[--with-xslt-config=/path/to/xslt-config]
desc "Install nokogiri 1.6.4 (11/2014)"
task :nokogiri do
 command = "gem install nokogiri -- --use-system-libraries "+
 "--with-xml2-config=/usr/local/Cellar/libxml2/2.9.1/lib "+
 "--with-xslt-config=/usr/local/Cellar/libxml2/2.9.1/include/"
 sh command
end

```

### La aplicación: my\_sinatra\_app.rb

```
[~/sinatra/sinatra-selenium/capybara-selenium(master)]$ cat my_sinatra_app.rb
require 'sinatra'
get '/' do
 "Hello Sinatra :)"
end
```

### Las Pruebas: spec/my\_sinatra\_app\_specs.rb

```
[~/sinatra/sinatra-selenium/capybara-selenium(master)]$ cat spec/my_sinatra_app_specs.rb
require_relative '../my_sinatra_app.rb' # this load the file you are testing
require 'spec_helper.rb' # It will load the configuration you set in spec_helper.rb

describe 'make API call to load path' do
 it "should load the home page" do
 visit 'http://0.0.0.0:4567'
 expect(page).to have_content("Sinatra")
 end
end
```

### spec/spec\_helper.rb

```
[~/sinatra/sinatra-selenium/capybara-selenium(master)]$ cat spec/spec_helper.rb
require 'capybara' # loading capybara
include Capybara::DSL
It contain all the methods you use for writing test.

=begin
it will tell capybara to use selenium. It should be noted that
By default, Capybara uses the :rack_test driver, which is fast but
```

```

limited: it does not support JavaScript, nor is it able to access HTTP
resources outside of your Rack application, such as remote
APIs and OAuth services.
=end
Capybara.default_driver = :selenium

def app
 Sinatra::Application
end

ENV['RACK_ENV'] = 'test'

RSpec.configure do |config|
config.expect_with :rspec do |expectations|
expectations.include_chain_clauses_in_custom_matcher_descriptions = true
end
#
config.mock_with :rspec do |mocks|
mocks.verify_partial_doubles = true
end

config.run_all_when_everything_filtered = true
config.filter_run :focus
config.order = :random

=begin

 config.disable_monkey_patching!

 config.warnings = true

 if config.files_to_run.one?
 # Use the documentation formatter for detailed output,
 # unless a formatter has already been configured
 # (e.g. via a command-line flag).
 config.default_formatter = 'doc'
 end

 config.profile_examples = 10

 Kernel.srand config.seed
=end
end

```

## Ejecución del Servidor

```

[~/sinatra/sinatra-selenium/capybara-selenium(master)]$ rake server
ruby my_sinatra_app.rb
== Sinatra/1.4.5 has taken the stage on 4567 for development with backup from Thin
Thin web server (v1.6.3 codename Protein Powder)
Maximum connections set to 1024
Listening on localhost:4567, CTRL+C to stop

```

## Ejecución de las pruebas

```
~/sinatra/sinatra-selenium/capybara-selenium(master)]$ rake
rspec -I. spec/my_sinatra_app_specs.rb
including Capybara::DSL in the global scope is not recommended!
Run options: include {:focus=>true}

All examples were filtered out; ignoring {:focus=>true}

make API call to load path
 should load the home page

Finished in 5.78 seconds (files took 0.66653 seconds to load)
1 example, 0 failures

Randomized with seed 53508
```

## Capítulo 37

# CoffeeScript y Sinatra

```
[~/Dropbox/src/ruby/sinatra/sinatra-coffeescript]$ tree
.
|-- app.rb
'-- views
 '-- application.coffee

1 directory, 2 files
[~/Dropbox/src/ruby/sinatra/sinatra-coffeescript]$ cat app.rb
You'll need to require coffee-script in your app
require 'sinatra'
require 'coffee-script'

get '/application.js' do
 x = coffee :application
 "<script>#{x}</script>"
end

[~/Dropbox/src/ruby/sinatra/sinatra-coffeescript]$ cat views/application.coffee
alert "hello world!"
```

# Capítulo 38

## Streaming

### 38.1. Introducción

En Sinatra 1.3 se introdujo el stream helper. El stream object imita a un objeto IO.

1. Código del método stream
2. Class Stream code: Class of the response body in case you use `#stream`

En ocasiones queremos empezar a enviar datos mientras se esta generando aún el cuerpo de la respuesta. Puede que incluso queramos mantener el envío hasta que el cliente cierra la conexión.

Para eso podemos usar el `stream` helper:

```
[~/sinatra/sinatra-streaming/intro-streaming(master)]$ cat legendary.rb
require 'sinatra'

before do
 content_type 'text/plain'
end

get '/' do
 stream do |out|
 out << "It's gonna be legen -\n"
 sleep 0.5
 out << " (wait for it) \n"
 sleep 1
 out << "- dary!\n"
 end
end
```

El objeto `out` que el método `stream` pasa al bloque es un objeto de la clase `Sinatra::Helpers::Stream`. Este objeto representa el cuerpo de la respuesta en el caso en que se use `stream`.

El método `stream` permite enviar datos al cliente incluso cuando el cuerpo del cliente no ha sido generado completamente.

### Utilidades del Streaming

Esto nos permite implementar

- streaming APIs,
- Server Sent Events. Server-sent events es una tecnología que proporciona notificaciones push desde el servidor a un navegador cliente en forma de eventos DOM

- y que puede ser usada como base para WebSockets. WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.
- Puede también ser utilizada para incrementar el throughput si parte pero no todo el contenido depende de un recurso que es lento.

## Dependencia del Servidor

Nótese que la conducta de streaming, en particular el número de solicitudes concurrentes, depende en gran medida del servidor web que sirve la aplicación.

- Algunos servidores como `Webrick` no soportan streaming. En tal caso el cuerpo es enviado en una tacada.
- `Shotgun` tampoco soporta streaming

## `Sinatra::Streaming`

1. Sinatra 1.3 introduced the stream helper.
2. The addon provided by the gem `sinatra-contrib` improves the streaming API by making the stream object immitate an `IO` object, making the body play nicer with middleware unaware of streaming.

This is useful when passing the stream object to a library expecting an `IO` or `StringIO` object.

```
06:31] [~/srcSTW/streaming/upandrunning_streaming]$ cat -n simple.rb
```

```
 1 # http://www.sinatrarb.com/contrib/streaming.html
 2 # $ gem install sinatra-contrib
 3 require 'sinatra'
 4 require 'sinatra/streaming'
 5 set server: 'thin'
 6 #set server: 'unicorn'
 7
 8 get '/' do
 9 stream do |out|
10 puts out.methods
11 out.puts "Hello World!", "How are you?"
12 out.write "Written #{out.pos} bytes so far!\n"
13 out.putc(65) unless out.closed?
14 out.flush
15 end
16 end
```

`out` es un objeto `Sinatra::Helpers::Stream`.

## `sinatra/streaming` en Aplicaciones Modulares

Use the top-level `helpers` method to introduce the methods in the module `Sinatra::Streaming` for use in route handlers and templates:

```
require "sinatra/base"
require "sinatra/streaming"

class MyApp < Sinatra::Base
 helpers Sinatra::Streaming
end
```

## Manteniendo la Conexión Abierta: `keep_open`

- Si en el método `stream` el parámetro opcional `keep_open` se pone a `true` `close` no será llamado al finalizar el bloque, permitiendo su cierre en un punto posterior del flujo de ejecución.
- Esto sólo funciona en *evented servers*

Broadly speaking, there are two ways to handle concurrent requests to a server:

1. *Threaded servers* use multiple concurrently-executing threads that each handle one client request,
2. *Evented servers* run a single event loop that handles events for all connected clients
3. Thin y Rainbows son ejemplos de servidores que pueden funcionar como *evented servers*.  
(Otros servidores quizás cerrarán el stream)

El siguiente ejemplo muestra como mantener una conexión persistente, abierta para enviar mensajes de broadcast a unos suscriptores:

```
[~/sinatra/sinatra-streaming/upandrunning-streaming(master)]$ cat a_simple_streaming_example.r
require 'sinatra'

before do
 content_type :txt
end

set server: 'thin', connections: []

get '/consume' do
 stream(:keep_open) do |out|
 # store connection for later on
 settings.connections << out
 logger.warn "connections.length = #{settings.connections.length}"

 # remove connection when closed properly
 out.callback do
 logger.warn "connection closed. out = #{out}"
 settings.connections.delete(out)
 logger.warn "connections.length = #{settings.connections.length}"
 end

 # remove connection when due to an error
 out.onerror do
 logger.warn "we just lost connection!. out = #{out}"
 settings.connections.delete(out)
 logger.warn "connections.length = #{settings.connections.length}"
 end
 end # stream
end

get '/produce/:message' do
 settings.connections.each do |out|
 out << "#{Time.now} -> #{params[:message]}" << "\n"
 end
end

"Sent #{params[:message]} to all clients."
end
```

Para usar este ejemplo utilizaremos este Rakefile:

```
[~/sinatra/sinatra-streaming/upandrunning-streaming(master)]$ cat Rakefile
task :default => :server

desc "run the server for the stream(:keep_open) example"
task :server do
 sh "ruby a_simple_streaming_example.rb"
end

desc "visit with browser 'localhost:4567/consume'"
task :consume do
 sh "open http://localhost:4567/consume"
end

desc "send messages to the consumer"
task :produce do
 (1..10).each do |i|
 sh "sleep 1; curl http://localhost:4567/produce/#{i}%0D"
 end
end

desc "start both consumer and producer"
task :all => [:consume, :produce]
```

1. Primero arrancamos el servidor. Obsérvese que el servidor usado es `thin`.

```
[~/sinatra/sinatra-streaming/upandrunning-streaming(master)]$ rake server
ruby a_simple_streaming_example.rb
== Sinatra/1.4.4 has taken the stage on 4567 for development with backup from Thin
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on localhost:4567, CTRL+C to stop
```

2. Después visitamos con un navegador la página `localhost:4567/consume`.

```
[~/sinatra/sinatra-streaming/upandrunning-streaming(master)]$ rake consume
open http://localhost:4567/consume
```

Esto abre (en MacOS X) un navegador que queda a la espera del servidor de que la ruta de producción genere algún contenido

3. Si se desea abra alguna otra página de navegación privada (nueva ventana de incógnito en Chrome) en la misma URL `localhost:4567/consume`

4. Arranquemos el productor:

```
[~/sinatra/sinatra-streaming/upandrunning-streaming(master)]$ rake produce
sleep 1; curl http://localhost:4567/produce/1%0D
to all clients.sleep 1; curl http://localhost:4567/produce/2%0D
to all clients.sleep 1; curl http://localhost:4567/produce/3%0D
to all clients.sleep 1; curl http://localhost:4567/produce/4%0D
to all clients.sleep 1; curl http://localhost:4567/produce/5%0D
to all clients.sleep 1; curl http://localhost:4567/produce/6%0D
to all clients.sleep 1; curl http://localhost:4567/produce/7%0D
```

```

to all clients.sleep 1; curl http://localhost:4567/produce/8%0D
to all clients.sleep 1; curl http://localhost:4567/produce/9%0D
to all clients.sleep 1; curl http://localhost:4567/produce/10%0D
to all clients.

```

5. Esto hace que en (los) navegadores/clientes que estaban viendo localhost:4567/consume aparezca algo parecido a esto:

```

2013-11-22 22:42:24 +0000 -> 1
2013-11-22 22:42:25 +0000 -> 2
2013-11-22 22:42:26 +0000 -> 3
2013-11-22 22:42:27 +0000 -> 4
2013-11-22 22:42:28 +0000 -> 5
2013-11-22 22:42:29 +0000 -> 6
2013-11-22 22:42:30 +0000 -> 7
2013-11-22 22:42:31 +0000 -> 8
2013-11-22 22:42:32 +0000 -> 9
2013-11-22 22:42:33 +0000 -> 10

```

En la consola del servidor aparecerá algo parecido a esto:

```

[~/sinatra/sinatra-streaming/upandrunning-streaming(master)]$ rake
ruby a_simple_streaming_example.rb
== Sinatra/1.4.4 has taken the stage on 4567 for development with backup from Thin
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on localhost:4567, CTRL+C to stop
W, [2013-11-22T22:46:17.132773 #21927] WARN -- : connections.length = 1
W, [2013-11-22T22:47:16.655453 #21927] WARN -- : connection closed. out = #<Sinatra::Helpers:
W, [2013-11-22T22:47:16.655557 #21927] WARN -- : connections.length = 0
W, [2013-11-22T22:47:16.655620 #21927] WARN -- : we just lost connection!. out = #<Sinatra::
W, [2013-11-22T22:47:16.655677 #21927] WARN -- : connections.length = 0
127.0.0.1 - - [22/Nov/2013 22:47:16] "GET /consume HTTP/1.1" 200 - 59.5292
127.0.0.1 - - [22/Nov/2013 22:50:32] "GET /produce/1%0D HTTP/1.1" 200 23 0.0009
127.0.0.1 - - [22/Nov/2013 22:50:33] "GET /produce/2%0D HTTP/1.1" 200 23 0.0008
127.0.0.1 - - [22/Nov/2013 22:50:34] "GET /produce/3%0D HTTP/1.1" 200 23 0.0009
127.0.0.1 - - [22/Nov/2013 22:50:35] "GET /produce/4%0D HTTP/1.1" 200 23 0.0008
127.0.0.1 - - [22/Nov/2013 22:50:36] "GET /produce/5%0D HTTP/1.1" 200 23 0.0009
127.0.0.1 - - [22/Nov/2013 22:50:37] "GET /produce/6%0D HTTP/1.1" 200 23 0.0009
127.0.0.1 - - [22/Nov/2013 22:50:38] "GET /produce/7%0D HTTP/1.1" 200 23 0.0009
127.0.0.1 - - [22/Nov/2013 22:50:39] "GET /produce/8%0D HTTP/1.1" 200 23 0.0007
127.0.0.1 - - [22/Nov/2013 22:50:40] "GET /produce/9%0D HTTP/1.1" 200 23 0.0006
127.0.0.1 - - [22/Nov/2013 22:50:41] "GET /produce/10%0D HTTP/1.1" 200 24 0.0009

```

El objeto `out` de la clase `Sinatra::Helpers::Stream` dispone de los métodos `#callback(&block) => Object` y `errback` (parece que `errback` es un alias de `callback`).

```

File 'lib/sinatra/base.rb'

def callback(&block)
 return yield if @closed
 @callbacks << block
end

```

1. Por debajo este ejemplo usa `Event::Machine`.

2. Event::Machine is a library for Ruby, C++, and Java programs.
3. It provides event-driven I/O using the Reactor pattern.
  - a) The *reactor design pattern* is an event handling pattern for handling service requests delivered concurrently to a service handler by one or more inputs.
  - b) The service handler then demultiplexes the incoming requests and dispatches them synchronously to the associated request handlers.
4. El módulo Module: EventMachine::Deferrable provee métodos `(Object) callback(&block)`. y `(Object) errback(&block)`
  - a) El método `(Object) callback(&block)` specifies a block to be executed if and when the Deferrable object receives a status of :succeeded.
 

```
out.callback do
 logger.warn "connection closed. out = #{out}"
 settings.connections.delete(out)
 logger.warn "connections.length = #{settings.connections.length}"
end
```
  - b) El método `(Object) errback(&block)` specifies a block to be executed if and when the Deferrable object receives a status of :failed.
 

```
out.errback do
 logger.warn "we just lost connection!. out = #{out}"
 settings.connections.delete(out)
 logger.warn "connections.length = #{settings.connections.length}"
end
```
  - c) Event::Machine (EM) adds two different formalisms for lightweight concurrency to the Ruby programmer's toolbox: spawned processes and *deferrables*.

## 38.2. Streaming y Valores de Retorno

El valor de retorno del bloque de una ruta determina el cuerpo de respuesta pasado al cliente HTTP, o al menos al siguiente middleware en la pila de Rack. Lo habitual es que sea una `String` pero puede ser otra cosa.

De hecho, podemos retornar cualquier tipo de objeto que sea una respuesta Rack válida, un objeto que disponga de un `each` o un código de estatus HTTP:

- Un `Array` con tres elementos: `[status (Fixnum), headers (Hash), response body (responds to #each)]`
- Un `Array` con dos elementos: `[status (Fixnum), response body (responds to #each)]`
- Un objeto que responde a `#each` y que le pasa strings al bloque dado
- Un `Fixnum` que representa el status code

Así podemos, implementar el siguiente ejemplo que funciona en streaming cuando se usa con puma:

```
[~/sinatra/sinatra-streaming/intro-streaming(master)]$ cat stream.rb
require 'sinatra'
```

```
before do
 content_type :txt
end
```

```

class Flujo
 def each
 5.times do |i|
 yield "#{i}: #{Time.now}\n"
 sleep 1
 end
 end
end

```

```

get '/' do
 puts env
 Flujo.new
end

```

Véase el código en [https://github.com/crguezl/sinatra\\_intro/tree/master/streaming](https://github.com/crguezl/sinatra_intro/tree/master/streaming).

```

[~/sinatra/sinatra-streaming/intro-streaming(master)]$ cat Rakefile
task :default => :puma
desc "run stream.rb using puma server"
task :puma do
 sh "ruby stream.rb -e production -s puma"
end

desc "run stream.rb using thin server"
task :thin do
 sh "ruby stream.rb -e production -s thin"
end

[~/sinatra/sinatra-streaming/intro-streaming(master)]$ rake
ruby stream.rb -e production -s puma
Puma 2.6.0 starting...
* Min threads: 0, max threads: 16
* Environment: development
* Listening on tcp://0.0.0.0:4567
== Sinatra/1.4.4 has taken the stage on 4567 for production with backup from Puma
127.0.0.1 - - [01/Dec/2013 21:05:32] "GET / HTTP/1.1" 200 - 5.0094

```

Nota: con `thin` no funciona en modo streaming.

Prepara la página completa y la suelta de golpe.

Cuando visitamos `localhost:4567` obtenemos una salida en la que cada línea aparece un segundo después que la anterior:

```

0: 2013-12-01 21:05:27 +0000
1: 2013-12-01 21:05:28 +0000
2: 2013-12-01 21:05:29 +0000
3: 2013-12-01 21:05:30 +0000
4: 2013-12-01 21:05:31 +0000

```

### 38.3. Sinatra usando Streaming, Rack MiddleWare y map

Véase el código en `sinatra_intro/streaming/better_middleware_handling.rb` en GitHub.

Otro beneficio que obtenemos cuando usamos `Sinatra::Streaming` (parte de `sinatra-contrib`) es que:

Blocks passed to `#map!` or `#map` will actually be applied when streaming takes place (véase <http://www.sinatrarb.com/contrib/streaming.html>):

```
[~/sinatra/sinatra-streaming/intro-streaming(master)]$ cat better_middleware_handling.rb
http://www.sinatrarb.com/contrib/streaming.html

require 'sinatra'
require 'sinatra/streaming'

class StupidMiddleware
 def initialize(app)
 @app = app
 end

 def call(env)
 status, headers, body = @app.call(env)
 # Blocks passed to #map! or #map will actually be applied when streaming takes place
 body.map! { |e| e.upcase }
 [status, headers, body]
 end
end

use StupidMiddleware

before do
 content_type :html
end

song = %q{
And now, the end is near
And so I face the final curtain
My friend, I'll say it clear
I'll state my case, of which I'm certain
I've lived a life that's full
I traveled each and ev'ry highway
And more, much more than this, I did it my way

Regrets, I've had a few
But then again, too few to mention
I did what I had to do , I saw it through without exemption
I planned each charted course, each careful step along the highway
And more, much more than this, I did it my way

Yes, there were times, I'm sure you knew
When I bit off more than I could chew
And through it all, when there was doubt
I ate it up and spit it out
I faced it all and I stood tall and did it my way

I've loved, I've laughed and cried
I've had my fill, my share of losing
And now, as tears subside, I find it all so amusing
To think I did all that
And may I say, not in a shy way,
"Oh, no, oh, no, not me, I did it my way"
}
```

```

For what is a man, what has he got?
If not himself, then he has naught
The right to say the things he feels and not the words of one who kneels
The record shows I took the blows and did it my way!
}.split(/\n/)

song.map! { |x| x.split(/\s+/) }

set :song, song
set :colors, ["red", "blue", "black", "green", "yellow", "blueviolet"]

get '/' do
 color = 0
 stream do |out|
 out.puts '<pre>'
 settings.song.each do |line|
 line.each do |w|
 out.print %Q{}#{w}
 color = (color+1) % settings.colors.size
 sleep 0.02
 end
 out.puts "\n"
 sleep 1
 end
 out.puts '</pre>'
 out.close
 end
end

```

## 38.4. Enlaces Relacionados

- Streaming Responses at Sinatra Intro
- Sinatra::Streaming

## 38.5. A simple demonstration of streaming Redis pub/sub data

Véase:

1. A simple demonstration of streaming Redis pub/sub data over HTTP via Sinatra's streaming capabilities.
2. A simple demonstration of streaming Redis pub/sub data over HTTP via Sinatra's streaming capabilities. forked version crguezl
3. Véa el capítulo *Redis y Sinatra* 43
4. redis gem en GitHub
5. redis documentación de la gema
6. Redis Pub/Sub
7. Redis::Subscription:
8. Heroku addon Redis To Go

## web.rb

```
[~/sinatra/sinatra-streaming/sinatra-streaming-example(master)]$ cat web.rb
require 'redis'
require 'sinatra'

configure do
 redis_url = ENV["REDISTOGO_URL"] || "redis://localhost:6379"
 uri = URI.parse(redis_url)
 set :redis, Redis.new(:host => uri.host, :port => uri.port, :password => uri.password)
end

get '/' do
 "<pre>curl -v https://sinatra-streaming-example.herokuapp.com/stream</pre>"
end

get '/stream' do
 puts "connection made"

 stream do |out|
 settings.redis.subscribe 'time' do |on|
 on.message do |channel, message|
 out << "#{$message}\n"
 end
 end
 end
end
end
```

Redis is a key-value store; it supports lists, hashes, sets, and ordered sets.

El código

```
redis_url = ENV["REDISTOGO_URL"] || "redis://localhost:6379"
```

decide si estamos en Heroku o no. Si estamos en Heroku deberemos haber instalado el Heroku add-on Redis To Go.

La llamada:

```
set :redis, Redis.new(:host => uri.host, :port => uri.port, :password => uri.password)
```

conecta con el servidor Redis.

The Redis class exports methods that are named identical to the commands they execute. The arguments these methods accept are often identical to the arguments specified on the Redis website.

In Redis, SUBSCRIBE, UNSUBSCRIBE and PUBLISH implement the Publish/Subscribe messaging paradigm where

1. In the *Publish/Subscribe* pattern, senders (publishers) are not programmed to send their messages to specific receivers (subscribers).
2. Rather, published messages are characterized into channels, without knowledge of what (if any) subscribers there may be.
3. Subscribers express interest in one or more channels, and only receive messages that are of interest, without knowledge of what (if any) publishers there are.
4. This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology.

For instance in order to subscribe to channels `foo` and `bar` the client issues a `SUBSCRIBE` providing the names of the channels:

```
SUBSCRIBE foo bar
```

Messages sent by other clients to these channels will be pushed by Redis to all the subscribed clients.

A client subscribed to one or more channels should not issue commands, although it can subscribe and unsubscribe to and from other channels.

The reply of the `SUBSCRIBE` and `UNSUBSCRIBE` operations are sent in the form of messages, so that the client can just read a coherent stream of messages where the first element indicates the type of message.

El objeto `on` pasado al bloque está en la clase `Redis::Subscription`:

```
settings.redis.subscribe 'time' do |on|
 on.message do |channel, message|
 out << "#{$message}\n"
 end
end
```

el objeto `on` dispone del método `message` que establece una callback que será ejecutada cada vez que se produzca un mensaje.

## worker.rb

```
[~/sinatra/sinatra-streaming/sinatra-streaming-example(master)]$ cat worker.rb
require 'uri'
require 'redis'

redis_url = ENV["REDISTOGO_URL"] || "redis://localhost:6379"
uri = URI.parse(redis_url)
r = Redis.new(:host => uri.host, :port => uri.port, :password => uri.password)

while true do
 puts "publishing..."
 r.publish "time", Time.now.utc
 sleep 1
end
```

## Procfile

```
[~/sinatra/sinatra-streaming/sinatra-streaming-example(master)]$ cat Procfile
web: bundle exec ruby web.rb -p $PORT
worker: bundle exec ruby worker.rb
```

The `web` process type, describes to Heroku how to start the web application server.

An interesting thing to mention here is that Heroku only auto-launches the web process in your `Procfile` when deploying, whereas `foreman` launches everything.

## Gemfile

```
[~/sinatra/sinatra-streaming/sinatra-streaming-example(master)]$ cat Gemfile
source 'https://rubygems.org'

gem 'foreman'
gem 'redis'
gem 'sinatra'
gem 'thin'
```

## Ejecución: Arranca el Servidor Redis

**Ejecución 2: Arranca la aplicación Sinatra y el Worker** Obsérvese como `bundle exec foreman start` arranca tanto a `worker` como a `web`:

```
[~/sinatra/sinatra-streaming/sinatra-streaming-example(master)]$ bundle exec foreman start
10:19:04 web.1 | started with pid 5606
10:19:04 worker.1 | started with pid 5607
10:19:05 worker.1 | publishing...
10:19:06 web.1 | == Sinatra/1.3.0 has taken the stage on 5000 for development with backup
10:19:06 web.1 | >> Thin web server (v1.2.11 codename Bat-Shit Crazy)
10:19:06 web.1 | >> Maximum connections set to 1024
10:19:06 web.1 | >> Listening on 0.0.0.0:5000, CTRL+C to stop
10:19:06 worker.1 | publishing...
10:19:07 worker.1 | publishing...
10:19:08 worker.1 | publishing...
10:19:37 worker.1 | publishing...
10:19:38 web.1 | connection made
10:19:39 worker.1 | publishing...
.....
```

### Ejecución 3: Arranca un Cliente

```
[~/sinatra/sinatra-streaming/sinatra-streaming-example(master)]$ curl -v http://localhost:5000
* Adding handle: conn: 0x7fc71280aa00
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7fc71280aa00) send_pipe: 1, recv_pipe: 0
```

```

* About to connect() to localhost port 5000 (#0)
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 5000 (#0)
> GET /stream HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:5000
> Accept: */*
>
< HTTP/1.1 200 OK
< X-Frame-Options: sameorigin
< X-XSS-Protection: 1; mode=block
< Content-Type: text/html; charset=utf-8
< Connection: close
* Server thin 1.2.11 codename Bat-Shit Crazy is not blacklisted
< Server: thin 1.2.11 codename Bat-Shit Crazy
<
2013-12-04 13:24:13 UTC
2013-12-04 13:24:14 UTC
2013-12-04 13:24:15 UTC
2013-12-04 13:24:16 UTC
2013-12-04 13:24:17 UTC
.....

```

## 38.6. Comet y Server Sent-Events

*Comet* is a web application architecture in which a long-held HTTP request allows a web server to push data to a browser, without the browser explicitly requesting it.

In Ajax the client pulls data from the server. With Comet the server pushes data to the client.

Other names for Comet are

1. *Server Push*
2. *Ajax Push*
3. *HTTP Streaming*

*Server Sent Events* defines a simple Comet API in the form of a `EventSource` object. Server-Sent Events (SSE) is a standard describing how servers can initiate data transmission towards clients once an initial client connection has been established.

### Server-Sent Events

The Server-Sent Events draft standard defines an `EventSource` object that makes *Comet applications*<sup>1</sup> trivial to write.

Simply pass a URL to the `EventSource()` constructor and then listen for message events on the returned object:

```

var ticker = new EventSource("/stockprices");
ticker.onmessage = function(e) {
 var type = e.type;
 var data = e.data;

 // Now process the event type and event data strings.
}

```

---

<sup>1</sup> The term Comet refers to a web application architecture that uses scripted HTTP. In a sense, Comet is the reverse of Ajax: in Comet, it is the web server that initiates the communication, asynchronously sending messages to the client

- The event object associated with a message event has a `data` property that holds whatever string the server sent as the payload for this event.
- The event object also has a `type` property like all event objects do. The default value is `message`, but the event source can specify a different string for the property.
- A single `onmessage` event handler receives all events from a given server event source, and can dispatch them, if necessary, based on their `type` property.

## The Server-Sent Event protocol

The *Server-Sent Event* protocol is straightforward:

1. The client initiates a connection to the server (when it creates the `EventSource` object) and the server keeps this connection open.
2. When an event occurs, the server writes lines of text to the connection.
3. An event going over the wire might look like this:

```
event: bid # sets the type of the event object
data: GOOG # sets the data property
data: 999 # appends a newline and more data
 # a blank line triggers the message event
```

4. There are some additional details to the protocol that allow events to be given IDs and allow a reconnecting client to tell the server what the ID of the last event it received was, so that a server can resend any events it missed.

### 38.6.1. Ejemplo Simple

#### Donde

- [~/local/src/ruby/sinatra/sinatra-streaming/jquery-streaming(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-streaming/jquery-streaming
- <https://github.com/crguezl/sinatra-eventsource-jquery>

#### app.rb

```
[~/local/src/ruby/sinatra/sinatra-streaming/jquery-streaming(master)]$ cat app.rb
require 'sinatra'
require 'pp'

set :server, :thin

get '/' do
 send_file "index.html"
end

get '/bottles' do
 content_type "text/event-stream"
 stream do |out|
 0.upto(50) do |i|
 out << "id: #{i}\n"
 out << "data: #{i} bottle(s) on a wall...\n\n"
 sleep 0.3
 end
 end
end
```

```

 end
 out << "data: CLOSE\n\n"
 end
end

```

### index.html

```
[~/local/src/ruby/sinatra/sinatra-streaming/jquery-streaming(master)]$ cat index.html
<!DOCTYPE html>
<html>
 <head>
 <title>Sinatra streaming example</title>
 <meta charset="utf-8" />
 <script src="http://code.jquery.com/jquery-1.9.0.min.js"></script>
 <style>body {font-family: sans-serif;}</style>
 </head>
 <body>
 <h1>Welcome to this Sinatra Streaming example!</h1>
 <input id="bottles-button" type="button" value="Bottles!" />
 <pre id="bottles-output">
 </pre>
 <script>
 $("#bottles-button").click(function() {
 var src = new EventSource('/bottles?foo=bar');
 src.onmessage = function(e) {
 console.log(e)
 $('#bottles-output').append("\n" + e.data)
 if (e.data == 'CLOSE') src.close()
 };
 });
 </script>
 </body>
</html>
```

### Véase

- Pushing updates in real-time with Server-Sent Events

#### 38.6.2. Enlaces Relacionados

- Real Time Rack por Konstantin Haase en <http://confreaks.com/>
- Slides for Real Time Rack". Contains the example used in the presentations. It is also of how to use Scott Chacon showoff presentation software
- El blog de Konstantin Haase desarrollador al cargo de Sinatra
- Konstantin Haase en GitHub
- Simple Chat Application using the Sinatra Streaming API
- Stream Updates With Server-Sent Events
- Using server-sent events
- EventSource

- Chat Server with server-sent Events
- A shared canvas where multiple clients can draw lines using EM-Websocket
- JQuery documentation
- What is the Document Object Model?
- JavaScript and HTML DOM Reference

### 38.6.3. Chat Utilizando Streaming y Server Sent Events (SSE)

Los fuentes de este ejemplo se encuentran en la rama `original` del repositorio `sinatra-streaming-example-chat`. La rama `simple` en <https://github.com/crguezl/sinatra-streaming-example-chat/tree/simple> contiene una versión equivalente pero con los templates separados.

```
[~/srcSTW/streaming/chat_with_streaming(master)]$ cat -n chat.rb
1 # coding: utf-8
2 require 'sinatra'
3 set server: 'thin', connections: []
4
5 get '/' do
6 halt erb(:login) unless params[:user]
7 erb :chat, locals: { user: params[:user].gsub(/\W/, '') }
8 end
```

Cuando se visita la raíz la primera vez `params[:user]` es `nil` y se muestra el formulario definido en `login.erb` que obtiene un alias para el usuario:

```
[~/sinatra/sinatra-streaming/chat_with_streaming(simple)]$ cat views/login.erb
<form action='/' method='GET'>
 <label for='user'>User Name:</label>
 <input name='user' value=' ' autofocus/>
 <input type='submit' value="GO!" />
</form>
```

El atributo `for='user'` de `label` indica que esta etiqueta está asociada con el campo `input` cuyo atributo `name` es `user`.

Una vez que el `<input name='user'>` es llenado el formulario es procesado por la misma ruta `/` que ahora muestra el resultado de la plantilla `chat`.

```
10 get '/stream', provides: 'text/event-stream' do
11 stream :keep_open do |out|
12 settings.connections << out
13 out.callback { settings.connections.delete(out) }
14 end
15 end
```

Routes may include a variety of matching conditions, such as the user `agent:`, `:host_name` and `:provides`:

```
get '/', :provides => ['rss', 'atom', 'xml'] do
 builder :feed
end
```

Sending an event stream from the source is a matter of constructing a plaintext response, served with a `text/event-stream` Content-Type, that follows the Server Sent Event (SSE) format.

Sigamos:

```

16
17 post '/' do
18 settings.connections.each { |out| out << "data: #{params[:msg]}\n\n" }
19 204 # response without entity body
20 end

```

## 204 No Content

The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metainformation.

If the client is a user agent, it SHOULD NOT change its document view from that which caused the request to be sent. This response is primarily intended to allow input for actions to take place without causing a change to the user agent's active document view, although any new or updated metainformation SHOULD be applied to the document currently in the user agent's active view.

The 204 response MUST NOT include a message-body, and thus is always terminated by the first empty line after the header fields.

```

21
22 __END__
23
24 @@ layout
25 <html>
26 <head>
27 <title>Super Simple Chat with Sinatra</title>
28 <meta charset="utf-8" />
29 <script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
30 </head>
31 <body><%= yield %></body>
32 </html>

```

The `<script>` tag is used to define a client-side script, such as a JavaScript.

The `<script>` element either contains scripting statements, or it points to an external script file through the `src` attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

The Google Hosted Libraries is a content distribution network for the most popular, open-source JavaScript libraries. To add a library to your site, simply use `<script>` tags to include the library. See <https://developers.google.com/speed/libraries/devguide>:

**jQuery**

```

snippet: <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js"></script>
site: http://jquery.com/
versions: 1.8.3, 1.8.2, 1.8.1, 1.8.0, ...
note: 1.2.5 and 1.2.4 are not hosted due to their short and unstable lives in the wild.

```

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

`jquery.min.js` is a minified version of the JQuery JavaScript library, which provides a number of basic functions for websites.

```

34 @@ login
35 <form action='/'>
36 <label for='user'>User Name:</label>
37 <input name='user' value=' '>
38 <input type='submit' value="GO!" />
39 </form>
40

```

```

41 @@ chat
42 <pre id='chat'></pre>
43
44 <script>
45 // reading
46 var es = new EventSource('/stream');
47 es.onmessage = function(e) { $('#chat').append(e.data + "\n") };
48
49 // writing
50 $("form").live("submit", function(e) {
51 $.post('/', {msg: "<%= user %>: " + $('#msg').val()});
52 $('#msg').val(''); $('#msg').focus();
53 e.preventDefault();
54 });
55 </script>
56
57 <form>
58 <input id='msg' placeholder='type message here...' />
59 </form>

```

- *Server-Sent Events (SSE)* are a standard describing how servers can initiate data transmission towards clients once an initial client connection has been established.
- They are commonly used to send message updates or continuous data streams to a browser client and designed to enhance native, cross-browser streaming through a JavaScript API called `EventSource`, through which a client requests a particular URL in order to receive an event stream.
- The idea behind *SSEs* is natural: a web app ”subscribes” to a stream of updates generated by a server and, whenever a new event occurs, a notification is sent to the client.
- When communicating using *SSEs*, a server can push data to your app whenever it wants, without the need to make an initial request. In other words, updates can be streamed from server to client as they happen.
- *SSEs* open a *single unidirectional channel* between server and client.
- A Ruby/EventMachine based server implementation for `WebSocket` and *Server-Sent Events* is provided by `Cramp`.
- The `EventSource` interface is used to manage *server-sent events*. The server-sent event API is contained in the `EventSource` interface.
- To open a connection to the server to begin receiving events from it, you create a new `EventSource` object, specifying the URI of a script that generates the events:

```
var es = new EventSource('/stream')
```

- Next, we set up a handler for the `message` event.

When updates are pushed from the server, the `onmessage` handler fires and new data is available in its `e.data` property:

```
47 es.onmessage = function(e) { $('#chat').append(e.data + "\n") };
```

- The magical part is that whenever the connection is closed, the browser will automatically reconnect to the source after ~3 seconds. Your server implementation can even have control over this reconnection timeout.

- You can also call `addEventListener()` to listen for events just like any other event source:

```
source.addEventListener('message', function(e) {
 console.log(e.data);
}, false);

source.addEventListener('open', function(e) {
 // Connection was opened.
}, false);

source.addEventListener('error', function(e) {
 if (e.readyState == EventSource.CLOSED) {
 // Connection was closed.
 }
}, false);
```

- Código HTML generado:

```
<html>
 <head>
 <title>Super Simple Chat with Sinatra</title>
 <meta charset="utf-8" />
 <script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
 </head>
 <body><pre id='chat'></pre>

<script>
 // reading
 var es = new EventSource('/stream');
 es.onmessage = function(e) { $('#chat').append(e.data + "\n") };

 // writing
 $("form").live("submit", function(e) {
 $.post('/', {msg: "casiano: " + $('#msg').val()});
 $('#msg').val(''); $('#msg').focus();
 e.preventDefault();
 });
</script>

<form>
 <input id='msg' placeholder='type message here...'/>
</form></body>
</html>
```

- We start with the dollar sign and parentheses: `$("form")` to specify a selector. The dollar sign (`$`) is simply shorthand for jQuery. With `$("form")` we select all the elements with tag name `form`
- `.append( content [, content] )`

- `content`: DOM element, HTML string, or jQuery object to insert at the end of each element in the set of matched elements.
- `content`: One or more additional *DOM* elements, arrays of elements, HTML strings, or jQuery objects to insert at the end of each element in the set of matched elements.

- **.live( events, handler(eventObject) ) Returns: jQuery**

Description: Attach an event handler for all elements which match the current selector, now and in the future.

- **events:** A string containing a JavaScript event type, such as `click` or `keydown`. As of jQuery 1.4 the string can contain multiple, space-separated event types or custom event names.
- **handler(eventObject):** A function to execute at the time the event is triggered.
- As of jQuery 1.7, the `.live()` method is deprecated. Use `.on()` to attach event handlers.  
These are equivalent templates:

```
$(selector).live(events, data, handler); // jQuery 1.3+
$(document).on(events, selector, data, handler); // jQuery 1.7+
```

The `events` argument can either be a space-separated list of event type names and optional namespaces, or an object of event name strings and handlers. The `data` argument is optional and can be omitted.

- **jQuery.post( url [, data] [, success(data, textStatus, jqXHR)] [, dataType] )**

- **url:** A string containing the URL to which the request is sent.
- **data:** A map or string that is sent to the server with the request.
- **success(data, textStatus, jqXHR):** A callback function that is executed if the request succeeds.
- **dataType:** The type of data expected from the server. Default: Intelligent Guess (xml, json, script, text, html).

This is an easy way to send a simple POST request to a server. It allows a single callback function to be specified that will be executed when the request is complete (and only if the response has a successful response code).

`$.post()` returns the `XMLHttpRequest` that it creates. In most cases you won't need that object to manipulate directly, but it is available if you need to abort the request manually.

- **.val**

Get the input value of the first matched element.

A value is returned for all input elements, including selects and textareas. For multiple selects an array of values is returned.

For example:

```
$('.select.foo option:selected').val(); // get the value from a dropdown select
$('.select.foo').val(); // get the value from a dropdown select even easier
$('input:checkbox:checked').val(); // get the value from a checked checkbox
$('input:radio[name=bar]:checked').val(); // get the value from a set of radio buttons
```

- **.focus( handler(eventObject) )
 .focus( [eventData], handler(eventObject) )
 .focus()**

`handler(eventObject):` A function to execute each time the event is triggered. `eventData:` A map of data that will be passed to the event handler.

This method is a shortcut for `.trigger('focus')`. The focus event is sent to an element when it gains focus. This event is implicitly applicable to a limited set of elements, such as form elements (`<input>`, `<select>`, etc.) and links (`<a href>`). In recent browser versions, the event can be extended to include all element types by explicitly setting the element's `tabindex` property. An element can gain focus via keyboard commands, such as the Tab key, or by mouse clicks on the element. Elements with focus are usually highlighted in some way by the browser, for example with a dotted line surrounding the element. The focus is used to determine which element is the first to receive keyboard-related events.

- `event.preventDefault()` Returns: `undefined`

Description: If this method is called, the default action of the event will not be triggered.

For example, clicked anchors will not take the browser to a new URL. We can use `event.isDefaultPrevented()` to determine if this method has been called by an event handler that was triggered by this event.

### 38.6.4. Código Completo del Chat

```
[17:51] [~/srcSTW/streaming/chat_with_streaming(master)]$ cat chat.rb
coding: utf-8
require 'sinatra'
set server: 'thin', connections: []

get '/' do
 halt erb(:login) unless params[:user]
 erb :chat, locals: { user: params[:user].gsub(/\W/, '') }
end

get '/stream', provides: 'text/event-stream' do
 stream :keep_open do |out|
 settings.connections << out
 out.callback { settings.connections.delete(out) }
 end
end

post '/' do
 settings.connections.each { |out| out << "data: #{params[:msg]}\n\n" }
 204 # response without entity body
end

__END__

@@ layout
<html>
 <head>
 <title>Super Simple Chat with Sinatra</title>
 <meta charset="utf-8" />
 <script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
 </head>
 <body><%= yield %></body>
</html>

@@ login
<form action='/'>
 <label for='user'>User Name:</label>
```

```

<input name='user' value=' ' />
<input type='submit' value="GO!" />
</form>

@@ chat
<pre id='chat'></pre>

<script>
 // reading
 var es = new EventSource('/stream');
 es.onmessage = function(e) { $('#chat').append(e.data + "\n") };

 // writing
 $("form").live("submit", function(e) {
 $.post('/', {msg: "<%= user %>: " + $('#msg').val()});
 $('#msg').val(''); $('#msg').focus();
 e.preventDefault();
 });
</script>

<form>
 <input id='msg' placeholder='type message here...' />
</form>[

```

### 38.6.5. Chat Simple

#### Donde

- [~/sinatra-streaming/blazeeboychat(master)]\$ pwd -P  
 /Users/casiano/local/src/ruby/sinatra/sinatra-streaming/blazeeboychat  
 [~/sinatra-streaming/blazeeboychat(master)]\$ git remote -v  
 origin git@github.com:crguezl/chat-blazee.git (fetch)  
 origin git@github.com:crguezl/chat-blazee.git (push)
- <https://github.com/crguezl/chat-blazee>
- <https://gist.github.com/blazeeboy/9289678>

#### Código

```
[~/sinatra-streaming/blazeeboychat(master)]$ cat chat.rb
require 'sinatra' # gem install sinatra --no-rdoc --no-ri
set :port, 3000
set :environment, :production

html = <<-EOT
<html><head><style>
#text{width:100%; font-size: 15px; padding: 5px; display: block;}
</style></head><body>
 <input id="text" placeholder="Write then press Enter."/>
 <div id="chat"></div>
 <script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
 <script>
 $('#text').keypress(function(e){
 if(e.keyCode==13){

```

```

$.get('/send',{text:$('#text').val()});
$('#text').val('');
}
});
last = 0;
setInterval(function(){
$.get('/update',{last:last},
function(response){
last = $('<p>').html(response).find('span').data('last');
$('#chat').append(response);
});
},1000);
</script>
</body></html>
EOT

chat = ['welcome..']
get('/') { html }
get '/send' do
 chat << "#{request.ip} : #{params['text']}"
 nil
end
get '/update' do
 updates = chat[params['last'].to_i...-1]
 last = ""
 if updates.size>0
 updates.join('
') + "#{last}
"
 else
 last
 end
end

```

### 38.6.6. Práctica: Chat con Mensajes Individuales

Mejore el chat utilizando streaming y *server sent events* descrito en la sección 38.6.3 para que:

- Si el usuario escribe en su caja /nickname: mensaje ese mensaje sólo se entregue al usuario nickname

### 38.6.7. Práctica: Chat con Estilo

Mejore la vista del chat utilizando streaming y server sent events descrito en la sección 38.6.3 para que:

- Usando contenedores tablas u otros mecanismos, haga que en un marco salgan los nicks de los usuarios conectados, en otro la conversación y en un tercero el texto que se va a enviar.

Mejore la práctica para que puedan producirse conversaciones por pares.

Véase la rama `master` en <https://github.com/crguezl/sinatra-streaming-example-chat/>

### 38.6.8. Práctica: Chat con TicTacToe

Modifique la práctica anterior añadiendo un nuevo contenedor en la vista con el tablero del tictactoe.

1. Los usuarios tiene dos estados jugando (en rojo) o libres (en verde).

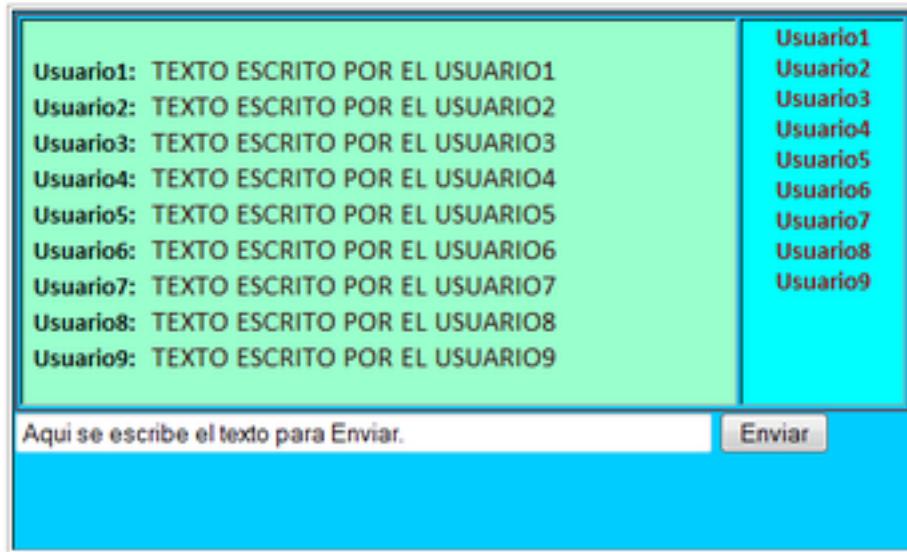


Figura 38.1: Típica disposición de un chat

2. Al cliquear en un usuario libre elegimos jugar con él. Se le mostrará a este la opción de aceptar o rechazar la oferta.
3. Si acepta se comenzará con el juego a dos del tictactoe actualizando el estado de estos usuarios.
4. Al finalizar la partida los jugadores pasan al estado de libres

### 38.7. Embedding Sinatra within EventMachine

Véase Sinatra Recipes. Embedding Sinatra within EventMachine.

Event::Machine is a very useful tool and sometimes you need to add a web-interface on top of it. Yes, EM does support this out of the box, but it can be ugly and hard to work with. Why not use something that everyone already knows and loves like Sinatra?

Below is a (working) code-sample for running a simple HelloWorld Sinatra app within Event::Machine. I've also provided a simple example of deferring tasks within your Sinatra call.

```
[~/sinatra/sinatra-eventmachine]$ cat em-sinatra-test.rb
require 'eventmachine'
require 'sinatra/base'
require 'thin'

This example shows you how to embed Sinatra into your EventMachine
application. This is very useful if you're application needs some
sort of API interface and you don't want to use EM's provided
web-server.

def run(opts)

 # Start the reactor
 EM.run do

 # define some defaults for our app
 server = opts[:server] || 'thin'
```

```

host = opts[:host] || '0.0.0.0'
port = opts[:port] || '8181'
web_app = opts[:app]

create a base-mapping that our application will set at. If I
have the following routes:
dispatch = Rack::Builder.app do
 map '/' do
 run web_app
 end
end

NOTE that we have to use an EM-compatible web-server. There
might be more, but these are some that are currently available.
unless ['thin', 'hatetepe', 'goliath'].include? server
 raise "Need an EM webserver, but #{server} isn't"
end

Start the web server. Note that you are free to run other tasks
within your EM instance.
Rack::Server.start({
 app: dispatch,
 server: server,
 Host: host,
 Port: port
})
end
end

Our simple hello-world app
class HelloApp < Sinatra::Base
 # threaded - False: Will take requests on the reactor thread
 # True: Will queue request for background thread
 configure do
 set :threaded, false
 end

 # Request runs on the reactor thread (with threaded set to false)
 get '/hello' do
 'Hello World'
 end

 # Request runs on the reactor thread (with threaded set to false)
 # and returns immediately. The deferred task does not delay the
 # response from the web-service.
 get '/delayed-hello' do
 EM.defer do
 sleep 5
 end
 'I\'m doing work in the background, but I am still free to take requests'
 end
end

```

```
start the application
run app: HelloApp.new
```

You can run this simply with the command:

```
ruby em-sinatra-test.rb # em-sinatra-test.rb is the filename of the above-code
```

You should also be able to test that it is working correctly with the following `ab` command:

```
ab -c 10 -n 100 http://localhost:8181/delayed-hello
```

*ApacheBench* (`ab`) is a single-threaded command line computer program for measuring the performance of HTTP web servers. Originally designed to test the Apache HTTP Server, it is generic enough to test any web server (Véase <http://httpd.apache.org/docs/2.2/programs/ab.html>).

If this finishes in `zero point something` seconds, then you have successfully setup Sinatra to run within EM and you are taking requests on the event-loop and deferring tasks to the background.

If it takes any longer than that, then you are most likely taking requests in the background which means when the EM queue fills up, you can't process your sinatra requests (not a good thing!). Make sure that you have `threaded` set to `false` and then try again.

Here is an execution:

```
[~/sinatra/sinatra-eventmachine]$ ab -c 10 -n 100 http://localhost:8181/delayed-hello
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
Benchmarking localhost (be patient).....done
```

```
Server Software: thin
Server Hostname: localhost
Server Port: 8181
```

```
Document Path: /delayed-hello
Document Length: 70 bytes
```

```
Concurrency Level: 10
Time taken for tests: 0.096 seconds
Complete requests: 100
Failed requests: 0
Write errors: 0
Total transferred: 30300 bytes
HTML transferred: 7000 bytes
Requests per second: 1043.96 [#/sec] (mean)
Time per request: 9.579 [ms] (mean)
Time per request: 0.958 [ms] (mean, across all concurrent requests)
Transfer rate: 308.91 [Kbytes/sec] received
```

#### Connection Times (ms)

	min	mean	[+/-sd]	median	max
Connect:	0	0	0.1	0	1
Processing:	2	9	11.6	5	44
Waiting:	1	8	11.7	5	43
Total:	2	9	11.6	6	44

```

Percentage of the requests served within a certain time (ms)
 50% 6
 66% 6
 75% 7
 80% 7
 90% 44
 95% 44
 98% 44
 99% 44
100% 44 (longest request)

```

Véase

1. Asynchronous responses in Rack por Patrick

## 38.8. Ejemplo de Server Sent Events: irb en el navegador

Véase

1. Real Time Rack presentation by Konstantin Haase en GitHub using Scott Chacon showoff
2. The corresponding talk Real Time Rack”by Konstantin Haase 2011. Confreaks videos.
3. La sección *Showoff* 106 en estos apuntes

Este código se encuentra en: <https://github.com/rkh/presentations/blob/realtime-rack/example.rb>

El javascript y el server se activan desde las trasparencias. En concreto en esta trasparencia que está en el fichero `slides/01_slides.md`:

```
!SLIDE center
```

```
Demo!
```

```
<iframe src="/events?" width="980" height="600"></iframe>
```

```
.notes Next: Rack
```

The `<iframe>` tag specifies an inline frame. An inline frame is used to embed another document within the current HTML document.

`src="/events?"` hace que se dispare el código correspondiente a la ruta `/events` descrita en el fichero `example.rb`.

La ruta `/events` está en el fichero `example.rb`:

```
get('/events') { slim :html }
```

El fichero `example.rb` es cargado desde el `config.ru`:

```
[~/local/src/ruby/sinatra/sinatra-streaming/konstantin_haase/presentations(realtime-rack)]$ cat config.ru
require 'showoff'
require './example'

use Example
run ShowOff
```

Obsérvese que es cargado por `config.ru` mediante `use`.

El template `html` contiene:

```

@@ html
html
 head
 title brirb
 link href="/events.css" rel="stylesheet" type="text/css"
 script src="/jquery.min.js" type="text/javascript"
 script src="/events.js" type="text/javascript"
 body
 #log
 form#form
 | >>
 input#input type='text'

```

Como vemos tenemos identificadores `log`, `form` y `input` para hablar de los correspondientes elementos implicados.

La carga de `/events.js` es también manejado por una ruta:

```
get('/events.js') { coffee :script }
```

La gema coffee-script provee el mecanismo para compilar el javascript y producir el correspondiente código JavaScript.

Este es el CoffeeScript contenido en el template `script`:

```

$(document).ready ->
 input = $("#input")
 log = $("#log")
 history = []
 count = 0
 output = (str) ->
 log.append str
 log.append "
"
 input.attr scrollTop: input.attr("scrollHeight")

 input.bind "keydown", (e) ->
 if e.keyCode == 38 or e.keyCode == 40
 count += e.keyCode - 39
 count = 0 if count < 0
 count = input.length + 1 if count > input.length
 input.val history[count]
 false
 else
 true

 $("#form").live "submit", (e) ->
 value = input.val()
 history.push value
 count++
 $.post '/run', code: input.val()
 output ">> #{value}"
 input.val ""
 input.focus()
 e.preventDefault()

src = new EventSource('/events.es')

```

```
src.onmessage = (e) -> output e.data
```

1. La llamada `output(str)` añade en el punto indicado por `#log` el texto `str`. Además se encarga del scrolling:

```
output = (str) ->
 log.append str
 log.append "
"
 input.attr scrollTop: input.attr("scrollHeight")
```

2. El método JQuery `.bind( eventType [, eventData ], handler(eventObject) )` Attaches a handler to an event for the elements. En este caso `eventType` es `keydown`.

3. La llamada:

```
src = new EventSource('/events.es')
```

Hace que nos suscribamos a los mensajes generados por `/events.es`

4. Cada vez que llega un mensaje lo volcamos en la página mediante `output`:

```
src.onmessage = (e) -> output e.data
```

5. Javascript Char Codes (Key Codes)

6. Creo que el código de la flecha arriba es 38 y el de abajo 40. Así pues lo que se suma a `count` es 1 o -1. Parece que navegamos en el histórico de comandos de esta forma:

```
input.bind "keydown", (e) ->
 if e.keyCode == 38 or e.keyCode == 40
 count += e.keyCode - 39
 count = 0 if count < 0
 count = input.length + 1 if count > input.length
 input.val history[count]
 false
 else
 true
```

7. Cuando introducimos nuestra expresión en el formulario y pulsamos retorno de carro se ejecuta la correspondiente callback. Mediante `$.post '/run', code: input.val()` enviamos al servidor la petición de que evalúe la entrada:

```
$("#form").live "submit", (e) ->
 value = input.val()
 history.push value
 count++
 $.post '/run', code: input.val()
 output ">> #{value}"
 input.val ""
 input.focus()
 e.preventDefault()
```

8. The `live` method attaches an event handler for all elements which match the current selector, now and in the future.

La petición es recibida en la correspondiente ruta

```

post '/run' do
 begin
 result = nil
 stdout = capture_stdout do
 result = eval("_ = (#{params[:code]})", settings.scope, "(irb)", settings.line)
 settings.line += 1
 end
 stdout << "=> " << result.inspect
 rescue Exception => e
 stdout = [e.to_s, *e.backtrace.map { |l| "\t#{l}" }].join("\n")
 end
 source = escape stdout
 Scope.send source
 ,
end

```

- a) El método `eval` tiene estos argumentos:

`eval(string [, binding [, filename [,lineno]]])`

- 1) Evaluates the Ruby expression(s) in string.
- 2) `binding` is a `Binding` object: the evaluation is performed in its context.
- 3) `filename` and `lineno` are used when reporting syntax errors.

- b) El método `capture_stdout` nos permite capturar la salida por `stdout` de una evaluación:

```
[~/Chapter6MethodsProcsLambdasAndClosures]$ pry
[1] pry(main)> require 'capture_stdout'
=> true
[2] pry(main)> string = 'yeah'
=> "yeah"
[3] pry(main)> output = capture_stdout { print(string) }
=> "yeah"
```

9. En el módulo `Scope` se define el método `Scope.send` el cual envía a todos los clientes el mensaje especificado:

```

module Scope
 def self.send(*args)
 Example.subscribers.each { |s| s.send(*args) }
 end

 def self.puts(*args)
 args.each { |str| send str.to_s }
 nil
 end

 def self.binding
 Kernel.binding
 end
end

```

- a) El método `send` recorre el array `subscribers` que es un array de objetos `EventSource` y delega en el método `send` del subscriptor el envío de los datos en `*args`
- b) El método `binding` delega en el correspondiente método del `Kernel`. El método es usado para guardar el binding en la variable `:scope` en la clase `Example`:

```

class Example < Sinatra::Base
 enable :inline_templates, :logging, :static
 set :public, File.expand_path('../public', __FILE__)
 set :subscribers => [], :scope => Scope.binding, :line => 1

```

y es posteriormente usado cuando se evalúa la expresión:

```

stdout = capture_stdout do
 result = eval("_ = (#{params[:code]})", settings.scope, "(irb)", settings.line)
 settings.line += 1 # número de línea
end

```

10. `Example.subscribers` es un array que es inicializado al comienzo de la clase `Example`:

```

class Example < Sinatra::Base
 enable :inline_templates, :logging, :static
 set :public_folder, File.expand_path('../public', __FILE__)
 set :subscribers => [], :scope => Scope.binding, :line => 1
 ...

```

11. `subscribers` se actualiza en el código asociado con la ruta `events.es` que es visitada - desde el código CoffeeScript - cada vez que se carga una nueva página:

```

$(document).ready ->
 input = $("#input")
 ...
 output = (str) ->
 ...
 input.bind "keydown", (e) ->
 ...
 $("#form").live "submit", (e) ->
 ...
 ...
src = new EventSource('/events.es')
src.onmessage = (e) -> output e.data

```

12. Este es el código de la ruta `events.es`:

```

get '/events.es' do
 content_type request.preferred_type("text/event-stream", "text/plain")
 body EventSource.new
 settings.subscribers << body
 EM.next_tick { env['async.callback'].call response.finish }
 throw :async
end

```

- a) Como se ha mencionado rack espera que el cuerpo de la respuesta sea un objeto que disponga de un método `each`. Con la llamada `body EventSource.new` establecemos el cuerpo de la respuesta. La definición de la clase `EventSource` aparece en el item 13
- b) El método

```
(Object) next_tick(pr = nil, &block)
```

Schedules a `Proc` for execution immediately after the next turn through the reactor core. An advanced technique, this can be useful for improving memory management and/or application responsiveness, especially when scheduling large amounts of data for writing to a network connection.

This method takes either a single argument (which must be a callable object) or a block.

Parameters:

```
pr (#call) (defaults to: nil) | A callable object to run
```

Raises:

```
(ArgumentError)
```

- c) El siguiente texto esta tomado de Asynchronous responses in Rack por Patrick April 15, 2012.

While there is not yet an async interface in the Rack specification, several Rack servers have implemented James Tucker's async scheme.

Rather than returning [status, headers, body], the app returns a status of -1, or throws the symbol :async.

The server provides env['async.callback'] which the app saves and later calls with the usual [status, headers, body] to send the response.

Note: returning a status of -1 is illegal as far as Rack::Lint is concerned. throw :async is not flagged as an error.

```
class AsyncApp
 def call(env)
 Thread.new do
 sleep 5 # simulate waiting for some event
 response = [200, {'Content-Type' => 'text/plain'}, ['Hello, World!']]
 env['async.callback'].call response
 end

 [-1, {}, []] # or throw :async
 end
end
```

In the example above, the request is suspended, nothing is sent back to the client, the connection remains open, and the client waits for a response.

The app returns the special status, and the worker process is able to handle more HTTP requests (i.e. it is not blocked). Later, inside the thread, the full response is prepared and sent to the client.

- d) Véase en StakOverflow la pregunta: Rack concurrency - rack.multithread, async.callback, or both?

There is another, more oft discussed means of achieving concurrency, involving EventMachine.defer and throw :async. Strictly speaking, requests are not handled using threads. They are dealt with serially, but pass their heavy lifting and a callback off to EventMachine, which uses async.callback to send a response at a later time. After request A has offloaded its work to EM.defer, request B is begun. Is this correct?

Respuesta de Konstantin:

Using `async.callback` in conjunction with `EM.defer` actually makes not too much sense, as it would basically use the thread-pool, too, ending up with a similar construct as described in Q1.

Using `async.callback` makes sense when only using eventmachine libraries for IO. Thin will send the response to the client once `env['async.callback']` is called with a normal Rack response as argument.

If the body is an `EM::Deferrable`, Thin will not close the connection until that deferrable succeeds.

A rather well kept secret: If you want more than just long polling (i.e. keep the connection open after sending a partial response), you can also return an

EM::Deferrable as body object directly without having to use throw :async or a status code of -1.

13. Un objeto EventSource tiene métodos each y send:

```
class EventSource
 include EventMachine::Deferrable

 def send(data, id = nil)
 data.each_line do |line|
 line = "data: #{line.strip}\n"
 @body_callback.call line
 end
 @body_callback.call "id: #{id}\n" if id
 @body_callback.call "\n"
 end

 def each(&blk)
 @body_callback = blk
 end
end
```

### example.rb

```
[~/sinatra/sinatra-streaming/konstantin_haase/presentations(realtime-rack)]$ cat example.rb
require 'sinatra/base'
require 'capture_stdout'
require 'escape_utils'
require 'slim'
require 'sass'
require 'coffee-script'
require 'eventmachine'

class EventSource
 include EventMachine::Deferrable

 def send(data, id = nil)
 data.each_line do |line|
 line = "data: #{line.strip}\n"
 @body_callback.call line
 end
 @body_callback.call "id: #{id}\n" if id
 @body_callback.call "\n"
 end

 def each(&blk)
 @body_callback = blk
 end
end

module Scope
 def self.send(*args)
 Example.subscribers.each { |s| s.send(*args) }
 end
end
```

```

def self.puts(*args)
 args.each { |str| send str.to_s }
 nil
end

def self.binding
 Kernel.binding
end
end

class Example < Sinatra::Base
 enable :inline_templates, :logging, :static
 set :public_folder, File.expand_path('../public', __FILE__)
 set :subscribers => [], :scope => Scope.binding, :line => 1

 def escape(data)
 EscapeUtils.escape_html(data).gsub("\n", "
").
 gsub("\t", " ").gsub(" ", " ")
 end

 get '/events.es' do
 content_type request.preferred_type("text/event-stream", "text/plain")
 body EventSource.new
 settings.subscribers << body
 EM.next_tick { env['async.callback'].call response.finish }
 throw :async
 end

 get('/events') { slim :html }
 get('/events.js') { coffee :script }
 get('/events.css') { sass :style }

 post '/run' do
 begin
 result = nil
 stdout = capture_stdout do
 result = eval("_ = (#{params[:code]})", settings.scope, "(irb)", settings.line)
 settings.line += 1
 end
 stdout << "=> " << result.inspect
 rescue Exception => e
 stdout = [e.to_s, *e.backtrace.map { |l| "\t#{l}" }].join("\n")
 end
 source = escape stdout
 Scope.send source
 ''
 end
 end
end

__END__

@@ script

```

```

$(document).ready ->
 input = $("#input")
 log = $("#log")
 history = []
 count = 0
 output = (str) ->
 log.append str
 log.append "
"
 input.attr scrollTop: input.attr("scrollHeight")

 input.bind "keydown", (e) ->
 if e.keyCode == 38 or e.keyCode == 40
 count += e.keyCode - 39
 count = 0 if count < 0
 count = input.length + 1 if count > input.length
 input.val history[count]
 false
 else
 true

 $("#form").live "submit", (e) ->
 value = input.val()
 history.push value
 count++
 $.post '/run', code: input.val()
 output ">> #{value}"
 input.val ""
 input.focus()
 e.preventDefault()

src = new EventSource('/events.es')
src.onmessage = (e) -> output e.data

@@ html
html
 head
 title brirb
 link href="/events.css" rel="stylesheet" type="text/css"
 script src="/jquery.min.js" type="text/javascript"
 script src="/events.js" type="text/javascript"
 body
 #log
 form#form
 >>
 input#input type='text'

@@ style
body
 font:
 size: 200%
 family: monospace
 input#input

```

```
font-size: 100%
font-family: monospace
border: none
padding: 0
margin: 0
width: 80%
&:focus
 border: none
 outline: none
```

## showoff.json

```
[~/local/src/ruby/sinatra/sinatra-streaming/konstantin_haase/presentations(realtime-rack)]$ cat showoff.json
{
 "name": "Real Time Rack",
 "sections": [
 { "section": "intro" },
 { "section": "slides" },
 { "section": "outro" }
]
}
```

## slides/01\_slides.md

```
[~/local/src/ruby/sinatra/sinatra-streaming/konstantin_haase/presentations(realtime-rack)]$ cat slides/01_slides.md
!SLIDE bullets

* ! [breaking] (breaking.png)

.notes Next: Warning

!SLIDE bullets incremental

Warning
* There will be a lot of code ...
* A lot!
* Also, this is the *Special Extended Director's Cut*!

.notes Next: good old web

!SLIDE center
![web] (ie.png)

.notes Next: ajax

!SLIDE center
![ajax] (ajax.png)

.notes Next: Comet

!SLIDE center
![comet] (comet.png)

.notes Next: Real Time
```

```

!SLIDE bullets

* ! [real_time] (real_time.jpg)

.notes Next: come again?

!SLIDE bullets incremental

Come again?

* streaming
* server push

.notes streaming, server push. --- Next: decide what to send while streaming, not upfront

!SLIDE bullets

* decide what to send while streaming, not upfront

.notes Next: usage example

!SLIDE bullets

* Streaming APIs
* Server-Sent Events
* Websockets

.notes Next: demo

!SLIDE center

Demo!

<iframe src="/events?" width="980" height="600"></iframe>

.notes Next: Rack

!SLIDE bullets incremental

Rack

* Ruby to HTTP to Ruby bridge
* Middleware API
* Powers Rails, Sinatra, Ramaze, ...

.notes HTTP bridge, middleware, frameworks. --- Next: rack stack

!SLIDE center

![rack] (rack_stack.png)

.notes Next: simple rack app

```

!SLIDE smallish

![working\_code](working\_code.png)  
![stack](endpoint.png)

```
@@@ ruby
welcome_app = proc do |env|
 [200, {'Content-Type' => 'text/html'},
 ['Welcome!']]
end
```

.notes Next: with any object

!SLIDE smallish

![working\_code](working\_code.png)  
![stack](endpoint.png)

```
@@@ ruby
welcome_app = Object.new

def welcome_app.call(env)
 [200, {'Content-Type' => 'text/html'},
 ['Welcome!']]
end
```

.notes Next: in sinatra

!SLIDE

![working\_code](working\_code.png)  
![stack](endpoint.png)

```
@@@ ruby
get('/') { 'Welcome!' }
```

.notes Next: pseudo handler

!SLIDE smallish

![pseudo\_code](pseudo\_code.png)  
![stack](handler.png)

```
@@@ ruby
env = parse_http

status, headers, body =
welcome_app.call env

io.puts "HTTP/1.1 #{status}"
headers.each { |k,v| io.puts "#{k}: #{v}" }
io.puts ""
```

```

body.each { |str| io.puts str }

close_connection

.notes Next: middleware

!SLIDE smallish

Middleware

.notes Next: uppercase example

!SLIDE smallish

! [working_code] (working_code.png)
! [stack] (middleware.png)

@@@ ruby
foo => FOO
class UpperCase
 def initialize(app)
 @app = app
 end

 def call(env)
 status, headers, body = @app.call(env)
 upper = []
 body.each { |s| upper << s.upcase }
 [status, headers, upper]
 end
end

.notes Next: config.ru

!SLIDE large

! [working_code] (working_code.png)
! [stack] (something_else.png)

@@@ ruby
set up middleware
use UpperCase

set endpoint
run welcome_app

.notes Next: call app (from before)

!SLIDE

! [working_code] (working_code.png)
! [stack] (handler.png)

```

```
@@@ ruby
status, headers, body =
welcome_app.call(env)

.notes Next: wrap in middleware

!SLIDE smallish

![working_code](working_code.png)
![stack](handler.png)

@@@ ruby
app = UpperCase.new(welcome_app)

status, headers, body = app.call(env)
```

.notes Next: streaming with each

```
!SLIDE
Streaming with #each
```

.notes Next: custom body object

!SLIDE smallish

```
![working_code](working_code.png)
![stack](handler.png)
```

```
@@@ ruby
my_body = Object.new
get('/') { my_body }

def my_body.each
 20.times do
 yield "<p>%s</p>" % Time.now
 sleep 1
 end
end
```

.notes Next: Let's build a messaging service!

!SLIDE bullets

- \* Let's build a messaging service!

.notes Next: sinatra app

!SLIDE smallish

```
![working_code](working_code.png)
![stack](endpoint.png)
```

```

@@@ ruby
subscribers = []

get '/' do
 body = Subscriber.new
 subscribers << body
 body
end

post '/' do
 subscribers.each do |s|
 s.send params[:message]
 end
end

```

.notes Next: subscriber object

!SLIDE smallish

![working\_code](working\_code.png)  
![stack](endpoint.png)

```

@@@ ruby
class Subscriber
 def send(data)
 @data = data
 @thread.wakeup
 end

 def each
 @thread = Thread.current
 loop do
 yield @data.to_s
 sleep
 end
 end
end

```

.notes Next: issues with this

!SLIDE bullets incremental

- \* blocks the current thread
- \* does not work well with some middleware
- \* does not work (well) on evented servers <br> (Thin, Goliath, Ebb, Rainbows!)

.notes blocks, middleware, evented servers. --- Next: evented streaming

!SLIDE

```
Evented streaming with async.callback
```

.notes Next: event loop graphics

```

!SLIDE center
![event loop] (eventloop1.png)

.notes Next: webscale

!SLIDE center
![event loop - webscale] (eventloop2.png)

.notes Next: without eventloop

!SLIDE

![working_code] (working_code.png)
![stack] (something_else.png)

@@@ ruby
sleep 10
puts "10 seconds are over"

puts Redis.new.get('foo')

.notes Next: with eventloop

!SLIDE smallish

![working_code] (working_code.png)
![stack] (something_else.png)

@@@ ruby
require 'eventmachine'

EM.run do
 EM.add_timer 10 do
 puts "10 seconds are over"
 end

 redis = EM::Hiredis.connect
 redis.get('foo').callback do |value|
 puts value
 end
end

.notes Next: async.callback

!SLIDE smallish

![pseudo_code] (pseudo_code.png)
![stack] (endpoint.png)

@@@ ruby
get '/' do
 EM.add_timer(10) do

```

```

 env['async.callback'].call [200,
 {'Content-Type' => 'text/html'},
 ['sorry you had to wait']]
 end

 "dear server, I don't have a " \
 "response yet, please wait 10 " \
 "seconds, thank you!"
end

.notes Next: throw

!SLIDE smallish

With #throw

![working_code] (working_code.png)
![stack] (endpoint.png)

@@@ ruby
get '/' do
 EM.add_timer(10) do
 env['async.callback'].call [200,
 {'Content-Type' => 'text/html'},
 ['sorry you had to wait']]
 end

 # will skip right to the handler
 throw :async
end

.notes Next: -1

!SLIDE smallish

Status Code

![working_code] (working_code.png)
![stack] (endpoint.png)

@@@ ruby
get '/' do
 EM.add_timer(10) do
 env['async.callback'].call [200,
 {'Content-Type' => 'text/html'},
 ['sorry you had to wait']]
 end

 # will go through middleware
 [-1, {}, []]
end

.notes Next: async-sinatra

```

```
!SLIDE smallish
```

```
![working_code] (working_code.png)
![stack] (endpoint.png)
```

```
@@@ ruby
gem install async-sinatra
require 'sinatra/async'

aget '/' do
 EM.add_timer(10) do
 body 'sorry you had to wait'
 end
end
```

```
.notes Next: with redis
```

```
!SLIDE smallish
```

```
![working_code] (working_code.png)
![stack] (endpoint.png)
```

```
@@@ ruby
redis = EM::Hiredis.connect

aget '/' do
 redis.get('foo').callback do |value|
 body value
 end
end
```

```
.notes Next: pseudo handler with callback
```

```
!SLIDE smallish
```

```
![pseudo_code] (pseudo_code.png)
![stack] (handler.png)
```

```
@@@ ruby
env = parse_http

cb = proc do |response|
 send_headers(response)
 response.last.each { |s| send_data(s) }
 close_connection
end

catch(:async) do
 env['async.callback'] = cb
 response = app.call(env)
 cb.call(response) unless response[0] == -1
end
```

```
.notes Next: postponing, not streaming
```

```
!SLIDE bullets incremental
```

- \* that's postponing ...
- \* ... not streaming

```
.notes Next: EM::Deferrable
```

```
!SLIDE
```

```
EM::Deferrable
```

```
.notes Next: Deferrable explained
```

```
!SLIDE smallish
```

```
![working_code] (working_code.png)
![stack] (something_else.png)
```

```
@@@ ruby
require 'eventmachine'

class Foo
 include EM::Deferrable
end

EM.run do
 f = Foo.new
 f.callback { puts "success!" }
 f.errback { puts "something went wrong" }
 f.succeed
end
```

```
.notes Next: pseudo handler - callback from before
```

```
!SLIDE smallish
```

```
![pseudo_code] (pseudo_code.png)
![stack] (handler.png)
```

```
@@@ ruby
cb = proc do |response|
 send_headers(response)
 response.last.each { |s| send_data(s) }
 close_connection
end
```

```
.notes Next: pseudo handler - new callback
```

```
!SLIDE smallish
```

```
! [pseudo_code] (pseudo_code.png)
! [stack] (handler.png)
```

```
@@@ ruby
cb = proc do |response|
 send_headers(response)
 body = response.last
 body.each { |s| send_data(s) }

 if body.respond_to? :callback
 body.callback { close_connection }
 body.errback { close_connection }
 else
 close_connect
 end
end
```

```
.notes Next: Evented Messaging System
```

```
!SLIDE
```

```
Evented Messaging System
```

```
.notes Next: old messaging system
```

```
!SLIDE smallish
```

```
! [working_code] (working_code.png)
! [stack] (endpoint.png)
```

```
@@@ ruby
THIS IS NOT EVENTED

subscribers = []

get '/' do
 body = Subscriber.new
 subscribers << body
 body
end

post '/' do
 subscribers.each do |s|
 s.send params[:message]
 end
end
```

```
.notes Next: new messaging system (sinatra app)
```

```
!SLIDE smallish
```

```
! [working_code] (working_code.png)
! [stack] (endpoint.png)
```

```

@@@ ruby
subscribers = []

aget '/' do
 body Subscriber.new
 subscribers << body
end

post '/' do
 subscribers.each do |s|
 s.send params[:message]
 end
end

```

.notes Next: new subscriber class

!SLIDE smallish

![working\_code](working\_code.png)  
![stack](endpoint.png)

```

@@@ ruby
class Subscriber
 include EM::Deferrable

 def send(data)
 @body_callback.call(data)
 end

 def each(&blk)
 @body_callback = blk
 end
end

```

.notes Next: callback again

!SLIDE smallish

![pseudo\_code](pseudo\_code.png)  
![stack](handler.png)

```

@@@ ruby
cb = proc do |response|
 send_headers(response)
 body = response.last
 body.each { |s| send_data(s) }

 if body.respond_to? :callback
 body.callback { close_connection }
 body.errback { close_connection }
 else
 close_connect
 end
end

```

```
 end
 end

.notes Next: new subscriber class (again)
```

!SLIDE smallish

```
! [working_code] (working_code.png)
! [stack] (endpoint.png)
```

```
@@@ ruby
class Subscriber
 include EM::Deferrable

 def send(data)
 @body_callback.call(data)
 end

 def each(&blk)
 @body_callback = blk
 end
end
```

```
.notes Next: delete subscribers
```

!SLIDE smallish

```
! [working_code] (working_code.png)
! [stack] (endpoint.png)
```

```
@@@ ruby
delete '/' do
 subscribers.each do |s|
 s.send "Bye bye!"
 s.succeed
 end

 subscribers.clear
end
```

```
.notes Next: Server-Sent Events
```

!SLIDE bullets

```
Server-Sent Events
```

```
* [dev.w3.org/html5/eventsource] (http://dev.w3.org/html5/eventsource/)
```

```
.notes Next: explained
```

!SLIDE bullets incremental

```
* Think one-way WebSockets
```

- \* Simple
- \* Resumable
- \* Client can be implemented in JS
- \* Degrade gracefully to polling

.notes one-way WS, simple, resumable, client in JS, degrade --- Next: js code

!SLIDE smallish

![working\_code](working\_code.png)  
![stack](client.png)

```
@@@ javascript
var source = new EventSource('/updates');

source.onmessage = function (event) {
 alert(event.data);
};
```

.notes Next: HTTP headers

!SLIDE

HTTP/1.1 200 OK  
Content-Type: text/event-stream

.notes Next: HTTP headers + 1

!SLIDE

HTTP/1.1 200 OK  
Content-Type: text/event-stream  
  
data: This is the first message.

.notes Next: HTTP headers + 2

!SLIDE

HTTP/1.1 200 OK  
Content-Type: text/event-stream  
  
data: This is the first message.  
  
data: This is the second message, it  
data: has two lines.

.notes Next: HTTP headers + 3

!SLIDE

HTTP/1.1 200 OK  
Content-Type: text/event-stream

```
data: This is the first message.

data: This is the second message, it
data: has two lines.

data: This is the third message.
```

.notes Next: with IDs

!SLIDE

HTTP/1.1 200 OK  
Content-Type: text/event-stream

```
data: the client
id: 1

data: keeps track
id: 2

data: of the last id
id: 3
```

.notes Next: EventSource in Ruby

!SLIDE smallish

![working\_code](working\_code.png)  
![stack](endpoint.png)

```
@@@ ruby
class EventSource
 include EM::Deferrable

 def send(data, id = nil)
 data.each_line do |line|
 line = "data: #{line.strip}\n"
 @body_callback.call line
 end
 @body_callback.call "id: #{id}\n" if id
 @body_callback.call "\n"
 end

 def each(&blk)
 @body_callback = blk
 end
end
```

.notes Next: WebSockets

!SLIDE bullets

```

WebSockets

* Think two-way EventSource

.notes Next: JS WebSockets

!SLIDE smallish

![working_code](working_code.png)
![stack](client.png)

@@@ javascript
var src = new WebSocket('ws://127.0.0.1/');

src.onmessage = function (event) {
 alert(event.data);
};

.notes Next: JS EventSource

!SLIDE smallish

![working_code](working_code.png)
![stack](client.png)

@@@ javascript
var src = new EventSource('/updates');

src.onmessage = function (event) {
 alert(event.data);
};

.notes Next: JS WebSocket

!SLIDE smallish

![working_code](working_code.png)
![stack](client.png)

@@@ javascript
var src = new WebSocket('ws://127.0.0.1/');

src.onmessage = function (event) {
 alert(event.data);
};

.notes Next: JS WebSocket with send

!SLIDE smallish

![working_code](working_code.png)
![stack](client.png)

```

```
@@@ javascript
var src = new WebSocket('ws://127.0.0.1/');

src.onmessage = function (event) {
 alert(event.data);
};

src.send("ok, let's go");
```

.notes Next: Ruby WebSocket

!SLIDE smallish

! [working\_code] (working\_code.png)  
! [stack] (something\_else.png)

```
@@@ ruby
options = { host: '127.0.0.1', port: 8080 }
EM::WebSocket.start(options) do |ws|
 ws.onmessage { |msg| ws.send msg }
end
```

.notes Next: WebSockets are hard to use

!SLIDE bullets incremental

# WebSockets are hard to use #

- \* Protocol upgrade (not vanilla HTTP)
- \* Specification in flux
- \* Client support incomplete
- \* Proxies/Load Balancers have issues
- \* Rack can't do it

.notes Protocol upgrade, in flux, client support, proxies, rack --- Next: sinatra streaming

!SLIDE bullets

# Sinatra Streaming API #

\* introduced in Sinatra 1.3

.notes Next: example

!SLIDE smallish

! [working\_code] (working\_code.png)  
! [stack] (endpoint.png)

```
@@@ ruby
get '/' do
 stream do |out|
 out << "It's gonna be legen -\n"
```

```

 sleep 0.5
 out << " (wait for it) \n"
 sleep 1
 out << "- dary!\n"
 end
end

.notes Next: keep open

!SLIDE smallish

![working_code] (working_code.png)
![stack] (endpoint.png)

@@@ ruby
connections = []

get '/' do
 # keep stream open
 stream(:keep_open) do |out|
 connections << out
 end
end

post '/' do
 # write to all open streams
 connections.each do |out|
 out << params[:message] << "\n"
 end
 "message sent"
end

```

.notes Next: sinatra chat  
!SLIDE bullets

- \* Let's build a Chat!
- \* Code: [gist.github.com/1476463] (<https://gist.github.com/1476463>)
- \* Demo: [sharp-night-9421.herokuapp.com] (<http://sharp-night-9421.herokuapp.com/>)

.notes Next: go there now

!SLIDE bullets

- \* Yes, go there now!
- \* Here's the link again:<br>[\*\*sharp-night-9421.herokuapp.com\*\*] (<http://sharp-night-9421.herokuapp.com/>)
- \* Yes, there is no CSS. Sorry.

.notes Next: demo

!SLIDE

## [\*\*sharp-night-9421.herokuapp.com\*\*] (<http://sharp-night-9421.herokuapp.com/>)  
<iframe src="http://sharp-night-9421.herokuapp.com/?showoff=1" width="980" height="600"></iframe>

```
.notes Next: code

!SLIDE small

Ruby Code

@@@ ruby
set server: 'thin', connections: []

get '/stream', provides: 'text/event-stream' do
 stream :keep_open do |out|
 settings.connections << out
 out.callback { settings.connections.delete(out) }
 end
end

post '/' do
 settings.connections.each { |out| out << "data: #{params[:msg]}\n\n" }
 204 # response without entity body
end

JavaScript Code

@@@ javascript
var es = new EventSource('/stream');
es.onmessage = function(e) { $('#chat').append(e.data) };

$("form").live("submit", function(e) {
 $.post('/', {msg: "<%= params[:user] %>: " + $('#msg').val()});
 e.preventDefault();
});

HTML

@@@ html
<pre id='chat'></pre> <form><input id='msg' /></form>

Code: [**gist.github.com/1476463**] (https://gist.github.com/1476463) -
Demo: [**sharp-night-9421.herokuapp.com**] (http://sharp-night-9421.herokuapp.com/)
```

.notes Next: javascript

!SLIDE small

.notes Next: done

### 38.9. Asynchronous responses in Rack

Asynchronous responses in Rack

With the Rack synchronous interface protocol, the entire body must be prepared immediately, or be otherwise quickly available, for example by reading from file. A response that must be waited upon will tie up the process or thread executing the HTTP request. In a multi-process Rack server such as

Unicorn, this will block the entire worker process, making it unavailable to serve other requests.

Some Rack servers provide an alternate interface that allows the request to be suspended, unblocking the worker. At some later time, the request may be resumed, and the response sent to the client.

While there is not yet an async interface in the Rack specification, several Rack servers have implemented James Tucker's async scheme.

Rather than returning `[status, headers, body]`, the app returns a status of `-1`, or throws the symbol `:async`.

The server provides `env['async.callback']` which the app saves and later calls with the usual `[status, headers, body]` to send the response.

Note: returning a status of `-1` is illegal as far as `Rack::Lint` is concerned. `throw :async` is not flagged as an error.

