Efficient Rails

Workflow Upgrades for Crafting Rails Apps with Superhuman Speed

version 1.0

Andrew Allen

Efficient Rails

Workflow Upgrades for Crafting Rails Apps with Superhuman Speed

Andrew Allen

This book is for sale at http://leanpub.com/efficient-rails

This version was published on 2016-06-22

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Andrew Allen by spreading the word about this book on Twitter!

The suggested hashtag for this book is #EfficientRails.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#EfficientRails

Contents

Introduction

This is not your average Rails book.

What it's not

- An introduction. I'll assume that you're familiar with at least the core concepts of both Ruby and Rails.
- A tutorial. At least not in the traditional sense.
- Academic. We won't be discussing much abstract programming theory.

What it is

- Practical. Every part of this book is focused on saving you time and making Rails development more enjoyable.
- Actionable. Learn and implement each solution in just a few minutes.
- Non-linear. You don't have to read cover-to-cover to get something out of this book.

Who it's for

This book is intended for someone who has at least some experience with Rails. There are tons of great resources out there to learn the basics, but once you hit a base level of proficiency, your options start to thin. This book aims to fill that space.

Efficient Rails is for someone who uses Rails frequently. It doesn't matter if you're junior or senior, the techniques in this book will save you time on common tasks. If you're using Rails every day, you'll get some serious productivity gains.

As your productivity rises, you'll level up as a Rails developer, enabling you to take on more clients, impress your team by getting that product out the door faster, or finally shed that 'Junior' part of your title.

How to use it

The book is divided into 3 parts.

The first part covers the tools you're using every day and how to get the most out of them. This includes the Terminal, Git and the Rails Console.

The second part covers the core aspects of the Rails framework and contains recipes for accomplishing common tasks.

The final part covers techniques that will speed up your testing and debugging workflows.

Pick a chapter that relates to something youâ€™re working on currently and open up to it. Within that chapter are between 10-12 sections in no particular order. Each section presents a problem and one or more solutions. Read the problem. If it describes a pain point you have or have had, then read the solution.

Book updates

As developers, you know nothing stays the same for long. New gems will be released, better ways of accomplishing repetitive tasks will be discovered. As the world of Rails changes, so too will this book, and you'll be the first to know. These updates will be sent to you automatically as soon as they're available.

I'll also be incorporating feedback into future revisions. To submit feedback on a particular section, click one of the links at the end of the section. You'll have a chance to provide more feedback after clicking. Of course, you can also send an email to andrew@EfficientRails.com[1].

With that said, there will be typos. There will be incomplete thoughts and run-on sentences. There will be more chapters and sections to come. But I'm confident you'll get a ton of value out of what's here.

> If you are not embarrassed by the first version of your product, you've launched too late.
>
> – Reid Hoffman

[1] mailto:andrew@efficientrails.com

Chapter 3: Rails Console

The Rails console is a powerful tool that comes out of the box with Rails. It's great for trying out Ruby code, instantiating models and even querying and exploring the database.

I have a Rails console open all day long and spend a non-trivial chunk of time using it each day.

In this chapter, I'll cover a handful of workflow upgrades that will make time spent working in the Rails console that much more enjoyable.

Setting up the Console

Problem

Rails comes with a pretty lackluster console by default. Sure, it's a handy tool, but it's missing a number of features that would make it much more powerful.

```
irb(main):001:0> Post.first
  Post Load (0.2ms)  SELECT  "posts".* FROM "posts"  ORDER BY "posts"."id" ASC LIMIT 1
=> #<Post id: 1, title: "Bushwick Migas Cleanse Xoxo Tilde Tumblr", author: "Olga Veum", body: "Iphone pinterest polaroid twee. Gluten-free fixie ...", tags: ["umami", "chicharro
nes"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">
irb(main):002:0>
```

The default Rails Console

Here are some things the Rails console is sorely missing:

1. Syntax highlighting
2. Pretty-printed output
3. A decent REPL (Pry > IRB)

Inspecting an object or collection spits out a chunk of monochromatic goop:

```
irb(main):001:0> Post.first
  Post Load (0.2ms)  SELECT  "posts".* FROM "posts"  ORDER BY "posts"."id" ASC LIMIT 1
=> #<Post id: 1, title: "Bushwick Migas Cleanse Xoxo Tilde Tumblr", author: "Olga Veum", body: "Iphone pinterest polaroid twee. Gluten-free fixie ...", tags: ["umami", "chicharro
nes"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">
irb(main):002:0> Post.all
  Post Load (5.3ms)  SELECT "posts".* FROM "posts"
=> #<ActiveRecord::Relation [#<Post id: 1, title: "Bushwick Migas Cleanse Xoxo Tilde Tumblr", author: "Olga Veum", body: "Iphone pinterest polaroid twee. Gluten-free fixie ...",
tags: ["umami", "chicharrones"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">, #<Post id: 2, title: "Stumptown Cronut I Phone Venmo", author: "Glenna Sha
nahan", body: "Bicycle rights meditation art party tattooed aesth...", tags: ["banjo"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">, #<Post id: 3, title
: "Kickstarter Tofu Synth Godard", author: "Nat Jast Jr.", body: "Tote bag tousled meh crucifix blue bottle. Distill...", tags: ["XOXO"], created_at: "2015-12-10 16:28:27", updat
ed_at: "2015-12-10 16:28:27">, #<Post id: 4, title: "Ethical Neutra Listicle Cray Pour Over Hammock Has...", author: "Mrs. Domingo Hickle", body: "Wayfarers direct trade church-k
ey dreamcatcher. Wa...", tags: ["flannel"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">, #<Post id: 5, title: "Leggings Echo Everyday Intelligentsia", a
uthor: "Prudence Jacobi", body: "Diy gastropub deep v typewriter quinoa squid green...", tags: ["marfa"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">, #
<Post id: 6, title: "Whatever Typewriter Mixtape Occupy", author: "Selmer Metz", body: "Five dollar toast chicharrones shabby chic hashtag...", tags: ["venmo"], created_at: "2015
-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">, #<Post id: 7, title: "Butcher Bespoke Try Hard Skateboard Meh Hashtag Dr...", author: "Luisa Yundt", body: "Hoodie cred mixt
ape taxidermy post-ironic beard xo...", tags: ["franzen", "gentrify", "pabst"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">, #<Post id: 8, title: "Ugh G
odard Pitchfork Plaid Williamsburg Typewriter...", author: "Agustin Mitchell", body: "Carry hoodie williamsburg. Yuccie roof fashion axe...", tags: ["loko", "pinterest", "humbleb
rag"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">, #<Post id: 9, title: "Celiac Aesthetic Fingerstache Diy Plaid Thundercat...", author: "Maximus Walte
r", body: "Flannel meh deep v banjo normcore swag yolo polaro...", tags: ["dreamcatcher", "gluten-free", "keytar"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:
28:27">, #<Post id: 10, title: "Selvage Tacos Intelligentsia Skateboard Biodiesel", author: "Iva Feeney Sr.", body: "Wayfarers hoodie retro actually tattooed messenger...", tags:
 ["vinegar"], created_at: "2015-12-10 16:28:27", updated_at: "2015-12-10 16:28:27">, ...]>
irb(main):003:0>
```

Inspecting a collection with the default Rails Console

I dare you to figure out what's going on in that output.

Solution: Jazz Fingers

Jazz Fingers aptly describes itself as "an opinionated set of console-related gems and a bit of glue". By installing Jazz Fingers, you get the following gems installed with sensible defaults:

- Pry for a powerful shell alternative to IRB.
- Awesome Print for stylish pretty print.
- Hirb for tabular collection output.

- Pry Doc to browse Ruby source, including C, directly from the console.
- Pry Git to teach the console about git. Diffs, blames, and commits on methods and classes, not just files.
- Pry Remote to connect remotely to a Pry console.
- Pry Coolline for syntax highlighting as you type.

We'll cover the details of some of these gems in further sections, but at the very least, Jazz Fingers gets you a Pry-based console by default, some nice syntax highlighting and pretty-printed objects.

