> Git _

- Sistema de control de versiones distribuido.
- Desarrollado por Linus Torvalds.
- Permite controlar los cambios realizados en la aplicación y distribuir de forma simple el código.

> Git _

- Configuración inicial:
  - Configuración a nivel de sistema:
    - `$ git config --global user.name "Su nombre"`
    - `$ git config --global user.email mi@correo.com`
- Creación del repositorio:
  - Ir al directorio del proyecto
    - `$ git init`
  - Git realiza un track de todos los archivos. Para evitar que se incluyan archivos que cambian constantemente se los debe agregar al archivo `.gitignore`:

```
.bundle
db/*.sqlite3
log/*.log
tmp/**/*
```

> Git _

- Agregar archivos al repositorio:
  - `$ git add .`
    - Git se encarga de agregar todos los archivos ubicados en el directorio actual de forma recursiva.
    - Los archivos se agregar a un área conocida como "escenificación" (*staging*).
- Para conocer el estado de los archivos:
  - `$ git status`
- Para almacenar los cambios (localmente):
  - `$ git commit –m "Proyecto Inicial"`

> Git _

- Por qué usar Git o un Sistema de Control de Cambios?
  - Qué tal si por error borran un directorio?
    - `$ rm –rf app/`
  - Recuperación de datos:
    - `$ git checkout –f`

> GitHub _

- Sitio social de código optimizado para almacenar y compartir repositorios de Git.
- Por qué colocar una copia del repositorio en Git?
  - Respaldo completo del repositorio
  - Permite la colaboración

> GitHub _

- Acceder a ww.github.com
  - Registrarse y crear una nueva cuenta (UserName).
  - Crear un repositorio.
- Enviar su aplicación a Git:
  - `$ git remote add origin git@github.com:UserName/ejemplo.git`
  - `$ git push origin master`

> Heroku _

- Opción para despliegue de aplicaciones escritas en RoR.
- El despliegue es sencillo si se emplea código bajo el sistema de versionamiento Git.

> Llaves SSH _

- SSH es un protocolo de red para comunicaciones seguras.
- Emplea criptografía de llave pública para autenticar.

> Generación de Llaves SSH _

- Generar una llave pública:
  - $ ssh-keygen –t rsa
- La llave se almacenará en ~/.ssh/id_rsa

> Heroku _

- Instalar la gema de Heroku:
  - `$ [sudo] gem install heroku`
- Agregar la llave generada
  - `$ heroku keys:add`
- Comprobar que la llave ha sido agregada:
  - `$ heroku keys`

> Heroku _

- Desde el directorio de la aplicación ejecutar:
  - `$ heroku create`
- Enviar la aplicación a heroku:
  - `$ git push heroku master`
- Si hay un error, editar el archivo gemfile y modificar sqlite:
  - `gem 'sqlite3-ruby', :group => :development`

> Heroku _

- Mediante el explorador web ingresar a la dirección especificada en Heroku.

> Ruby _

- Lenguaje Interpretado
- Orientado a Objetos
  - Todo es un objeto
  - Toda operación es una llamada a un método en algún objeto
- Tipos dinámicos: los objetos tienen un tipo, no las variables.
- Dinámico:
  - Metaprogramación: agregar, modificar código en tiempo de ejecución.
  - Reflexión: capacidad de preguntar a un objeto sobre si mismo.