```
class AsyncApp
 def call(env)
 Thread.new do
 sleep 5 # simulate waiting for some event
 response = [200, {'Content-Type' => 'text/plain'}, ['Hello, World!']]
 env['async.callback'].call response
 end

 [-1, {}, []] # or throw :async
 end
end
```

In the example above, the request is suspended, nothing is sent back to the client, the connection remains open, and the client waits for a response.

The app returns the special status, and the worker process is able to handle more HTTP requests (i.e. it is not blocked). Later, inside the thread, the full response is prepared and sent to the client.

### 38.9.1. Deferred or streaming response bodies

The example shows a one-shot request, wait, response cycle. But it is possible to have multiple wait, response segments to allow the headers to be sent to the client immediately, or the body to trickle in slowly without blocking the worker process.

Async Rack servers, when `env['async.callback']` is called, send the status and headers to the client and then begin iterating through each part of the body with `#each`.

After the last body part the server must decide if the connection to the client should be closed (entire body has been provided) or if it should remain open (body parts will be provided later). The details of this decision are implementation-specific. For now, assume the connection is not closed. To send additional body parts, `env['async.callback']` may not be called a second time since the status code and headers have already been sent to the client and can not be changed. The app takes advantage of the server's iteration through the body with `#each`:

the server calls `body.each(&block)`, and the trick is to save `&block` for later use. This turns the iteration inside-out: rather than the sever iterating through a body, the app takes control to send each part of the body itself.

```
class DeferredBody
 def each(&block)
 # normally we'd yield each part of the body, but since
 # it isn't available yet, we save &block for later
 @server_block = block
 end

 def send(data)
```

```

calling the saved &block has the same effect as
if we had yielded to it
@server_block.call data
end
end

class AsyncApp
 def call(env)
 Thread.new do
 sleep 5 # simulate waiting for some event
 body = DeferredBody.new
 response = [200, {'Content-Type' => 'text/plain'}, body]
 env['async.callback'].call response

 # at this point, the server may send the status and headers,
 # but the body was empty

 body.send 'Hello, '
 sleep 5
 body.send 'World'
 end

 [-1, {}, []] # or throw :async
 end
end

```

Note that the above won't quite work because we haven't signaled to the server that the body will be deferred and streamed in part by part.

### 38.10. Código en sinatra/base.rb: la clase Stream

The close parameter specifies whether `Stream#close` should be called after the block has been executed. This is only relevant for evented servers like Thin or Rainbows.

#### método stream

Fichero `sinatra/base.rb`:

```

class Stream
 ...
 def stream(keep_open = false)
 # Soporta el servidor callbacks asincronas? (webrick no, thin si)
 scheduler = env['async.callback'] ? EventMachine : Stream
 current = @params.dup
 body Stream.new(scheduler, keep_open) { |out|
 with_params(current) { yield(out) } # llamamos al bloque
 }
 end
 ...
end

```

Si `keep_open` es `true` el navegador se queda esperando datos y no cierra la conexión.

The `body` method set or retrieve the response body.

When a block is given, evaluation is deferred until the body is read with `each`.

```
def body(value = nil, &block)
 if block_given?
 def block.each; yield(call) end
 response.body = block
 elsif value
 headers.delete 'Content-Length' unless request.head? || value.is_a?(Rack::File) || val
 response.body = value
 else
 response.body
 end
end
```

### initialize

```
class Stream
 def self.schedule(*) yield end
 def self.defer(*) yield end

 def initialize(scheduler = self.class, keep_open = false, &back)
 @back, @scheduler, @keep_open =
 back.to_proc, scheduler, keep_open
 @callbacks, @closed = [], false
 end
 ...
end
```

### with\_params

```
class Stream
 ...
 def with_params(temp_params)
 original, @params = @params, temp_params # save params
 yield # call the block
 ensure
 @params = original if original # recover params
 end
end
```

# Capítulo 39

## Web Sockets

### 39.1. WebSockets

#### 39.1.1. Que es WebSocket y para que sirve

*WebSocket* es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.

The *WebSocket* specification—developed as part of the HTML5 initiative—introduced the *WebSocket JavaScript interface*, which defines a full-duplex single socket connection over which messages can be sent between client and server.

The *WebSocket* standard simplifies much of the complexity around bi-directional web communication and connection management.

One of the more unique features *WebSockets* provide is *its ability to traverse firewalls and proxies*, a problem area for many applications.

Comet-style applications typically employ long-polling as a rudimentary line of defense against firewalls and proxies.

The technique is effective, but is not well suited for applications that have sub-500 millisecond latency or high throughput requirements.

Plugin-based technologies such as Adobe Flash, also provide some level of socket support, but have long been burdened with the very proxy and firewall traversal problems that *WebSockets* now resolve.

A *WebSocket* detects the presence of a proxy server and automatically sets up a tunnel to pass through the proxy.

The tunnel is established by issuing an HTTP CONNECT statement to the proxy server, which requests for the proxy server to open a TCP/IP connection to a specific host and port.

Once the tunnel is set up, communication can flow unimpeded through the proxy.

Since HTTP/S works in a similar fashion, secure *WebSockets* over SSL can leverage the same HTTP CONNECT technique.

Note that *WebSockets* are just beginning to be supported by modern browsers (Chrome now supports *WebSockets* natively).

However, backward-compatible implementations that enable today's browsers to take advantage of this emerging technology are available.

En el lado del cliente, *WebSocket* está ya implementado en Mozilla Firefox 8, Google Chrome 4 y Safari 5, así como la versión móvil de Safari en el iOS 4.2.1

#### 39.1.2. Negociación del protocolo *WebSocket*

Para establecer una conexión *WebSocket*, el cliente manda una petición de negociación *WebSocket*, y el servidor manda una respuesta de negociación *WebSocket*, como se puede ver en el siguiente ejemplo:

Petición del navegador al servidor:	Respuesta del servidor:
GET /demo HTTP/1.1	HTTP/1.1 101 WebSocket Protocol Handshake
Host: example.com	Upgrade: WebSocket
Connection: Upgrade	Connection: Upgrade
Sec-WebSocket-Key2: 12998 5 Y3 1 S <del>000</del> WebS <del>o</del> cket-Origin: http://example.com	Sec-WebSocket-Location: ws://example.com/demo
Sec-WebSocket-Protocol: sample	Sec-WebSocket-Protocol: sample
Upgrade: WebSocket	
Sec-WebSocket-Key1: 4 01 46546xW%01 1 5	
Origin: http://example.com	8jKS'y:G*Co,Wxa-
~n:ds[4U	

The WebSocket protocol was designed to work well with the existing Web infrastructure. As part of this design principle, the protocol specification defines that the WebSocket connection starts its life as an HTTP connection, guaranteeing full backwards compatibility with the pre-WebSocket world.

The protocol switch from HTTP to WebSocket is referred to as a the WebSocket handshake.

Una vez establecida, las tramas WebSocket de datos pueden empezar a enviarse en ambos sentidos entre el cliente y el servidor en modo full-duplex.

Las tramas de texto pueden ser enviadas en modo full-duplex también, en ambas direcciones al mismo tiempo.

Tramas de datos binarios no están soportadas todavía en el API.

- About HTML5 WebSockets en <http://www.websocket.org>

## 39.2. websocket/rack

## 39.3. Ruby y WebSockets: TCP for the Browser

Esta sección está sacada del blog de Ilya Grigorik .

WebSockets in HTML5 were designed from the ground up to be data agnostic (binary or text) with support for full-duplex communication.

WebSockets are TCP for the web-browser. They require only a single connection, which translates into much better resource utilization for both the server and the client.

Likewise, WebSockets are proxy and firewall aware, can operate over SSL and leverage the HTTP channel to accomplish all of the above - your existing load balancers, proxies and routers will work just fine.

### The Server

```
[~/sinatra/sinatra-streaming/websockets-tcp-for-the-browser(development)]$ cat server.rb
require 'em-websocket'

EventMachine::WebSocket.start(:host => "0.0.0.0", :port => 8080) do |ws|
 ws.onopen { ws.send "Hello Client!"}
 ws.onmessage { |msg| ws.send "Pong: #{msg}" }
 ws.onclose { puts "WebSocket closed" }
end
```

### The Client

```
[~/sinatra/sinatra-streaming/websockets-tcp-for-the-browser(development)]$ cat index.html
<html>
```

```

<head>
 <script src='http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js'></script>
 <script>
 $(document).ready(function(){
 function debug(str){
 $("#debug").append("<p>" + str + "</p>");
 };

 ws = new WebSocket("ws://www.example.com:8080/websocket");
 ws.onmessage = function(evt) {
 $("#msg").append("<p>" + evt.data + "</p>");
 };
 ws.onclose = function() {
 debug("socket closed");
 };
 ws.onopen = function() {
 debug("connected...");
 ws.send("hello server");
 };
 });
 </script>
</head>
<body>
 <div id="debug"></div>
 <div id="msg"></div>
</body>
</html>

```

You open up a WebSocket connection simply by calling the `WebSocket` constructor:

```
var connection = new WebSocket('ws://html5rocks.websocket.org/echo', ['soap', 'xmpp']);
```

Notice the `ws:`. This is the new URL schema for WebSocket connections. There is also `wss:` for secure WebSocket connection the same way `https:` is used for secure HTTP connections.

Attaching some event handlers immediately to the connection allows you to know when the connection is opened, received incoming messages, or there is an error.

The second argument accepts optional subprotocols. It can be a string or an array of strings. Each string should represent a subprotocol name and server accepts only one of passed subprotocols in the array. Accepted subprotocol can be determined by accessing `protocol` property of `WebSocket` object.

```

// When the connection is open, send some data to the server
connection.onopen = function () {
 connection.send('Ping') // Send the message 'Ping' to the server
};

// Log errors
connection.onerror = function (error) {
 console.log('WebSocket Error ' + error);
};

// Log messages from the server
connection.onmessage = function (e) {
 console.log('Server: ' + e.data);
};

```

As soon as we have a connection to the server (when the open event is fired) we can start sending data to the server using the send('your message') method on the connection object. It used to support only strings, but in the latest spec it now can send binary messages too. To send binary data, you can use either `Blob` or `ArrayBuffer`.

The server might send us messages at any time. Whenever this happens the `onmessage` callback fires. The callback receives an `event` object and the actual message is accessible via the `data` property.

parrafoEjecución

```
[~/sinatra/sinatra-streaming/websockets-tcp-for-the-browser(development)]$ ruby server.rb
```

Cuando abrimos `index.html` con el navegador obtenemos una página como esta:

connected...

Hello Client!

Pong: hello server

Véase

1. Ruby and WebSockets: TCP for the Browser
2. Introducing WebSockets: Bringing Sockets to the Web by Malte Ubl and Eiji Kitamura 2010
3. Clean, simple websockets on Sinatra example by Austen Conrad Austen Conrad YouTube

### 39.4. Una Aplicación Usando Websockets en la que Múltiples Clientes Dibujan en un Lienzo

server.rb

```
[19:36] [~/srcSTW/streaming/websocketsDrawEM(master)]$ cat -n server.rb
 1 require 'em-websocket'
 2 require 'json'
 3 require 'sinatra/base'
 4
 5 EventMachine.run {
 6 @channel = EM::Channel.new
 7 @users = {}
 8 @messages = []
 9
10 EventMachine::WebSocket.start(:host => "0.0.0.0", :port => 8080) do |ws|
11
12 ws.onopen {
13 #Subscribe the new user to the channel with the callback function for the push a
14 new_user = @channel.subscribe { |msg| ws.send msg }
15
16 #Add the new user to the user list
17 @users[ws.object_id] = new_user
18
19 #Push the last messages to the user
20 @messages.each do |message|
21 ws.send message
22 end
23 }
}
```

```

24
25 ws.onmessage { |msg|
26
27 #append the message at the end of the queue
28 @messages << msg
29 @messages.shift if @messages.length > 10
30
31 #Broadcast the message to all users connected to the channel
32 @channel.push msg
33 }
34
35 ws.onclose {
36 @channel.unsubscribe(@users[ws.object_id])
37 @users.delete(ws.object_id)
38 }
39 end
40
41 #Run a Sinatra server for serving index.html
42 class App < Sinatra::Base
43 set :public_folder, settings.root
44
45 get '/' do
46 send_file 'index.html'
47 end
48 end
49 App.run!
50 }

```

### index.html.haml

```

!!! 5
%html
%head
 %meta(charset="utf-8")
 %meta(content="IE=edge,chrome=1" http-equiv="X-UA-Compatible")
 %meta(name="viewport" content="width=device-width, user-scalable=0, initial-scale=1.0, max
 %link(rel="stylesheet" href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css")
 %script(type="text/javascript" src="http://code.jquery.com/jquery-1.6.4.min.js")
 %script(type="text/javascript" src="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.mi
 %title
 WebSockets Drawing
%body
 :javascript
 var WebSocket = window.WebSocket || window.MozWebSocket;

 $(function() {

 var socket = new WebSocket("ws://" + location.hostname + ":8080");

 var currentX = 0;
 var currentY = 0;
 var lastX;
 var lastY;
 var lastReceivedX;

```

```

var lastReceivedY;

var ctx = $('#canvas')[0].getContext('2d');

$('#canvas').bind('vmousemove',function(ev){
 ev = ev || window.event;
 currentX = ev.pageX || ev.clientX;
 currentY = ev.pageY || ev.clientY;
});

socket.onopen = function(event) {
 setInterval(function(){
 if(currentX !== lastX || currentY !== lastY){
 lastX = currentX;
 lastY = currentY;
 socket.send(JSON.stringify({x:currentX, y: currentY}));
 }
 }, 30); // send every 300 milliseconds if position has changed
}
socket.onmessage = function(event) {
 var msg = $.parseJSON(event.data);

 ctx.beginPath();
 ctx.moveTo(lastReceivedX,lastReceivedY);
 ctx.lineTo(msg.x,msg.y);
 ctx.closePath();
 ctx.stroke();
 lastReceivedX = msg.x;
 lastReceivedY = msg.y;
}
});

%div(data-role="page")
 %canvas#canvas(width='1000' height='1000')

```

## index.html

```

<!DOCTYPE html>
<html>
 <head>
 <meta charset='utf-8' />
 <meta content='IE=edge,chrome=1' http-equiv='X-UA-Compatible' />
 <meta content='width=device-width, user-scalable=0, initial-scale=1.0, maximum-scale=1.0;' />
 <link href='http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css' rel='stylesheet' />
 <script src='http://code.jquery.com/jquery-1.6.4.min.js' type='text/javascript'></script>
 <script src='http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js' type='text/javascript'></script>
 <title>
 WebSockets Drawing
 </title>
</head>
<body>
 <script type='text/javascript'>
 //<![CDATA[
</pre>

```

```

var WebSocket = window.WebSocket || window.MozWebSocket;

$(function() {

 var socket = new WebSocket("ws://" + location.hostname + ":8080");

 var currentX = 0;
 var currentY = 0;
 var lastX;
 var lastY;
 var lastReceivedX;
 var lastReceivedY;

 var ctx = $('#canvas')[0].getContext('2d');

 $('#canvas').bind('mousemove',function(ev){
 ev = ev || window.event;
 currentX = ev.pageX || ev.clientX;
 currentY = ev.pageY || ev.clientY;
 });

 socket.onopen = function(event) {
 setInterval(function(){
 if(currentX !== lastX || currentY !== lastY){
 lastX = currentX;
 lastY = currentY;
 socket.send(JSON.stringify({x:currentX, y: currentY}));
 }
 }, 30); // send every 300 milliseconds if position has changed
 }

 socket.onmessage = function(event) {
 var msg = $.parseJSON(event.data);

 ctx.beginPath();
 ctx.moveTo(lastReceivedX,lastReceivedY);
 ctx.lineTo(msg.x,msg.y);
 ctx.closePath();
 ctx.stroke();
 lastReceivedX = msg.x;
 lastReceivedY = msg.y;
 }
});

//]]>
</script>
<div data-role='page'>
 <canvas height='1000' id='canvas' width='1000'></canvas>
</div>
</body>
</html>

```

### 39.4.1. Enlaces Relacionados

- A shared canvas where multiple clients can draw lines using EM-Websocket

- Este ejemplo está tomado del material del curso *Mobile Web Applications Development with HTML5* [11]

### 39.5. Using WebSockets on Heroku with Ruby

1. Using WebSockets on Heroku with Ruby
2. Ruby WebSockets Chat Demo en Heroku
3. heroku-examples / ruby-websockets-chat-demo en GitHub

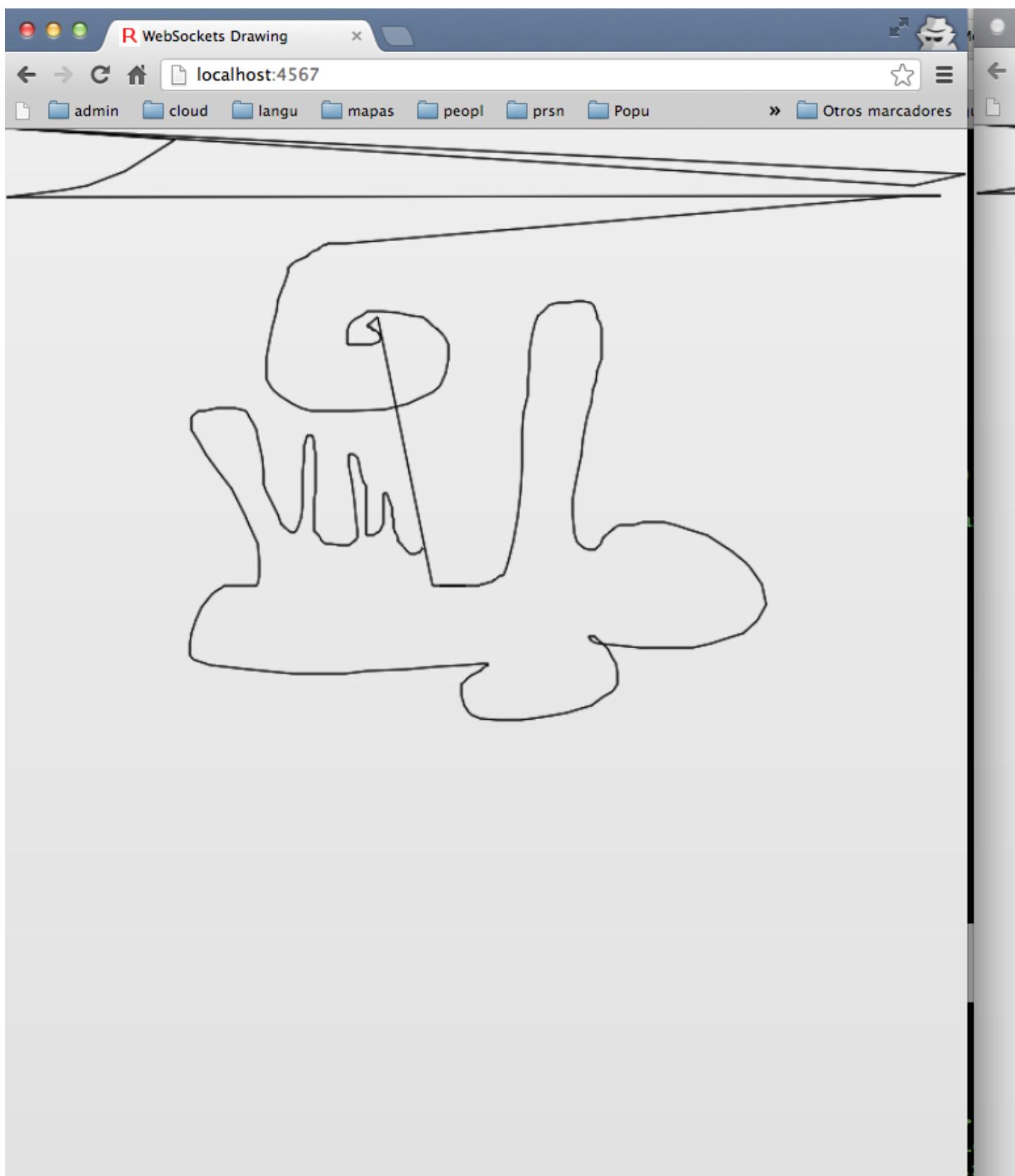


Figura 39.1: Múltiples clientes pueden dibujar en el lienzo

## Capítulo 40

# Openid y Sinatra

OpenID provides sites and services with a decentralized protocol for authenticating users through a wide variety of providers. What this means is that a site integrating OpenID can allow its users to log in using, for example, their Yahoo!, Google, or AOL accounts. Not only can the consuming site avoid having to create a login system itself, but it can also take advantage of the accounts that its users already have, thereby increasing user registration and login rates.

In addition to simple authentication, OpenID also offers a series of extensions through which an OpenID provider can allow sites to obtain a user's profile information or integrate additional layers of security for the login procedure.

What makes OpenID so intriguing is the fact that it offers a standard that is fully decentralized from the providers and consumers. This aspect is what allows a single consuming site to allow its users to log in via Yahoo! and Google, while another site may want to allow logins via Blogger or WordPress. Ultimately, it is up to the OpenID consumer (your site or service) to choose what login methods it would like to offer its user base.

### 40.1. Referencias. Véase Tambien

- GitHub ahx/sinatra-openid-consumer-example
- Google Offers Named OpenIDs por Jeff Atwood
- How do I log in with OpenID?
- Programming Social Applications por Jonathan Leblanc. O'Reilly. 2011.

## Capítulo 41

# Bootstrap your Web Application with Ruby and Sinatra

### 41.1. BootStrap

Para descargar BootStrap nos vamos a la página:

<http://getbootstrap.com/>

The folks over at MaxCDN provide CDN support for Bootstrap's CSS and JavaScript. Just use these Bootstrap CDN links.

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css">

<!-- Optional theme -->
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap-theme.min.css">

<!-- Latest compiled and minified JavaScript -->
<script src="//netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.js"></script>
```

Once downloaded, unzip the compressed folder to see the structure of (the compiled) Bootstrap.

**Usando Bootstrap con Sinatra** Véase:

1. Bootstrap your Web Application with Ruby and Sinatra por David Sale.
2. Véase también: [sinatra-bootstrap](#)
3. [crguezl/sinatra-bootstrap-example](#) en GitHub en GitHub.

### Tutoriales de BootStrap

1. [w3schools.com](#)
2. Tutoriales en YouTube del 1 al 15 por <http://www.creativitytuts.org/> :
  - a) Twitter Bootstrap Tutorial 1 - Introduction / Setup por CreativityTuts
  - b) Twitter Bootstrap Tutorial 2 - Forms por CreativityTuts
  - c) Twitter Bootstrap Tutorial 3 - In-line & Search Form por CreativityTuts
  - d) Twitter Bootstrap Tutorial 4 - Tables por CreativityTuts
3. Tutorials on Using Bootstrap for the Easy Start
4. BootStrap Tutorial Parts I and II

## Capítulo 42

# Ajax in Sinatra

### 42.1. Un Programa para Experimentar con las Expresiones Regulares Usando Ajax

regular expression testing playground with ruby and sinatra

```
[~/sinatra/regexp(master)]$ cat regex-tester.rb
require 'sinatra' # gem install sinatra --no-ri --no-rdoc
set :port, 3000
html = <<-EOT
<html><head><style>
#regex,#text{ width:100%; font-size:15px; display:block; margin-bottom:5px; }
#text{ height: 200px; }
span{ background:rgb(230,191,161); display:inline-block; border-radius:3px;}>
</style></head><body>

<input id="regex" placeholder="Regex"/>
<textarea id="text" placeholder="Text"></textarea>
<div id="result"></div>

<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script>
$('#regex,#text').keyup(function(){
 $.get('/preview',{
 reg:$('#regex').val(),
 text:$('#text').val()
 },function(r){
 $('#result').html(r);
 });
});
</script>

</body></html>
EOT

get('/') { html }
get '/preview' do
 begin
 params['text'].gsub(/#{params['reg']}/, '\1')
 rescue
 'Your regex is invalid'
```

```
end
end
```

## 42.2. Un Ejemplo Simple

Véase [sinatra-jquery-ajax](#) en GitHub.

```
~/sinatra/sinatra-jquery-ajax(master)]$ tree
. |--- Gemfile
|--- Gemfile.lock
|--- README
|--- config.ru
|--- public
| |--- css
| | '--- style.css
| '--- js
| '--- app.js
|--- sinatra_jquery_test.rb
'--- views
 |--- app.erb
 '--- layout.erb
```

```
[~/sinatra/sinatra-jquery-ajax(master)]$ cat sinatra_jquery_test.rb
require 'sinatra'
```

```
get '/' do
 erb :app
end

get '/play' do
 if request.xhr?
 %q{

Hello! back</h1>} else "<h1>Not an Ajax request!</h1>" end end


```

### request.xhr?

El predicado `request.xhr?` nos permite saber si este es una request ajax.

```
[~/sinatra/sinatra-jquery-ajax(master)]$ cat views/layout.erb
<!DOCTYPE html>
<html>
 <head>
 <link rel="stylesheet" type="text/css" href="css/style.css">
 <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
 <script src="js/app.js"></script>
 </head>
 <body>
 <%= yield %>
 </body>
</html>
```

**Content Delivery Network (CDN)** A *Content Delivery Network (CDN)* is a distributed system of web servers that aim to deliver content to users quickly and efficiently.

A large number of CDNs are available for JavaScript libraries.

The idea is that if lots of sites use the same CDN or *hotlink* (véase Hotlinking), it will be cached locally on the user's machine, saving the user from an extra download across all those sites.

The downside is a loss of control and the chance (however small) that the CDN might go down and be unavailable.

1. Google Hosted Libraries - Developer's Guide
2. jQuery download

### views/app.erb

```
[~/sinatra/sinatra-jquery-ajax(master)]$ cat views/app.erb
<div id="div1">
 <h2 class="pink">Let jQuery AJAX Change This Text</h2>
 <button type="button">Get External Content</button>
</div>
```

Inside a `<button>` element you can put content, like text or images. This is the difference between this element and buttons created with the `<input>` element.

Always specify the type attribute for a `<button>` element. Different browsers use different default types for the `<button>` element.

HTML5 has the following new attributes: `autofocus`, `form`, `formaction`, `formenctype`, `formmethod`, `formnovalidate`, and `formtarget`.

### public/js/app.js

```
[~/sinatra/sinatra-jquery-ajax(master)]$ cat public/js/app.js
$(document).ready(function(){
 $("button").click(function(){
 $("#div1").load("/play",function(responseTxt,statusTxt,xhr){
 /* if(statusTxt=="success") alert("External content loaded successfully!"); */
 if(statusTxt=="error")
 alert("Error: "+xhr.status+": "+xhr.statusText);
 });
 });
});
```

`$(document).ready(function(){ ... })` No es posible interactuar de forma segura con el contenido de una página hasta que el documento no se encuentre preparado para su manipulación.

jQuery permite detectar dicho estado a través de la declaración

```
$(document).ready()
```

de forma tal que el bloque se ejecutará sólo una vez que la página este disponible.

**selectores CSS** El concepto más básico de jQuery es el de *seleccionar algunos elementos y realizar acciones con ellos*.

La biblioteca soporta gran parte de los selectores CSS3 y varios más no estandarizados.

```
$("#div1").load("/play",function(responseTxt,statusTxt,xhr){ ... })
```

En <http://api.jquery.com/category/selectors/> se puede encontrar una completa referencia sobre los selectores de la biblioteca.

## Controladores de Eventos (Event Handlers)

jQuery provee métodos para asociar *controladores de eventos* (en inglés *event handlers*) a selectores. Cuando un evento ocurre, la función provista es ejecutada.

```
$("button").click(function(){ ... })
```

Dentro de la función, la palabra clave **this** hace referencia al elemento en que el evento ocurre.

Para más detalles sobre los eventos en jQuery, puede consultar <http://api.jquery.com/category/events/>.

## El Evento

La función del controlador de eventos puede recibir un objeto.

Este objeto puede ser utilizado para determinar la naturaleza del evento o, por ejemplo, prevenir el comportamiento predeterminado de éste.

Para más detalles sobre el objeto del evento, visite

## XMLHttpRequest(XHR)

<http://api.jquery.com/category/events/event-object/>

El método **XMLHttpRequest(XHR)** permite a los navegadores comunicarse con el servidor sin la necesidad de recargar la página.

Este método, también conocido como *Ajax (Asynchronous JavaScript and XML)*, permite la creación de aplicaciones ricas en interactividad.

Las peticiones Ajax son ejecutadas por el código JavaScript, el cual

1. envía una petición a una URL y
2. cuando recibe una respuesta, una *función de devolución (callback)* puede ser ejecutada
3. Esta función recibe como argumento la respuesta del servidor y realiza algo con ella.
4. Debido a que la respuesta es asíncrona, el resto del código de la aplicación continua ejecutándose, por lo cual, es imperativo que una función de devolución sea ejecutada para manejar la respuesta.

## Métodos jQuery para Ajax

A través de varios métodos, jQuery provee soporte para Ajax, permitiendo abstraer las diferencias que pueden existir entre navegadores.

Los métodos en cuestión son

1. `$.get()` `.get()`
2. `$.getScript()` `.getScript()`
3. `$.getJSON()` `.getJSON()`
4. `$.post()` `.post()`
5. `$(...).load()` `.load()`
- 6.
7. `$(...).ajax()` `jQuery.ajax()`

A pesar que la definición de Ajax posee la palabra **XML**, la mayoría de las aplicaciones no utilizan dicho formato para el transporte de datos, sino que en su lugar se utiliza HTML plano o información en formato **JSON (JSON JavaScript Object Notation)**.

Generalmente, jQuery necesita algunas instrucciones sobre el tipo de información que se espera recibir cuando se realiza una petición Ajax.

En algunos casos, el tipo de dato es especificado por el nombre del método, pero en otros casos se lo debe detallar como parte de la configuración del método:

1. **text** Para el transporte de cadenas de caracteres simples.

2. **html** Para el transporte de bloques de código HTML que serán ubicados en la página.
3. **script** Para añadir un nuevo script con código JavaScript a la página.
4. **json** Para transportar información en formato JSON, el cual puede incluir cadenas de caracteres, arreglos y objetos.  
Es recomendable utilizar los mecanismos que posea el lenguaje del lado de servidor para la generación de información en JSON.
5. **jsonp** Para transportar información JSON de un dominio a otro.
6. **xml** Para transportar información en formato XML.

**.load()** La sintaxis de .load() es:

```
.load(url [, data] [, complete(responseText, textStatus, XMLHttpRequest)])
```

En nuestro ejemplo lo hemos usado así:

```
$("#div1").load("/play",function(responseTxt,statusTxt,xhr){
 /* if(statusTxt=="success") alert("External content loaded successfully!"); */
 if(statusTxt=="error")
 alert("Error: "+xhr.status+": "+xhr.statusText);
});
```

Load data from the server and place the returned HTML into the matched element. Por ejemplo:

```
$("#result").load("ajax/test.html");
```

If no element is matched by the selector, in this case,

```
.load(url [, data] [, complete(responseText, textStatus, XMLHttpRequest)])
```

if the document does not contain an element with `id="result"`, the Ajax request will not be sent.

**.load() usa GET si no se especifica data** The POST method is used if `data` is provided as an object; otherwise, GET is assumed.

**Especificando un Objeto del Documento Remoto via un Selector** The .load() method, unlike `$.get()`, allows us to specify a portion of the remote document to be inserted.

This is achieved with a special syntax for the `url` parameter.

If one or more space characters are included in the string, the portion of the string following the first space is assumed to be a jQuery selector that determines the content to be loaded.

```
$("#result").load("ajax/test.html #container");
```

When this method executes, it retrieves the content of `ajax/test.html`, but then jQuery parses the returned document to find the element with an ID of `container`.

This element, along with its contents, is inserted into the element with an ID of `result`, and the rest of the retrieved document is discarded.

We could modify the example above to use only part of the document that is fetched:

### **.innerHTML y .load**

jQuery uses the browser's `.innerHTML` property to parse the retrieved document and insert it into the current document.

During this process, browsers often filter elements from the document such as `<html>`, `<title>`, or `<head>` elements. As a result, the elements retrieved by `.load()` may not be exactly the same as if the document were retrieved directly by the browser.

## 42.3. Ajax, Sinatra y RightJS

Ajax has been around for a while now but that doesn't mean it is any less fun. The nice thing about Sinatra is you are left to do as much or little JavaScript as you like and you can do it in any way that you want as well.

In this ditty I hope to show that it's easy to add some Ajax magic to a Sinatra app (with a little help from a JavaScript framework).

1. An example of Sinatra working with Ajaxified JQuery based on some pieces of code published by Rafael George on the Sinatra Google Group
2. Ajax, Sinatra y RightJS Darren Jones

## 42.4. Un Chat con Ajax y JQuery

### Donde

- [~/sinatra/sinatra-streaming/blazeeboychat(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-streaming/blazeeboychat
- Código en GitHub

### chat.rb

```
[~/sinatra/sinatra-streaming/blazeeboychat(master)]$ cat chat.rb
require 'sinatra'
require 'sinatra/reloader' if development?
#set :port, 3000
#set :environment, :production

chat = ['welcome...']

get('/') { erb :index }

get '/send' do
 return [404, {}, "Not an ajax request"] unless request.xhr?
 chat << "#{request.ip} : #{params['text']}"
 nil
end

get '/update' do
 return [404, {}, "Not an ajax request"] unless request.xhr?
 @updates = chat[params['last'].to_i..-1] || []

 @last = chat.size
 erb <<-HTML', :layout => false
 <% @updates.each do |phrase| %>
 <%= phrase %>

 <% end %>
 <span data-last="<%= @last %>">
 HTML
end
```

## views/index.erb

```
[~/sinatra/sinatra-streaming/blazeeboychat(master)]$ cat views/layout.erb
<html>
 <head>
 <link rel="stylesheet" type="text/css" href="simple.css">
 </head>
 <body>
 <%= yield %>

 <script src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
 <script src= "chat.js"></script>
 </body>
</html>
```

```
[~/sinatra/sinatra-streaming/blazeeboychat(master)]$ cat views/index.erb
<input id="text" placeholder="Write then press Enter." autofocus />
<div id="chat"></div>
```

## chat.js

```
[~/sinatra/sinatra-streaming/blazeeboychat(master)]$ cat public/chat.js
$('#text').keypress(
 function(e){
 if(e.keyCode==13){
 $.get('/send',{text:$('#text').val()});
 $('#text').val('');
 }
 }
);

(function() {
 var last = 0;
 setInterval(
 function(){
 $.get('/update',{last: last},
 function(response){
 last = $('<p>').html(response).find('span').data('last');
 $('#chat').append(response);
 }
);
 },
 1000);
})();
```

## El método get de jQuery

jQuery.get( url [, data ] [, success ] [, dataType ] ) Returns: jqXHR

- Description: Load data from the server using a HTTP GET request.
- **url:** Type: **String** A string containing the URL to which the request is sent.
- **data:** Type: PlainObject or **String**  
A plain object or string that is sent to the server with the request.

- **success:**

Type: Function( PlainObject data, String textStatus, jqXHR jqXHR )

A callback function that is executed if the request succeeds.

- **dataType** Type: String: The type of data expected from the server. Default: Intelligent Guess (xml, json, script, or html).

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
 data: data,
 success: success,
 dataType: dataType
});
```

The **success** callback function is passed the returned data, which will be an XML root element, text string, JavaScript file, or JSON object, depending on the MIME type of the response.

It is also passed the text status of the response.

## Rakefile

```
[~/sinatra/sinatra-streaming/blazeeboychat(master)]$ cat Rakefile
task :default => :server

desc "run the chat server"
task :server do
 sh "bundle exec ruby chat.rb"
end

desc "make a non Ajax request via curl"
task :noajax do
 sh "curl -v http://localhost:4567/update"
end

desc "make an Ajax request via curl"
task :ajax do
 sh %q{curl -v -H "X-Requested-With: XMLHttpRequest" localhost:4567/update}
end

desc "Visit the GitHub repo page"
task :open do
 sh "open https://github.com/crguezl/chat-blazee"
end
```

## Gemfile

```
~/sinatra/sinatra-streaming/blazeeboychat(master)]$ cat Gemfile
source 'https://rubygems.org'

gem 'thin'

group :development do
 gem 'sinatra-contrib'
end
```

## public/simple.css

```
[~/sinatra/sinatra-streaming/blazeeboychat(master)]$ cat public/simple.css
#text {width:100%; font-size: 15px; padding: 5px; display: block;}
```

## 42.5. Práctica: Chat Usando Ajax y jQuery

Modifique el chat visto en la sección *Un Chat con Ajax y JQuery* 42.4 para que:

- Deberán registrarse los usuarios
- Si el nombre del usuario esta tomado se señalara el error y se le pedirá que cambie de nombre
- Una vez en el chat:
  - En un contenedor aparecen los nombres de usuario
  - En otro los mensajes
  - En un tercero con un botón `send` la entrada para el siguiente mensaje del usuario
- Escriba las pruebas. Use Selenium
- Despliegue las pruebas en Travis
- Haga un análisis de cubrimiento usando Coveralls Repase la sección *Coveralls: Análisis de Cubrimiento* 20.7
- Despliegue en Heroku

Estos enlaces pueden ser de ayuda:

1. *Un Chat con Ajax y JQuery* 42.4
2. Código del chat en GitHub
3. *BootStrap 76.1*
4. *Integración Contínua: Travis 20*
5. *Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis* 36.0.2
6. *Sample app with Sinatra Rspec Capybara and Selenium* 36.0.4
7. *Capybara 18*
8. *Coveralls: Análisis de Cubrimiento* 20.7
9. *Ajax 64*

## 42.6. Práctica: TicTactoe Usando Ajax

Extienda la práctica del TicTacToe enunciada e la sección 33.29 para que la página no se recarge cada vez que el jugador hace click en una de las casillas.

El código Javascript se encargará de que el navegador envíe la jugada elegida por el usuario `b2`. Si la jugada es correcta (la casilla `b2` no está ocupada) el servidor retornará al navegador la información necesaria para que pueda proceder a mostrar los movimientos elegidos por el jugador y el computador. En caso contrario el servidor envía un código de jugada ilegal. El código javascript es el que modifica la clase de la casilla a `cross` o `circle` de manera adecuada.

Mejore las hojas de estilo usando SAAS 96. Despliegue la aplicación en Heroku.

1. jQuery .click()
2. jQuery .get()
3. jQuery .addClass()
4. jQuery .data()
5. Como obtener en JS el valor de un elemento sobre el que se ha hecho click?. Véase una solución en jsfiddle
6. HTML5 Custom Data Attributes (data-\*)
7. Puede usar plain text para la comunicación. Opcionalmente si se quiere usar JSON, la gema sinatra-json añade un JSON helper que permite retornar documentos JSON. Se pueden consultar estas fuentes:
  - a) json
  - b) How to: Return JSON from Sinatra
  - c) A very small example app showing how to accept and return JSON as an API

## Capítulo 43

# Redis y Sinatra

1. Redis Quick Start
2. How to Build a Shortlink App with Ruby and Redis Charlie Somerville en <http://net.tutsplus.com/>
3. Sinatra-Redis-Heroku-App en GitHub
4. sinatra-redis extension
5. Redis Cloud Redis Cloud is a managed cloud service for hosting and running your Redis dataset through its add-on for Heroku

## Capítulo 44

# MongoDB y Sinatra

1. Getting Started with MongoDB
2. Mongoid
3. MongoDB Ruby Driver Tutorial
4. MongoDB Ruby Tutorial en GitHub
5. Sinatra Recipes: MongoDB
6. Un ejemplo de spatial programming usando mongoid: [https://github.com/crguezl/spatial-ruby-openshift-quickstart](https://github.com/crguezl/spatial-rubyOpenshift-quickstart). Véase también: Spatial apps on OpenShift PaaS using Ruby, Sinatra and MongoDB

## Capítulo 45

# Building Backbone.js Apps With Ruby, Sinatra, MongoDB and Haml

Véase Building Backbone.js Apps With Ruby, Sinatra, MongoDB and Haml

# Capítulo 46

## DataMapper y Sinatra

### 46.1. Introducción a Los Object Relational Mappers (ORM)

What is a Object Relational Mapper?

A simple answer is that you wrap your tables or stored procedures in classes in your programming language, so that instead of writing SQL statements to interact with your database, you use methods and properties of objects.

In other words, instead of something like this:

```
String sql = "SELECT ... FROM persons WHERE id = 10"
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0] ["FIRST_NAME"];
```

you do something like this:

```
Person p = Person.Get(10);
```

or similar code (lots of variations here). The framework is what makes this code possible.

Now, benefits:

1. First of all, you hide the SQL away from your logic code
2. This has the benefit of allowing you to more easily support more database engines
3. For instance, MS SQL Server and Oracle have different names on typical functions, and different ways to do calculations with dates. This difference can be put away from your logic code.
4. Additionally, you can focus on writing the logic, instead of getting all the SQL right.
5. The code will typically be more readable as well, since it doesn't contain all the plumbing necessary to talk to the database.

### 46.2. Introducción al Patrón DataMapper

Martin Fowler (Catalog of Patterns of Enterprise Application Architecture):

1. Objects and relational databases have different mechanisms for structuring data.
2. Many parts of an object, such as collections and inheritance, aren't present in relational databases.
3. When you build an object model with a lot of business logic it's valuable to use these mechanisms (creo que se refiere a la herencia, etc.) to better organize the data and the behavior that goes with it.

4. Doing so leads to variant schemas; that is, the object schema and the relational schema don't match up.
5. You still need to transfer data between the two schemas, and this data transfer becomes a complexity in its own right.
6. If the in-memory objects know about the relational database structure, changes in one tend to ripple to the other.
7. The *Data Mapper* is a layer of software that separates the in-memory objects from the database.
8. *Its responsibility is to transfer data between the two and also to isolate them from each other*
9. With Data Mapper the in-memory objects needn't know even that there's a database present; they need no SQL interface code, and certainly no knowledge of the database schema.
10. (The database schema is always ignorant of the objects that use it.)

- DataMapper en la Wikipedia
- Martin Fowler: DataMapper
- Proyecto sinatra-datamapper-sample en GitHub
- Documentación de DataMapper
- Sinatra Recipes: DataMapper
- Sinatra Book: DataMapper

### 46.3. Ejemplo Simple de uso de DataMapper: Agenda

- [~/sinatra/sinatra-datamapper/data\_mapper\_example\_agenda(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/STW/data\_mapper\_example2
- [~/sinatra/sinatra-datamapper/data\_mapper\_example\_agenda(master)]\$ git remote -v  
origin git@github.com:crguezl/datamapper\_example.git (fetch)  
origin git@github.com:crguezl/datamapper\_example.git (push)
- Agenda en Sinatra usando DataMapper

#### DataMapper.setup

```
[~/sinatra/sinatra-datamapper/data_mapper_example_agenda(master)]$ cat app.rb
#!/usr/bin/env ruby
require 'rubygems'
require 'sinatra'
require 'haml'
require 'data_mapper'

full path!
DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/agenda.db")
```

We must specify our database connection.

We need to make sure to do this before you use our models, i.e. before we actually start accessing the database.

## Modelos

```
Define the model
class Contact
 include DataMapper::Resource

 property :id, Serial
 property :firstname, String
 property :lastname, String
 property :email, String
end
```

The `Contact` model is going to need to be persistent, so we'll include `DataMapper::Resource`. The convention with model names is to use the singular, not plural version.

Module: `DataMapper::Resource`

```
+ (Object) included(model) private
```

Makes sure a class gets all the methods when it includes Resource.

Inside your class, call the `property` method for each property you want to add. The only two required arguments are the `name` and `type`, everything else is optional.

*Primary keys* are not automatically created for you. You MUST configure at least one key property on your data-store.

More often than not, you'll want an *auto-incrementing* integer as a primary key, so DM has a shortcut:

```
property :id, Serial

(Property) property(name, type, options = {})
```

Defines a Property on the Resource

See the section Properties in [datamapper.org](http://datamapper.org).

`DataMapper.auto_upgrade!`

1. This tries to make the schema match the model.
2. It will CREATE new tables, and add columns to existing tables.
3. It won't change any existing columns though (say, to add a NOT NULL constraint) and it doesn't drop any columns.
4. Both these commands also can be used on an individual model (e.g. `Song.auto_migrate!`)

## Rutas

```
Show list of contacts
%w{/ /contacts/?}.each do |r|
 get r do
 haml :list, :locals => { :cs => Contact.all }
 end
end

Show form to create new contact
get '/contacts/new' do
```

```

haml :form, :locals => {
 :c => Contact.new,
 :action => '/contacts/create'
}
end

Create new contact
post '/contacts/create' do
 c = Contact.new
 c.attributes = params
 c.save

 redirect("/contacts/#{c.id}")
end

Show form to edit contact
get '/contacts/:id/edit' do |id|
 c = Contact.get(id)
 haml :form, :locals => {
 :c => c,
 :action => "/contacts/#{c.id}/update"
 }
end

Show form to find a contact by firstname
get '/contacts/findform' do
 haml :findform, :locals => {
 :action => "/contacts/find"
 }
end

post '/contacts/find' do
 puts "params = #{params}"
 s = params.select { |k,v| v != '' }
 #s[:order] = [:lastname.desc]
 s[:order] = [:lastname.asc]
 puts "selected #{s}"
 cs = Contact.all(s)
 puts cs
 haml :list, :locals => { :cs => cs }
end
#
Edit a contact
post '/contacts/:id/update' do |id|
 puts "en /contacts/:id/update: params = #{params} id = #{id}"
 c = Contact.get(id)
 puts c.inspect

 puts "Fail while recording #{c}" unless c.update(
 :firstname => params[:firstname],
 :lastname => params[:lastname],
 :email => params[:email])

```

```

 redirect "/contacts/#{id}"
end

Delete a contact
post '/contacts/:id/destroy' do |id|
 c = Contact.get(id)
 c.destroy

 redirect "/contacts/"
end

View a contact
TODO: Put at bottom?
get '/contacts/:id' do |id|
 c = Contact.get(id)
 haml :show, :locals => { :c => c }
end

```

--END--

## Vistas

```

--END--
@@ layout
%html
 %head
 %title Agenda
 %body
 = yield
 %a(href="/contacts/") Contact List
 %a(href="/contacts/findform") Find Contact

@@form
%h1 Create a new contact
%form(action="#{action}" method="POST")
 %label(for="firstname") First Name
 %input(type="text" name="firstname" value="#{c.firstname}")
 %br

 %label(for="lastname") Last Name
 %input(type="text" name="lastname" value="#{c.lastname}")
 %br

 %label(for="email") Email
 %input(type="text" name="email" value="#{c.email}")
 %br

 %input(type="submit")
 %input(type="reset")
 %br

- unless c.id == 0
 %form(action="/contacts/#{c.id}/destroy" method="POST")
 %input(type="submit" value="Destroy")

```

```

@@show

| | |
|---|--------------------|
| First Name | <%= c.firstname %> |
| Last Name | <%= c.lastname %> |
| Email | <%= c.email %> |
| Edit Contact | |

@@list

Contacts

New Contact

| | |
|---|---|
| - | cs.each do c |
| | <%td= c.firstname %> |
| | <%td= c.lastname %> |
| | <%td= c.email %> |
| | <%td |
| | Show |
| | Edit |

@@findform

Find a contact for firstname

<%form(action="#{action}" method="POST")
 %label(for="firstname") First Name
 %input(type="text" name="firstname" value="")
 %br

 %label(for="lastname") Last Name
 %input(type="text" name="lastname" value="")
 %br

 %label(for="email") Email
 %input(type="text" name="email" value="")
 %br

 %input(type="submit")
 %input(type="reset")
 %br

```

## Enlaces

1. Documentación del módulo DataMapper en RubyDoc
2. <https://github.com/crguezl/datamapper-example>
3. <https://github.com/crguezl/datamapper-intro>

## 46.4. Ejemplo de Uso de DataMapper: Canciones de Sinatra

### Donde

- [~/sinatra/sinatra-datamapper-jump-start(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-datamapper-jump-start
- [~/sinatra/sinatra-datamapper-jump-start(master)]\$ git remote -v  
origin git@github.com:crguezl/sinatra-datamapper-jump-start.git (fetch)  
origin git@github.com:crguezl/sinatra-datamapper-jump-start.git (push)
- Este ejemplo en GitHub
- <http://sinadm.herokuapp.com/> (Puede que este caída)

### Enlaces

1. Documentación del módulo DataMapper en RubyDoc
2. <https://github.com/crguezl/datamapper-example>
3. <https://github.com/crguezl/datamapper-intro>

### La Clase Song

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat song.rb
require 'dm-core'
require 'dm-migrations'

class Song
 include DataMapper::Resource
 property :id, Serial
 property :title, String
 property :lyrics, Text
 property :length, Integer

end
```

The Song model is going to need to be persistent, so we'll include `DataMapper::Resource`.

The convention with model names is to use the singular, not plural version... but that's just the convention, we can do whatever we want.

```
configure do
 enable :sessions
 set :username, 'frank'
 set :password, 'sinatra'
end
```

### DataMapper.finalize

```
DataMapper.finalize
```

This method performs the necessary steps to finalize `DataMapper` for the current repository. It should be called after loading all models and plugins. It ensures foreign key properties and anonymous join models are created. These are otherwise lazily declared, which can lead to unexpected errors. It also performs basic validity checking of the `DataMapper` models.

## Mas código de Song.rb

```
get '/songs' do
 @songs = Song.all
 slim :songs
end

get '/songs/new' do
 halt(401, 'Not Authorized') unless session[:admin]
 @song = Song.new
 slim :new_song
end

get '/songs/:id' do
 @song = Song.get(params[:id])
 slim :show_song
end

get '/songs/:id/edit' do
 @song = Song.get(params[:id])
 slim :edit_song
end
```

## Song.create

If you want to create a new resource with some given attributes and then save it all in one go, you can use the `#create` method:

```
post '/songs' do
 song = Song.create(params[:song])
 redirect to("/songs/#{song.id}")
end

put '/songs/:id' do
 song = Song.get(params[:id])
 song.update(params[:song])
 redirect to("/songs/#{song.id}")
end

delete '/songs/:id' do
 Song.get(params[:id]).destroy
 redirect to('/songs')
end
```

## Una sesión con pry probando DataMapper

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ pry
[1] pry(main)> require 'sinatra'
=> true
[2] pry(main)> require './song'
=> true
```

## DataMapper.setup

We must specify our database connection.

We need to make sure to do this before you use our models, i.e. before we actually start accessing the database.

```

If you want the logs displayed you have to do this before the call to setup
DataMapper::Logger.new($stdout, :debug)

An in-memory Sqlite3 connection:
DataMapper.setup(:default, 'sqlite::memory:')

A Sqlite3 connection to a persistent database
DataMapper.setup(:default, 'sqlite:///path/to/project.db')

A MySQL connection:
DataMapper.setup(:default, 'mysql://user:password@hostname/database')

A Postgres connection:
DataMapper.setup(:default, 'postgres://user:password@hostname/database')

```

Note: that currently you must setup a :default repository to work with DataMapper (and to be able to use the repository methods)

In our case:

```
[4] pry(main)> pry(main)> DataMapper.setup(:default,'sqlite:development.db')
```

## Multiple Data-Store Connections

DataMapper sports a concept called a context which encapsulates the data-store context in which you want operations to occur. For example, when you setup a connection you are defining a context known as :default

```
DataMapper.setup(:default, 'mysql://localhost/dm_core_test')
```

If you supply another context name, you will now have 2 database contexts with their own unique loggers, connection pool, identity map....one *default context* and one *named context*.

```
DataMapper.setup(:external, 'mysql://someother_host/dm_core_test')
```

To use one context rather than another, simply wrap your code block inside a `repository` call. It will return whatever your block of code returns.

```
DataMapper.repository(:external) { Person.first }
hits up your :external database and retrieves the first Person
```

This will use your connection to the :external data-store and the first Person it finds. Later, when you call `.save` on that person, it'll get saved back to the :external data-store; An **object is aware of what context it came from and should be saved back to**.

## El Objeto DataMapper::Adapters

```
=> #<DataMapper::Adapters::SqliteAdapter:0x007fad2c0f6a50
@field_naming_convention=DataMapper::NamingConventions::Field::Underscored,
@name=:default,
@normalized_uri=
#<DataObjects::URI:0x007fad2c0f62a8
@fragment="{Dir.pwd}/development.db",
@host="",
@password=nil,
@path=nil,
@port=nil,
@query=
```

```

{"scheme"=>"sqlite3",
 "user"=>nil,
 "password"=>nil,
 "host"=>"",
 "port"=>nil,
 "query"=>nil,
 "fragment"=>"{Dir.pwd}/development.db",
 "adapter"=>"sqlite3",
 "path"=>nil},
 @relative=nil,
 @scheme="sqlite3",
 @subscheme=nil,
 @user=nil>,
@options=
 {"scheme"=>"sqlite3",
 "user"=>nil,
 "password"=>nil,
 "host"=>"",
 "port"=>nil,
 "query"=>nil,
 "fragment"=>"{Dir.pwd}/development.db",
 "adapter"=>"sqlite3",
 "path"=>nil},
@resource_naming_convention=
 DataMapper::NamingConventions::Resource::UnderscoredAndPluralized>

```

**Creando las tablas con DataMapper.auto\_migrate!** We can create the table by issuing the following command:

[4] pry(main)> DataMapper.auto\_migrate!

1. This will issue the necessary CREATE statements (DROPPING the table first, if it exists) to define each storage according to their properties.
2. After `auto_migrate!` has been run, the database should be in a pristine state.
3. All the tables will be empty and match the model definitions.

**DataMapper.auto\_upgrade!** This wipes out existing data, so you could also do:

`DataMapper.auto_upgrade!`

1. This tries to make the schema match the model.
2. It will CREATE new tables, and add columns to existing tables.
3. It won't change any existing columns though (say, to add a NOT NULL constraint) and it doesn't drop any columns.
4. Both these commands also can be used on an individual model (e.g. `Song.auto_migrate!`)

## Métodos de la Clase Mapeada

[5] pry(main)> song = Song.new  
=> #<Song @id=nil @title=nil @lyrics=nil @length=nil @released\_on=nil>  
[6] pry(main)> song.save

```

=> true
[7] pry(main)> song
=> #<Song @id=1 @title=<not loaded> @lyrics=<not loaded> @length=<not loaded> @released_on=<no
[8] pry(main)> song.title = "My Way"
=> "My Way"
[9] pry(main)> song.lyrics
=> nil
[10] pry(main)> song.lyrics = "And now, the end is near ..."
=> "And now, the end is near ..."
[11] pry(main)> song.length = 435
=> 435
[42] pry(main)> song.save
=> true
[43] pry(main)> song
=> #<Song @id=1 @title="My Way" @lyrics="And now, the end is near ..." @length=435 @released_o

```

**El método create** If you want to create a new resource with some given attributes and then save it all in one go, you can use the `#create` method.

```
[28] pry(main)> Song.create(title: "Come fly with me", lyrics: "Come fly with me, let's fly, let's fly away")
=> #<Song @id=2 @title="Come fly with me" @lyrics="Come fly with me, let's fly, let's fly away"
```

1. If the creation was successful, `#create` will return the newly created `DataMapper::Resource`
2. If it failed, it will return a new resource that is initialized with the given attributes and possible default values declared for that resource, but that's not yet saved
3. To find out whether the creation was successful or not, you can call `#saved?` on the returned resource
4. It will return `true` if the resource was successfully persisted, or `false` otherwise

**first\_or\_create** If you want to either find the first resource matching some given criteria or just create that resource if it can't be found, you can use `#first_or_create`.

```
s = Song.first_or_create(:title => 'New York, New York')
```

This will first try to find a `Song` instance with the given `title`, and if it fails to do so, it will return a newly created `Song` with that `title`.

If the criteria you want to use to query for the resource differ from the attributes you need for creating a new resource, you can pass the attributes for creating a new resource as the second parameter to `#first_or_create`, also in the form of a `#Hash`.

```
s = Song.first_or_create({ :title => 'My Way' }, { :lyrics => '... the end is not near' })
```

This will search for a `Song` named '`My Way`' and if it can't find one, it will return a new `Song` instance with its name set to '`My Way`' and the `lyrics` set to `... the end is not near`

1. You can see that for creating a new resource, both hash arguments will be merged so you don't need to specify the query criteria again in the second argument Hash that lists the attributes for creating a new resource
2. However, if you really need to create the new resource with different values from those used to query for it, the second Hash argument will overwrite the first one.

```
s = Song.first_or_create({ :title => 'My Way' }, {
 :title => 'My Way Home',
 :lyrics => '... the end is not near'
})
```

This will search for a `Song` named 'My Way' but if it fails to find one, it will return a `Song` instance with its title set to 'My Way Home' and its `lyrics` set to '... the end is not near'.

### Comprobando con sqlite3

Podemos abrir la base de datos con el gestor de base de datos y comprobar que las tablas y los datos están allí:

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ sqlite3 development.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"

sqlite> .schema
CREATE TABLE "songs" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
 "title" VARCHAR(50), "lyrics" TEXT, "length"
 INTEGER, "released_on" TIMESTAMP);
sqlite> select * from songs;
1|My Way|And now, the end is near ...|435|
2|Come fly with me|Come fly with me, let's fly, let's fly away ...|199|
sqlite>
```

**Búsquedas y Consultas** DataMapper has methods which allow you to grab a single record by key, the first match to a set of conditions, or a collection of records matching conditions.

```
song = Song.get(1) # get the song with primary key of 1.
song = Song.get!(1) # Or get! if you want an ObjectNotFoundError on failure
song = Song.first(:title => 'Girl') # first matching record with the title 'Girl'
song = Song.last(:title => 'Girl') # last matching record with the title 'Girl'
songs = Song.all # all songs

[29] pry(main)> Song.count
=> 2

[30] pry(main)> Song.all
=> [#<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>, #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=199 @released_on=nil>]

[31] pry(main)> Song.get(1)
=> #<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>

[32] pry(main)> Song.first
=> #<Song @id=1 @title=nil @lyrics=<not loaded> @length=nil @released_on=nil>

[33] pry(main)> Song.last
=> #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=199 @released_on=nil>

[35] pry(main)> x = Song.first(title: 'Come fly with me')
=> #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=199 @released_on=nil>

[44] pry(main)> y = Song.first(title: 'My Way')
=> #<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=435 @released_on=nil>

[45] pry(main)> y.length
=> 435

[46] pry(main)> y.update(length: 275)
=> true
```

En Sqlite3:

```
sqlite> select * from songs;
1|My Way|And now, the end is near ...|275|
2|Come fly with me|Come fly with me, let's fly, let's fly away ...|199|
```

## Borrando

```
[47] pry(main)> Song.create(title: "One less lonely girl")
=> #<Song @id=3 @title="One less lonely girl" @lyrics=<not loaded> @length=<not loaded> @released_on=<not loaded>
[48] pry(main)> Song.last.destroy
=> true
[49] pry(main)> Song.all
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=275 @released_on=nil>, #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=105 @released_on=nil>, #<Song @id=3 @title="One less lonely girl" @lyrics=<not loaded> @length=<not loaded> @released_on=<not loaded>]
```

**Búsqueda con Condiciones** Rather than defining conditions using SQL fragments, we can actually specify conditions using a hash.

The examples above are pretty simple, but you might be wondering how we can specify conditions beyond equality without resorting to SQL. Well, thanks to some clever additions to the `Symbol` class, it's easy!

```
exhibitions = Exhibition.all(:run_time.gt => 2, :run_time.lt => 5)
=> SQL conditions: 'run_time > 1 AND run_time < 5'
```

Valid symbol operators for the conditions are:

```
gt # greater than
lt # less than
gte # greater than or equal
lte # less than or equal
not # not equal
eq # equal
like # like
```

Veamos un ejemplo de uso con nuestra clase `Song`:

```
[31] pry(main)> Song.all.each do |s|
[31] pry(main)* s.update(length: rand(400))
[31] pry(main)* end
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=122 @released_on=nil>,
 #<Song @id=2 @title="Come fly with me" @lyrics=<not loaded> @length=105 @released_on=nil>,
 #<Song @id=4 @title="Girl from Ipanema" @lyrics=<not loaded> @length=389 @released_on=nil>]
[32] pry(main)> long = Song.all(:length.gt => 120)
=> [#<Song @id=1 @title="My Way" @lyrics=<not loaded> @length=122 @released_on=nil>,
 #<Song @id=4 @title="Girl from Ipanema" @lyrics=<not loaded> @length=389 @released_on=nil>]
```

**Insertando SQL** Sometimes you may find that you need to tweak a query manually:

```
[40] pry(main)> songs = repository(:default).adapter.select('SELECT title FROM songs WHERE length > 120')
=> ["My Way", "Girl from Ipanema"]
```

Note that this will not return `Song` objects, rather the raw data straight from the database

## main.rb

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat main.rb
require 'sinatra'
require 'slim'
require 'sass'
require './song'

configure do
 enable :sessions
 set :username, 'frank'
 set :password, 'sinatra'
end

configure :development do
 DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/development.db")
end

configure :production do
 DataMapper.setup(:default, ENV['DATABASE_URL'])
end

get('/styles.css'){ scss :styles }

get '/' do
 slim :home
end

get '/about' do
 @title = "All About This Website"
 slim :about
end

get '/contact' do
 slim :contact
end

not_found do
 slim :not_found
end

get '/login' do
 slim :login
end

post '/login' do
 if params[:username] == settings.username && params[:password] == settings.password
 session[:admin] = true
 redirect to('/songs')
 else
 slim :login
 end
end
```

```

get '/logout' do
 session.clear
 redirect to('/login')
end

```

## 46.5. Ejemplo de uso de Sinatra y DataMapper: Acortador de URLs

- Acortador de URLs
- [~/srcSTW/url\_shortener\_with\_datamapper(initial)]\$ pwd -P  
 /Users/casiano/local/src/ruby/STW/url\_shortener\_with\_datamapper  
 [~/srcSTW/url\_shortener\_with\_datamapper(master)]\$ git remote -v  
 origin git@github.com:crguezl/url\_shortener\_with\_datamapper.git (fetch)  
 origin git@github.com:crguezl/url\_shortener\_with\_datamapper.git (push)

## 46.6. Configurando la Base de Datos en Heroku con DataMapper. Despliegue

Heroku utiliza la base de datos PostgreSQL y una URL en una variable de entorno ENV['DATABASE\_URL'].

```

configure :development, :test do
 DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/development.db")
end

configure :production do
 DataMapper.setup(:default, ENV['DATABASE_URL'])
end

```

Estas líneas especifican que se usa SQLite en desarrollo y testing y PostgreSQL en producción. Obsérvese que el `Gemfile` debe estar coherente:

```

[~/sinatra/sinatra-datamapper-jump-start(master)]$ cat Gemfile
source 'https://rubygems.org'
gem "sinatra"
gem "slim"
gem "sass"
gem "dm-core"
gem "dm-migrations"
gem "thin"
gem "pg", :group => :production
gem "dm-postgres-adapter", :group => :production
gem "dm-sqlite-adapter", :group => [:development, :test]

```

o mejor:

```

group :production do
 gem "pg"
 gem "dm-postgres-adapter"
end

group :development, :test do
 gem "dm-sqlite-adapter"
end

```

## Bundle: installing groups. General considerations

By default, bundle install will install all gems in all groups in your Gemfile, except those declared for a different platform<sup>1</sup>.

However, you can explicitly tell bundler to skip installing certain groups with the `--without` option. This option takes a space-separated list of groups.

While the `--without` option will skip installing the gems in the specified groups, it will still download those gems and use them to resolve the dependencies of every gem in your Gemfile.

This is so that installing a different set of groups on another machine (such as a production server) will not change the gems and versions that you have already developed and tested against.

Bundler guarantees that the third-party code you are running in development and testing is also the third-party code you are running in production. You can choose to exclude some of that code in different environments, but you will never be caught flat-footed by different versions of third-party code being used in different environments.

For a simple illustration, consider the following Gemfile:

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

gem "sinatra"

group :production do
 gem "rack-perf-tools-profiler"
end
```

In this case, `sinatra` depends on any version of `Rack (>= 1.0)`, while `rack-perf-tools-profiler` depends on `1.x (~> 1.0)`.

When you run `bundle install --without production` in development, we look at the dependencies of `rack-perf-tools-profiler` as well.

That way, you do not spend all your time developing against `Rack 2.0`, using new APIs unavailable in `Rack 1.x`, only to have bundler switch to `Rack 1.2` when the `production` group is used.

This should not cause any problems in practice, because bundle does not attempt to install the gems in the excluded groups, and only evaluates them as part of the dependency resolution process.

This also means that you cannot include different versions of the same gem in different groups, because doing so would result in different sets of dependencies used in development and production.

Because of the vagaries of the dependency resolution process, this usually affects more than just the gems you list in your Gemfile, and can (surprisingly) radically change the gems you are using.

## Recordando los pasos para el despliegue en Heroku

```
heroku create ...
git push heroku master
heroku open
heroku logs --source app
```

## Creando/Modificando la base de datos en Heroku con heroku run console

Ahora ejecutamos la consola de heroku:

```
heroku run console
```

lo que nos abre una sesión `irb`.