Here's the result:

The Rails Console with Jazz Fingers

Install

To install, simply add the following to your Gemfile:

```
group :development, :test do
  gem 'jazz_fingers'
end
```

Now, the next time you boot up your console, you'll notice a much more stylish and practical interface.

Save Frequently Used Commands

Problem

You probably notice yourself reusing the same commands over and over when you're in the Rails console.

For example, I often find my test user in the local database by typing:

```
me = User.find_by(email: 'andrew@example.com')
```

Or maybe you have some really long model or class names that are unwieldy to type over and over.

And how many times have you typed something this just to have a hash to play with:

```
my_hash = {a: 'b', c: 'd'}
```

How can we save ourselves a few keystrokes in the Rails console?

Solution

Both Pry and IRB allow us to define config files local to a project. Within these config files, we can define our own convenience methods to make our lives a little bit better.

Within the root of our project, we'll add a `.pryrc` file (or `.irbrc` file if not using Pry, the syntax is the same).

your_app/.pryrc

```
class Object
  def me
    User.find_by(email: 'andrew@example.com')
  end

  def slmn
    SomeLongModelName
  end

  def an_array
    (1..5).to_a
  end

  def a_hash
```

```
15      {a: 'b', c: 'd'}
16    end
17  end
```

Now, every time we boot up a console, we'll have access to these methods. That means we can call methods on our test user by just typing `me`. We also have an alias to make tying out long class names a thing of the past. We can simply type `slmn.where(...)` instead of `SomeLongModelName.where(...)`.

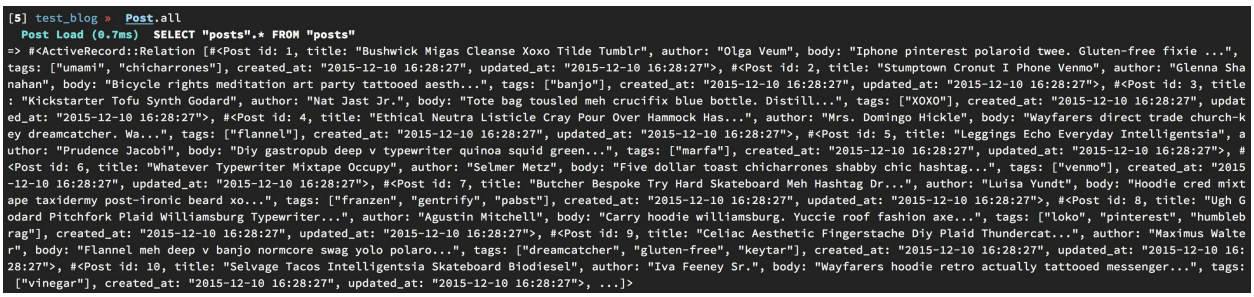
And when we need a dummy array or hash to try something on, we can just type `a_hash` or `an_array`.

What convenience methods will you define?

Inspecting ActiveRecord Collections

Problem

When trying to view large collections of ActiveRecord objects, the Rails console prints a tangled, truncated mess:



Viewing a collection of ActiveRecord objects

This is far from useful.

Solution

Jazz Fingers (see Setting up the Console) bundles together some nice gems to improve our Rails console experience. One of those is Hirb, a gem that enables tabular output when viewing ActiveRecord collections in the console.

However, Hirb is not enabled by default. You can toggle Hirb manually when you need it by running the following commands in the console:

```
1 Hirb.enable  #=> enables tabular output
2 Hirb.disable #=> disables tabular output
```

After enabling Hirb, we can browse our ActiveRecord collections with tables:

id	title	author	body	tags	created_at	updated_at
1	Bushwick Migas Cleanse Xo...	Olga Veum	Iphone pinterest polaroid...	umami, chicharrones	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
2	Stumptown Cronut I Phone ...	Glenna Shanahan	Bicycle rights meditation...	banjo	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
3	Kickstarter Tofu Synth Go...	Nat Jast Jr.	Tote bag tousled meh cruc...	XOXO	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
4	Ethical Neutra Listicle C...	Mrs. Domingo Hickle	Wayfarers direct trade ch...	flannel	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
5	Leggings Echo Everyday In...	Prudence Jacobi	Diy gastropub deep v type...	marfa	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
6	Whatever Typewriter Mixta...	Selmer Metz	Five dollar toast chichar...	venmo	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
7	Butcher Bespoke Try Hard ...	Luisa Yundt	Hoodie cred mixtape taxid...	franzen, gentrify, pabst	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
8	Ugh Godard Pitchfork Plai...	Agustin Mitchell	Carry hoodie williamsburg...	loko, pinterest, humblebrag	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
9	Celiac Aesthetic Fingerst...	Maximus Walter	Flannel meh deep v banjo ...	dreamcatcher, gluten-free...	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
10	Selvage Tacos Intelligent...	Iva Feeney Sr.	Wayfarers hoodie retro ac...	vinegar	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
11	Pitchfork Cornhole Diy Ca...	Eliza Toy	Vice wayfarers everyday b...	migas	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
12	Schlitz Kogi I Phone Quin...	Braeden Upton	Bitters chambray fap salv...	typewriter, heirloom	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
13	Bespoke Microdosing Chia ...	Zula Reichert V	Lomo fanny pack waistcoat...	leggings	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
14	Viral 8 Bit Tousled Pbr&B...	Jennie Satterfield	Bushwick pug farm-to-tabl...	tumblr, mlkshk	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
15	Hella Organic Heirloom Bi...	Miss Aniyah Padberg	Aesthetic viral tattooed ...	fixie, quinoa	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
16	Drinking Semiotics Wayfar...	Charlotte Wolf Jr.	Authentic narwhal bushwic...	literally, Yuccie, locavore	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
17	Yuccie Freegan Celiac Meg...	Pearlie Cormier	Banjo authentic everyday ...	mixtape, ethical	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
18	Everyday Vhs Kombucha Lomo	Allie Fahey	Yr synth kitsch banjo. Bi...	echo	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
19	Etsy Aesthetic Tacos Mast...	Tatum Erdman	Freegan occupy craft beer...	hoodie	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
20	Fingerstache Pop Up Ethic...	Eliezer Balistreri	Portland typewriter vinyl...	meggings, chia	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC
21	Taxidermy Vice Cornhole C...	Tillman Lind	Synth distillery tumblr k...	bitters	2015-12-10 16:28:27 UTC	2015-12-10 16:28:27 UTC

Hirb displays output in a tabular format

Re-Execute Commands

Problem

Working with the Rails Console (or any console for that matter) is an exercise in trial and error. To get a desired outcome, we'll try a few commands, tweak them, then go back and try some more. This usually means executing the same set of commands a few times over.

To save time, most developers are familiar with using the up arrow to walk back through the command history. This is a great feature of most modern command line interfaces, but what if you've run a large number of commands between the target command and now? You'll have to page up through all of those intermediate commands in order to get to the one you want.

Solution: Reverse Search

Most modern command line interface have a lesser-known but highly-useful method of accessing the command history called reverse search, and the Rails console is no different.

Pressing Ctrl + r in the Rails console triggers reverse search mode. From this mode, we can start typing any part of the command we're looking for, and the most recent matching command will be shown. From here, we have a few options:

- Press Ctrl + r again to see the next most recent matching command
- Press Return to execute the matching command
- Press Ctrl + g to exit reverse search
- Press Esc to exit reverse search, but keep the matching command for editing

```
[8] test_blog »
(search:Post): Post.last(5)
```

Reverse search in the Rails Console

Change Rails Environments

Problem

By default, the Rails Console starts in the development environment. If a RAILS_ENV environment variable is set, like in production, it will use that environment.

But what if we want to specify a different environment?

Solution #1: Start up Rails Console in a Different Environment

If we want the Rails console to start up in a different environment, we can specify one explicitly. The easiest way is to add the environment as an argument to the `rails console` command.

```
$ bundle exec rails console test
```

Additionally, you can override the `RAILS_ENV` environment variables.

```
$ RAILS_ENV=test bundle exec rails console
```

Solution #2: Hot Swap Database Connections

What if you're got a console session running in the development environment but you want to switch to the test database? This usually happens to me when I want to debug a FactoryGirl factory but I don't want to litter my development database with stray objects.