> Convenciones _

- Clase:
  - Pascal Case

```
class Usuario
…
end
```

- Métodos y variables:
  - Snake Case

```
def es_usuario?
…
end

def calculo_iva
…
end
```

> Convenciones _

- Constantes:
```
MODO_PRUEBA = true
```
- Globales:
```
$MODO_PRUEBA = false
```
- Símbolos: Cadenas inmutables cuyo valor es el mismo símbolo
```
comida_favorita = :pizza
:pizza.to_S == 'pizza'
'pizza'.to_sym == :pizza
:pizza == 'pizza'            # FALSO
```

> Variables, Arreglos, Hashes _

- No hay declaraciones.
  - Las variables locales deben estar asignadas antes de ser usadas.
  - Variables de instancia y de clase son `nil` hasta ser asignadas.

```
x = 3; x = 'pizza'
```

- Arreglos:
```
x = [1, 'dos', :tres]
x[1] == 'dos'
x.lenght == 3
```
- Hash:
```
x = {'a'=>1, :b=>[2, 3]}
x[:b][0] == 2
x.keys == ['a', :b]
```

> Métodos _

- Todo se pasa por referencia.

```
def suma(x, y)
   return [x, y+1]
end

def suma(x, y=0)
   [x, y+1]
end

def suma(x, y=0); [x, y+1]; end
```

- Para llamar al método:

```
a, b = suma(1, 2)
a, b = suma(1)
```

> Comparadores y Booleanos _

- Básicos:

```
==
!=
<
>
=~
!~
true
false
nil
```

> Cadenas / Expresiones Regulares _

- Strings:

```
"Esto es una cadena"
%Q{cadena}
'Esto es una cadena'
%q{cadena}
A = 41
"El resultado es #{A+1}"
```

- Expresiones Regulares:

```
"david@dominio.COM" =~ /(.*)@(.*)\.com$/i
```

Si no hay valores coincidentes, el resultado es falso.

Si hay valores coincidentes, las variables `$1`, ..., `$n` capturan los valores entre paréntesis.

> Todo es un objeto _

- Todo es un objeto:
  - Incluso un int, nil o true
  - Todas las llamadas a métodos se realizan en objetos.

```
57.methods
nil.respond_to?(:to_S)
```

- Las conversiones implícitas no son parte de los tipos del sistema, sino son métodos de instancia.
- Método send:

```
1 + 2                   1.send(:+, 2)
miCadena.lenght         miCadena.send(:length)
miArreglo[4]            miArreglo.send(:[], 4)
miArreglo[4]="abc"      miArreglo.send(:[], 4, "abc")
if( x == 3 )            if( x.send(:==, 3))
miFuncion(5)            self.send(:miFuncion, 5)
```

> Todo es un objeto _

- **a.b** significa:
  - Llamar al método **b** en el objeto **a**.
  - **a** es el receptor, al cuál se le enviará la llamada, asumiendo que **a** conoce cómo responder a dicho método.

  No significa: `b` es una variable de instancia (propiedad) de `a`.

  No significa: `b` es un miembro de la estructura `a`.

> Todo es un objeto _

- Cada operación es una llamada a un método.

```
y = [1, 2]
y = y + [“abc”, :def]
y << 5
y << [6, 7]
```

- Métodos de instancia. No son operadores del lenguaje.
  - 5 + 3, “a” + “b”, [a, b] + [c, d] (métodos denominados +)
    - Numeric#+,
    - String#+
    - Array#+

> Hashes _

```
h = {“nombre” => “David”, :edad =>
31 }
h.has_key?(“nombre”)
h[“no hay esta llave”]
h.delete(:edad)
```

> Modo Poeta _

- Los hashes permiten pasar argumentos “similares a palabras”.
- Las {} pueden omitirse si el último argumento de un método es un hash.
- Los () pueden omitirse.
- Ejemplo:

```
link_to(“Edit”,
{:controller=>’estudiantes’, :action =>
‘edit’ }
```

- ```
  link_to “Edit”,
  :controller=>’estudiantes’, :action =>
  ‘edit’
  ```
  ```

> Modo Poeta _

- Ejemplo:

```
a.should(be.send(:>=, 7))
a.shoud(be >= 7)
a.should be >= 7
```

- `(redirect_to(login_page)) and return() unless logged_in?`

```
redirect_to login_page and return unless logged_in?
```

> Clases y Herencia_

- Ejemplo:

```
class Ahorro < Cuenta
    def initialize(balance_inicial = 0)
            @balance = balance_inicial
    end
    def balance
            @balance
    end
    def balance=(valor)
            @balance = valor
    end
    def deposito(valor)
            @blance += valor
    end
    @@nombre_banco = "Mi Banco"
    def self.nombre_banco
            @@nombre_banco
    end
end
```

> Clases y Herencia_

- Constructor:

```
def initialize(balance_inicial = 0)
      @balance = balance_inicial
end
```

- Variable de Instancia (no es accesible desde otras clases):

```
@balance
```

- Método de Instancia (*getter*):

```
def balance
      @balance
end
```

- Método de Instancia (*setter*):

```
def balance=(valor)
      @balance=valor
end
```

> Clases y Herencia_

- Variable de Clase:

```
@@nombre_banco = "Mi Banco"
```

- Método de Clase:

```
def self.nombre_banco
     @@nombre_banco
end
```

> Clases y Herencia_

- *Shortcut*:

```
class Ahorro < Cuenta
  def initialize(balance_inicial=0)
      @balance = balance_inicial
  end

  attr_accessor :balance
end
```

> Cuenta Internacional _

- Ejemplo:

```
cuenta.deposito(100)
cuenta.deposito(100.euros)
```

- No hay problema con clases abiertas:

```
class Numeric
     def euros ; self * 1.6 ; end
end
```

- Pero que tal si:

```
cuenta.deposito(1.euro)
```

- O tal vez:

```
cuenta.deposito(1.yen)
cuenta.deposito(300.rubis)
```

> Cuenta Internacional _

- Metodo method_missing

```
class Numeric
  @@tipos
={'yen'=>0.01, 'euro'=>1.2, 'rupee'=>0.9}

  def method_missing(method_id, *args, &block)
    tipo = method_id.to_s.gsub( /s$/, '')
    if @@tipos.has_key?(tipo)
      self * @@tipos[tipo]
    else
      super
    end
  end
end
```

> Lazos _

- Ejemplos

```
["manzana", "pera", "mora"].each do |fruta|
     puts frita
end

for i in (1..10) do
     puts i
end

1.upto 10 do |numero|
     puts numero
End

3.times { print "hola, " }
```

> Iterador _

- Permite que el objeto maneje su recorrido:

```
(1..10).each do |x| ... end
(1..10).each { |x| ... }
1.upto(10) do |x| ... end
```

- Recorrido en rango:

```
miArreglo.each do |x| ... end
```

- Recorrido en arreglo:

```
miHash.each_key do |llave| ... end
miHash.each_pair do |llave, valor| ... end
```

- Recorrido en hash:

```
10.times do { ... }
```

> Ejemplos _

- Permite que el objeto maneje su recorrido:

```
x=['manzana', 'pera', 'mora']
x.sort
x.uniq.reverse
x.reverse!  # destructivo, modifica x
x.map do |fruta|
     fruta.reverse
end.sort

x.collect { |f| f.include?("e") }

x.any? { |f| f.length>5 }
```

> Módulos _

- Un módulo es una colección de clases y métodos de instancia que no pertenecen a una clase.
  - No es necesario instanciarlos.
- Se puede hacer un mix de un método en una clase:

```
class A < B
    include Modulo1
end
```

  - Un método se buscará en A, luego en Modulo1 y finalmente en B.

> Mix-in: Contrato _

- Ejemplo: **Enumerable** asume que los objetos de la clase responden a **each**.
  - Provee: all?, any?, collect, find, include?, map, ...
- Ejemplo: **Comparable** asume que los objetos de la clase responden a **<=>**.
  - Provee: <, >, =>, <=, ==, between?, ...
- La clase del objeto no importa, solo importa que respondan a los métodos.

> Módulo vs Clase _

- Módulo: reúso de comportamiento
  - Comportamientos que se pueden aplicar a varias clases.
  - Ejemplo: **Enumerable, Comparable**.
  - Mecanismo: mixin (include Enumerable)
- Clase: reúso de implementación
  - Subclase reúsa o sobreescribe métodos de la superclase.
  - Mecanismo: herencia (A < B)

RoR

- La idea básica de diseñar una aplicación en Rails consiste en:
  - Crear el esqueleto de la aplicación
  - Definir las rutas
  - Especificar la base de datos y las migraciones
  - Definir modelos y ActiveRecord
  - Definir controladores y vistas

> RoR y MVC_

- Modelo:
  - Subclase de ActiveRecord::Base
  - Capa de asociación de objetos con entidades relacionales.
  - `models/*.rb`
- Vista:
  - Subclase de ActionView
  - `views/*.html.erb`
  - `Views/*.html.haml`
- Controlador:
  - Subclase de ApplicationController
  - `controllers/*.rb`

Creación del esqueleto

- Definir la raíz de la aplicación.
- Crear un directorio y dentro de este ejecutar:

```
$ rails new mispeliculas -T
```

- Ingresar a `mispeliculas/`

Gemas

- Ruby incluye una librería estándar con una basta colección de clases.
- Las gemas son alternativas para expandir la funcionalidad de Ruby (librería).
- La gema **Bundler** busca el archivo `Gemfile`, el cual especifica las gemas requeridas así como las versiones de las gemas.
- `Gemfile` permite automatizar la instalación de las diferentes gemas.

Gemfile

- Agregar en el archivo Gemfile:

```
gem 'haml'

group :development,  :test do
   gem 'ruby-debug19'
end
```

Instalación de gemas

- Ejecutar:

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

- **Bundler** es un ejemplo de automatización para repetición.
  - Permite instalar las gemas automáticamente, en lugar de instalarlas una por una de forma manual.

Rutas

- Abrir un terminal para monitorear los errores.

```
$ tail –f log/develpment.log
```

- Ingresar a la aplicación: http//:localhost:3000
- Ingresar a la aplicación: http//:localhost:3000/peliculas
- Ejecutar

```
$ rake routes
```

> Funcionamiento básico de RoR _

1. Una Ruta asocia un URL con una acción del controlador y extrae posibles parámetros.
   - Los parámetros están disponibles en el hash `params[]`.
2. La acción del controlador instancia variables visibles desde la vista.
   - Subdirectorios y archivos en `views/` concuerdan con los nombres del controlador y las acciones.
3. La acción del controlador eventualmente renderiza la vista

> Funcionamiento básico de RoR _

- URL: http://localhost:3000/peliculas/3
- Ruta:
  - `config/routes.rb`

```
GET /peliculas/:id {action=>'show', controller=>'peliculas'}
```

- Controlador:
  - `app/controllers/peliculas_controller.rb`

```
def show
  id=params[:id]
  @pelicula = Pelicula.find(id)
end
```

- Vista:
  - `app/views/peliculas/show.html.haml`

```
%li
  Nombre =@pelicula.calificacion
```

> Filosofía de Rails _

- Convención en lugar de configuración.
  - Si los nombres siguen una convención, no es necesario archivos de configuración.

PeliculasController#show

en

peliculas_controller.rb

Entonces la vista está en:
views/peliculas/show.html.haml

Rails y RESTful

- Rails permite crear rutas RESTful, para las cuatro operaciones básicas (CRUD) de un modelo.
  - Ruta RESTful: especifica una solicitud autocontenida de la operación a realizar y las entidades o recursos en las cuales debe realizarse esta operación.

Creación de Rutas

- Editar el archivo config/routes.rb:

```
Mispeliculas::Application.routes
.draw do
  resources :peliculas
  root :to => redirect('/peliculas')
end
```

Creación de Rutas

- Eliminar el archivo `public/index.html`
- Ejecutar:

```
$ rake routes
```

- Ingresar a: http://localhost:3000/peliculas

> Base de Datos _

- Datos valiosos del usuario.
- Solución de Rails:
  - Tres ambientes:
    - Desarrollo
    - Producción
    - Pruebas
  - Cada ambiente dispone de su propia base de datos.
  - Migración: script que indica los cambios que se deben realizar en las diferentes bases de datos.

> Base de Datos _

- Ventaja de migración:
  - Se puede identificar cada migración, y conocer cuál y cuándo será aplicada.
    - Pueden ser reversibles
  - Puede manejar control de versionamiento.
  - Automatización
    - Repetición confiable
  - Herramienta para automatizar

> Generador de Código _

- El generador de código permite generar los diferentes elementos de forma automática.

  `rails generate migration create_peliculas`
  - Crea la migración, pero no la aplica.
  - La migración está definida en `db/migrate`
- Para aplicar la migración:

  `rake db:migrate`
  - Al aplicar la migración, se aplican los cambios en la base de datos.
- Para aplicar la migración en producción:

  `heroku db:migrate`
- Para deshacer los cambios:

  `rake db:rollback`

> Generador de Código _

- ####_create_peliculas.rb:

```
class CreatePeliculas < ActiveRecord::Migration
  def up
    create_table 'peliculas' do |t|
      t.string 'titulo'
      t.string 'calificacion'
      t.text 'descricion'
      t.datetime 'fecha_estreno'
      t.timestamps
    end
  end

  def down
    drop_table 'peliculas'
  end
end
```

> CRUD _

- Rails genera las sentencias SQL en tiempo de ejecución basado en el código Ruby.
- Operaciones básicas: Create, Read, Update, Delete:

INSERT
SELECT
UPDATE
DELETE

> Modelo _

- Subclase de ActiveRecord::Base
  - Conecta el modelo a la base de datos.
  - Provee las operaciones CRUD.
- El nombre de la tabla se deriva del nombre del modelo.
- Los nombres de las columnas, son atributos en el modelo (getter/setter)

> Modelo _

- Crear el archivo app/models/pelicula.rby agregar:

```
class Pelicula < ActiveRecord::Base
end
```

- Para explorar la funcionalidad, iniciar la consola de Rails:

```
$ rails console
```

> Modelo _

- ActiveRecord emplea convención en lugar de configuración:
  1. Usa el nombre de la clase (**Pelicula**) para determinar el nombre de la tabla (**peliculas**).
  2. Consulta la base de dato para determinar que columnas existen en la tabla.
     - De esta manera conoce que atributos y que tipos son legales.
  3. Asigna atributos (*getter* y *setter*).

> Modelo _

- Los objetos del modelo están en memoria y son independientes de las copias en la base de datos.
  - La copia debe actualizarse de forma explícita.
  - Los métodos **save** y **create** causan que el objeto se escriba en la base de datos.
  - Cambiar los atributos no provoca que se escriba en la base de datos.

> Modelo _

- Para buscar objetos se emplea `method_missing`.
- El método `find_by_`**`atributo`** busca un **atributo** que corresponda al atributo del modelo.

```
find_by_calificacion_and_fecha_estreno
```

- El método `find_by` retorna un objeto, el método `find_all_by` retorna un conjunto de objetos (`Enumerable`)

> Modelo _

- Create:
  - Método `create`:
    - Si algo falla retorna nil
  - Método `create!`
    - Si algo falla genera una excepción
    - Peligroso

> Modelo _

- Create:

```
starwars =
Pelicula.Create!(:titulo=>'Star wars',
:fecha_estreno=>'25/4/1977',
:calificacion => 'PG')

piratas =
Pelicula.New(:titulo=>'Piratas del
Caribe', :fecha_estreno=>'Oct 27,
2004', :calificacion => 'PG-13')
piratas.save!
```

> Modelo _

- Read:

```
pelicula_pg =
Pelicula.where("calificacion = 'PG'
")

Pelicula_pg=Pelicula.find_by_califi
cacion('PG')

Pelicula.find_by_id(1)
```

> Modelo _

- Update:

```
Starwars.update_attributes(:descrip
cion=>"Increibles efectos")

pelicula_pg.calificacion='PG-13'
pelicula_pg.save!
```

> Modelo _

- Delete:

```
pelicula_pg.destroy!