Ahora creamos la base de datos en Heroku:

---

<sup>1</sup> Platforms are essentially identical to groups, except that you do not need to use the `--without` install-time flag to exclude groups of gems for other platforms

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku run console
Running `console` attached to terminal... up, run.8011
irb(main):001:0> require './main'
=> true
irb(main):002:0> DataMapper.auto_migrate!
=> #<DataMapper::DescendantSet:0x007fb89c878230 @descendants=#<DataMapper::SubjectSet:0x007fb8
```

Véase también la práctica TicTacToe 33.28 y el capítulo *Despliegue en Heroku* 52.

## 46.7. Asociaciones Simples

- Associations are a way of declaring relationships between models, for example a blog Post "has many" Comments or a Comment belongs to a Post.
- Associations add a series of methods to your models which allow you to create relationships and retrieve related models along with a few other useful features.
- Which records are related to which are determined by their *foreign keys* (*claves foráneas*).

The types of associations currently in DataMapper are:

```
has n
has 1
belongs_to
has n, :things, :through => Resource
has n, :things, :through => :model
```

Este código se encuentra en: app.rb

```
[~/rubytesting/database/datamapper/two(master)]$ cat app.rb
require 'data_mapper'

DataMapper::Logger.new($stdout, :debug)
DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/example.db")

class Post
 include DataMapper::Resource

 property :id, Serial
 property :text, Text

 has n, :comments
end

class Comment
 include DataMapper::Resource

 property :id, Serial
 property :text, Text

 belongs_to :post # defaults to :required => true
end
```

```

DataMapper.finalize

require 'dm-migrations'

#DataMapper.auto_migrate!
DataMapper.auto_upgrade!

```

The `belongs_to` method accepts a few options.

- `belongs_to` relationships will be `required` by default

You can make the parent resource optional by passing `:required => false` as an option to `belongs_to`.

- If the relationship makes up (part of) the key of a model, you can tell DataMapper to include it as part of the primary key by adding the `:key => true` option.

Arrancamos Pry

```
[~/rubytesting/database/datamapper/two]$ pry
```

```
y cargamos ./app.rb
```

```
[1] pry(main)> load './app.rb'
~ (0.000397) PRAGMA table_info("posts")
~ (0.000019) SELECT sqlite_version(*)
~ (0.001237) CREATE TABLE "posts" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "text" TEXT)
~ (0.000011) PRAGMA table_info("comments")
~ (0.000986) CREATE TABLE "comments" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "text" TEXT)
~ (0.000964) CREATE INDEX "index_comments_post" ON "comments" ("post_id")
=> true
```

Este es el esquema de la base de datos creada a partir de las clases Post y Comment:

```
[~/rubytesting/database/datamapper/two(master)]$ sqlite3 example.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"

sqlite> .schema
CREATE TABLE "comments" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
 "text" TEXT,
 "post_id" INTEGER NOT NULL);
CREATE TABLE "posts" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
 "text" TEXT);
CREATE INDEX "index_comments_post" ON "comments" ("post_id");
```

Ahora podemos crear un objeto Post:

```
[2] pry(main)> pp = Post.create(:text => 'hello world!')
~ (0.000888) INSERT INTO "posts" ("text") VALUES ('hello world!')
=> #<Post @id=1 @text="hello world!">
```

Sus comments están vacíos:

```
[3] pry(main)> pp.comments
~ (0.000051) SELECT "id", "post_id" FROM "comments" WHERE "post_id" = 1 ORDER BY "id"
=> []
```

Adding resources to a relationship, can be done in different ways.

- [2] pry(main)> co = Comment.new(:text => "Nice greeting!")
=> #<Comment @id=nil @text="Nice greeting!" @post\_id=nil>
[3] pry(main)> pp = Post.create(:text => 'hello world!')
~ (0.001294) INSERT INTO "posts" ("text") VALUES ('hello world!')
=> #<Post @id=1 @text="hello world!">
[4] pry(main)> co.post=pp
=> #<Post @id=1 @text="hello world!">
[5] pry(main)> co.save
~ (0.002974) INSERT INTO "comments" ("text", "post\_id") VALUES ('Nice greeting!', 1)
=> true
[7] pry(main)> pp.comments
~ (0.000074) SELECT "id", "post\_id" FROM "comments" WHERE "post\_id" = 1 ORDER BY "id"
=> [#<Comment @id=1 @text=<not loaded> @post\_id=1>]
[8] pry(main)> pp.comments[0].text
~ (0.000193) SELECT "id", "text" FROM "comments" WHERE "id" = 1 ORDER BY "id"
=> "Nice greeting!"

- `belongs_to` relationships will be required by default

We can make the parent resource optional by passing `:required => false` as an option to `belongs_to`.

- [12] pry(main)> c2 = Comment.create(:text => 'silly comment')
=> #<Comment @id=nil @text="silly comment" @post\_id=nil>
[13] pry(main)> c2.save
=> false
[14] pry(main)> c2.post = pp
=> #<Post @id=1 @text="hello world!">
[15] pry(main)> c2.save
~ (0.001910) INSERT INTO "comments" ("text", "post\_id") VALUES ('silly comment', 1)
=> true
- [6] pry(main)> pp.comments.push(Comment.create(:text => "Nice greeting!"))
=> [#<Comment @id=nil @text="Nice greeting!" @post\_id=1>]
[4] pry(main)> pp
=> #<Post @id=1 @text="hello world!">
[5] pry(main)> pp.comments
=> [#<Comment @id=nil @text="Nice greeting!" @post\_id=1>]

Esto no ha guardado `pp` en la base de datos. Ha sido modificado en memoria. Tampoco se ha guardado en la base de datos el objeto comentario.

```
[7] pry(main)> pp.save
~ (0.002523) INSERT INTO "comments" ("text", "post_id") VALUES ('Nice greeting!', 1)
=> true
```

Ahora las tablas contienen:

```
sqlite> select * from posts;
1|hello world!
sqlite> select * from comments;
1|Nice greeting!|1
```

- [8] pry(main)> c2 = Comment.create(:text => "Do we know each other?")
=> #<Comment @id=nil @text="Do we know each other?" @post\_id=nil>

```

[9] pry(main)> c2.post= pp
=> #<Post @id=1 @text="hello world!">
[10] pry(main)> c2
=> #<Comment @id=nil @text="Do we know each other?" @post_id=1>
[11] pry(main)> c2.save
~ (0.007430) INSERT INTO "comments" ("text", "post_id") VALUES ('Do we know each other?')
=> true
[12] pry(main)> pp.comments
=> [#<Comment @id=1 @text="Nice greeting!" @post_id=1>]

[16] pry(main)> aa = Post.all
~ (0.000065) SELECT "id" FROM "posts" ORDER BY "id"
=> [#<Post @id=1 @text=<not loaded>>]
[17] pry(main)> aa[0].text
~ (0.000052) SELECT "id", "text" FROM "posts" WHERE "id" = 1 ORDER BY "id"
=> "hello world!"
[20] pry(main)> aa[0].comments
~ (0.000052) SELECT "id", "post_id" FROM "comments" WHERE "post_id" = 1 ORDER BY "id"
=> [#<Comment @id=1 @text=<not loaded> @post_id=1>,
 #<Comment @id=2 @text=<not loaded> @post_id=1>]

```

## 46.8. Asociaciones Through

```

[~/rubytesting/database/datamapper/through]$ cat app.rb
require 'data_mapper'

DataMapper::Logger.new($stdout, :debug)
DataMapper.setup(:default, "sqlite3://#{Dir.pwd}/example.db")

class Photo
 include DataMapper::Resource

 property :id, Serial

 has n, :taggings
 has n, :tags, :through => :taggings
end

class Tag
 include DataMapper::Resource

 property :id, Serial

 has n, :taggings
 has n, :photos, :through => :taggings
end

class Tagging
 include DataMapper::Resource

 belongs_to :tag, :key => true
 belongs_to :photo, :key => true
end

```

```

DataMapper.finalize

require 'dm-migrations'

DataMapper.auto_migrate!
#DataMapper.auto_upgrade!

[1] pry(main)> load 'app.rb'
NameError: uninitialized constant Photo::DataMapper
from app.rb:2:in `<class:Photo>'

[2] pry(main)> load 'app.rb'
~ (0.003772) PRAGMA table_info("photos")
~ (0.000014) PRAGMA table_info("tags")
~ (0.000008) PRAGMA table_info("taggings")
~ (0.000027) SELECT sqlite_version(*)
~ (0.000073) DROP TABLE IF EXISTS "photos"
~ (0.000009) PRAGMA table_info("photos")
~ (0.001200) CREATE TABLE "photos" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT)
~ (0.000126) DROP TABLE IF EXISTS "tags"
~ (0.000025) PRAGMA table_info("tags")
~ (0.000993) CREATE TABLE "tags" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT)
~ (0.000101) DROP TABLE IF EXISTS "taggings"
~ (0.000015) PRAGMA table_info("taggings")
~ (0.001372) CREATE TABLE "taggings" ("tag_id" INTEGER NOT NULL, "photo_id" INTEGER NOT NULL,
~ (0.000965) CREATE INDEX "index_taggings_tag" ON "taggings" ("tag_id")
~ (0.001084) CREATE INDEX "index_taggings_photo" ON "taggings" ("photo_id")
=> true

SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE "photos" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT);
CREATE TABLE "taggings" ("tag_id" INTEGER NOT NULL, "photo_id" INTEGER NOT NULL, PRIMARY KEY("tag_id", "photo_id"));
CREATE TABLE "tags" ("id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT);
CREATE INDEX "index_taggings_photo" ON "taggings" ("photo_id");
CREATE INDEX "index_taggings_tag" ON "taggings" ("tag_id");

[2] pry(main)> ph = Photo.create
~ (0.002460) INSERT INTO "photos" DEFAULT VALUES
=> #<Photo @id=1>

[4] pry(main)> tg = Tag.create()
~ (0.002559) INSERT INTO "tags" DEFAULT VALUES
=> #<Tag @id=1>

[5] pry(main)> g = Tagging.create(:photo => ph, :tag => tg)
~ (0.002547) INSERT INTO "taggings" ("tag_id", "photo_id") VALUES (1, 1)
=> #<Tagging @tag_id=1 @photo_id=1>

[7] pry(main)> ph.tags
~ (0.000100) SELECT "tags"."id" FROM "tags" INNER JOIN "taggings" ON "tags"."id" = "taggings".tag_id
=> [#<Tag @id=1>]

[8] pry(main)> tg.photos
~ (0.000080) SELECT "photos"."id" FROM "photos" INNER JOIN "taggings" ON "photos"."id" = "taggings".photo_id
=> [#<Photo @id=1>]

```

```

9] pry(main)> tg2 = Tag.create(:photos => [ph])
~ (0.005787) INSERT INTO "tags" DEFAULT VALUES
~ (0.000534) SELECT "tag_id", "photo_id" FROM "taggings" WHERE ("photo_id" = 1 AND "tag_id" =
~ (0.000049) SELECT "tag_id", "photo_id" FROM "taggings" WHERE ("tag_id" = 2 AND "photo_id" =
~ (0.001444) INSERT INTO "taggings" ("tag_id", "photo_id") VALUES (2, 1)
=> #<Tag @id=2>
[10] pry(main)> ph2 = Photo.create(:tags => [tg, tg2])
~ (0.001175) INSERT INTO "photos" DEFAULT VALUES
~ (0.000056) SELECT "tag_id", "photo_id" FROM "taggings" WHERE ("tag_id" = 1 AND "photo_id" =
~ (0.000045) SELECT "tag_id", "photo_id" FROM "taggings" WHERE ("photo_id" = 2 AND "tag_id" =
~ (0.001047) INSERT INTO "taggings" ("tag_id", "photo_id") VALUES (1, 2)
~ (0.000079) SELECT "tag_id", "photo_id" FROM "taggings" WHERE ("photo_id" = 2 AND "tag_id" =
~ (0.001070) INSERT INTO "taggings" ("tag_id", "photo_id") VALUES (2, 2)
=> #<Photo @id=2>
[11] pry(main)> ph2.tags
=> [#<Tag @id=1>, #<Tag @id=2>]

```

## 46.9. Práctica: Acortador de URLs

Expanda el ejemplo *Acortador de URLs* presentado en la sección 46.5 para que admita además abreviaciones elegidas por el usuario.

La entrada sería similar a esta:

### Otros Requisitos

- Despliegue en Heroku. Repase *Configurando la Base de Datos en Heroku con DataMapper. Despliegue* en 46.6 y Deploying Rack-based Apps
- Si el usuario esta autenticado se le ofrece una opción para ver la lista de sus links. Si no esta autenticado no se visualiza ninguna lista pero si que puede acortar un link.

- Use el (los) proveedor(es) de OAuth que prefiera: Google o GitHub o Twitter, etc.

```

require 'omniauth-oauth2'
require 'omniauth-google-oauth2'
require 'omniauth-facebook'

```

- Aisle la parte de autentificación en un fichero aparte auth.rb
- Utilice session para guardar al información sobre el usuario cuando se nos de en la ruta de callback para utilizarla posteriormente:

```

get '/auth/:name/callback' do
 session[:auth] = @auth = request.env['omniauth.auth']
 session[:name] = @auth['info'].name
 session[:image] = @auth['info'].image
 session[:url] = @auth['info'].urls.values[0]
 session[:email] = @auth['info'].email

 flash[:notice] =
 %Q{<div class="chuchu">Autenticado como #{@auth['info'].name}</div>}

 # Añadir a la base de datos directamente, siempre y cuando no exista
 if !User.first(:username => session[:email])
 u = User.create(:username => session[:email])
 u.save
 end
end

```

```
 end
```

```
 redirect '/'
 end
```

- Recuerde establecer la ruta por si falla la autenticación:

```
get '/auth/failure' do
 flash[:notice] =
 %Q{<div class="error-auth">Error: #{params[:message]}</div>}
 redirect '/'
end
```

- SFEley/sinatra-flash en GitHub

- Añada pruebas. Repase la sección *Testing en Sinatra* 36
- Escriba las vistas en Haml
  - La sección *Haml* en 97
  - Haml tutorial

#### Véase

- Véase Acortador de URLs.
- La sección *Configurando la Base de Datos en Heroku con DataMapper. Despliegue* en 46.6
- La sección *Haml* en 97
- Haml tutorial
- Deploying Rack-based Apps
- Véase Using Transactions with Ruby DataMapper by Ludo van den Boom Geschreven 21-01-10

### 46.10. Práctica: Estadísticas de Visitas al Acortador de URLs

En esta práctica extendemos la anterior 46.9 con información estadística acerca de las visitas. Se pide que se presenten gráficos estadísticos (barras, etc.) del número de visitas por día, por país, etc.

La información de las visitas se guardará en una tabla `Visit`.

Cada objeto `Visit` representará una visita a la URL corta y contendrá información acerca de la visita:

- la fecha de la visita y
- la procedencia de la misma.

**La Tabla Principal** Un objeto de la tabla principal tiene asociados varios objetos `Visit`.

```
class ShortenedUrl
 include DataMapper::Resource

 property :id, Serial
 property :short, Text
 property :url, Text

 has n, :visits
end
```

**Obteniendo el País para una IP dada** Queremos encontrar el país desde el que viene cada visitante. Cada vez que se crea un objeto `Visit` y que se le asocia con el objeto de la tabla principal intentaremos obtener la dirección IP y a partir de ella la información sobre el país.

Veamos una sesión con Pry para obtener el país asociado con una IP:

```
[1] pry(main)> require 'restclient'
=> true
[2] pry(main)> require 'xmlsimple'
=> true

[3] pry(main)> xml = RestClient.get "http://api.hostip.info/get_xml.php?ip=193.145.124.21";
[4] pry(main)> puts xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<HostipLookupResultSet version="1.0.1" xmlns:gml="http://www.opengis.net/gml" xmlns:xsi="http:
<gml:description>This is the Hostip Lookup Service</gml:description>
<gml:name>hostip</gml:name>
<gml:boundedBy>
 <gml:Null>inapplicable</gml:Null>
</gml:boundedBy>
<gml:featureMember>
 <Hostip>
 <ip>193.145.124.21</ip>
 <gml:name>(Unknown city)</gml:name>
 <countryName>SPAIN</countryName>
 <countryAbbrev>ES</countryAbbrev>
 <!-- Co-ordinates are unavailable -->
 </Hostip>
</gml:featureMember>
</HostipLookupResultSet>
=> nil
[5] pry(main)> xml.class
=> String

[10] pry(main)> cc = XmlSimple.xml_in(xml.to_s)
=> {"version"=>"1.0.1",
 "xmlns:gml"=>"http://www.opengis.net/gml",
 "xmlns:xsi"=>"http://www.w3.org/2001/XMLSchema-instance",
 "xsi:noNamespaceSchemaLocation"=>
 "http://www.hostip.info/api/hostip-1.0.1.xsd",
 "description"=>["This is the Hostip Lookup Service"],
 "name"=>["hostip"],
 "boundedBy"=>[{"Null"=>["inapplicable"]}],
 "featureMember"=>
 [{"Hostip"=>
 [{"ip"=>["193.145.124.21"],
 "name"=>["(Unknown city)"],
 "countryName"=>["SPAIN"],
 "countryAbbrev"=>["ES"]}]}]}
[11] pry(main)> cc["featureMember"][0]['Hostip'][0]['countryAbbrev']
=> ["ES"]
```

To implement this, we use the `after` callback mechanism in the `Visit` object, where we call a method `after` the object is created.

```
after :create, :set_country
```

## La Tabla Visit

Para lograr que cada vez que se crea un objeto Visit se guarde el país en la tabla Visit la clase será algo parecido a esto:

```
class Visit
 include DataMapper::Resource

 property :id, Serial
 property :created_at, DateTime
 property :ip, IPAddress
 property :country, String
 belongs_to :link

 after :create, :set_country

 def set_country
 xml = RestClient.get "http://api.hostip.info/get_xml.php?ip=#{ip}..."
 self.country = XmlSimple.xml_in(xml.to_s, ...
 self.save
 end
 ...
end
```

## Averiguar la IP

Este método ilustra múltiples formas de obtener la IP de la visita:

```
def get_remote_ip(env)
 puts "request.url = #{request.url}"
 puts "request.ip = #{request.ip}"
 if addr = env['HTTP_X_FORWARDED_FOR']
 puts "env['HTTP_X_FORWARDED_FOR'] = #{addr}"
 addr.split(',').first.strip
 else
 puts "env['REMOTE_ADDR'] = #{env['REMOTE_ADDR']}"
 env['REMOTE_ADDR']
 end
end
```

## Gráficos con chartkick

Este ejemplo crguezl/sinatra-chartkick-example:

```
~/sinatra/sinatra-chartkick(master)]$ git remote -v
origin git@github.com:crguezl/sinatra-chartkick-example.git (fetch)
origin git@github.com:crguezl/sinatra-chartkick-example.git (push)
[~/sinatra/sinatra-chartkick(master)]$ pwd -P
/Users/casiano/local/src/ruby/sinatra/sinatra-chartkick
```

muestra como usar chartkick para generar gráficos:

```
[~/sinatra/sinatra-chartkick(master)]$ cat sinchart2.rb
require 'sinatra'
require 'sinatra/reloader' if development?
require 'chartkick'

template :layout do
```

```
<<LAYOUT
<html>
 <head>
 <script src="http://www.google.com/jsapi"></script>
 <script src="chartkick.js"></script>
 </head>
 <body>
 <%= yield %>
 </body>
</html>
LAYOUT
end

template :index do
 <<INDEX
 <%= pie_chart({"Football" => 10, "Basketball" => 5}) %>
INDEX
end

get '/' do
 erb :index
end
```

## Capítulo 47

# Sequel y Sinatra

1. README (tutorial)

## Capítulo 48

# ActiveRecord y Sinatra

El siguiente texto esta tomado de la lista Sinatra Q&A How do I use ActiveRecord migrations?

To use ActiveRecord's migrations with Sinatra (or other non-Rails project), add the following to your Rakefile:

```
namespace :db do
 desc "Migrate the database"
 task(:migrate => :environment) do
 ActiveRecord::Base.logger = Logger.new(STDOUT)
 ActiveRecord::Migration.verbose = true
 ActiveRecord::Migrator.migrate("db/migrate")
 end
end
```

This assumes you have a task called `:environment` which loads your app's environment (requires the right files, sets up the database connection, etc).

Now you can create a directory called `db/migrate` and fill in your migrations. I usually call the first one `001_init.rb`. (I prefer the old sequential method for numbering migrations vs. the datetime method used since Rails 2.1, but either will work.)

- ActiveRecord Migration for cuba and sinatra por Revath S. Kumar
- ActiveRecord migrations without Rails por Wes Bailey
- Up and Running with Sinatra and ActiveRecord por Matt GaidicA

### 48.1. Práctica: Servicio para Abreviar URLs

Escriba un acortador de URLs usando ActiveRecords y sinatra-activerecord Asegúrese de tener instalados:

1. gem install activerecord
2. gem install sinatra-activerecord
3. gem install sqlite3

Este es un ejemplo de estructura de la aplicación en su forma final:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ tree -A
.
|-- Gemfile
|-- Gemfile.lock
|-- README
```

```

|-- Rakefile
|-- app.rb
|-- db
| |-- config.yml
| '-- migrate
| '-- 20121017115717_shortened_urls.rb
|-- shortened_urls.db
|-- shortened_urls_bak.db
`-- views
 |-- index.haml
 |-- layout.haml
 '-- success.haml

```

3 directories, 12 files

Una vez instaladas las gemas implicadas:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat Gemfile
source 'https://rubygems.org'
```

```
#gem 'alphadecimal'
gem 'sinatra-activerecord'
gem 'sqlite3'
```

con bundle install, procedemos a añadir objetivos al Rakefile:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat Rakefile
$: << '.' # add current path to the search path
require 'sinatra/activerecord/rake'
require 'app'

desc "Reset the data base to initial state"
task :clean do
 sh "mv shortened_urls.db tmp/"
 sh "mv db/migrate /tmp/"
end

desc "Create the specific ActiveRecord migration for this app"
task :create_migration do
 sh "rake db:create_migration NAME=create_shortened_urls"
end

desc "shows the code you have to have in your db/migrate/#number_shortened_urls.rb file"
task :edit_migration do
 source = <<EOS
class ShortenedUrls < ActiveRecord::Migration
 def up
 create_table :shortened_urls do |t|
 t.string :url
 end
 add_index :shortened_urls, :url
 end
 def down

```

```

 drop_table :shortened_urls
 end
end
EOS
 puts "Edit the migration and insert this code:"
 puts source
end

desc "run the url shortener app"
task :run do
 sh "ruby app.rb"
end

```

Este Rakefile tiene los siguientes objetivos:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ rake -T
rake clean # Reset the data base to initial state
rake create_migration # Create the specific ActiveRecord migration for this app
rake db:create_migration # create an ActiveRecord migration in ./db/migrate
rake db:migrate # migrate the database (use version with VERSION=n)
rake db:rollback # roll back the migration (use steps with STEP=n)
rake edit_migration # shows the code you have to have in your db/migrate/#number_shorten
rake run # run the url shortener app
[~/srcSTW/url_shortener_with_active_records(master)]$
```

De ellos los tres que nos importan ahora son `db:create_migration`, `db:migrate` y `db:rollback` que son creados por la línea `require 'sinatra/activerecord/rake'`.

Las migraciones nos permiten gestionar la evolución de un esquema utilizado por varias bases de datos. Es una solución al problema habitual de añadir un campo para proveer una nueva funcionalidad en nuestra base de datos, pero no tener claro como comunicar dicho cambio al resto de los desarrolladores y al servidor de producción. Con las migraciones podemos describir las transformaciones mediante clases autocintenidas que pueden ser añadadas a nuestro repositorio `git` y ejecutadas contra una base de datos que puede estar una, dos o cinco versiones atrás.

Comenzemos configurando el modelo para nuestra aplicación. En el directorio `db` creamos el fichero `config.yml`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat db/config.yml
development:
 adapter: sqlite3
 encoding: utf8
 database: shortened_urls_dev.sqlite

test:
 adapter: sqlite3
 encoding: utf8
 database: shortened_urls_test.sqlite

production:
 adapter: sqlite3
 encoding: utf8
 database: shortened_urls_live.sqlite
```

Comenzaremos creando nuestro modelo. Para ello ejecutamos `rake db:create_migration`. Como vemos debemos pasar una opción `NAME` para nuestra migración. En nuestro caso pasamos `NAME=create_shortened`. Esto crea la carpeta `db/migrate` que contienen nuestra migración.

```
[~/srcSTW/url_shortener_with_active_records(master)]$ tree db
db
|-- config.yml
 '-- migrate
 '-- 20121017115717_shortened_urls.rb
```

ahora rellenamos los métodos up y el down:

```
1 directory, 2 files
[~/srcSTW/url_shortener_with_active_records(master)]$ cat db/migrate/20121017115717_shortened_
class ShortenedUrls < ActiveRecord::Migration
 def up
 create_table :shortened_urls do |t|
 t.string :url
 end
 add_index :shortened_urls, :url
 end

 def down
 drop_table :shortened_urls
 end
end
```

cuando ejecutamos `rake db:migrate` se ejecuta la migración y crea la base de datos con la tabla `shortened_urls`.

En nuestro fichero `app.rb` creamos nuestro modelo `ShortenedUrl`. Para ello escribimos:

```
class ShortenedUrl < ActiveRecord::Base
end
```

Después incluimos algunas validaciones:

```
class ShortenedUrl < ActiveRecord::Base
 # Validates whether the value of the specified attributes are unique across the system.
 validates_uniqueness_of :url
 # Validates that the specified attributes are not blank
 validates_presence_of :url
 #validates_format_of :url, :with => /.*/
 validates_format_of :url,
 :with => %r{^(https?|ftp)://.+}i,
 :allow_blank => true,
 :message => "The URL must start with http://, https://, or ftp:// ."
end
```

A continuación escribimos las rutas:

```
get '/' do
```

```
end
```

```
post '/' do
end
```

Creamos también una ruta `get` para redireccionar la URL acortada a su destino final:

```
get '/:shortened' do
end
```

Supongamos que usamos el `id` de la URL en la base de datos para acortar la URL. Entonces lo que tenemos que hacer es encontrar mediante el método `find` la URL:

```
get '/:shortened' do
 short_url = ShortenedUrl.find(params[:shortened].to_i(36))
 redirect short_url.url
end
```

Mediante una llamada de la forma `Model.find(primary_key)` obtenemos el objeto correspondiente a la clave primaria especificada. que casa con las opciones suministradas.

El SQL equivalente es:

```
SELECT * FROM shortened_urls WHERE (url.id = params[:shortened].to_i(36)) LIMIT 1
```

`Model.find(primary_key)` genera una excepción `ActiveRecord::RecordNotFound` si no se encuentra ningún registro.

Veamos una sesión on `sqlite3`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ sqlite3 shortened_urls.db
SQLite version 3.7.7 2011-06-25 16:35:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE "schema_migrations" ("version" varchar(255) NOT NULL);
CREATE TABLE "shortened_urls" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "url" varchar(255));
CREATE INDEX "index_shortened_urls_on_url" ON "shortened_urls" ("url");
CREATE UNIQUE INDEX "unique_schema_migrations" ON "schema_migrations" ("version");
sqlite> select * from shortened_urls;
1|https://mail.google.com/mail/u/0/
2|https://plus.google.com/u/0/
3|http://campusvirtual.ull.es/1213m2/mod/forum/discuss.php?d=5629
4|http://www.sinatrarb.com/intro#Accessing%20Variables%20in%20Templates
sqlite> select * from shortened_urls where (id = 3) limit 1;
3|http://campusvirtual.ull.es/1213m2/mod/forum/discuss.php?d=5629
sqlite> .quit
```

Este es el código completo de `app.rb`:

```
$ cat app.rb
require 'sinatra'
require 'sinatra/activerecord'
require 'haml'

set :database, 'sqlite3:///shortened_urls.db'
#set :address, 'localhost:4567'
set :address, 'exthost.etsii.ull.es:4567'

class ShortenedUrl < ActiveRecord::Base
 # Validates whether the value of the specified attributes are unique across the system.
 validates_uniqueness_of :url
 # Validates that the specified attributes are not blank
 validates_presence_of :url
 #validates_format_of :url, :with => /.*/
 validates_format_of :url,
 :with => %r{^(https?|ftp)://.+}i,
```

```

:allow_blank => true,
:message => "The URL must start with http://, https://, or ftp:// ."
end

get '/' do
 haml :index
end

post '/' do
 @short_url = ShortenedUrl.find_or_create_by_url(params[:url])
 if @short_url.valid?
 haml :success, :locals => { :address => settings.address }
 else
 haml :index
 end
end

get '/:shortened' do
 short_url = ShortenedUrl.find(params[:shortened].to_i(36))
 redirect short_url.url
end

```

Las Vistas. Primero el fichero `views/layout.haml`:

```

$ cat views/layout.haml
!!!
%html
 %body
 =yield
 %form(action="/" method="POST")
 %label(for="url")
 %input(type="text" name="url" id="url" accesskey="s")
 %input(type="submit" value="Shorten")

```

Creamos un formulario que envía con `method="POST"` a la raíz `action="/".` tiene un elemento `input` para obtener la URL.

El formulario es procesado por la ruta `post '/':`

```

post '/' do
 @short_url = ShortenedUrl.find_or_create_by_url(params[:url])
 if @short_url.valid?
 haml :success, :locals => { :address => settings.address }
 else
 haml :index
 end
end

```

El método `find_or_create` encontrará la URL o creará una nueva.

El fichero `views/index.haml`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat views/index.haml
- if @short_url.present? && !@short_url.valid?
 %p Invalid URL: #{@short_url.url}
```

El fichero `views/success.haml`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat views/success.haml
%p #{params}
%p http://#{address}/#{@short_url.id.to_s(36)}
```

Puede ir añadiendo extensiones a la práctica:

1. Añada una opción para mostrar la lista de URLs abreviadas El método `find` puede serle útil:

```
get '/show' do
 urls = ShortenedUrl.find(:all)
 ...
 haml :show
end
```

2. Añada una opción para buscar por una abreviación y mostrar la URL
  3. Añada una opción para buscar por una URL y mostrar la abreviación
  4. Añada una opción que permita una abreviación personalizada, siempre que esté libre. Por ejemplo, abreviar `http://www.sinatrararb.com/documentation` a `http://localhost:4567/sindoc`
- Esto obliga a poner una opción para ello en el formulario:

```
%form(action="/" method="POST")
 %label(for="url") URL
 %input(type="text" name="url" id="url" accesskey="s")
 %br
 %label(for="custom") Custom Shortened URL(optional)
 %input(type="text" name="custom" id="custom" accesskey="t")
 %br
 %input(type="submit" value="Shorten" class="btn btn-primary")
```

y a comprobar de alguna manera si la opción `custom` contiene algo.

## Véase

1. URL Shortner App using Ruby, Sinatra and MongoDB (originally Redis) en Github

## 48.2. Práctica: Servicio para Abreviar URLs Teniendo en Cuenta el País de Visita

Añádale a la práctica 48.1 la funcionalidad de mostrar el número de visitas y el número de visitas por país.

Puede usar para ello la API de `http://www.hostip.info/`:

```
after :create, :set_country

def set_country
 xml = RestClient.get "http://api.hostip.info/get_xml.php?ip=#{ip}"
 self.country = XmlSimple.xml_in(xml.to_s, { 'ForceArray' => false })['featureMember'][0]
 self.save
end
```

- XMLSimple le puede ayudar a parsear el XML: <http://xml-simple.rubyforge.org/>
- <https://github.com/sausheong/tinyclone>
- Véase la API de `http://www.hostip.info/`

## Capítulo 49

# Google Plus y Sinatra

### 49.1. Ejemplo Simple

Véase <https://github.com/crguezl/sinatra-google-plus-simple>

```
[~/src/ruby/sinatra/sinatra-google-plus(master)]$ tree
.
|--- Gemfile
|--- Gemfile.lock
|--- README.md
|--- clientID.txt
|--- config.yml
|--- config.yml.example
|--- public
| |--- images
| | '--- favicon.ico
| |--- javascript
| '--- stylesheets
| '--- style.css
|--- sinatragplus.rb
'--- views
 |--- 404.erb
 |--- 500.erb
 |--- index.erb
 |--- layout.erb
 '--- show.erb

5 directories, 14 files

[~/src/ruby/sinatra/sinatra-google-plus(master)]$ cat sinatragplus.rb
sinatragplus.rb
require 'sinatra'
require "sinatra/config_file"
require 'google_plus'

config_file 'config.yml'

error do
 erb :'500'
end
```

```

#class
class GPlus
 def initialize(apikey, gid)
 @apikey = apikey
 @gid = gid
 get_info
 end

 attr_reader :row0, :row1, :row2, :logo
 private
 #Get info about specific G+ ID
 def get_info
 begin
 GooglePlus.api_key = @apikey
 person = GooglePlus::Person.get(@gid.to_i)
 @row0 = person.display_name
 @row1 = person.list_activities.items
 @row2 = person.url
 properties = person.attributes
 puts properties.class
 puts properties['image'].class
 @logo = properties['image']['url']
 properties.each { |key, value| puts "%-20s %20s" % [key, value] }
 rescue Exception => msg
 # display the system generated error message
 puts msg
 end
 end
 end

 # Display Google+ details
 get '/' do
 @key = settings.key
 @user = settings.user
 @gplus = GPlus.new(@key, @user)
 erb :show
 end
end

```

## Capítulo 50

# Las Manos en la Masa: Nuestro Propio Blog Engine

# Capítulo 51

## Depuración en Sinatra

### 51.1. Depurando una Ejecución con Ruby

```
[~/sinatra/sinatra-debug/example1]$ ls
Gemfile Rakefile my_sinatra.rb
Gemfile.lock config.ru rackmiddleware.rb

[~/sinatra/sinatra-debug/example1]$ cat Gemfile
source 'http://rubygems.org'

group :development, :test do
 gem 'awesome_print'
 gem 'racksh'
 gem 'debugger'
 gem 'pry'
 gem 'pry-debugger'
end

[~/sinatra/sinatra-debug/example1]$ cat my_sinatra.rb
my_sinatra.rb
require 'debugger'
require 'sinatra'
require './rackmiddleware'
use RackMiddleware
get '/:p' do |x|
 # debugger
 "Welcome to #{x}"
end

[~/sinatra/sinatra-debug/example1]$ cat rackmiddleware.rb
class RackMiddleware
 def initialize(app)
 @appl = appl
 end

 def call(env)
 debugger
 start = Time.now
 status, headers, body = @appl.call(env) # call our Sinatra app
 stop = Time.now
 puts "Response Time: #{stop-start}" # display on console
 [status, headers, body]
 end
end
```

```

end
end

[~/sinatra/sinatra-debug/example1]$ ruby my_sinatra.rb
== Sinatra/1.4.3 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on localhost:4567, CTRL+C to stop

```

Al conectar al servidor queda en espera:

```
[~/sinatra/sinatra-debug]$ curl 'http://localhost:4567/canarias'
```

En la otra terminal el servidor se detiene en el primer breakpoint señalado:

```

[~/sinatra/sinatra-debug/example1]$ ruby my_sinatra.rb
== Sinatra/1.4.3 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.5.1 codename Straight Razor)
>> Maximum connections set to 1024
>> Listening on localhost:4567, CTRL+C to stop
/Users/casiano/Dropbox/src/ruby/sinatra/sinatra-debug/example1/rackmiddleware.rb:8
start = Time.now

```

```
[3, 12] in /Users/casiano/Dropbox/src/ruby/sinatra/sinatra-debug/example1/rackmiddleware.rb
 3 @appl = appl
 4 end
 5
 6 def call(env)
 7 debugger
=> 8 start = Time.now
 9 status, headers, body = @appl.call(env) # call our Sinatra app
10 stop = Time.now
11 puts "Response Time: #{stop-start}" # display on console
12 [status, headers, body]
```

Ahora podemos ir paso a paso e inspeccionar variables:

```
(rdb:1) p start
2013-07-04 15:40:11 +0100
(rdb:1) n
/Users/casiano/Dropbox/src/ruby/sinatra/sinatra-debug/example1/rackmiddleware.rb:10
stop = Time.now
```

```
[5, 14] in /Users/casiano/Dropbox/src/ruby/sinatra/sinatra-debug/example1/rackmiddleware.rb
 5
 6 def call(env)
 7 debugger
 8 start = Time.now
 9 status, headers, body = @appl.call(env) # call our Sinatra app
=> 10 stop = Time.now
11 puts "Response Time: #{stop-start}" # display on console
12 [status, headers, body]
13 end
14 end
(rdb:1) p body
["Welcome to canarias"]
```

```
(bdb:1) p status
200
(bdb:1) p headers
{"Content-Type"=>"text/html; charset=utf-8", "Content-Length"=>"19"}
```

## Capítulo 52

# Despliegue en Heroku

### 52.1. Introducción

**Prerequisitos** Estos son los prerequisitos (Octubre 2013)

1. Basic Ruby knowledge, including an installed version of Ruby 2.0.0, Rubygems, and Bundler.
2. Basic Git knowledge
3. Your application must run on Ruby (MRI) 2.0.0.
4. Your application must use Bundler.
5. A Heroku user account.

#### Instala el Heroku Toolbelt

1. Crea una cuenta en Heroku
2. El Heroku Toolbelt se compone de:
  - a) Heroku client - CLI tool for creating and managing Heroku apps
  - b) Foreman - an easy option for running your apps locally
  - c) Git - revision control and pushing to Heroku

La primera vez te pedirá las credenciales:

```
$ heroku login
Enter your Heroku credentials.
Email: adam@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub
```

La clave la cargas en la sección SSH keys add key de <https://dashboard.heroku.com/account>

```
[~/rack/rack-rock-paper-scissors(test)]$ heroku --version
heroku-gem/2.39.4 (x86_64-darwin11.4.2) ruby/1.9.3

[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(test)]$ which heroku
[~/src/ruby/STW/alus/1314/angela_hernandez_delgado/P2-STYW(sinatra)]$ which heroku
/usr/bin/heroku
[~/src/ruby/STW/alus/1314/angela_hernandez_delgado/P2-STYW(sinatra)]$ cat /usr/bin/heroku
```

```

#!/usr/local/heroku/ruby/bin/ruby
encoding: UTF-8

resolve bin path, ignoring symlinks
require "pathname"
bin_file = Pathname.new(__FILE__).realpath

add locally vendored gems to libpath
gem_dir = File.expand_path("../vendor/gems", bin_file)
Dir["#{gem_dir}/**/lib"].each do |libdir|
 $:.unshift libdir
end

add self to libpath
$:.unshift File.expand_path("../lib", bin_file)

inject any code in ~/.heroku/client over top
require "heroku/updater"
Heroku::Updater.inject_libpath

start up the CLI
require "heroku/cli"
Heroku.user_agent = "heroku-toolbelt/#{Heroku::VERSION} (#{RUBY_PLATFORM}) ruby/#{RUBY_VERSION}"
Heroku::CLI.start(*ARGV)

```

Seguramente tienes que instalar una versión del toolbet por cada versión de Ruby con la que quieras usarlo.

Para desinstalarlo:

```
$ gem uninstall heroku --all
```

**Actualizaciones** The Heroku Toolbelt will automatically keep itself up to date.

1. When you run a heroku command, a background process will be spawned that checks a URL for the latest available version of the CLI.
2. If a new version is found, it will be downloaded and stored in `~/.heroku/client`.
3. This background check will happen at most once every 5 minutes.
4. The heroku binary will check for updated clients in `~/.heroku/client` before loading the system-installed version.

## Ayuda

```
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(master)]$ heroku --help
Usage: heroku COMMAND [--app APP] [command-specific-options]
```

Primary help topics, type "heroku help TOPIC" for more details:

```

addons # manage addon resources
apps # manage apps (create, destroy)
auth # authentication (login, logout)
config # manage app config vars
domains # manage custom domains

```

```

logs # display logs for an app
ps # manage dynos (dynos, workers)
releases # manage app releases
run # run one-off commands (console, rake)
sharing # manage collaborators on an app

```

Additional topics:

```

account # manage heroku account options
certs # manage ssl endpoints for an app
db # manage the database for an app
drains # display syslog drains for an app
fork # clone an existing app
git # manage git for apps
help # list commands and display help
keys # manage authentication keys
labs # manage optional features
maintenance # manage maintenance mode for an app
pg # manage heroku-postgresql databases
pgbackups # manage backups of heroku postgresql databases
plugins # manage plugins to the heroku gem
regions # list available regions
stack # manage the stack for an app
status # check status of heroku platform
update # update the heroku client
version # display version

```

## Specify Ruby Version and Declare dependencies with a Gemfile

Heroku recognizes an app as Ruby by the existence of a `Gemfile`.

Even if your app has no gem dependencies, you should still create an empty `Gemfile` in order that it appear as a Ruby app.

In local testing, you should be sure to run your app in an isolated environment (via `bundle exec` or an empty RVM gemset), to make sure that all the gems your app depends on are in the `Gemfile`.

In addition to specifying dependencies, you'll want to specify your Ruby Version using the ruby DSL provided by Bundler.

Here's an example `Gemfile` for a Sinatra app:

```

source "https://rubygems.org"
ruby "2.0.0"
gem 'sinatra', '1.1.0'

[~/sinatra/rockpaperscissors(master)]$ cat Gemfile
source 'https://rubygems.org'
gem 'sinatra'
gem 'haml'
gem 'puma'

```

Run `bundle install` to set up your bundle locally.

1. Run:

```
$ bundle install
```

2. This ensures that all gems specified in `Gemfile`, together with their dependencies, are available for your application.

3. Running bundle install also generates a `Gemfile.lock` file, which should be added to your git repository.
4. `Gemfile.lock` ensures that your deployed versions of gems on Heroku match the version installed locally on your development machine.

## Declare process types with Procfile

Process types are declared via a file named `Procfile` placed in the root of your app.

Its format is one process type per line, with each line containing:

```
<process type>: <command>
```

The syntax is defined as:

1. `<process type>` – an alphanumeric string, is a name for your command, such as
  - a) `web`,
  - b) `worker`,
  - c) `urgentworker`,
  - d) `clock`, etc.
2. `<command>` – a command line to launch the process, such as `rake jobs:work`.

The `web` process type is special as it's the only process type that will receive HTTP traffic from Heroku's routers.

1. Use a `Procfile`, a text file in the root directory of your application, to explicitly declare what command should be executed to start a web dyno.
2. Assume for instance, that we want to execute `web.rb` using Ruby. Here's a `Procfile`:

```
web: bundle exec ruby web.rb -p $PORT
```

3. If we are instead deploying a straight Rack app, here's a `Procfile` that can execute our `config.ru`:

```
web: bundle exec rackup config.ru -p $PORT
```

```
[~/sinatra/rockpaperscissors(spec)]$ cat config.ru
#\! -s puma
require './rps'
run RockPaperScissors::App
```

1. This declares a single process type, `web`, and the command needed to run it.
2. The name `web` is important here. It declares that this process type will be attached to the HTTP routing stack of Heroku, and receive web traffic when deployed.

## Foreman

1. It's important when developing and debugging an application that the local development environment is executed in the same manner as the remote environments.
2. This ensures that incompatibilities and hard to find bugs are caught before deploying to production and treats the application as a holistic unit instead of a series of individual commands working independently.

3. Foreman is a command-line tool for running Procfile-backed apps. It's installed automatically by the Heroku Toolbelt.
4. If you had a Procfile with both web and worker process types, Foreman will start one of each process type, with the output interleaved on your terminal
5. We can now start our application locally using Foreman (installed as part of the Toolbelt):

```
$ foreman start
16:39:04 web.1 | started with pid 30728
18:49:43 web.1 | [2013-03-12 18:49:43] INFO WEBrick 1.3.1
18:49:43 web.1 | [2013-03-12 18:49:43] INFO ruby 2.0.0p247 (2013-06-27 revision 41674)
18:49:43 web.1 | [2013-03-12 18:49:43] INFO WEBrick::HTTPServer#start: pid=30728 port=30728
```

6. Our app will come up on port 5000. Test that it's working with `curl` or a web browser, then `Ctrl-C` to exit.

## Setting local environment variables

Config vars saved in the `.env` file of a project directory will be added to the environment when run by Foreman.

For example we can set the `RACK_ENV` to `development` in your environment.

```
$ echo "RACK_ENV=development" >>.env
$ foreman run irb
> puts ENV["RACK_ENV"]
> development
```

Do not commit the `.env` file to source control. It should only be used for local configuration.

## Procfile y Despliegue

Véase la descripción de los contenidos del Procfile en 52.1.

1. A Procfile is not necessary to deploy apps written in most languages supported by Heroku.
2. The platform automatically detects the language, and creates a default web process type to boot the application server.
3. Creating an explicit `Procfile` is recommended for greater control and flexibility over your app.
4. For Heroku to use your Procfile, add the Procfile to the root of your application, then push to Heroku:

```
$ git add .
$ git commit -m "Procfile"
$ git push heroku
...
-----> Procfile declares process types: web, worker
 Compiled slug size is 10.4MB
-----> Launching... done
 http://strong-stone-297.herokuapp.com deployed to Heroku
```

```
To git@heroku.com:strong-stone-297.git
 * [new branch] master -> master
```

## Store your app in Git

```
$ git init
$ git add .
$ git commit -m "init"

[~/sinatra/rockpaperscissors(master)]$ git remote -v
origin git@github.com:crguezl/sinatra-rock-paper-scissors.git (fetch)
origin git@github.com:crguezl/sinatra-rock-paper-scissors.git (push)
```

## Deploy your application to Heroku

Create the app on Heroku:

```
[~/sinatra/rockpaperscissors(master)]$ heroku create
Creating mysterious-falls-4594... done, stack is cedar
http://mysterious-falls-4594.herokuapp.com/ | git@heroku.com:mysterious-falls-4594.git
Git remote heroku added

[~/sinatra/rockpaperscissors(spec)]$ cat Rakefile
desc "start server using rackup ..."
task :default do
 sh "rackup"
end

require 'rspec/core/rake_task'

RSpec::Core::RakeTask.new do |task|
 task.rspec_opts = ["-c", "-f progress"]
 task.pattern = 'spec/**/*_spec.rb'
end

[~/sinatra/rockpaperscissors(master)]$ git remote -v
heroku git@heroku.com:mysterious-falls-4594.git (fetch)
heroku git@heroku.com:mysterious-falls-4594.git (push)
origin git@github.com:crguezl/sinatra-rock-paper-scissors.git (fetch)
origin git@github.com:crguezl/sinatra-rock-paper-scissors.git (push)
```

Deploy your code:

```
[~/sinatra/rockpaperscissors(master)]$ git push heroku master
Counting objects: 31, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (29/29), done.
Writing objects: 100% (31/31), 9.09 KiB, done.
Total 31 (delta 11), reused 0 (delta 0)

-----> Ruby/Rack app detected
-----> Installing dependencies using Bundler version 1.3.2
 Running: bundle install --without development:test --path vendor/bundle --binstubs vend
 Fetching gem metadata from https://rubygems.org/.....
 Fetching gem metadata from https://rubygems.org/..
 Installing tilt (1.4.1)
 Installing haml (4.0.3)
 Installing rack (1.5.2)
 Installing puma (2.0.1)
```

```

Installing rack-protection (1.5.0)
Installing sinatra (1.4.2)
Using bundler (1.3.2)
Your bundle is complete! It was installed into ./vendor/bundle
Post-install message from haml:
HEADS UP! Haml 4.0 has many improvements, but also has changes that may break
your application:
* Support for Ruby 1.8.6 dropped
* Support for Rails 2 dropped
* Sass filter now always outputs <style> tags
* Data attributes are now hyphenated, not underscored
* html2haml utility moved to the html2haml gem
* Textile and Maruku filters moved to the haml-contrib gem
For more info see:
http://rubydoc.info/github/haml/haml/file/CHANGELOG.md
Cleaning up the bundler cache.

-----> Discovering process types
Procfile declares types -> (none)
Default types for Ruby/Rack -> console, rake, web

-----> Compiled slug size: 1.3MB
-----> Launching... done, v4
http://mysterious-falls-4594.herokuapp.com deployed to Heroku

```

```

To git@heroku.com:mysterious-falls-4594.git
 * [new branch] master -> master
[~/sinatra/rockpaperscissors(master)]$
```

## Visit your application

You've deployed your code to Heroku, and specified the process types in a Procfile.

You can now instruct Heroku to execute a process type.

Heroku does this by running the associated command in a dyno - a lightweight container which is the basic unit of composition on Heroku.

Let's ensure we have one dyno running the web process type:

```
$ heroku ps:scale web=1
```

Veamos que dice la ayuda:

```
$ heroku help ps
Usage: heroku ps
```

```
list processes for an app
```

Additional commands, type "heroku help COMMAND" for more details:

```
ps:restart [PROCESS] # ps:restart [PROCESS]
ps:scale PROCESS1=AMOUNT1 ... # ps:scale PROCESS1=AMOUNT1 ...
ps:stop PROCESS # ps:stop PROCESS
```

```
$ heroku help ps:scale
Usage: heroku ps:scale PROCESS1=AMOUNT1 ...
```

```
scale processes by the given amount
```

```
Example: heroku ps:scale web=3 worker+1
```

You can check the state of the app's dynos. The heroku ps command lists the running dynos of your application:

```
$ heroku ps
== web: 'bundle exec ruby web.rb -p $PORT'
web.1: up for 9m
```

Here, one dyno is running.

```
[~/sinatra/sinatra-rock-paper-scissors/sinatra-rockpaperscissors(master)]$ heroku ps
Process State Command

web.1 idle for 8h bundle exec rackup config.ru -p $P..
```

We can now visit the app in our browser with heroku open.

```
[~/sinatra/rockpaperscissors(master)]$ heroku open
Opening http://mysterious-falls-4594.herokuapp.com/
[~/sinatra/rockpaperscissors(master)]$
```

## Dyno sleeping and scaling

1. Having only a single web dyno running will result in the dyno going to sleep after one hour of inactivity.
2. This causes a delay of a few seconds for the first request upon waking.
3. Subsequent requests will perform normally.
4. To avoid this, you can scale to more than one web dyno. For example:

```
$ heroku ps:scale web=2
```

5. For each application, Heroku provides 750 free dyno-hours.
6. Running your app at 2 dynos would exceed this free, monthly allowance, so let's scale back:

```
$ heroku ps:scale web=1
```

## View the logs

Heroku treats logs as streams of time-ordered events aggregated from the output streams of all the dynos running the components of your application.

Heroku's Logplex provides a single channel for all of these events.

View information about your running app using one of the logging commands, heroku logs:

```
$ heroku logs
2013-03-13T04:10:49+00:00 heroku[web.1]: Starting process with command 'bundle exec ruby web.r
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO WEBrick 1.3.1
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO ruby 2.0.0p247 (2013-06-27 r
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO WEBrick::HTTPServer#start: p
```

## Console

1. Heroku allows you to run commands in a one-off dyno - scripts and applications that only need to be executed when needed - using the `heroku run` command.
2. You can use this to launch an interactive Ruby shell (`bundle exec irb`) attached to your local terminal for experimenting in your app's environment:

```
$ heroku run console
Running `console` attached to terminal... up, ps.1
irb(main):001:0>
```

3. By default, `irb` has nothing loaded other than the Ruby standard library. From here you can require some of your application files. Or you can do it on the command line:

```
$ heroku run console -r ./web
```

## Rake

Rake can be run in an attached dyno exactly like the console:

```
$ heroku run rake db:migrate
```

## Using a SQL database

By default, non-Rails apps aren't given a SQL database.

This is because you might want to use a NoSQL database like Redis or CouchDB, or you don't need any database at all.

If you need a SQL database for your app, do this:

1. `$ heroku addons:add heroku-postgresql:dev`
2. You must also add the Postgres gem to your app in order to use your database. Add a line to your `Gemfile` like this:

```
gem 'pg'
```

3. You'll also want to setup a local PostgreSQL database.

## Webserver

By default your app (Rack) will use `Webrick`.

This is fine for testing, but for production apps you'll want to switch to a more robust webserver. On Cedar, they recommend `Unicorn` as the webserver.

## 52.2. Logging

Heroku aggregates three categories of logs for your app:

1. App logs - Output from your application.

This will include logs generated from

- a) within your application,
- b) application server and
- c) libraries.

(Filter: `--source app`)

## 2. System logs -

Messages about actions taken by the Heroku platform infrastructure on behalf of your app, such as:

- a) restarting a crashed process,
- b) sleeping or waking a web dyno, or
- c) serving an error page due to a problem in your app.

(Filter: `--source heroku`)

## 3. API logs -

Messages about administrative actions taken by you and other developers working on your app, such as:

- a) deploying new code,
- b) scaling the process formation, or
- c) toggling maintenance mode.

(Filter: `--source heroku --ps api`)

```
[~/rack/rack-rock-paper-scissors(master)]$ heroku logs --source heroku --ps api
2013-10-23T21:33:41.105090+00:00 heroku[api]: Deploy 5ec1351 by chuchu.chachi.leon@gmail.com
2013-10-23T21:33:41.154690+00:00 heroku[api]: Release v7 created by chuchu.chachi.leon@gmail.com
```

Logplex is designed for collating and routing log messages, not for storage. It keeps the last 1,500 lines of consolidated logs.

Heroku recommends using a separate service for long-term log storage; see Syslog drains for more information.

## Writing to your log

Anything written to standard out (stdout) or standard error (stderr) is captured into your logs. This means that you can log from anywhere in your application code with a simple output statement:

```
puts "Hello, logs!"
```

To take advantage of the realtime logging, you may need to disable any log buffering your application may be carrying out. For example, in Ruby add this to your config.ru:

```
$stdout.sync = true
```

Some frameworks send log output somewhere other than stdout by default.

## To fetch your logs

```
$ heroku logs
2010-09-16T15:13:46.677020+00:00 app[web.1]: Processing PostController#list (for 208.39.138.12)
2010-09-16T15:13:46.677023+00:00 app[web.1]: Rendering template within layouts/application
2010-09-16T15:13:46.677902+00:00 app[web.1]: Rendering post/list
2010-09-16T15:13:46.678990+00:00 app[web.1]: Rendered includes/_header (0.1ms)
2010-09-16T15:13:46.698234+00:00 app[web.1]: Completed in 74ms (View: 31, DB: 40) | 200 OK [ht
2010-09-16T15:13:46.723498+00:00 heroku[router]: at=info method=GET path=/posts host=myapp.h
2010-09-16T15:13:47.893472+00:00 app[worker.1]: 2 jobs processed at 16.6761 j/s, 0 failed ...
```

In this example, the output includes log lines from one of the app's web dynos, the Heroku HTTP router, and one of the app's workers.

The logs command retrieves 100 log lines by default.

## Log message ordering

When retrieving logs, you may notice that the logs are not always in order, especially when multiple components are involved.

This is likely an artifact of distributed computing.

Logs originate from many sources (router nodes, dynos, etc) and are assembled into a single log stream by logplex.

It is up to the logplex user to sort the logs and provide the ordering required by their application, if any.

## Log history limits

You can fetch up to 1500 lines using the `-num` (or `-n`) option:

```
$ heroku logs -n 200
```

Heroku only stores the last 1500 lines of log history. If you'd like to persist more than 1500 lines, use a logging add-on or create your own syslog drain<sup>1</sup>.

## Log format

Each line is formatted as follows:

1. timestamp source[dyno]: message
2. Timestamp - The date and time recorded at the time the log line was produced by the dyno or component. The timestamp is in the format specified by RFC5424, and includes microsecond precision.
3. Source -
  - a) All of your app's dynos (web dynos, background workers, cron) have a source of app.
  - b) All of Heroku's system components (HTTP router, dyno manager) have a source of heroku.
4. Dyno - The name of the dyno or component that wrote this log line. For example, `worker #3` appears as `worker.3`, and the Heroku HTTP router appears as `router`.
5. Message - The content of the log line. Dynos can generate messages up to approximately 1024 bytes in length and longer messages will be truncated.

## Realtime tail

1. Similar to `tail -f`, realtime tail displays recent logs and leaves the session open for realtime logs to stream in.
2. By viewing a live stream of logs from your app, you can gain insight into the behavior of your live application and debug current problems.
3. You may tail your logs using `--tail` (or `-t`).

```
$ heroku logs --tail
```

When you are done, press Ctrl-C to close the session.

---

<sup>1</sup>Logplex drains allow you to forward your Heroku logs to an external syslog server for long-term archiving. You must configure the service or your server to be able to receive syslog packets from Heroku, and then add its syslog URL (which contains the host and port) as a syslog drain.

## Filtering

If you only want to fetch logs with a certain source, a certain dyno, or both, you can use the `--source` (or `-s`) and `--ps` (or `-p`) filtering arguments:

```
$ heroku logs --ps router
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path=/stylesheets/dev-cent
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path=/articles/bundler hos

$ heroku logs --source app
2012-02-07T09:45:47.123456+00:00 app[web.1]: Rendered shared/_search.html.erb (1.0ms)
2012-02-07T09:45:47.123456+00:00 app[web.1]: Completed 200 OK in 83ms (Views: 48.7ms | ActiveR
2012-02-07T09:45:47.123456+00:00 app[worker.1]: [Worker(host:465cf64e-61c8-46d3-b480-362bfd4ec
2012-02-07T09:46:01.123456+00:00 app[web.6]: Started GET "/articles/buildpacks" for 4.1.81.209

$ heroku logs --source app --ps worker
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-362bfd4ec
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-362bfd4ec
```

When filtering by dyno, either the base name, `--ps web`, or the full name, `--ps web.1`, may be used.

You can also combine the filtering switches with `--tail` to get a realtime stream of filtered output.

```
$ heroku logs --source app --tail
```

## 52.3. Troubleshooting

If you push your app and it crashes, `heroku ps` shows state crashed:

```
== web (1X): `bundle exec thin start -R config.ru -e $RACK_ENV -p $PORT'
web.1: crashed 2013/10/24 20:21:34 (~ 1h ago)
```

check your logs to find out what went wrong.

Here are some common problems.

### Failed to require a sourcefile

If your app failed to require a sourcefile, chances are good you're running Ruby 1.9.1 or 1.8 in your local environment.

The load paths have changed in Ruby 1.9 which applies to Ruby 2.0.

Port your app forward to Ruby 2.0.0 making certain it works locally before trying to push to Cedar again.

**Encoding error** Ruby 1.9 added more sophisticated encoding support to the language which applies to Ruby 2.0.

Not all gems work with Ruby 2.0. If you hit an encoding error, you probably haven't fully tested your app with Ruby 2.0.0 in your local environment.

Port your app forward to Ruby 2.0.0 making certain it works locally before trying to push to Cedar again.

### Missing a gem

If your app crashes due to missing a gem, you may have it installed locally but not specified in your Gemfile.

You must isolate all local testing using `bundle exec`.

For example, don't run `ruby web.rb`, run

```
bundle exec ruby web.rb
```

```
Don't run rake db:migrate, run
```

```
bundle exec rake db:migrate.
```

Another approach is to create a blank RVM gemset to be absolutely sure you're not touching any system-installed gems:

```
$ rvm gemset create myapp
$ rvm gemset use myapp
```

## Runtime dependencies on development/test gems

If you're still missing a gem when you deploy, check your Bundler groups.

Heroku builds your app without the `development` or `test` groups, and if your app depends on a gem from one of these groups to run, you should move it out of the group.

One common example using the `RSpec` tasks in your Rakefile. If you see this in your Heroku deploy:

```
$ heroku run rake -T
Running 'rake -T' attached to terminal... up, ps.3
rake aborted!
no such file to load -- rspec/core/rake_task
```

Then you've hit this problem.

First, duplicate the problem locally like so:

```
$ bundle install --without development:test
...
$ bundle exec rake -T
rake aborted!
no such file to load -- rspec/core/rake_task
```

Now you can fix it by making these Rake tasks conditional on the gem load. For example:

```
begin
 require "rspec/core/rake_task"

 desc "Run all examples"
 RSpec::Core::RakeTask.new(:spec) do |t|
 t.rspec_opts = %w[--color]
 t.pattern = 'spec/*_spec.rb'
 end
rescue LoadError
end
```

Confirm it works locally, then push to Heroku.

## Rack::Sendfile

Heroku does not support the use of Rack::Sendfile.

Rack::Sendfile usually requires that there is a frontend webserver like nginx or apache is running on the same machine as the application server.

This is not how Heroku is architected. Using the Rack::Sendfile middleware will cause your file downloads to fail since it will send a body with Content-Length of 0.

**Número máximo de aplicaciones** If you have unverified account then the maximum number of apps is 5 only and for verified account number is 100.

## 52.4. Configuration

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku help config
Usage: heroku config
```

display the config vars for an app

```
-s, --shell # output config vars in shell format
```

Examples:

```
$ heroku config
A: one
B: two
```

```
$ heroku config --shell
A=one
B=two
```

Additional commands, type "heroku help COMMAND" for more details:

```
config:get KEY # display a config value for an app
config:set KEY1=VALUE1 [KEY2=VALUE2 ...] # set one or more config vars
config:unset KEY1 [KEY2 ...] # unset one or more config vars
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config -s
DATABASE_URL=postgres://bhhatrhhjhwcvt:hjgjfhgjfhfuWH7ls_PJKK5QD@ec2-54-204-35-132.compute-1.
HEROKU_POSTGRESQL_BLACK_URL=postgres://bhjshfdhakwcvt:hQssnhq1y1jhgfhgls_PGNu5QD@ec2-54-204-35
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:set C=4
Setting config vars and restarting crguezl-songs... done, v6
```

C: 4

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:get C
4
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:unset C
Unsetting C and restarting crguezl-songs... done, v7
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:get C
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$]
```

## 52.5. Make Heroku run non-master Git branch

Make Heroku run non-master Git branch You can push an alternative branch to Heroku using Git.

```
git push heroku-dev test:master
```

This pushes your local test branch to the remote's master branch (on Heroku).

El manual de git push dice:

To push a local branch to an established remote, you need to issue the command:

```
git push <REMOTENAME> <BRANCHNAME>
```

This is most typically invoked as `git push origin master`.

If you would like to give the branch a different name on the upstream side of the push, you can issue the command:

```
git push <REMOTENAME> <LOCALBRANCHNAME>:<REMOTEBRANCHNAME>
```

## 52.6. Account Verification and add-ons

You must verify your account by adding a credit card before you can add any add-on to your app other than `heroku-postgresql:dev` and `pgbackups:plus`.

Adding a credit card to your account lets you

1. use the free add-ons,
2. allows your account to have more than 5 apps at a time (verified accounts may have up to 100 apps),
3. and gives you access to turn on paid services any time with a few easy clicks.
4. The easiest way to do this is to go to your account page and click **Add Credit Card**.
5. Alternatively, when you attempt to perform an action that requires a credit card, either from the Heroku CLI or through the web interface, you will be prompted to visit the credit card page.

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku addons:add rediscloud:20
Adding rediscloud:20 on dgjgxcl-songs... failed
! Please verify your account to install this add-on
! For more information, see http://devcenter.heroku.com/categories/billing
! Verify now at https://heroku.com/verify
```

## 52.7. Véase

- Heroku: Getting Started with Ruby on Heroku
- SitePoint: Get Started with Sinatra on Heroku by Jagadish Thaker. Published August 12, 2013
- Deploying Rack-based Apps
- Heroku: List of Published Articles for Ruby
- Foreman
  - 1. Introducing Foreman by David Dollar
  - 2. Foreman man pages
  - 3. Applying the Unix Process Model to Web Apps by Adam Wiggins
- Ruby Kickstart - Session 6 de Joshua Cheek (Vimeo)
- sinatra-rock-paper-scissors en GitHub. Ejemplo listo para desplegar en Heroku (2013)
- Ejemplo en GitHub usando Sinatra listo para desplegar en Heroku (2014)
- The Procfile is your friend 13 January, 2012. Neil Middleton

## 52.8. Práctica: Despliegue en Heroku

Despliegue en Heroku la práctica [Aplicación Web con Sinatra: Contar la Popularidad de Nuestros Amigos en Twitter](#) escrita en la sección 33.27.

Añada pruebas. Esta práctica se hace en parejas. Use los issues de GitHub para documentar el histórico de desarrollo.

## Donde encontrar Ejemplos

### ■ number\_cruncher

- Artículo original de Daz en SitePoint: Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis por Darren Jones
- [~/sinatra/sinatra-number\_cruncher(master)]\$ git remote -v  
daz git@github.com:daz4126/number\_cruncher.git (fetch)  
daz git@github.com:daz4126/number\_cruncher.git (push)  
heroku git@heroku.com:primefactorization.git (fetch)  
heroku git@heroku.com:primefactorization.git (push)  
origin git@github.com:crguezl/number\_cruncher.git (fetch)  
origin git@github.com:crguezl/number\_cruncher.git (push)
- [~/sinatra/sinatra-number\_cruncher(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-number\_cruncher
- GitHub repo forked from Daz **number\_cruncher**

### ■ sinatra-rock-paper-scissors

- sinatra-rock-paper-scissors en GitHub
- [~/sinatra/sinatra-rock-paper-scissors/sinatra-rockpaperscissors(master)]\$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-rock-paper-scissors/sinatra-rockpapersc
- [~/sinatra/sinatra-rock-paper-scissors/sinatra-rockpaperscissors(master)]\$ git remote -v  
heroku git@heroku.com:rps-crguezl.git (fetch)  
heroku git@heroku.com:rps-crguezl.git (push)  
origin git@github.com:crguezl/sinatra-rock-paper-scissors.git (fetch)  
origin git@github.com:crguezl/sinatra-rock-paper-scissors.git (push)
- Despliegue en heroku

### ■ loadfileontotexarea

- Ejemplo en GitHub
- [~/javascript/jquery/loadfileontotexarea(master)]\$ git remote -v  
heroku git@heroku.com:pllexer.git (fetch)  
heroku git@heroku.com:pllexer.git (push)  
origin git@github.com:crguezl/loadfileontotexarea.git (fetch)  
origin git@github.com:crguezl/loadfileontotexarea.git (push)
- [~/javascript/jquery/loadfileontotexarea(master)]\$ pwd -P  
/Users/casiano/local/src/javascript/jquery/loadfileontotexarea
- Despliegue en Heroku

## Capítulo 53

# Envío de SMSs y Mensajes: Twilio y Clockworks

1. How to create a Twilio app on Heroku de Morten Baga
2. Twilio HackPack for Heroku and Sinatra
3. heroku-twilio A fork of the Twilio node library that is Heroku friendly
4. Twilio HackPack for Sinatra and Heroku Posted by Oscar Sanchez on July 05, 2012 in Tips, Tricks and Sample Code
  1. Clockworks SMS
  2. Documentación de Clockworks SMS
  3. Ruby gem para Clockworks. En Github

```
require 'clockwork'

api = Clockwork::API.new('API_KEY_Goes_Here')
message = api.messages.build(:to => '441234123456', :content => 'This is a test message.')
response = message.deliver

if response.success
 puts response.message_id
else
 puts response.error_code
 puts response.error_description
end
```

# Capítulo 54

## Rest

### 54.1. Introducción

**The REST architectural style** *Representational state transfer (REST)* is an abstraction of the architecture of the World Wide Web;

*REST* is an architectural style consisting of a coordinated set of architectural constraints applied to

- components,
- connectors, and
- data elements,

within a distributed hypermedia system.

The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine.

The REST architectural style is often applied to the development of *web services* as an alternative to other distributed-computing specifications such as *SOAP* (*Simple Object Access protocol*).

### Web Services

A Web Service is a method of communication between two electronic devices over a network.

It is a software function provided at a network address over the web with the service always on as in the concept of *utility computing*<sup>1</sup>.

The W3C defines a Web service generally as:

A software system designed to support interoperable machine-to-machine interaction over a network.

**RESTful Web Services** One can characterize web services as *RESTful* if they conform to the constraints described in the architectural constraints.

*Web service APIs* that adhere to the architectural constraints are called *RESTful*.

HTTP based RESTful APIs are defined with these aspects:

1. [base URI](#), such as <http://example.com/resources/>

---

<sup>1</sup>Utility computing is a service provisioning model in which a service provider makes computing resources and infrastructure management available to the customer as needed, and charges them for specific usage rather than a flat rate

2. an Internet media type for the data. This is often JSON<sup>2</sup> but can be any other valid Internet media type (e.g. XML, Atom<sup>3</sup>, microformats<sup>4</sup>, etc.)
3. standard HTTP methods (e.g., GET, PUT, POST, or DELETE)
4. hypertext links to reference state
5. hypertext links to reference related resources

The following table shows the HTTP methods that are typically used to implement a RESTful API.

Resource	GET	PUT	POST
Collection URI, such as <code>http://example.com/resources</code>	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new collection. This is assigned a new URI.
Element URI, such as <code>http://example.com/resources/1</code>	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it doesn't exist, create it.	Not generally addressed in its own right; instead, it is usually created as part of a new collection.

## Safe and Idempotent Methods

- The PUT and DELETE methods are referred to as *idempotent*, meaning that the operation will produce the same result no matter how many times it is repeated.
- The GET method is a *safe method* (or *nullipotent*), meaning that calling it produces no side-effects. In other words, retrieving or accessing a record doesn't change it.
- The POST method is not necessarily idempotent, and therefore sending an identical POST request multiple times may further affect state or cause further side effects such as when a user does not realize that their action will result in sending another request, or they did not receive adequate feedback that their first request was successful.

While web browsers may show alert dialog boxes to warn users in some cases where reloading a page may re-submit a POST request, it is generally up to the web application to handle cases where a POST request should not be submitted more than once.

- Note that whether a method is idempotent is not enforced by the protocol or web server. It is perfectly possible to write a web application in which (for example) a database insert or other non-idempotent action is triggered by a GET or other request. Ignoring this recommendation, however, may result in undesirable consequences, if a user agent assumes that repeating the same request is safe when it isn't.
- Safe and Idempotent Methods part of Hypertext Transfer Protocol – HTTP/1.1 RFC 2616 Fielding, et al.

Unlike SOAP-based web services, there is no "official" standard for RESTful web APIs. This is because REST is an architectural style, unlike SOAP, which is a protocol.

---

<sup>2</sup>JavaScript Object Notation, is an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML

<sup>3</sup>The Atom Syndication Format is an XML language used for web feeds. A web feed is a data format used for providing users with frequently updated content

<sup>4</sup>XHTML and HTML standards allow for the embedding and encoding of semantics within the attributes of markup tags. Microformats take advantage of these standards by indicating the presence of metadata using some attributes

## 54.2. Ejemplo de un servicio RESTfull en Sinatra

### Donde

1. Uno! Use Sinatra to Implement a REST API. by Dan Schaefer. Published March 10, 2014
2. Código en GitHub <https://github.com/crguezl/sinatra-rest-api-example>
3. `[~/sinatra/sinatra-rest-api-example(master)]$ pwd -P  
/Users/casiano/local/src/ruby/sinatra/sinatra-rest-api-example`

### Ejecución del servidor

```
[~/sinatra/sinatra-rest-api-example(master)]$ ruby uno-server.rb
[2014-09-13 15:41:24] INFO WEBrick 1.3.1
[2014-09-13 15:41:24] INFO ruby 2.1.2 (2014-05-08) [x86_64-darwin13.0]
== Sinatra/1.4.5 has taken the stage on 4567 for production with backup from WEBrick
[2014-09-13 15:41:24] INFO WEBrick::HTTPServer#start: pid=94715 port=4567
```

### Ejecución del cliente

```
[~/sinatra/sinatra-rest-api-example(master)]$ pry
[1] pry(main)> require './uno-client'
=> true
[2] pry(main)> bob = UnoClient.new 'bob'
=> #<UnoClient:0x007fbde4459468 @name="bob">
[3] pry(main)> carol = UnoClient.new 'carol'
=> #<UnoClient:0x007fbde43f1de0 @name="carol">
[4] pry(main)> ted = UnoClient.new 'ted'
=> #<UnoClient:0x007fbde438aa50 @name="ted">
[5] pry(main)> alice = UnoClient.new 'alice'
=> #<UnoClient:0x007fbde4323b48 @name="alice">
[6] pry(main)> ralph = UnoClient.new 'ralph'
=> #<UnoClient:0x007fbde42c0250 @name="ralph">
[7] pry(main)> bob.join_game
{:status=>"welcome"}
=> nil
[8] pry(main)> carol.join_game
{:status=>"welcome"}
=> nil
[9] pry(main)> ted.join_game
{:status=>"welcome"}
=> nil
[10] pry(main)> alice.join_game
{:status=>"welcome"}
=> nil
[11] pry(main)> ralph.join_game
{:status=>"sorry - game not accepting new players"}
=> nil
[12] pry(main)> bob.deal
{"status":"success"}
=> nil
[13] pry(main)> bob.get_cards
{"cards": ["6-heart", "jack-diamond", "7-spade", "queen-club", "jack-spade"]}
=> nil
```

```
[14] pry(main)> carol.get_cards
>{"cards":["9-heart","4-spade","10-spade","9-spade","10-club"]}
=> nil
[15] pry(main)> ted.get_cards
>{"cards":["6-spade","ace-diamond","ace-club","3-heart","2-heart"]}
=> nil
[16] pry(main)> alice.get_cards
>{"cards":["9-diamond","8-spade","8-club","8-heart","jack-club"]}
=> nil
[17] pry(main)>
```

## Capítulo 55

# Sinatra + Sprockets + Slim + Sinatra-reloader Example

- <https://github.com/crguezl/sinatra-sprockets-slim-backbone-example>

# Capítulo 56

## Sinatra::Flash

Sinatra::Flash is an extension that lets you store information between requests.

Often, when an application processes a request, it will redirect to another URL upon finishing, which generates another request.

This means that any information from the previous request is lost (due to the stateless nature of HTTP).

Sinatra::Flash overcomes this by providing access to the `flash`—a hash-like object that stores temporary values such as error messages so that they can be retrieved later—usually on the next request.

It also removes the information once it's been used.

All this can be achieved via sessions (and that's exactly how Sinatra::Flash does it), but Sinatra::Flash is easy to implement and provides a number of helper methods.

### Ejemplo

```
[~/sinatra/sinatra-flash]$ cat app.rb
require 'sinatra'
require 'sinatra/flash'

enable :sessions

get '/blah' do
 # This message won't be seen until the NEXT Web request that accesses the flash collection
 flash[:blah] = "You were feeling blah at #{Time.now}."

 # Accessing the flash displays messages set from the LAST request
 "Feeling blah again? That's too bad. #{flash[:blah]}"
end

get '/pum' do
 # This message won't be seen until the NEXT Web request that accesses the flash collection
 flash[:pum] = "You were feeling pum at #{Time.now}."

 # Accessing the flash displays messages set from the LAST request
 "Feeling pum again? That's too bad. #{flash[:pum]}"
end
```

### Gemfile

```
[~/sinatra/sinatra-flash]$ cat Gemfile
source 'https://rubygems.org'

gem 'sinatra'
```

```
gem 'sinatra-flash'
```

## **Capítulo 57**

# **Alternativas a Sinatra**

### **57.1. Cuba**

Cuba es una alternativa a Sinatra.

1. cuba

### **57.2. Grape**

Grape es una alternativa a Sinatra.

1. Grape

### **57.3. Ramaze**

Ramaze es una alternativa a Rails

1. Ramaze

# Capítulo 58

## Padrino

### 58.1. Introducción

1. Darío Javier Cravero says 'The best way to get started is to have a look at the guides'
2. Padrino Blog Tutorial
3. Gist Template para seguir el tutorial del blog en Padrino: [achiu / blog-template.rb](#)

```
[~/srcpadrino/padrino-book(master)]$ padrino --version
Padrino v. 0.11.4
```

```
[~/srcpadrino/padrino-book(master)]$ padrino --help
Tasks:
```

```
padrino console # Boots up the Padrino application irb console (alternatively use 'c').
padrino generate # Executes the Padrino generator with given options (alternatively use 'gen' or 'g').
padrino help # Describe available tasks or one specific task
padrino rake # Execute rake tasks.
padrino runner # Run a piece of code in the Padrino application environment (alternatively use 'run').
padrino start # Starts the Padrino application (alternatively use 's').
padrino stop # Stops the Padrino application (alternatively use 'st').
padrino version # Show current Padrino version.
```

Options:

```
-c, [--chdir=CHDIR] # Change to dir before starting.
-e, --environment=ENVIRONMENT # Padrino Environment.
 # Default: development
[--help] # Show help usage
```

```
[~/srcpadrino/padrino-book(master)]$ padrino help generate
```

Usage:

```
padrino generate
```

Options:

```
-c, [--chdir=CHDIR] # Change to dir before starting.
-e, --environment=ENVIRONMENT # Padrino Environment.
 # Default: development
[--help] # Show help usage
```

Executes the Padrino generator with given options (alternatively use 'gen' or 'g').

```
gem uninstall activesupport -v 4.0.0
```

To create a Padrino application, the best place to start is using the convenient Padrino generator. Similar to Rails, Padrino has a project generator which will create a skeleton application with all the files you need to begin development of your new idea. Padrino is an agnostic framework and supports using a variety of different template, testing, JavaScript and database components. You can learn more by reading the generators guide.

