Migrating Legacy Rails Apps to Rails 3

A tutorial by Clinton R. Nixon for RailsConf 2011
@crnixon
crnixon@crnixon.com
http://pinboard.in/u:crnixon/t:rails3upgrade/

http://bit.ly/rails3up
Top 10 reasons to upgrade

10. The new ARel syntax is sweet.
9. You can use jQuery.
8. Bundler is fun!
7. Easier to hire new developers.
Top 10 reasons to upgrade

5. Way faster than Rails 2.*

4. ActiveModel lets you treat everything like ActiveRecord.

3. You don’t have to use ActiveRecord.

2. Cut your controllers down to size with respond_to.

1. You don’t have to get up for a 9 AM tutorial.
Getting ready with Rails 2.3
“How do I figure out which new Rails 3 features are worth upgrading to (above and beyond the minimum to attain compatibility)? Maybe none and I should just start using them as I write new code.”
Rails 2.3.11

Remember to run `rake rails:update`. Getting rid of all deprecations should get you ready for Rails 3.

Install `rails_xss` plugin and fix views.

And if you still have `.rhtml`, `.rxml`, and `.rjs` files, rename them.
Safe strings are copied into the output, unsafe strings are escaped

Strings are not safe by default

String interpolation always makes strings not safe

Built-in helpers emit safe strings

`.html_safe` marks strings as safe

`raw` copies strings verbatim

No need for `h` any more
module ApplicationHelper
  def clippy(text, bgcolor='FFFFFF')
    html = <<-EOF
      <object classid="clsid:d27cdb6e-ae6d-11cf-96b8"
          id="clippy">
          <param name="movie" value="/clippy.swf"/>
          <param NAME="FlashVars" value="text=#{text}"/>
          <param name="bgcolor" value="#{bgcolor}"/>
          <embed src="/clippy.swf"
            name="clippy"
            FlashVars="text=#{text}"
            bgcolor="#{bgcolor}"/>
      </object>
    EOF
  end
end
module ApplicationHelper
  def clippy(text, bgcolor='#FFFFFF')
    html = <<-EOF
      <object classid="clsid:d27cdb6e-ae6d-11cf-96b8"
          id="clippy" >
          <param name="movie" value="/clippy.swf"/>
          <param NAME="FlashVars" value="text=#{text}"/>
          <param name="bgcolor" value="#{bgcolor}"/>
          <embed src="/clippy.swf" name="clippy"
            FlashVars="text=#{text}" bgcolor="#{bgcolor}"/>
      </object>
    EOF
    html.html_safe
  end
end

Safe
Localization

If you have HTML in your YAML locale files (config/locales/), make sure all keys pointing to HTML end in _html. Rails 3 will auto-escape these values otherwise.
fake_arel

Hat tip to Grant Ammons

https://github.com/gammons/fake_arel

Uses named scopes and a few patches to give you the new ARel syntax

Not really “fake”: actually lazily loaded
Reply.where(:id => 1)
Reply.select("content, id").where("id > 1").order("id desc").limit(1)
Topic.joins(:replies).limit(1)

class Reply < ActiveRecord::Base
  named_scope :by_john, where(:name => "John")
  named_scope :recent, lambda { |t|
    where("created_at > ? ", t.minutes.ago) }
  named_scope :recent_by_john, recent(15).by_john
end
RESTful Controllers

Now is a great time to refactor your controllers to be RESTful.

The new routes syntax in Rails 3 works great with RESTful controllers.
Other improvements

Use YAJL for JSON processing if installed: `gem install yajl-ruby`

Flash: `alert` and `notice` promoted

Object#presence: returns the object if it’s #present? otherwise returns nil.
flash[:notice] = 'Post was created'
redirect_to(@post)

# becomes:
redirect_to(@post, :notice => 'Post was created')

class ActionController::Base
  # Convenience accessor for flash[:alert]
  def alert
    flash[:alert]
  end

  # Convenience accessor for flash[:alert]=
  def alert=(message)
    flash[:alert] = message
  end
end
def display_name
  if name.present?
    name
  elsif login.present?
    login
  else
    "Anonymous user"
  end
end

# OR
def display_name
  name.presence || login
end

# becomes:
def display_name
  name.present? ? name : (login.present? ? login : "Anonymous user")
end
Other improvements

Object#returning is removed in Rails 3. Change your code to use Object#tap.