Pelicula.find_all_by_calificacion('
PG').each do |x|
  x.destroy
end
```

> Seed _

- Permite poblar la base de datos.
- Copiar en `db/seeds.rb`:

```
varias_peliculas =
 [{:titulo=>'Aladin', :calificacion=>'G', :fecha_estreno='10-nov-1992'},
{:titulo=>'Jinetes', :calificacion=>'G', :fecha_estreno='10-oct-1998'},
{:titulo=>'Todos', :calificacion=>'PG-15', :fecha_estreno='1-feb-1992'},
{:titulo=>'Nadie', :calificacion=>'PG-13', :fecha_estreno='14-jun-2005'}]

Pelicula.send(:attr_accessible, :titulo, :calificacion, :fecha_estreno)
varias_peliculas.each do | pelicula |
        Pelicula.create!(pelicula)
end
```

- Ejecutar

```
$ rake db:seed
```

> Controladores y Vistas _

- Rutas RESTful generadas:
  - index
  - show
  - new/create
  - edit/update
  - destroy

> Controladores y Vistas _

- Las Rutas RESTful generadas esperan que el controlador defina las siguientes acciones:
  - Index
  - Show
  - New/Create
  - Edit/Update
  - Destroy

> Acción _

- La acción debe retornar una colección de todas las películas que se encuentran en la tabla.
- Se requiere un método de controlador que obtenga la colección y una vista HTML para presentar estos datos.

> Acción _

- Convención en lugar de configuración:
  - La clase **Pelicula** se define en el archivo `app/models/pelicula.rb`
  - El controlador **PeliculasController,** se define en `app/controllers/peliculas_controller.rb`
  - Cada método del controlador debe nombrarse usando snake_case, de acuerdo a la acción que manipula,
    - El método show manipula la acción Show.
  - La vista se define en `app/views/peliculas/show.html.haml`

> Acción _

- Crear el archivo `app/controllers/peliculas_controller.rb` y especificar la acción **index**:

```
class Peliculas_Controller < ApplicationController
   def index
         @peliculas = Pelicula.all
   end
end
```

> Acción _

- Crear el directorio `app/views/peliculas/`
- Crear el archivo `app/views/peliculas/index.html.haml` y especificar:

> Acción _

```
%h2 Listado de peliculas
%table#peliculas
    %thead
            %tr
                    %th Titulo
                    %th Calificacion
                    %th Fecha Estreno
                    %th Mas informacion
    %tbody
       - @peliculas.each do |pelicula|
            %tr
                    %td= pelicula.titulo
                    %td= pelicula.calificacion
                    %td= pelicula.fecha_estreno
                    %td= link_to "Mas sobre
#{pelicula.titulo}", pelicula_path(pelicula)
```

> Helper _

- Método helper:
  - **link_to** permite crear vistas. Su primer argumento es un string que aparecerá como un enlace en la página, y el segundo argumento se emplea para crear el URI del enlace.
  - **pelicula_path()**: Toma una instancia de un recurso y genera el URI para dicho objeto

    `/peliculas/:id`

> Helper _

Helper	URI	Ruta RESTful	Acción
peliculas_path	/peliculas	GET /peliculas	index
peliculas_path	/peliculas	POST /peliculas	create
new_pelicula_path	/peliculas/new	GET /peliculas/new	new
edit_pelicula_path(m)	/peliculas/1/edit	GET /peliculas/:id/edit	edit
pelicula_path(m)	/peliculas/1	GET /peliculas/:id	show
pelicula_path(m)	/peliculas/1	PUT /peliculas/:id	update
pelicula_path(m)	/peliculas/1	DELETE /peliculas/:id	destroy

> Vista _

- La página HTML generada por el template Haml empleado incluye código HTML que no estaba definido en Haml.
  - CSS: `assets/application.css`
  - HTML común: `app/views/layouts/application.html.erb`

> Vista _

- Eliminar el archivo
  `app/views/layouts/application.html.erb`
- Crear el archivo
  `app/views/layouts/application.html.haml:`