For this sample application, we will use the ActiveRecord ORM, the Haml templating language, the Shoulda testing framework and the jQuery JavaScript library. With that in mind, let us generate our new project:

```
[~/srcpadrino]$ padrino g project sample_blog -t shoulda -e haml -c sass -s jquery -d activerecord
 create
 create .gitignore
 create config.ru
 create config/apps.rb
 create config/boot.rb
 create public/favicon.ico
 create public/images
 create public/javascripts
 create public/stylesheets
 create tmp
 create .components
 create app
 create app/app.rb
 create app/controllers
 create app/helpers
 create app/views
 create app/views/layouts
 create Gemfile
 create Rakefile
 applying activerecord (orm)...
 apply orms/activerecord
 insert Gemfile
 insert Gemfile
 insert app/app.rb
 create config/database.rb
 applying shoulda (test)...
 apply tests/shoulda
 insert Gemfile
 insert Gemfile
 create test/test_config.rb
 create test/test.rake
 skipping mock component...
 applying jquery (script)...
 apply scripts/jquery
 create public/javascripts/jquery.js
 create public/javascripts/jquery-ujs.js
 create public/javascripts/application.js
 applying haml (renderer)...
 apply renderers/haml
 insert Gemfile
 applying sass (stylesheet)...
 apply stylesheets/sass
 insert Gemfile
 insert app/app.rb
```

```
create lib/sass_initializer.rb
create app/stylesheets
 identical .components
 force .components
 force .components
Bundling application dependencies using bundler...
 run bundle install from "."
Fetching gem metadata from https://rubygems.org/.....
Fetching gem metadata from https://rubygems.org/...
Resolving dependencies...
Using rake (10.1.1)
Using i18n (0.6.9)
Using multi_json (1.8.2)
Using activesupport (3.2.16)
Using builder (3.0.4)
Installing activemodel (3.2.16)
Installing arel (3.0.3)
Using tzinfo (0.3.38)
Installing activerecord (3.2.16)
Using bundler (1.3.5)
Using tilt (1.4.1)
Using haml (4.0.4)
Using rack (1.5.2)
Using url_mount (0.2.1)
Using http_router (0.11.0)
Using mime-types (1.25.1)
Using polyglot (0.3.3)
Using treetop (1.4.15)
Using mail (2.5.4)
Using rack-protection (1.5.1)
Using sinatra (1.4.4)
Using thor (0.17.0)
Using padrino-core (0.11.4)
Using padrino-helpers (0.11.4)
Using padrino-admin (0.11.4)
Using padrino-cache (0.11.4)
Using padrino-gen (0.11.4)
Using padrino-mailer (0.11.4)
Using padrino (0.11.4)
Using rack-test (0.6.2)
Using sass (3.2.13)
Installing shoulda-context (1.1.6)
Installing shoulda-matchers (2.4.0)
Installing shoulda (3.5.0)
Using sqlite3 (1.3.8)
Your bundle is complete!
Use 'bundle show [gemname]' to see where a bundled gem is installed.
```

```
=====
sample_blog is ready for development!
=====
```

```
$ cd ./sample_blog
=====
```

This command will generate our basic Padrino project and print out a nice report of the files generated.

Notice the `-b` flag in the previous command which automatically instructs bundler to install all dependencies. All we need to do now is `cd` into our brand new application:

```
[~/srcpadrino]$ cd sample_blog/
[~/srcpadrino/sample_blog]$ tree
.
|--- Gemfile
|--- Gemfile.lock
|--- Rakefile
|--- app
| |--- app.rb
| |--- controllers
| |--- helpers
| |--- stylesheets
| '--- views
| '--- layouts
|--- config
| |--- apps.rb
| |--- boot.rb
| '--- database.rb
|--- config.ru
|--- lib
| '--- sass_initializer.rb
|--- public
| |--- favicon.ico
| |--- images
| |--- javascripts
| | |--- application.js
| | |--- jquery-ujs.js
| | '--- jquery.js
| '--- stylesheets
|--- test
| |--- test.rake
| '--- test_config.rb
'--- tmp
```

14 directories, 15 files

Padrino generators do not lock you into using any particular database, ORM, testing framework, templating engine or javascript library. In fact, when generating an application you can actually tell Padrino which components you would like to use!

```
[~/srcpadrino/myproject]$ bundle install
Fetching gem metadata from https://rubygems.org/..... .
Fetching gem metadata from https://rubygems.org/..
Resolving dependencies...
Installing rake (10.1.1)
Installing i18n (0.6.9)
Using multi_json (1.8.2)
```

```

Installing activesupport (3.2.16)
Using addressable (2.3.5)
Using bcrypt-ruby (3.1.2)
Using bundler (1.3.5)
Using data_objects (0.10.13)
Using dm-core (1.2.1)
Using dm-aggregates (1.2.0)
Using dm-constraints (1.2.0)
Using dm-do-adapter (1.2.0)
Using dm-migrations (1.2.0)
Using do_sqlite3 (0.10.13)
Using dm-sqlite-adapter (1.2.0)
Using dm-timestamps (1.2.0)
Using fastercsv (1.5.5)
Using json (1.8.1)
Using stringex (1.5.1)
Using uuidtools (2.1.4)
Using dm-types (1.2.2)
Using dm-validations (1.2.0)
Using tilt (1.4.1)
Using haml (4.0.4)
Using rack (1.5.2)
Using url_mount (0.2.1)
Using http_router (0.11.0)
Installing mime-types (1.25.1)
Using polyglot (0.3.3)
Using treetop (1.4.15)
Using mail (2.5.4)
Installing metaklass (0.0.1)
Installing mocha (0.14.0)
Using rack-protection (1.5.1)
Using sinatra (1.4.4)
Using thor (0.17.0)
Using padrino-core (0.11.4)
Using padrino-helpers (0.11.4)
Using padrino-admin (0.11.4)
Using padrino-cache (0.11.4)
Using padrino-gen (0.11.4)
Using padrino-mailer (0.11.4)
Using padrino (0.11.4)
Using rack-test (0.6.2)
Installing rr (1.1.2)
Installing riot (0.12.7)
Your bundle is complete!
Use 'bundle show [gemname]' to see where a bundled gem is installed.
[~/srcpadrino/myproject]$
```

Now, the terminal should be inside the root of our newly generated application with all necessary gem dependencies installed. Let us take a closer look at the particularly important generated files before we continue on with development.

1. **Gemfile** – Be sure to include any necessary gem dependencies for your app in this file!
2. **app/app.rb** – This is the primary configuration file for your core application.

3. `config/apps.rb` – This defines which applications are mounted in your project.
4. `config/database.rb` – This defines the connection details for your chosen database adapter.

The following important directories are also generated:

1. `app/controllers` – This is where the Padrino route definitions should be defined.
2. `app/helpers` – This is where helper methods should be defined for your application.
3. `app/views` – This should contain your template views to be rendered in a controller.
4. `lib` – This should contain any extensions, libraries or other code to be used in your project.
5. `public` – This is where images, style sheets and JavaScript files should be stored.
6. `test` – This is where your model and controller tests should be stored.

Now, let us examine the `config/database.rb` file to make sure the database connection settings are correct. For now, the defaults are OK for this tutorial. A sqlite3 database will be used that is stored inside `db/sample_blog_development.db`.

Let us also setup a few simple routes in our application to demonstrate the Padrino routing system. Let's go into the `app/app.rb` file and enter the following routes:

```
<= Padrino leaves the gun, takes the cannoli
[~/srcpadrino/sample_blog]$ cat Gemfile
source 'https://rubygems.org'

Distribute your app as a gem
gemspec

Server requirements
gem 'thin' # or mongrel
gem 'trinidad', :platform => 'jruby'

Optional JSON codec (faster performance)
gem 'oj'

Project requirements
gem 'rake'

Component requirements
gem 'bcrypt-ruby', :require => 'bcrypt'
gem 'sass'
gem 'haml'
gem 'activerecord', '>= 3.1', :require => 'active_record'
gem 'sqlite3'

Test requirements
gem 'shoulda', :group => 'test'
gem 'rack-test', :require => 'rack/test', :group => 'test'

Padrino Stable Gem
gem 'padrino', '0.11.4'

Or Padrino Edge
```

```

gem 'padrino', :github => 'padrino/padrino-framework'

Or Individual Gems
%w(core gen helpers cache mailer admin).each do |g|
gem 'padrino-' + g, '0.11.4'
end

[~/srcpadrino]$ padrino help start
Usage:
 padrino start

Options:
 -a, [--server=SERVER] # Rack Handler (default: autodetect)
 -h, --host=HOST # Bind to HOST address.
 # Default: 127.0.0.1
 -p, --port=N # Use PORT.
 # Default: 3000
 -d, [--daemonize] # Run daemonized in the background.
 -i, [--pid=PID] # File to store pid.
 [-d, --debug] # Set debugging flags.
 -c, [--chdir=CHDIR] # Change to dir before starting.
 -e, --environment=ENVIRONMENT # Padrino Environment.
 # Default: development
 [--help] # Show help usage

```

Starts the Padrino application (alternatively use 's').

```

[~/srcpadrino/sample_blog]$ bundle exec padrino rake -T
=> Executing Rake -T ...
rake ar:abort_if_pending_migrations # Raises an error if there are pending migrations
rake ar:charset # Retrieves the charset for the current environment's database
rake ar:collation # Retrieves the collation for the current environment's database
rake ar:create # Create the database defined in config/database.yml for the current environment
rake ar:create:all # Create all the local databases defined in config/database.yml
rake ar:drop # Drops the database for the current Padrino.env
rake ar:drop:all # Drops all the local databases defined in config/database.yml
rake ar:forward # Pushes the schema to the next version
rake ar:migrate # Migrates the database through scripts in db/migrate and db/undo_migrate
rake ar:migrate:down # Runs the "down" for a given migration VERSION
rake ar:migrate:redo # Rollbacks the database one migration and re-migrates up
rake ar:migrate:reset # Resets your database using your migrations for the current environment
rake ar:migrate:up # Runs the "up" for a given migration VERSION
rake ar:reset # Drops and recreates the database from db/schema.rb for the current environment
rake ar:rollback # Rolls the schema back to the previous version
rake ar:schema:dump # Create a db/schema.rb file that can be portably used against any database
rake ar:schema:load # Load a schema.rb file into the database
rake ar:setup # Create the database, load the schema, and initialize with fixtures
rake ar:structure:dump # Dump the database structure to a SQL file
rake ar:translate # Generates .yml files for I18n translations
rake ar:version # Retrieves the current schema version number
rake db:seed # Load the seed data from db/seeds.rb
rake routes[query] # Displays a listing of the named routes within a project
rake routes:app[app] # Displays a listing of the named routes for a given app [app]

```

```
rake secret # Generate a secret key
rake test # Run application test suite
rake test:models # Run tests for test:models
```

## 58.2. Generadores

The usage for the project generator is quite simple:

```
$ padrino g project <the_app_name> </path/to/create/app> --<component-name> <value>
```

The simplest possible command to generate a base application would be:

```
$ padrino g project demo_project
```

Esto crea un config.ru como este:

```
1 #!/usr/bin/env rackup
2
3 require File.expand_path("../config/boot.rb", __FILE__)
4
5 run Padrino.application
```

El fichero boot.rb contiene:

```
1 PADRINO_ENV = ENV['PADRINO_ENV'] ||= ENV['RACK_ENV'] ||= 'development' unless defined?(PADRINO_ENV)
2 PADRINO_ROOT = File.expand_path('../..', __FILE__) unless defined?(PADRINO_ROOT)
3
4 require 'rubygems' unless defined?(Gem)
5 require 'bundler/setup'
6 Bundler.require(:default, PADRINO_ENV)
7
8 Padrino.before_load do
9 end
10
11 Padrino.after_load do
12 end
13
14 Padrino.load!
```

La que parece ser la aplicación contiene este código:

```
[~/srcpadrino/demo_project]$ cat app/app.rb
module DemoProject
 class App < Padrino::Application
 register Padrino::Rendering
 register Padrino::Mailer
 register Padrino::Helpers

 enable :sessions

 end
end
```

You can also define specific components to be used:

```
$ padrino g project demo_project -t rspec -e haml -m rr -s jquery -d datamapper -c sass
```

You can also instruct the generator to skip a certain component to avoid using one at all (or to use your own):

```
$ padrino g project demo_project --test none --renderer none
```

You can also specify an alternate name for your core application using the `--app` option:

```
$ padrino g project demo_project --app alternate_app_name # alias -n
```

The generator uses the `bundler` gem to resolve any application dependencies when the application is newly created. The necessary `bundler` command can be executed automatically through the generator with:

```
$ padrino g project demo_project --run_bundler # alias -b
```

or this can be done manually through executing command `bundle install` in the terminal at the root of the generated application.

For more examples of using the project generator in common cases, check out the Basic Projects guide.

The generator framework within Padrino is extensible and additional components and tools can be added easily.

This would be achieved through forking our project and reading through the code in `lib/generators/project`. and the setup instructions inside the relevant files within `lib/generators/components/`.

## configuration options

The project generator has several available configuration options:

Options	Default	Aliases	Description
<code>bundle</code>	false	<code>-b</code>	execute bundlerer dependencies installation
<code>root</code>	.	<code>-r</code>	the root destination path forward the project
<code>dev</code>	false	<code>none</code>	use edge version from local git checkout
<code>app</code>	nil	<code>-n</code>	specify app name different from the project name
<code>tiny</code>	false	<code>-i</code>	generate tiny project skeleton
<code>adapter</code>	<code>sqlite</code>	<code>-a</code>	specify orrm db adapter (mysql, sqlite, postgres)

The available components and their default options are listed below:

Component	Default	Aliases	Options
<code>orm</code>	<code>none</code>	<code>-d</code>	mongoid, activerecord,below minirecord, datamapper, couchrest, monogramatic, mongomapper, ohm, ripple, sequel
<code>test</code>	<code>none</code>	<code>-t</code>	bacon, shoulda, cucumber, testspec, riot, rspec, minitest
<code>script</code>	<code>none</code>	<code>-s</code>	prototype, rightjs, jquery, mootools, extcore, dojo
<code>renderer</code>	<code>slim</code>	<code>-e</code>	erb, erubis, haml, slim, liquid
<code>stylesheet</code>	<code>none</code>	<code>-c</code>	sass, less, scss, compass
<code>mock</code>	<code>none</code>	<code>-m</code>	rr, mocha

Note: Be careful with your naming when using generators and do not have your project name, or any models or controllers overlap. Avoid naming your app `Posts` and then your controller or subapp with the same name.

## Some examples

Generate a project with a different application name from the project path

```
padrino g my_project -n blog
```

this will generate the project at path `my_project/` but the applications name will be `Blog`.

```
1 module MyProject
2 class Blog < Padrino::Application
3 register Padrino::Rendering
4 register Padrino::Mailer
5 register Padrino::Helpers
6
7 enable :sessions
8
9 end
10 end
```

Generate a project with mongoid and run bundler after

```
padrino g project your_project -d mongoid -b
```

Generate a project with riot test and rr mocking

```
padrino g project your_project -t riot -m rr
```

Generate a project with sequel with mysql

```
padrino g project your_project -d sequel -a mysql
```

Generate a tiny project skeleton

```
padrino g project your_project --tiny
```

Choose a root for your project

```
padrino g project your_project -r /usr/local/padrino
```

This will create a new padrino project in `/usr/local/padrino/your_project/`

Use Padrino from a git cloned repository

```
padrino g project your_project [--dev] # Use padrino from a git checkout
```

### 58.3. Enlaces

- A Simple Admin for Padrino. Dan Schaefer. Published February 28, 2014

## Capítulo 59

# Desarrolladores de Sinatra

### 59.1. Konstantin Haase

- Real Time Rack por Konstantin Haase en <http://confreaks.com/>
- El blog de Konstantin Haase desarrollador al cargo de Sinatra
- Konstantin Haase en GitHub
- Jose Mota

## **Parte IV**

# **JAVASCRIPT, HTML y CSS**

# Capítulo 60

## Introducción

```
[~/javascript/node.js/nathan closures(master)]$ node readsync.js
Contents: var fs = require('fs');
var readFile = function () {
 var data;
 data = fs.readFileSync('readsync.js');
 console.log('Contents: ' + data);
};

readFile();
```

### 60.1. Challenge Problem

JavaScript Review

#### 60.1.1. Iterative

```
var count = function(tree) {
 stack = [tree];
 c = 0;
 while (stack.length > 0) {
 node = stack.pop();
 if (node !== null) {
 c += 1;
 stack.push(node.left);
 stack.push(node.right);
 }
 }
 return c;
};
```

# Capítulo 61

## Objetos

### 61.1. Tutoriales de OOP en JavaScript en la Web

1. The Basics of Object-Oriented JavaScript Leigh Kaszick on Nov 11th 2009 en NetTuts+
2. Understanding JavaScript OOP por Quildreen Motta.
3. Object.defineProperty(obj, prop, descriptor)
4. What is the 'new' keyword in JavaScript?
5. Constructors considered mildly confusing
6. Google I/O 2011: Learning to Love JavaScript por Alex Russell, Mayo 2011. ¡Excelente!

### 61.2. Ejercicios

1. ¿Cuál es la salida?

```
> z
{ x: 3, y: 1 }
> Object.keys(z)
?????
> Object.keys(z).forEach(function(i) { console.log(i+" -> "+z[i]); })
?????
```

2. ¿Qué queda finalmente en z?

```
> z
{ x: 2, y: 1 }
> Object.defineProperty(z, 'y', {writable : false})
{ x: 2, y: 1 }
> z.x = 3
?????
> z.y = 2
?????
> z
```

3. ¿Cuáles son las salidas?

```

> obj = { x : 1, y : 2}
{ x: 1, y: 2 }
> bValue = 5
5
> Object.defineProperty(obj, "b", {get: function(){ return bValue; },
 set: function(y){ bValue = y; }})
{ x: 1, y: 2 }
> obj.b = "hello"

> bValue

> obj.b

> bValue = "world"

> obj.b

>

```

#### 4. ¿Cuales son las salidas?

```

> var o = {};
undefined
> Object.defineProperty(o, "a", { value : 1, enumerable:true });
{ a: 1 }
> Object.defineProperty(o, "b", { value : 2, enumerable:false });
{ a: 1 }
> Object.defineProperty(o, "c", { value : 3 }); // enumerable defaults to false
{ a: 1 }
> o.d = 4; // enumerable defaults to true when creating a property by setting it
4
>
undefined
> for (var i in o) {
... console.log(i);
...
???????
> Object.keys(o);
???????
> o.propertyIsEnumerable('a');
?????
> o.propertyIsEnumerable('b');
?????
> o.propertyIsEnumerable('c');
?????
> o.b
?????
> o["b"]
?????

```

#### 5. > function foo(a, b){return a \* b;}

```

undefined
> f = function foo(a, b){return a * b;}

```

```
[Function: foo]
> foo.length
2
> foo.constructor
[Function: Function]
> foo.prototype
{}
> typeof foo.prototype
'object'
> [1, 2].constructor
[Function: Array]
```

6. ¿Cuáles son las salidas?

```
> Object.getPrototypeOf({ a: 1})

> Object.getPrototypeOf([1,2,3])

> Object.getPrototypeOf([1,2,3]) == Array.prototype

> Object.getPrototypeOf({ a:1 }) === Object.prototype

> Object.getPrototypeOf(Array.prototype)

> Object.getPrototypeOf(Object.prototype)

> Object.getPrototypeOf(function() {})

> Object.getPrototypeOf(Object.getPrototypeOf(function() {}))

> [1,2,3].__proto__

> [1,2,3].__proto__ == Array.prototype
```

Figura 61.1: Jerarquía de Prototipos Nativos

```
var b = new Foo(20);
var c = new Foo(30);
```

Figura 61.2: \_\_proto\_\_ and prototypes

### 61.3. Comprobando Propiedades

```
> o = { x: 1}
{ x: 1 }
```

```
> "x" in o
true
> "y" in o
false
> "toString" in o
true
> o.hasOwnProperty('x')
true
> o.hasOwnProperty('toString')
false
```

## 61.4. Enumeración de Propiedades

```
> o = { x: 1 }
{ x: 1 }
> b = Object.create(o)
{}
> b.y = 2
2
> b.propertyIsEnumerable('x')
false
> b.propertyIsEnumerable('y')
true
> Object.prototype.propertyIsEnumerable('toString')
false
> a = {x:1, y:2}
{ x: 1, y: 2 }
> b = Object.create(a)
{}
> b.z = 3
3
> for(i in b) console.log(b[i])
3
1
2
undefined
> for(i in b) console.log(i)
z
x
y
undefined
> b.propertyIsEnumerable("toString")
false
```

# Capítulo 62

## Funciones

### 62.0.1. Los Métodos call y apply

Los métodos `call` y `apply` nos permiten invocar una función como si fuera un método de algún otro objeto.

```
[~/Dropbox/src/javascript/learning]$ cat call.js
var Bob = {
 name: "Bob",
 greet: function() {
 console.log("Hi, I'm " + this.name);
 }
}

var Alice = {
 name: "Alice",
};

Bob.greet.call(Alice);
```

```
[~/Dropbox/src/javascript/learning]$ node call.js
Hi, I'm Alice
```

#### 1. Function.apply and Function.call in JavaScript

```
> function f() { console.log(this.x); }
undefined
> f.toString()
'function f() { console.log(this.x); }'
> z = { x : 99 }
{ x: 99 }
> f.call(z)
99
undefined
>
```

```
2. > o = { x : 15 }
{ x: 15 }
> function f(m) { console.log(m+" "+this.x); }
undefined
> f("invoking f")
invoking f 10
```

```
undefined
> f.call(o, "invoking f via call");
invoking f via call 15
undefined
```

## 62.1. Programación Funcional

# Capítulo 63

## Clases y Módulos

Véase el libro *Learning JavaScript Design Pattern* [?]

### 63.1. Herencia

```
[~/Dropbox/src/javascript/inheritance]$ cat inh1.js
//Shape - superclass
function Shape() {
 this.x = 0;
 this.y = 0;
}

Shape.prototype.toString = function() {
 return "("+this.x+", "+this.y+")";
}

Shape.prototype.move = function(x, y) {
 this.x += x;
 this.y += y;
 console.info("Shape moved to "+this);
};

// Rectangle - subclass
function Rectangle() {
 Shape.call(this); //call super constructor.
}

// Rectangle inherits from Shape
Rectangle.prototype = Object.create(Shape.prototype);

var rect = new Rectangle();

console.log("x = "+rect);
console.log("rect is an instance of Rectangle? "+(rect instanceof Rectangle)) //true.
console.log("rect is an instance of Shape? "+(rect instanceof Shape)) //true.

rect.move(1, 2); //Outputs, "Shape moved.

[~/Dropbox/src/javascript/inheritance]$ node inh1.js
x = (0, 0)
rect is an instance of Rectangle? true
```

```
rect is an instance of Shape? true
Shape moved to (1, 2)
```

## 63.2. Ejercicios

1. ¿Cuál es la salida?

```
> String.prototype.repeat = function(times) {
... return new Array(times+1).join(this)
...
[Function]
>
undefined
> "hello".repeat(3)

>
```

2. ¿Cuáles son los resultados?

```
> z = Object.create({x:1, y:2})
{}
> z.x

> z.y

> z.__proto__

> z.__proto__.__proto__

> z.__proto__.__proto__.__proto__
```

3. Describa las salidas:

```
> obj = {x : 'something' }
{x: 'something'}
> w = Object.create(obj)
{}
> w.x
'something'
> w.x = "another thing"
'another thing'
> w.__proto__

> obj == w.__proto__
```

4. Explique la salida:

```
> obj = {x : { y : 1} }
{x: {y: 1}}
> w = Object.create(obj)
{}
```

```

> w.x == obj.x
true
> w.x.y = 2
2
> obj
{ x: { y: 2 } }
>

```

5. Explique las salidas:

```

> inherit = Object.create
[Function: create]
> o = {}
{}
> o.x = 1
1
> p = inherit(o)
{}
> p.x
1
> p.y = 2
2
> p
{ y: 2 }
> o
{ x: 1 }
> o.y
undefined
> q = inherit(p)
{}
> q.z = 3
3
> q
{ z: 3 }
> s = q.toString()
'[object Object]'
> q.x+q.y+q.z
6
> o.x
1
> o.x = 4
4
> p.x
4
> q.x
4
> q.x = 5
5
> p.x
4
> o.x
4

```

# Capítulo 64

## Ajax

### 64.1. Enlaces y Bibliografía

- <http://www.w3schools.com/ajax/>
- Jhon Resig, Bear Bibeault *Secrets of the JavaScript Ninja* Manning
- jQuery Cookbook. O'Reilly
- Scripting with javaScript and Ajax. Charles Wyke-Smith. New Riders. 2010

### 64.2. JSOn y JSONP: When ajax calls are in a different domain from the server.

JSON versus JSONP Tutorial y [https://github.com/craic/json\\_jsonp\\_tutorial](https://github.com/craic/json_jsonp_tutorial)

```
[~/sinatra/sinatra-json_jsonp(master)]$ pwd -P
/Users/casiano/local/src/ruby/sinatra/sinatra-json_jsonp
[~/sinatra/sinatra-json_jsonp(master)]$ git remote -v
origin git@github.com:craic/json_jsonp_tutorial.git (fetch)
origin git@github.com:craic/json_jsonp_tutorial.git (push)
```

## Capítulo 65

# JavaScript en el Lado del Servidor

Node es un intérprete JavaScript escrito en C++ y con una API ligada a Unix para trabajar con procesos, fichero, sockets, etc. Los programas Node por defecto nunca se bloquean. Node utiliza manejadores de eventos que a menudo se implementan haciendo uso de funciones anidadas y clausuras.

### 65.1. Instalar Node.js

1. <http://nodejs.org/download/>
2. Should I install node.js on Ubuntu using package manager or from source?
3. The Node Beginner Book

Hay muchos libros de JavaScript y Node.js gratuitos disponibles. Véase el proyecto en GitHub:

- [free-programming-books](#)

y en concreto las secciones:

- [JavaScript](#)
- [Node.js](#)
- [JQuery](#)

### 65.2. Primeros Pasos. Un Ejemplo Simple

```
[~/src/javascript/node.js/hector_correa_introduction_to_node(master)]$ cat -n hello_world.js
1 console.log("Hello world!");
2 a = ['batman', 'robin'];
3 a.push("superman");
4 console.log(a);
5 h = { name: 'jane rodriguez-leon', department: 'IT' };
6 console.log(h);
7 console.log(h['name']);

[~/src/javascript/node.js/hector_correa_introduction_to_node(master)]$ node hello_world.js
Hello world!
['batman', 'robin', 'superman']
{ name: 'jane rodriguez-leon', department: 'IT' }
jane rodriguez-leon
```

```
[~/Dropbox/academica/ETSII/grado/LPP/LPPbook]$ node
> .help
.break Sometimes you get stuck, this gets you out
.clear Alias for .break
.exit Exit the repl
.help Show repl options
.load Load JS from a file into the REPL session
.save Save all evaluated commands in this REPL session to a file

> console.log("Hello world!")
Hello world!
undefined
> a = ['batman', 'robin']
['batman', 'robin']
> a.push("superman")
3
> a
['batman', 'robin', 'superman']
> h = { name: 'jane rodriguez-leon', department: 'IT' }
{ name: 'jane rodriguez-leon',
 department: 'IT' }
> h['name']
>jane rodriguez-leon'
> 4+2
6
> _ # ultimo valor evaluado
6
> _+1
7
>
> a = [1,2,3]
[1, 2, 3]
> a.forEach(function(e) { console.log(e); })
1
2
3
> a.forEach(function(v) {
... console.log(v
..... .break
> a
[1, 2, 3]
> .exit # también CTRL-D en Unix
[~/Dropbox/academica/ETSII/grado/LPP/LPPbook]$
```

### 65.3. Usando REPL desde un programa

Es posible crear un bucle REPL en cualquier punto de nuestro programa - quizá para depurarlo. Para ello usamos la función `repl.start`. Esta función retorna una instancia REPLServer. Acepta como argumento un objeto `options` que toma los siguientes valores:

1. `prompt` - the prompt and stream for all I/O. Defaults to `;`.
2. `input` - the readable stream to listen to. Defaults to `process.stdin`.

3. `output` - the writable stream to write readline data to. Defaults to `process.stdout`.
4. `terminal` - pass true if the stream should be treated like a TTY, and have ANSI/VT100 escape codes written to it. Defaults to checking `isTTY` on the output stream upon instantiation.
5. `eval` - function that will be used to eval each given line. Defaults to an async wrapper for `eval()`.
6. `useColors` - a boolean which specifies whether or not the writer function should output colors. If a different writer function is set then this does nothing. Defaults to the repl's terminal value.
7. `useGlobal` - if set to true, then the repl will use the global object, instead of running scripts in a separate context. Defaults to false.
8. `ignoreUndefined` - if set to true, then the repl will not output the return value of command if it's undefined. Defaults to false.
9. `writer` - the function to invoke for each command that gets evaluated which returns the formatting (including coloring) to display. Defaults to `util.inspect`.

```
[~/Dropbox/src/javascript/node.js/repl(master)]$ cat repl.js
var repl = require("repl");

connections = 0;

repl.start({
 prompt: "node via stdin> ",
 input: process.stdin,
 output: process.stdout
});

[~/Dropbox/src/javascript/node.js/repl(master)]$ node repl.js
node via stdin> 2+3
5
node via stdin> .exit
```

el bucle REPL proporciona acceso a las variables de ámbito global. Es posible hacer explícitamente visible una variable al REPL asignándosela al `context` asociado con el REPLServer. Por ejemplo:

```
[~/Dropbox/src/javascript/node.js/repl(master)]$ cat repl2.js
var repl = require("repl");

z = 4
repl.start({
 prompt: "node via stdin> ",
 input: process.stdin,
 output: process.stdout
}).context.m = "message";
```

Las variables en el objeto `context` se ven como locales al REPL:

```
[~/Dropbox/src/javascript/node.js/repl(master)]$ node repl2.js
node via stdin> z
4
node via stdin> m
'message'
```

## 65.4. Usando REPL via un socket TCP

```
[~/Dropbox/src/javascript/node.js/repl(master)]$ cat repl_server.js
var net = require("net"),
 repl = require("repl");

connections = 0;

net.createServer(function (socket) {
 connections += 1;
 repl.start({
 prompt: "node via TCP socket> ",
 input: socket,
 output: socket
 }).on('exit', function() {
 socket.end();
 });
}).listen(5001);

[~/Dropbox/src/javascript/node.js/repl(master)]$ node repl_server.js
```

Podemos ahora usar *netcat* para comunicar con el servidor:

```
[~/Dropbox/src/javascript/node.js/hector_correa_introduction_to_node(master)]$ nc -v localhost
nc: connect to localhost port 5001 (tcp) failed: Connection refused
Connection to localhost 5001 port [tcp/commplex-link] succeeded!
node via TCP socket> a = 2+3
5
node via TCP socket> a
5
node via TCP socket> .exit
[~/Dropbox/src/javascript/node.js/hector_correa_introduction_to_node(master)]$
```

## 65.5. Referencias sobre REPL

1. Véase Node.js v0.8.18 Manual & Documentation
2. Véase How do I use node's REPL? en <http://docs.nodejitsu.com/>.

## 65.6. Entrada Salida en Node.js

1. How To Read User Input With NodeJS por Nikolay V. Nemshilov

## 65.7. Debugger

1. Node.js debugger

## 65.8. Modulos

### 65.8.1. Introducción

```
[~/javascript/node.js/creating_modules(master)]$ cat foo.js
var circle = require('./circle.js');
```

```

console.log('The area of a circle of radius 4 is '
 + circle.area(4));

[~/javascript/node.js/creating_modules(master)]$ cat circle.js
var PI = Math.PI;

exports.area = function (r) {
 return PI * r * r;
};

exports.circumference = function (r) {
 return 2 * PI * r;
};

```

[~/javascript/node.js/creating\_modules(master)]\$ node foo.js The area of a circle of radius 4

El módulo `circle.js` exporta las funciones `area()` y `circumference()`. Para exportar un objeto lo añadimos al objeto especial `exports`.

Las variables locales al módulo serán privadas. En este ejemplo la variable `PI` es privada a `circle.js`.

```

[~/javascript/node.js/creating_modules(master)]$ node debug foo.js
< debugger listening on port 5858
connecting... ok
break in foo.js:1
1 var circle = require('./circle.js');
2 console.log('The area of a circle of radius 4 is '
3 + circle.area(4));
debug> n
break in foo.js:2
1 var circle = require('./circle.js');
2 console.log('The area of a circle of radius 4 is '
3 + circle.area(4));
4
debug> repl
Press Ctrl + C to leave debug repl
> circle
{ circumference: [Function],
 area: [Function] }
> circle.area(2)
12.566370614359172
> PI
ReferenceError: PI is not defined
>

```

### 65.8.2. Ciclos

```

[~/javascript/node.js/creating_modules/cycles(master)]$ cat a.js
console.log('a starting');
exports.done = false;
var b = require('./b.js');
console.log('in a, b.done = %j', b.done);
exports.done = true;
console.log('a done');

```

```
[~/javascript/node.js/creating_modules/cycles(master)]$ cat b.js
console.log('b starting');
exports.done = false;
var a = require('./a.js');
console.log('in b, a.done = %j', a.done);
exports.done = true;
console.log('b done');

[~/javascript/node.js/creating_modules/cycles(master)]$ cat main.js
console.log('main starting');
var a = require('./a.js');
var b = require('./b.js');
console.log('in main, a.done=%j, b.done=%j', a.done, b.done);
```

When `main.js` loads `a.js`, then `a.js` in turn loads `b.js`. At that point, `b.js` tries to load `a.js`. In order to prevent an infinite loop an unfinished copy of the `a.js` exports object is returned to the `b.js` module. `b.js` then finishes loading, and its exports object is provided to the `a.js` module.

By the time `main.js` has loaded both modules, they're both finished. The output of this program would thus be:

```
[~/javascript/node.js/creating_modules/cycles(master)]$ node main.js
main starting
a starting
b starting
in b, a.done = false
b done
in a, b.done = true
a done
in main, a.done=true, b.done=true
```

### 65.8.3. Especificación de Ficheros Conteniendo Módulos

1. If the exact filename is not found, then node will attempt to load the required filename with the added extension of `.js`, `.json`, and then `.node`.
2. `.js` files are interpreted as JavaScript text files, and `.json` files are parsed as JSON text files. `.node` files are interpreted as compiled addon modules
3. A module prefixed with `'/'` is an absolute path to the file. For example, `require('/home/marco/foo.js')` will load the file at `/home/marco/foo.js`.
4. A module prefixed with `'./'` is relative to the file calling `require()`.
5. Without a leading verb—`'/'` or `'./'` to indicate a file, the module is either a *core module* or is loaded from a `node_modules` folder.
6. If the given path does not exist, `require()` will throw an Error with its code property set to `MODULE_NOT_FOUND`.

### 65.8.4. Carga desde Carpetas `node_modules`

1. If the module identifier passed to `require()` is not a native module, and does not begin with `'/'`, `'..'`, or `'./'`, then node starts at the parent directory of the current module, and adds `/node_modules`, and attempts to load the module from that location.
2. If it is not found there, then it moves to the parent directory, and so on, until the root of the tree is reached.

For example, if the file at '/home/ry/projects/foo.js' called `require('bar.js')`, then node would look in the following locations, in this order:

```
/home/ry/projects/node_modules/bar.js
/home/ry/node_modules/bar.js
/home/node_modules/bar.js
/node_modules/bar.js
```

This allows programs to localize their dependencies, so that they do not clash.

### 65.8.5. Las Carpetas Usadas Como Módulos

It is convenient to organize programs and libraries into self-contained directories, and then provide a single entry point to that library.

There are a few ways in which a folder may be passed to `require()` as an argument.

1. The first is to create a `package.json` file in the root of the folder, which specifies a `main` module. An example `package.json` file might look like this:

```
{ "name" : "some-library",
 "main" : "./lib/some-library.js" }
```

If this was in a folder at `./some-library`, then `require('./some-library')` would attempt to load `./some-library/lib/some-library.js`.

This is the extent of Node's awareness of `package.json` files.

2. If there is no `package.json` file present in the directory, then node will attempt to load an `index.js` or `index.node` file out of that directory.

For example, if there was no `package.json` file in the above example, then `require('./some-library')` would attempt to load:

```
./some-library/index.js
./some-library/index.node
```

### 65.8.6. Caching

1. Modules are cached after the first time they are loaded. This means (among other things) that every call to `require('foo')` will get exactly the same object returned, if it would resolve to the same file.
2. Multiple calls to `require('foo')` may not cause the module code to be executed multiple times. This is an important feature. With it, *partially done* objects can be returned, thus allowing transitive dependencies to be loaded even when they would cause cycles.
3. If you want to have a module execute code multiple times, then export a function, and call that function.
4. Modules are cached based on their **resolved filename**. Since modules may resolve to a different filename based on the location of the calling module (loading from `node_modules` folders), it is not a guarantee that `require('foo')` will always return the exact same object, if it would resolve to different files.

### 65.8.7. El Objeto module y module.exports

1. In each module, the `module` free variable is a reference to the object representing the current module.
2. In particular `module.exports` is the same as the `exports` object.
3. `module` isn't actually a global but rather local to each module.
4. The `exports` object is created by the `Module` system. Sometimes this is not acceptable, many want their module to be an instance of some class. To do this assign the desired export object to `module.exports`.

```
■ [~/javascript/node.js/creating_modules/module_exports(master)]$ cat a.js
var EventEmitter = require('events').EventEmitter;

module.exports = new EventEmitter();

// Do some work, and after some time emit
// the 'ready' event from the module itself.
setTimeout(function() {
 module.exports.emit('ready');
}, 1000);

■ $ cat main.js
var a = require('./a');
a.on('ready', function() {
 console.log('module a is ready');
});

■ $ node main.js
module a is ready
```

La asignación a `module.exports` debe hacerse inmediatamente. No puede hacerse en un callback.

### 65.8.8. Algoritmo de Búsqueda Ejecutado por require require(X) from module at path Y

1. If X is a core module,
  - a) return the core module
  - b) STOP
2. If X begins with './' or '/' or '../'
  - a) LOAD\_AS\_FILE(Y + X)
  - b) LOAD\_AS\_DIRECTORY(Y + X)
3. LOAD\_NODE\_MODULES(X, dirname(Y))
4. THROW "not found"

#### LOAD\_AS\_FILE(X)

1. If X is a file, load X as JavaScript text. STOP
2. If X.js is a file, load X.js as JavaScript text. STOP
3. If X.node is a file, load X.node as binary addon. STOP

## **LOAD\_AS\_DIRECTORY(X)**

1. If X/package.json is a file, a. Parse X/package.json, and look for "main" field. b. let M = X + (json main field) c. LOAD\_AS\_FILE(M)
2. If X/index.js is a file, load X/index.js as JavaScript text. STOP
3. If X/index.node is a file, load X/index.node as binary addon. STOP

## **LOAD\_NODE\_MODULES(X, START)**

1. let DIRS=NODE\_MODULES\_PATHS(START)
2. for each DIR in DIRS: a. LOAD\_AS\_FILE(DIR/X) b. LOAD\_AS\_DIRECTORY(DIR/X)

## **NODE\_MODULES\_PATHS(START)**

1. let PARTS = path split(START)
2. let ROOT = index of first instance of "node\_modules" in PARTS, or 0
3. let I = count of PARTS - 1
4. let DIRS = []
5. while I >= ROOT, a. if PARTS[I] = "node\_modules" CONTINUE c. DIR = path join(PARTS[0 .. I] + "node\_modules") b. DIRS = DIRS + DIR c. let I = I - 1
6. return DIRS

## **65.9. Como Crear tu Propio Módulo en Node.js**

### **65.9.1. Introducción**

Cuando Node carga nuestro fichero JavaScript crea un nuevo ámbito. Cuando estamos en nuestro módulo, no podemos ver el ámbito externo lo que evita las colisiones de nombres.

### **65.9.2. Un Fichero package.json**

Creamos un fichero en la raíz de nuestro proyecto con nombre `package.json`. Este fichero describe nuestro proyecto. Es esencial si vamos a publicar nuestro proyecto con `npm`.

Podemos especificar en este fichero:

1. Name, version, description, and keywords to describe your program.
2. A homepage where users can learn more about it.
3. Other packages that yours depends on.

Si hemos instalado `npm` podemos usar el comando `npm init` para empezar. Véase `npm help json` para obtener información sobre este fichero:

La cosa mas importante a especificar cuando estamos escribiendo un programa para su uso por otros, es el módulo `main`. Este constituirá el punto de entrada a nuestro programa.

Es esencial documentar las dependencias.

El siguiente es un ejemplo de fichero `package.json` tomado del proyecto `ebnf-parser`:

```
[~/javascript/PLgrado/ebnf-parser(master)]$ cat -n package.json
 1 {
 2 "name": "ebnf-parser",
 3 "version": "0.1.1",
 4 "description": "A parser for BNF and EBNF grammars used by jison",
 5 "main": "ebnf-parser.js",
 6 "scripts": {
 7 "test": "make test"
 8 },
 9 "repository": "",
10 "keywords": [
11 "bnf",
12 "ebnf",
13 "grammar",
14 "parser",
15 "jison"
16],
17 "author": "Zach Carter",
18 "license": "MIT",
19 "devDependencies": {
20 "jison": "0.4.x",
21 "lex-parser": "0.1.0",
22 "test": "*"
23 }
24 }
```

### 65.9.3. README y otros documentos

Pon la información basica acerca del módulo en la raíz del proyecto. Como ejemplo veamos el `README.md` (observa que está en formato `markdown`) del proyecto `ebnf-parser`:

```
$ cat README.md

ebnf-parser

A parser for BNF and EBNF grammars used by jison.
```

```
install

 npm install ebnf-parser
```

```
build
```

To build the parser yourself, clone the git repo then run:

```
 make
```

This will generate ‘`parser.js`’, which is required by ‘`ebnf-parser.js`’.

```
usage
```

The parser translates a string grammar or JSON grammar into a JSON grammar that `jison` can use

```

var ebnfParser = require('ebnf-parser');

// parse a bnf or ebnf string grammar
ebnfParser.parse("%start ... %");

// transform an ebnf JSON grammar
ebnfParser.transform({"ebnf": ...});

```

## example grammar

The parser can parse its own BNF grammar, shown below:

```

%start spec

/* grammar for parsing jison grammar files */

%{
var transform = require('./ebnf-transform').transform;
var ebnf = false;
%}

%%

spec
: declaration_list '%%' grammar optional_end_block EOF
 {$$ = $1; return extend($$, $$); }
| declaration_list '%%' grammar '%%' CODE EOF
 {$$ = $1; yy.addDeclaration($$, {include:$5}); return extend($$, $$); }
;

optional_end_block
:
| '%%'
;

declaration_list
: declaration_list declaration
 {$$ = $1; yy.addDeclaration($$, $$); }
|
 {$$ = {};}
;

declaration
: START id
 {$$ = {start: $$}; }
| LEX_BLOCK
 {$$ = {lex: $$}; }
| operator
 {$$ = {operator: $$}; }
| ACTION
 {$$ = {include: $$}; }
;

```

```

operator
: associativity token_list
 {$$ = [$1]; $$.push .apply ($$, $2);}
;

associativity
: LEFT
 {$$ = 'left';}
| RIGHT
 {$$ = 'right';}
| NONASSOC
 {$$ = 'nonassoc';}
;

token_list
: token_list symbol
 {$$ = $1; $$.push ($2);}
| symbol
 {$$ = [$1];}
;

grammar
: production_list
 {$$ = $1;}
;

production_list
: production_list production
 {$$ = $1;
 if ($2[0] in $$) $$[$2[0]] = $$[$2[0]].concat ($2[1]);
 else $$[$2[0]] = $2[1];}
| production
 {$$ = {}; $$[$1[0]] = $1[1];}
;

production
: id ':' handle_list ';'
 {$$ = [$1, $3];}
;

handle_list
: handle_list '|' handle_action
 {$$ = $1; $$.push ($3);}
| handle_action
 {$$ = [$1];}
;

handle_action
: handle prec action
 {$$ = [($1.length ? $1.join (' ') : '')];
 if ($3) $$.push ($3);
 if ($2) $$.push ($2);}
;

```

```

 if ($.length === 1) $$ = $$[0];
 }
;

handle
: handle expression_suffix
{$$ = $1; $$.push($2)}
|
{$$ = [];}
;

handle_sublist
: handle_sublist '| handle
{$$ = $1; $$.push($3.join(' '));}
| handle
{$$ = [$1.join(' ')];}
;

expression_suffix
: expression suffix
{$$ = $expression + $suffix; }
;

expression
: ID
{$$ = $1; }
| STRING
{$$ = ebnf ? ""+$1+"": $1; }
| '(' handle_sublist ')'
{$$ = (' + $handle_sublist.join(' | ') + ')'; }
;

suffix
: {$$ = ''}
| '*'
| '?'
| '+'
;

prec
: PREC symbol
{$$ = {prec: $2};}
|
{$$ = null;}
;

symbol
: id
{$$ = $1; }
| STRING
{$$ = yytext; }
;

```

```

id
: ID
 {$$ = yytext;}
;

action
: '{' action_body '}''
 {$$ = $2;}
| ACTION
 {$$ = $1;}
| ARROW_ACTION
 {$$ = '$$ =' +$1+';';}
|
 {$$ = '';}
;

action_body
:
 {$$ = '';}
| ACTION_BODY
 {$$ = yytext;}
| action_body '{' action_body '}', ACTION_BODY
 {$$ = $1+$2+$3+$4+$5;}
| action_body '{' action_body '}''
 {$$ = $1+$2+$3+$4;}
;

%%

// transform ebnf to bnf if necessary
function extend (json, grammar) {
 json.bnf = ebnf ? transform(grammar) : grammar;
 return json;
}

license

```

MIT

En general se anima a que uses el formato markdown. Salva el fichero como `README.md`.

La documentación adicional se pone en un directorio `./docs`. Los ficheros markdown terminan en `.md` y los html en `.html`.

#### 65.9.4. Véase También

- How To: Create Your Own Node.js Module por Isaac Z. Schlueter autor de npm. Véase también este gist en GitHub
- Creating Custom Modules
- How to Build a Nodejs Npm Package From Scratch. May 2012 Decodize

## 65.10. JQuery en Node.js

Es posible instalar JQuery en Node.js. Tuve algún problema para instalar jquery con algunas versiones de Node pero funcionó con la 0.10.10:

```
~/sinatra/rockpaperscissors(master)]$ n
* 0.10.10
 0.11.2
 0.8.17
```

El programa n es un gestor de versiones de Node.js. Otro gestor de versiones del intérprete es nvm.

Una vez instalado, podemos usarlo desde coffeescript via node.js:

```
coffee> $ = require 'jquery'; null
null
coffee> $("<h1>test passes</h1>").appendTo "body" ; null
null
coffee> console.log $("body").html()
<h1>test passes</h1>
undefined
coffee>
coffee> $.each [4,3,2,1], (i,v)-> console.log "index: #{i} -> value: #{v}"
index: 0 -> value: 4
index: 1 -> value: 3
index: 2 -> value: 2
index: 3 -> value: 1
[4, 3, 2, 1]
```

## 65.11. Mas sobre Node

1. Véase el libro 'Learning Node' de S. Powers [?].
2. Node.js
3. docs.nodejitsu.com: collection of node.js how-to articles. These articles range from basic to advanced. They provide relevant code samples and insights into the design and philosophy of node itself
4. <http://howtonode.org/> contiene un número creciente de tutoriales
5. El manual de node.js puede encontrarse en formato pdf en el proyecto <https://github.com/zeMirco/nodejs-pdf-docs> en GitHub. En concreto en este enlace
6. Guías de node.js de Felix Geisendörfer
7. Introduction to Node.js por Hector Correa
8. El Libro para Principiantes en Node.js por Manuel Kiessling y Herman A. Junge

Para aprender JavaScript podemos usar el libro eloquent JavaScript de Marijn Haverbeke [?].

# **Capítulo 66**

## **Backbone**

1. Developing Backbone.js Applications

# Capítulo 67

## Closure Tools

### 67.1. Véase También

1. Google I/O 2011: JavaScript Programming in the Large with Closure Tools (YouTube)
2. The Closure Tools project is an effort by Google to open source the tools used in many of Google's sites and web applications
3. Herramientas:
  - a) Closure Compiler en Google-Code
  - b) Closure Library en Google-Code
  - c) Closure Template en Google-Code
  - d) Closure Linter en Google-Code
4. Getting Started with the Closure Library (Hello World!)

## Capítulo 68

# Semantic Templates

### 68.1. Moustache

- <http://mustache.github.io/>
- Tutorial: HTML Templates with Mustache.js

# Capítulo 69

## Pruebas

### 69.1. Testing en JavaScript: Fácil y Rápido

1. Quick Tip: Quick and Easy JavaScript Testing with “Assert” por Jeffrey Way

### 69.2. Unit Testing, TDD y BDD con Jasmine

1. Jasmine: BDD Style JavaScript Testing Hello World por Chris McNabb (YouTube Sep. 2012)
2. Download Jasmine
3. Testing Your JavaScript with Jasmine Andrew Burgess on Aug 4th 2011
4. Unit Testing in JavaScript via Jasmine (Youtube)
5. Jasmine en GitHub
6. Behavior Driven Testing with Jasmine (YouTube, Davis Frank de Pivotal Labs, Contiene una introducción a BDD)
7. Jasmine Wiki
8. Testem tutorial en net.tutplus (trabaja sobre Jasmine y sobre Coffee)
9. Jasmine Matchers: Class jasmine.Matchers

## Capítulo 70

# Buenas Prácticas y Patrones

### 70.1. Véase También

1. Google I/O 2011: Learning to Love JavaScript por Alex Russell, Mayo 2011. ¡Excelente!
2. traceur compiler
3. Google I/O 2011: JavaScript Programming in the Large with Closure Tools

## Capítulo 71

# Herramientas para JavaScript

### 71.1. npm

#### 71.1.1. Specifying dependencies in Node.js

- Put a `package.json` file in the root of your project
- List your deps in that file

```
{ "name" : "my-project"
, "version" : "1.0.0"
, "dependencies" : { "express" : "1.0.0" } }
```

- `npm install` Since you're calling this with no args, and not in global mode, it'll just install all your deps locally.
- `require("express")`

### 71.2. n

`n` es una herramienta parecida a `rvm` para Node.js:

```
$sudo npm install n -g
y
[~/Dropbox/src/javascript/node.js/creating_modules(master)]$ n help
Usage: n [options] [COMMAND] [config]
Commands:
```

<code>n</code>	Output versions installed
<code>n latest [config ...]</code>	Install or activate the latest node release
<code>n stable [config ...]</code>	Install or activate the latest stable node release
<code>n &lt;version&gt; [config ...]</code>	Install and/or use node <version>
<code>n use &lt;version&gt; [args ...]</code>	Execute node <version> with [args ...]
<code>n bin &lt;version&gt;</code>	Output bin path for <version>
<code>n rm &lt;version ...&gt;</code>	Remove the given version(s)
<code>n prev</code>	Revert to the previously activated version
<code>n --latest</code>	Output the latest node version available
<code>n --stable</code>	Output the latest stable node version available
<code>n ls</code>	Output the versions of node available

Options:

```
-V, --version Output current version of n
-h, --help Display help information
```

Aliases:

```
which bin
use as
list ls
- rm
```

```
$ sudo n latest
```

### 71.3. Google Chrome y Javascript

#### Enlaces Relacionados

1. Chrome: Developper Tools Tips and Tricks).
2. Chrome Developer Tools Tutorial: Elements (Part 1/2) (YouTube)
3. Editing styles and DOM
4. Building Browser Apps with Google Chrome

### 71.4. Plugins, Editores, IDEs

- jslint lint plugin para vim
- vim plugins for HTML and CSS hi-speed coding. disponible en <http://www.vim.org> y en github <https://github.com/mattn/zencoding-vim/>
- Vim Essential Plugin: Sparkup parecido a zenconding. El tutorial es de 2011.

### 71.5. Grunt

1. Grunt - The Basics Youtube
2. Grunt home page

### 71.6. Beautifiers, Pretty-Printers

1. beautifier de Rickeyski

### 71.7. Modulos

1. NODE.JS Modules

## Capítulo 72

# Google Maps JavaScript API

- Google Maps JavaScript API v3
- This is How You Use the Google Maps API – screencast NetTuts
- Google Maps API Tutorial W3Schools Home

## **Capítulo 73**

# **Mobile Web Applications Development with HTML5**

1. Lectures
2. Resources
3. Examples
4. Slides

## Capítulo 74

# Semantic Templates en Javascript

### 74.1. Moustache

- <http://mustache.github.io/>
- Tutorial: HTML Templates with Mustache.js

### 74.2. handlebars

Handlebars provides the power necessary to let you build semantic templates effectively with no frustration.

Mustache templates are compatible with Handlebars, so you can take a Mustache template, import it into Handlebars, and start taking advantage of the extra Handlebars features.

- <https://github.com/crguezl/handlebars-examples>
- <http://handlebarsjs.com/>
- Demo of Handlebars, and why you should consider a templating engine por Raymond Camden
- Using the Handlebars precompiler, you can precompile your Handlebars templates to save time on the client and reduce the required runtime size of the handlebars library

# Capítulo 75

## CSS

- Create a Responsive Website Using HTML5 and CSS3 YouTube
- Responsive Web Design by ETHAN MARCOTTE May 25, 2010

## Capítulo 76

# Bootstrap: Javascript y Hojas de Estilo

### 76.1. BootStrap

Para descargar BootStrap nos vamos a la página:

<http://getbootstrap.com/>

The folks over at MaxCDN provide CDN support for Bootstrap's CSS and JavaScript. Just use these Bootstrap CDN links.

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css">

<!-- Optional theme -->
<link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap-theme.min.css">

<!-- Latest compiled and minified JavaScript -->
<script src="//netdna.bootstrapcdn.com/bootstrap/3.0.3/js/bootstrap.min.js"></script>
```

Once downloaded, unzip the compressed folder to see the structure of (the compiled) Bootstrap.

**Usando Bootstrap con Sinatra** Véase:

1. Bootstrap your Web Application with Ruby and Sinatra por David Sale.
2. Véase también: [sinatra-bootstrap](#)
3. [crguezl/sinatra-bootstrap-example](#) en GitHub en GitHub.

### Tutoriales de BootStrap

1. [w3schools.com](#)
2. Tutoriales en YouTube del 1 al 15 por <http://www.creativitytuts.org/> :
  - a) Twitter Bootstrap Tutorial 1 - Introduction / Setup por CreativityTuts
  - b) Twitter Bootstrap Tutorial 2 - Forms por CreativityTuts
  - c) Twitter Bootstrap Tutorial 3 - In-line & Search Form por CreativityTuts
  - d) Twitter Bootstrap Tutorial 4 - Tables por CreativityTuts
3. Tutorials on Using Bootstrap for the Easy Start
4. BootStrap Tutorial Parts I and II

# Capítulo 77

## JQuery Mobile

1. Query Mobile uses a very simple and powerful approach to define the content of the webapp
2. JQuery Mobile uses an *unobtrusive approach*, meaning that our HTML documents will work even without jQuery Mobile loading properly
3. The main unit of the framework is the *page*.
4. A page is just a **div** element with a specific *role*
5. One HTML document can host one page or many pages inside the same file
6. We will be able to link to pages inside the same HTML document, or to pages on external HTML documents, using simple HTML markup, such as the **a** tag.
  1. jQuery Mobile uses standard HTML markup, such as the **div** tag
  2. To define what the framework should do with that **div**, we define a **role**
  3. A **role** in jQuery Mobile is defined using the attribute **data-role**
  4. For example,

```
<div data-role="page">
```

1. The usage of **data-<something>** or **data-\*** attributes on an HTML tag is an HTML5 feature called *custom data attributes* that allows us to define whatever attribute we want to add to a tag while maintaining an HTML-valid document. It is useful for adding custom metadata to tags without invalidating the markup.
2. jQuery Mobile often uses this ability to define custom attributes for the framework
3. **data-role** is not a new HTML5 new attribute

### Role

### Description

page	Defines a page, the unit that jQuery Mobile uses to show content
header	Header of a page
content	Content of a page
footer	Footer of a page
navbar	Defines a navigation bar, typically inside a header button Renders a visual
controlgroup	Renders a component
collapsible	Collapsible panel of content inside a page
collapsible-set	Group of collapsible panels (accordion)
fieldcontain	Container for form fields

listview	Content of multiple items as a list
dialog	Dialog page
slider	Visual slider for Boolean values
nojs	Element that will be hidden on jQuery Mobile's compatible browsers

**Parte V**

**COFFESCRIP<sup>T</sup>**

# Capítulo 78

## Introducción

CoffeeScript is a programming language that looks like this:

```
[~/coffee/jump_start_coffeescript/chapter01/casiano(master)]$ coffee
coffee> hello = (name) -> "Hello, #{name}!"
[Function]
coffee> hello('world!')
'Hello, world!!'
coffee> hello 'world!'
'Hello, world!!'
```

Here, we're defining and then calling a function, `hello`, which accepts a single parameter, `name`.

### Instalación

The CoffeeScript compiler is itself written in CoffeeScript, using the Jison parser generator. The command-line version of `coffee` is available as a Node.js utility. The core compiler however, does not depend on Node, and can be run in any JavaScript environment, or in the browser.

To install, first make sure you have a working copy of the latest stable version of Node.js, and npm (the Node Package Manager). You can then install CoffeeScript with npm:

```
npm install -g coffee-script
```

(Leave off the `-g` if you don't wish to install globally.)

If you'd prefer to install the latest master version of CoffeeScript, you can clone the CoffeeScript source repository from GitHub, or download the source directly. To install the lastest master CoffeeScript compiler with npm:

```
npm install -g http://github.com/jashkenas/coffee-script/tarball/master
```

Or, if you want to install to `/usr/local`, and don't want to use npm to manage it, open the `coffee-script` directory and run:

```
sudo bin/cake install
```

CoffeeScript includes a (very) simple build system similar to Make and Rake. Naturally, it's called `Cake`, and is used for the tasks that build and test the CoffeeScript language itself. Tasks are defined in a file named `Cakefile`, and can be invoked by running `cake [task]` from within the directory. To print a list of all the tasks and options, just type `cake`.

## **Enlaces Relacionados**

1. CoffeeScript book
2. Railcast: CoffeeScript
3. vim plugin para CoffeScript
4. A CoffeeScript Intervention. Five Things You Thought You Had to Live with in JavaScript por Trevor Burnham en PragPub
5. A taste of CoffeeScript (part 2)
6. Some real world examples of Coffescript and jQuery por Stefan Bleibinhaus

## Capítulo 79

# Ambito/Scope

- Lexical Scope in CoffeeScript por Reg Braithwaite raganwald
- Reg Braithwaite raganwald
- ristrettolo.gy, CoffeeScript Ristretto Online

**Parte VI**

**HERRAMIENTAS**

## Capítulo 80

# Ruby en Windows

- The easiest way to install Ruby and RubyGems if you use Windows is to use the Ruby Installer.  
[rubyinstaller](#)
- Un pequeño tutorial en Español de como instalar Ruby en Windows
- RailsInstaller is a way to install Ruby on Rails on Windows
- Pik: Ruby version manager for Windows
- [pik en GitHub](#)
- Setting up Sinatra and DataMapper on Windows
- Tutorial en Español de como instalar Ruby en Windows

# Capítulo 81

## Ruby Version Manager: RVM

rvm nos permite instalar y gestionar varios intérpretes ruby y conjuntos de gemas (gemsets).

### 81.1. Instalación de RVM

Para instalarla debemos hacer:

```
$ \curl -#L https://get.rvm.io | bash -s stable --autolibs=3 --ruby
```

Point to be noted is, there is a backslash before curl. This prevents misbehaving if you have aliased it with configuration in your `~/.curlrc` file.

Instalela como un usuario normal. Nunca como root a menos que quiera hacer una instalación multiusuario. Asegúrese que su usuario no pertenece a los grupos de administración.

If the install script is run as a standard, non-root user, RVM will install into the current user's home directory. RVM by default will modify user startup files.

Véase la información en Ruby Version Manager.

### 81.2. Actualización

Para obtener la versión:

```
[~/local/src]$ rvm -v
rvm 1.22.7 (stable) by Wayne E. Seguin <wayneeseguin@gmail.com>, Michal Papis <mpapis@gmail.co
```

Es conveniente actualizar tan a menudo como sea posible la propia rvm. Véase Upgrading RVM

Una vez instalada `rvm get stable` permite actualizar `rvm`:

```
rvm get {stable|latest|latest-x.y|x.y.z|head|master|branch|help} [--auto-dotfiles] [--autolibs]
```

Una vez actualizada `rvm reload` nos permite la versión recién instalada:

```
[~]$ rvm list
```

```
A RVM version 1.20.9 (stable) is installed yet 1.18.18 () is loaded.
```

```
Please do one of the following:
```

- \* 'rvm reload'
- \* open a new shell
- \* 'echo rvm\_auto\_reload\_flag=1 >> ~/.rvmrc' # for auto reload with msg.
- \* 'echo rvm\_auto\_reload\_flag=2 >> ~/.rvmrc' # for silent auto reload.

```
[~]$ rvm reload
```

```
RVM reloaded!
```

Ahora hemos cambiado de versión:

```
[~]$ rvm --version
```

```
rvm 1.20.9 (stable) by Wayne E. Seguin <wayneeseguin@gmail.com>, Michal Papis <mpapis@gmail.co
```

### 81.3. Versiones Conocidas del Intérprete

El comando `rvm list known` permite conocer que versiones del intérprete pueden ser instaladas via rvm:

```
[~]$ rvm list known
```

```
MRI Rubies
[ruby-]1.8.6[-p420]
[ruby-]1.8.7[-p371]
[ruby-]1.9.1[-p431]
[ruby-]1.9.2[-p320]
[ruby-]1.9.3-p125
[ruby-]1.9.3-p194
[ruby-]1.9.3-p286
[ruby-]1.9.3-p327
[ruby-]1.9.3-p362
[ruby-]1.9.3-p374
[ruby-]1.9.3-p385
[ruby-]1.9.3-p392
[ruby-]1.9.3[-p429]
[ruby-]1.9.3-head
[ruby-]2.0.0-rc1
[ruby-]2.0.0-rc2
[ruby-]2.0.0-p0
[ruby-]2.0.0[-p195]
ruby-head
```

```
GoRuby
goruby
```

```
Topaz
```

### 81.4. Instalar un Intérprete

```
rvm install 1.9.2
```

```
[~/ruby/rubytesting]$ rvm list rubies
```

```
rvm rubies
```

```
jruby-1.7.3 [x86_64]
rbx-2.0.0.rc1 [x86_64]
ruby-1.8.7-p352 [i686]
ruby-1.9.2-p290 [x86_64]
ruby-1.9.3-head [x86_64]
*= ruby-1.9.3-p392 [x86_64]
 ruby-2.0.0-p0 [x86_64]
 ruby-2.0.0-p195 [x86_64]
```

```
=> - current
*= - current && default
* - default
```

## 81.5. Intérpretes Instalados

```
[~/LPPbook/booktt]$ rvm list # muestra los instalados
```

```
rvm rubies
```

```
ruby-1.8.7-p352 [i686]
```

```
ruby-1.9.2-head [x86_64]
```

```
ruby-1.9.2-p290 [x86_64]
```

```
ruby-1.9.3-head [x86_64]
```

Para usar una determinada versión del intérprete:

```
[~/LPPbook/booktt]$ rvm use 1.9.3-head
Using /Users/casiano/.rvm/gems/ruby-1.9.3-head
```

```
[~/LPPbook/booktt]$ ruby -v
ruby 1.9.3p184 (2012-04-15 revision 35335) [x86_64-darwin11.3.0]
```

Otros comandos relacionados:

```
rvm system # For system ruby, with fallback to default
rvm use jruby # For current session only
rvm use --default 1.9.3 # For current and new sessions
rvm use --ruby-version rbx # For current session and this project
```

## 81.6. Suprimir un intérprete

`rvm remove` is the preferred way of removing rubies from rvm.

By default, not only will it remove the ruby and its source files, it will also get rid of aliases, wrappers, environments and any associated binaries - in other words, it cleans up most of the install.

```
rvm remove
```

As an example, to remove rubinius 1.0.0, you could do:

```
$ rvm remove rbx-1.0.0

info: Removing /Users/sutto/.rvm/src/rbx-1.0.0-20100514...
info: Removing /Users/sutto/.rvm/rubies/rbx-1.0.0-20100514...
info: Removing rbx-1.0.0-20100514 aliases...
info: Removing rbx-1.0.0-20100514 wrappers...
info: Removing rbx-1.0.0-20100514 environments...
info: Removing rbx-1.0.0-20100514 binaries...
```

For the most basic case (e.g. you want to try clearing out a bad install), rvm uninstall literally just removes the folder under `~/.rvm/rubies`.

```
rvm uninstall VERSION_NUMBER
```

## 81.7. .rvmrc, .ruby-version y .ruby-gemset

The project `.rvmrc` files are intended to be used to setup your project's ruby environment when you switch to the project root directory.

Véase Use rvmrc or ruby-version file to set a project gemset with RVM? en StackOverFlow

If your `.rvmrc` file contains custom shell code, continue using `.rvmrc` as it allows you to include any shell code.

If your only aim is to switch Ruby versions, then use `.ruby-version` which is supported by other Ruby version switchers such as `rbenv` or `chruby`. This file also does not require trusting as it is just the name of a Ruby version and will not be executed in any way.

If you use `.ruby-version` you can include `@gemset` in the file but this will not be compatible with other switchers. To maintain compatibility use the gemset name in a separate file `.ruby-gemset` which is ignored by other tools.

For example, if you have a simple `.rvmrc`:

```
rvm use 1.9.3@my-app
```

It can be transformed to `.ruby-version`:

```
1.9.3
```

And `.ruby-gemset`:

```
my-app
```

Be sure to remove the `.rvmrc` file as it takes precedence over any other project configuration files:

```
rm .rvmrc
```

## 81.8. rvmrc

You can have a `.rvmrc` file in three different places:

- `/etc/rvmrc` sets options for the whole system;
- `~/.rvmrc` sets options for the current user;
- a `.rvmrc` file in a specific directory customizes things for that project.

The global and user `.rvmrc` files are mainly compile options for installing new Rubies. However, the project-specific `.rvmrc` files are the fun ones.

Here's how you do it:

- when you start a new project, you'll what to create a `.rvmrc` file in that directory.
- Let's say you have a rails app that needs Ruby 1.8.6 to run; you have Ruby 1.9.2 as the default, and you don't want to have to remember to switch Rubies when you're working on that project.
- So, type this command:

```
rvm --rvmrc --create 1.8.6-p334
```

This will create a heavily commented `.rvmrc` file that will change your Ruby environment when you switch into that directory.

It gets even better with gemsets.

- I use Ruby 1.9.2 for all my projects, but I like to have a different gemset for each project. So, if I've got a project that's called `superproj`, I can do this:

```
rvm --rvmrc --create 1.9.2@superproj
```

Just add an at-sign after the Ruby name and follow that with your gemset name.

- This will load the right gems for that project. If this gemset hasn't been created yet, RVM will do that for you, too.

Sigue un ejemplo. Creo un `.rvmrc` en un directorio contenido el proyecto de un alumno:

```
[~/eleazar-src/prueba-activedirectory(testeo)]$ rvm --rvmrc --create 1.9.3-p392@eleazar-pfc
```

Veamos la cabecera del `.rvmrc`:

```
[~/eleazar-src/prueba-activedirectory(testeo)]$ head -3 .rvmrc
#!/usr/bin/env bash
```

```
This is an RVM Project .rvmrc file, used to automatically load the ruby
```

Ahora si entro en el directorio, la primera vez me lanza un warning:

```
[~/eleazar-src]$ rvm use 2.0.0-p0
Using /Users/casiano/.rvm/gems/ruby-2.0.0-p0
[~/eleazar-src]$ ruby -v
ruby 2.0.0p0 (2013-02-24 revision 39474) [x86_64-darwin11.4.2]
[~/eleazar-src]$ cd prueba-activedirectory/
You are using '.rvmrc', it requires trusting, it is slower and
it is not compatible with other ruby managers,
you can switch to '.ruby-version' using
 'rvm rvmrc to [.]ruby-version'
* Do you wish to trust '/Users/casiano/eleazar-src/prueba-activedirectory/.rvmrc'?
y[es], n[o], v[iew], c[ancel]> y
-bash: /usr/bin/shasum: Permission denied
[~/eleazar-src/prueba-activedirectory(testeo)]$ ruby -v
ruby 1.9.3p392 (2013-02-22 revision 39386) [x86_64-darwin11.4.2]
[~/eleazar-src/prueba-activedirectory(testeo)]$
```

Para hacer que el mensaje de advertencia no aparezca:

```
$ rvm rvmrc warning ignore /Users/casiano/eleazar-src/prueba-activedirectory/.rvmrc
Using /Users/casiano/.rvm/gems/ruby-2.0.0-p0
$ cd prueba-activedirectory/
-bash: /usr/bin/shasum: Permission denied
$ ruby --version
ruby 1.9.3p392 (2013-02-22 revision 39386) [x86_64-darwin11.4.2]
```

Véanse:

- <https://rvm.io/workflow/rvmrc/>
- [NetTuts+: Why You Should Use RVM. Andrew Burgess on Apr 12th 2011 with 20 Comments](#)

## 81.9. Gemsets

Las gemas constituyen la forma típica de distribución de códigos Ruby. Puesto que a menudo diferentes versiones de una misma gema entran en conflicto, es conveniente tener espacios de trabajo separados que contienen diferentes conjuntos de gemas. A estos espacios rvm los denomina *gemsets*.

```
[~]$ rvm gemset list
```

```
gemsets for ruby-2.0.0-p0 (found in /Users/casiano/.rvm/gems/ruby-2.0.0-p0)
 (default)
 global
=> rails3tutorial2ndEd
```

RVM by default allows creating multiple environments for one ruby - called *gemsets*.

Gemsets can be specified together with ruby name using gemsets separator(@):

```
ruby-1.9.3-p125@my-project
```

During installation of Ruby, RVM creates two gemsets:

- **default** - automatically selected when no @gemset specified: `rvm use 1.9.3`
- **global** - super gemset, inherited by all other gemsets for the given ruby

A little bit about where the default and global gemsets differ:

- If you don't use a gemset at all, you get the gems in the **default** set. **default** is the gemset used without selecting one for a specific installed ruby.
- If you use a specific gemset (say **@testing**), it will inherit gems from that ruby's **@global**. The **global** gemset is to allow you to share gems to all your gemsets.

## 81.10. Lista de Comandos para trabajar con gemsets

<code>rvm use 1.8.7</code>	# use the ruby to manage gemsets for
<code>rvm gemset create project_name</code>	# create a gemset
<code>rvm gemset use project_name</code>	# use a gemset in this ruby
<code>rvm gemset list</code>	# list gemsets in this ruby
<code>rvm gemset delete project_name</code>	# delete a gemset
<code>rvm 1.9.1@other_project_name</code>	# use another ruby and gemset
<code>rvm 1.9.3@_project --create --rvmrc</code>	# use and create gemset & .rvmrc

## 81.11. Ejemplo de uso de gemsets

To illustrate the point, let's talk about a common use case. Assume you are testing out a rails application against a new Rails release. RVM makes such testing very easy, by letting you quickly switch between multiple Rails versions. First, let's set up the environments:

```
$ rvm 1.9.2-head
$ gem install rails -v 2.3.3

$ rvm gemset create rails222 rails126
Gemset 'rails222' created.
Gemset 'rails126' created.
```

```
$ rvm 1.9.2-head@rails222
$ gem install rails -v 2.2.2

$ rvm 1.9.2-head@rails126
$ gem install rails -v 1.2.6

$ rvm 1.8.7
$ gem install rails -v 1.2.3
```

Note that, for each of the ruby installs above, you can have completely separate versions!

Now that your environments are set up, you can simply switch between Rails versions and Ruby versions as follows.

```
$ rvm 1.9.2-head@rails126 ; rails --version
```

Rails 1.2.6

```
$ rvm 1.8.7 ; rails --version
```

Rails 1.2.3

```
$ rvm 1.9.2-head@rails220 ; rails --version
```

Rails 2.2.0

```
$ rvm 1.9.2-head ; rails --version
```

Rails 2.3.3

If you are deploying to a server, or you do not want to wait around for `rdoc` and `ri` to install for each gem, you can disable them for gem installs and updates. Just add these two lines to your `~/.gemrc` or `/etc/gemrc`:

```
install: --no-rdoc --no-ri
update: --no-rdoc --no-ri
```

## 81.12. RVM gives you a separate gem directory for each and every Ruby version and gemset

This means that gems must be explicitly installed for each revision and gemset.