Putting non-ActiveRecord fixtures in test/fixtures/ will break in Rails 3. Move those out of there.
returning([]) do |guitarists|
guitarists << "Eddie Van Halen"
guitarists << "Buzz Osborne"
guitarists << "Vernon Reid"
end

[].tap do |guitarists|
guitarists << "Eddie Van Halen"
guitarists << "Buzz Osborne"
guitarists << "Vernon Reid"
end

# => ["Eddie Van Halen", "Buzz Osborne", "Vernon Reid"]
Another cool trick with .tap

```ruby
@guitars = Musicians.where(instrument: 'guitar')
  .map { |dude| dude.instrument_name }
  .uniq
```

Me debugging this normally

```ruby
musicians = Musicians.where(instrument: 'guitar')
p musicians
all_names = musicians.map { |dude| dude.instrument_name }
p all_names
@guitars = all_names.uniq
```

The power of .tap

```ruby
@guitars = Musicians.where(instrument: 'guitar')
  .tap { |ms| p ms.all }
  .map { |dude| dude.instrument_name }
  .tap { |names| p names }
  .uniq
```
Working with Bundler
Bundler: not that scary

Bundler was a little shaky at first, but is rock-solid now.

Worth doing before an upgrade to Rails 3.

Auto-generate your initial Gemfile using the rails_upgrade plugin and rake rails:upgrade:gems
source 'http://rubygems.org'
gem 'rails', '2.3.11'
gem 'pg'
gem 'domainatrix'

group :development, :test do
  gem 'rspec-rails', '< 2.0'
end

group :development do
  gem 'query_reviewer',
    :git => 'git://github.com/nesquena/query_reviewer.git'
end

group :test do
  gem 'capybara'
end
Using Bundler effectively

Install gems locally instead of into system location:

```
bundle install --path vendor
```

Package gems:

```
bundle package
```

Check your Gemfile.lock into version control
alias be='bundle exec'
alias bi='bundle install --path vendor'
alias bp='bundle package'
alias bu='bundle update'
alias binit='bi && bp &&
    echo "vendor/ruby" >> .gitignore'
Plugins & gems

Many popular plugins are now released in gem form.

Replace plugins with gems so that Bundler can manage them and upgrading is easier.

If you’ve edited code in vendor/plugins, put your version into VCS and pull in Gemfile.
Using the rails_upgrade plugin
rails_upgrade

Written by Jeremy McAnally, maintained by Rails Core

script/plugin install git://github.com/rails/rails_upgrade.git

or

rails plugin install git://github.com/rails/rails_upgrade.git
rake rails:upgrade:check

Generates a report on what you’ll need to update.

Save this report for checking later.
> rake rails:upgrade:check

**Old router API**
The router API has totally changed.
More information: http://yehudakatz.com/2009/12/26/the-rails-3-router-rack-it-up/

The culprits:
- config/routes.rb

**Deprecated constant(s)**
Constants like RAILS_ENV, RAILS_ROOT, and RAILS_DEFAULT_LOGGER are now deprecated.
More information: http://litanyagainstfear.com/blog/2010/02/03/the-rails-module/

The culprits:
- lib/tasks/rspec.rake
- app/models/link.rb
rake rails:upgrade:backup

Copies all files likely to be overwritten and adds a .rails2 extension.

Not that necessary, given the use of a good VCS.
> rake rails:upgrade:backup

* backing up .gitignore to .gitignore.rails2
* backing up app/controllers/application_controller.rb to app/controllers/application_controller.rb.rails2
* backing up app/helpers/application_helper.rb to app/helpers/application_helper.rb.rails2
* backing up config/routes.rb to config/routes.rb.rails2
* backing up config/environment.rb to config/environment.rb.rails2
* backing up config/environments/development.rb to config/environments/development.rb.rails2
* backing up config/environments/production.rb to config/environments/production.rb.rails2
* backing up config/database.yml to config/database.yml.rails2
* backing up doc/README_FOR_APP to doc/README_FOR_APP.rails2
* backing up test/test_helper.rb to test/test_helper.rb.rails2

This is a list of the files analyzed and backed up (if they existed); you will probably not want the generator to replace them since you probably modified them (but now they're safe if you accidentally do!).
rake rails:upgrade:gems

Checks config/environment.rb for config.gem lines and then spits out a Gemfile.