```
!!! 5
%html
  %head
     %title Mi Aplicacion en Rails
     = stylesheet_link_tag 'application'
     = javascript_include_tag 'application'
     = csrf_meta_tags

  %body
     =yield
```

> Acción show _

- En el archivo
  `app/controllers/peliculas_controller.rb` agregar:

```
def show
    id = params[:id]
    @pelicula = Pelicula.find(id)
end
```

> Vista _

- En el archivo `app/views/peliculas/show.html.haml` agregar:

```
%h2 Detalles sobre #{@pelicula.titulo}

%ul#detalles
  %li
     Calificacion:
     = @pelicula.calificacion
  %li
     Estrenada el:
     = @pelicula.fecha_estreno.strftime("%B %d, %Y")

%h3 Descripcion:
%p#descripcion= @pelicula.descripcion

= link_to 'Regresar a la lista de peliculas',
peliculas_path
```

> CSS _

- Aplicar el estilo CSS en app/assets/stylesheets/application.css:

```
html, body {
        margin: 0;
        padding: 0;
        background: White;
        color: DarkSlateGrey;
        font-family: Tahoma, Verdana, sans-serif;
        font-size: 10pt;
}
```

> CSS _

```
div#main {
        margin: 0;
        padding: 0 20px 20px;
}

a {
        background: transparent;
        color: maroon;
        text-decoration: underline;
        font-weight: bold;
}
```

> CSS _

```
h1 {
        color: maroon;
        font-size: 150%;
        font-style: italic;
        display: block;
        width: 100%;
        border-bottom: 1px solid DarkSlateGrey;
}
```

> CSS _

```
h1.titulo {
        margin: 0 0 1em;
        padding: 10px;
        background-color: orange;
        color: white;
        font-size: 2em;
        font-style: normal;
        border-bottom: 4px solid gold;
}
```

> CSS _

```
tables#peliculas {
        margin: 10px;
        border-collapse: collapse;
        width: 100%;
        border-bottom: 2px solid black;
}
tables#peliculas  th{
        border: 2px solid white;
        font-weight: bold;
        background-color: wheat;
}
tables#peliculas  th, tables#peliculas  td{
        padding: 4px;
        text-align: left;
}
```

> CSS _

```
#notice, #warning {
        background: rosybrown;
        margin: 1em 0;
        padding: 4px;
}
}
form label{
        display: block;
        line-height: 25px;
        font-weight: bold;
        color: maroon;
}
```

> New/Create _

- Editar index.html.haml y agregar al final:

```
= link_to 'Agregar nueva
película', new_pelicula_path
```

- Cuando el usuario haga clic en este enlace, se producirá la acción **new**.
- Agregar un método **new** trivial al controlador (`peliculas_controller.rb`):

```
def new
end
```

> Vista _

- Para generar formularios, rails dispone de helpers: `form_tag`.
- `form_tag` requiere una ruta a la cual el formulario será enviado.
  - Ejemplo
    - URI: `peliculas_path`
    - Método: post

> Vista _

- Crear `app/views/peliculas/new.html.haml`:

```
%h2 Nueva Película!