```
[~/rvm]$ rvm list

rvm rubies

jruby-1.7.3 [x86_64]
ruby-1.8.7-p352 [i686]
ruby-1.9.2-head [x86_64]
ruby-1.9.2-p290 [x86_64]
ruby-1.9.3-head [x86_64]
=> ruby-1.9.3-p392 [x86_64]
* ruby-2.0.0-p0 [x86_64]

=> - current
```

```

*= - current && default
* - default

[~/rvm]$ rvm gemset list

gemsets for ruby-1.9.3-p392 (found in /Users/casiano/.rvm/gems/ruby-1.9.3-p392)
=> (default)
 eleazar-pfc
 global
 rails3tutorial2ndEd

[~/rvm]$ cat ~/.rvm/environments/ruby-1.9.3-p392@eleazar-pfc
export PATH ; PATH="/Users/casiano/.rvm/gems/ruby-1.9.3-p392@eleazar-pfc/bin:
 /Users/casiano/.rvm/gems/ruby-1.9.3-p392@global/bin:
 /Users/casiano/.rvm/rubies/ruby-1.9.3-p392/bin:
 /Users/casiano/.rvm/bin:$PATH"
export rvm_env_string ; rvm_env_string='ruby-1.9.3-p392@eleazar-pfc'
export rvm_path ; rvm_path='/Users/casiano/.rvm'
export rvm_ruby_string ; rvm_ruby_string='ruby-1.9.3-p392'
export rvm_gemset_name ; rvm_gemset_name='eleazar-pfc'
export RUBY_VERSION ; RUBY_VERSION='ruby-1.9.3-p392'
export GEM_HOME ; GEM_HOME='/Users/casiano/.rvm/gems/ruby-1.9.3-p392@eleazar-pfc'
export GEM_PATH ; GEM_PATH='/Users/casiano/.rvm/gems/ruby-1.9.3-p392@eleazar-pfc:/Users/casian
export MY_RUBY_HOME ; MY_RUBY_HOME='/Users/casiano/.rvm/rubies/ruby-1.9.3-p392'
export IRBRC ; IRBRC='/Users/casiano/.rvm/rubies/ruby-1.9.3-p392/.irbrc'
unset MAGLEV_HOME
unset RBXOPT

```

### 81.13. Desinstalar rvm

rvm implode

o bien:

rm -rf ~/.rvm

Véase How do I completely clean out all traces of RVM from my system, including for system wide installs? en rvm.io

### 81.14. Véase También

1. NetTuts+: Why You Should Use RVM. Andrew Burgess on Apr 12th 2011 with 20 Comments
2. Tutorial Ruby 2: Instalar Ruby con RVM en YouTube por DevCode.LA
3. El proyecto pik permite manejar múltiples versiones de Ruby en Windows
4. How to Install Pik and Run Commands on Git Bash on Windows

### 81.15. Práctica: RVM: Instalación y Manejo

Estudie el capítulo rvm (Ruby Version Manager) en 81. Instále rvm e instale varias versiones de Ruby. Cree un directorio-proyecto con una versión de ruby y gemset específicos

# Capítulo 82

## rbenv

1. rbenv
2. Moving From RVM to Rbenv
3. rbenv: A Simple, New Ruby Version Management Tool

# Capítulo 83

## RubyGems: installing, updating and removing rubygems

### 83.1. Introducción

#### Breve Introducción a RubyGems

**Instalando una gema en local** You can use `gem install --local path_to_gem/filename.gem` This will skip the usual gem repository scan that happens when you leave off `--local`.

#### Toda la información sobre una gema:`gem spec`

```
[~/Dropbox/src/ruby/sinatra/bootstrap_example(master)]$ gem spec sinatra
--- !ruby/object:Gem::Specification
name: sinatra
version: !ruby/object:Gem::Version
 version: 1.4.2
platform: ruby
authors:
- Blake Mizerany
- Ryan Tomayko
- Simon Rozet
- Konstantin Haase
autorequire:
bindir: bin
cert_chain: []
date: 2013-03-21 00:00:00.000000000 Z
dependencies:
- !ruby/object:Gem::Dependency
 name: rack
 requirement: !ruby/object:Gem::Requirement
 requirements:
 - - '>='
 - !ruby/object:Gem::Version
 version: 1.5.2
 - - '^>'
 - !ruby/object:Gem::Version
 version: '1.5'
type: :runtime
prerelease: false
version_requirements: !ruby/object:Gem::Requirement
```

```

requirements:
- - '>='
 - !ruby/object:Gem::Version
 version: 1.5.2
- - '^>'
 - !ruby/object:Gem::Version
 version: '1.5'
- !ruby/object:Gem::Dependency
 name: tilt
 requirement: !ruby/object:Gem::Requirement
 requirements:
 - - '>='
 - !ruby/object:Gem::Version
 version: 1.3.4
 - - '^>'
 - !ruby/object:Gem::Version
 version: '1.3'
 type: :runtime
 prerelease: false
 version_requirements: !ruby/object:Gem::Requirement
 requirements:
 - - '>='
 - !ruby/object:Gem::Version
 version: 1.3.4
 - - '^>'
 - !ruby/object:Gem::Version
 version: '1.3'
- !ruby/object:Gem::Dependency
 name: rack-protection
 requirement: !ruby/object:Gem::Requirement
 requirements:
 - - '^>'
 - !ruby/object:Gem::Version
 version: '1.4'
 type: :runtime
 prerelease: false
 version_requirements: !ruby/object:Gem::Requirement
 requirements:
 - - '^>'
 - !ruby/object:Gem::Version
 version: '1.4'
description: Sinatra is a DSL for quickly creating web applications in Ruby with minimal effort.
email: sinatrarb@googlegroups.com
executables: []
extensions: []
extra_rdoc_files:
- README.de.md
- README.es.md
- README.fr.md
- README.hu.md
- README.jp.md
- README.ko.md

```

```

- README.md
- README.pt-br.md
- README.pt-pt.md
- README.ru.md
- README.zh.md
- LICENSE
files:
- README.de.md
- README.es.md
- README.fr.md
- README.hu.md
- README.jp.md
- README.ko.md
- README.md
- README.pt-br.md
- README.pt-pt.md
- README.ru.md
- README.zh.md
- LICENSE
homepage: http://www.sinatrarb.com/
licenses: []
metadata: {}
post_install_message:
rdoc_options:
- --line-numbers
- --inline-source
- --title
- Sinatra
- --main
- README.rdoc
- --encoding=UTF-8
require_paths:
- lib
required_ruby_version: !ruby/object:Gem::Requirement
 requirements:
 - - '>='
 - !ruby/object:Gem::Version
 version: '0'
required_rubygems_version: !ruby/object:Gem::Requirement
 requirements:
 - - '>='
 - !ruby/object:Gem::Version
 version: '0'
requirements: []
rubyforge_project:
rubygems_version: 2.0.3
signing_key:
specification_version: 3
summary: Classy web-development dressed in a DSL
test_files: []

```

## Véase También

- Ruby for Newbies: Working with Gems

- Artículo de Linux Journal sobre RubyGems
- RubyGems.org home page
- docs.rubygems.org home page
- help.rubygems.org home page
- Véase RubyGems User Guide.

## Seguridad

1. module Gem::Security

## 83.2. Servidores de Gemas

### 83.2.1. Geminabox

Geminabox lets you host your own gems, and push new gems to it just like with rubygems.org. The bundler dependencies API is supported out of the box. Authentication is left up to either the web server, or the Rack stack. For basic auth, try Rack::Auth.

# Capítulo 84

## ssh

### 84.1. Introducción

#### Definición

SSH es un protocolo de red y un conjunto de estándares que permiten una conexión encriptada entre dos computadoras. Usa criptografía de clave pública para autenticar al computador y al usuario. Por defecto suele usar el puerto TCP 22.

SSH se usa para acceder a una máquina remota y ejecutar comandos en una máquina remota. También sirve para proteger conexiones inseguras y transferir ficheros de manera segura. SSH utiliza un modelo cliente-servidor. Un programa cliente (normalmente denominado `ssh`) es utilizado para establecer una conexión a otro programa que da el servicio y que se ejecuta en la máquina remota (normalmente denominado `sshd` o SSH daemon).

SSH soporta autenticación basada en RSA y DSA. RSA fué el primer algoritmo publicado que permite el cifrado y la firma digital. Se cree que es seguro si las claves son lo suficientemente largas. Se usa aún en comercio electrónico. La alternativa más usada es DSA cuyas iniciales corresponden a *Digital Signature Algorithm*. DSA es propiedad del gobierno de los Estados Unidos de América.

Hay criptosistemas -a los que RSA pertenece - en los cuales el cifrado y desencriptado se hace utilizando claves separadas y no es posible derivar la clave de desencriptado del conocimiento de la clave de encriptado. El cliente utiliza un par de claves pública/privada para la autenticación. El servidor sólo conoce la clave pública. Funciona como una pareja llave/cerradura en la que el conocimiento de la cerradura no permite deducir la forma de la llave.

#### Historia

- Tatu Ylönen diseña y publica la primera versión del protocolo SSH (SSH-1)
- Ylönen funda *SSH Communications Security* a finales de 1995. La versión original - que usaba software libre - evoluciona a software propietario
- Aparece en 1996 el protocolo SSH-2, que es incompatible con SSH-1. Se mejora la seguridad (vía el algoritmo Diffie-Hellman para el intercambio de claves) y la comprobación de la integridad vía mensajes de autenticación. Además permite ejecutar un número arbitrario de sesiones shell sobre una única conexión SSH.
- En 1998 se encuentra una vulnerabilidad en SSH 1.5. El fallo ha sido resuelto en la mayoría de las implementaciones
- En 1999 comienza el desarrollo de lo que será la primera versión abierta de SSH: OpenSSH
- En 2005 OpenSSH se convierte en la implementation ssh más popular
- El protocolo SSH-2 protocol es propuesto como Internet standard en 2006
- En el 2008 se descubre una vulnerabilidad en el cifrado en SSH-2: Fue resuelta en OpenSSH 5.2.

## Arquitectura

El protocolo SSH-2 tiene una arquitectura de capas. Las capas son:

- La *capa de transporte*. Esta capa maneja el intercambio inicial de claves y la autentificación del servidor y establece los métodos de cifrado, compresión y verificación de la integridad. Deja visible a la capa superior una interfaz para el envío y recepción de paquetes de texto plano de hasta 32 768 bytes cada uno. La capa de transporte también organiza el reintercambio de claves - normalmente después de que haya pasado una hora o se haya transferido más de 1 GB de datos.
- La *capa de autentificación*. Maneja la autentificación del cliente y proporciona un conjunto de métodos de autentificación. Entre los métodos de autentificación están:
  - password
  - publickey: para el uso de parejas DSA o RSA
  - keyboard-interactive
  - GSSAPI: permite el uso de mecanismos externos como Kerberos 5

El proceso de autentificación es dirigido por el cliente.

- La *capa de conexión*. Esta capa define el concepto de canal, peticiones de canal y peticiones globales para el uso de los servicios proporcionados por SSH. Una única sesión SSH puede alojar simultáneamente múltiples canales. Los canales transfieren datos en ambas direcciones. Las peticiones de canal son usadas para pasar datos específicos fuera-del-canal como el cambio de tamaño de una ventana de terminal o el código de salida de un proceso que se ejecuta en el servidor. El cliente SSH usa una petición global para solicitar la redirección de un puerto del lado del servidor. Entre los tipos estandar de canal se encuentran:
  - canal shell: para las shell de terminales y peticones SFTP y exec (incluyendo transferencias vía SCP)
  - canal direct-tcpip: para la redirección de conexiones cliente-servidor
  - canal forwarded-tcpip: para la redirección de conexiones servidor-clientes
- Los registros SSHFP DNS proporcionan las huellas (fingerprints) para las claves públicas de las máquinas para ser usadas en el proceso de autentificación de las máquinas.

## 84.2. Conexión SSH a Una máquina por Primera Vez

SSH utiliza cifrado de clave pública para autenticar la máquina remota y permitir a la máquina remota la autentificación del usuario.

### El Fichero .ssh/known\_hosts

En SSH1 y OpenSSH las claves de máquinas se mantienen en `etc/ssh_knownhosts` y en el directorio del usuario `~/.ssh/known_hosts`. Es posible cambiar la ubicación de estos ficheros usando en el fichero de configuración `~/.ssh/config` la opción `UserKnownHostsFile`.

Al establecer por primera vez una conexión con `ssh` o `sftp` se nos avisa de la posibilidad de que la máquina no sea quien aparenta:

```
nereida:~> sftp user@machine.ull.es
Connecting to machine.ull.es...
The authenticity of host 'machine.ull.es (193.312.112.190)' can't be established.
RSA key fingerprint is a4:1e:f1:35:0d:b1:6c:7d:5c:3e:86:ef:4c:17:8a:a9.
Are you sure you want to continue connecting (yes/no)? yes
```

Si respondemos `yes` la conexión continúa:

```
Warning: Permanently added 'machine.ull.es' (RSA) to the list of known hosts.
user@machine.ull.es's password:
sftp> bye
```

### Algoritmo de Autentificación de las Máquinas

Esta precaución se toma por si el adversario subvierte el servidor de nombre para que `machine.ull.es` sea resuelta a su máquina. A continuación puede hacer un forward de la conexión solicitada a la verdadera máquina `machine.ull.es`. Este es un caso de *man-in-the-middle-attack*.

- Cuando un cliente SSH hace una conexión cada una de las máquinas prueba su identidad a la otra utilizando criptografía de clave pública.
- Cada servidor SSH dispone de un único ID secreto que se denomina *host key* que utiliza para identificarse ante los clientes.
- La primera vez que nos conectamos a una máquina remota el cliente obtiene y almacena la parte pública de la *host key* en su fichero `~/.ssh/known_hosts`.
- Cada vez que nos conectamos con esa máquina el cliente comprueba la identidad del servidor remoto utilizando su correspondiente clave pública.

El cliente `ssh` mantiene en `~/.ssh/known_hosts` una base de datos conteniendo las máquinas a las que el usuario se ha conectado. Cualquier nuevo hosts es añadido al fichero del usuario:

```
nereida:~> grep 'machine.ull.es' ~/.ssh/known_hosts
machine.ull.es ssh-rsa KEY.....
```

### Borrando una Entrada de `.ssh/known_hosts`

Si la relación entre la IP de la máquina y su nombre cambian `ssh` nos avisa. Si confiamos en el cambio podemos borrar la entrada en el fichero `~/.ssh/known_hosts`. De otro modo `ssh` se negará a efectuar la conexión. Podemos borrarla manualmente o con el comando:

```
casiano@cc116:~$ ssh-keygen -R rala
/home/casiano/.ssh/known_hosts updated.
Original contents retained as /home/casiano/.ssh/known_hosts.old
```

### Hashing

Supongamos que el atacante ha descubierto nuestra clave y ha entrado en nuestro servidor: ahora puede hacer `cat ~/.ssh/known_hosts` y deducir que otros servidores solemos visitar. Es incluso probable que la clave sea la misma o que tengamos un sistema de autentificación sin passphrase.

Si se quiere proteger el fichero `known_hosts` se puede usar la opción `-H` de `ssh-keygen`:

```
casiano@cc116:~/.ssh$ ssh-keygen -H
/home/casiano/.ssh/known_hosts updated.
Original contents retained as /home/casiano/.ssh/known_hosts.old
WARNING: /home/casiano/.ssh/known_hosts.old contains unhashed entries
Delete this file to ensure privacy of hostnames
```

Este comando reemplaza los nombres de los servidores y las direcciones con representaciones hash. Estas versiones hash pueden ser usadas por `ssh` pero son ilegibles para un humano. Es posible usar un fichero con una combinación de entradas hash y entradas legibles.

La opción `HashKnownHosts` permite en los ficheros de configuración permitir indicarle a `ssh` que debe hacer hashing sobre los nombres y las direcciones en `~/.ssh/known_hosts`. Esto no afecta a las ya existentes en el fichero.

Es posible asociar varios nombres/direcciones con la misma clave (o repetirlas):

```
foo1,foo,192.168.1.1,192.168.1.10 ssh-rsa AAAAB3Nsomething[...]
foo2,foo,192.168.1.2,192.168.1.10 ssh-rsa AAAAB3Nsomethingelse[...]
```

## Búsqueda de Entradas Hash

Es posible encontrar las entradas en un fichero hash con la opción -F

```
casiano@cc116:~/.ssh$ ssh-keygen -F millo
Host millo found: line 10 type RSA
|1|BXG98oZImA1C02AQMAmuDb0jadQ=...
```

## Ejercicios con known\_hosts

**Ejercicio 84.2.1.** Elimine una entrada en el fichero si existe e intente la conexión ¿que ocurre?

**Ejercicio 84.2.2.** Pase a modo hashed su fichero

**Ejercicio 84.2.3.** Busque por una entrada en el fichero hashed

## 84.3. Claves Pública y Privada: Estableciendo Autentificación No Interactiva

### ssh-keygen

Recordemos como se establece una autentificación por clave pública-privada. La generación de una pareja de claves pública-privada se realiza ejecutando en el cliente ssh-keygen:

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx user@machine
```

En esta ocasión no indicamos passphrase.

Las claves generadas por ssh-keygen se almacenan por defecto en `~/.ssh/`, quedando el directorio así:

```
$ ls -l
total 12
-rw----- 1 user user 883 2005-08-13 14:16 id_rsa
-rw-r--r-- 1 user user 223 2005-08-13 14:16 id_rsa.pub
-rw-r--r-- 1 user user 1344 2005-08-04 02:14 known_hosts
```

Los ficheros `id_rsa` e `id_rsa.pub` contienen respectivamente las claves privada y pública. El fichero `known_hosts` contiene la lista de las claves públicas de las máquinas reconocidas.

Estas son algunas de las opciones que admite ssh-keygen:

- `-b 2048` establece el número de bits en la clave a 2048. Por defecto es 1024.
- `-f mykey` establece el nombre del fichero de clave: `mykey` y `mykey.pub`. Si se omite, pregunta
- `-N passphrase` la passphrase a utilizar. Si se omite, pregunta
- `-C comentario` el comentario a añadir en la correspondiente línea de la clave pública. Si se omite es `username@host`:

```
ssh-dss AAAAB3N... pepe@machinon
```

- La opción `-p` puede ser usada para cambiar la passphrase de una determinada identidad:

```
pp2@europa:~/.ssh$ ssh-keygen -p -f id_dsa
Key has comment 'id_dsa'
Enter new passphrase (empty for no passphrase):
```

o bien:

```
ssh-keygen -p -f mykeyfile -P mysecretpassword -N mynewsecretpassword
```

- Es posible cambiar el comentario para una clave utilizando `-c`:

```
ssh-keygen -c -f mykeyfile -P mysecretpassword -C 'my new comment'
```

### Computando la Huella de una Máquina

Es posible generar el `fingerprint` de una clave pública usando la opción `-l`:

```
$ ssh-keygen -l -f mykeyfile.pub
```

Ya vimos que cuando nos conectábamos a un host por primera vez via ssh obteníamos un mensaje como:

```
teide:~$ ssh teide
The authenticity of host 'teide (134.2.14.48)' can't be established.
RSA key fingerprint is 9e:1a:5e:27:16:4d:2a:13:90:2c:64:41:bd:25:fd:35.
Are you sure you want to continue connecting (yes/no)?
```

Normalmente respondemos `yes` y entramos nuestra clave. ¿Cómo podemos comprobar que el fingerprint es auténtico?

¿Dónde están los ficheros de claves del servidor? Normalmente en el directorio `/etc/ssh/`.

```
teide:~$ ls /etc/ssh/*key*
/etc/ssh/ssh_host_dsa_key /etc/ssh/ssh_host_key.pub
/etc/ssh/ssh_host_dsa_key.pub /etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_key /etc/ssh/ssh_host_rsa_key.pub
```

Este servidor tiene tres claves. Sólo el root puede leer las privadas, pero las públicas tienen permisos de lectura. En el mensaje anterior, el cliente ssh muestra el fingerprint esperado. Podemos obtenerlo con:

```
teide:~$ ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
2048 9e:1a:5e:27:16:4d:2a:13:90:2c:64:41:bd:25:fd:35 /etc/ssh/ssh_host_rsa_key.pub
```

El primer número (2048) es la longitud en bits de la clave. El segundo `9e:1a:...:fd:35` es la huella, el último `/etc/ssh/ssh_host_rsa_key.pub` es el nombre del fichero con la clave pública.

#### Ejercicio 84.3.1. Este método es inseguro. ¿Por qué?

*El método seguro es contactar al administrador para obtener la huella (se supone que el administrador la guardó en lugar seguro) en vez de utilizar un canal que pueda estar comprometido.*

## ssh-copy-id

Ahora se debe copiar la clave pública al servidor, al fichero `~/.ssh/authorized_keys`. El fichero `authorized_keys` es el fichero que contiene las claves públicas utilizadas durante el proceso de autentificación pública.

Para la copia se utiliza el comando `ssh-copy-id`:

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub user@machine1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub user@machine2
```

`ssh-copy-id` es un script que se conecta a la máquina y copia el archivo (indicado por la opción `-i`) en `~/.ssh/authorized_keys`, y ajusta los permisos de forma adecuada:

- El home del usuario en la máquina remota
- El directorio `~/.ssh`
- El fichero `~/.ssh/authorized_keys`

Una configuración de permisos inadecuada puede prevenir que podamos acceder a la máquina, sobre todo si la configuración del servicio SSH tiene activado `StrictModes` (que suele ser la opción por defecto):

```
root@server:/etc/ssh# grep -i strictmode *
sshd_config:StrictModes yes
```

Si se desea publicar la clave en un cierto número de máquinas que comparten la misma clave es posible (pero no recomendable) hacerlo usando el programa `sshpass` con un comando como este:

```
$ for i in host1 host2 ... hostN; do sshpass -pmipassword ssh-copy-id -i .ssh/identity $i; done
```

## Copia Manual

Si no se dispone del programa `ssh-copy-id` se puede realizar una copia manual a la máquina remota del fichero conteniendo la clave pública (por ejemplo usando `scp` o `sftp`) y añadir su contenido al fichero `~/.ssh/authorized_keys`.

En la máquina servidor añadimos una línea en el fichero `~/.ssh/authorized_keys` que contiene exactamente la única línea que estaba en el fichero en el que se guardó la clave pública (`id_rsa.pub` o `filename.pub` o como se llamara). Después de eso el fichero `.ssh/authorized_keys` tendría una línea parecida a esta:

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAybmcqaU/Xos/GhYCzkV+kDsK8+A50jaK
5WgLMqmu38aPo560d10RQ3EiB42DjRVY8trXS1NH4jbURQPERR2LHCCYq6tHJYfJNhUX
/C0wHs+ozNPE83CYDhK4AhabahnltFE5ZbefwXW4FoK00+n8AdDfSX0azpPas8jXi5bE
wNf7heZT++a/Qxbu9JHF1huThuDux0tIWl07G+tKqzggFVknM5CoJCFxaik911NGgu20
TKfY94c/ieETOXE5L+fVrbt0h7DTFMjIYAWNxy4t1MR/59UVw5dapAxH9J21Zglkj0w0
LwFI+7hZu9XvNfMKMKg+ERAz9XHYH3608RL1RQ== Este comentario describe la
clave
```

excepto que he descompuesto la línea en varias líneas para hacerla mas legible.

La forma usual de completar el proceso requeriría de dos autentificaciones: una para la copia con `scp` y otra para añadir la identidad en el fichero `.ssh/authorized_keys` y asegurarse de los permisos.

Una forma de realizar el proceso con una sola autentificación es la siguiente:

```
europa:~/.ssh$ cat prueba.pub | \
ssh nereida "umask u=rwx,g=,o=; test -d .ssh || mkdir .ssh ; cat >> .ssh/authorized_keys"
```

La máscara por defecto `umask u=rwx,g=,o=` se ha establecido de manera que los ficheros creados tengan permisos `rw-----` y los nuevos directorios tengan permisos `rwx-----` (véase la entrada de `umask` en la Wikipedia).

## Conexión sin Clave

Ahora la conexión debería funcionar sin necesidad de introducir la clave.

```
ssh -i ~/.ssh/filename remotehost
```

o bien:

```
slogin -i ~/.ssh/filename remotehost
```

Una vez que se ha establecido un esquema de autentificación automática es trivial ejecutar un comando en la máquina remota:

```
pp2@nereida:/tmp$ ssh remotename@machine uname -a
Linux machine 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

## Permisos en .ssh

Si no funciona es posible que sea un problema de permisos en los ficheros. Los permisos correctos deben ser similares a estos:

```
$ chmod go-w $HOME $HOME/.ssh
$ chmod 600 $HOME/.ssh/authorized_keys
```

Inténtelo de nuevo usando la opción `-v` de `ssh`:

```
pp2@nereida:/tmp$ ssh -v -v remotename@machine uname -a
```

## Véase también

- OpenSSH FAQ (Frequently asked questions)

## Ejercicios

**Ejercicio 84.3.2.** Cree una identidad y transfírala manualmente.

**Ejercicio 84.3.3.** Cámbiele la passphrase a una identidad

**Ejercicio 84.3.4.** Cámbiele el comentario a una identidad

**Ejercicio 84.3.5.** Copie el fichero de clave privada de su cuenta de usuario `u1` en `m1` a otra cuenta `u2` en la máquina `m1`. Intente ahora conectarse siendo el usuario `u2` de `m1` a la máquina `rm` en la que situó la clave pública como usuario `ru`:

```
u2@m1:~/
```

¿Sigue funcionando la autentificación automática?

**Ejercicio 84.3.6.** Copie ahora el fichero de clave privada de su cuenta de usuario `u1` en `m1` a otra cuenta `w1` en una máquina distinta `mw`. Conéctese desde `mw` a `rm` en la que situó la clave pública como usuario `ru`:

```
mw@w1:~/
```

¿Funciona? ¿Cómo afectan los resultados a su percepción de la seguridad de las comunicaciones con `ssh`?

**Ejercicio 84.3.7.** Considere una intranet en la que su `HOME` de usuario está en un sistema de archivos compartido. Genere (si no la ha echo ya) su pareja de claves privada y pública. Publique la clave usando `ssh-copy-id` o bien - si `ssh-copy-id` no está disponible - copiando la clave pública en el fichero `authorized_keys`. ¿Se obtiene autorización automática entre dos máquinas cualesquiera de la red?

## 84.4. El fichero authorized\_keys

Cada línea del fichero SSH1 `authorized_keys` (habitualmente en `~/.ssh/authorized_keys`) contiene una clave pública. Tiene el siguiente formato: Cada entrada va en una sola línea. Los campos se separan por comas. Las líneas en blanco y las que empiezan por # se ignoran.

- Primero van las opciones, separadas por comas, por ejemplo:

- `from="pattern-list"` El servidor `sshd` puede limitar que máquinas pueden conectarse a la máquina usando wrappers TCP, pero no a nivel de usuario. Utilizando esta opción es posible limitar las conexiones para una identidad a un conjunto específico de máquinas. Los hosts se separan por comas y pueden contener comodines como \* y ?. Se pueden rechazar hosts específicos prefijándolos con !. Por ejemplo:

```
from="!enemy.my_isp.net,*.my_isp.net,home.example.com"
```

- `command="command"`
- `environment="NAME=value"` Se usa si se quiere tener un entorno específico cuando se usa esta clave. Por ejemplo podemos poner un `$PATH` restrictivo para que no pueda ejecutar ciertos comandos, cambiar `$HOME`, etc.

Si se tiene varias personas accediendo vía diferentes identidades a la misma cuenta - usando tal vez un comando forzado - puede ser útil establecer la variable `$LOGNAME` de manera que identifique quién se conecta.

- `notty`

Este campo es opcional. Su presencia se detecta viendo si la línea comienza por un número o no

- La clave pública: bits, exponente y módulo
- Un comentario (`user@machine.domain.es` en el ejemplo que sigue)

Sigue un ejemplo. Primero generemos una pareja especial para esto:

```
$ ssh-keygen -tdsa -C 'limited command to millo' -P '' -f ~/.ssh/limited_command
$ ssh-copy-id -i ~/.ssh/limited_command.pub millo
```

Ahora podemos entrar a la máquina usando esa pareja con:

```
$ ssh -i ~/.ssh/limited_command millo
Last login: Mon Apr 20 09:05:11 2009 from 192.168.100.242
millo:~$ vi ~/.ssh/authorized_keys
millo:~$
```

Editamos el fichero `~/.ssh/authorized_keys` y añadimos la opción comando

```
17 # Prueba de comando limitado
18 command="ls" ssh-dss AAAAB3NzaC1kc3...= limited command to millo
```

La presencia de la opción `command` permite dar acceso a nuestra cuenta a usuarios restringiendo el uso de los comandos que pueden utilizar.

```
tonga:~/.ssh$ ssh -i ~/.ssh/limited_command millo
ADVENS_OF_SHERLOCK.html cshrc Imágenes LSimple-Scope
asignas Desktop LEyapp machines.sample
autosave doc Llhp Makefile.PL
bidirwithnamedpipes.pl Documentos LPLdoc Música
bin exrc LPLsrc myprofile
cat help LPL-Tutu Net-ParSCP.tar.gz
Connection to millo closed.
```

Es posible obtener los argumentos escritos en la línea de comandos de la llamada al cliente consultando la variable de entorno `SSH_ORIGINAL_COMMAND`. Existen otras variables de entorno que están definidas cuando se ejecuta una conexión (véase la sección `ENVIRONMENT` en la página del manual de `ssh`) Para ilustrar el punto usaremos el siguiente programa como comando forzado:

```
$ cat -n get_ssh_env
1 #!/usr/bin/perl -w
2 use strict;
3
4 my @keys = qw{
5 SSH_ORIGINAL_COMMAND
6 SSH_CONNECTION
7 SSH_AUTH_SOCK
8 SSH_TTY
9 DISPLAY
10 HOME
11 LOGNAME
12 MAIL PATH TZ USER
13 };
14 my ($args, $con, $sock, $tty,
15 $display, $home, $logname, $mail, $path, $tz, $user
16) = map { $ENV{$_} || '' } @keys;
17
18 print << "INFO";
19 args=<$args>
20 con=<$con>
21 sock=<$sock>
22 tty=<$tty>
23 display=<$display>
24 home=<$home>
25 logname=<$logname>
26 mail=<$mail>
27 path=<$path>
28 tz=<$tz>
29 user=<$user>
30 INFO
```

Cuando se fuerza en `authorized_keys` y se ejecuta la conexión se obtiene:

```
casiano@tonga:~$ ssh -i ~/.ssh/show_arguments millo chum cham chim
args=<chum cham chim>
con=<XXX.XXX.XXX.XXX 55286 XXX.XXX.XXX.XXX 22>
sock=<>
tty=<>
display=<localhost:11.0>
home=</home/casiano>
logname=<casiano>
mail=</var/mail/casiano>
path=</usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/soft/perl5/lib/b
tz=<>
user=<casiano>
```

Si el administrador tiene configurada la opción `PermitUserEnvironment` de `sshd_config` a `yes` (el valor por defecto es `no`), `ssh` leerá el fichero `~/.ssh/environment` (si existe) añadiendo al entorno líneas con el formato `VARNAME=value`.

## Véase También

- <http://www.jms1.net/ssh.shtml>
- <http://troy.jdmz.net/rsync/index.html>
- <http://www.hackinglinuxexposed.com/articles/20030109.html>
- <http://www.snailbook.com/faq/restricted-scp.auto.html>

## 84.5. Deshabilitar la Asignación de una TTY

Normalmente, cuando nos conectamos via SSH-1 el servidor abre una seudoterminal. Esto no es así si se ejecuta un comando no interactivo. La variable de entorno `SSH_TTY` contiene el nombre de la terminal asignada. La asignación ocurre incluso si se ha configurado un comando forzado Por ejemplo, si en `authorized_keys` tenemos:

```
command="echo SSH_TTY is [$SSH_TTY]" ssh-rsa AAA...
```

Tenemos:

```
someone@localhost:~$ ssh -l user machine.domain.es
SSH_TTY is [/dev/pts/5]
Connection to orion closed.
someone@localhost:~$ ssh -l user machine.domain.es echo "tutu"
SSH_TTY is []
```

Use la opción `no-pty` para deshabilitar la asignación de TTY. Esto funciona incluso si el cliente utiliza la opción `-t` para requerir una TTY.

## 84.6. Agentes SSH

Un agente SSH es un programa que guarda de forma segura las frases-clave asociadas con las claves privadas y responde a consultas autenticadas de los clientes SSH. El agente recuerda la passphrase de manera que el usuario no tenga que teclearla cada vez que la clave es usada.

El agente crea un socket en un subdirectorio de `/tmp` en el que escucha las peticiones de conexión SSH. Cualquiera que tenga acceso a dicho socket puede acceder al agente. Los permisos unix para el socket son suficientemente restrictivos.

Los programas que se usan son `ssh-agent` y `ssh-add`. Utilizamos `ssh-add` para añadir claves privadas a la cache del agente ssh. A partir de ese momento el cliente ssh no nos importunará solicitando la passphrase, comunicándose con el agente para superar el proceso de autenticación. El agente `ssh-agent` actúa como un daemon que permanece dando el servicio de autenticación a los clientes SSH.

### ssh-add

El programa `ssh-add` permite añadir, listar y suprimir claves de la cache del agente:

```
someone@localhost:~/.ssh$ ssh-add ~/.ssh/conpass
Enter passphrase for /home/someone/.ssh/conpass: *****
Identity added: /home/someone/.ssh/conpass (/home/someone/.ssh/conpass)
someone@localhost:~/.ssh$ ssh-add -l
2048 da:13:....:ee /home/someone/.ssh/conpass (RSA)
```

Una vez establecida la clave en el agente es posible conectarse sin necesidad de proporcionar la frase clave:

```
someone@localhost:~/.kde/shutdown$ ssh remote.machine
Linux remote.machine 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/\*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.

```
Last login: Tue Mar 11 14:03:59 2008 from localhost.deioc.ull.es
user@remote.machine:~$
```

## Arrancando un Agente SSH

Para arrancar el agente SSH se ejecuta el comando:

```
eval `ssh-agent`
```

Esto se hace así porque el resultado returnedo por `ssh-agent` es una cadena conteniendo un script bash. Este guión establece las variables de entorno necesarias para que las futuras llamadas a clientes SSH puedan encontrar al agente. Observe el resultado de una ejecución sin `eval` ni backticks:

```
$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-jaDVTd5583/agent.5583; export SSH_AUTH_SOCK;
SSH_AGENT_PID=5584; export SSH_AGENT_PID;
echo Agent pid 5584;
```

La variable `SSH_AUTH_SOCK` contiene el camino al socket de dominio UNIX que será utilizado por los clientes `ssh`, `scp`, etc. para comunicarse con el agente.

El pid proporcionado en `SSH_AGENT_PID` nos permite eliminar al agente mediante `kill`. Sin embargo una llamada a `kill` no elimina las variables de entorno establecidas. La forma correcta de terminar el agente es usando la opción `-k` de `ssh-agent`:

```
pp2@localhost:~/Lbook$ eval `ssh-agent`
Agent pid 5606
pp2@localhost:~/Lbook$ env | grep -i ssh
SSH_AGENT_PID=5606
SSH_AUTH_SOCK=/tmp/ssh-ALUJyJ5605/agent.5605
CVS_RSH=/usr/bin/ssh
pp2@localhost:~/Lbook$ eval `ssh-agent -k`
Agent pid 5606 killed
pp2@localhost:~/Lbook$ env | grep -i ssh
CVS_RSH=/usr/bin/ssh
pp2@localhost:~/Lbook$
```

## El Protocolo

1. El usuario escribe el comando que da lugar a la conexión inicial y especifica el `username` así como la petición de usar una clave. El programa cliente envía el `username` y la petición de uso de la clave.
2. El demonio de SSH mira en el fichero `authorized_keys` y construye un reto basado en esa clave y lo envía al cliente.
3. El cliente recibe el reto asociado con la clave y lo redirige al agente. Supongamos que la clave fué previamente cargada con `ssh-add`

4. El agente construye la respuesta y la devuelve al proceso `ssh` el cual la envía al proceso `sshd` que escucha en el otro extremo. El cliente `ssh` nunca ve la clave privada, lo que ve es la respuesta.
5. El servidor `sshd` valida la respuesta. Si es correcta permite el acceso al sistema.

**Ejercicio 84.6.1.** *Establezca claves con passphrase, instale un agente ssh y compruebe su buen funcionamiento.*

### Bloqueando el acceso a un Agente

Es posible cerrar el acceso al agente usando la opción `-x` de `ssh-add`:

```
pp2@europa:~$ ssh-add -x
Enter lock password:
Again:
Agent locked.
pp2@europa:~$ ssh orion
Enter passphrase for key '/home/pp2/.ssh/id_dsa': *****
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

The programs included with the Debian GNU/Linux system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/\*/\*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

```
Last login: Mon Apr 27 13:29:32 2009 from europa.deioc.ull.es
casiano@orion:~$
```

Para abrir el acceso al agente usaremos la opción `-X` de `ssh-add`:

```
pp2@europa:~$ ssh-add -X
Enter lock password: *****
Agent unlocked.
pp2@europa:~$
```

### Poniendo Fecha de Caducidad

La opción `-t lifetime` de `ssh-add` puede ser utilizada para limitar el tiempo de caducidad de las identidades en un agente. El tiempo se expresará en segundos o bien de la forma `time[qualifier]`, donde `time` es un entero positivo y `qualifier` es:

nada	segundos
s   S	segundos
m   M	minutos
h   H	horas
d   D	días
w   W	semanas

### Véase también

- Wikipedia Ssh-agent
- Steve Friedl's Unixwiz.net Tech Tips An Illustrated Guide to SSH Agent Forwarding
- Using ssh-agent with ssh

## 84.7. Mejor un Sólo Agente

Si decidimos poner la línea eval ‘ssh-agent’ en nuestro `~/.bash_profile` (véase `.bash_profile` vs `.bashrc`) tenemos un problema: con cada sesión se lanza una nueva copia del agente ssh. Es cierto que el asunto no es preocupante si sólo abrimos una o dos cónsolas, pero si - como es mi caso - abrimos muchas en una sesión X tendremos que escribir la passphrase bastantes veces. Un único agente debería ser suficiente.

Otro problema son las tareas automáticas que usan `cron`. Las tareas arrancadas mediante `cron` no heredan las variables de entorno `SSH_AUTH_SOCK` y `SSH_AGENT_PID` y por tanto no saben que proceso `ssh-agent` deben contactar.

### ssh-agent y Sesiones X

Es común ver que en ciertos Unix el arranque de la sesión X lleva aparejado el arranque de un agente SSH. Si ese es nuestro caso, tenemos la mayor parte del trabajo hecho. Por ejemplo en una configuración típica de Kubuntu vemos lo siguiente:

```
$ cat -n /etc/X11/Xsession.options
1 # $Id: Xsession.options 189 2005-06-11 00:04:27Z branden $
2 #
3 # configuration options for /etc/X11/Xsession
4 # See Xsession.options(5) for an explanation of the available options.
5 allow-failsafe
6 allow-user-resources
7 allow-user-xsession
8 use-ssh-agent
9 use-session-dbus
```

La opción `use-ssh-agent` hace que se dispare un agente en el momento de arrancar las X. En efecto si vemos los procesos en el sistema descubrimos la presencia de un agente:

```
casiano@europa:~$ ps -fA | grep agent
casiano 6976 6911 0 10:57 ? 00:00:00 /usr/bin/ssh-agent x-session-manager
casiano 8018 7625 0 11:07 pts/3 00:00:00 grep agent
```

Viendo las variables de entorno podemos ver el lugar en el que está el socket:

```
casiano@europa:~$ env | grep -i ssh
SSH_AGENT_PID=6976
SSH_AUTH_SOCK=/tmp/ssh-qkuyC6911/agent.6911
```

Podemos ver que el socket tiene los permisos puestos para que pueda ser accedida por el usuario que arrancó las X:

```
casiano@europa:~$ ls -ld /tmp/ssh-qkuyC6911/
drwx----- 2 casiano casiano 4096 2009-04-27 10:57 /tmp/ssh-qkuyC6911/
casiano@europa:~$ ls -l /tmp/ssh-qkuyC6911/
total 0
srw----- 1 casiano casiano 0 2009-04-27 10:57 agent.6911
```

por tanto el usuario que arrancó las X puede acceder a ese agente. Veamos que claves están en ese agente:

```
rw----- 1 casiano casiano 0 2009-04-27 10:57 agent.6911
casiano@europa:~$ ssh-add -l
1024 49:4c:e1:b4:1a:ea:e6:73:fc:a1:e5:6b:54:c9:da:62 /home/casiano/.ssh/id_dsa (DSA)
```

Ya esta la clave `/home/casiano/.ssh/id_dsa`. Probablemente lo haya hecho en alguna prueba anterior. Para empezar esta nueva prueba desde una situación inicial vamos a suprimir todas las claves del agente.

```
casiano@europa:~$ ssh-add -D
All identities removed.
```

Si ahora intento conectarme a una máquina con esa clave, me es solicitada la passphrase:

```
casiano@europa:~$ ssh orion
Enter passphrase for key '/home/casiano/.ssh/id_dsa':
^C
```

Interrumpo la conexión pulsando CTRL-C y paso a añadir la clave al agente:

```
casiano@europa:~$ ssh-add /home/casiano/.ssh/id_dsa
Enter passphrase for /home/casiano/.ssh/id_dsa:
Identity added: /home/casiano/.ssh/id_dsa (/home/casiano/.ssh/id_dsa)
```

Ahora puedo hacer la conexión sin necesidad de introducir la passphrase:

```
casiano@europa:~$ ssh orion
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

## Arrancando el Agente en KDE

Una solución parcial es arrancar el agente tan pronto como se inicia la sesión del escritorio. El siguiente procedimiento funciona en un escritorio KDE:

- Cree `/.kde/env/ssh-agent.sh`

```
mkdir ~/.kde/env
vim ~/.kde/env/ssh-agent.sh
chmod u+x ~/.kde/env/ssh-agent.sh
```

con estos contenidos:

```
someone@localhost:~/.kde/env$ cat -n ssh-agent.sh
1 #!/bin/sh
2 /usr/bin/ssh-agent -s > $HOME/.ssh/agent-env.sh
3 $HOME/.ssh/agent-env.sh > /dev/null
```

La opción `-s` de `ssh-agent` le indica que debe generar comandos de Bourne shell en `stdout`. Esta es la conducta por defecto a menos que `SHELL` sea de la familia de shells `csh`. La alternativa es usar la opción `-c` la cual genera comandos C-shell. Esta es la conducta por defecto si `SHELL` es `csh` o `tcsh`.

La redirección de la línea 2 produce el guión que es ejecutado en la línea 3:

```
someone@localhost:~/src$ cat $HOME/.ssh/agent-env.sh
SSH_AUTH_SOCK=/tmp/ssh-LDdzn31114/agent.31114; export SSH_AUTH_SOCK;
SSH_AGENT_PID=31115; export SSH_AGENT_PID;
echo Agent pid 31115;
```

- Cree `~/.kde/shutdown`:

```
mkdir ~/.kde/shutdown
vim ~/.kde/shutdown/shutdown-ssh.sh
chmod u+x ~/.kde/shutdown/shutdown-ssh.sh
```

con los siguientes contenidos:

```
someone@localhost:~/.kde/shutdown$ cat -n shutdown-ssh.sh
1 #!/bin/sh
2 /usr/bin/ssh-agent -k
```

La opción recomendada es aprovechar el agente ssh creado al arrancar las X. Puede que quiera que las identidades se añadan tan pronto como entra en su escritorio KDE. En ese caso cambie los contenidos de `./kde/env/ssh-agent.sh` por algo parecido a esto:

```
someone@localhost:~/.kde/env$ cat -n ssh-agent.sh
1 #!/bin/sh
2 /usr/bin/ssh-add $HOME/.ssh/id_dsa
3 /usr/bin/ssh-add $HOME/.ssh/thatmachinebackupidentity
```

### Introducción a keychain

La metodología usada en los párrafos anteriores no es válida si no somos el usuario que arranca las X. Este es el caso si estamos ya en una sesión `ssh` o hemos cambiado de usuario ejecutando `su` (switch user) o bien queremos ejecutar un proceso por lotes bajo un `cron`.

En estos casos `keychain` puede ayudar.

### Suprimiendo los Agentes Activos con keychain

Comencemos usando la opción `-k` para suprimir los agentes activos y asegurarnos que empezamos desde una situación inicial:

```
lusasoft@LusaSoft:~/.ssh$ keychain -k all
```

```
KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL
```

```
* All lusasoft's ssh-agent(s) (5678) are now stopped
* All lusasoft's gpg-agent(s) (6823) are now stopped
```

La opción `-k` tiene la sintaxis:

```
-k which
```

Las siguientes opciones son válidas para `which`:

- `all` eliminar a todos los agentes y terminar
- `others` eliminar a los agentes que no sean de `keychain`
- `mine` eliminar a los agentes `keychain` dejando el resto.

### Funcionamiento de keychain

Cuando se ejecuta `keychain` comprueba si existe ya un agente `ssh-agent` ejecutándose. Si no es el caso arranca uno. Si existe no cargará uno nuevo. Sin embargo es necesario que el usuario o el script llamen a `~/.keychain/europa-sh` para tener acceso al agente.

```
lusasoft@LusaSoft:~/.ssh$ keychain ~/.ssh/id_dsa
```

```
KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL
```

```

* Initializing /home/lusasoft/.keychain/LusaSoft-sh file...
* Initializing /home/lusasoft/.keychain/LusaSoft-csh file...
* Initializing /home/lusasoft/.keychain/LusaSoft-fish file...
* Starting ssh-agent
* Initializing /home/lusasoft/.keychain/LusaSoft-sh-gpg file...
* Initializing /home/lusasoft/.keychain/LusaSoft-csh-gpg file...
* Initializing /home/lusasoft/.keychain/LusaSoft-fish-gpg file...
* Starting gpg-agent
* Adding 1 ssh key(s)...
Identity added: /home/lusasoft/.ssh/id_dsa (/home/lusasoft/.ssh/id_dsa)

```

Salva las variables de entorno en `~/.keychain/${HOSTNAME}-sh`,

```

lusasoft@LusaSoft:~$ tree .keychain/
.keychain/
|-- LusaSoft-csh
|-- LusaSoft-csh-gpg
|-- LusaSoft-fish
|-- LusaSoft-fish-gpg
|-- LusaSoft-sh
`-- LusaSoft-sh-gpg

```

Estos ficheros contienen los comandos para establecer las variables de entorno. Por ejemplo:

```

$ cat -n .keychain/europa-sh
 1 SSH_AUTH_SOCK=/tmp/ssh-ctTDo22823/agent.22823; export SSH_AUTH_SOCK;
 2 SSH_AGENT_PID=22824; export SSH_AGENT_PID;

```

De este modo las subsiguientes llamadas no interactivas a `ssh` - por ejemplo, cron jobs - pueden hacer un `source` del correspondiente guión y conectarse con el único agente obviando así la necesidad de solicitar las claves.

```

$ source ~/.keychain/europa-sh
$ ssh orion
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux

```

Si abrimos otra terminal deberemos hacer de nuevo la llamada a `source ~/.keychain/europa-sh`. Por supuesto, lo mejor es poner la llamada a `keychain` y a `source ~/.keychain/europa-sh` en nuestro fichero de arranque `~/.bashrc` o `bash_profile`.

```

/usr/bin/keychain ~/.ssh/id_dsa
source ~/.keychain/europa-sh

```

### Cuando el Agente no Conoce la Clave

Cuando se ejecuta, `keychain` verifica que la clave especificada es conocida por el agente. Si no es así nos solicitará la passphrase.

Véase la siguiente secuencia de comandos. Primero ponemos una passphrase a la identidad por defecto:

```

lusasoft@LusaSoft:~$ ssh-keygen -p
Enter file in which the key is (/home/lusasoft/.ssh/id_rsa): /home/lusasoft/.ssh/id_dsa
Key has comment '/home/lusasoft/.ssh/id_dsa'
Enter new passphrase (empty for no passphrase): ****
Enter same passphrase again: ****
Your identification has been saved with the new passphrase.

```

A continuación limpiamos los agentes:

```
lusasoft@LusaSoft:~$ keychain --clear

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

* Found existing ssh-agent (6927)
* Found existing gpg-agent (6951)
* ssh-agent: All identities removed.
* gpg-agent: All identities removed.
```

Cargamos de nuevo keychain con la identidad por defecto:

```
lusasoft@LusaSoft:~$ keychain ~/.ssh/id_dsa

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

* Found existing ssh-agent (6927)
* Found existing gpg-agent (6951)
* Adding 1 ssh key(s)...
Identity added: /home/lusasoft/.ssh/id_dsa (/home/lusasoft/.ssh/id_dsa)
```

Ahora se nos solicita la passphrase (por defecto mediante `ssh-askpass`). Después de escribirla la identidad es añadida al agente.

```
lusasoft@LusaSoft:~$ keychain --nogui ~/.ssh/id_dsa

KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

* Found existing ssh-agent (6927)
* Found existing gpg-agent (6951)
* Adding 1 ssh key(s)...
Enter passphrase for /home/lusasoft/.ssh/id_dsa: ****
Identity added: /home/lusasoft/.ssh/id_dsa (/home/lusasoft/.ssh/id_dsa)
```

## Tipos de Agentes

`keychain` soporta también `gpg-agent`. Por defecto arranca todos los agentes disponibles. Se puede limitar los agentes arrancados usando la opción `--agents`. Por ejemplo `--agents 'gpg,ssh'`

## Agentes en Mal Estado

La llamada a `keychain` creará un nuevo agente si el fichero `/home/pp2/.keychain/europa-sh` queda obsoleto (por ejemplo, si `SSH_AUTH_SOCK` referencia un socket que no existe o no existe un proceso con pid el referenciado en `SSH_AGENT_PID`). Véase la siguiente sesión de comandos:

```
pp2@europa:~$ ps -fA | grep agent
casiano 6976 6911 0 10:57 ? 00:00:00 /usr/bin/ssh-agent x-session-manager
pp2 22824 1 0 12:42 ? 00:00:00 ssh-agent
pp2 22848 1 0 12:42 ? 00:00:00 gpg-agent --daemon
pp2 32569 30843 0 13:06 pts/13 00:00:00 grep agent
pp2@europa:~$ kill -9 22824
pp2@europa:~$ ps -fA | grep agent
```

```

pp2 3165 30843 0 13:07 pts/13 00:00:00 grep agent
casiano 6976 6911 0 10:57 ? 00:00:00 /usr/bin/ssh-agent x-session-manager
pp2 22848 1 0 12:42 ? 00:00:00 gpg-agent --daemon
pp2@europa:~$ keychain

```

KeyChain 2.6.8; <http://www.gentoo.org/proj/en/keychain/>  
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

```

* Initializing /home/pp2/.keychain/europa-sh file...
* Initializing /home/pp2/.keychain/europa-csh file...
* Initializing /home/pp2/.keychain/europa-fish file...
* Starting ssh-agent
* Found existing gpg-agent (22848)

```

```

pp2@europa:~$ ps -fA | grep agent
pp2 3198 1 0 13:07 ? 00:00:00 ssh-agent
pp2 3225 30843 0 13:08 pts/13 00:00:00 grep agent
casiano 6976 6911 0 10:57 ? 00:00:00 /usr/bin/ssh-agent x-session-manager
pp2 22848 1 0 12:42 ? 00:00:00 gpg-agent --daemon

```

### La opción `-eval` de keychain

La opción `-eval` de `keychain` hace que `keychain` envíe a `stdout` el contenido del script. Así podemos arrancar el agente y añadirle la clave en un paso con:

```
pp2@europa:~/Lbook$ eval `keychain -eval ~/.ssh/id_dsa`
```

### Véase También

- Common threads: OpenSSH key management, Part 1: Understanding RSA/DSA authentication
- Common threads: OpenSSH key management, Part 2: Introducing ssh agent and keychain
- Common threads: OpenSSH key management, Part 3: Agent forwarding and keychain improvements
- Automate backups on Linux. No excuses: do-it-yourself, secure, distributed network backups made easy
- keychain en Gentoo

## 84.8. Redireccionado al Agente SSH

El mismo agente SSH puede servir los retos producidos por conexiones SSH indirectas, es decir conexiones SSH arrancadas desde una conexión SSH. La idea general es que el reto producido por el servidor SSH será redirigido via los servidores intermedios hasta el agente en la máquina local. El reto será resuelto por este último y la solución emprenderá el camino de vuelta hasta el último servidor en la lista de saltos.

Para habilitar *Agent forwarding* usamos la opción `-A`.

```
desktop$ ssh -A user@remotehost
```

Si se quiere habilitar agent-forwarding desde el fichero de configuración pondremos `ForwardAgent yes` en la correspondiente entrada para ese host en `~/.ssh/config`:

```
$ cat ~/.ssh/config
Host shellsserver
 ForwardAgent yes

Host management-server
 ForwardAgent yes

Host *
 ForwardAgent no
```

Obsérvese la presencia de la sección restrictiva Host \* en el fichero de configuración.  
Veamos un ejemplo:

- El cliente ssh conecta con el servidor SSH remoto, se produce la autentificación via el agente que previamente ha sido cargado con la identidad. Se le solicita al servidor agent-forwarding.

```
lusasoft@LusaSoft:~$ ssh -A orion
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

El servidor crea un socket en /tmp/ssh-XXXXXXX/agent.##### y inicializa la variable SSH\_AUTH\_SOCK. El daemon SSH abre nuestra shell y comenzamos a trabajar en el servidor.

```
Last login: Fri May 1 10:21:38 2009 from 85.155.13.48.dyn.user.ono.com
casiano@orion:~$ env | grep -i SSH
SSH_CLIENT=85.155.13.48 35107 22
SSH_TTY=/dev/pts/3
SSH_AUTH_SOCK=/tmp/ssh-ngwpX26103/agent.26103
SSH_CONNECTION=85.155.13.48 35107 193.145.105.17 22
casiano@orion:~$ ls -lR /tmp/ssh-ngwpX26103/agent.26103
srxr-xr-x 1 casiano casiano 0 2009-05-01 10:53 /tmp/ssh-ngwpX26103/agent.26103
```

En la máquina en la que estamos no existen ninguna identidades operativas:

```
casiano@orion:~$ ls -ltra .ssh
total 60
-rw-rw-r-- 1 casiano casiano 505 2009-03-30 08:53 config
-rw-r--r-- 1 casiano casiano 4960 2009-04-04 11:43 known_hosts
-rw----- 1 casiano casiano 6466 2009-04-04 11:43 authorized_keys
drwx----- 2 casiano casiano 4096 2009-05-01 10:19 .
drwxr-x--- 50 casiano casiano 12288 2009-05-01 11:13 ..
```

- Ahora decidimos conectarnos en orion via ssh a otra máquina europa. El cliente ssh observa la presencia de la variable SSH\_AUTH\_SOCK y se conecta al socket. El cliente le indica al servidor que quiere usar la misma clave que se ha usado para esta conexión.

```
casiano@orion:~$ ssh europa
Linux europa 2.6.24-23-generic #1 SMP Wed Apr 1 21:43:24 UTC 2009 x86_64
casiano@europa:~$
```

Vemos que hemos entrado sin necesidad de introducir una clave por teclado. La clave pública de la identidad que reside en la máquina inicial ha sido instalada también en europa. ¿Que ha ocurrido?

El servidor en europa prepara el reto y se lo envía al cliente. El cliente en orion recibe el reto y se lo pasa a su servidor el cual actúa - desde el punto de vista del cliente - como si fuera un agente. El servidro sshd en orion mueve el reto al cliente de la primera conexión. Este primer cliente se lo pasa al agente inicial que es quien conoce la clave privada. La solución es preparada por el agente. Ahora la solución recorre el camino de vuelta.

## Véase También

- ssh and ssh-agent por Brian Hatch
- Steve Friedl's Unixwiz.net Tech Tips An Illustrated Guide to SSH Agent Forwarding

## 84.9. Consideraciones sobre la Seguridad del Uso de Agentes

El uso de agentes previene que la passphrase o la clave tengan que viajar por la red y nos proteje contra el robo de la clave.

Sin embargo, existe la posibilidad de que alguien tenga acceso al socket usado por el agente. Cualquiera que pueda leer/escribir en ese socket tiene la posibilidad de suplantarnos. En particular el administrador de un sistema tiene siempre esa posibilidad.

Supongamos que el usuario **casiano** tiene acceso a dos máquinas: **orion** y **beowulf** a las que accede desde **europa**. Supongamos que el administrador de **orion** es malicioso e intenta obtener acceso a las máquinas de sus usuarios. Supongamos que tanto **europa** como **beowulf** son máquinas fiables. Asumimos también que la máquina **beowulf** contiene información importante y es un servidor que el usuario **casiano** tiene en su casa.

La sesión que sigue muestra como el malvado administrador de **orion** puede secuestrar al agente utilizado por **casiano** para obtener acceso a **beowulf**:

1. Supongamos que cierto usuario **casiano** abre una conexión SSH con **orion** desde la máquina **europa**, utilizando *agent forwarding*

```
casiano@europa:~$ ps -fA | grep ssh-agent
casiano 7618 1 0 Apr28 ? 00:00:00 /usr/bin/ssh-agent -s
casiano 16288 16223 0 Apr29 ? 00:00:00 /usr/bin/ssh-agent x-session-manager
casiano 16970 13271 0 17:25 pts/31 00:00:00 grep ssh-agent
pp2 28048 1 0 May02 ? 00:00:00 ssh-agent
casiano@europa:~$ ssh-add -l
1024 49:Xc:eX:bX:Xa:ea:eX:XX:fc:aX:eX:Xb:XX:cX:da:XX /home/casiano/.ssh/id_dsa (DSA)
1024 56:XX:XX:dX:XX:eX:ca:XX:Xd:Xf:Xb:XX:Xd:fX:Xc:XX /home/casiano/.ssh/orionbackupidenti
casiano@europa:~$ ssh -A orion
Linux orion 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux
```

Posteriormente el usuario se conecta a una tercera máquina **beowulf** desde su sesión en **orion**:

```
casiano@orion:~$
casiano@orion:~$ ssh beowulf
Linux beowulf 2.6.15-1-686-smp #2 SMP Mon Mar 6 15:34:50 UTC 2006 i686
```

The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/\*/\*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.

No mail.

```
Last login: Mon May 4 17:44:35 2009 from orion.pcg.ull.es
casiano@beowulf:~$
```

2. En otra sesión en **orion** el malvado administrador busca por sockets de agentes en **/tmp**:

```

root@orion:~# ls -ltr /tmp/ssh*
total 0
srwxr-xr-x 1 casiano casiano 0 2009-05-04 17:35 agent.7932
root@orion:~# ls -ldtr /tmp/ssh*
drwx----- 2 casiano casiano 4096 2009-05-04 17:35 /tmp/ssh-jrQQgf7932

```

El root puede ver que el usuario **casiano** tiene un agente arrancado.

En el caso del agente, la opción **-a** de **ssh-agent** permite definir un lugar alternativo para el socket Unix a usar:

```

lusasoft@LusaSoft:~$ eval `ssh-agent -a myagentsocket`
Agent pid 6475
lusasoft@LusaSoft:~$ ls -ltdr my*
srw----- 1 lusasoft lusasoft 0 2009-05-02 08:56 myagentsocket

```

esta opción sin embargo no afecta al socket creado por el daemon en **orion**

3. El malvado administrador vigila lo que hace el usuario **casiano**:

```

root@orion:~# ps -fu casiano
UID PID PPID C STIME TTY TIME CMD
casiano 7682 7680 0 17:16 ? 00:00:00 sshd: casiano@pts/1
casiano 7683 7682 0 17:16 pts/1 00:00:00 -bash
casiano 7932 7930 0 17:35 ? 00:00:00 sshd: casiano@pts/0
casiano 7933 7932 0 17:35 pts/0 00:00:00 -bash
casiano 7939 7933 0 17:35 pts/0 00:00:00 ssh beowulf

```

Ummmmhhh... tiene una conexión ssh a una máquina llamada **beowulf**? demasiado breve para ser el nombre real de una máquina. Debe estar definida en su fichero de configuración.

4. Bueno, el root decide usar el socket de **casiano**:

```
root@orion:~# export SSH_AUTH_SOCK=/tmp/ssh-jrQQgf7932/agent.7932
```

El administrador malvado ya puede conectarse. Usará el fichero de configuración de **casiano** (opción **-F**) para que la conexión ocurra según la configuración de ese usuario (por supuesto, antes comprueba que existe tal configuración):

5. **root@orion:~# ssh -F ~casiano/.ssh/config casiano@beowulf**  
**casiano@beowulf:~\$**

El administrador de **orion** ha conseguido acceso a una máquina en la que no tenía cuenta. Ahora podría proceder a instalar una autenticación automática en la cuenta de **caiano** en **beowulf**, de modo que le permita posteriores visitas.

## 84.10. Depuración/Debugging

La opción **-v** hace que los clientes **scp** y **ssh** proporcionen información de depuración. Observemos la siguiente copia remota:

```
pp2@nereida:~/Lbook$ scp procesos.tex orion:
procesos.tex 100% 250KB 249.5KB/s 00:01
```

La opción **-v** nos informa de la existencia de un problema:

```

pp2@nereida:~/Lbook$ scp -v procesos.tex orion:
Executing: program /usr/bin/ssh host orion, user (unspecified), command scp -v -t .
OpenSSH_4.3p2 Debian-9, OpenSSL 0.9.8c 05 Sep 2006
debug1: Reading configuration data /home/pp2/.ssh/config
debug1: Applying options for orion
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to orion [193.145.105.17] port 22.
debug1: Connection established.
debug1: identity file /home/pp2/.ssh/identity type -1
debug1: identity file /home/pp2/.ssh/id_rsa type 1
debug1: identity file /home/pp2/.ssh/id_dsa type 2
debug1: Remote protocol version 2.0, remote software version OpenSSH_4.3p2 Debian-9
debug1: match: OpenSSH_4.3p2 Debian-9 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_4.3p2 Debian-9
debug1: An invalid name was supplied
Configuration file does not specify default realm

debug1: An invalid name was supplied
A parameter was malformed
Validation error

debug1: An invalid name was supplied
Configuration file does not specify default realm

debug1: An invalid name was supplied
A parameter was malformed
Validation error

debug1: SSH2_MSG_KEXINIT sent
.....
debug1: Transferred: stdin 0, stdout 0, stderr 0 bytes in 0.3 seconds
debug1: Bytes per second: stdin 0.0, stdout 0.0, stderr 0.0
debug1: Exit status 0

```

Después de buscar en Google vemos cual es la causa del problema:

*What happens is that SSH will hang for a long period of time before providing a password prompt or performing key authentication. In either case the problem is the GSSAPI that is now being used by clients and not by most servers.*

La solución recomendada es desactivar GSSAPIAuthentication:

```

pp2@nereida:~/Lbook$ vi ~/.ssh/config
pp2@nereida:~/Lbook$ grep -i gss ~/.ssh/config
GSSAPIAuthentication no

```

Ejecutamos con -v de nuevo. El error ha desaparecido. La velocidad de transferencia a aumentado ligeramente:

```

pp2@nereida:~/Lbook$ scp procesos.tex orion:
procesos.tex 100% 253KB 252.9KB/s 00:00

```

## **Lista de Comprobaciones en Caso de Error**

1. Añada la opción `-v` el número de veces que sea necesario
2. Añada la opción de depuración al servidor; `/usr/sbin/sshd -d -p 2222`. El número 2222 se refiere aquí al puerto.
3. Observe el fichero de log del servicio SSH usando `tail -f`: `tail -f /var/log/auth.log` (en el servidor)
4. Asegúrese que el agente se está ejecutando: `ps aux|grep ssh-agent`. Si se está usando `keychain` asegúrese de que el agente, el PID y el socket están correctamente actualizados
5. Asegúrese que su clave ha sido añadida al agente SSH. Use `ssh-add -l` para comprobarlo.
6. Compruebe los permisos en sus directorios `HOME`, `~/.ssh` y en el fichero `authorized_keys`. El servidor rechazará el uso de claves públicas si esta en modo `StrictModes` on si los permisos no son suficientemente restrictivos.

### **Ejercicio 84.10.1. Establezca una conexión SSH:**

*¿Que protocolo se esta usando en la conexión?*

*¿Que identidad se esta usando?*

### **Véase también**

- Debugging SSH public key authentication problems

## **84.11. Los Ficheros de Configuración**

En vez de llenar nuestra llamada a `ssh` de parámetros lo aconsejable es configurar la llamada en uno de los ficheros de configuración (`man ssh_config`). SSH consulta las opciones en el siguiente orden:

1. Línea de comandos
2. Fichero de configuración del usuario `~/.ssh/config` Este fichero debe tener permisos de lectura/escritura para el usuario y no ser accesible al resto.
3. Fichero de configuración global `/etc/ssh/ssh_config`

### **Estructura de un Fichero de Configuración**

Para cada parámetro el primer valor encontrado será el utilizado. Los ficheros de configuración contienen secciones. Cada sección comienza con una especificación de `Host`. Las especificaciones en esa sección sólo se aplican a los máquinas (tal y como se específico en la línea de comandos) que casen con el patrón especificado en `Host`. Por ejemplo, si escribo en la línea de comandos:

```
pp2@nereida:~/Lbook$ ssh rbeo
```

Se buscará en mi fichero de configuración `~/.ssh/config` por una especificación `Host rbeo`. En efecto, en mi fichero existe una:

...

```
Host rbeo
user otheruser
Hostname localhost
```

```
Port 2048
IdentityFile /home/pp2/.ssh/ursu
```

...

La sección dice que opciones deben aplicarse cuando nos conectamos a `rbeo`:

- La opción `user` indica que debo proporcionar como usuario `otheruser`
- La opción `Hostname` dice que el nombre auténtico de la máquina es `localhost`
- La opción `Port` indica que nos debemos conectar via el puerto 2048
- La opción `IdentityFile` nos dice que debemos usar autentificación automática con fichero de identidad (que fué previamente generado con `ssh-keygen`) `/home/pp2/.ssh/ursu`

Las líneas que comienzan por el carácter `#` son tratadas como comentarios. Es posible añadir líneas en blanco para aumentar la legibilidad.

El resto de las líneas deben seguir el patrón

## PALABRACLAVE argumentos

### Palabras Clave

Algunas palabras clave importantes son:

- `Host` Se pueden usar patrones como `*` y `?`. Define la sección. Si el nombre proveído en la línea de comandos casa con él las reglas de la sección serán aplicadas.
- `BatchMode` Se desactivará la petición de `passphrase/password`. Además las opciones `ServerAliveInterval`<sup>1</sup> y `SetupTimeOut`<sup>2</sup> se establecen a 300 segundos. Esta opción es útil cuando se usan procesos por lotes o programas - por ejemplo cuando se usa `GRID::Machine` - en los que no hay un usuario presente para proporcionar una clave y es necesario detectar la caída de la red. El argumento debe ser `yes` o `no`. El valor por defecto es `no`.
- `BindAddress` Se usa para especificar la dirección de la máquina local. Sólo es útil si la máquina dispone de mas de una dirección. Esta opción no funciona si `UsePrivilegedPort`<sup>3</sup> se pone a `yes`.
- `Compression` y `CompressionLevel` Si se establece la opción `Compression` se compactarán los datos según diga `CompressionLevel` (de 1 a 9)
- `ConnectionAttempts` Especifica el número de intentos antes de abandonar. El argumento debe ser un número (por defecto 1). Puede ser útil cuando se esta usando un procesado por lotes y la conexión falla a menudo.
- `ConnectTimeout` Especifica el límite de tiempo (segundos) antes de abandonar cuando se conecta a un servidor ssh
- `HostName` El nombre real del host al que entramos. Esto nos permite tener apodos y abreviaciones. El valor usado por defecto es el proveído en la línea de comandos. Se permite poner direcciones IP numéricas.

<sup>1</sup>La opción `ServerAliveInterval` indica que hay que enviar una señal al servidor cada cierto tiempo si no se han comunicado datos durante ese intervalo

<sup>2</sup>Normalmente el cliente ssh permanece a la espera de recibir la cabecera del protocolo desde el servidor. Bajo ciertas circunstancias esto puede producir el 'cuelgue' del programa. Al establecer la opción `SetupTimeOut` el cliente abandonará si no se reciben datos en el tiempo especificado. Esta opción es específica de Debian

<sup>3</sup>La opción `UsePrivilegedPort` especifica si se debe utilizar un puerto privilegiado para la salida. Por defecto es `no`

- **IdentityFile** Especifica el fichero contenido la identidad de autentificación (RSA o DSA) a ser usado. Por defecto toma el valor `~/.ssh/identity` para la versión 1 del protocolo y `~/.ssh/id_rsa` y `~/.ssh/id_dsa` para la versión 2. Si hay un agente cargado el agente intentará también las otras identidades. Es posible especificar múltiples identidades, las cuales serán intentadas en secuencia.

- **NoHostAuthenticationForLocalhost**

Esta opción es conveniente cuando el directorio `home` es compartido entre un conjunto de máquinas (vía NFS por ejemplo). En ese caso `localhost` refiere a una máquina distinta en cada una de las máquinas del conjunto. En tal caso, las conexiones a `localhost` dan lugar a un buen número de warnings indicando que la clave ha cambiado.

- **Port** Especifica el número de puerto.

- **ServerAliveInterval** Si después del intervalo especificado en segundos no se obtiene respuesta alguna del servidor, se envía un mensaje al servidor para solicitar una respuesta del mismo. Por defecto es 0, indicando que estos mensajes no deben ser emitidos.

Es útil probar con esta opción si la conexión se desconecta después de un corto periodo de inactividad. Ciertos administradores configuran sus servidores de manera que responden muy tardíamente. Sigue un fragmento de mi configuración desde fuera de la Universidad:

```
Host cardon
user pepito
Hostname cardon.ull.es
ForwardX11 yes
ServerAliveInterval=30
```

- **ServerAliveCountMax**

Establece el número de mensajes *alive* que pueden ser enviados por el cliente `ssh` sin que este reciba respuesta desde el servidor. Si se supera este umbral el cliente abandona, finalizando la sesión. El valor por defecto es 3. (Sólo para el protocolo 2).

- **TCPKeepAlive**

Especifica si se deben enviar mensajes TCP `keepalive` al otro lado. Si se envían se podrán detectar los fallos en las máquinas remotas. El valor por defecto es `yes`, de modo que el cliente se percate de la caída del remoto.

- **User** El usuario

- **UserKnownHostsFile** especifica un fichero alternativo a `~/.ssh/known_hosts` para las claves de máquinas

## Ejemplo

Sigue un ejemplo de fichero de configuración de usuario:

```
pp2@nereida:~/.ssh$ cat -n config
1 # man ssh_config
2
3 GSSAPIAuthentication no
4
5 Host somemachine
6 user myname
7 Hostname somemachine.pcg.ull.es
8 #ForwardX11 yes
```

```

9
10 Host ursu
11 user otheruser
12 Hostname somemachine.pcg.ull.es
13 IdentityFile /home/pp2/.ssh/ursu
14 #ForwardX11 yes
15
16 Host chazam chazam.pcg.ull.es chazam.deioc.ull.es chum
17 user myname
18 # The real name of the machine
19 Hostname chazam.pcg.ull.es
20
21 # Example to be used when connecting via reverse tunneling
22 # myname@somemachine:~$ ssh -R2048:localhost:22 pp2@nereida
23 # Logic name for the machine
24 Host rbeo
25 # user in the remote machine
26 user otheruser
27 # The 'real name' of the machine after the tunnel
28 Hostname localhost
29 # Port to connect to
30 Port 2048
31 IdentityFile /home/pp2/.ssh/ursu

```

### El fichero rc

Cada vez que ocurre una conexión SSH el servidor ejecuta el script en `/etc/sshrc`. Si existe un fichero `~/.ssh/rc` (SSH1, OpenSSH) o `~/.ssh2/rc` (SSH2), será invocado. Su presencia inhibe la ejecución de `/etc/sshrc`. Se ejecuta tanto si la sesión es interactiva como si es un comando.

## 84.12. Copia Segura de un Fichero

### Transferencia de Ficheros con scp

El protocolo *SCP* implementa la transferencia de ficheros. El cliente se conecta a la máquina remota usando SSH y solicita la ejecución de un programa servidor (normalmente el mismo programa cliente `scp` es capaz de funcionar como servidor).

Para la subida de ficheros el cliente alimenta al servidor con los ficheros que deberán ser cargados, incluyendo opcionalmente los atributos básicos como permisos y fechas.

Para las descargas el cliente envía una solicitud indicando los ficheros y directorios que deben ser descargados.

Veamos un ejemplo:

```
nereida:~> scp mm.c orion:/tmp/
mm.c
```

La sintaxis de `scp` es:

```
scp [-1246BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file]
 [-l limit] [-o ssh_option] [-P port] [-S program]
 [[user@]host1:]file1 ... [[user@]host2:]file2
```

Es posible copiar entre dos máquinas remotas emitiendo la orden desde una tercera máquina:

```
portatil@user:/tmp$ scp europa:/tmp/prueba.batch nereida:/tmp/prueba.batch
```

Es posible usar comodines (wildcards), protegiéndolos de la shell local:

```
portatil@user:/tmp$ scp 'europa:/tmp/*.tex' nereida:/tmp/
```

Es posible copiar de dos máquinas remotas a una tercera:

```
pp2@europa:~/Lnet-parscp$ scp orion:Makefile.PL nereida:texput.log beowulf:
pp2@europa:~/Lnet-parscp$ ssh beowulf ls -ltr | tail -2
-rw-r--r-- 1 casiano casiano 615 mar 26 13:56 texput.log
-rw-r--r-- 1 casiano casiano 1485 mar 26 13:56 Makefile.PL
```

Hemos copiado desde la máquina orion y desde la máquina nereida diferentes ficheros. La máquina beowulf es la máquina de destino a la que fueron transferidos.

La opción **-r** permite copiar subdirectorios. La opción **-p** permite preservar permisos.

### Edición de Ficheros Remotos con vim via scp

Un comando como este edita dos ficheros en sendos tabs:

```
vi -p scp://orion/AUTOMATICBACKUPCHECKOUT /home/casiano/europabackup.log
```

El primero es un fichero `~/AUTOMATICBACKUPCHECKOUT` en una máquina remota con nombre `orion`. El segundo es un fichero local. El editor `vim` emite el comando `scp` apropiado para copiar el fichero remoto en un fichero temporal.

Durante la edición, cada vez que salve el fichero `~/AUTOMATICBACKUPCHECKOUT` de `orion`, `vim` tasladará la orden `:w` en el comando `scp` adecuado:

```
:!scp -q '/tmp/v611122/1' 'orion:AUTOMATICBACKUPCHECKOUT'
```

o bien podemos emitir una orden de edición de un nuevo fichero remoto:

```
:vsplits scp://casiano@some.machine.com/somefile.txt
```

Es incluso posible ver los contenidos de un directorio:

```
:tabedit scp://orion/pp2/
```

#### 84.12.1. Transferencia de ficheros por sftp

El protocolo *SFTP* no es el resultado de ejecutar *FTP* sobre SSH. Se trata de un nuevo protocolo para la transferencia segura de ficheros.

Si se establece autenticación no interactiva con una máquina que provee un servicio SFTP es posible usar `sftp` en modo batch:

```
pp2@mymachine:~/Lbook$ ssh-copy-id -i ~/.ssh/id_dsa.pub casiano@ftp.someplace.ull.es
25
Now try logging into the machine, with "ssh 'casiano@ftp.someplace.ull.es'", and check in:
.ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

Ahora escribimos un guión para el cliente `sftp`:

```
pp2@nereida:~/Lbook$ cat -n instituto.sftp
1 cd asignas/asignas/PRGPAR2/perlexamples
2 lcd /home/pp2/public_html/perlexamples/
3 put *
```

La opción **-b** de `sftp` nos permite hacer la transferencia de forma automática:

```
pp2@nereida:~/Lbook$ sftp -b instituto.sftp casiano@ftp.instituto.ull.es >/dev/null
```

## 84.12.2. Copias de Seguridad con rsync

### El Algoritmo

El programa `rsync` permite la sincronización de ficheros y directorios entre dos máquinas minimizando el tamaño de la trasmisión mediante el cálculo de las diferencias entre las versiones existentes en las dos localizaciones implicadas.

`rsync` opera e la siguiente forma (véase Rsync:

1. El usuario especifica las máquinas de origen y destino así como los ficheros a actualizar
2. La máquina de destino descompone el fichero objetivo en pequeños bloques de tamaño fijo y genera checksums para cada bloque
3. La máquina destino transmite sus checksums a la máquina fuente
4. La máquina fuente busca el fichero y encuentra los bloques para los cuales los checksum casan con los suyos
5. La máquina fuente genera un conjunto de instrucciones que la máquina destino puede usar para actualizar el fichero
6. La máquina fuente envía a la máquina de destino las instrucciones así como los datos de aquellas partes que no casan al destino
7. La máquina de destino utiliza las instrucciones y los nuevos datos para actualizar sus ficheros

### Mode de Uso

Normalmente se usara como en el ejemplo vía SSH:

```
rsync -aue ssh /home/user/public_html/ user@orion:/home/user/public_html/directory/
```

La opción `-a` es equivalente a `-rlptgoD`. El significado de cada una de estas opciones es:

- `-r, --recursive` Desciende recursivamente en los subdirectorios
- `-l, --links` Copia enlaces
- `-p, --perms` Preserva permisos
- `-t, --times` Preserva los tiempos
- `-g, --group` Preserva los grupos
- `-o, --owner` Preserva el propietarios (solo si es el root)
- `-D, --devices` Preserva los dispositivos (sólo si es el root)

La opción `-u, --update` permite saltarse aquellos ficheros que son mas recientes en el receptor.

La opción `-e` permite especificar que `ssh` será utilizada.

### Los módulos File::Rsync y File::RsyncP

El módulo `File::Rsync` ofrece una API de acceso a `rsync`:

```
pp2@nereida:~/LCALL$ perl -MFile::Rsync -wde 0
main:::(-e:1): 0
DB<1> $obj = File::Rsync->new({ archive => 1, rsh => '/usr/bin/ssh -l loginname', \
 'rsync-path' => '/usr/bin/rsync' })
DB<2> $obj->exec({ src => '/home/pp2/public_html/', dest => 'machine:/home/loginname/public'
DB<3> q
pp2@nereida:~/LCALL$ ssh loginname@machine 'ls -ltr | tail -3'
```

```

drwxr-xr-x 38 loginname loginname 4096 2007-03-21 07:26 pp2
drwxr-xr-x 60 loginname loginname 4096 2007-03-21 08:08 pl
drwxr-xr-x 7 loginname loginname 4096 2007-03-21 12:20 public_html
pp2@nereida:~/LCALL$ ssh loginname@machine date
mié mar 21 12:26:37 WET 2007

```

Otro módulo es `File::RsyncP`, enteramente escrito en Perl.

### Backups con rsync en un Medio Externo

Veamos como hacer una copia diaria de una máquina remota (`someserver`) a un disco USB externo en otra máquina `mylaptop`.