Sure, we could stop the current session and restart in the target environment, or even open another session in a new tab. But there's another way. We can define a helper method in our `~/.pryrc` (or `~/.irbrc`) that can hot swap database connections.

~/.pryrc

```
class Object
  def switch_db(env)
    config = Rails.configuration.database_configuration
    raise ArgumentError, 'Invalid Environment' unless config[env].present?

    ActiveRecord::Base.establish_connection(config[env])
    Logger.new(STDOUT).info("Successfully changed to #{env} environment")
  end
end
```

Now, we can simply enter `switch_db 'test'` to change over to the test database.

Note that this method will only change the database connection, not the Rails environment.

Ignore Sluggish Output

Problem

When you perform an ActiveRecord query in the Rails console, it prints out the result of the query, which can take several seconds if the query contains more than a few records. But let's say you're setting up a query and saving it to a variable for additional querying.

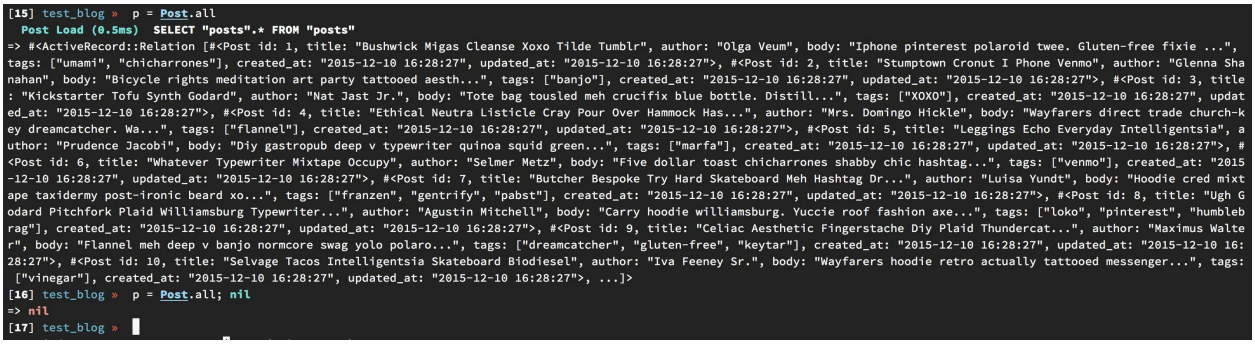
ActiveRecord is lazy by default, meaning you can build up a query by chaining additional methods and the query doesn't actually get run until the results are needed. But since the Rails console executes every line and renders its output, ActiveRecord queries lose their lazy properties.

If you're saving the query to a variable to run additional queries on, you don't actually need to see every record printed out. In fact, we'd prefer that the query be lazy and not execute until it's been narrowed down.

Solution

We can take advantage of a bit of Ruby trickery to keep ActiveRecord queries lazy and prevent them from spending time rendering output we don't care about.

By simply appending `; nil` to the end of the query, we split the line in two, letting the first one execute lazily and returning `nil` from the second. The Rails console only sees the `nil` return value and does not execute the query to display it. Now, that lazy query is saved to a variable and we can continue operating on it without the distraction of seeing the output.



Comparing the result with and without '; nil'

Don’t wear out the Backspace Key

Problem

The Rails console is a command line application, which means when you’re in it, you aren’t able to edit your input in the ways you’re accustomed to. For example, let’s say you have a command at the prompt, but you want to delete the whole thing.

If you’re used to Vim, you would press `dd` to delete the line. If you’re used to Mac shortcuts, you would press `Cmd+Backspace` to delete from your cursor to the beginning of the line. Or if you’re a mouse user (shame! just kidding), you might highlight the line and press backspace.

Well, none of this works in the command line (at least not without some serious customization).

Instead, you’d end up pressing backspace over and over, or holding it down until the line has been scrubbed out on character at a time. The same goes for moving around the line of text, like if you wanted to prepend some text (maybe a variable assignment) to the beginning of the line. `I` doesn’t work for Vim users, `Cmd+Left` doesn’t work for Mac users, and clicking still doesn’t work. So we’re left with holding left until the arrow key falls off.