=form_tag peliculas_path, :method =>:post do
  = label :pelicula, :titulo, 'Titulo'
  = text_field :pelicula, :titulo
  = label :pelicula, :calificacion: 'Calificacion'
  = select :pelicula, :calificacion, ['G', 'PG',
'PG-13', 'R', 'NC-17']

  = label :pelicula, :fecha_estreno 'Estrenada en'
  = date_select :pelicula, :fecha_estreno

  = submit_tag 'Guardar'
```

> Redirección y el "Flash"_

- Luego de crear la película, se podría enviar al usuario a una página informativa.
  - Ejemplo: create.html.haml
- Sin embargo, podría optarse por reenviar al usuario al index y presentar en el index un mensaje de éxito o fracaso.
- Para enviar al usuario a una página en particular rails dispone del método `redirect_to`
  - Causa que la acción de un controlador no termine en una vista, sino que reinicie un nuevo pedido a una acción diferente.

> Redirección y el "Flash"_

- En peliculas_controller.rb:

```
def create
     @pelicula = Pelicula.create!(params[:pelicula])
     redirect_to  peliculas_path
end
```

> Redirección y el "Flash"_

- `redirect_to` realiza una nueva solicitud HTTP, y debido a que HTTP no conserva el estado, todas las variables asociadas con la acción **create** se perderán.
  - `flash[]` es un método especial que persiste entre solicitudes.
  - Por convención: `flash[:notice]` se usa para mensajes de información y `flash[:warning]` para mensajes de error.

> Redirección y el "Flash"_

- En peliculas_controller.rb:

```
def create
     @pelicula = Pelicula.create!(params[:pelicula])
     flahs[:notice] = "#{@pelicula.titulo} se creo!"
     redirect_to  peliculas_path
end
```

> Redirección y el "Flash"_

- El template empleado para generar las vistas app/views/layouts/application.html.haml es un buen candidato para presentar los mensajes de flash.
- En dicho archivo agregar (entre `%body` y `=yield`):

```
- if flash[:notice]
  #notice.message= flash[:notice]
- elsif flash[:warning]
  #warning.message= flash[:warning]
```

> Edit/Update _

- Editar `app/views/peliculas/show.html.haml` y agregar al final:

```
= link_to 'Editar',
edit_pelicula_path(@pelicula)
= link_to 'Regresar'
peliculas_path
```

- Cuando el usuario haga clic en el primer enlace, se producirá la acción **edit**.

> Edit/Update _

	Create	Update
Parámetros pasados a la vista	Ninguno	Instancia de la clase (Pelicula)
Valores por defecto en el formulario	Ninguno	Atributos de la instancia
Etiqueta del botón	"Crear"/"Guardar"	"Actualizar"/"Guadar"
Acción del controlador	**new** genera el formulario y **create** recibe el formulario y modifica la base de datos	**edit** genera el formulario y **update** recibe el formulario y modifica la base de datos
params[]	Atributos para la nueva instancia (pelicula)	Atributos modificados de la instancia (pelicula)

> Edit/Update _

- Editar `peliculas.controller.rb` y agregar:

```
def edit
     @pelicula = Pelicula.find params[:id]
end

def update
     @pelicula = Pelicula.find params[:id]
     @pelicula.update_atributes!(params[:id])
     flash[:notice] = "#{@pelicula.titulo} fue
actualizada!!"
     redirect_to pelicula_path(@pelicula)
end
```

- La acción **edit** presentará información en el form, y la acción **update** actualizará los cambios realizados.

> Vista_

- Para la vista usar la vista de crear.
  - Cambiar las rutas y los paths necesarios.

> Vista_

- Cambios:
  - :method => :put
  - pelicula_path(@pelicula)

> Delete _

- Agregar en `peliculas.controller.rb`:

```
def destroy
    @pelicula = Pelicula.find params[:id]
    @pelicula.destroy
    flash[:notice] = "La pelicula
#{@pelicula.titulo} fue destruida!!"
    redirect_to peliculas_path
end
```

- En el archivo show.html.haml agregar:

```
= link_to 'Eliminar',
pelicula_path(@pelicula), :method =>
:delete
```

> Delete _

- En el archivo show.html.haml cambiar, el enlace, por un botón:

```
= button_to 'Eliminar',
pelicula_path(@pelicula),
:method => :delete, confirm =>
'Estas seguro?'
```