#### El fichero /etc/fstab

La última línea del fichero `/etc/fstab` muestra la información de montaje:

```

root@mylaptop:~# cat /etc/fstab
/etc/fstab: static file system information.
#
<file system> <mount point> <type> <options> <dump> <pass>
proc /proc proc defaults 0 0
/dev/sda1
UUID=45797091-2b2f-45b2-95c1-2bfaaea20253 / ext3 relatime,errors=remount-ro 0
/dev/sda5
UUID=83fce336-1f21-4690-9544-dc7dd988ea71 none swap sw 0 0
/dev/scd0 /media/cdrom0 udf,iso9660 user,noauto,exec,utf8 0 0
/dev/fd0 /media/floppy0 auto rw,user,noauto,exec,utf8 0 0
/dev/sdb1 /media/PORTABLE ext3 rw,nosuid,nodev,uhelper=hal,data=ordered

```

### El Programa de Copia

```

casiano@mylaptop:~/bin$ cat -n someserverbackuptoportabledisk.pl
1 #!/usr/bin/perl
2 use strict;
3 use Log::Log4perl qw{:easy};
4
5 my $ismounted = `mount 2>&1`;
6
7 # log file is in /home/casiano/backup.log
8 Log::Log4perl::init('/home/casiano/bin/logconfig');
9 my $log = Log::Log4perl->get_logger();
10
11 if ($ismounted =~ m{/media/PORTABLE}) {
12
13 my $agent = `keychain --eval --agents ssh 2>&1`;
14 $log->logdie("Error executing keychain") if $?;
15
16 $agent =~ m{SSH_AUTH_SOCK=([^;]+)};;
17 $ENV{SSH_AUTH_SOCK} = $1;
18
19 $log->info("Executing ssh someserverbackup");
20 my $backupinfo = `rsync -ae ssh someserverbackup:/root/ /media/PORTABLE/someserver/root`;
21 $log->info($backupinfo);
22
23 $backupinfo = `rsync -ae ssh someserverbackup:/home/ /media/PORTABLE/someserver/home/ 2>

```

```

24 $log->info($backupinfo);
25
26 exit($?);
27 }
28 $log->logdie("/media/PORTABLE not mounted\n");

```

La clave está protegida con una passphrase. Previamente se ha cargado un agente y se le ha dado la clave `/home/casiano/.ssh/someserverbackupidentity` que es la utilizada en las conexiones que efectúan las dos llamadas a `rsync`.

### Configuraciones SSH en `.ssh/config` y `/etc/sshd_config`

Deberá existir una entrada como esta en el fichero `~/.ssh/config`:

```

Host someserverbackup
HostName someserver
user root
IdentityFile /home/casiano/.ssh/someserverbackupidentity
BatchMode yes

```

la clave `/home/casiano/.ssh/someserverbackupidentity` ha sido publicada en `someserver`.

La máquina debe permitir el acceso al administrador vía SSH:

```

root@mylaptop:/etc/ssh# grep -i permitroot sshd_config
PermitRootLogin yes

```

### El fichero de cron

```
casiano@mylaptop:~$ crontab -e
```

El fichero fué generado con `kcron`:

```

backup de someserver a las 14 y a las 0 horas en disco portable
0 3,21 * * * /home/casiano/bin/someserverbackuptoportabledisk.pl
This file was written by KCron. Copyright (c) 1999, Gary Meyer
Although KCron supports most crontab formats, use care when editing.
Note: Lines beginning with "#\" indicates a disabled task.

```

### Comprobación de la Copia

Es necesario comprobar que las copias se están haciendo correctamente. En mi agenda pongo una tarea que consiste en comprobar la copia. En mi home en cada una de máquinas hay un fichero `AUTOMATICBACKUPCHECKOUT` que contiene la fecha. Si la copia de seguridad de dicho fichero contiene la fecha del día anterior, es señal de que el proceso copió el fichero. En el momento de comprobar la copia actualizo manualmente la fecha almacenada en dicho fichero. Este proceso es facilitado por el siguiente guión:

```

casiano@mylaptop:~/bin$ cat -n checkbackup
1 #!/bin/bash
2 for f in /backup/someserver/home/casiano/AUTOMATICBACKUPCHECKOUT \
 /media/PORTABLE/someserver/home/casiano/AUTOMATICBACKUPCHECKOUT; do
3 echo "Observe the date of $f:";
4 ls -l $f;
5
6 echo "Contents of $f";

```

```
7 echo "*****";
8 cat $f;
9 echo "*****";
10 done
11
12 echo "Now we are going to edit AUTOMATICBACKUPCHECKOUT and the backup log file,"
13 echo "press any key to continue and fill the file with today's date"
14
15 # Wait for key to be pressed
16 read -n 1
17 # Edit both remote and local files
18 vi -p scp://someserver/AUTOMATICBACKUPCHECKOUT /home/casiano/mylaptopbackup.log
```

En el caso de errores puede ser útil revisar el fichero de logs:

```
casiano@mylaptop:~$ cat backup.log
2009/05/06 15:53:42 INFO someserverbackuptoportabledisk.pl-19-main::: Executing ssh someserver
2009/05/06 15:57:12 INFO someserverbackuptoportabledisk.pl-19-main::: Executing ssh someserver
```

# Capítulo 85

## Git

### 85.1. Instalar Git

- Getting Started - Installing Git
- Git for Windows

### 85.2. Introducción al uso de Git con GitHub en GitHub

- GitHub
- <http://help.github.com/>
  1. Set Up Git
  2. Create a Repo
  3. Fork a Repo
  4. Be Social

### 85.3. Un poco de SSH

- Claves Pública y Privada: Estableciendo Autentificación No Interactiva
- Los Ficheros de Configuración de SSH
- Agentes SSH
- Getting started with GitHub by Charles Max Wood

### 85.4. Ramificaciones en Git

- ¿Qué es una rama?
- Procedimientos básicos para ramificar y fusionar
- Gestión de ramificaciones
- Flujos de trabajo ramificados
- Ramas Remotas
- Reorganizando el trabajo realizado
- Recapitulación

Véanse Chapter 3. Ramificaciones en Git del libro de Scott Chacon y las anotaciones en este pdf.

## 85.5. Véase: Tutoriales de Git

- Pro Git book, written by Scott Chacon
- Git Tutorial por Lars Vogel
- 
- Watch Tom Preston-Werner "Mastering Git Basics"
- Watch Bart Trojanowski "Git The Basics Tutorial 1"
- Watch Bart Trojanowski "Git The Basics Tutorial 2"
- Slides from Bart Trojanowski "Git The Basics Tutorial (pdf)"
- Scott Chacon: git talk en blip.tv [12]

## 85.6. Diferencias entre fetch y pull

Git was designed to support a distributed model with no need for a central repository (though you can certainly use one if you like.) Also git was designed so that the client and the "server" don't need to be online at the same time.

Git was designed so that people on an unreliable link could exchange code via email, even.

In order to support this model git maintains a local repository with your code and also an additional local repository that mirrors the state of the remote repository. By keeping a copy of the remote repository locally, git can figure out the changes needed even when the remote repository is not reachable.

Later when you need to send the changes to someone else, git can transfer them as a set of changes from a point in time known to the remote repository.

- The command `git fetch` is the command that says *bring my local copy of the remote repository up to date.*
- The command `git pull` says "bring the changes in the remote repository where I keep my own code."

Normally `git pull` does this by doing a `git fetch` to bring the local copy of the remote repository up to date, and then merging the changes into your own code repository and possibly your working copy.

The take away is to keep in mind that there are often at least *three copies* of a project on your workstation.

- One copy is your own repository with your own commit history.
- The second copy is your working copy where you are editing and building.
- The third copy is your local cached copy of a remote repository.

## 85.7. Mezclando Ficheros Específicos desde otra Rama

- `(master)$git checkout branchname file1 file2 files3`  
`git checkout [-p|--patch] [<tree-ish>] [--] <pathspec>...`

When `<paths>` or `--patch` are given, `git checkout` does not switch branches. It updates the named paths in the working tree from the

index file or from a named <tree-ish> (most often a commit). The <tree-ish> argument can be used to specify a specific tree-ish (i.e. commit, tag or tree) to update the index for the given paths before updating the working tree.

- The following sequence checks out the master branch, reverts the Makefile to two revisions back, deletes hello.c by mistake, and gets it back from the index.

```
$ git checkout master (1)
$ git checkout master^2 Makefile (2)
$ rm -f hello.c
$ git checkout hello.c (3)
```

1. switch branch
2. take a file out of another commit
3. restore hello.c from the index

- If you want to check out all C source files out of the index, you can say

```
$ git checkout -- *.c'
```

Note the quotes around \*.c.

The removed file `hello.c` will also be checked out, even though it is no longer in the working tree, because the file globbing is used to match entries in the index (not in the working tree by the shell).

- If you have an unfortunate branch that is named `hello.c`, this step:

```
$ git checkout hello.c
```

would be confused as an instruction to switch to that branch.

You should instead write:

```
$ git checkout -- hello.c
```

- Git Tip: How to "Merge" Specific Files from Another Branch

## 85.8. Configuración y Algunos alias

### 85.8.1. Alias lg

```
~/local/src/ruby/rubytesting/twitter/location_freq(master)]$ cat ~/.gitconfig
[user]
 email = blabla@gmail.com
 name = Blabla Perez Garcia
[color]
 status = auto
 diff = auto
[merge]
 tool = vimdiff
[alias]
```

```

lg = log --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr %an)%Creset' --
wipe = "!git reset --hard;git clean -fd"
co = checkout
ci = commit
[core]
excludesfile = /Users/casiano/.gitignore_global
[difftool "sourcetree"]
cmd = opendiff \"$LOCAL\" \"$REMOTE\"
path =
[mergetool "sourcetree"]
cmd = /Applications/SourceTree.app/Contents/Resources/opendiff-w.sh \"$LOCAL\" \"$REMOTE\" -
trustExitCode = true
[credential]
helper = osxkeychain

```

Para crear un alias usamos el comando `git config`. Por ejemplo:

```
git config --global alias.st status
```

Uno de mis alias favoritos es `git lg`:

```
[~/local/src/ruby/rubytesting/twitter/twitter-test(develop)]$ git lg
ccaa00f - (HEAD, master, develop) minor changes (17 minutes ago Casiano Rodriguez Leon)
41098fe - (origin/rubystyle, origin/master, test, rubystyle) transformed to a more Ruby style
fb3e662 - (origin/javastyle, javastyle) instructions (6 days ago Casiano Rodriguez Leon)
ea79bc3 - more instructions (6 days ago Casiano Rodriguez Leon)
9b469b1 - added template for configure.rb (6 days ago Casiano Rodriguez Leon)
87d8a07 - un cambio (7 days ago Casiano Rodriguez Leon)
```

### 85.8.2. Auto-completado

Otra utilidad importante es la posibilidad de auto-completar comandos git. En mi fichero de arranque bash tengo una línea como:

```
source ~/.git-completion.bash
```

El fichero `.git-completion.bash` es posible encontrarlo en la distribución de git: <https://github.com/git/git/blob/completion.bash>

Con él es posible auto-completar un comando git pulsando TABULADOR:

```
[~/local/src/ruby/rubytesting/twitter/twitter-test(develop)]$ git help che<TABULADOR>
check-attr check-ref-format checkout-index cherry-pick
check-ignore checkout cherry
```

### 85.8.3. Poner la rama en el prompt de la shell

En mi fichero de arranque bash tengo una línea como:

```
source ~/.git_prompt.sh
```

Esto permite ver la rama actual en el prompt. El fichero `.git-prompt.sh` es posible encontrarlo en la distribución de git: <https://github.com/git/git/blob/master/contrib/completion/git-prompt.sh>

## 85.9. Hub

hub is a command line tool that wraps git in order to extend it with extra features and commands that make working with GitHub easier.

- clone

```
$ hub clone rtomayko/tilt

expands to:
$ git clone git://github.com/rtomayko/tilt.git
```

- create

```
$ hub create
[repo created on GitHub]
> git remote add origin git@github.com:YOUR_USER/CURRENT_REPO.git

with description:
$ hub create -d 'It shall be mine, all mine!'

$ hub create recipes
[repo created on GitHub]
> git remote add origin git@github.com:YOUR_USER/recipes.git

$ hub create sinatra/recipes
[repo created in GitHub organization]
> git remote add origin git@github.com:sinatra/recipes.git
```

## 85.10. Vim y Git. FuGitive: a Git wrapper so awesome, it should be illegal

- vim-fugitive en GitHub
- The Fugitive Series - a retrospective
- Fugitive.vim - resolving merge conflicts with vimdiff
- Fugitive.vim - a complement to command line git

# Capítulo 86

## Debugging, Depurando

### 86.1. Véase

- Jason R. Clark - Spelunking in Ruby. Baruco 2014
- Tutorial: Debugging in Ruby by dontangg on November 5, 2010
- Debugging Ruby programs 101. How to use Ruby's debugger to fix problems in your code. Pat Eyler (pat.eyler@gmail.com), Co-founder, Seattle Ruby Brigade
- 
- Sand Diego Ruby Community: Episode 111 Ruby Debugging: Life After "Puts"
  - podcast
  - slides

### 86.2. ruby-debug Cheat Sheet

### 86.3. Depuración con Pry

1. Rubyists, It's Time to PRY Yourself Off IRB! by Benjamin Tan Wei Hao Published November 25, 2013
2. Pry home page
3. Documentación de pry-stack-explorer y pry-stack-explorer en github
4. pry-debugger

# Capítulo 87

## Rake

### 87.1. Argumentos en una Tarea

Véase How do I pass command line arguments to a rake task? en Stackoverflow.

You can specify formal arguments in rake by adding symbol arguments to the task call. For example:

```
require 'rake'

task :my_task, :arg1, :arg2 do |t, args|
 puts "Args were: #{args}"
end

task :invoke_my_task do
 Rake.application.invoke_task("my_task[1, 2]")
end

or if you prefer this syntax...
task :invoke_my_task_2 do
 Rake::Task[:my_task].invoke(3, 4)
end

a task with prerequisites passes its
arguments to it prerequisites
task :with_prerequisite, :arg1, :arg2, :needs => :my_task

equivalently...
task :with_prerequisite_2, [:arg1, :arg2] => :my_task

to specify default values,
we take advantage of args being a Rake::TaskArguments object
task :with_defaults, :arg1, :arg2 do |t, args|
 args.with_defaults(:arg1 => :default_1, :arg2 => :default_2)
 puts "Args with defaults were: #{args}"
end
```

Then, from the command line:

```
> rake my_task[1,2]
Args were: {:arg1=>"1", :arg2=>"2"}

> rake "my_task[1, 2]"
Args were: {:arg1=>"1", :arg2=>"2"}
```

```

> rake invoke_my_task
Args were: {::arg1=>"1", :arg2=>"2"}

> rake invoke_my_task_2
Args were: {::arg1=>3, :arg2=>4}

> rake with_prerequisite[5,6]
Args were: {::arg1=>"5", :arg2=>"6"}

> rake with_prerequisite_2[7,8]
Args were: {::arg1=>"7", :arg2=>"8"}

> rake with_defaults
Args with defaults were: {::arg1=>:default_1, :arg2=>:default_2}

> rake with_defaults['x','y']
Args with defaults were: {::arg1=>"x", :arg2=>"y"}

```

As demonstrated in the second example, if you want to use spaces, the quotes around the target name are necessary to keep the shell from splitting up the arguments at the space.

## 87.2. Rake::TestTask

La clase Rake::TestTask nos permite crear una tarea que ejecuta un conjunto de pruebas;

```

Rake::TestTask.new do |t|
 t.libs << "test"
 t.test_files = FileList['test/test*.rb']
 t.verbose = true
end

```

Si se llama a `rake` con la opción `TEST=filename` entonces sólo se ejecutará esa prueba.

Si se invoca con la opción `TESTOPTS=options`, las opciones se pasarán a `Test::Unit`. Ejemplos:

```

rake test # run tests normally
rake test TEST=just_one_file.rb # run just one test file.
rake test TESTOPTS="-v" # run in verbose mode

```

Sigue otro ejemplo:

```

MacBookdeJohn:htmlRB john$ cat Rakefile
$:.unshift 'lib/'
require 'rake/testtask'

rake test TEST=just_one_file.rb # run just one test file.
rake test TESTOPTS="-v" # run in verbose mode
Rake::TestTask.new do |t|
 # t.libs: List of directories to be added to $LOAD_PATH. (default is 'lib')
 t.libs << "test"
 t.pattern = 'test/tc*.rb' # or t.test_files = FileList['test/tc*.rb']
 t.warning = true
 t.verbose = true # verbose test output. (default is false)
end

```

Ejecuciones:

```

MacBookdeJohn:htmlRB john$ ls -ltr test/
total 16
-rw-r--r-- 1 john staff 835 29 nov 12:27 tc_01_smoke.rb
-rw-r--r-- 1 john staff 1023 9 dic 11:35 tc_02_smoke.rb
MacBookdeJohn:htmlRB john$ rake test
(in /Users/john/Dropbox/src/projects-ull-john/ruby/htmlRB)
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby -w -I"lib:test" "/System/L
./lib/html.rb:40: warning: '&' interpreted as argument prefix
Loaded suite /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/1.8/rake/rake
Started
...
Finished in 0.001125 seconds.

3 tests, 7 assertions, 0 failures, 0 errors

```

Con la opción TESTOPTS="-v" la salida es mas detallada:

```

MacBookdeJohn:htmlRB john$ rake test TESTOPTS="-v"
(in /Users/john/Dropbox/src/projects-ull-john/ruby/htmlRB)
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby -w -I"lib:test" "/System/L
./lib/html.rb:40: warning: '&' interpreted as argument prefix
Loaded suite /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/1.8/rake/rake
Started
test_attr(TestHTML): .
test_attr2(TestHTML): .
test_simplepage(TestHTML): .

Finished in 0.000844 seconds.

3 tests, 7 assertions, 0 failures, 0 errors

```

Ejecución de un solo programa de prueba:

```

MacBookdeJohn:htmlRB john$ rake test TEST=tc_02_smoke.rb
(in /Users/john/Dropbox/src/projects-ull-john/ruby/htmlRB)
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby -w -I"lib:test" "/System/L
./lib/html.rb:40: warning: '&' interpreted as argument prefix
Loaded suite /System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/lib/ruby/1.8/rake/rake
Started
..
Finished in 0.000623 seconds.

2 tests, 6 assertions, 0 failures, 0 errors

```

## Donde

- Este ejemplo en gitHub
- [~/rubytesting/htmlRB(master)]\$ pwd -P  
`/Users/casiano/local/src/ruby/rubytesting/htmlRB`  
[~/rubytesting/htmlRB(master)]\$ git remote -v  
origin git@github.com:crguezl/html-method\_missing-example.git (fetch)  
origin git@github.com:crguezl/html-method\_missing-example.git (push)

### 87.3. Enlaces

- Rake Tutorial por Jason Seifer
- Using the Rake Build Language por Martin Fowler (2005)
- Building with Rake por Jim Weirich (2003)
- **Rake**
- Using Rake to Automate Tasks por Stuart Ellis
- Rake User Guide por Jim Weirich
- Custom Rake Tasks en RailCasts por Ryan Bates

# Capítulo 88

## Documentando el Código: RDoc, Yard, UML

### 88.1. RDoc

1. RDoc is an application that produces documentation for one or more Ruby source files.
2. It works similarly to JavaDoc, parsing the source, and extracting the definition for classes, modules, and methods (along with includes and requires).
3. It associates with these optional documentation contained in the immediately preceding comment block, and then render the result using a pluggable output formatter.

#### 88.1.1. Opciones

```
[~/LPPbook(master)]$ rdoc --help
Usage: rdoc [options] [names...]
```

Available formatters:

```
darkfish - HTML generator, written by Michael Granger
ri - creates ri data files
```

-a, --all	Synonym for --visibility=private.
-f, --fmt, --format=FORMAT	Set the output formatter. One of: darkfish

#### 88.1.2. RDoc y Rake

1. The `rdoc` command ships with Ruby and by default generates all `.rb` files in or below the current directory.
2. You probably want something like `rdoc lib` to just include your project's main Ruby code.
3. Integrate RDoc into Rake. RDoc comes with RDoc/Task which can be configured for your project, so you can just run `rake rdoc`. Once generated, you can view the generated output at `doc/index.html`. The RDocTask will create the following targets:

```
:rdoc
Main task for this RDOC task.
:clobber_rdoc
Delete all the rdoc files. This target is automatically added to the main clobber target.
```

```
:rerdoc
Rebuild the rdoc files from scratch, even if they are not out of date.
```

Simple Example:

```
require 'rdoc/task'

Rake::RDocTask.new do |rd|
 rd.main = "README.rdoc"
 rd.rdoc_files.include("README.rdoc", "lib/**/*.rb")
end
```

You may wish to give the task a different name, such as if you are generating two sets of documentation.

For instance, if you want to have a *development set of documentation* including private methods:

```
require 'rdoc/task'

Rake::RDocTask.new(:rdoc_dev) do |rd|
 rd.main = "README.doc"
 rd.rdoc_files.include("README.rdoc", "lib/**/*.rb")
 rd.options << "--all"
end
```

The tasks would then be named :rdoc\_dev, :clobber\_rdoc\_dev, and :rerdoc\_dev.

```
[~/srcLPPRuby/rdoc(master)]$ rake -T
rake clobber_rdoc # Remove RDoc HTML files
rake rdoc # Build RDoc HTML files
rake rerdoc # Rebuild RDoc HTML files
```

### 88.1.3. Servidor de Documentación Local

Rubygems has built-in support for generating and viewing the RDoc for installed gems.

RDoc is generated when the gem is installed (using `gem install --no-rdoc [name]` skips the local RDoc generation).

You can then use `gem server` to view your local gem RDoc at `http://localhost:8808`.

### 88.1.4. Ejemplo

A typical small Ruby program commented using RDoc might be as follows. You can see the formattted result in EXAMPLE.rb and Anagram.

#### Donde

- En el repositorio <https://github.com/crguezl/rdoc-example> puede encontrar los fuentes.
- ```
[~/srcLPPRuby/rdoc(master)]$ pwd -P
/Users/casiano/local/src/ruby/LPP/rdoc
```

example.rb

```
# The program takes an initial word or phrase from
# the command line (or in the absence of a
# parameter from the first line of standard
# input). It then reads successive words or
# phrases from standard input and reports whether
# they are anagrams of the first word.
#
# Author::    Dave Thomas (mailto:dave@x.y)
# Copyright:: Copyright (c) 2002 The Pragmatic Programmers, LLC
# License::   Distributes under the same terms as Ruby

# This class holds the letters in the original
# word or phrase. The is_anagram? method allows us
# to test if subsequent words or phrases are
# anagrams of the original.
class Anagram

  # Remember the letters in the initial word
  def initialize(text)
    @initial_letters = letters_of(text)
  end

  # Test to see if a new word contains the same
  # letters as the original
  def is_anagram?(text)
    @initial_letters == letters_of(text)
  end

  # Determine the letters in a word or phrase
  #
  # * all letters are converted to lower case
  # * anything not a letter is stripped out
  # * the letters are converted into an array
  # * the array is sorted
  # * the letters are joined back into a string
  def letters_of(text)
    text.downcase.delete('`a-z').split('').sort.join
  end
end
```

HTML generado

-
-
-

Mostrando la documentación con pry

```
[1] pry(main)> require './example'
=> true
[2] pry(main)> a = Anagram.new('foobar')
=> #<Anagram:0x007fd6d318f1a8 @initial_letters="abfoor">
```

```
[3] pry(main)> show-doc a.is_anagram?

From: /Users/casiano/local/src/ruby/LPP/rdoc/example.rb @ line 22:
Owner: Anagram
Visibility: public
Signature: is_anagram?(text)
Number of lines: 2

Test to see if a new word contains the same
letters as the original
```

88.1.5. Buenas Normas de Documentación

Template to employ for documenting Ruby methods. It creates a somewhat standard, easily readable comment structure.

See Beautiful standardized RDoc comments for your Ruby / Rails methods

```
#  
#  
# * *Args* :  
# - ++ ->  
# * *Returns* :  
# -  
# * *Raises* :  
# - ++ ->  
#  
def method_name  
  
end
```

88.1.6. Enlaces

- Using RDoc a tutorial by Mark Perham
- Cheat sheet
- RDoc markup reference
- RDoc how to
- Manual de Sintáxis: RDoc::Markup
- *RDOC::Markup*
- RubyDoc en WikiBooks
- *RDOC* en sourceforge
- Programming With Ruby Episode 12, Documentation en YouTube

88.2. YARD

Una alternativa a RDoc es YARD. YARD does not impose a specific markup. It can use standard RDoc markup formatting, but YARD also supports textile and markdown via the command-line switch or `.yardopts` file. This means that you are free to use whatever formatting you like.

88.3. Diagramas UML

88.3.1. umlify

1. umlify: umlify takes your ruby project's source code and creates an uml class diagram out of it.
2. Go to your gem project directory
3. type: `umlify lib/*/*`
4. Open `uml.png`

Capítulo 89

Bundler

In a nutshell, you'll be editing a file that describes what gems your project depends on and slightly modifying how you require those gems.

Repase la sección *Creando Gemas y Publicándolas en rubygems.org 90.*

89.1. bundle show: Que versión de una gema estamos usando

```
[~/Dropbox/src/ruby/sinatra/bootstrap_example(master)]$ cat Gemfile
source 'https://rubygems.org'
gem 'sinatra'

[~/Dropbox/src/ruby/sinatra/bootstrap_example(master)]$ bundle show

Gems included by the bundle:
* bundler (1.3.5)
* rack (1.5.2)
* rack-protection (1.5.0)
* sinatra (1.4.2)
* tilt (1.4.1)
```

89.2. bundle show: searching

The show command still has one more trick up it's sleeve, though:

```
bundle show --paths
```

Printing a list of paths may not sound terribly useful, but it makes it trivial to search through the source of every gem in your bundle. Want to know where ActionDispatch::RemoteIp is defined? It's a one-liner:

```
$ grep ActionDispatch::RemoteIp 'bundle show --paths'
```

Whether you use grep, ack, or ag, it's very easy to set up a shell function that allows you to search the current bundle in just a few characters. Here's mine:

```
function back () {
ack "$@" 'bundle show --paths'
}
```

With that function, searching becomes even easier:

```
$ back ActionDispatch::RemoteIp
```

Véase Hack your bundle for fun and profit

89.3. La opción PATH (:path) del método gem

You can specify that a gem is located in a particular location on the file system. Relative paths are resolved relative to the directory containing the Gemfile.

Similar to the semantics of the :git option, the :path option requires that the directory in question either contains a .gemspec for the gem, or that you specify an explicit version that bundler should use.

```
gem "rails", :path => "vendor/rails"
```

89.4. Véase

- *Hack your bundle for fun and profit*
- *Repase la sección Creando Gemas y Publicándolas en rubygems.org 90.*
- *Bundler*
- *Three Bundler Benefits. Bundler Gives You Pay-as-you-go Gem Management por Paolo Perrotta*
- *Getting Started with Bundler por Michael Morin*
- *Bundler Best Practices por Chris Jones en viget.com*

Capítulo 90

Creando Gemas y Publicándolas en rubygems.org

Un ejemplo parecido al usado en este capítulo está disponible en GitHub.

90.1. Eligiendo Nombre

Antes de elegir el nombre una gema hay que estar seguro de que no está cogido. Visite <http://rubygems.org/gems/> y compruebe que está disponible. Véanse las recomendaciones en: Name your gem

90.2. Creando la Estructura Inicial de la Gema

```
gem install bundler
```

Once you've installed Bundler, you should change directories to the location that you want to create your gem folder (for me, that location is ~/code/gems).

Once there you will create the gem like so:

```
bundle gem my_gem
```

Where my_gem is your gem's name. Note that if you are building a command line tool you should add the -b flag to this command (bundle gem my_gem -b).

You should see output that looks something like this:

```
create  my_gem/Gemfile
create  my_gem/Rakefile
create  my_gem/.gitignore
create  my_gem/my_gem.gemspec
create  my_gem/lib/my_gem.rb
create  my_gem/lib/my_gem/version.rb
```

Initializing git repo in /Users/mbleigh/code/gems/my_gem

Bundler has just generated a skeleton for your gem. You should recognize many of the files of the general structure of a gem.

Some of the nice things that Bundler has done for us include:

- **.gitignore:** *There are certain files that you will not want to have to your git repository. Bundler automatically excludes the most common of these.*
- **version.rb:** *Bundler creates my_gem/lib/my_gem/version.rb automatically. This file is used as a single place to track the current version of your library. By default, Bundler sets the version to 0.0.1.*

- **Rakefile:** The Rakefile created by Bundler is already set up with Bundler's gem helpers.

These helpers allow you to release a new version of your gem (including creating a release tag in git and pushing the compiled gem to RubyGems) with a single command.

```
$ rake -T
rake build    # Build my_gem-0.0.1.gem into the pkg directory
rake install   # Build and install my_gem-0.0.1.gem into system gems
rake release   # Create tag v0.0.1 and build and push my_gem-0.0.1.gem to Rubygems
rake spec      # Run RSpec code examples
```

Estos son los contenidos del Rakefile:

```
$ cat Rakefile
$:.unshift File.dirname(__FILE__) + 'lib'
require "bundler/gem_tasks"

require 'rspec/core/rake_task'
RSpec::Core::RakeTask.new
task :default => :spec
```

- **Gemfile** este es un ejemplo - ya modificado - del Gemfile generado:

```
[~/srcLPPRuby/building_gems/bleighTutorial_to_building_gems/my_gem(master)]$ cat Gemfile
source 'https://rubygems.org'
```

```
# Specify your gem's dependencies in my_gem.gemspec
gemspec
```

```
#gem 'guard'
#gem 'guard-rspec'
#gem 'guard-bundler'
#gem 'rb-fsevent', '~= 0.9.1'
```

- **gemspec**

```
$ cat -n my_gem.gemspec
 1 # -*- encoding: utf-8 -*-
 2 lib = File.expand_path('../lib', __FILE__)
 3 $LOAD_PATH.unshift(lib) unless $LOAD_PATH.include?(lib)
 4 require 'my_gem/version'
 5
 6 Gem::Specification.new do |gem|
 7   gem.name          = "my_gem"
 8   gem.version       = MyGem::VERSION
 9   gem.authors       = ["John Smith"]
10   gem.email         = ["some.name@pochomail.com"]
11   gem.description   = %q{TODO: Write a gem description}
12   gem.summary        = %q{TODO: Write a gem summary}
13   gem.homepage      = ""
14
15   gem.files         = `git ls-files`.split($/)
16   gem.executables   = gem.filesgrep(%r{^bin/})
                           .map{ |f| File.basename(f) }
17   gem.test_files    = gem.filesgrep(%r{^(test|spec|features)/})
```

```

18   gem.require_paths = ["lib"]
19
20   gem.add_dependency 'sinatra'
21   gem.add_development_dependency 'rspec', '>=2.11'
22 end

■ version.rb

$ cat -n lib/my_gem/version.rb
1 module MyGem
2   VERSION = "0.0.1"
3 end

```

90.3. Underscores and Dashes

Véanse las recomendaciones en: *Name your gem.*

Use underscores for multiple words

If a class or module has multiple words, use underscores to separate them. This matches the file the user will require, making it easier for the user to start using your gem.

Use dashes for extensions

If you're adding functionality to another gem, use a dash. This usually corresponds to a / in the require statement (and therefore your gem's directory structure) and a :: in the name of your main class or module.

Mix underscores and dashes appropriately

If your class or module has multiple words and you're also adding functionality to another gem, follow both of the rules above. For example, net-http-digest_auth adds HTTP digest authentication to net/http. The user will require net/http/digest_auth to use the extension (in class Net::HTTP::DigestAuth).

Don't use UPPERCASE letters

OS X and Windows have case-insensitive filesystems by default. Users may mistakenly require files from a gem using uppercase letters which will be non-portable if they move it to a non-windows or OS X system. While this will mostly be a newbie mistake we don't need to be confusing them more than necessary.

Ventajas By following the community conventions you:

- Make your library easier to "guess" how to use because developers can use common idioms to which they have become accustomed.
- Make your library easier for contributors to join by providing familiar patterns and locations for code

90.4. Dependencias de la Gema

Nearly all gems are going to have dependencies upon other libraries that they require in order to function. These dependencies are explicitly declared in the gemspec so that when a gem is installed, all of the gems that are needed to run that gem can automatically be installed with it.

There are two types of gem dependencies, runtime dependencies and development dependencies.

- *Runtime dependencies* are libraries that your gem needs in order to function. For example, if you built a Rails extension, you would need to add Rails to the runtime dependencies of your project.
- *Development dependencies*, on the other hand, are gems that are only needed by people who are going to contribute to the code of the gem.
- Gems are, by default, installed only with the runtime dependencies. This avoids cluttering up an environment with development dependencies if they aren't needed.

Adding dependencies to your gemspec is extremely simple; all you need to do is add a line to the gemspec file for each dependency:

```
# -*- encoding: utf-8 -*-
require File.expand_path('../lib/my_gem/version', __FILE__)

Gem::Specification.new do |gem|
  # ...

  gem.add_dependency 'rails'
  gem.add_development_dependency 'rspec', '>= 2.7'
end
```

In addition to specifying the gem name of a dependency, you can also specify a version of the gem to get even more specific. Rubygems will default to the latest version if no specific version is required.

```
git add .
git commit -m "Initial Import"
```

90.5. Jerarquía de Ficheros y Directorios, Nombres y Requires

- *How you name your files and where you put them can improve the usability of your library by other developers.*
- *For instance, in a GitHub repository page one can simply press t and start typing a filename, then look at the code for that file by pressing return.*
- *By creating sane, easily guessable filenames you are making it simple for others to read your code.*

In general, you will have one class per file and one folder per module namespace, so for instance:

- *MyGem becomes lib/my_gem.rb*
- *MyGem::Widget becomes lib/my_gem/widget.rb*
- *MyGem::Widgets::FooBar becomes lib/my_gem/widgets/foo_bar.rb*

When a developer wants to use your library, they should be able to do so (in almost all cases) by making a single `require` statement that is identical to the gem name.

That means that in the root file you need to make any additional require statements necessary for your gem to function. So if I have a MyGem module, a MyGem::Widget class, and a MyGem::Widgets::FooBar class, the lib/my_gem.rb file in my gem might look like this:

```

require 'external_library' # require any external dependencies

module MyGem # it is best to declare the base module at the top
end

require 'my_gem/version' # created by Bundler
require 'my_gem/widget'
require 'my_gem/widgets/foo_bar'

```

By requiring all of the files necessary for your gem to run in the base file you make it easier for developers to use your library.

90.6. Gemas con Un Gran Número de Utilidades Independientes

Some gems, however, may be made up of multiple parts that could be used independently of each other. ActiveSupport, for example, provides a large number of useful utilities that, while they function together, can also function separately.

If I add require active_support to my code I load all of ActiveSupport. While this may be what I want in some cases (like inside a Rails application) in other cases I may just want a specific piece of ActiveSupport. Luckily, ActiveSupport is designed to handle this well. If I, for instance, add require active_support/core_ext I will only be loading the Ruby core extensions that are a part of ActiveSupport.

How can you make this work in your library? It's quite simple: your base file should, when required, require all the other parts of your library. However, each part of your library should, at its top, require any other parts or external dependencies so that it may be included without the user having previously required the base file. Let's take a look at an example:

```

# in lib/my_gem.rb

require 'my_gem/widget'
require 'my_gem/widgets/foo_bar'
require 'my_gem/widgets/baz'

# in lib/my_gem/widget.rb

require 'external_library'

module MyGem
  class Widget
  end
end

# in lib/my_gem/widgets/foo_bar

require 'my_gem/widget'

module MyGem
  module Widgets
    class FooBar < MyGem::Widget
    end
  end
end

```

Each of the files in the above example can be required independently, giving developers the flexibility to use only a subset of your library's functionality if needed.

Remember that `require` statements will only load the code from a file once, so it is safe to require the same file multiple times.

90.7. Sharing Source Code From a Public git Repository

The simplest way (from the author's perspective) to share a gem for other developers' use is to distribute it in source code form. If you place the full source code for your gem on a public git repository, then other users can install it with [Bundler's git functionality](#).

For example, you can install the latest code for the `wicked_pdf` gem in a project by including this line in your `Gemfile`:

```
gem "wicked_pdf", :git => "git://github.com/mileszs/wicked_pdf.git"
```

(Véase `bundle help gemfile`)

Installing a gem directly from a git repository is a feature of Bundler, not a feature of RubyGems. Gems installed this way will not show up when you run `gem list`.

90.8. Publishing to RubyGems.org

The simplest way to distribute a gem for public consumption is to use RubyGems. Gems that are published to RubyGems can be installed via the `gem install` command or through the use of tools such as Bundler.

Crear una cuenta en rubygems.org

To begin, you'll need to create an account on RubyGems.org. Visit the [sign up](#) page and supply an email address that you control, a handle (username) and a password.

El "`handle`" que te solicita es tu nombre de usuario (`loginname`).

Autenticación

After creating the account, use your email and password when pushing the gem. (rubygems saves the credentials in `~/.gem/credentials` for you so you only need to log in once.)

```
~]$ cat .gem/credentials
---
:rubygems_api_key: XXXXXXXXXXXXXXXXXXXXXXXYYY
[~]$
```

gem build *Para construir la gema podemos usar:*

```
[/tmp/tutu/my_gem]$ gem build my_gem.gemspec
WARNING: no homepage specified
Successfully built RubyGem
Name: my_gem
Version: 0.0.1
File: my_gem-0.0.1.gem
```

rake build

o bien podemos usar `rake build`. para contruir la gema.

gem push *To publish version 0.1.0 of a new gem named squid-utils:*

```
$ gem push squid-utils-0.1.0.gem
Enter your RubyGems.org credentials.
```

Don't have an account yet? Create one at https://rubygems.org/sign_up

Email: gem_author@example

Password:

Signed in.

Pushing gem to RubyGems.org...

Successfully registered gem: squid-utils (0.1.0)

Congratulations! Your new gem is now ready for any ruby user in the world to install!

rake release

We can also use rake release:

```
[/tmp/tutu/my_gem]$ rake -T
rake build      # Build my_gem-0.0.1.gem into the pkg directory
rake install    # Build and install my_gem-0.0.1.gem into system gems
rake release    # Create tag v0.0.1 and build and push my_gem-0.0.1.gem to Rubygems
```

gem install: Instalar una gema

Para instalarla podemos hacer uso de gem install:

```
[/tmp/tutu/my_gem]$ gem install ./my_gem-0.0.1.gem
Successfully installed my_gem-0.0.1
1 gem installed
```

o bien rake install.

gem uninstall: Desinstalar una gema

Para desinstalarla:

```
[/tmp/tutu]$ gem uninstall --force my_gem
Successfully uninstalled my_gem-0.0.1
[/tmp/tutu]$ pry
[1] pry(main)> require 'my_gem'
LoadError: cannot load such file -- my_gem
from /Users/casiano/.rvm/rubies/ruby-2.0.0-p247/lib/ruby/2.0.0/rubygems/core_ext/kernel_requir
[2] pry(main)>
```

90.9. gem owner: Compartiendo la Propiedad de una Gema

If you have multiple maintainers for your gem you can give your fellow maintainers permission to push the gem to RubyGems.org through the gem owner command.

```
gem owner my_gem -a foo@example.com
```

Estas son las opciones que admite:

```
$ gem help owner
Usage: gem owner GEM [options]
```

Options:

-k, --key KEYNAME

Use the given API key
from ~/.gem/credentials

| | |
|--|---|
| <code>-a, --add EMAIL</code> | Add an owner |
| <code>-r, --remove EMAIL</code> | Remove an owner |
|
Local/Remote Options: | |
| <code>-p, --[no-]http-proxy [URL]</code> | Use HTTP proxy for remote operations |
|
Common Options: | |
| <code>-h, --help</code> | Get help on this command |
| <code>-V, --[no-]verbose</code> | Set the verbose level of output |
| <code>-q, --quiet</code> | Silence commands |
| <code>--config-file FILE</code> | Use this config file instead of default |
| <code>--backtrace</code> | Show stack backtrace on errors |
| <code>--debug</code> | Turn on Ruby debugging |

Arguments:
`GEM` gem to manage owners for

Summary:
Manage gem owners on RubyGems.org.

90.10. gem cert: Gem Security

Installing a gem allows that gem's code to run in the context of your application. Clearly this has security implications: [installing a malicious gem on a server could ultimately result in that server being completely penetrated by the gem's author](#). Because of this, the security of gem code is a topic of active discussion within the Ruby community.

RubyGems *has had the ability to cryptographically sign gems since version 0.8.11.*

gem cert

This signing works by using the gem cert command to create a key pair, and then packaging signing data inside the gem itself.

```
[~/srcLPPruby/matrices_dispersas(master)]$ gem help cert
Usage: gem cert [options]
```

| | |
|-------------------------------------|---|
| Options: | |
| <code>-a, --add CERT</code> | Add a trusted certificate. |
| <code>-l, --list [FILTER]</code> | List trusted certificates where the subject contains FILTER |
| <code>-r, --remove FILTER</code> | Remove trusted certificates where the subject contains FILTER |
| <code>-b, --build EMAIL_ADDR</code> | Build private key and self-signed certificate for EMAIL_ADDR |
| <code>-C, --certificate CERT</code> | Signing certificate for --sign |
| <code>-K, --private-key KEY</code> | Key for --sign or --build |
| <code>-s, --sign CERT</code> | Signs CERT with the key from -K and the certificate from -C |

Common Options:

| | |
|---------------------------------|---|
| <code>-h, --help</code> | Get help on this command |
| <code>-V, --[no-]verbose</code> | Set the verbose level of output |
| <code>-q, --quiet</code> | Silence commands |
| <code>--config-file FILE</code> | Use this config file instead of default |
| <code>--backtrace</code> | Show stack backtrace on errors |
| <code>--debug</code> | Turn on Ruby debugging |

Summary:

Manage RubyGems certificates and signing settings

Description:

The cert command manages signing keys and certificates for creating signed gems. Your signing certificate and private key are typically stored in `~/.gem/gem-public_cert.pem` and `~/.gem/gem-private_key.pem` respectively.

To build a certificate for signing gems:

```
gem cert --build you@example
```

If you already have an RSA key, or are creating a new certificate for an existing key:

```
gem cert --build you@example --private-key /path/to/key.pem
```

If you wish to trust a certificate you can add it to the trust list with:

```
gem cert --add /path/to/cert.pem
```

You can list trusted certificates with:

```
gem cert --list
```

or:

```
gem cert --list cert_subject_substring
```

If you wish to remove a previously trusted certificate:

```
gem cert --remove cert_subject_substring
```

To sign another gem author's certificate:

```
gem cert --sign /path/to/other_cert.pem
```

For further reading on signing gems see 'ri Gem::Security'

The user needs to add the author or source site public key as a trusted certificate (you only need to do this once per author/site):

```
gem cert --add <(curl -Ls https://gist.github.com/sferik/4701180/raw/public_cert.pem)
```

Security Policy: gem install twitter -P HighSecurity

The `gem install` command optionally lets you set a security policy (via the option `-P` or `--trust-policy POLICY` which specifies gem trust policy) and you can verify the signing key for a gem before you install it:

```
gem install twitter -P HighSecurity
```

Available Policies

Available policies are:

1. *NoSecurity - Well, no security at all. Signed packages are treated like unsigned packages.*
2. *LowSecurity - Pretty much no security. If a package is signed then RubyGems will make sure the signature matches the signing certificate, and that the signing certificate hasn't expired, but that's it. A malicious user could easily circumvent this kind of security.*
3. *MediumSecurity - Better than LowSecurity and NoSecurity, but still fallible. Package contents are verified against the signing certificate, and the signing certificate is checked for validity, and checked against the rest of the certificate chain. The biggest improvement over LowSecurity is that MediumSecurity won't install packages that are signed by untrusted sources. Unfortunately, MediumSecurity still isn't totally secure – a malicious user can still unpack the gem, strip the signatures, and distribute the gem unsigned.*
4. *HighSecurity - The HighSecurity policy is identical to the MediumSecurity policy, except that it does not allow unsigned gems. A malicious user can't modify the package contents without invalidating the signature, and he can't modify or remove signature or the signing certificate chain*

However, this method of securing gems is not widely used. It requires a number of manual steps on the part of the developer, and there is no well-established chain of trust for gem signing keys.

Véase

1. *Ruby Gems: Using High Security Trust Policy*
2. *A Practical Guide to Using Signed Ruby Gems - Part 1: Bundler*
3. *A Practical Guide to Using Signed Ruby Gems - Part 2: Heroku*

90.11. `gem yank`: Retirando una Versión de una Gema de Ruby-Gems.org

You can't push the same gem version twice. Yanking just marks a particular version as 'invalid':

```
$ gem list tictactoe -ra
*** REMOTE GEMS ***
tic-tac.toe (0.3.4, 0.3.3, 0.3.2, 0.3.1)

gem yank tic-tac-toe -v0.3.4
```

Veamos la página de ayuda:

```
[~/ruby_quiz/tictactoe/solutions/eduardo_segredo(john)]$ gem help yank
Usage: gem yank GEM -v VERSION [-p PLATFORM] [--undo] [--key KEY_NAME] [options]
```

Options:

| | |
|------------------------------------|---------------------------------------|
| <code>-v, --version VERSION</code> | Specify version of gem to remove |
| <code>--platform PLATFORM</code> | Specify the platform of gem to remove |

```
--undo  
-k, --key KEY_NAME           Use API key from your gem credentials file
```

Common Options:

| | |
|--------------------|---|
| -h, --help | Get help on this command |
| -V, --[no-]verbose | Set the verbose level of output |
| -q, --quiet | Silence commands |
| --config-file FILE | Use this config file instead of default |
| --backtrace | Show stack backtrace on errors |
| --debug | Turn on Ruby debugging |

Arguments:

GEM name of gem

Summary:

Remove a specific gem version release from RubyGems.org

Description:

Remove a specific gem version release from RubyGems.org

In previous versions we had to install gemcutter. It will add global yank command for RubyGems.

```
$ rvm rubygems latest  
$ gem update --system  
$ gem install gemcutter
```

It seems that the current version of RubyGems already includes yank.

90.12. Semantic Versioning

Definición

A versioning policy is merely a set of simple rules governing how version numbers are allocated.

1. *It can be very simple (e.g. the version number is a single number starting with 1 and incremented for each successive version),*
2. *or it can be really strange (Knuth's TeX project had version numbers: 3, 3.1, 3.14, 3.141, 3.1415; each successive version added another digit to PI).*

Por que usar Semantic Versioning

The RubyGems team urges gem developers to follow the Semantic Versioning standard for their gem's versions.

1. *The RubyGems library itself does not enforce a strict versioning policy, but using an irrational policy will only be a disservice to those in the community who use your gems.*

Ejemplo

1. *Suppose you have a stack gem that holds a Stack class with both push and pop functionality. Your CHANGELOG¹ might look like this if you use semantic versioning:*

¹ A changelog is a log or record of changes made to a software project, usually including such records as bug fixes, new features, etc. Some open source projects include a changelog as one of the top level files in their distribution.

Version 0.0.1: The initial Stack class is released.
Version 0.0.2: Switched to a linked list implementation because it is cooler.
Version 0.1.0: Added a depth method.
Version 1.0.0: Added top and made pop return nil (pop used to return the old top item).
Version 1.1.0: push now returns the value pushed (it used to return nil).
Version 1.1.1: Fixed a bug in the linked list implementation.
Version 1.1.2: Fixed a bug introduced in the last fix.

Resumen

1. *Semantic versioning boils down to:*

- a) PATCH level *0.0.x*: *changes for implementation level detail changes, such as small bug fixes*
- b) MINOR level *0.x.0* *changes for any backwards compatible API changes, such as new functionality/features*
- c) MAJOR level *x.0.0* *changes for backwards incompatible API changes, such as changes that will break existing users code if they update*

Ejercicio

Ejercicio 90.12.1. ¿Porqué el cambio en el `pop` produce un cambio en el MAJOR level y el cambio en el `push` sólo produce un cambio en el MINOR level en este CHANGELOG?

Version 0.0.1: The initial Stack class is released.
Version 0.0.2: Switched to a linked list implementation because it is cooler.
Version 0.1.0: Added a depth method.
Version 1.0.0: Added top and made pop return nil (pop used to return the old top item).
Version 1.1.0: push now returns the value pushed (it used to return nil).
Version 1.1.1: Fixed a bug in the linked list implementation.
Version 1.1.2: Fixed a bug introduced in the last fix.

Véase

1. Véase en <http://semver.org/> la definición completa del estandard

90.13. Declaring dependencies: Runtime vs. Development

RubyGems provides two main types of dependencies: runtime and development.

1. Runtime dependencies are what your gem needs to work (such as rails needing activesupport).
2. Development dependencies are useful for when someone wants to make modifications to your gem.

`gem install -dev`

When you specify development dependencies, another developer can run

```
gem install --dev your_gem
```

and RubyGems will grab both sets of dependencies (runtime and development).

Typical development dependencies include test frameworks (like rspec) and build systems (like rake).

gemspec: add_runtime_dependency y add_development_dependency

Setting dependencies

in your gemspec is easy.

Just use add_runtime_dependency and add_development_dependency:

```
Gem::Specification.new do |s|
  s.name = "hola"
  s.version = "2.0.0"
  s.add_runtime_dependency "daemons",
    ["= 1.1.0"]
  s.add_development_dependency "bourne",
    [">= 0"]
```

gem "wirble", :groups => [:development, :test] *Inside the Gemfile we can also specify groups. Each gem may specify membership in one or more groups. Any gem that does not specify membership in any group is placed in the default group.*

```
gem "rspec", :group => :test
gem "wirble", :groups => [:development, :test]
```

bundle install --without development

then we can run bundle install --without development test to omit some groups.

90.14. Optimistic and Pessimistic Version Constraint

If your gem properly follows semantic versioning with its versioning scheme, then other Ruby developers can take advantage of this when choosing a version constraint to lock down your gem in their application.

Let's say the following releases of a gem exist:

```
Version 2.1.0 | Baseline
Version 2.2.0 | Introduced some new (backward compatible) features.
Version 2.2.1 | Removed some bugs
Version 2.2.2 | Streamlined your code
Version 2.3.0 | More new features (but still backwards compatible).
Version 3.0.0 | Reworked the interface. Code written to version 2.x might not work.
```

Someone who wants to use your gem has determined that

1. *version 2.2.0 works with their software,*
2. *but version 2.1.0 doesn't have a feature they need.*
3. *Adding a dependency in a gem (or a Gemfile from Bundler) might look like:*

```
# gemspec
spec.add_runtime_dependency 'library',
  '>= 2.2.0'
```

or

```
# bundler
gem 'library', '>= 2.2.0'
```

This is an optimistic version constraint.

It's saying that all changes from 2.x on will work with my software, but for version 3.0.0 this may not be true.

4. The alternative here is to use pessimistic version constraint.
This explicitly excludes the version that might break your code.

```
# gemspec
spec.add_runtime_dependency 'library',
['>= 2.2.0', '< 3.0']
```

or

```
# bundler
gem 'library', '>= 2.2.0', '< 3.0'
```

5. RubyGems provides a shortcut for this, commonly known as the twiddle-wakka:

```
# gemspec
spec.add_runtime_dependency 'library',
'~> 2.2'
```

```
# bundler
gem 'library', '~> 2.2'
```

Notice that we dropped the PATCH level of the version number.

6. Had we said `~> 2.2.0`, that would have been equivalent to `[>= 2.2.0, < 2.3.0]`.
7. If you want to allow use of newer backwards-compatible versions² but need a specific bug fix you can use a compound requirement:

```
# gemspec
spec.add_runtime_dependency 'library', '~> 2.2', '>= 2.2.1'
```

```
# bundler
gem 'library', '~> 2.2', '>= 2.2.1'
```

8. The important note to take home here is to be aware others will be using your gems, so guard yourself from potential bugs/failures in future releases by using `~>` instead of `>=` if at all possible.

90.15. Controlando nuestras dependencias: bundle update, bundle outdated

bundle outdated, bundle update Run `bundle outdated` to print a list of gems that could be upgraded.

If you want to upgrade a specific gem, run

```
bundle update GEM
```

or run

```
bundle update
```

² In computing, a product or technology is backward compatible or downward compatible if it can work with input generated by an older product or technology such as a legacy system.

to update everything. After the update finishes, make sure all your tests pass before you commit your new Gemfile.lock!

`bundle update` won't just install gems - *it will try to upgrade every single gem in your bundle.* That's usually a disaster unless you really meant to do it.

The `update` command is for when gems you use has been updated, and you want your bundle to have the newest version that your Gemfile will allow

```
[~/ruby/faye]$ bundle help outdated
```

Usage:

```
bundle outdated [GEM]
```

Options:

```
--pre=Check for newer pre-release gems  
--source=Check against a specific source  
--local=Do not attempt to fetch gems remotely and use the gem cache instead  
--no-color=Disable colorization in output  
-V, [--verbose=Enable verbose output mode]
```

Description:

Outdated lists the names and versions of gems that have a newer version available in the given source. Calling outdated with [GEM [GEM]] will only check for newer versions of the given gems. Prerelease gems are ignored by default. If your gems are up to date, Bundler will exit with a status of 0. Otherwise, it will exit 1.

```
[~/ruby/faye]$ ls -l Gemfile*  
-rw-r--r-- 1 casiano staff 267 15 nov 18:01 Gemfile  
-rw-r--r-- 1 casiano staff 2373 15 nov 18:02 Gemfile.lock  
[~/ruby/faye]$ bundle outdated  
Fetching gem metadata from https://rubygems.org/.....  
Fetching gem metadata from https://rubygems.org/..  
Resolving dependencies...
```

Outdated gems included in the bundle:

```
* em-websocket (0.5.0 > 0.3.8)  
* rainbows (4.5.0 > 4.4.3)
```

Dependency Graph: `bundle viz`

Bundler uses your Gemfile to create what is technically called a dependency graph, where there are many gems that have various dependencies on each other.

It can be pretty cool to see that dependency graph drawn as a literal graph, and that's what the `bundle viz` command does.

You need to install GraphViz and the ruby-graphviz gem:

```
$ brew install graphviz # for mac os X. apt-get for Linux  
$ gem install ruby-graphviz  
$ bundle viz
```

Once you've done that, though, you get a pretty picture that's a lot of fun to look at. Here's the graph for a Gemfile:

Gemnasium

Gemnasium <https://gemnasium.com> is a service that learns your gem dependencies, listens for new versions of those gems and notifies you when they're released.

Gemnasium sends you an email whenever a new version is released for a gem that you use. You can turn off notifications for particular gems or for entire repositories, or choose to receive them in a digest email on a daily or weekly basis.

Gemnasium represents the statuses of your dependencies with three colors:

1. green =*dependency is up to date with latest version available*
2. yellow =*there is at least one newer stable version available*
3. red =*the dependency is behind a security fix or an important update (broken API, deprecation, ...)*

Gemnasium looks the Gemfile.lock files. If your repository has one, Gemnasium will use it. They're informational and easily parsed with a little help from Bundler.

Unfortunately, it's not always wise to commit your Gemfile.lock. In those cases, Gemnasium looks for a Gemfile instead.

If all else fails, Gemnasium looks for a gemspec file in your repository's root directory.

If any of these files exists, Gemnasium parses them to determine your repository's gem dependencies.

90.16. Local Git Repos

Bundler also allows you to work against a git repository locally instead of using the remote version. This can be achieved by setting up a local override:

```
bundle config local.GEM_NAME /path/to/local/git/repository
```

For example, in order to use a local sinatra repository, a developer could call:

```
[~/sinatra/sinatra-streaming/intro-streaming(master)]$ bundle config local.sinatra /Users/casiano/sinatra-streaming/intro-streaming  
[~/sinatra/sinatra-streaming/intro-streaming(master)]$ bundle update  
Fetching gem metadata from https://rubygems.org/.....  
Resolving dependencies...  
Using columnize 0.8.9  
Using debugger-linecache 1.2.0  
Using slop 3.6.0  
Using byebug 3.5.1  
Using coderay 1.1.0  
Using method_source 0.8.2  
Using pry 0.10.1  
Using pry-byebug 2.0.0  
Using rack 1.5.2  
Using rack-protection 1.5.3  
Using tilt 2.0.1 (was 1.4.1)  
Using sinatra 1.4.5 from git://github.com/sinatra/sinatra.git (at /Users/casiano/local/src/ruby/sinatra-src  
Using bundler 1.6.2  
Your bundle is updated!
```

```
[~/sinatra/sinatra-streaming/intro-streaming(master)]$ bundle show sinatra  
/Users/casiano/local/src/ruby/sinatra/sinatra-src
```

90.17. Badges

You might have seen some nice green badges in your favorite GitHub repository, indicating whether the tests associated with the code passed or not. Adding these to the README.md is straightforward enough:

```
[! [Build Status] (https://travis-ci.org/Integralist/Sinderella.png?branch=master)] (https://travis-ci.org/Integralist/Sinderella)  
[! [Gem Version] (https://badge.fury.io/rb/sinderella.png)] (http://badge.fury.io/rb/sinderella)  
[! [Coverage Status] (https://coveralls.io/repos/Integralist/Sinderella/badge.png)] (https://coveralls.io/repos/Integralist/Sinderella)
```

- The first badge is provided by Travis CI, which you can read more about in the documentation.
- The second is provided by RubyGems. You'll notice on your gem's page a [badge](#) link, which provides the required code and format (in this case, in Markdown format).
- The third is provided by Coveralls. When you visit your repository page in the Coveralls application, you'll see a link to [README badge](#); from there, you can select the relevant format.

90.18. Preparando la Documentación

Véase la sección Documentado el Código: RDoc, Yard, UML 88.

90.18.1. Inch

Inch is a documentation measurement tool for Ruby that was created to help people document their code.

Es posible añadir un badge que mida la calidad de la documentación. Véase <http://inch-pages.github.io/>.

90.19. Enlaces Relacionados

- How To Build A Ruby Gem With Bundler, Test-Driven Development, Travis CI And Coveralls, Oh My! By Mark McDonnell
- Hack your bundle for fun and profit by André Arko
- gemwhisperer posts Twitter updates about gems
- Polishing Rubies
- New Gem with Bundler
- Removing a published RubyGem
- Mitchell Hashimoto: BASIC RubyGem Development
- Mitchell Hashimoto: INTERMEDIATE Intermediate RubyGem Development
- Gemnasium is an online tool to monitor your project dependencies. It works with Rubygems and NPM: <https://gemnasium.com>

Capítulo 91

Herramientas para la Programacion en Pares

91.1. Introduction to Pair Programming

- *Pair Programming by Thomas Sundberg*
- *Ben Orenstein - Live Coding with Ben - Ancient City Ruby 2013*
- *Remote Pairing. Collaborative Tools for Distributed Development by Joe Kutner The Pragmatic Bookshelf. 2013.*

Why should you do pair programming? There are a number of reasons:

- *Solving the right problem – two persons that has understood the problem has a better chance to solve the right problem and not the problem one person has misunderstood.*
- *Quality – the co-driver will assist the driver to develop the algorithm and act as a guard against mistakes.*
- *Knowledge spread – both participants in the pair will have a great knowledge about the solution when it has been developed in co-operation.*
- *Constant code review – instead of having code reviews from time to time you have them all the time. The co-driver will constantly review the code that is being written and suggest improvements.*
- *Reduced costs – bringing new developers into the team is expensive. If they are brought in using pair programming then they will learn fast and can soon be productive. A new developer is productive*
 - *within minutes instead of weeks or even months.*
- *Single point of failure – what happens if only one person knows about a certain area of the code and chooses to quit or has an accident on his way to work? Keep your bus count higher than 1. The bus*
 - *count is the number of buses needed to erase the knowledge your team has.*
- *Spread knowledge about the tools being used – every developer knows something different about the development environment and will therefore constantly learn new things from his partner.*
- *Discipline – it is really hard not to focus at the task at hand while pair programming. You will therefore get more work done.*

- *Focus – your partner will ensure that you focus on the task at hand and not a random page on the web.*
- *Direction – your partner will act as guide to ensure that you don't stray away from the proper path by accident.*

91.1.1. Remote pair programming

- *Pair Programming by Thomas Sundberg*

Pair programming is best done with the pair at the same location. This is sometimes not possible. How can pair programming be done remote?

One simple setup is a voice channel using Skype, screen sharing using TeamViewer and file sharing using Github.

So by talking to your partner over Skype and at the same time share the desktop on one of your computers you will be able to pair program remote. File sharing may be nice but is actually not necessary. One problem I have experienced is when you have a need for scribbling a model on a piece of paper. That is hard when your partner is somewhere else. You can use tools like Google draw and scribble, but it is really difficult and a lot harder compared to a quick drawing on the back of an envelope.

Has this setup been tested? Yes it has. I have done remote pair programming from Finland and Sweden with a friend, Alexandru Bolboaca, in Romania. It worked very well. Our only problem was that we did it late at Friday evenings, our families had some opinions after a few hours and we were tired after a week of work.

91.2. tmux

- *tmux Quick Start*
- *Pair Programming with Tmux Screencast por Flaviu Simihaian*
- *Remote Pairing With SSH, Tmux, and Vim*
- *Some people call me "the remote pairing guy"... por Evan Light, The Hippy Hacker*
- *TMUX – The Terminal Multiplexer (Part 1)*
- *TMUX – The Terminal Multiplexer (Part 2)*
- *LA Ruby Conf 2013 - Impressive Ruby Productivity with Vim and Tmux by Chris Hunt*
 - *dot-files*
 - *ctrlp Full path fuzzy file, buffer, mru, tag, ... finder for Vim*

91.2.1. Utilidades para Pair Programming con tmux

- *github-auth allows you to quickly pair with anyone who has a GitHub account by adding and removing their public ssh keys from your authorized_keys file.*

91.3. MadEye

- *<http://madeye.io/>*
- *James Gill & Mike Risse: Madeye - Meteor Devshop 2 Lightning Talk*

91.4. Screenhero

Screenhero is for code review and pair programming. With simultaneous control, both sides can switch between driving and navigating.

- <http://screenhero.com/>

Capítulo 92

Cloud9

92.1. Introducción: Cloud IDEs y Cloud9

A cloud IDE is a web-based integrated development platform (IDE).

An IDE is a programming environment that has been packaged as an application, typically consisting of a code editor, a compiler, a debugger, and a graphical user interface (GUI) builder. Frequently, cloud IDEs are not only cloud-based but also designed for the creation of cloud apps. However, some cloud IDEs are optimized for the creation of native apps for smartphones, tablets and other mobile devices.

The benefits of cloud IDEs include accessibility from anywhere in the world, from any compatible device; minimal-to-nonexistent download and installation; and ease of collaboration among geographically dispersed developers.

The emergence of HTML 5 is often cited as a key enabler of cloud IDEs because that standard supports browser-based development. Other key factors include the increasing trends toward mobility, cloud computing and open source software. (See [icloud IDE definition](#))

- *The top 20 online coding tools en <http://www.netmagazine.com/>*
- *Review: 4 killer cloud IDEs Browser-based JSFiddle, Icenium, Cloud9, and Codenvy stretch from client-side JavaScript and HTML5 to server-side Java and Web stacks por Peter Wayner. InfoWorld*
- *<https://c9.io>*
- *Solving the Two Week Problem by Developing in the Cloud with Cloud9 IDE and OpenShift*

92.2. Instalación local

- *ajaxorg / cloud9 en GitHub*

Install:

```
git clone https://github.com/ajaxorg/cloud9.git  
cd cloud9  
npm install
```

The above install steps create a cloud9 directory with a bin/cloud9.sh script that can be used to start Cloud9:

bin/cloud9.sh

Optionally, you may specify the directory you'd like to edit:

```
bin/cloud9.sh -w ~/git/myproject
```

Cloud9 will be started as a web server on port -p 3131, you can access it by pointing your browser to: <http://localhost:3131>

92.3. Ruby en Cloud9

- *Ruby en cloud9*
- *Running Ruby Sinatra on Cloud 9*

The correct command to run a Sinatra application on Cloud 9 is:

```
ruby app.rb -p $PORT -o $IP
```

92.4. Programación en Parejas y Cloud9

- *Cloud9 y Pair Programming*
- *The New Cloud9*

92.5. Rails en Cloud9

Cloud9 IDE also supports the ability to run a Rails application. The rails command is only available on the terminal.

To run a rails application:

- *Open the terminal and type gem install rails*

```
Welcome to Cloud9 IDE
casiano@demo_app:~/514754 $ gem install rails
Fetching: i18n-0.6.1.gem (100%)
Fetching: multi_json-1.7.4.gem (100%)
Fetching: activesupport-3.2.13.gem (100%)
...
...
```

Despues de la instalación tenemos:

```
casiano@demo_app:~/514754 $ rails --version
Rails 3.2.13
```

- *When the bundle is done, type rails new.*

```
casiano@demo_app:~/514754 $ rails new demo_app
create
create  README.rdoc
create  Rakefile
...
run bundle install
...
```

- *Edit your database configuration in configs/database.yml*
- *Cambiamos al directorio y arrancamos rails tecleando: rails s -b \$IP -p \$PORT:*

```
john@demo_app:~/514754 $ cd demo_app/
john@demo_app:~/514754/demo_app $ echo $IP
127.6.67.1
john@demo_app:~/514754/demo_app $ echo $PORT
8080
```

```

john@demo_app:~/514754/demo_app $ rails s -b $IP -p $PORT
=> Booting WEBrick
=> Rails 3.2.13 application starting in development on http://127.6.67.1:8080
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2013-05-31 12:48:13] INFO  WEBrick 1.3.1
[2013-05-31 12:48:13] INFO  ruby 1.9.3 (2012-11-10) [x86_64-linux]
[2013-05-31 12:48:13] INFO  WEBrick::HTTPServer#start: pid=6788 port=8080
Cloud9 Your application is running at http://demo_app.john.c9.io

```

92.6. Cloud9 y GitHub: How do I push my Cloud9 project to GitHub

Véase el post por Daniela Gavidia.

- It all depends on how you created your project. First, have you linked your Cloud9 account with your GitHub account? You can check that by going to the Dashboard and seeing if it is activated under Add-on Services (you can see how to link your GitHub account here).
- Now, if the Cloud9 account and the GitHub account are linked, you can just create a new repository in GitHub and then just clone from url in Cloud9 (choose Clone from url in the Dashboard). This will create a project in Cloud9 that is already linked to the repository on GitHub, so you should be able to go to the console and just do a git push to update the GitHub repo.
- Alternatively, you can create the Cloud9 project first (by going to the Dashboard and choosing Create a New Project). The new Cloud9 project is not yet linked to a GitHub repo. The next step is to go to GitHub and create a repo for your project. Then, follow the instructions on GitHub (we also have some instructions in the README.md file included in any new Cloud9 project). In short (in the Cloud9 console):

- Add a git remote in the Cloud9 console. Should look like this (replace the git url with your repo url):

```
git remote add origin git@github.com:C9Support/testPush.git
```

Add files and commit them:

```
git add .
git commit -m "First commit"
```

- Push to github. You'll get an error:

```
git push -u origin master
Warning: Permanently added 'github.com,207.97.227.239' (RSA) to the list of known hosts.
Permission denied (publickey).
fatal: The remote end hung up unexpectedly
```

- You can find your SSH key on your Cloud 9 Dashboard, on the right of the screen under Account Settings. Copy that and paste it into a new SSH entry on your GitHub account at <https://github.com/settings/ssh>.
- Now everything goes smoothly:

```
yoenc9@demo_app:~/514754/demo_app (master) $ git push origin master
Counting objects: 104, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (93/93), done.
Writing objects: 100% (104/104), 32.80 KiB, done.
```

```
Total 104 (delta 8), reused 0 (delta 0)
To git@github.com:yoengithub/c9-demo-app.git
 * [new branch]      master -> master
```

92.7. Deploy your app to Heroku using the Cloud9 console

Véase el post de Daniela Gavidia.

You can use the Cloud9 console to deploy your app. To do so, please follow these instructions:

1. Create a Heroku deploy server. You can do that using the Cloud9 integration (choose *Create new* in the *Add deploy target* window; el icono del "globito"):

```
https://skitch.com/c9support/85qpk/hello-world-cloud9
```

2. Verify that the deploy target was created by logging in to your Heroku account (at heroku.com) and verifying that the deploy target exists under My Apps

3. In the Cloud9 console, add the Heroku deploy target as a remote with git:

```
git remote add heroku-target git@heroku.com:c9testtheworld.git|
```

where `heroku-target` is the name you want to give to your deploy target (you can name it whatever you want) and

`git@heroku.com:c9testtheworld.git`

is the git repo in Heroku for your deploy target. You can get the git repo by going to

[Heroku.com](#) → My Apps,

clicking on General info for the app.

4. Create a `package.json` file and a `Procfile` for your app. For more info on the format of these files, see the link below. You only need to add these two files. Ignore the other steps at the link: <https://devcenter.heroku.com/articles/nodejs>

Please make sure you use the correct name in the `Procfile`. On the link they use `web.js`, but you probably have `server.js` as your main node file.

5. Commit all your files with git:

```
git add .
git commit -m "Adding all my files"
```

6. Push to Heroku:

```
git push heroku-target master
```

If pushing the file succeeds, you will get a link to the deployed app. Otherwise, you should get some informative error messages from Heroku.

Véase también

- Deploy your application to Heroku en [cloud9 IDE/Forums/Using Cloud9 IDE/How To](#)
- How can I push changes directly from Cloud9 IDE to Heroku?
- How to rake database to Heroku on cloud9 en [Stackoverflow](#)

Capítulo 93

OAuth

93.1. Introduction to OAuth

OAuth 2 is rapidly becoming a preferred authorization protocol, and is used by major service providers such as Google, Facebook and Github.

An excellent introduction to Oauth is in the article [Introduction to OAuth \(in plain English\)](#). by Ron Sobers

Valet Key for the Web

Many luxury cars come with a [valet key](#). It is a special key you give the [parking attendant](#) and unlike your regular key, will only allow the [car](#) to be driven a short distance while [blocking access to the trunk and the onboard cell phone](#).

Regardless of the restrictions the valet key imposes, the idea is very clever. You give someone limited access to your car with a special key, while using another key to unlock everything else.

As the web grows, more and more sites rely on distributed services and cloud computing:

- a photo lab printing your Flickr photos,
- a social network using your Google address book to look for friends, or
- a third-party application utilizing APIs from multiple services.

The problem is, in order for these applications to access user data on other sites, they ask for usernames and passwords. Not only does this require exposing user passwords to someone else – often the same passwords used for online banking and other sites – it also provides these application unlimited access to do as they wish. They can do anything, including changing the passwords and lock users out.

OAuth provides a method for [users](#) (you) to grant [third-party](#) (parking attendant) access to their [resources](#) (your luxury car) without sharing their [passwords](#) (the key of your car). It also provides a way to [grant limited access](#) (in scope, duration, etc. the equivalent of not having access to the trunk or the onboard cell phone).

For example,

- a web [user](#) (resource owner) can grant a
- [printing service \(client/consumer\)](#)
- [access to her private photos \(partial resource\)](#)
- [stored at a photo sharing service \(service provider\) \(server\),](#)
- [without sharing her username and password with the printing service.](#)

Instead, she authenticates directly with the photo sharing service which issues the printing service delegation-specific credentials.

In OAuth, the **client** requests access to resources controlled by the **resource owner** and hosted by the **resource server**, and is issued a different set of credentials than those of the resource owner.

Instead of using the resource owner's credentials to access protected resources, the **client** obtains an **access token** – a string denoting a specific scope, lifetime, and other access attributes.

Access tokens are issued to third-party clients by an **authorization server** with the approval of the **resource owner**.

The client uses the **access token** to access the **protected resources** hosted by the **resource server**.

1. The OAuth 2.0 Authorization Framework proposed standard document

Roles in OAuth

OAuth defines four roles:

1. resource owner

An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

2. resource server

The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

3. client

An application making protected resource requests on behalf of the resource owner and with its authorization.

The term "**client**" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

4. authorization server

The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

From an application developer's point of view, a service's API fulfills both the resource and authorization server roles.

The authorization server may be the same server as the resource server or a separate entity. A single authorization server may issue access tokens accepted by multiple resource servers.

Abstract Flow

1. The application requests authorization to access service resources from the user
2. If the user authorized the request, the application receives an authorization grant
3. The application requests an access token from the authorization server (API) by presenting authentication of its own identity, and the authorization grant
4. If the application identity is authenticated and the authorization grant is valid, the authorization server (API) issues an access token to the application. Authorization is complete.
5. The application requests the resource from the resource server (API) and presents the access token for authentication
6. If the access token is valid, the resource server (API) serves the resource to the application

The actual flow of this process will differ depending on the authorization grant type in use, but this is the general idea.

Application Registration

Before using OAuth with your application, you must register your application with the service.

This is done through a registration form in the developer or API portion of the service's website, where you will provide the following information (and probably details about your application):

Application Name

Application Website

Redirect URI or Callback URL

The redirect URI is where the service will redirect the user after they authorize (or deny) your application, and therefore the part of your application that will handle authorization codes or access tokens.

Client ID and Client Secret

Once your application is registered, the service will issue client credentials in the form of a client identifier and a client secret.

1. The Client ID is a publicly exposed string that is used by the service API to identify the application, and is also used to build authorization URLs that are presented to users.
2. The Client Secret is used to authenticate the identity of the application to the service API when the application requests to access a user's account, and must be kept private between the application and the API.

Authorization Grant

In the Abstract Protocol Flow above, the first four steps cover obtaining an authorization grant and access token.

The authorization grant type depends on the method used by the application to request authorization, and the grant types supported by the API.

OAuth 2 defines four grant types, each of which is useful in different cases:

- Authorization Code: used with server-side Applications
- Implicit: used with Mobile Apps or Web Applications (applications that run on the user's device)
- Resource Owner Password Credentials: used with trusted Applications, such as those owned by the service itself
- Client Credentials: used with Applications API access

OAuth 2.0 is a version of the protocol which introduces different flows for web, mobile, and desktop applications. It also has the notion of token expiration (similar to cookie expiration), requires SSL, and reduces the complexity for developers by no longer requiring signing.

Grant Type: Authorization Code

The authorization code grant type is the most commonly used because it is optimized for server-side applications, where source code is not publicly exposed, and Client Secret confidentiality can be maintained.

This is a redirection-based flow, which means that the application must be capable of interacting with the user-agent (i.e. the user's web browser) and receiving API authorization codes that are routed through the user-agent.

Now we will describe the authorization code flow:

1. User Authorization Request

a) **The user is redirected to the service provider.**

This is the scary part. If the consumer were super-shady Evil Co. it could pop up a window that looked like the service provider but was really phishing for your username and password. Always be sure to verify that the URL you're directed to is actually the service provider

b) **Authorization Code Link**

First, the user is given an authorization code link that looks like the following:

```
https://cloud.digitalocean.com/v1/oauth/authorize?response_type=code&
client_id=CLIENT_ID&
redirect_uri=CALLBACK_URL&scope=read
```

Here is an explanation of the link components:

- `https://cloud.digitalocean.com/v1/oauth/authorize`: the API authorization endpoint
- `client_id=CLIENT_ID`: the application's client ID (how the API identifies the application)
- `redirect_uri=CALLBACK_URL`: where the service redirects the user-agent after an authorization code is granted
- `response_type=code`: specifies that your application is requesting an authorization code grant
- `scope=read`: specifies the level of access that the application is requesting

c) **User Authorizes Application**

When the user clicks the link, they must first log in to the service, to authenticate their identity (unless they are already logged in).

Then they will be prompted by the service to authorize or deny the application access to their account.

Here is an example authorize application prompt: This particular screenshot is of DigitalOcean's authorization screen, and we can see that Thedropletbook App is requesting authorization for read access to the account of manicas@digitalocean.com

2. **Application Receives Authorization Code** *If the user clicks Authorize Application, the service redirects the user-agent to the application redirect URI, which was specified during the client registration, along with an authorization code.*

The redirect would look something like this (assuming the application is dropletbook.com):

```
https://dropletbook.com/callback?code=AUTHORIZATION_CODE
```

3. **Application Requests Access Token** *The application requests an access token from the API, by passing the authorization code along with authentication details, including the client secret, to the API token endpoint.*

Here is an example POST request to DigitalOcean's token endpoint:

```
https://cloud.digitalocean.com/v1/oauth/token?client_id=CLIENT_ID&
client_secret=CLIENT_SECRET&
grant_type=authorization_code&
code=AUTHORIZATION_CODE&
redirect_uri=CALLBACK_URL
```

4. **Application Receives Access Token**

If the authorization is valid, the API will send a response containing the access token (and optionally, a refresh token) to the application.