Solution

Luckily, the Rails console comes with a few keyboard shortcuts designed to alleviate the most painful of these navigational concerns:

Rails Console Text Navigation Keyboard Shortcuts

Command	Result
Ctrl+a	Jump to the beginning of the line
Ctrl+e	Jump to the end of the line
Ctrl+u	Delete from the cursor to the beginning of the line
Esc+f	Move forward one word
Esc+b	Move backward one word
Esc+d	Delete the next word
Esc+Backspace	Delete the previous word
Ctrl+w	Delete the previous word (to the previous space)

Reloading Models After Changing Them

Problem

Given that I have a Rails console open all day long, I tend to run into the issue where my underlying models change after the Rails environment has already been booted up in the console. This can happen naturally as I make changes to a model, or from checking out a different branch.

Because the Rails console has already instantiated all classes in the Rails application, it will fail to acknowledge the changes to the models. This usually means I'm heading for a restart (did you try turning it off and on again?).

Solution

The Rails console comes with a convenience method, `reload!`, which will reload any objects defined, including ActiveRecord models, picking up any changes that have been made to the classes since they were first instantiated. This saves us from needing to exit and restart the console, a non-trivial operation for a large Rails application.

Note that if you have any objects saved to a variable, you'll need to either call reload on them, or reinstantiate them.

If you find yourself needing to use `reload!` frequently, it could be a sign that you're doing too much testing in the console. In this case, it's better to use tools like RSpec which are designed to handle creating and reloading objects before each test. As a bonus, you are also documenting your test cases, preventing future regression. See the chapter on Testing to learn how to streamline your testing workflow.

Sandbox

Problem

The Rails console is a great way to try out commands as you're writing code, but you maybe you want to do so without disturbing the state of your development environment. Or maybe you want to boot up the Rails console in production to try and reproduce a bug without affecting real data.

Solution

The Rails console comes with a handy sandbox mode that makes it possible to try out new ideas without worrying about trashing your development database or disturbing the state of production data. It does this by wrapping the entire session in a gigantic database transaction that gets rolled back when the session is over. This way, no changes that you make during the sandbox session persist after the session is closed.

To start Rails console in sandbox mode, simply run:

```
1 $ bundle exec rails console --sandbox
```

Forgot to Save That

Problem

You write out a long query in the console to fetch a specific object, but you forget to assign the result to a variable. Now, you have to go back and edit the original command, sticking a `my_variable =` at the beginning of the line. And just to add insult to injury, if the database query was particularly expensive, it will need to execute again.

Command history and some handy keyboard shortcuts make this somewhat painless, but it still takes a few seconds.

Solution

This happens so often that there's a special variable built into the Rails console for just this purpose. _ refers to the result of the last command run. As an added bonus, the result is stored in memory, meaning that the database query doesn't need to be re-run.

You can reassign the contents of _ to a new variable and carry on like nothing even happened.

```
1  > my_variable = _
```

Call Private & Protected Methods

Problem

It's considered a best practice of Object-Oriented design to expose only the public interface of a class and encapsulate implementation-specific logic in private or protected methods.

Private methods reduce the surface area of a class, limiting the number of ways other classes can interact with it. This helps mitigate complexity in an application by discouraging coupling between objects.

A private method in Ruby

```
1  class Post < ActiveRecord::Base
2    # ...
3
4    private
5
6    def a_private_method
7      "Hey! Get outta here! This is private business!"
8    end
9  end
```

But what about when you're debugging in the console and to really understand what's going on, you need to see the response of a private or protected method? If you try calling the method in question directly, you'll get an (appropriate) error.

```
[4] test_blog »  Post.new.a_private_method
NoMethodError: private method `a_private_method' called for #<Post:0x007fcf3cc40f00>
from /Users/andrewallen/.rbenv/versions/2.2.0/lib/ruby/gems/2.2.0/gems/activemodel-4.2.1/lib/active_
model/attribute_methods.rb:430:in `method_missing'
```

Calling private methods results in an error

Do you comment out the private declaration and restart your test? That's a pain. And maybe you are running the console in production because that's the only place you can reproduce the issue and you can't modify the code.

How do you break the rules in this case and execute that private method?