Doesn’t check config/environments/*.rb files.
# From config/environment.rb
config.gem 'domainatrix'
config.gem 'pg'

# From config/environments/
config.gem 'rspec', :version => '1.3.2',
  :lib => 'spec'
config.gem 'rspec-rails', :version => '1.3.4',
  :lib => 'spec/rails'

# Results from rake rails:upgrade:gems
source 'http://rubygems.org'

gem 'rails', '3.0.6'
gem 'domainatrix'
gem 'pg'
rake rails:upgrade:routes

Automatically converts all old-style routes to Rails 3-style routes.

Old syntax deprecated, but works for now in Rails 3.

Not perfect.
# config/routes.rb

ActionController::Routing::Routes.draw do |map|
  map.connect '', :controller => 'links'

  map.connect 'rubyurl/remote', :controller => 'links', :action => 'create'

  map.connect 'about', :controller => 'links', :action => 'about'
  map.connect 'api', :controller => 'links', :action => 'api'
  map.connect 'report-abuse', :controller => 'links', :action => 'report'
  map.connect 'home', :controller => 'links', :action => 'home'

  map.resources :links, :name_prefix => 'api_', :path_prefix => 'api', :controller => 'api/links'

  # Install the default route as the lowest priority.
  map.connect ':controller/:action/:id'

  map.connect ':token', :controller => 'links', :action => 'redirect'
end
# Output from rake rails:upgrade:routes

Rubyurl::Application.routes.draw do
  match '' => 'links#index'
  match 'rubyurl/remote' => 'links#create'
  match 'about' => 'links#about'
  match 'api' => 'links#api'
  match 'report-abuse' => 'links#report'
  match 'home' => 'links#home'
  resources :links
    match '/:controller(/:action(/:id))'
    match ':token' => 'links#redirect'
end
Generates a new application.rb from config/environment.rb.

Ok, but not overly useful.
Taking the Rails 3 plunge
Generate a new Rails app

Run "rails new . -d <database>".
Will prompt you about overwriting files.
This is where a good VCS comes in handy.
rails new . -d sqlite3

**conflict**   README
Overwrite README? (enter "h" for help) [Ynaqdh] y

**force**   README

**conflict**   Rakefile
Overwrite Rakefile? (enter "h" for help) [Ynaqdh] y

**force**   Rakefile

**create**   config.ru

**conflict**   .gitignore
Overwrite .gitignore? (enter "h" for help) [Ynaqdh] y

**force**   .gitignore

**create**   Gemfile

**exist**   app

**conflict**   app/controllers/application_controller.rb
Overwrite app/controllers/application_controller.rb? (enter "h" for help) [Ynaqdh] y
> git add --patch
diff --git a/app/controllers/application_controller.rb b/app/controllers/application_controller.rb
index dd5e2ff..e8065d9 100644
--- a/app/controllers/application_controller.rb
+++ b/app/controllers/application_controller.rb
@@ -1,11 +1,3 @@
 class ApplicationController < ActionController::Base
-   before_filter :calculate_links
-   
-   # refactor to some sort of cache tool
-   def calculate_links
-      @cached_data = {:link_count => Link.count}
-   end
+   protect_from_forgery
 end
Stage this hunk [y,n,q,a,d,/,s,e,?]?
git add --patch

Prompts you interactively about each change.

Choose ‘s’ for split to break up changes.

For finer control, use “git add --edit”.
Immediate fixes

Remove preinitializer.rb if you were using Bundler before.

lib/ directory not autoloaded by default:
config.load_paths += %W(#{config.root}/lib)
Rails 3 has moved to unobtrusive JS.

To make everything work, call `javascript_include_tag :defaults` in view.

Add `csrf_meta_tag` inside `<head>`.
Removed in Rails 3:
button_to_remote
submit_to_remote
observe_field
observe_form
periodically_call_remote

Replaced:
link_to_remote
form_remote_tag
form_remote_for
remote_form_for

To get these back:
gem 'prototype_legacy_helper', :git => 'git://github.com/rails/prototype Legacy_helper.git'
# Rails 2
link_to_remote "Pizza", pies_path

# Rails 3
link_to "Pizza", pies_path, :remote => true

# Rails 2
form_remote_tag "/contact"

# Rails 3
form_tag "/contact", :remote => true

# Rails 2
remote_form_for @tuba

# Rails 3
form_for @tuba, :remote => true
More form fixes

Gone:
error_messages_for :model
form.error_messages
input('model', 'field')
form('model')

Install `dynamic_form` gem to get them back.