The entire response will look something like this:

```
{
  "access_token": "ACCESS_TOKEN",
  "token_type": "bearer",
  "expires_in": 2592000,
  "refresh_token": "REFRESH_TOKEN",
  "scope": "read",
  "uid": 100101,
  "info": {"name": "Mark E. Mark", "email": "mark@thefunkybunch.com"}
}
```

Now the application is authorized! It may use the token to access the user's account via the service API, limited to the scope of access, until the token expires or is revoked.

If a refresh token was issued, it may be used to request new access tokens if the original token has expired

Once the application has an access token, it may use the token to access the user's account via the API, limited to the scope of access, until the token expires or is revoked.

Here is an example of an API request, using curl. Note that it includes the access token:

```
curl -X POST -H "Authorization: Bearer ACCESS_TOKEN" "https://api.digitalocean.com/v2/$OBJECT"
```

Assuming the access token is valid, the API will process the request according to its API specifications.

If the access token is expired or otherwise invalid, the API will return an invalid_request error.

Véase

- *Introduction to OAuth (In Plain English) by Rob Sobers*
- *An Introduction to OAuth 2 by Mitchell Anicas*
- *Nacho Coloma: Our love-hate relationship with OAuth*

93.2. Google Developers Console

93.2.1. Managing projects and applications

A project consists of

1. *a set of applications,*
2. *along with activated APIs,*
3. *Google Cloud resources, and*
4. *the team and billing information associated with those resources.*

Credentials such as API keys are specific to an application rather than to a project.

However, all applications within a given project use the same branding information (logo, email address, etc.) on their user consent screen, as described in Setting up OAuth 2.0.

Applications within a project also share

1. *activated APIs,*
2. *permissions, and*
3. *billing information.*

Managing project members

If you create a project, you have owner-level permissions and can grant owner-level permissions to other project members. Those with owner-level permissions are **project owners**.

Only project owners can add and remove other project members and edit their permission levels. Project owners can share a project with an email address that represents a group, but every project must have at least one project member that is an individual, not a group.

To manage project members, do the following:

1. Visit the Google Developers Console
2. Select a project, or create a new one.
3. In the sidebar on the left, select Permissions.
4. To add a team member or group, select Add Member.
 - You must provide an **email address** that is associated with a Google account.
 - If the email address belongs to an individual, an invitation flow is triggered, and the new project member must accept the invitation before they can access the project.
 - If the email address belongs to a group, the group is added right away, with no invitation step.
5. To change the permission setting for a project member, click the dropdown box in the Permission column and select a new permission level. The new permission level is saved automatically.
6. To delete a project member, click the trash icon to the right of the project member's permission setting.

93.2.2. Keys, access, security, and identity

Each request to an API that is represented in the Google Developers Console must include a **unique identifier**.

Unique identifiers enable the Developers Console to tie requests to specific projects in order to

- monitor traffic,
- enforce quotas, and
- handle billing.

Google supports two mechanisms for creating unique identifiers:

1. OAuth 2.0 client IDs

For applications that use the OAuth 2.0 protocol to call Google APIs, you can use an OAuth 2.0 client ID to generate an access token. The token contains a unique identifier.

2. API keys

An API key (either a server key or a browser key) is a unique identifier that you generate using the Developers Console. Using an API key does not require user action or consent. **API keys do not grant access to any account information, and are not used for authorization.**

Use an API key when your application is running on a server and accessing one of the following kinds of data:

- Data that the data owner has identified as public, such as a public calendar or blog.

- Data that is owned by a Google service such as Google Maps or Google Translate. (Access limitations may apply.)
- If you are only calling APIs that do not require user data, such as the Google Custom Search API, then API keys might be simpler to use than OAuth 2.0 access tokens. However, if your application already uses an OAuth 2.0 access token, then there is no need to generate an API key as well. Google ignores passed API keys if a passed OAuth 2.0 access token is already associated with the corresponding project.

93.3. OmniAuth gem: Standardized Multi-Provider Authentication for Ruby

- *OmniAuth*

OmniAuth is a library that standardizes multi-provider authentication for web applications. Any developer can create strategies for OmniAuth that can authenticate users via disparate systems.

OmniAuth strategies have been created for everything from Facebook to LDAP.

To use OmniAuth, you need only

1. *to redirect users to /auth/:provider, where :provider is the name of the strategy (for example, developer or twitter).*
2. *From there, OmniAuth will take over and take the user through the necessary steps to authenticate them with the chosen strategy.*
3. *Once the user has authenticated, OmniAuth sets a special hash called the Authentication Hash on the Rack environment of a request to /auth/:provider/callback.*
4. *This hash contains as much information about the user as OmniAuth was able to glean from the utilized strategy.*
5. *You should set up an endpoint in your application that matches to the callback URL and then performs whatever steps are necessary for your application.*

Getting Started

To use OmniAuth in a project with a Gemfile, just add each of the [strategies](#) you want to use individually:

```
gem 'omniauth-github'
gem 'omniauth-openid'
```

Now you can use the `OmniAuth::Builder` Rack middleware [to build up your list of OmniAuth strategies for use in your application](#):

Para saber mas sobre Rack y sobre Middlewares Rack, véanse las secciones

- *Rack en ??,*
- *Middleware y la Clase Rack::Builder en ?? y*
- *La Estructura de una Aplicación Rack en ??*

```
use OmniAuth::Builder do
  provider:github, ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']
  provider:openid, :store => OpenID::Store::Filesystem.new('/tmp')
end
```

By default, OmniAuth will return auth information to the path /auth/:provider/callback inside the Rack environment.

In Sinatra, for example, a callback might look something like this:

```
# Support both GET and POST for callbacks
%w(get post).each do |method|
  send(method, "/auth/:provider/callback") do
    env['omniauth.auth'] # => OmniAuth::AuthHash
  end
end
```

Also of note, by default, if user authentication fails on the provider side, OmniAuth will catch the response and then redirect the request to the path /auth/failure, passing a corresponding error message in a parameter named `message`.

You may want to add an action to catch these cases. Continuing with the previous Sinatra example, you could add an action like this:

```
get '/auth/failure' do
  flash[:notice] = params[:message] # if using sinatra-flash or rack-flash
  redirect '/'
end
```

Strategies

In this link we can find a list of the strategies that are available for OmniAuth: [List of Strategies for Omniauth](#).

93.3.1. Auth Hash Schema

OmniAuth is a flexible authentication system utilizing Rack middleware.

OmniAuth will always return a hash of information after authenticating with an external provider in the Rack environment under the key `omniauth.auth`.

This information is meant to be as normalized as possible, so the schema below will be filled to the greatest degree available given the provider upon authentication. Fields marked required will always be present.

- - **provider** (required) The provider with which the user authenticated (e.g. `twitter` or `facebook`)
 - **uid** (required) An identifier unique to the given provider, such as a **Twitter user ID**. Should be stored as a string.
 - **info** (required) A hash containing information about the user
 - **name** (required) The best display name known to the strategy. Usually a concatenation of first and last name, but may also be an arbitrary designator or nickname for some strategies
 - **email** The e-mail of the authenticating user. Should be provided if at all possible (but some sites such as Twitter do not provide this information)
 - **nickname** The username of an authenticating user (such as your @-name from Twitter or GitHub account name)
 - **first_name**
 - **last_name**
 - **location** The general location of the user, usually a city and state.
 - **description** A short description of the authenticating user.

- **image** A URL representing a profile image of the authenticating user. Where possible, should be specified to a square, roughly 50x50 pixel image.
- **phone** The telephone number of the authenticating user (no formatting is enforced).
- **urls** A hash containing key value pairs of an identifier for the website and its URL. For instance, an entry could be

```
"Blog" => "http://intridea.com/blog"
```
- **credentials** If the authenticating service provides some kind of access token or other credentials upon authentication, these are passed through here.
- **token** Supplied by OAuth and OAuth 2.0 providers, the access token.
- **secret** Supplied by OAuth providers, the access token secret.
- **extra** Contains extra information returned from the authentication provider. May be in provider-specific formats.
- **raw_info** A hash of all information gathered about a user in the format it was gathered. For example, for Twitter users this is a hash representing the JSON hash returned from the Twitter API.

Ejemplos

- Omniauth Sinatra Example (Twitter y OpenID)
- Omniauth Sinatra Gist Example: Facebook, Twitter, GitHub

Documentación de la Gema

- *API doc: OmniAuth: Standardized Multi-Provider Authentication*
- *Module: OmniAuth*
- *Esta clase contiene información sobre el usuario. Class: OmniAuth::AuthHash::InfoHash*

Véase

- *Blog Post: ColdFusion and OAuth part 3- Google authentication. Raymond Camden*

93.4. OmniAuth OAuth2 gem

- *omniauth-oauth2*

This gem contains a generic OAuth2 strategy for OmniAuth.

It is meant to serve as a building block strategy for other strategies and not to be used independently, since it has no inherent way to gather uid and user info.

93.5. The gem omniauth-google-oauth2

Introducción The

gem omniauth-google-oauth2 provides a strategy to authenticate with Google via OAuth2 in OmniAuth.

Get your API key at:

<https://code.google.com/apis/console/>

Note the Client ID and the Client Secret.

For more details, read the Google docs:

<https://developers.google.com/accounts/docs/OAuth2>.

Configuration

You can configure several options, which you pass in to the `provider` method via a hash:

■

- `scope` A comma-separated list of permissions you want to request from the user. See the Google OAuth 2.0 Playground for a full list of available permissions.

Caveats:

- The `userinfo.email` and `userinfo.profile` scopes are used by default. By defining your own scope, you override these defaults. If you need these scopes, don't forget to add them yourself!
- Scopes starting with `https://www.googleapis.com/auth/` do not need that prefix specified. So while you can use the smaller scope `books` since that permission starts with the mentioned prefix, you should use the full scope URL `https://docs.google.com/feeds/` to access a user's docs, for example.
- `prompt` A space-delimited list of string values that determines whether the user is re-prompted for authentication and/or consent. Possible values are:

- `none` No authentication or consent pages will be displayed; it will return an error if the user is not already authenticated and has not pre-configured consent for the requested scopes. This can be used as a method to check for existing authentication and/or consent.
- `consent` The user will always be prompted for consent, even if he has previously allowed access a given set of scopes.
- `select_account` The user will always be prompted to select a user account. This allows a user who has multiple current account sessions to select one amongst them.
- If no value is specified, the user only sees the `authentication page` if he is not logged in
- and only sees the `consent page` the first time he authorizes a given set of scopes.

- `image_aspect_ratio` The shape of the user's profile picture. Possible values are:

- `original` Picture maintains its original aspect ratio.
- `square` Picture presents equal width and height. Defaults to `original`.

- `image_size` The size of the user's profile picture.

The image returned will have width equal to the given value and variable height, according to the `image_aspect_ratio` chosen.

Additionally, a picture with specific width and height can be requested by setting this option to a hash with `width` and `height` as keys.

If only `width` or `height` is specified, a picture whose `width` or `height` is closest to the requested size and requested aspect ratio will be returned.

Defaults to the original width and height of the picture.

- `name` The name of the strategy. The default name is `google_oauth2` but it can be changed to any value, for example `google`. The OmniAuth URL will thus change to `/auth/google` and the `provider` key in the `auth hash` will then return `google`.

- `access_type` Defaults to `offline`, so a refresh token is sent to be used when the user is not present at the browser. Can be set to `online`.

Note that if you need a refresh token, google requires you to also specify the option `prompt: 'consent'`, which is not a default.

- **login_hint** When your app knows which user it is trying to authenticate, it can provide this parameter as a hint to the authentication server. Passing this hint suppresses the account chooser and either pre-fill the email box on the sign-in form, or select the proper session (if the user is using multiple sign-in), which can help you avoid problems that occur if your app logs in the wrong user account. The value can be either an email address or the sub string, which is equivalent to the user's Google+ ID.
- **include_granted_scopes** If this is provided with the value `true`, and the authorization request is granted, the authorization will include any previous authorizations granted to this user/application combination for other scopes. See Google's Incremental Authorization for additional details.

Here's an example of a possible configuration where

- the strategy name is changed,
- the user is asked for extra permissions,
- the user is always prompted to select his account when logging in and
- the user's profile picture is returned as a thumbnail:

```
Rails.application.config.middleware.use OmniAuth::Builder do
  provider :google_oauth2, ENV["GOOGLE_CLIENT_ID"], ENV["GOOGLE_CLIENT_SECRET"],
  {
    :name => "google",
    :scope => "userinfo.email, userinfo.profile, plus.me, http://gdata.youtube.com",
    :prompt => "select_account",
    :image_aspect_ratio => "square",
    :image_size => 50
  }
end
```

Auth Hash Here's an example of an authentication hash available in the callback by accessing `request.env["omniauth.auth"]`:

```
{
  :provider => "google_oauth2",
  :uid => "123456789",
  :info => {
    :name => "John Doe",
    :email => "john@company_name.com",
    :first_name => "John",
    :last_name => "Doe",
    :image => "https://lh3.googleusercontent.com/url/photo.jpg"
  },
  :credentials => {
    :token => "token",
    :refresh_token => "another_token",
    :expires_at => 1354920555,
    :expires => true
  },
  :extra => {
    :raw_info => {
      :id => "123456789",
      :email => "user@domain.example.com",
    }
  }
}
```

```

    :verified_email => true,
    :name => "John Doe",
    :given_name => "John",
    :family_name => "Doe",
    :link => "https://plus.google.com/123456789",
    :picture => "https://lh3.googleusercontent.com/url/photo.jpg",
    :gender => "male",
    :birthday => "0000-06-25",
    :locale => "en",
    :hd => "company_name.com"
  }
}

}

```

93.6. Ejemplos de uso de omniauth

- *alu0100699494/pl-prct09 Práctica de Procesadores de Lenguajes en Sinatra*

```
~/srcPLalus1314/daniel_herzog_cruz/pl0-constant-folding(master)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/alus/1314/daniel_herzog_cruz/pl-prct09
```

- *https://github.com/crguezl/omniauth-google-oauth2-sample*
- *crguezl/ull-etsii-grado-pl-jsoncalc*
- *Omniauth Sinatra Example (Twitter y OpenID) de Intridea*
- *Omniauth Sinatra Gist Example: Facebook, Twitter, GitHub*

93.7. Práctica: Usando OAuth con omniauth

Configure su propia versión del ejemplo de uso de OmniAuth en <https://github.com/crguezl/omniauth-google-oauth2-sample>.

Cree para ello una nueva aplicación en Google's Console con su ID de cliente y su secreto de cliente. Habilite las APIs de Google que vaya a usar.

Despliegue la aplicación en Heroku.

93.8. Using OAuth 2.0 to Access Google APIs

Véase Using OAuth 2.0 to Access Google APIs.

Google APIs use the OAuth 2.0 protocol for authentication and authorization. Google supports common OAuth 2.0 scenarios such as those for web server, installed, and client-side applications.

1. *OAuth 2.0 is a relatively simple protocol. To begin, you obtain OAuth 2.0 credentials from the Google Developers Console.*
2. *See the Video in YouTube Obtaining a developer key for the YouTube Data API v3 and the Analytics API*
3. *Then your client application requests an [access token](#) from the [Google Authorization Server](#), extracts a token from the response, and sends the token to the [Google API](#) that you want to access.*

To get access keys, go to the Google APIs Console and specify your Google Developers Console application's name and the Google APIs it will access. For simple access, Google generates an API key that uniquely identifies your application in its transactions with the Google Auth server.

For authorized access, you must also tell Google your website's protocol and domain. In return, Google generates a client ID. Your application submits this to the Google Auth server to get an OAuth 2.0 access token.

93.9. Google OAuth 2.0 Playground

Google OAuth 2.0 Playground

93.10. Sign-in with Google +

Google+ Sign-in provides the OAuth 2.0 authentication mechanism along with additional access to Google desktop and mobile features.

- *<https://developers.google.com/+/> provides the OAuth 2.0 authentication mechanism along with additional access to Google desktop and mobile features.*
- *Direct access to an authentication service based on the standardized OpenID Connect mechanism is also available. <https://developers.google.com/accounts/docs/OAuth2Login>*

93.11. Revoking Access to an App

Go to <https://security.google.com/settings/security/permissions>

*Or to your configuration <https://www.google.com/settings/personalinfo> and there to **security**. From there go to the section **Account permissions** (Control which apps and websites have access to your account information). Choose the link **View All**.*

- *Remember to kill the session if you want to check that the permit has been effectively removed.*
- *Go to chrome and open a window in **incognito mode***
- *Attempt to access the URL that requires Google Authentication in your service*
- *It has to ask you for permits again*

93.12. Google + API for Ruby

Donde

- *<https://github.com/crguezl/gplus-quickstart-ruby>*
- *[~/sinatra/gplus-quickstart-ruby(master)]\$ pwd -P
/Users/casiano/local/src/ruby/sinatra/gplus-quickstart-ruby
[~/sinatra/gplus-quickstart-ruby(master)]\$ git remote -v
googleplus git@github.com:googleplus/gplus-quickstart-ruby.git (fetch)
googleplus git@github.com:googleplus/gplus-quickstart-ruby.git (push)
origin git@github.com:crguezl/gplus-quickstart-ruby.git (fetch)
origin git@github.com:crguezl/gplus-quickstart-ruby.git (push)*
- *<https://github.com/crguezl/gplus-quickstart-ruby>*
- *<https://developers.google.com/+/quickstart/ruby>*

The app demonstrates:

- Using the Google+ Sign-In button to get an OAuth 2.0 refresh token.
- Exchanging the refresh token for an access token.
- Making Google+ API requests with the access token, including getting a list of people that the user has circled.
- Disconnecting the app from the user's Google account and revoking tokens.

Step 1: Enable the Google+ API

Create a Google APIs Console project, OAuth 2.0 client ID, and register your JavaScript origins: To register a new application, do the following:

- Go to the Google Cloud Console.
- Select a project, or create a new one.
- In the sidebar on the left, select APIs & auth. In the displayed list of APIs, make sure the Google+ API status is set to ON.
- In the sidebar on the left, select Registered apps.
- At the top of the page, select Register App.
- Fill out the form and select Register.

Register the origins where your app is allowed to access the Google APIs. The origin is the unique combination of protocol, hostname, and port. You can enter multiple origins to allow for your app to run on different protocols, domains or subdomains. Wildcards are not allowed.

- Expand the OAuth 2.0 Client ID section.
- In the Web origin field, enter your origin:
`http://localhost:4567`
- Press ENTER to save your origin. You can then click the + symbol to add additional origins.
- Note or copy the client ID and client secret that your app will need to use to access the APIs.

Client Secrets

93.13. Google+ Sign-In for server-side apps

To take advantage of all of the benefits of Google+ Sign-In you must use a hybrid server-side flow where a user authorizes your app on the client side using the JavaScript API client and you send a special one-time authorization code to your server.

Your server exchanges this one-time-use code to acquire its own access and refresh tokens from Google for the server to be able to make its own API calls, which can be done while the user is offline.

This one-time code flow has security advantages over both a pure server-side flow and over sending access tokens to your server.

The Google+ Sign-In server-side flow differs from the OAuth 2.0 for Web server applications flow.

93.14. Authentication using the Google APIs Client Library for JavaScript

See.

Capítulo 94

MacOS homebrew Package Manager

Homebrew installs packages to their own directory

Homebrew installs the stuff you need that Apple didn't.

```
$ brew install wget
```

Homebrew installs packages to their own directory and then symlinks their files into /usr/local.

```
$ cd /usr/local  
$ find Cellar  
Cellar/wget/1.12  
Cellar/wget/1.12/bin/wget  
Cellar/wget/1.12/share/man/man1/wget.1  
  
$ ls -l bin  
bin/wget -> ../Cellar/wget/1.12/bin/wget
```

Homebrew won't install files outside its prefix, and you can place a Homebrew installation wherever you like.

Create your own Homebrew packages

Trivially create your own Homebrew packages.

```
$ brew create http://foo.com/bar-1.0.tgz  
Created /usr/local/Library/Formula/bar.rb
```

It's all git and ruby underneath, so hack away with the knowledge that you can easily revert your modifications and merge upstream updates.

```
$ brew edit wget # opens in $EDITOR!
```

Homebrew formulae are simple Ruby scripts:

```
require 'formula'  
  
class Wget < Formula  
  homepage 'http://www.gnu.org/wget/'  
  url 'http://ftp.gnu.org/wget-1.12.tar.gz'  
  md5 '308a5476fc096a8a525d07279a6f6aa3'  
  
  def install  
    system "./configure --prefix=#{prefix}"  
    system 'make install'  
  end  
end
```

Homebrew complements OS X. Install your gems with gem, and their dependencies with brew.

Comandos

```
[~/local/src/ruby/sinatra/jump-start-sinatra]$ brew --help
```

Example usage:

```
brew [info | home | options ] [FORMULA...]
brew install FORMULA...
brew uninstall FORMULA...
brew search [foo]
brew list [FORMULA...]
brew update
brew upgrade [FORMULA...]
```

Troubleshooting:

```
brew doctor
brew install -vd FORMULA
brew [--env | --config]
```

Brewing:

```
brew create [URL [--no-fetch]]
brew edit [FORMULA...]
open https://github.com/mxcl/homebrew/wiki/Formula-Cookbook
```

Further help:

```
man brew
brew home
```

HOMEBREW_GITHUB_API_TOKEN

A personal GitHub API Access token, which you can create at <https://github.com/settings/applications>. If set, GitHub will allow you a greater number of API requests. See <http://developer.github.com/v3/#rate-limiting> for more information.

Homebrew uses the GitHub API for features such as `brew search`.

```
$ brew search postgres
Error: GitHub Server Error
You may want to create an API token: https://github.com/settings/applications
and then set HOMEBREW_GITHUB_API_TOKEN.
```

Vete a la página y crea el token. Luego añade una línea como esta en tu `.bashrc`:

```
$ export HOMEBREW_GITHUB_API_TOKEN fx00011111111111111111eeeeeeeeeee
$ brew search postgres
No formula found for "postgres". Searching open pull requests...
```

Véase

1. *brew(1) – The missing package manager for OS X*
2. *Homebrew Demystified: OS X's Ultimate Package Manager*

Capítulo 95

PostgreSQL

1. *Postgres.app*

2. *Chapter 15. Installation from Source Code*

```
$ brew update
$ brew search postgresQL
postgresql      postgresql8      postgresql9      postgresql91      postgresql92

$ brew install postgresql --no-tcl # fail

[/tmp]$ brew install postgresql92 --no-tcl
Warning: It appears you have MacPorts or Fink installed.
Software installed with other package managers causes known problems for
Homebrew. If a formula fails to build, uninstall MacPorts/Fink and try again.
==> Downloading http://ftp.postgresql.org/pub/source/v9.2.4/postgresql-9.2.4.tar.bz2
Already downloaded: /Library/Caches/Homebrew/postgresql92-9.2.4.tar.bz2
==> Patching
patching file src/pl/plpython/Makefile
patching file contrib/uuid-ossp/uuid-ossp.c
==> ./configure --prefix=/usr/local/Cellar/postgresql92/9.2.4 --datadir=/usr/local/Cellar/post
==> make install-world
==> Caveats
initdb /usr/local/var/postgres -E utf8      # create a database
postgres -D /usr/local/var/postgres          # serve that database
PGDATA=/usr/local/var/postgres    # ...alternatively

uilds of PostgreSQL 9 are failing and you have version 8.x installed,
may need to remove the previous version first. See:
https://github.com/mxcl/homebrew/issues/issue/2510

igrate existing data from a previous major version (pre-9.2) of PostgreSQL, see:
http://www.postgresql.org/docs/9.2/static/upgrading.html

machines may require provisioning of shared memory:
http://www.postgresql.org/docs/9.2/static/kernel-resources.html#SYSVIPC

When installing the postgres gem, including ARCHFLAGS is recommended:
ARCHFLAGS="-arch x86_64" gem install pg

To install gems without sudo, see the Homebrew wiki.
```

To have launchd start postgresql92 at login:

```
ln -sfv /usr/local/opt/postgresql92/*.plist ~/Library/LaunchAgents
```

Then to load postgresql92 now:

```
launchctl load ~/Library/LaunchAgents/homebrew.mxcl.postgresql92.plist
```

Or, if you don't want/need launchctl, you can just run:

```
pg_ctl -D /usr/local/var/postgres -l /usr/local/var/postgres/server.log start
```

=> Summary

```
/usr/local/Cellar/postgresql92/9.2.4: 2831 files, 38M, built in 2.8 minutes
```

Capítulo 96

SASS (Syntactically Awesome StyleSheets)

96.1. Introducción

Sass is an extension of CSS that adds power and elegance to the basic language. It allows you to use variables, nested rules, mixins, inline imports, and more, all with a fully CSS-compatible syntax. Sass helps keep large stylesheets well-organized, and get small stylesheets up and running quickly, particularly with the help of the Compass style library.

96.2. Modos de Uso

Sass can be used in three ways:

1. as a command-line tool,
2. as a standalone Ruby module,
3. and as a plugin for any Rack-enabled framework, including Ruby on Rails and Sinatra.

The first step for all of these is to install the Sass gem:

```
gem install sass
```

To run Sass from the command line, just use

```
sass input.scss output.css
```

You can also tell Sass to watch the file and update the CSS every time the Sass file changes:

```
sass --watch input.scss:output.css
```

If you have a directory with many Sass files, you can also tell Sass to watch the entire directory:

```
sass --watch app/sass:public/stylesheets
```

Use sass --help for full documentation.

```
[~/sinatra/sinatra-tictactoe/sinatra-tictactoe-ajax(master)]$ sass --help
Usage: sass [options] [INPUT] [OUTPUT]
```

Description:

Converts SCSS or Sass files to CSS.

Options:

| | |
|------------------------------|--|
| -s, --stdin | Read input from standard input instead of an input file |
| --trace | Show a full traceback on error |
| --unix-newlines | Use Unix-style newlines in written files. |
| --scss | Use the CSS-superset SCSS syntax. |
| --watch | Watch files or directories for changes. |
| | The location of the generated CSS can be set using a colon:
sass --watch input.sass:output.css
sass --watch input-dir:output-dir |
| --update | Compile files or directories to CSS.
Locations are set like --watch. |
| --stop-on-error | If a file fails to compile, exit immediately.
Only meaningful for --watch and --update. |
| --poll | Check for file changes manually, rather than relying on the OS.
Only meaningful for --watch. |
| -f, --force | Recompile all Sass files, even if the CSS file is newer.
Only meaningful for --update. |
| -c, --check | Just check syntax, don't evaluate. |
| -t, --style NAME | Output style. Can be nested (default), compact, compressed or expanded. |
| --precision NUMBER_OF_DIGITS | How many digits of precision to use when outputting decimal numbers. |
| -q, --quiet | Silence warnings and status messages during compilation. |
| --compass | Make Compass imports available and load project configuration. |
| -g, --debug-info | Emit extra information in the generated CSS that can be used for debugging. |
| -l, --line-numbers | Emit comments in the generated CSS indicating the corresponding line number. |
| --line-comments | |
| -i, --interactive | Run an interactive SassScript shell. |
| -I, --load-path PATH | Add a sass import path. |
| -r, --require LIB | Require a Ruby library before running Sass. |
| --cache-location PATH | The path to put cached Sass files. Defaults to .sass-cache. |
| -C, --no-cache | Don't cache to sassc files. |
| -E encoding | Specify the default encoding for Sass files. |
| -?, -h, --help | Show this message. |
| -v, --version | Print version. |

Using Sass in Ruby code is very simple. After installing the Sass gem, you can use it by running require "sass" and using `Sass::Engine` like so:

```
template = File.load('stylesheets/sassy.sass')
sass_engine = Sass::Engine.new(template)
output = sass_engine.render
puts output
```

96.3. Referencias

1. *Sass (Syntactically Awesome StyleSheets): Reference Guide*
2. *Sass (Syntactically Awesome StyleSheets): Sass Basics*

Capítulo 97

Haml

97.1. Filters

The colon character designates a filter. This allows you to pass an indented block of text as input to another filtering program and add the result to the output of Haml. The syntax is simply a colon followed by the name of the filter. For example:

```
%p
:markdown
# Greetings

Hello, *World*
```

is compiled to:

```
<p>
  <h1>Greetings</h1>

  <p>Hello, <em>World</em></p>
</p>
```

Filters can have Ruby code interpolated with `#{}.` For example:

```
- flavor = "raspberry"
#content
:textile
  I *really* prefer _#{flavor}_ jam.
```

is compiled to

```
<div id='content'>
  <p>I <strong>really</strong> prefer <em>raspberry</em> jam.</p>
</div>
```

Haml comes with the following filters defined:

1. :cdata

Surrounds the filtered text with CDATA tags.

2. :coffee

Compiles the filtered text to Javascript using Coffeescript. You can also reference this filter as :coffeescript. This filter is implemented using Tilt.

3. `:css`

Surrounds the filtered text with `jstyle`; and (optionally) CDATA tags. Useful for including inline CSS. Use the `:cdata` option to control when CDATA tags are added.

4. `:erb`

Parses the filtered text with ERb, like an RHTML template. Not available if the `:suppress_eval` option is set to true. Embedded Ruby code is evaluated in the same context as the Haml template. This filter is implemented using Tilt.

5. `:escaped`

Works the same as plain, but HTML-escapes the text before placing it in the document.

6. `:javascript`

Surrounds the filtered text with `jscript`; and (optionally) CDATA tags. Useful for including inline Javascript. Use the `:cdata` option to control when CDATA tags are added.

7. `:less`

Parses the filtered text with Less to produce CSS output. This filter is implemented using Tilt.

8. `:markdown`

Parses the filtered text with Markdown. This filter is implemented using Tilt.

9. `:maruku`

Parses the filtered text with Maruku, which has some non-standard extensions to Markdown.

As of Haml 4.0, this filter is defined in Haml contrib but is loaded automatically for historical reasons. In future versions of Haml it will likely not be loaded by default. This filter is implemented using Tilt.

10. `:plain`

Does not parse the filtered text. This is useful for large blocks of text without HTML tags, when you don't want lines starting with . or - to be parsed.

11. `:preserve`

Inserts the filtered text into the template with whitespace preserved. preserved blocks of text aren't indented, and newlines are replaced with the HTML escape code for newlines, to preserve nice-looking output. See also Whitespace Preservation.

12. `:ruby`

Parses the filtered text with the normal Ruby interpreter. Creates an IO object named `haml_io`, anything written to it is output into the Haml document. Not available if the `:suppress_eval` option is set to true. The Ruby code is evaluated in the same context as the Haml template.

13. `:sass`

Parses the filtered text with Sass to produce CSS output. This filter is implemented using Tilt.

14. `:scss`

Parses the filtered text with Sass like the `:sass` filter, but uses the newer SCSS syntax to produce CSS output. This filter is implemented using Tilt.

15. `:textile`

Parses the filtered text with Textile. Only works if RedCloth is installed.

As of Haml 4.0, this filter is defined in Haml contrib but is loaded automatically for historical reasons. In future versions of Haml it will likely not be loaded by default. This filter is implemented using Tilt.

97.2. HTML5 Custom Data Attributes

1. *HTML5 allows for adding custom non-visible data attributes to elements using attribute names beginning with `data-`.*
2. *Custom data attributes can be used in Haml by using the key `:data` with a Hash value in an attribute hash.*
3. *Each of the key/value pairs in the Hash will be transformed into a custom data attribute.*
4. *For example:*

```
%a{:href=>"/posts", :data => {:_author_id => 123}} Posts By Author
```

will render as:

```
<a data-author-id='123' href='/posts'>Posts By Author</a>
```

5. *Notice that the underscore in `author_id` was replaced by a hyphen.*
6. *If you wish to suppress this behavior, you can set Haml's `:hyphenate_data_attrs` option to `false`, and the output will be rendered as:*

```
<a data-author_id='123' href='/posts'>Posts By Author</a>
```

Capítulo 98

Selenium

98.1. Introducción

Selenium es un entorno usado para la automatización de aplicaciones web cuyo propósito son las pruebas.

- También puede automatizar tareas de administración.
- Selenium se apoya en los principales navegadores y está en proceso de llegar a formar parte de los mismos.
- También se utiliza en otras herramientas de automatización de navegadores, APIs y frameworks.

98.2. Componentes

Selenium-IDE

The Selenium-IDE (Integrated Development Environment) is the tool you use to develop your Selenium test cases.

It's an easy-to-use Firefox plug-in and is generally the most efficient way to develop test cases.

It also contains a context menu that allows you to first select a UI element from the browser's currently displayed page and then select from a list of Selenium commands with parameters pre-defined according to the context of the selected UI element.

This is not only a time-saver, but also an excellent way of learning Selenium script syntax.

Véase Selenium IDE documentation

Selenium WebDriver

It drives a browser natively as a user. Selenium-WebDriver makes direct calls to the browser using each browser's native support for automation.

How these direct calls are made, and the features they support depends on the browser you are using.

WebDriver and the Selenium-Server

You may, or may not, need the Selenium Server, depending on how you intend to use Selenium-WebDriver.

If you will be only using the WebDriver API you do not need the Selenium-Server.

If your browser and tests will all run on the same machine, and your tests only use the WebDriver API, then you do not need to run the Selenium-Server; WebDriver will run the browser directly.

98.3. Setting Up a Selenium-WebDriver Project

To install Selenium means to set up a project in a development so you can write a program using Selenium. How you do this depends on your programming language and your development environment.

To add Selenium to your Ruby environment run the following command from a command-line.

```
gem install selenium-webdriver
```

There are many other Selenium gems out there, but this is the official, maintained gem.

If you're looking for a slightly higher level API built on the same technology, you may want to check out capybara.

Capybara helps you test web applications by simulating how a real user would interact with your app. It is agnostic about the driver running your tests and comes with Rack::Test and Selenium support built in.

Capítulo 99

ETSII

99.1. Prácticas: Centro de Cálculo

1. *Modo de trabajo en el sistema de archivos del CC de la ETSII*
2. *Ubicación de las salas*

99.2. Despliegue de una Aplicación Web en la ETSII

Para desplegar una aplicación web usaremos exthost2 (en 2013).

Veamos que puertos están libres usando netstat :

```
casiano@exthost2:~$ netstat -an | less
```

o bien usamos lsof :

```
lsof -i | less
```

y después - si es necesario - terminamos el proceso que ya estuviera escuchando en el puerto

```
kill -9 PID
```

Veamos una simple aplicación usando rack:

```
casiano@exthost2:~/src/ruby/simplewebapp$ cat hello.rb
require 'rack'
```

```
app = lambda { |env| [200, {"Content-Type" => "text/plain"}, ["Hello. The time is #{Time.now}"]]
Rack::Handler::WEBrick.run app,:Port => 4567
```

La ejecutamos:

```
casiano@exthost2:~/src/ruby/simplewebapp$ ruby hello.rb
[2013-10-28 09:58:54] INFO  WEBrick 1.3.1
[2013-10-28 09:58:54] INFO  ruby 1.9.3 (2011-10-30) [i686-linux]
[2013-10-28 09:58:54] WARN  TCPServer Error: Address already in use - bind(2)
[2013-10-28 09:58:54] INFO  WEBrick::HTTPServer#start: pid=16597 port=4567
```

Ya tenemos disponible la página en exthost2 en el puerto correspondiente.

El acceso al servidor está limitado a la red de la ULL.

Véase también

1. *Gemas instaladas en local 99.3*

99.3. Gemas instaladas en local

Del man de bundle-install:

```
--path=<path>
```

The location to install the gems in the bundle to. This defaults to the gem home, which is the location that gem install installs gems to. This means that, by default, gems installed without a --path setting will show up in gem list. This setting is a

```
casiano@exthost2:~/sinatra/sinatra-up-and-running-blog (master)$ bundle install --path /home/c  
Fetching gem metadata from https://rubygems.org/.....  
Fetching gem metadata from https://rubygems.org/..  
Installing daemons (1.1.9)  
Installing eventmachine (1.0.3)  
Installing rack (1.5.2)  
Installing rack-cache (1.2)  
Installing rack-protection (1.5.1)  
Installing rdiscount (2.1.7)  
Installing redcarpet (3.0.0)  
Installing tilt (1.4.1)  
Installing sinatra (1.4.4)  
Installing thin (1.6.0)  
Using bundler (1.3.5)  
Your bundle is complete!  
It was installed into /home/casiano/gems
```

Arrancamos el servidor:

```
casiano@exthost2:~/sinatra/sinatra-up-and-running-blog (master)$ rackup -Ilib -I/home/casiano/  
cache: [GET /] miss, store  
10.213.5.185 - - [28/Oct/2013 12:24:29] "GET / HTTP/1.1" 200 3892 5.6308  
cache: [GET /css/blog.css] miss, store  
10.213.5.185 - - [28/Oct/2013 12:24:29] "GET /css/blog.css HTTP/1.1" 200 375 0.0706  
cache: [GET /js/jquery.timeago.js] miss, store  
10.213.5.185 - - [28/Oct/2013 12:24:29] "GET /js/jquery.timeago.js HTTP/1.1" 200 4491 0.1020  
cache: [GET /js/blog.js] miss, store  
10.213.5.185 - - [28/Oct/2013 12:24:29] "GET /js/blog.js HTTP/1.1" 200 66 0.0427
```

Capítulo 100

jsFiddle

jsFiddle es una Online Javascript IDE. De la wikipedia Online Javascript IDE:

- An online Javascript IDE (or browser based Javascript IDE, Javascript live coding environment, web playground, javascript sandbox) is an Integrated Development Environment (IDE) that is hosted in a browser, with an aim to ease Javascript, HTML, and CSS based web development.
 - Generally, they allow users to edit Javascript code in the browser, and see the results of executing the code. Many will also allow the editing of HTML or CSS content.
 - Many of them support saving the work or sharing links with others, leading to their prevalence for showing examples on such sites such as StackOverflow, where a question can be answered by pointing to a working snippet of code.
 - Many libraries for Javascript, provide links to demonstration code that can be edited by users.
 - They are also used as a pedagogical tool by institutions such as Khan Academy to allow students to experience writing code in an environment where they can see the output of their programs, without needing any setup beyond a web browser.
-
- Introduction to jsFiddle YouTube
 - Documentación de jsFiddle
 - Tutorail de jsFiddle

Capítulo 101

Codenvy

101.1. Introducción

Codenvy provisions, shares and scales developer environments. Offered as SaaS and is based upon open Eclipse projects.

- *Deploying a Ruby App to Heroku in Codenvy* Youtube
- *Real Time Collaboration and Chat. Code Folding and Editor Themes in Codenvy.* Vimeo

Capítulo 102

Editores Entornos de Desarrollo

- *Redcar*

Capítulo 103

Website wireframes

A website wireframe, also known as a page schematic or screen blueprint, is a visual guide that represents the skeletal framework of a website. Wireframes are created for the purpose of arranging elements to best accomplish a particular purpose. The purpose is usually being informed by a business objective and a creative idea. The wireframe depicts the page layout or arrangement of the website's content, including interface elements and navigational systems, and how they work together. The wireframe usually lacks typographic style, color, or graphics, since the main focus lies in functionality, behavior, and priority of content. In other words, it focuses on what a screen does, not what it looks like. Wireframes can be pencil drawings or sketches on a whiteboard, or they can be produced by means of a broad array of free or commercial software applications. Wireframes are generally created by business analysts, user experience designers, developers, visual designers and other roles with expertise in interaction design, information architecture and user research.

1. Mashable: 10 wireframing tools
2. Mockingbird

Capítulo 104

Markdown

Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).

Thus, “Markdown” is two things:

1. *a plain text formatting syntax; and*
2. *a software tool, written in Perl, that converts the plain text formatting to HTML.*

Try it out, right now, using the online Dingus.

The overriding design goal for Markdown’s formatting syntax is to make it as readable as possible.

The idea is that a Markdown-formatted document should be publishable as-is, as plain text, without looking like it’s been marked up with tags or formatting instructions.

104.1. Véase

Para saber más del formato Markdown:

1. *UN parser Markdown en Ruby: kramdown.*
Véase el fuente en *github*: <https://github.com/getalong/kramdown>
2. *Markdown: Basics*
3. *Markdown: Syntax*
4. *Markdown Cheatsheet*
5. *Vim Essential Plugin: Markdown to HTML*

Capítulo 105

Jekyll

105.1. Introducción

```
[~/sinatra/sinatra-documentation(master)]$ jekyll --help  
NAME:
```

```
jekyll
```

```
DESCRIPTION:
```

```
Jekyll is a blog-aware, static site generator in Ruby
```

```
COMMANDS:
```

build	Build your site
default	
docs	Launch local server with docs for Jekyll v1.1.2
doctor	Search site and print specific deprecation warnings
help	Display global or [command] help documentation.
import	Import your old blog to Jekyll
new	Creates a new Jekyll site scaffold in PATH
serve	Serve your site locally

```
ALIASES:
```

hyde	doctor
server	serve

```
GLOBAL OPTIONS:
```

```
-s, --source [DIR]  
  Source directory (defaults to ./)  
  
-d, --destination [DIR]  
  Destination directory (defaults to ./_site)  
  
--safe
```

Véase: [Tutorial y Documentación](#)

1. *Jekyll: Sites Made Simple by Simon Pascal Klein SitePoint Jekyll documentation*

2. Learning Jekyll By Example: Build a Jekyll Website, Start to Finish
3. Blogging With Jekyll And Latex
4. Jekyll Blogging Oct 24, 2011 Cecil Woebker
5. Intro to Jekyll por Johan Ronsse, YouTube
6. Jekyll Static Site Generator RubyOnRailsPhoenix YouTube
7. Hosting a static (Jekyll) blog on Amazon S3 Posted on 01 Feb 2013

Comentarios y Botones de Compartir

- *Installing Disqus in Jekyll*
- *IMPLEMENTATION OF SOCIAL SHARING INTO JEKYLL. How to add social sharing buttons to your Jekyll Powered Blog*
- *Jekyll Sharing Buttons por David Kelso*
- *Para obtener la API Key de Facebook: <https://developers.facebook.com/apps>*

Plugins

- <http://www.jekyll-plugins.com/>
- *Liquid-LaTeX Jekyll Plugin por Fèlix Galindo Allué (flx.cat). Repositorio en GitHub*
- *Liquid-Lipsum Jekyll Plugin por Fèlix Galindo Allué (flx.cat). Repositorio en GitHub*
- *Advanced Jekyll Plugins and Features - Custom Post Types, Category Pages, Pagination, and More By Jake Johnson Saturday, June 29, 2013*

Liquid

Liquid is a ruby library for rendering safe templates which cannot affect the security of the server they are rendered on. Liquid does NOT allow ruby code inside the markup.

```
[1] pry(main)> require 'liquid'
=> true
[2] pry(main)> @template = Liquid::Template.parse("hi {{name}}")
=> #<Liquid::Template:0x007f820f8206e8
@root=
#<Liquid::Document:0x007f820f8205a8
@odelist=
["hi ",
 #<Liquid::Variable:0x007f820f8203a0
 @filters=[],
 @markup="name",
 @name="name">>]>>
[3] pry(main)> @template.render('name' => 'tobi')
=> "hi tobi"
```

Filters can be applied on output tags:

```
[~/rubytesting/liquid]$ cat filters.rb
require 'liquid'

template = <<-'EOT'
Hello {{ 'tobi' | upcase }}
Hello 'tobi' has {{ 'tobi' | size }} letters!
Today is {{ 'now' | date: "%A %dth of %B %Y" }}
EOT

@template = Liquid::Template.parse(template)

puts @template.render()
[~/rubytesting/liquid]$ ruby filters.rb
Hello TOBI
Hello 'tobi' has 4 letters!
Today is Saturday 11th of January 2014
```

You can't give access to objects of arbitrary classes to end users. Due security concerns, only String, Numeric, Hash, Array, Proc, Boolean or Liquid::Drop are allowed by default. The final value rendered in the template is the result of sending to_liquid message to the resolved object. Liquid extends some of the ruby standard classes with that method.

```
[~/rubytesting/liquid]$ cat if.rb
require 'liquid'

User = Struct.new(:name)
user = User.new('Juana')
def user.to_liquid
{
  'name' => self.name,
  'age'  => 9
}
end

template = <<-'EOT'
{% if user %}
  Hello {{ user.name}} is {{user.age}} years old
{% endif %}
EOT

@template = Liquid::Template.parse(template)
puts @template.render("user" => user)

[~/rubytesting/liquid]$ ruby if.rb
```

Hello Juana is 9 years old

Creating filters is very easy. Basically, they are just methods which take one parameter and return a modified string. You can use your own filters by passing an array of modules to the render call like this: @template.render(assigns, [MyTextFilters, MyDateFilters]).

Alternatively, you can register your filters globally:

```
[~/rubytesting/liquid]$ cat create_filter.rb
require 'liquid'
```

```

require 'redcloth'

module TextFilter
  def textilize input
    RedCloth.new(input).to_html
  end
end

Liquid::Template.register_filter(TextFilter)

t = <<"EOI"
{{ 'Do not *ever* pull this _lever_. ' | textilize}}
EOI
template = Liquid::Template.parse t
puts template.render()

[~/rubystuff/liquid]$ ruby create_filter.rb
<p>Do not <strong>ever</strong> pull this <em>lever</em>. </p>
[~/rubystuff/liquid]$

```

- *Liquid wiky at GitHub*
- *Liquid: Ruby library for rendering safe templates which cannot affect the security of the server they are rendered on.*
- *Old Emmanuel Oga's Weblog (new one is at www.emmanueloga.com). Liquid Coolness*

105.2. Jekyll en GitHub

105.2.1. User and Organization Pages

- *The repository must use the `username/username.github.io` naming scheme.*
- *Content from the master branch will be used to build and publish the Pages.*
- *To generate User and Organization pages, you'll need to create a repository named `username.github.io` or `orgname.github.io` first.*
- *The `username` or organization name must be your own or Pages will not build.*
- *The automatic page generator is accessible via the repository's admin page after you've created the repository.*

Pasos/Steps

1. *Create a repository. Head over to GitHub and create a new repository named `username.github.io`, where `username` is your username (or organization name) on GitHub.*
2. *Clone the repository*

```

git clone https://github.com/username/username.github.io
$git checkout master

```

3. *Enter the project folder and add an `index.html` file:*

```

~$cd username.github.io
~$echo "Hello World" > index.html

```

4. Add, commit, and push your changes:

```
~$git add --all  
~$git commit -m "Initial commit"  
~$git push
```

5. Fire up a browser and go to <http://username.github.io>.

Give it a couple of minutes for your page to show up—there will be a delay this very first time. In the future, changes will show up pretty much instantly.

105.2.2. Project Pages

Unlike User and Org Pages, Project Pages are kept in the same repository as the project they are for.

These pages are similar to User and Org Pages, with a few slight differences:

- The `gh-pages` branch is used to build and publish from.
- A custom domain on user/org pages will apply the same domain redirect to all project pages hosted under that account, unless the project pages use their own custom domain .
- If no custom domain is used, the project pages are served under a subpath of the user pages: `username.github.io/projectname`
- Custom 404s will only work if a custom domain is used, otherwise the User Pages 404 is used

105.2.3. Véase

1. *GitHub pages*
2. *Using Github Pages to host a website (v2)* por Vincent Knight (YouTube)
3. *Jekyll: setting up on GitHub* por Thomas Bradley (YouTube)
4. *Hosting websites with GitHub* por Oscar Cortez (YouTube)
5. *Set up a personal website using GitHub.io* por Mitchell Lee (YouTube)
6. *Using Jekyll Plugins on Github Pages*

105.3. Jekyll en Heroku

- Automatically build and deploy Jekyll sites to heroku (from github) Alternative title: Building and deploying a jekyll site with an ajax contact form
- How To Deploy Website Generated Using Jekyll To Heroku As Rack App
- Jekyll-Bootstrap for Heroku

Capítulo 106

Showoff

ShowOff is a Sinatra web app that reads simple configuration files for a presentation.

The idea is that you setup your markdown slide files in section subdirectories and then startup the showoff server in that directory. It will read in your showoff.json file for which sections go in which order and then will give you a URL to present from.

1. <https://github.com/puppetlabs/showoff>

2. <https://github.com/crguezl/showoff>

```
[~/sinatra/showoff(master)]$ showoff --help
```

NAME

showoff -

SYNOPSIS

```
showoff [global options] command [command options] [arguments...]
```

VERSION

0.7.0

GLOBAL OPTIONS

```
--help      - Show this message
--version   - Display the program version
```

COMMANDS

```
add, new      - Add a new slide at the end in a given dir
create, init  - Create new showoff presentation
github        - Puts your showoff presentation into a gh-pages branch
help          - Shows a list of commands or help for one command
heroku        - Setup your presentation to serve on Heroku
serve         - Serves the showoff presentation in the specified (or current) directory
static        - Generate static version of presentation
```

Capítulo 107

GitHub

107.1. Organizations

1. *GithHub Teams and Organizations for Hackathons de Matthew McCullough*

107.2. GitHub Pages

1. *Creating Pages with the automatic generator*
2. *Using Jekyll with GitHub Pages*

Capítulo 108

Vim

108.1. Plugin para parejas (paréntesis, etc.)

- *surround*

108.2. Plugins para HTML

- *Emmet tutorial*
- *Emmet documentation*
- *Sintáxis de Emmet*
- *Screencast #129: Emmet (is awesome) by Chris Coyier YouTube*

108.3. Plugins para git

- *vim-fugitive en GitHub*
- *The Fugitive Series - a retrospective*
- *Fugitive.vim - resolving merge conflicts with vimdiff*
- *Fugitive.vim - a complement to command line git*

108.4. Vim airline

- *vim-airline en GitHub: Lean and mean status/tabline for vim that's light as air*
- *Powerline alternatives*

Capítulo 109

Apache

109.1. Apache en Mac OS X

109.1.1. Configuring Apache

1. *The default system document root is still found at - `http://localhost/` The files are shared in the filing system at -*

`/Library/WebServer/Documents/`

2. *The other web root directory which is missing by default is the `~/Sites` folder in the User account.*

3. *Check that you have a `username.conf` file under:*

`/etc/apache2/users/`

If you don't (very likely), then create one named by the short username of the account with the suffix `.conf`, its location and permissions/ownership is best tackled by using the Terminal:

```
[~/LPPbook(master)]$ ls -l /etc/apache2/users/
total 40
-rw-r--r-- 1 root  wheel  143 15 feb  2010 casiano.conf
```

4. [~/LPPbook(master)]\$ cat /etc/apache2/users/casiano.conf

```
<Directory "/Users/casiano/Sites/">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

5. *Permissions on the file should be:*

```
-rw-r--r-- 1 root  wheel  298 Jun 28 16:47 username.conf
```

6. *Restart Apache for the new file to be read:*

```
sudo apachectl restart
```

7. *Then this user level document root will be viewable at:*

`http://localhost/~username/`

109.1.2. PHP

PHP 5.4.17 is loaded in the latest preview of OSX 10.9 Mavericks and needs to be turned on by uncommenting a line in the httpd.conf file.

```
sudo vi /etc/apache2/httpd.conf
```

Search for php this will land you on the right line then uncomment the line:

```
LoadModule php5_module libexec/apache2/libphp5.so
```

Reload apache to kick in

```
sudo apachectl restart
```

To see and test PHP, create a file name it phpinfo.php and file it in your document root with the contents below, then view it in a browser.

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

109.1.3. Perl

```
MacBook-Air-de-casiano-2:apache2 root# pwd  
/etc/apache2
```

```
MacBook-Air-de-casiano-2:apache2 root# tail httpd.conf
```

```
Include /private/etc/apache2/other/*.conf
```

```
MacBook-Air-de-casiano-2:apache2 root# cat other/perl.conf  
LoadModule perl_module libexec/apache2/mod_perl.so  
Alias /perl /Library/WebServer/perl  
<Location /perl>  
    SetHandler perl-script  
    PerlResponseHandler ModPerl::Registry  
    Options ExecCGI  
    PerlSendHeader On  
    Order allow,deny  
    Allow from all  
</Location>
```

```
[/Library/WebServer/CGI-Executables]$ pwd  
/Library/WebServer/CGI-Executables  
[/Library/WebServer/CGI-Executables]$ cat hello.cgi  
#!/usr/bin/perl  
use strict;  
use warnings;  
use CGI qw/:standard/;  
use Data::Dumper;  
print header, start_html, h1('it works'), end_html;  
  
# apachectl restart
```

```

[/Library/WebServer/CGI-Executables]$ curl -v localhost/cgi-bin/hello.cgi
* Adding handle: conn: 0x7fc47c008c00
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7fc47c008c00) send_pipe: 1, recv_pipe: 0
* About to connect() to localhost port 80 (#0)
* Trying ::1...
* Connected to localhost (::1) port 80 (#0)
> GET /cgi-bin/hello.cgi HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 09 Nov 2013 12:31:00 GMT
* Server Apache/2.2.24 (Unix) PHP/5.4.17 DAV/2 mod_ssl/2.2.24 OpenSSL/0.9.8y mod_perl/2.0.7 Pe
< Server: Apache/2.2.24 (Unix) PHP/5.4.17 DAV/2 mod_ssl/2.2.24 OpenSSL/0.9.8y mod_perl/2.0.7 P
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=ISO-8859-1
<
<!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" lang="en-US" xml:lang="en-US">
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<h1>it works</h1>
</body>
* Connection #0 to host localhost left intact
</html>

-rwxr-xr-x 1 casiano staff 132 9 nov 12:09 hello.pl
[/Library/WebServer/perl]$ vi hello.pl
[/Library/WebServer/perl]$ pwd
/Library/WebServer/perl
[/Library/WebServer/perl]$ cat hello.pl
#!/usr/bin/perl
use strict;
use warnings;
use CGI qw/:standard/;
use Data::Dumper;
print header, start_html, h1('with Perl-module works'), end_html;
[/Library/WebServer/perl]$


[/Library/WebServer/perl]$ curl -v 'http://localhost/perl/hello.pl'
* Adding handle: conn: 0x7f7f99804000
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7f7f99804000) send_pipe: 1, recv_pipe: 0

```

```

* About to connect() to localhost port 80 (#0)
*   Trying ::1...
* Connected to localhost (::1) port 80 (#0)
> GET /perl/hello.pl HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 09 Nov 2013 12:38:07 GMT
* Server Apache/2.2.24 (Unix) PHP/5.4.17 DAV/2 mod_ssl/2.2.24 OpenSSL/0.9.8y mod_perl/2.0.7 Pe
< Server: Apache/2.2.24 (Unix) PHP/5.4.17 DAV/2 mod_ssl/2.2.24 OpenSSL/0.9.8y mod_perl/2.0.7 Pe
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=ISO-8859-1
<
<!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" lang="en-US" xml:lang="en-US">
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<h1>with Perl-module works</h1>
</body>
* Connection #0 to host localhost left intact

```

109.1.4. MySQL

MySQL is again a missing component in OS X 10.9 and needs to be dowloaded from the MySQL site use the Mac OS X ver. 10.7 (x86, 64-bit), DMG Archive version (works fine on 10.9).

When downloading you don't have to sign up, look for No thanks, just take me to the downloads! - go straight to the download mirrors and download the software from a mirror which is closest to you.

Once downloaded install the 3 components.

You may need to adjust the Security and Privacy System Pref to allow installs of 3rd party apps because of the new security feature of Mountain Lion known as the Gatekeeper, which keeps unscrupulous installer packages at bay.

To get around this without changing the global preferences (better!) right click or command click the .pkg installer to bring up the contextual menu and select open, then you get the warning - then click Open.

109.1.5. Enlaces

1. *Get your Local Web Development Server Up and Running on OSX 10.9 Mavericks*

Capítulo 110

DataMapper

1. *Ruby for Newbies: Working with DataMapper por Andrew Burgess on Apr 19th 2011*

Capítulo 111

Perpetuity

Perpetuity is a simple Ruby object persistence layer that attempts to follow Martin Fowler's Data Mapper pattern, allowing you to use plain-old Ruby objects in your Ruby apps in order to decouple your domain logic from the database as well as speed up your tests.

There is no need for your model classes to inherit from another class or even include a mix-in. Your objects will hopefully eventually be able to be persisted into whichever database you like.

- *jgaskins / perpetuity Jamie Gaskins February 01, 2013*
- *YouTube Perpetuity: Ruby gem for Data Mapper persistence by Jamie Gaskins*

Servidor/Daemon/mongod

```
[~/rubytesting/perpetuity]$ mongod -vvvv  
all output going to: /usr/local/var/log/mongodb/mongo.log
```

Gemfile

```
[~/rubytesting/perpetuity]$ cat Gemfile  
source 'https://rubygems.org'  
  
gem 'perpetuity-postgres'  
gem 'perpetuity-mongodb', '1.0.0.beta'
```

Salvando Objetos

```
[~/rubytesting/perpetuity]$ cat app.rb  
require 'perpetuity'  
require 'perpetuity/mongodb'  
  
Perpetuity.data_source :mongodb, 'my_perpetuity_database'  
  
#Perpetuity.configure do  
#  data_source :mongodb  
#end
```

```
class Article  
  attr_accessor :title, :body  
  def to_s  
    "title: #{@title}\nbody: #{body}\n"  
  end
```

```

end

Perpetuity.generate_mapper_for Article do
  attribute :title
  attribute :body
end

article = Article.new
article.title = 'New Article'
article.body = 'This is an article.'

Perpetuity[Article].insert article

puts article

```

Recuperando Objetos

```

[~/rubytesting/perpetuity]$ cat app_load.rb
require 'perpetuity'
require 'perpetuity/mongodb'
require 'pp'

Perpetuity.data_source :mongodb, 'my_perpetuity_database'

class Article
  attr_accessor :title, :body
  def to_s
    "title: #{@title}\nbody: #{@body}\n"
  end
end

Perpetuity.generate_mapper_for Article do
  attribute :title
  attribute :body
end

articles = Perpetuity[Article].all
articles.each do |article| # This is when the DB gets hit
  puts article
end

```

Ejecución

```

[~/rubytesting/perpetuity]$ ruby app_load.rb
title: New Article
body: This is an article.
title: New Article
body: This is an article.
[~/rubytesting/perpetuity]$ ruby app.rb
title: New Article
body: This is an article.
[~/rubytesting/perpetuity]$ ruby app_load.rb
title: New Article
body: This is an article.

```

```
title: New Article
body: This is an article.
title: New Article
body: This is an article.
```

Consola mongo

```
[~/rubytesting/perpetuity]$ mongo
MongoDB shell version: 2.4.8
connecting to: test
Server has startup warnings:
Sat Mar 15 14:59:07.435 [initandlisten]
Sat Mar 15 14:59:07.435 [initandlisten] ** WARNING: soft rlimits too low. Number of files is 2
> show dbs
dict_dev          0.203125GB
local      0.078125GB
my_perpetuity_database 0.203125GB
sinatra-example-dev    0.203125GB
spatialapp        0.203125GB
test        0.203125GB
> use my_perpetuity_database
switched to db my_perpetuity_database
> show collections
Article
system.indexes
> db.Article.find()
{ "title" : "New Article", "body" : "This is an article.", "_id" : ObjectId("53246ee6a11460aeef")
{ "title" : "New Article", "body" : "This is an article.", "_id" : ObjectId("532472dba11460220")
{ "title" : "New Article", "body" : "This is an article.", "_id" : ObjectId("53247498a11460d52")
{ "title" : "New Article", "body" : "This is an article.", "_id" : ObjectId("532476f3a11460314")
{ "title" : "New Article", "body" : "This is an article.", "_id" : ObjectId("53247756a11460257")
{ "title" : "New Article", "body" : "This is an article.", "_id" : ObjectId("532480afa1146012f")
```

Capítulo 112

twitter

```
[~/local/src/ruby/rubytesting/twitter-test(master)]$ pry
[1] pry(main)> require "twitter"
=> true
[3] pry(main)> require "./configure"
=> true
```

Véase en la sección 8.15 los contenidos del fichero `configure.rb`

```
[4] pry(main)> Twitter.user_timeline("timoreilly").first
=> #<Twitter::Tweet:0x007fd824025c70
@attrs=
{:created_at=>"Wed Sep 18 01:08:06 +0000 2013",
:id=>380136112389234690,
:id_str=>"380136112389234690",
:text=>
"@isaiah_saxon_ If you want an intro to the folks there, I'd be glad to make it.",
:source=>
"<a href=\"http://www.tweetdeck.com\" rel=\"nofollow\">TweetDeck</a>",
:truncated=>false,
:in_reply_to_status_id=>380135816275578881,
:in_reply_to_status_id_str=>"380135816275578881",
:in_reply_to_user_id=>432803063,
:in_reply_to_user_id_str=>"432803063",
:in_reply_to_screen_name=>"isaiah_saxon_",
:user=>
{:id=>2384071,
:id_str=>"2384071",
:name=>"Tim O'Reilly",
:screen_name=>"timoreilly",
:location=>"Sebastopol, CA",
:description=>
"Founder and CEO, O'Reilly Media. Watching the alpha geeks, sharing their stories, helping
:url=>"http://t.co/5086iX7oyT",
:entities=>
{:url=>
{:urls=>
[{:url=>"http://t.co/5086iX7oyT",
:expanded_url=>"http://radar.oreilly.com",
:display_url=>"radar.oreilly.com",
:indices=>[0, 22]}],
:description=>{:urls=>[]}}},
```

```
:protected=>false,
:followers_count=>1723740,
:friends_count=>1105,
:listed_count=>25791,
:created_at=>"Tue Mar 27 01:14:05 +0000 2007",
:favourites_count=>143,
:utc_offset=>-25200,
:time_zone=>"Pacific Time (US & Canada)",
:geo_enabled=>true,
:verified=>true,
:statuses_count=>27026,
:lang=>"en",
:contributors_enabled=>false,
:is_translator=>false,
:profile_background_color=>"9AE4E8",
:profile_background_image_url=>
  "http://a0.twimg.com/profile_background_images/3587880/notes.gif",
:profile_background_image_url_https=>
  "https://si0.twimg.com/profile_background_images/3587880/notes.gif",
:profile_background_tile=>false,
:profile_image_url=>
  "http://a0.twimg.com/profile_images/2823681988/f4f6f2bed8ab4d5a48dea4b9ea85d5f1_n",
:profile_image_url_https=>
  "https://si0.twimg.com/profile_images/2823681988/f4f6f2bed8ab4d5a48dea4b9ea85d5f1_n",
:profile_link_color=>"0000FF",
:profile_sidebar_border_color=>"87BC44",
:profile_sidebar_fill_color=>"E0FF92",
:profile_text_color=>"000000",
:profile_use_background_image=>true,
:default_profile=>false,
:default_profile_image=>false,
:following=>nil,
:follow_request_sent=>false,
:notifications=>nil},
:geo=>nil,
:coordinates=>nil,
:place=>nil,
:contributors=>nil,
:retweet_count=>0,
:favorite_count=>0,
:entities=>
  {:hashtags=>[],  

   :symbols=>[],  

   :urls=>[],  

   :user_mentions=>  

    [ { :screen_name=>"isaiah_saxon_ ",  

        :name=>"Isaiah Saxon",  

        :id=>432803063,  

        :id_str=>"432803063",  

        :indices=>[0, 14] } ] },
:favorited=>false,
:retweeted=>false,
```

```

... etc., etc.
:profile_sidebar_border_color=>"87BC44",
:profile_sidebar_fill_color=>"E0FF92",
:profile_text_color=>"000000",
:profile_use_background_image=>true,
:default_profile=>false,
:default_profile_image=>false,
:following=>nil,
:follow_request_sent=>false,
:notifications=>nil},
:geo=>nil,
:coordinates=>nil,
:place=>nil,
:contributors=>nil,
:retweet_count=>0,
:favorite_count=>0,
:entities=>
{:hashtags=>[], 
:symbols=>[], 
:urls=>[], 
:user_mentions=>
[{:screen_name=>"isaiah_saxon_",
:name=>"Isaiah Saxon",
:id=>432803063,
:id_str=>"432803063",
:indices=>[0, 14]}]},
:favorited=>false,
:retweeted=>false,
:lang=>"en"}>

```

```
[5] pry(main)> Twitter.user("timoreilly").location
=> "Sebastopol, CA"
```

```
[10] pry(main)> Twitter.search("#ruby -rt", :lang => "es", :count => 1)
=> #<Twitter::SearchResults:0x007fd8249c2d50
```

```

@attrs=
{:statuses=>
[{:metadata=>{:result_type=>"recent", :iso_language_code=>"es"}, 
:created_at=>"Wed Sep 18 07:15:41 +0000 2013",
:id=>380228616136556544,
:id_str=>"380228616136556544",
:text=>"Uf mucho mas :) #Ruby.",
:source=>"web",
:truncated=>false,
:in_reply_to_status_id=>nil,
:in_reply_to_status_id_str=>nil,
:in_reply_to_user_id=>nil,
:in_reply_to_user_id_str=>nil,
:in_reply_to_screen_name=>nil,
:user=>
{:id=>1476069577,
:id_str=>"1476069577",
:name=>"Teresa Bautista",
:screen_name=>"teressabautista",
```

```

:location=>"Obregón, Sonora",
:description=>
  "no tengo idea si voy a volver a verte, pero por lo pronto contenta de conocerte",
:url=>nil,
:entities=>{:description=>{:urls=>[]}},
:protected=>false,
:follower_count=>21,
:friends_count=>21,
... etc, etc.