Solution

As is so happens, there's a 'backdoor' way to call private or protected methods on Ruby objects. The `send` method effectively bypasses interface declarations, letting us call any method on a class, not just its public methods.

```
[2] test_blog »  Post.new.send(:a_private_method)
=> "Hey! Get outta here! This is private business!"
```

Send bypasses private and protected designations

While it might raise some eyebrows to learn that methods are never truly private in Ruby, this trick can be a huge time-saver when inspecting code in the console.

Pasting Multi-line Code into the Console

Problem

Ruby lets us chain methods together, and while you should be careful not to break the Law of Demeter doing so, ActiveRecord is built on method chaining. When building complex queries, it's common to have half a dozen or more methods chained together, causing your code to quickly exceed the 80 character screen width limit. In cases like this, Ruby style guides recommend putting each method call on its own line. This provides readability and also allows the order of method calls to be changed by simply swapping lines.

```
1  Movie
2    .comedies
3    .english_language
4    .rated(['PG-13', 'R'])
5    .released_after(1990)
6    .score_greater_than(80)
```

This is all well and good, but what happens when you go to test this query in the Rails console?

```
[2] pry(main)> Movie
=> Movie
[3] pry(main)> .comedies
Error: there was a problem executing system command: comedies
[4] pry(main)> .english_language
Error: there was a problem executing system command: english_language
[5] pry(main)> .rated(['PG-13', 'R'])
sh: -c: line 0: syntax error near unexpected token `['PG-13','
sh: -c: line 0: `rated(['PG-13', 'R'])'
Error: there was a problem executing system command: rated(['PG-13', 'R'])
[6] pry(main)> .released_after(1990)
sh: -c: line 0: syntax error near unexpected token `1990'
sh: -c: line 0: `released_after(1990)'
Error: there was a problem executing system command: released_after(1990)
[7] pry(main)> .score_greater_than(80)
sh: -c: line 0: syntax error near unexpected token `80'
sh: -c: line 0: `score_greater_than(80)'
Error: there was a problem executing system command: score_greater_than(80)
[8] pry(main)>
```

Multiline codes gets broken up when pasted in the console

The console assumes each line is an independent line of Ruby and executes after every line break. This means you would have to edit the text into one line before pasting.

Solution

Assuming you've configured your console to use Pry by default, you can use Pry's `edit` command to evaluate multiple lines at once.

To do this:

1. Copy the code you want to paste into the console.
2. Type `edit` without any arguments in the console. This will open a temporary file in your configured editor (let's assume this will be Vim).
3. Paste the code into the temporary file, save and exit (for Vim we'll use the handy `ZZ` shortcut to save and exit at once).
4. Now, Pry will evaluate your entire pasted statement.

Continue Reading

Well this is it, you've reached the end of the sample content.

I hope at this point you've discovered some new time-saving shortcuts for using the Rails Console that are going to make you more productive and bring a smile to your face every time you use them.

But this doesn't have to be the end! This is just one of 10 chapters in Efficient Rails.

Here are some things you'll learn in the rest of the book:

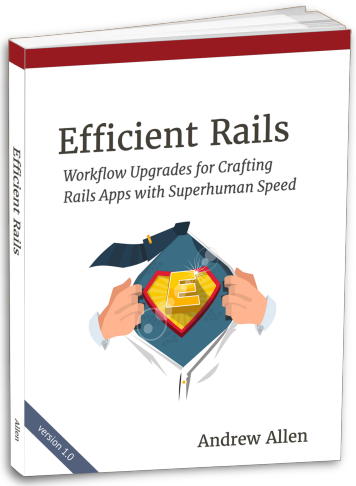
- Novel solutions to common problems in Rails – from the database to the view and everything in between
- Time-saving techniques that will make working with tools like Git and the Terminal a breeze
- Best practices for maintaining a solid test suite and a healthy codebase
- Advanced debugging tips (forget about puts!)
- And much more

Here's what people have been saying:

"Great book – a ton of value here and clear, concise writing"

"Loving the book. So cool, and really appreciate the structure"

"I can't stop reading it and trying it in my terminal"



Click here to get the full edition

– Andrew @allenan_