Or, use `simple_form` and be happy.
RAILS_*

* constants are deprecated in favor of a Rails module.

RAILS_ENV  ->  Rails.env
RAILS_ROOT  ->  Rails.root

There's a lot of cool things you can do with the new Rails module.
ActiveRecord

named_scope is now scope.

ActiveRecord::Errors now has an array-like syntax instead of .on.

Whole new query syntax.
class Visit < ActiveRecord::Base
  belongs_to :link

  # Rails 2
  named_scope :spam,
              :conditions => ['flagged = 'spam''],

  # Rails 3
  scope :spam, where(flagged: 'spam')
end
View deprecations

All helpers that emit text, even ones that take blocks use `<%=` now.
<%= form_for @link, :url => links_url do |f| %>
  <label for="link_website_url">
    Create a RubyURL
  </label>
  <%= f.text_field :website_url %>
  <%= submit_tag 'Go!', :class => 'go' -%>
<% end %>
Mailers

Mailers have a completely new syntax and a new place to live: app/mailers/.

The old syntax works, but is deprecated.

Mailers now based off ActionController, which lets you do some neat stuff.
# Rails 2

class UserMailer < ActionMailer::Base
  def password_reset(user)
    recipients = user.email
    from = "info@spkrtown.com"
    subject = "Password Reset Instructions"
    body = :user => user
  end
end

# Rails 3

class UserMailer < ActionMailer::Base
  default :from => "info@spkrtown.com"

  def password_reset(user)
    @user = user
    mail(:to => user.email,
         :subject => "Password Reset Instructions")
  end
end
# Multipart emails

class UserMailer < ActionMailer::Base
  default :from => "info@spkrtown.com"

  def password_reset(user)
    @user = user
    attachments['sorry.pdf'] = File.read('/path/sorry.pdf')

    mail(:to => user.email,
         :subject => "Password Reset Instructions") do |format|
      format.html { render 'password_reset' }
      format.text { render 'password_reset' }
    end
  end
end
# Sending emails

UserMailer.password_reset(@user).deliver
Fixing & replacing dependencies
Instead of

```ruby
require 'spec'
require 'spec/rails'
```

use

```ruby
require 'rspec'
require 'rspec/rails'
```

You also need to upgrade to rspec 2.
Old configuration:

```ruby
Spec::Runner.configure do |config|
  # ...
end
```

New configuration:

```ruby
RSpec.configure do |config|
  # ...
end
```
RSpec

Two other changes to make:

Make sure rspec-rails is in the development and test groups in your Gemfile.

Cucumber no longer uses its own environment - uses test instead.
If you use view matchers in RSpec:

```ruby
it "should display Here you go in a h1 tag" do
  rendered.should have_tag('dt', :text => 'Here you go')
end
```

then you need to install `rspec2-rails-views-matchers`.
factory_girl

You need to use the
factory_girl_rails gem.
There is a rails3 branch of will_paginate on GitHub.

A good replacement: kaminari.
Searchlogic

Searchlogic is busted with Rails 3.

But you don’t need it!

New ARel syntax gets you part of the way.

MetaSearch is a gem that can do everything Searchlogic can and has similar syntax.
# Searchlogic

def index
    @search = Goat.search(params[:search])
    @search.order ||= :ascend_by_name
    @goats = @search.all.paginate :page => params[:page]
end

<%= order @search, :by => 'name', :as => 'name'.humanize %>

# MetaSearch

def index
    @search = Goat.search(params[:search])
    @search.meta_sort ||= 'name.asc'
    @goats = @search.all.paginate :page => params[:page]
end

<%= sort_link @search, 'name', 'name'.humanize %>
/ Same syntax for Searchlogic and MetaSearch

= form_for @search do |f|
  = f.label :name_like, 'Name'
  = f.text_field :title_like, :size => 15
  = f.submit "Search"
  = "#{@search.count} matches"
Authlogic

Authlogic 3 is compatible with Rails 3.
Many people recommend Devise.
Check it out.
AttachmentFu

Bad news: doesn’t work easily. There are some instructions out there you can try.

Paperclip is one option.

CarrierWave is a more radical option.
Heavy metal cat plays air guitar.