[11] pry(main)> Twitter.search("#ruby -rt", :lang => "es", :count => 1).results.first.text
=> "Uf mucho mas :) #Ruby."
[12] pry(main)> Twitter.search("#ruby -rt", :lang => "es", :count => 1).results.first.to_s
=> "#<Twitter::Tweet:0x007fd825070828>"

```

Véase la documentación en `Twitter::API::Search`. Para ver como usar la API de búsqueda de Twitter véanse

1. Using the Twitter Search API.

2. GET search/tweets

Hay un número de operadores que podemos usar en nuestra query:

Example	Finds tweets...
<code>twitter search</code>	containing both "twitter" and "search". This is the default operator
<code>"happy hour"</code>	containing the exact phrase "happy hour"
<code>love OR hate</code>	containing either "love" or "hate" (or both)
<code>beer -root</code>	containing "beer" but not root
<code>#haiku</code>	containing the hashtag "haiku"
<code>from:twitterapi</code>	sent from the user @twitterapi
<code>to:twitterapi</code>	sent to the user @twitterapi
<code>place:opentable:2</code>	about the place with OpenTable ID 2
<code>place:247f43d441defc03</code>	about the place with Twitter ID 247f43d441defc03
<code>@twitterapi</code>	mentioning @twitterapi
<code>superhero since:2011-05-09</code>	containing "superhero" and sent since date "2011-05-09" (year-month-day).
<code>twitterapi until:2011-05-09</code>	containing "twitterapi" and sent before the date "2011-05-09".
<code>movie -scary :)</code>	containing "movie", but not "scary", and with a positive attitude.
<code>flight :(</code>	containing "flight" and with a negative attitude.
<code>traffic ?</code>	containing "traffic" and asking a question.
<code>hilarious filter:links</code>	containing "hilarious" and with a URL.
<code>news source:tweet_button</code>	containing "news" and entered via the Tweet Button

Anatomía de un usuario Twitter

```

[13] pry(main)> Twitter.user('timoreilly')
=> #<Twitter::User:0x007fd8260c1448
@attrs=
{:id=>2384071,
:id_str=>"2384071",
:name=>"Tim O'Reilly",
:screen_name=>"timoreilly",
:location=>"Sebastopol, CA",
:description=>
  "Founder and CEO, O'Reilly Media. Watching the alpha geeks, sharing their stories, helping
:url=>"http://t.co/5086ix7oyT",
:entities=>

```

```

{:url=>
  {:urls=>
    [[:url=>"http://t.co/5086iX7oyT",
     :expanded_url=>"http://radar.oreilly.com",
     :display_url=>"radar.oreilly.com",
     :indices=>[0, 22]}],
    :description=>{:urls=>[]}}},
:protected=>false,
:followers_count=>1723753,
:friends_count=>1105,
:listed_count=>25791,
:created_at=>"Tue Mar 27 01:14:05 +0000 2007",
:favourites_count=>143,
:utc_offset=>-25200,
:time_zone=>"Pacific Time (US & Canada)",
:geo_enabled=>true,
:verified=>true,
:statuses_count=>27026,
:lang=>"en",
:status=>
  {:created_at=>"Wed Sep 18 01:08:06 +0000 2013",
   :id=>380136112389234690,
   :id_str=>"380136112389234690",
   :text=>
     "@isaiah_saxon_ If you want an intro to the folks there, I'd be glad to make it.",
   :source=>
     "<a href=\"http://www.tweetdeck.com\" rel=\"nofollow\">TweetDeck</a>",
   :truncated=>false,
   :in_reply_to_status_id=>380135816275578881,
   :in_reply_to_status_id_str=>"380135816275578881",
   :in_reply_to_user_id=>432803063,
   :in_reply_to_user_id_str=>"432803063",
   :in_reply_to_screen_name=>"isaiah_saxon_",
   :geo=>nil,
   :coordinates=>nil,
   :place=>nil,
   :contributors=>nil,
   :retweet_count=>0,
   :favorite_count=>0,
   :entities=>
     {:hashtags=>[], 
      :symbols=>[], 
      :urls=>[], 
      :user_mentions=>
        [[:screen_name=>"isaiah_saxon_",
          :name=>"Isaiah Saxon",
          :id=>432803063,
          :id_str=>"432803063",
          :indices=>[0, 14]}]},
   :favorited=>false,
   :retweeted=>false,
   :lang=>"en"}, 
  :contributors_enabled=>false,

```

```

:is_translator=>false,
:profile_background_color=>"9AE4E8",
:profile_background_image_url=>
  "http://a0.twimg.com/profile_background_images/3587880/notes.gif",
:profile_background_image_url_https=>
  "https://si0.twimg.com/profile_background_images/3587880/notes.gif",
:profile_background_tile=>false,
:profile_image_url=>
  "http://a0.twimg.com/profile_images/2823681988/f4f6f2bed8ab4d5a48dea4b9ea85d5f1_normal.jpeg",
:profile_image_url_https=>
  "https://si0.twimg.com/profile_images/2823681988/f4f6f2bed8ab4d5a48dea4b9ea85d5f1_normal.jpeg",
:profile_link_color=>"0000FF",
:profile_sidebar_border_color=>"87BC44",
:profile_sidebar_fill_color=>"E0FF92",
:profile_text_color=>"000000",
:profile_use_background_image=>true,
:default_profile=>false,
:default_profile_image=>false,
:following=>true,
:follow_request_sent=>false,
:notifications=>false}>

```

Véase También Véase

1. <https://github.com/crguezl/twitter-test> los fuentes de un ejemplo usando twitter
2. [twitter gem documentación de la gema twitter](#)
3. [En este enlace registramos nuestra aplicación en Twitter: Register your application in twitter](#)
4. [Un artículo sobre como usar la gema twitter: Data mining with Ruby and Twitter The interesting side to a Twitter API por Tim Jones en IBM developerWorks](#)
5. [Hay una versión de esta gema que provee un programa llamado t que nos da una interfaz de línea de comandos para acceder a twitter.](#)
6. [Una gema relacionada: TweetStream es una gema que proporciona acceso a la API de Twitter para streaming](#)

Capítulo 113

Ruport

- Documentación de la API: <http://api.rubyreports.org/>
- Ejemplos: <http://www.rubyreports.org/examples.html>

```
~/rubytesting/TheRubyProgrammingLanguage/chapter8ReflectionandMetaprogramming$ cat -n ruport_ex1.rb
 1  require 'ruport'
 2
 3  table = Ruport::Data::Table.new(
 4      :column_names => [ "alu", "nota"],
 5      :data => [
 6          ["Almeida Gonzalez", 4.5],
 7          ["Hernandez Perez", 6.5],
 8          ["Mendez Chavez", 4.5],
 9          ["Rodriguez Luis", 9.5]
10      ]
11  )
12
13 puts table.to_text
14
15 dubious = table.rows_with_nota(4.5)
16 dubious.each do |cal|
17     puts cal.to_csv
18 end

~$ sudo gem install ruport
Password:
Fetching: fastercsv-1.5.4.gem (100%)
Fetching: color-1.4.1.gem (100%)
Fetching: hoe-2.12.3.gem (100%)
Fetching: transaction-simple-1.4.0.gem (100%)
WARNING: transaction-simple-1.4.0 has an invalid nil value for @cert_chain
Fetching: pdf-writer-1.1.8.gem (100%)
WARNING: pdf-writer-1.1.8 has an invalid nil value for @cert_chain
Fetching: ruport-1.6.3.gem (100%)
Successfully installed fastercsv-1.5.4
Successfully installed color-1.4.1
Successfully installed hoe-2.12.3
Successfully installed transaction-simple-1.4.0
Successfully installed pdf-writer-1.1.8
Successfully installed ruport-1.6.3
```

```

6 gems installed
Installing ri documentation for fastercsv-1.5.4...
Installing ri documentation for color-1.4.1...
Installing ri documentation for hoe-2.12.3...
Installing ri documentation for transaction-simple-1.4.0...
Installing ri documentation for pdf-writer-1.1.8...
Installing ri documentation for ruport-1.6.3...
Installing RDoc documentation for fastercsv-1.5.4...
Installing RDoc documentation for color-1.4.1...
Installing RDoc documentation for hoe-2.12.3...
Installing RDoc documentation for transaction-simple-1.4.0...
Installing RDoc documentation for pdf-writer-1.1.8...
Installing RDoc documentation for ruport-1.6.3...

```

Instale también ruport-util

```

sudo gem install ruport-util
$ sudo gem install scruffy
$ sudo gem install gruff

```

```

~/rubytesting/TheRubyProgrammingLanguage/chapter8ReflectionandMetaprogramming$ gem query --loc
ruport (1.6.3)

```

```

$ export RUBYOPT=rubygems
~/rubytesting/TheRubyProgrammingLanguage/chapter8ReflectionandMetaprogramming$ ruby ruport_exam
+-----+
|     alu      | nota |
+-----+
| Almeida Gonzalez |  4.5 |
| Hernandez Perez  |  6.5 |
| Mendez Chavez    |  4.5 |
| Rodriguez Luis   |  9.5 |
+-----+
Almeida Gonzalez,4.5
Mendez Chavez,4.5

```

```

~$ gem which ruport
/Library/Ruby/Gems/1.8/gems/ruport-1.6.3/lib/ruport.rb

```

```

vi /Library/Ruby/Gems/1.8/gems/ruport-1.6.3/lib/ruport/data/table.rb

```

```

868     def method_missing(id,*args,&block)
869         return as($1.to_sym,*args,&block) if id.to_s =~ /^to_(.*)/
870         return rows_with($1.to_sym => args[0]) if id.to_s =~ /^rows_with_(.*)/
871         super
872     end

```

Vemos que la llamada dubious = table.rows_with_nota(4.5) se convierte en la línea 870 en una llamada a rows_with(nota.to_sym => args[0])

Capítulo 114

ostruct

```
~/rubytesting/TheRubyProgrammingLanguage/chapter8ReflectionandMetaprogramming$ cat -n ostruct.rb
1  require 'ostruct'
2
3  alu = OpenStruct.new
4  alu.nombre = "Jose"
5  alu.apellidos = "Rodriguez Quintero"
6  alu.edad = 97
7  alu.calificacion = 10
8
9  puts alu

~/rubytesting/TheRubyProgrammingLanguage/chapter8ReflectionandMetaprogramming$ ruby ostruct.rb
#<OpenStruct nombre="Jose", apellidos="Rodriguez Quintero", edad=97, calificacion=10>

~/rubytesting/TheRubyProgrammingLanguage/chapter8ReflectionandMetaprogramming$ cat -n my_ostruct.rb
1  class MyOpenStruct
2    def initialize(h = {})
3      @attrs = h
4    end
5
6    def method_missing(name, *args)
7      attr = name.to_s
8      if attr =~ /=\$/#
9          @attrs[attr.chop] = args[0]
10     else
11         @attrs[attr]
12     end
13   end
14
15   def to_s
16     self.inspect
17   end
18 end
19
20 alu = MyOpenStruct.new
21 alu.nombre = "Jose"
22 alu.apellidos = "Rodriguez Quintero"
23 alu.edad = 97
24 alu.calificacion = 10
25
26 puts alu.nombre
```

```
27 puts alu.edad  
28 puts alu.calificacion  
29 puts alu
```

```
~/rubytesting/TheRubyProgrammingLanguage/chapter8ReflectionandMetaprogramming$ ruby my_ostruct  
Jose  
97  
10  
#<MyOpenStruct:0x10016a408 @attrs={"nombre"=>"Jose", "calificacion"=>10, "edad"=>97, "apellido"=>"Garcia", "telefono"=>1234567890}
```

Capítulo 115

flickr

```
~$ sudo gem install flickr  
Password:  
Fetching: xml-simple-1.1.1.gem (100%)  
Fetching: flickr-1.0.2.gem (100%)  
WARNING: flickr-1.0.2 has an invalid nil value for @cert_chain  
Successfully installed xml-simple-1.1.1  
Successfully installed flickr-1.0.2  
2 gems installed  
Installing ri documentation for xml-simple-1.1.1...  
Installing ri documentation for flickr-1.0.2...  
Installing RDoc documentation for xml-simple-1.1.1...  
Installing RDoc documentation for flickr-1.0.2...
```

<http://www.flickr.com/help/api/>

Capítulo 116

Camping

```
MacBookdeCasiano:~ casiano$ sudo gem install camping
```

```
Password:
```

```
Successfully installed camping-2.1.467
```

```
1 gem installed
```

```
Installing ri documentation for camping-2.1.467...
```

```
Installing RDoc documentation for camping-2.1.467...
```

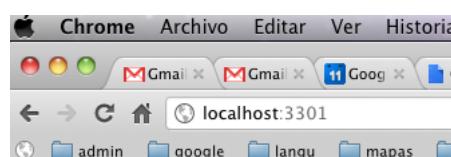
```
MacBookdeCasiano:~ casiano$ gem which camping
```

```
(checking gem camping-2.1.467 for camping)
```

```
/Library/Ruby/Gems/1.8/gems/camping-2.1.467/lib/camping.rb
```

```
MacBookdeCasiano:chapter8ReflectionandMetaprogramming casiano$ camping nuts.rb
```

```
** Starting Mongrel on 0.0.0.0:3301
```



Camping Problem!

/ not found

Figura 116.1: Visitando la página (1)

Añadimos este código a nuts.rb:

```
module Nuts::Controllers
  class Index < R '/'
    def get
      Time.now.to_s
    end
  end
end
```

y lo ejecutamos de nuevo:

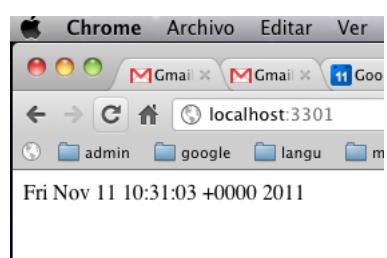


Figura 116.2: Visitando la página (2)

Parte VII

BITÁCORA DE LPP

Capítulo 117

Presentación

117.1. Información General

- *Códigos de la Asignatura:*
139 26 3014: Tercero del Grado en Ingeniería Informática
- *Bloque Formativo: Fundamentos Tecnológicos de la Ingeniería Informática*
- *Módulo: Programación*
- *Materia: Algoritmos, Estructuras de Datos y Programación*
- *Plan: G026*
- *Curso: 3º*
- *Cuatrimestre: Primero*
- *Créditos ECTS: 6*
- *Horas Teóricas: 2 (3 ECTS)*
- *Horas de Problemas: 1 (1,5 ECTS)*
- *Horas de Laboratorio: 1 (1,5 ECTS)*
- *Carácter: Obligatoria*
- *Departamento: Ingeniería Informática y Sistemas, <http://tinyurl.com/informaticaysistemas>, <http://www.ull.es/informaticaysistemas>*
- *Dirección web: <http://tinyurl.com/lpp1415>, <https://campusvirtual.ull.es/1415/course/view.php?id=5661>*

117.2. Profesores

- *Casiano Rodríguez León (Teoría, turno de tarde, grupos 3C2 y 3P3)*
 1. *Lugar Tutoría: Despacho 96 de la 4ta Planta del Edificio de Física y Matemáticas*
 2. *Horario Tutoría: Lunes, Martes, Miércoles, Jueves de 09:00 a 10:30.*
 3. *Teléfono (despacho/tutoría): 922 31 81 87*
 4. *Correo electrónico: casiano at ull.edu.es*
- *Eduardo Segredo González (Prácticas)*
 1. *Lugar Tutoría: Edificio de Informática. Laboratorio DSIC-3*
 2. *Horario Tutoría: Lunes y Jueves de 9 a 12.*

- 3. Teléfono (despacho/tutoría): 922 31 91 91
- 4. Correo electrónico: esegredo at ull dot edu dot es
- Coromoto León Hernández (Teoría, turno de mañana)

117.3. Matrícula

Para matricularse en el curso moodle se conectan al enlace <https://campusvirtual.ull.es/1415/course/view.php?> o <http://tinyurl.com/lpp1415> y usan la clave del grupo de tarde.

117.4. Contexto

1. *Fundamentos Tecnológicos de Ingeniería Informática (66 ECTS)*

- a) Programación
- b) Ingeniería de Computadores
- c) Sistemas Operativos, Sistemas Distribuidos y Redes
- d) Ingeniería del Software, Sistemas de Información y Sistemas Inteligentes
- e) Establecer una base común y amplia de fundamentos que garanticen la formación generalista del Ingeniero Informático, prestando especial atención a los conocimientos y habilidades para entender y dominar las tecnologías actuales así como para comprender y participar en la evolución futura de estas tecnologías.

2. *Perfil Profesional: Ingeniero Técnico en Informática*

117.5. Horarios

Tutorías:

- Localización: Despacho #96, 4ta Planta Edificio Física/Matemáticas.
- Reserva: Tfno. 922 31 81 87 Fax. 922 31 81 70
- e-mail: casiano@ull.es

Para la tutoría virtual contacta al profesor en google plus usando una "hangout". Mi identificador g+ de ull.edu.es es 101973567008975662199. También puede ir a la página <http://gplus.to/crguezl>. Crea una identidad g+ asociada a tu cuenta de ull.edu.es y úsala para las tutorías.

Véase también el calendario oficial de la ULL en http://www.ull.es/viewcalendar/institucional/ull/Calendario_1415.html

117.6. Ejercicio: Hágase miembro de la comunidad google+ ULL-ETSII-GRADO-LPP-14-15

Solicite hacerse miembro de la comunidad **ULL-ETSII-GRADO-LPP-14-15**

117.7. Programa de Teoría

- Tema 1. *Introducción: Modelos de Programación. Lenguajes y Herramientas de Programación. Control de Versiones. Pruebas Unitarias. Desarrollo Dirigido por Pruebas.*
- Tema 2. *Programación imperativa. Estructuras de Datos. Subprogramas. Abstracción y Encapsulamiento.*
- Tema 3. *Programación Orientada a Objetos. Clases, Objetos, Métodos. Encapsulamiento, Abs-tracción, Herencia, Polimorfismo. Mixins. Meta Programación. Lenguajes de Dominio Específi-co. Programación Orientada a Aspectos.*
- Tema 4. *Programación Declarativa: Lógica y Funcional. Funcional: Funciones de Orden Su-perior. Lambdas. Clausuras. Evaluación Perezosa. Memoización. Listas Infinitas. Declarativa: Cláusulas. Hechos. Consultas. Reglas.*
- Tema 5. *Programación Concurrente y Paralela. Corrutinas. Hilos. Procesos. Programación Dis-tribuida. Computación de Alto Rendimiento. Computación en la Nube.*

117.8. Programa de Prácticas

Se trata de prácticas tutorizadas que se realizan durante la clase con la ayuda del profesor. Usare-mos git y GitHub y Bitbucket para los repositorios remotos.

Habrán algunas Prácticas Virtuales que pueden tomar diversas formas: ejercicios online, moodle workshops, etc.

117.9. Evaluación

1. *Es obligatorio asistir a clases y hacer uso de los foros y tutorías tanto presenciales como virtuales.*
2. *El sistema de evaluación de la asignatura incluye:*
 - a) *Un examen final (pruebas objetivas) que constituye el 60 % de la calificación y*
 - b) *Evaluación continua (Valoración de las actividades prácticas en el laboratorio e Informes memorias de prácticas) que constituye el 40 % restante de la calificación.*
3. *Se aplicará un esquema de evaluación continua combinando:*
 - *Ejercicios prácticos*
 - *Prácticas de programación individuales y en grupo*
4. *En caso de no superar la evaluación continua, después de finalizar las clases del semestre el alumno dispondrá del examen de la parte práctica de la asignatura en los períodos de exámenes oficiales, con las convocatorias fijadas por la Universidad.*
5. *Para calcular la calificación final se exigirá el cumplimiento de dos condiciones:*
 - a) *Tener una puntuación total de, al menos, 5'0 puntos en cada una de las actividades prácticas de evaluación continua, o en el examen de prácticas y*
 - b) *Obtener, al menos, 5'0 puntos en el examen final.*

en caso de incumplir alguna de las condiciones anteriores, la nota final será de suspenso y como valor de la calificación aquella que no supere los 5'0 puntos.

Capítulo 118

Septiembre

1. 10/09/2014: Presentación 117

2. 16/09/2014: Tareas:

- Ruby en Windows: 80
- Ruby en Unix: Ruby Version Manager
- Instalar Git 85 81

3. 17/09/2014

- Introducción a los Lenguajes y Paradigmas de Programación (Véase la sección 1). Tarea: Leer y entender.
- Rosetta Code: A site with solutions to the same task in as many different languages as possible, to demonstrate how languages are similar and different. Tarea: Ver como se implementa el quicksort en Haskell, Prolog y Smalltalk 1
- Cloud9 92. Tarea: abrir cuenta en c9.io
- Pry. Véanse las secciones 8.1.2 y 86.3. Tarea: instalar pry.
- Tarea: Hacer cuestionario en el Moodle

4. 22/09/2014

- Véanse las secciones sobre Git en estos apuntes 85, 85.2 (tutorial de GitHub),
- Pro Git book, written by Scott Chacon
 - a) Herramientas de Programación: Control de Versiones - Git (instalación, configuración, preparación y confirmación)

5. 23/09/2014

- Tarea: Lea el artículo Programming Language de la Wikipedia
- Véanse las secciones sobre Git en estos apuntes 85, 85.2 (tutorial de GitHub),
- Tutorial de como usar criptografía de clave pública para la autentificación con SSH 85.3
- Pro Git book, written by Scott Chacon
 - a) Herramientas de Programación: Control de Versiones - Git (instalación, configuración, preparación y confirmación)
 - b) Fundamentos de Git - Obteniendo un repositorio Git

6. 24/09/2014

- Véanse las secciones sobre Git en estos apuntes 85, 85.2 (tutorial de GitHub),

- *Practica de Laboratorio #1. Sistema de control de versiones Git [GitHub, Bitbucket, Cloud9] Archivo*

7. 29/09/2014

- *Ramificaciones en Git. Véase la sección 85.4 Chapter 3. Ramificaciones en Git del libro de Scott Chacon y las anotaciones en este pdf*
 - *¿Qué es una rama?*
 - *Procedimientos básicos para ramificar y fusionar*

8. 30/09/2014

- *Ramificaciones en Git. Véase la sección 85.4 Chapter 3. Ramificaciones en Git del libro de Scott Chacon y las anotaciones en este pdf*
 - *Gestión de ramificaciones*
 - *Flujos de trabajo ramificados*
 - *Ramas Remotas*

Capítulo 119

Octubre

1. 01/10/2014

- *Ramificaciones en Git.* Véase la sección 85.4 Chapter 3. Ramificaciones en Git del libro de Scott Chacon y las anotaciones en este pdf
 - *Ramas Remotas: Deleting Remote Branches*
 - *Reorganizando el trabajo realizado: git rebase*
 - *Rewriting history Atlassian tutorial*
 - *Git Tools - Debugging with Git (git blame, git bisect)*
 - *Las herramientas de Git - Guardado rápido provisional*
 - *Git Tools - Revision Selection*
- *Bitbucket*
 - a) *Curso: Bitbucket 101: Una introducción a git usando bitbucket*
 - b) *Vídeo: Creación y Compartición de Repositorios Git en Bitbucket*

2. 06/10/2014

- a) *Estudie la sección Programación Imperativa 2*
- b) *Primeros Pasos 8.1*
- c) *irb 8.1.1*
- d) *pry 8.1.2*
- e) *Donde Encontrar Ruby y Como Instalarlo 8.2*
- f) *Donde esta Ruby en Nuestra Máquina 8.2.1*
- g) *RVM 8.2.2*
- h) *Ejecución de un Programa Ruby 8.2.3*
- i) *Ejercicio: Calcule el Factorial de un Número 8.3*
- j) *Lints in Ruby 8.4*

3. 07/10/2014

- a) *Ejecución de un Programa con el Depurador 8.5*
- b) *Averiguando que tiempo hace 8.6*

4. 08/10/2014

- a) *Práctica de Laboratorio #3. Primeros Pasos con Ruby.*
- b) *Ejercicios: Expresiones Regulares. Un programa que convierte de Celsius a Fahrenheit 8.7*
- c) *Ejercicio: Excepciones y Expresiones Regulares. Pasar de Hexadecimal a Decimal 8.8*
- d) *Bloques 8.9*

e) El método gets 8.10

5. 13/10/2014

a) Ejemplo de Clases, objetos y operadores: [crguezl/UnitTestingPoint-1 en GitHub](#)

```
[~/srcLPPRuby/UnitTestingPoint(master)]$ pwd -P  
/Users/casiano/local/src/ruby/LPP/UnitTestingPoint  
[~/srcLPPRuby/UnitTestingPoint(master)]$ git remote -v  
origin git@github.com:crguezl/UnitTestingPoint-1.git (fetch)  
origin git@github.com:crguezl/UnitTestingPoint-1.git (push)
```

b) Definición de una Clase Simple 14.1

c) Creando accessors con class_eval 15.2.2

6. 14/10/2014

a) Test/Unit 16.1

b) Ejemplo de Unit Testing: [crguezl/UnitTestingPoint-1 en GitHub](#)

```
[~/srcLPPRuby/UnitTestingPoint(master)]$ pwd -P  
/Users/casiano/local/src/ruby/LPP/UnitTestingPoint  
[~/srcLPPRuby/UnitTestingPoint(master)]$ git remote -v  
origin git@github.com:crguezl/UnitTestingPoint-1.git (fetch)  
origin git@github.com:crguezl/UnitTestingPoint-1.git (push)
```

c) Rake 87

7.

8. 15/10/2014

a) Test/Unit 16.1

b) setup y tear_down 16.1.4

c) Organizando las pruebas 16.1.3

d) Ejemplo de Unit Testing: [crguezl/UnitTestingPoint-1 en GitHub](#)

```
[~/srcLPPRuby/UnitTestingPoint(master)]$ pwd -P  
/Users/casiano/local/src/ruby/LPP/UnitTestingPoint  
[~/srcLPPRuby/UnitTestingPoint(master)]$ git remote -v  
origin git@github.com:crguezl/UnitTestingPoint-1.git (fetch)  
origin git@github.com:crguezl/UnitTestingPoint-1.git (push)
```

e) Tabla de Operadores en Ruby 14.1.7

f) Rake 87: Argumentos y Rake::TestTask 87.2

g) Comprobación de Tipos y Tipado Pato (Duck Typing) 14.1.8

h) Coerción 14.1.9

9. 20/10/2014

a) Test Driven Development (TDD) y Rspec 17

b) Test-Driven Development in Ruby Video Tutorials

10. 21/10/2014

a) Test Driven Development (TDD) y Rspec 17

11. 22/10/2014

- a) *Test Driven Development (TDD) y Rspec* 17
- b) *Atributos de clase y métodos de clase. Secciones Un Método de Clase 14.1.16 y Variables de Clase. Atributos de la Clase* 14.1.18
- c) *rspec -init* Véase la sección *rspec opciones* 17.2
- d) *Building a gem*
 - *Sección Creando Gemas* 90
 - *How To Build A Ruby Gem With Bundler, Test-Driven Development, Travis CI And Coveralls, Oh My!* by Mark McDonnell
 - *Step-by-Step Guide to Building Your First Ruby Gem*
 - *Developing a RubyGem using Bundler*

12. 27/10/2014

- a) *Subclases y Herencia* 14.3
- b) *super* 14.3.2
- c) *Visibilidad de los métodos: Público, Privado, Protegido* 14.2

13.

14. 28/10/2014

- a) *Subclases y Herencia* 14.3
- b) *Visibilidad de los métodos: Público, Privado, Protegido* 14.2
- c) *Predominancia de Métodos Privados* 14.3.3
- d) *Módulos*
 - 1) *Todas las clases son módulos* 14.1.5
 - 2) *Módulos* 14.7
 - 3) *Carga y Solicitud de Módulos* 14.9
 - 4) *Mixins estándar: Enumerable, Comparable*
 - a' *Definiendo Operadores (La Clase Point)* 14.1.6
 - b' *Enumerable*
 - c' *Comparable*
- e) *Struct.* Véase *Clases Mutables: Fácil y Rapido* 14.1.15 y *Subclases y Herencia* 14.3

15. 29/10/2014

- a) *Predominancia de Métodos Privados* 14.3.3
- b) *Constantes* 14.1.17
- c) *Struct.* Véase *Clases Mutables: Fácil y Rapido* 14.1.15 y *Subclases y Herencia* 14.3
- d) *Los Métodos Singleton y la Singleton Class o Eigenclass* 14.12
- e) *La Búsqueda por Métodos de Instancia* 14.13

Capítulo 120

Noviembre

1. Lunes 03/11/2014

- a) Los Métodos Singleton y la Singleton Class o Eigenclass 14.12
- b) Clase y Tipo de un Objeto 10.8.4 y Polimorfismo: Comprobación de Tipos y Tipado Pato (Duck Typing) 14.1.8
- c) Igualdad de Objetos 10.8.5. Ejemplo: Igualdad de Puntos 14.1.12
- d) La Búsqueda por Métodos de Instancia 14.13
- e) Guard 21. Guard es una gema que nos permite observar ficheros y realizar acciones basadas en dichos cambios.
- f) Ejemplo en c9

2. Martes 04/11/2014

- a) Guard 21. Guard es una gema que nos permite observar ficheros y realizar acciones basadas en dichos cambios.
- b) Creación e Inicialización de Objetos 14.4
- c) Limitando el Número de Instancias de una Clase 14.4.4
- d) Copia de Objetos. Marshalling 10.8.9
- e) Freezing: Congelando Objetos 10.8.12
- f) Objetos Manchados: Tainting 10.8.11

3. Miércoles 05/11/2014

- a) Variables de Clase. Atributos de la Clase 14.1.18
- b) Variables de Instancia de Clase 14.1.19
- c) Herencia y Métodos de Clase 14.3.5
- d) Herencia y Variables de la Instancia 14.3.6
- e) Herencia y Variables de Clase 14.3.7
- f) Herencia y Constantes 14.3.8
- g) Averiguando los Descendientes de una Clase (descendants) 14.3.9

4. Lunes 10/11/2014

- a) Delegación 14.3.10
- b) Métodos y Clases: Construyendo un Iterador 14.5
- c) Integración Contínua: Travis 20

5. Martes 11/11/2014

- a) *Integración Contínua: Travis 20*
- b) *Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis 36.0.2*
- c) *Sample app with Sinatra Rspec Capybara and Selenium 36.0.4*
- d) *Capybara 18*

6. *Miércoles 12/11/2014*

- a) *Integración Contínua: Travis 20*
- b) *Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis 36.0.2*
- c) *Sample app with Sinatra Rspec Capybara and Selenium 36.0.4*
- d) *Capybara 18*
- e) *Coveralls: Análisis de Cubrimiento 20.7*

7. *Lunes 17/11/2014*

- a) *Búsqueda de Métodos de Clase 14.14*
- b) *La Búsqueda de Constantes 14.15*
- c) *Coerción 14.1.9*

8. *Martes 18/11/2014*

- a) *Creando Gemas y Publicándolas en rubygems.org 90 (hasta Semantic Versioning)*

9. *Miércoles 19/11/2014*

- a) *Creando Gemas y Publicándolas en rubygems.org 90 (desde Controlando nuestras dependencias: bundle update, bundle outdated)*
- b) *RDoc 88.1*

10. *Lunes 24/11/2014*

- a) *Bloques/Blocks 8.9*
- b) *Argumentos de un Método 13.4 (Hasta Parámetros con Nombre. Named Parameters in Ruby 2.0)*

11. *Martes 25/11/2014*

- a) *Presentación de actividades de la dirección de la Sección de Informática de la ESIT.*
- b) *Argumentos Bloque (13.4.6)*
- c) *Procs y Lambdas 13.5 hasta En que Forma las Lambdas Difieren de los Procs 13.5.5 inclusive)*

12. *Miércoles 26/11/2014*

- a) *Procs y Lambdas 13.5 (Desde Enumeradores 13.5.6)*

Capítulo 121

Diciembre

1. Lunes 1/12/2014

- a) *Ejemplo: La Clase Filter 13.5.8*
- b) *Clausuras 13.6*
- c) *Objetos Method 13.8*
- d) *Programación Funcional 13.9*
 - 1) *Memoización 13.9.4*

2. Martes 9/12/2014

- a) *Reflexión y Metaprogramación 15*

Capítulo 122

Enero

Capítulo 123

Centro de Cálculo

123.1. Prácticas: Centro de Cálculo

1. *Modo de trabajo en el sistema de archivos del CC de la ETSII*
2. *Ubicación de las salas*

123.2. Despliegue de una Aplicación Web en la ETSII

Para desplegar una aplicación web usaremos exthost2 (en 2013).

Veamos que puertos están libres usando netstat :

```
casiano@exthost2:~$ netstat -an | less
```

o bien usamos lsof :

```
lsof -i | less
```

y después - si es necesario - terminamos el proceso que ya estuviera escuchando en el puerto

```
kill -9 PID
```

Veamos una simple aplicación usando rack:

```
casiano@exthost2:~/src/ruby/simplewebapp$ cat hello.rb
require 'rack'
```

```
app = lambda { |env| [200, {"Content-Type" => "text/plain"}, ["Hello. The time is #{Time.now}"]]
Rack::Handler::WEBrick.run app,:Port => 4567
```

La ejecutamos:

```
casiano@exthost2:~/src/ruby/simplewebapp$ ruby hello.rb
[2013-10-28 09:58:54] INFO  WEBrick 1.3.1
[2013-10-28 09:58:54] INFO  ruby 1.9.3 (2011-10-30) [i686-linux]
[2013-10-28 09:58:54] WARN  TCPServer Error: Address already in use - bind(2)
[2013-10-28 09:58:54] INFO  WEBrick::HTTPServer#start: pid=16597 port=4567
```

Ya tenemos disponible la página en exthost2 en el puerto correspondiente.

El acceso al servidor está limitado a la red de la ULL.

Véase también

1. *Gemas instaladas en local 99.3*

Índice general

I LENGUAJES Y PARADIGMAS DE PROGRAMACION	3
1. Introducción a los Lenguajes y Paradigmas de Programación	4
1.1. Modelos de Programación	4
1.2. Paradigmas de Programación	5
1.3. El Arte del Diseño de Lenguajes	5
1.4. El Espectro de los Lenguajes de Programación	6
1.5. ¿Porqué estudiar lenguajes?	7
2. Programación Imperativa	8
2.1. Estructuras de Datos	8
2.1.1. Tipos de Datos Abstractos/Abstract Data Types	8
2.2. Abstracción y Encapsulamiento	8
2.3. Subprogramas	9
2.4. Programación Estructurada	9
2.4.1. Go To Statement Considered Harmful	9
2.4.2. El teorema del programa estructurado	9
3. Erlang	11
4. Elixir	12
4.0.1. Ejemplo: Suma de los elementos de una lista	12
4.0.2. Quicksort	13
5. Programación Lógica	14
6. Paralelismo y Computación de Alto Rendimiento en Ruby	15
7. Go	16
II EL LENGUAJE DE PROGRAMACIÓN RUBY	17
8. Introducción	18
8.1. Primeros Pasos	18
8.1.1. irb	18
8.1.2. pry	18
8.2. Donde Encontrar Ruby y Como Instalarlo	18
8.2.1. Donde esta Ruby en Nuestra Máquina	19
8.2.2. RVM	19
8.2.3. Ejecución de un Programa Ruby	19
8.3. Ejercicio: Calcule el Factorial de un Número	19
8.4. Lints in Ruby: rubocop, reek, etc.	19
8.5. Ejecución de un Programa con el Depurador	21
8.6. Averiguando que tiempo hace	25

<i>8.7. Ejercicios: Expresiones Regulares. Un programa que convierte de Celsius a Fahrenheit</i>	26
<i>8.8. Ejercicio: Excepciones y Expresiones Regulares. Pasar de Hexadecimal a Decimal</i>	29
<i>8.9. Bloques/Blocks</i>	31
<i>8.10. El método gets</i>	35
<i>8.11. Práctica: Producto y Suma de Matrices</i>	36
<i>8.11.1. Sugerencias</i>	36
<i>8.12. Práctica: Evaluar una Expresión en Postfijo</i>	39
<i>8.12.1. Pistas</i>	39
<i>8.13. Práctica: Traducción de notación infija a postfijo</i>	43
<i>8.14. Ejercicios</i>	43
<i>8.15. Ejemplo en Ruby: Accediendo a Twitter</i>	47
<i>8.16. Práctica: Contar la Popularidad de Nuestros Amigos en Twitter</i>	52
<i>8.17. Véase También</i>	53
<i>8.18. Resolviendo Sudokus</i>	53
9. La Estructura y Ejecución de los Programas Ruby	57
<i>9.1. Estructura Léxica</i>	57
<i>9.2. Estructura Sintáctica</i>	57
<i>9.3. Estructura de Fichero</i>	57
<i>9.4. Codificación de un Programa</i>	57
<i>9.5. Ejecución de un Programa</i>	57
<i>9.6. Práctica: Descarga Páginas de la Wikipedia</i>	57
<i>9.7. Práctica: Tweet Fetching</i>	57
<i>9.8. Práctica: SQL An introduction to relational databases and their query languages</i>	57
10. Tipos de Datos y Objetos	58
<i>10.1. Números</i>	58
<i>10.1.1. Constantes Racionales en Ruby 2.1</i>	58
<i>10.2. Texto</i>	58
<i>10.3. Arrays</i>	58
<i>10.4. Hashes</i>	58
<i>10.5. Rangos</i>	58
<i>10.6. Símbolos</i>	58
<i>10.7. Booleanos y Nil</i>	58
<i>10.7.1. Véase También</i>	58
<i>10.8. Objetos</i>	59
<i>10.8.1. Referencias a Objetos</i>	59
<i>10.8.2. Vida de un Objeto</i>	60
<i>10.8.3. Identidad de un Objeto</i>	60
<i>10.8.4. Clase y Tipo de un Objeto. Polimorfismo</i>	61
<i>10.8.5. Igualdad de Objetos</i>	63
<i>10.8.6. Orden en Objetos</i>	64
<i>10.8.7. Conversión de Objetos</i>	64
<i>10.8.8. Copia de Objetos</i>	64
<i>10.8.9. Marshalling</i>	64
<i>10.8.10. Freezing: Congelando Objetos</i>	74
<i>10.8.11. Objetos Manchados: Tainting</i>	75
<i>10.8.12. Congelación: freezing</i>	76
<i>10.9. Ejercicios</i>	76
<i>10.10. Ejercicios</i>	78
<i>10.11. Práctica: Ordenar por Calificaciones</i>	83
<i>10.12. Ejercicios</i>	84
<i>10.13. Ejercicios</i>	85

11. Expresiones y Operadores	87
11.1. Literales y Palabras Reservadas	87
11.2. Variables	87
11.2.1. Variables No Inicializadas	87
11.3. Constantes	87
11.4. Invocación de Métodos	87
11.5. Asignaciones	87
11.5.1. Asignación a Variables	87
11.5.2. Asignación a Constantes	87
11.5.3. Asignación a Atributos y a Elementos de Arrays	87
11.5.4. Asignaciones Abreviadas	87
11.5.5. Asignaciones Paralelas	87
11.6. Operadores	87
11.6.1. + y - Unario	87
11.6.2. Exponenciación	87
11.6.3. Operadores Binarios	87
11.6.4. Shift y Append	87
11.6.5. Complemento, Unión e Intersección	87
11.6.6. Comparación	87
11.6.7. Igualdad	87
11.6.8. Operadores Lógicos	87
11.6.9. Rangos y Flip-Flops	87
11.6.10. Condicional ?	88
11.6.11. Operadores de Asignación	88
11.6.12. El Operador defined?	88
11.6.13. Modificadores de Sentencias (Sufijos)	88
11.6.14. Símbolos de Puntuación que no son Operadores	88
12. Sentencias y Estructuras de Control	89
12.1. Condicionales	89
12.2. Bucles	90
12.3. Iteradores y Objetos Enumerables	90
12.3.1. Enumeradores	90
12.4. Bloques	94
12.5. Alterando el Flujo de Control	99
12.5.1. throw y catch son Sentencias de Control	99
12.6. Manejo de Excepciones	100
12.7. BEGIN y END	100
12.8. Threads, Fibras y Continuaciones	100
12.8.1. Fibras	100
12.8.2. Threads	101
12.9. Práctica: Reto Dropbox. El Problema de la Dieta	101
13. Métodos, Procs, Lambdas y Clausuras	103
13.1. Definiendo Métodos Simples	103
13.1.1. Valor Retornado por un Método	103
13.1.2. Métodos y Manejo de Excepciones	103
13.1.3. Invocando un Método en un Objeto	103
13.1.4. Definiendo Métodos Singleton	103
13.1.5. Borrando (Undefining) Métodos	104
13.2. Nombres de Métodos	104
13.2.1. Métodos Operadores	105
13.2.2. Alias de Métodos	105
13.3. Métodos y Paréntesis	105

13.4. Argumentos de un Método	105
13.4.1. Parámetros por Defecto	105
13.4.2. Listas de Argumentos de Longitud Variable y Arrays	106
13.4.3. Asignando Argumentos a los Parámetros	107
13.4.4. Hashes para Argumentos con Nombre	107
13.4.5. Parámetros con Nombre. Named Parameters in Ruby 2.0	108
13.4.6. Argumentos Bloque	109
13.5. Procs y Lambdas	113
13.5.1. Creacion de Procs	113
13.5.2. Invocación de Procs y Lambdas	115
13.5.3. La Aridad de un Proc	115
13.5.4. Igualdad de Procs	115
13.5.5. En que Forma las Lambdas Difieren de los Procs	116
13.5.6. Enumeradores	120
13.5.7. Bloques para las Transacciones	125
13.5.8. Ejemplo: La Clase Filter	126
13.6. Clausuras	127
13.6.1. Clausuras y Variables Compartidas	127
13.6.2. Clausuras y Bindings	129
13.7. Repaso	130
13.8. Objetos Method	131
13.8.1. Objetos Method no Ligados (Unbound Method Objects)	132
13.9. Programación Funcional	134
13.9.1. Aplicando una Función a un Enumerable	134
13.9.2. Composición de Funciones	135
13.9.3. Aplicación Parcial de Funciones	136
13.9.4. Memoización	137
13.9.5. Símbolos, Métodos y Procs	139
13.9.6. Véase También	141
13.10 Práctica: La Calculadora	141
13.11 Ejercicios	143
13.12 Práctica: Un Motor para las Expresiones Regulares en Pocas Líneas	146
14. Clases y Módulos	150
14.1. Definición de una Clase Simple	150
14.1.1. Creando una Clase	150
14.1.2. Instanciando un Punto	150
14.1.3. Inicializando un Punto	150
14.1.4. Definiendo un método <code>to_s</code>	152
14.1.5. Acceso a los Atributos	152
14.1.6. Definiendo Operadores (La Clase Point)	154
14.1.7. Tabla de Operadores en Ruby	157
14.1.8. Polimorfismo, Comprobación de Tipos y Tipado Pato (Duck Typing)	157
14.1.9. Coerción	159
14.1.10. Acceso a Arrays y a Hashes	160
14.1.11. Enumeración de Coordenadas	161
14.1.12. Igualdad de Puntos	161
14.1.13. Ordenando Puntos	162
14.1.14. Un Punto Mutable	163
14.1.15. Creando Clases con Struct	164
14.1.16. Un Método de Clase	165
14.1.17. Constantes	167
14.1.18. Variables de Clase. Atributos de la Clase	168

14.1.19.Variables de Instancia de Clase	168
14.1.20.Práctica: La Clase Punto del Plano	170
14.2. Visibilidad de los métodos: Público, Privado, Protegido	173
14.2.1. Ejemplo de Visibilidad de Métodos Protegidos	177
14.2.2. Ejemplo de Visibilidad	178
14.3. Subclases y Herencia	179
14.3.1. Heredando Métodos	180
14.3.2. super	180
14.3.3. Predominancia/Invalidación de Métodos (overriding)	182
14.3.4. Aumentando la Conducta Mediante Encadenamiento	185
14.3.5. Herencia y Métodos de Clase	186
14.3.6. Herencia y Variables de la Instancia	187
14.3.7. Herencia y Variables de Clase y Variables de Instancia de una Clase	188
14.3.8. Herencia y Constantes	190
14.3.9. Averiguando los Descendientes de una Clase (descendants)	191
14.3.10 Delegación	192
14.4. Creación e Inicialización de Objetos	194
14.4.1. new, allocate e initialize	194
14.4.2. Métodos Factoría	195
14.4.3. dup, clone e initialize_copy	196
14.4.4. Limitando el Número de Instancias de una Clase	199
14.5. Métodos y Clases: Construyendo un Iterador	200
14.6. Práctica: Conjuntos	202
14.7. Módulos	203
14.7.1. Los Módulos como Espacios de Nombres	203
14.7.2. Los Módulos como Mixins	206
14.7.3. Ejemplo	211
14.7.4. Módulos Espacio de Nombres Incluibles	211
14.8. Prepend	212
14.9. Carga y Solicitud de Módulos	213
14.9.1. El Camino de Búsqueda	213
14.9.2. Ejecutando Código Cargado	213
14.9.3. Autoloading de Módulos	214
14.10. Práctica: Matrices	214
14.11. Práctica: Matrices Dispersas	214
14.12. Los Métodos Singleton y la Singleton Class o Eigenclass	216
14.13. La Búsqueda por Métodos de Instancia	217
14.14. Búsqueda de Métodos de Clase	219
14.15. La Búsqueda de Constantes	220
14.16. Véase	221
14.17. Jugador de TicTacToe (Tres en Raya)	221
14.17.1. Ejemplo de Partida	221
14.17.2. Programa Principal	223
14.17.3. La Clase Game	223
14.17.4. La Clase Player: Un Ejemplo de Strategy Pattern	224
14.17.5. La Clase Board	225
14.17.6. La Clase Row	227
14.17.7. Código Completo del TicTacToe	228

15. Reflexión y Metaprogramación	234
15.1. Tipos, Clases y Módulos	234
15.1.1. Antepasados y Módulos	235
15.1.2. Definiendo Clases y Módulos	237
15.2. Evaluando Strings y Bloques	239
15.2.1. Bindings (encarpetados) y eval	240
15.2.2. instance_eval y class_eval	241
15.2.3. instance_exec y class_exec	244
15.3. Variables y Constantes	244
15.3.1. Buscando, Dando Valores y Suprimiendo Variables y Constantes	245
15.4. Métodos	246
15.4.1. Listando y Comprobando Métodos	246
15.4.2. Obteniendo los Métodos de Objetos	246
15.4.3. Llamando a los Métodos Dinámicamente	246
15.4.4. Definiendo, Suprimiendo y Haciendo Alias de Métodos	247
15.4.5. Manejando Métodos No Definidos: method_missing	248
15.4.6. Ejercicios	249
15.5. Hooks (Ganchos)	250
15.5.1. El Hook inherited	250
15.5.2. El Hook included	250
15.5.3. Ganchos/Hooks: Sustituyendo (y delegando en) un método existente	253
15.5.4. Ganchos/Hooks: Interviniendo en el Momento en que un Objeto es Creado	253
15.6. Trazo	255
15.6.1. set_trace_func	255
15.6.2. caller	256
15.7. Los Módulos ObjectSpace y GC	257
15.8. Estructuras de Control a la Carta	259
15.8.1. Creando una Palabra Clave	259
15.9. Métodos Missing y Constantes Missing	263
15.9.1. Creando Dinámicamente los Métodos dentro de method_missing	263
15.10 Creando Métodos Dinámicamente	265
15.10.1 Definiendo Métodos con class_eval	265
15.10.2 Definiendo Métodos con define_methods	265
15.11 Encadenamiento de Alias	265
15.12 Lenguajes de Dominio Específico. Domain Specific Languages. DSL	265
15.12.1. Un Lenguaje de Dominio Específico para Describir Recetas de Cocina	265
15.12.2. Un DSL para Procesar Documentos XML usando instance_eval	268
15.12.3. Dos DSLs: Generando XML con Validación via Generación de Métodos	273
15.12.4. Creando un ORM	278
15.13 Práctica: DSL: Redacción de Cuestionarios I (Sin Contexto)	279
15.14 Práctica: DSL: Redacción de Cuestionarios II (Con Contexto)	281
15.15 Práctica: HTML DSL	283
15.16 Práctica: HTML DSL con Git y Rake	285
15.17 Repaso	286
15.18 Repaso	294
15.19 Repaso	298
15.20 Referencias. Véase También	302
16. Pruebas Unitarias	303
16.1. Test/Unit	303
16.1.1. Ejemplo Sencillo de uso de test/unit	303
16.1.2. Cuando una assertion falla	304
16.1.3. Organizando las pruebas	306

16.1.4. Setup y Teardown	307
16.1.5. Veáse	308
16.2. Minitest	308
17. Pruebas. Test Driven Development (TDD) y Rspec	309
17.1. Introducción	309
17.1.1. 0: Red	309
17.1.2. 1: Green	310
17.1.3. 2: Red	311
17.1.4. 3: Green	312
17.1.5. 4: refactor	312
17.1.6. 5: Red: should and should_not	313
17.1.7. 6: Green	315
17.1.8. 7: un poco mas tarde	316
17.2. rspec opciones	318
17.3. Resumen y Enlaces	319
18. Pruebas. Capybara y Cucumber	322
18.1. Introducción	322
18.2. Usando Capybara Directamente	322
18.2.1. Usando la Sesión Directamente	322
18.2.2. Usando Capybara Directamente Incluyendo el Módulo Capybara::DSL	325
18.2.3. Aprendiendo Capybara con Ficheros HTML Locales	326
18.3. Ejemplo de uso de Capybara, RSpec y Cucumber con una Aplicación Web Externa	327
18.3.1. Poltergeist	330
18.4. Capybara y Rack::Test	331
18.4.1. Testeando una Aplicación Sinatra	332
18.4.2. Testing con Rack::Test y Cucumber	335
18.5. La API de Capybara	338
18.5.1. Búsqueda de Elementos HTML	338
18.5.2. Clicks en Botones y Enlaces	339
18.5.3. Envío de Formularios	340
18.6. Buscadores/Finders, Ámbito/Scope y Coincidencias/Matches Múltiples	345
18.7. Véase	349
19. Pruebas. Other Testing tools	350
20. Integración Contínua: Travis	351
20.1. Uso de Travis	351
20.2. Enlaces	354
20.3. Notificaciones/Notifications	355
20.4. Límites de Tiempo/Build Timeouts	355
20.5. Databases and other services	355
20.6. Travis en la Línea de Comandos	356
20.7. Coveralls: Análisis de Cubrimiento	358
21. Guard	360
22. Programación Orientada a Eventos	365
22.1. Introducción a Programación Orientada a Eventos	365
22.2. Building our own I/O Event Loop	366
22.3. EventMachine	366
22.3.1. Un server	369
22.3.2. Deferrable	372

22.3.3. Véase	374
22.4. Véase	374
23. Programación distribuida/Distributed Programming	375
23.1. DRb	375
23.2. Distributed Ruby and SSH: drbssh	375
24. Actores	376
24.1. Celluloid	376
24.1.1. DCell	376
24.2. El Problema de los Filósofos Comensales / The Dining Philosophers Problem	378
24.3. Véase	379
25. Threads	380
25.1. Enlaces /Véase	380
25.2. Creación de Threads	380
25.3. Exclusión Mutua y la Clase Mutex	382
25.4. Un Servidor Multithreaded (and a Client)	384
25.5. Colas	386
25.6. El Problema de los Filósofos Comensales / The Dining Philosophers Problem	387
26. Juegos con Gosu	389
26.1. Enlaces	389
26.2. El Juego de la Vida	389
26.3. Starfighter	389
26.4. Random bouncy particles using ruby Gosu games library	392
27. Shoes	394
28. Herramientas para el Control de la Calidad	395
28.1. Lints: rubocop y reek	395
28.2. Véase	396
29. La Plataforma Ruby	397
29.1. Expresiones Regulares	397
30. El Entorno Ruby	398
30.1. Invocando al intérprete Ruby	398
30.2. El Entorno al Nivel mas Alto	398
30.3. Atajos para la Extracción e Informes (Tipo Perl)	398
30.4. Llamando al Sistema Operativo	398
30.5. Seguridad	398
30.6. El Compilador de Ruby	398
III SINATRA	399
31. Rack, un Webserver Ruby Modular	400
31.1. Introducción	400
31.2. Analizando env con pry-debugger	402
31.2.1. Introducción	402
31.2.2. REQUEST_METHOD, QUERY_STRING y PATH_INFO	404
31.3. Detectando el Proceso que está Usando un Puerto	405
31.4. Usando PATH_INFO y erubis para construir una aplicación (Noah Gibbs)	406
31.5. HTTP	407

31.5.1. Introducción	407
31.5.2. Sesiones HTTP	408
31.5.3. Métodos de Petición	409
31.5.4. Véase	410
31.6. Rack::Request y Depuración con pry-debugger	410
31.6.1. Conexión sin Parámetros	410
31.6.2. Conexión con Parámetros	412
31.7. Rack::Response	414
31.7.1. Introducción	414
31.7.2. Ejemplo Simple	415
31.7.3. Ejemplo con POST	415
31.8. Cookies y Rack	417
31.9. Gestión de Sesiones	422
31.9.1. Ejercicio: Race Conditions. Concurrencia	424
31.10 Ejemplo Simple Combinando Rack::Request, Rack::Response y Middleware (Lobster)	426
31.11 Práctica: Accediendo a Twitter y Mostrando los últimos twitts en una página	429
31.12 Ejemplo: Basic Authentication con Rack::Auth::Basic	429
31.13 Redirección	433
31.14 La Estructura de una Aplicación Rack: Ejemplo de Middleware	434
31.15 rackup	434
31.16 Rack::Static	437
31.17 Un Ejemplo Simple: Piedra, Papel, tijeras	440
31.17.1 Práctica: Rock, Paper, Scissors: Debugging	446
31.17.2 Práctica: Añadir Template Haml a Rock, Paper, Scissors	446
31.17.3 Práctica: Añada Hojas de Estilo a Piedra Papel Tijeras	448
31.18 Middleware y la Clase Rack::Builder	449
31.19 Ejemplo de Middleware: Rack::ETag	454
31.20 Construyendo Nuestro Propio Rack::Builder	455
31.21 Código de Rack::Builder	457
31.22 Rack::Cascade	460
31.23 Rack::Mount	462
31.24 Rack::URLMap	463
31.25 El método run de Rack::Handler::WEBrick	464
31.26 Documentación	466
31.27 Pruebas/Testing	466
31.27.1 Pruebas Unitarias	466
31.27.2 Rspec con Rack	469
31.28 Práctica: Añada Pruebas a Rock, Paper, Scissors	471
31.29 Prácticas: Centro de Cálculo	472
31.30 Despliegue de una Aplicación Web en la ETSII	472
31.31 Práctica: Despliegue en Heroku su Aplicación Rock, Paper, Scissors	472
31.32 Faking Sinatra with Rack and Middleware	472
31.33 Véase También	473
32. Primeros Pasos	474
32.1. Introducción	474
32.1.1. Referencias sobre Sinatra	474
32.1.2. Ejercicio: Instale la Documentación en sinatra.github.com	474

33. Fundamentos	475
33.1. Ejemplo Simple de uso de Sinatra	475
33.2. Rutas/Routes	475
33.2.1. Verbos HTTP en Sinatra/Base	479
33.3. Ficheros Estáticos	479
33.4. Vistas	480
33.4.1. Templates Inline	480
33.4.2. Named Templates	482
33.4.3. Templates Externos	483
33.4.4. Templates Externos en Subcarpetas	484
33.4.5. Variables en las Vistas	486
33.4.6. Pasando variables a la vista explícitamente via un hash	488
33.4.7. Opciones pasadas a los Métodos de los Templates	490
33.5. Filtros	491
33.6. Manejo de Errores	492
33.7. The methods body, status and headers	493
33.8. Acceso al Objeto Request	494
33.9. Caching / Caches	494
33.10 Sesiones y Cookies en Sinatra	494
33.11 Downloads / Descargas / Attachments	498
33.12 Uploads. Subida de Ficheros en Sinatra	499
33.13 halt	501
33.14 Passing a Request	501
33.15 Triggering Another Route: calling call	502
33.16 Logging	502
33.17 Generating URLs	504
33.18 Redireccionamientos/Browser Redirect	504
33.19 Configuration / Configuración	505
33.20 Configuring attack protection	505
33.21 Settings disponibles/Available Settings	506
33.22 Environments	507
33.23 Correo	508
33.24 Ambito	508
33.25 Sinatra Authentication	511
33.25.1 Autentificación Básica en Sinatra con Rack::Auth::Basic	511
33.25.2 Ejemplo con Warden	512
33.25.3 Referencias	512
33.26 Sinatra como Middleware	512
33.27 Práctica: Aplicación Web con Sinatra: Contar la Popularidad de Nuestros Amigos en Twitter	513
33.28 Práctica: TicTacToe	514
33.29 Práctica: TicTacToe usando DataMapper	523
33.30 Práctica: Servicio de Syntax Highlighting	523
34. Sinatra desde Dentro	530
34.1. tux	530
34.2. Aplicación y Delegación	530
34.3. Helpers y Extensiones	530
34.4. Petición y Respuesta	530
35. Aplicaciones Modulares	531

36. Testing en Sinatra	532
36.0.1. Véase	532
36.0.2. Build a Sinatra API Using TDD, Heroku, and Continuous Integration with Travis	532
36.0.3. Mini MiniTest Tutorial	534
36.0.4. Sample app with Sinatra Rspec Capybara and Selenium	535
37. CoffeeScript y Sinatra	539
38. Streaming	540
38.1. Introducción	540
38.2. Streaming y Valores de Retorno	545
38.3. Sinatra usando Streaming, Rack MiddleWare y map	546
38.4. Enlaces Relacionados	548
38.5. A simple demonstration of streaming Redis pub/sub data	548
38.6. Comet y Server Sent-Events	552
38.6.1. Ejemplo Simple	553
38.6.2. Enlaces Relacionados	554
38.6.3. Chat Utilizando Streaming y Server Sent Events (SSE)	555
38.6.4. Código Completo del Chat	560
38.6.5. Chat Simple	561
38.6.6. Práctica: Chat con Mensajes Individuales	562
38.6.7. Práctica: Chat con Estilo	562
38.6.8. Práctica: Chat con TicTacToe	562
38.7. Embedding Sinatra within EventMachine	563
38.8. Ejemplo de Server Sent Events: irb en el navegador	566
38.9. Asynchronous responses in Rack	593
38.9.1. Deferred or streaming response bodies	594
38.10 Código en sinatra/base.rb: la clase Stream	595
39. Web Sockets	597
39.1. WebSockets	597
39.1.1. Que es WebSocket y para que sirve	597
39.1.2. Negociación del protocolo WebSocket	597
39.2. websocket/rack	598
39.3. Ruby y WebSockets: TCP for the Browser	598
39.4. Una Aplicación Usando Websockets en la que Múltiples Clientes Dibujan en un Lienzo	600
39.4.1. Enlaces Relacionados	603
39.5. Using WebSockets on Heroku with Ruby	604
40. Openid y Sinatra	606
40.1. Referencias. Véase Tambien	606
41. Bootstrap your Web Application with Ruby and Sinatra	607
41.1. BootStrap	607
42. Ajax in Sinatra	608
42.1. Un Programa para Experimentar con las Expresiones Regulares Usando Ajax	608
42.2. Un Ejemplo Simple	609
42.3. Ajax, Sinatra y RightJS	613
42.4. Un Chat con Ajax y JQuery	613
42.5. Práctica: Chat Usando Ajax y jQuery	616
42.6. Práctica: TicTactoe Usando Ajax	616
43. Redis y Sinatra	618

44. MongoDB y Sinatra	619
45. Building Backbone.js Apps With Ruby, Sinatra, MongoDB and Haml	620
46. DataMapper y Sinatra	621
46.1. Introducción a Los Object Relational Mappers (ORM)	621
46.2. Introducción al Patrón DataMapper	621
46.3. Ejemplo Simple de uso de DataMapper: Agenda	622
46.4. Ejemplo de Uso de DataMapper: Canciones de Sinatra	627
46.5. Ejemplo de uso de Sinatra y DataMapper: Acortador de URLs	635
46.6. Configurando la Base de Datos en Heroku con DataMapper. Despliegue	635
46.7. Asociaciones Simples	637
46.8. Asociaciones Through	640
46.9. Práctica: Acortador de URLs	642
46.10Práctica: Estadísticas de Visitas al Acortador de URLs	643
47. Sequel y Sinatra	647
48. ActiveRecord y Sinatra	648
48.1. Práctica: Servicio para Abreviar URLs	648
48.2. Práctica: Servicio para Abreviar URLs Teniendo en Cuenta el País de Visita	654
49. Google Plus y Sinatra	655
49.1. Ejemplo Simple	655
50. Las Manos en la Masa: Nuestro Propio Blog Engine	657
51. Depuración en Sinatra	658
51.1. Depurando una Ejecución con Ruby	658
52. Despliegue en Heroku	661
52.1. Introducción	661
52.2. Logging	669
52.3. Troubleshooting	672
52.4. Configuration	674
52.5. Make Heroku run non-master Git branch	674
52.6. Account Verification and add-ons	675
52.7. Véase	675
52.8. Práctica: Despliegue en Heroku	675
53. Envío de SMSs y Mensajes: Twilio y Clockworks	677
54. Rest	678
54.1. Introducción	678
54.2. Ejemplo de un servicio RESTfull en Sinatra	680
55. Sinatra + Sprockets + Slim + Sinatra-reloader Example	682
56. Sinatra::Flash	683
57. Alternativas a Sinatra	685
57.1. Cuba	685
57.2. Grape	685
57.3. Ramaze	685

58. Padrino	686
58.1. Introducción	686
58.2. Generadores	693
58.3. Enlaces	695
59. Desarrolladores de Sinatra	696
59.1. Konstantin Haase	696
 IV JAVASCRIPT, HTML y CSS	 697
60. Introducción	698
60.1. Challenge Problem	698
60.1.1. Iterative	698
61. Objetos	699
61.1. Tutoriales de OOP en JavaScript en la Web	699
61.2. Ejercicios	699
61.3. Comprobando Propiedades	701
61.4. Enumeración de Propiedades	702
62. Funciones	703
62.0.1. Los Métodos call y apply	703
62.1. Programación Funcional	704
63. Clases y Módulos	705
63.1. Herencia	705
63.2. Ejercicios	706
64. Ajax	708
64.1. Enlaces y Bibliografía	708
64.2. JSON y JSONP: When ajax calls are in a different domain from the server.	708
65. JavaScript en el Lado del Servidor	709
65.1. Instalar Node.js	709
65.2. Primeros Pasos. Un Ejemplo Simple	709
65.3. Usando REPL desde un programa	710
65.4. Usando REPL via un socket TCP	712
65.5. Referencias sobre REPL	712
65.6. Entrada Salida en Node.js	712
65.7. Debugger	712
65.8. Modulos	712
65.8.1. Introducción	712
65.8.2. Ciclos	713
65.8.3. Especificación de Ficheros Conteniendo Módulos	714
65.8.4. Carga desde Carpetas node_modules	714
65.8.5. Las Carpetas Usadas Como Módulos	715
65.8.6. Caching	715
65.8.7. El Objeto module y module.exports	716
65.8.8. Algoritmo de Búsqueda Ejecutado por require	716
65.9. Como Crear tu Propio Módulo en Node.js	717
65.9.1. Introducción	717
65.9.2. Un Fichero package.json	717
65.9.3. README y otros documentos	718
65.9.4. Véase También	722

65.10 JQuery en Node.js	723
65.11 Mas sobre Node	723
66. Backbone	724
67. Closure Tools	725
67.1. Véase También	725
68. Semantic Templates	726
68.1. Moustache	726
69. Pruebas	727
69.1. Testing en JavaScript: Fácil y Rápido	727
69.2. Unit Testing, TDD y BDD con Jasmine	727
70. Buenas Prácticas y Patrones	728
70.1. Véase También	728
71. Herramientas para JavaScript	729
71.1. npm	729
71.1.1. Specifying dependencies in Node.js	729
71.2. n	729
71.3. Google Chrome y Javascript	730
71.4. Plugins, Editores, IDEs	730
71.5. Grunt	730
71.6. Beautifiers, Pretty-Printers	730
71.7. Modulos	730
72. Google Maps JavaScript API	731
73. Mobile Web Applications Development with HTML5	732
74. Semantic Templates en Javascript	733
74.1. Moustache	733
74.2. handlebars	733
75. CSS	734
76. Bootstrap: Javascript y Hojas de Estilo	735
76.1. BootStrap	735
77. JQuery Mobile	736
V COFFESCIPT	738
78. Introducción	739
79. Ambito/Scope	741
VI HERRAMIENTAS	742
80. Ruby en Windows	743

81.Ruby Version Manager: RVM	744
81.1. Instalación de RVM	744
81.2. Actualización	744
81.3. Versiones Conocidas del Intérprete	745
81.4. Instalar un Intérprete	745
81.5. Intérpretes Instalados	746
81.6. Suprimir un intérprete	746
81.7. <i>.rvmrc</i> , <i>.ruby-version</i> y <i>.ruby-gemset</i>	747
81.8. <i>rvmrc</i>	747
81.9. Gemsets	749
81.10 <i>Lista de Comandos para trabajar con gemsets</i>	749
81.11 <i>Ejemplo de uso de gemsets</i>	749
81.12 <i>RVM gives you a separate gem directory for each and every Ruby version and gemset</i>	750
81.13 <i>Desinstalar rvm</i>	751
81.14 <i>Véase También</i>	751
81.15 <i>Práctica: RVM: Instalación y Manejo</i>	751
82.rbenv	752
83.RubyGems: installing, updating and removing rubygems	753
83.1. Introducción	753
83.2. Servidores de Gemas	756
83.2.1. <i>Geminabox</i>	756
84.ssh	757
84.1. Introducción	757
84.2. Conexión SSH a Una máquina por Primera Vez	758
84.3. Claves Pública y Privada: Estableciendo Autentificación No Interactiva	760
84.4. El fichero <i>authorized_keys</i>	764
84.5. Deshabilitar la Asignación de una TTY	766
84.6. Agentes SSH	766
84.7. Mejor un Sólo Agente	769
84.8. Redireccionado al Agente SSH	774
84.9. Consideraciones sobre la Seguridad del Uso de Agentes	776
84.10 <i>Depuración/Debugging</i>	777
84.11 <i>Los Ficheros de Configuración</i>	779
84.12 <i>Copia Segura de un Fichero</i>	782
84.12.1. <i>Transferencia de ficheros por sftp</i>	783
84.12.2. <i>Copias de Seguridad con rsync</i>	784
85.Git	788
85.1. Instalar Git	788
85.2. Introducción al uso de Git con GitHub en GitHub	788
85.3. Un poco de SSH	788
85.4. Ramificaciones en Git	788
85.5. Véase: <i>Tutoriales de Git</i>	789
85.6. Diferencias entre <i>fetch</i> y <i>pull</i>	789
85.7. Mezclando Ficheros Específicos desde otra Rama	789
85.8. Configuración y Algunos alias	790
85.8.1. Alias <i>lg</i>	790
85.8.2. Auto-completado	791
85.8.3. Poner la rama en el prompt de la shell	791
85.9. Hub	792
85.10 <i>Vim y Git. FuGitive: a Git wrapper so awesome, it should be illegal</i>	792

86.Debugging, Depurando	793
86.1. Véase	793
86.2. ruby-debug Cheat Sheet	793
86.3. Depuración con Pry	793
87.Rake	794
87.1. Argumentos en una Tarea	794
87.2. Rake::TestTask	795
87.3. Enlaces	797
88.Documentando el Código: RDoc, Yard, UML	798
88.1. RDoc	798
88.1.1. Opciones	798
88.1.2. RDoc y Rake	798
88.1.3. Servidor de Documentación Local	799
88.1.4. Ejemplo	799
88.1.5. Buenas Normas de Documentación	801
88.1.6. Enlaces	801
88.2. YARD	801
88.3. Diagramas UML	802
88.3.1. umlify	802
89.Bundler	803
89.1. bundle show: Que versión de una gema estamos usando	803
89.2. bundle show: searching	803
89.3. La opción PATH (:path) del método gem	804
89.4. Véase	804
90.Creando Gemas y Publicándolas en rubygems.org	805
90.1. Eligiendo Nombre	805
90.2. Creando la Estructura Inicial de la Gema	805
90.3. Underscores and Dashes	807
90.4. Dependencias de la Gema	807
90.5. Jerarquía de Ficheros y Directorios, Nombres y Requires	808
90.6. Gemas con Un Gran Número de Utilidades Independientes	809
90.7. Sharing Source Code From a Public git Repository	810
90.8. Publishing to RubyGems.org	810
90.9. gem owner: Compartiendo la Propiedad de una Gema	811
90.10gem cert: Gem Security	812
90.11gem yank: Retirando una Versión de una Gema de RubyGems.org	814
90.12Semantic Versioning	815
90.13Declaring dependencies: Runtime vs. Development	816
90.14Optimistic and Pessimistic Version Constraint	817
90.15Controlando nuestras dependencias: bundle update, bundle outdated	818
90.16Local Git Repos	820
90.17Badges	821
90.18Preparando la Documentación	821
90.18.1Inch	821
90.19Enlaces Relacionados	821

91. Herramientas para la Programacion en Pares	822
91.1. Introduction to Pair Programming	822
91.1.1. Remote pair programming	823
91.2. tmux	823
91.2.1. Utilidades para Pair Programming con tmux	823
91.3. MadEye	823
91.4. Screenhero	824
92. Cloud9	825
92.1. Introducción: Cloud IDEs y Cloud9	825
92.2. Instalación local	825
92.3. Ruby en Cloud9	826
92.4. Programación en Parejas y Cloud9	826
92.5. Rails en Cloud9	826
92.6. Cloud9 y GitHub: How do I push my Cloud9 project to GitHub	827
92.7. Deploy your app to Heroku using the Cloud9 console	828
93. OAuth	829
93.1. Introduction to OAuth	829
93.2. Google Developers Console	833
93.2.1. Managing projects and applications	833
93.2.2. Keys, access, security, and identity	834
93.3. OmniAuth gem: Standardized Multi-Provider Authentication for Ruby	835
93.3.1. Auth Hash Schema	836
93.4. OmniAuth OAuth2 gem	837
93.5. The gem omniauth-google-oauth2	837
93.6. Ejemplos de uso de omniauth	840
93.7. Práctica: Usando OAuth con omniauth	840
93.8. Using OAuth 2.0 to Access Google APIs	840
93.9. Google OAuth 2.0 Playground	841
93.10 Sign-in with Google +	841
93.11 Revoking Access to an App	841
93.12 Google + API for Ruby	841
93.13 Google+ Sign-In for server-side apps	842
93.14 Authentication using the Google APIs Client Library for JavaScript	842
94. Mac OS homebrew Package Manager	843
95. PostgreSQL	845
96. SASS (Syntactically Awesome StyleSheets)	847
96.1. Introducción	847
96.2. Modos de Uso	847
96.3. Referencias	848
97. Haml	849
97.1. Filters	849
97.2. HTML5 Custom Data Attributes	851
98. Selenium	852
98.1. Introducción	852
98.2. Componentes	852
98.3. Setting Up a Selenium-WebDriver Project	852

99.ETSII	854
99.1. Prácticas: Centro de Cálculo	854
99.2. Despliegue de una Aplicación Web en la ETSII	854
99.3. Gemas instaladas en local	855
100jsFiddle	856
101Codenvy	857
101.1Introducción	857
102Editores Entornos de Desarrollo	858
103Website wireframes	859
104Markdown	860
104.1.Véase	860
105Jekyll	861
105.1Introducción	861
105.2Jekyll en GitHub	864
105.2.1.User and Organization Pages	864
105.2.2Project Pages	865
105.2.3.Véase	865
105.3Jekyll en Heroku	865
106Showoff	866
107GitHub	867
107.1Organizations	867
107.2GitHub Pages	867
108Vim	868
108.1Plugin para parejas (paréntesis, etc.)	868
108.2Plugins para HTML	868
108.3Plugins para git	868
108.4.Vim airline	868
109Apache	869
109.1Apache en Mac OS X	869
109.1.1.Configuring Apache	869
109.1.2.PHP	870
109.1.3.Perl	870
109.1.4.MySQL	872
109.1.5.Enlaces	872
110DataMapper	873
111Perpetuity	874
112twitter	877
113Ruport	883
114poststruct	885
115flickr	887

VII BITÁCORA DE LPP	890
117Presentación	891
117.1Información General	891
117.2Profesores	891
117.3Matrícula	892
117.4Contexto	892
117.5Horarios	892
117.6Ejercicio: Hágase miembro de la comunidad google+ ULL-ETSII-GRADO-LPP-14-15	892
117.7Programa de Teoría	893
117.8Programa de Prácticas	893
117.9Evaluación	893
118Septiembre	894
119Octubre	896
120Noviembre	899
121Diciembre	901
122Enero	902
123Centro de Cálculo	903
123.1Prácticas: Centro de Cálculo	903
123.2Despliegue de una Aplicación Web en la ETSII	903

Índice de figuras

<i>15.1. Formulario generado</i>	274
<i>20.1. Página de Travis Mostrando los Resultados de unas Pruebas con Rspec</i>	354
<i>31.1. La pila Rack</i>	451
<i>33.1. El Objeto Request</i>	528
<i>33.2. La pila Rack</i>	529
<i>38.1. Típica disposición de un chat</i>	563
<i>39.1. Múltiples clientes pueden dibujar en el lienzo</i>	605
<i>61.1. Jerarquía de Prototipos Nativos</i>	701
<i>61.2. <code>--proto--</code> and prototypes</i>	701
<i>116.1.Visitando la página (1)</i>	888
<i>116.2.Visitando la página (2)</i>	889

Índice de cuadros

Índice alfabético

- ámbito, 238, 239
send, 139
- abstract data type*, 8
- Abstracta*, 183
- abstracta*, 183
- Abstraction*, 8
- acceptance tests*, 327, 335
- adapters*, 400
- After filters*, 492
- Agent forwarding*, 774
- agent forwarding*, 776
- Ajax*, 611
- Ajax Push*, 552
- ancestors*, 179
- ApacheBench*, 565
- aplicación modular sinatra*, 531
- aplicación parcial*, 136
- aplicaciones clásicas sinatra*, 531
- aridad*, 115
- Array*, 66, 77
- assignment expressions*, 153
- Asynchronous JavaScript and XML*, 611
- Atom*, 679
- Authentication Hash*, 835
- authorization server*, 830
- auto-incrementing*, 623
- BA*, 429
- backwards-compatible versions*, 818
- bare hash*, 108
- BasicObject*, 241
- Before filters*, 491
- bind*, 140
- Binding*, 240, 241
- binding*, 129, 240, 241
- bindings*, 129
- bloque*, 109
- bloques*, 113
- browser cookie*, 495
- código de estado*, 408
- callback*, 611
- capa de autentificación*, 758
- capa de conexión*, 758
- capa de transporte*, 758
- cascading-by-chaining*, 127
- case equality*, 63
- catch*, 99
- CDN*, 610
- chaining*, 127
- CI*, 351
- claves foráneas*, 637
- Class*, 237
- class_eval*, 242–244, 264, 294
- class_exec*, 244
- class_variables*, 244
- clausura*, 127
- client*, 830
- closures*, 31, 94
- Cloud9*, 826
- cloud9*, 826
- Code blocks*, 31, 94
- Comet*, 552
- Comet applications*, 552
- compiladores*, 4
- concreta*, 183
- concurrencia*, 7
- CONNECT*, 410
- constants*, 244
- Content Delivery Network*, 610
- context*, 225
- contexto*, 225
- continuous delivery*, 351
- Continuous integration*, 351
- controladores de eventos*, 611
- cookie*, 417, 495
- core module*, 714
- coroutine*, 92, 122
- curried proc*, 137
- curry*, 137
- custom data attributes*, 736
- Data Mapper*, 622
- data-driven programming*, 365
- Declarative programming*, 5
- declarative programming*, 8
- default context*, 629
- defer*, 368
- deferrables*, 545
- define_method*, 132, 238, 239

DELETE, 410
dependency graph, 819
Desarrollo Dirigido por las Pruebas, 471
descendants, 179
Development dependencies, 816
development dependencies, 807
development set of documentation, 799
Digital Signature Algorithm, 757
Dining Philosophers problem, 378
Distributed Ruby, 375
DOM, 558
DRb, 375
DRY, 166
DSA, 757
Duck Typing, 159
EDP, 365
Edsger Djisktra, 378
Eigenclass, 216
eigenclass, 141, 216, 217, 236, 237
ejemplos, 327, 335
Ejercicio
 Instale la Documentación en sinatra.github.com,
 474
embedded systems, 365
encabezado, 408
Encapsulation, 8, 173
end-user, 830
entity tag, 454
env, 403
escenarios, 327, 335
estrategias, 225
eval, 129, 239–241, 245
event handlers, 611
event handling, 365
event selection, 365
event-based programming, 365
event-driven programming, 365
Evented servers, 542
evented servers, 542
EventSource, 557
examples, 327, 335
expansiones de macros, 4
extend, 211, 236, 237, 251
external iterator, 90, 120
external templates, 480
factory methods, 195
features, 327, 335
Fibonacci, 138
File, 77
finders, 345
First-party cookies, 418
Fixnum, 258
foreign keys, 637
four rules of simple design, 313
FTP, 783
función de devolución, 611
garbage collection, 60
gemsets, 749
GET, 409
Gherkin, 336
GIL, 382
Global Interpreter Lock, 382
global_variables, 244
GoF, 225
gsub, 84
handlers, 400
hash, 63
HEAD, 409
headless browser, 324
hooks, 250
host key, 759
hotlink, 610
HTML templates, 480
HTTP Basic authentication, 429
HTTP cookie, 495
HTTP Streaming, 552
idempotent, 679
immediate values, 104
Imperative programming, 5, 8
include, 206, 251
include?, 236
included, 250, 252
infinite sequences, 92, 122
inherited, 250
inline templates, 480
instance_eval, 241, 242, 244, 263, 267, 268, 294
instance_exec, 244
instance_method, 139
instance_variables, 244
internal iterator, 90, 120
IO, 65, 66
iterador, 109
JavaScript Object Notation, 611
jsFiddle, 856
JSON, 611
Kent Beck, 313
Kernel, 253
línea de estatus, 408
labeled break, 99
lambda, 113
lenguaje máquina, 4

lenguajes dataflow, 7
lenguajes de restricciones, 7
lenguajes de scripting, 7
lenguajes de von Neumann, 7
lenguajes dinámicos, 7
lenguajes funcionales, 7
lenguajes lógicos, 7
Lenguajes orientados a objetos, 7
load, 213
local_variables, 238, 244
Logic programming, 5

método singleton, 104
métodos abstractos, 183
métodos factoría, 195
métodos introspectivos, 234
métodos singleton, 237
módulo, 203
MAJOR level, 816
man-in-the-middle-attack, 759
Marshal, 64
matcher, 314
Math, 212
member functions, 306, 307
memoizacion, 137
Metaclass, 216
metaclass, 236
metalanguage, 234
metaprogramación, 234
method, 131, 139
method cascading, 127
Method chaining, 126
method lookup, 217
method resolution, 217
method_missing, 218
microformats, 679
middleware, 400, 440
MINOR level, 816
mixin, 206
Module, 237, 242, 298
module functions, 212
module_eval, 242
module_exec, 244
module_function, 212

name, 132, 238
named context, 629
nesting, 237
netcat, 712
nullipotent, 679

Oauth, 51
Object, 246, 253
object language, 234

Object-oriented programming, 5
ObjectSpace, 257, 258
Online Javascript IDE, 856
operador de splat, 106
optimistic version constraint, 817
OPTIONS, 410
owner, 132

page, 736
Pair Programming, 823, 826
parameters, 132
partially done, 715
PATCH, 410
PATCH level, 816
patrón estrategia, 225
persistent, 496
persistent cookie, 418
pessimistic version constraint, 818
polimorfismo, 157
Polymorphism, 62
popen, 68
POST, 409
Práctica
 Añada Hojas de Estilo a Piedra Papel Tijeras, 448
 Añada Pruebas a Rock, Paper, Scissors, 471
 Añadir Template Haml a Rock, Paper, Scissors, 446
 Accediendo a Twitter y Mostrando los últimos twitts en una página, 429
 Acortador de URLs, 642
 Aplicación Parcial, 137
 Aplicación Web con Sinatra: Contar la Popularidad de Nuestros Amigos en Twitter, 513
 Chat con Estilo, 562
 Chat con Mensajes Individuales, 562
 Chat con TicTacToe, 562
 Chat Usando Ajax y jQuery, 616
 Conjuntos, 202
 Contar la Popularidad de Nuestros Amigos en Twitter, 52
 Descarga Páginas de la Wikipedia, 57
 Despliegue en Heroku, 675
 Despliegue en Heroku su Aplicación Rock, Paper, Scissors, 472
 DSL: Redacción de Cuestionarios I (Sin Contexto), 279
 DSL: Redacción de Cuestionarios II (Con Contexto), 281
 Estadísticas de Visitas al Acortador de URLs, 643
 Evaluar una Expresión en Postfijo, 39
 HTML DSL, 283

HTML DSL con Git y Rake, 285
La Calculadora, 141
La Clase Punto del Plano, 170
Matrices, 214
Matrices Dispersas, 214
Ordenar por Calificaciones, 83
Procesos Concurrentes, 74
Producto y Suma de Matrices, 36
Reto Dropbox. El Problema de la Dieta, 101
Rock, Paper, Scissors: Debugging, 446
RVM: Instalación y Manejo, 751
Servicio de Syntax Highlighting, 523
Servicio para Abreviar URLs, 648
Servicio para Abreviar URLs Teniendo en Cuenta el País de Visita, 654
SQL An introduction to relational databases and their query languages, 57
TicTacToe, 514
TicTactoe Usando Ajax, 616
TicTacToe usando DataMapper, 523
Traducción de notación infija a postfix, 43
Tweet Fetching, 57
Un Motor para las Expresiones Regulares en Pocas Líneas, 146
Usando OAuth con omniauth, 840
Primary keys, 623
private, 157
Proc, 71, 110, 241
proc, 113
Process, 65, 66
Process::Status, 66, 67
programming model, 4
programming paradigm, 5
Programming paradigms, 5
protected, 157
public, 157
public_method, 131
public_send, 139
Publish/Subscribe, 549
pura, 137
PUT, 410

Rack, 400
Rack middleware, 400
RDOC, 149, 281, 801
RDOC::Markup, 149, 281, 801
reactor design pattern, 545
Reactor pattern, 366
receiver, 132
red (failing), 309
Refactoring, 312
refactoring, 312
reflection, 234
reflexivity, 234

Remote Pairing With SSH, 823
Representational state transfer (REST), 678
require, 213
resource owner, 830
resource server, 830
REST, 678
RESTful, 678
role, 736
router, 462
RSA, 757
Runtime dependencies, 816
runtime dependencies, 807

safe method, 679
Sandi Metz, 159
scenario, 327, 335
scope gates, 238
SCP, 782
secure attribute, 418
secure cookie, 418
security policy, 814
semántica de invocación, 120
semántica yield, 120
semantic versioning, 817
Semantic Versioning standard, 815
send, 139
Server Push, 552
Server Sent Events, 552
server sent events, 562
Server-Sent Event, 553
server-sent events, 557
Server-Sent Events (SSE), 557
sesión, 407
session cookie, 418
session identifier, 422, 423
Session management, 422
session token, 422
setter method, 153
SFTP, 783
Simple Object Access protocol, 678
Singleton Class, 216
singleton class, 216, 217, 236, 237
singleton method, 165, 196
singleton_class, 236
SOAP, 678
source_location, 132
SSE, 557
SSH, 757
step definitions, 328
steps, 327, 335
strategies, 835
strategy, 225
strategy pattern, 225
streaming, 494

String, 71, 143, 239
superclases, 219
Symbol, 258
syntactic sugar, 127

TDD, 309, 471
Template View, 480
test case, 306
Test-Driven Development, 309
tests, 306

The Bastards Book of Ruby, A Programming Primer for Counting and Other Unconventional Tasks, 57

The Gang of Four, 225

The Uniform Access Principle, 173

Third-party cookies, 418

thread-safe, 382

Threaded servers, 542

throw, 99

TicTacToe, 223

Tipado Pato, 159

Tmux, 823

to_proc, 131, 139

to_s, 238

TRACE, 410

tracking cookie, 495

tracking cookies, 418

Tres en Raya, 223

twiddle-wakka, 818

type, 62

type-versus-class, 62

unbind, 133

Unit testing, 303

unobtrusive approach, 736

use, 440

utility computing, 678

versioning policy, 815

von Neumann model, 5

web cache validation, 454

Web service APIs, 678

web services, 678

WebSocket, 597

WebSocket JavaScript interface, 597

When Trouble Strikes, 24

XP, 351

YARD, 149, 281

Bibliografía

- [1] *Nguyen Dan.* The Bastards Book of Ruby. A Programming Primer for Counting and Other Unconventional Tasks. <http://ruby.bastardsbook.com/>, 2012.
- [2] *David Flanagan and Yukihiro Matsumoto.* The Ruby Programming Language. O'Reilly. <http://www.netpro.me/uploads>, first edition, 2008.
- [3] *Gregory T. Brown.* Ruby Best Practices. O'Reilly Media, Inc. <http://www.humbug.in/docs/ruby-best-practices/index.html>, 1st edition, 2009.
- [4] *Jeffrey E.F. Friedl.* Mastering Regular Expressions. O'Reilly, USA, 1997. ISBN 1-56592-257-3.
- [5] *Russ Olsen.* Design Patterns in Ruby. Addison-Wesley Professional, 2008.
- [6] *Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.* Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
- [7] *Dave Thomas, Chad Fowler, and Andy Hunt.* Programming Ruby 1.9: The Pragmatic Programmers' Guide. Pragmatic Bookshelf, 3rd edition, 2009.
- [8] *Paolo Perrotta.* Metaprogramming Ruby. The Pragmatic programmers, 2010.
- [9] *Russ Olsen.* Eloquent Ruby. Addison-Wesley Professional, 1st edition, 2011.
- [10] *M. Robbins.* Application Testing with Capybara. Community experience distilled. Packt Publishing, 2013.
- [11] *Claudio Riva, Mikael Blomberg, and Håkan Mitts.* Mobile Web Applications Development with HTML5. <http://aaltowebapps.com/index.html>, 2012.
- [12] *Scott Chacon.* Pro Git. Apress, Berkely, CA, USA, 1st edition, 2009.