Owning Ownership

Who am I?

- Sean Griffin
- Rails Committer
- Maintainer of Active Record
- Creator of Diesel
- Bikeshed co-host
- @sgrif on Twitter

Ruby, JavaScript, and Python are dynamically typed

Your program still has types

Your program still has types

Semantics still exist, regardless of whether the language gives you a way to express or enforce them

Let's talk about ownership

Let's talk about ownership

What does it mean to own an object?

What does it mean to destroy an object?

What's an example of an object that can no longer be used at some point?

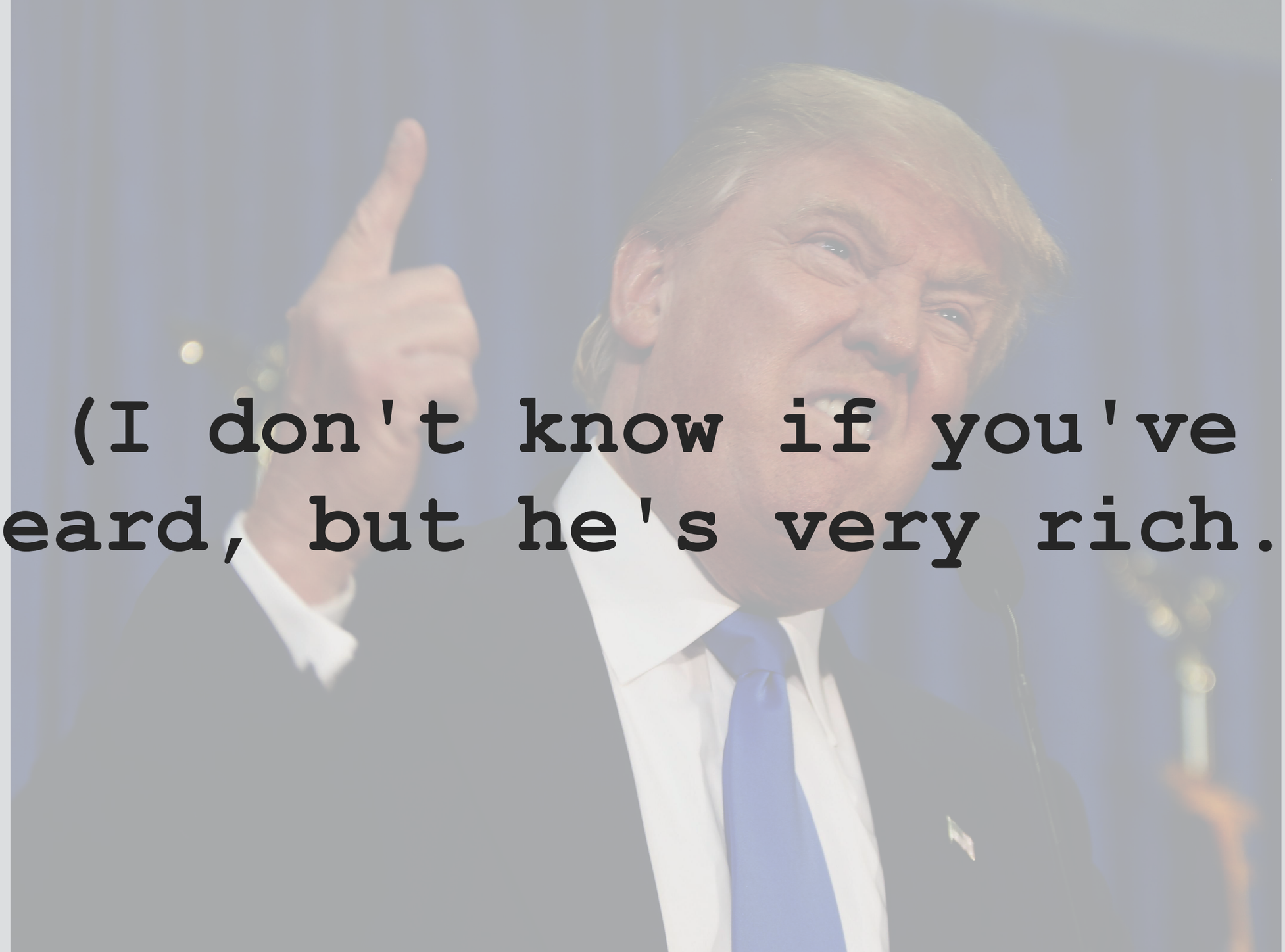
When will you stop answering questions with more questions?

Code time!

```
fn do_stuff_with_file(f: File) {
    // ...
    // f will be destroyed when this function returns
}

fn main() {
    let file = File::open("hello.txt");
    do_stuff_with_file(file);
    // ownership of file is moved into `do_stuff_with_file`

    do_more_stuff_with_file(file);
    // ERROR: Use of moved value `file`
}
```

```
fn do_stuff_with_file(f: &File) {
    //                        ^~~~~ Note the ampersand.
    // ...
    // f is borrowed, and therefore not destroyed
}

fn main() {
    let file = File::open("hello.txt");
    do_stuff_with_file(&file);
    // do_stuff_with_file borrows `file`, ownership is not moved.

    do_more_stuff_with_file(file);
    // This works fine now.
}
```

Objects can own things as well as functions



(I don't know if you've heard, but he's very rich.)

Caveat: Some types don't have owners

Ownership recap

- Owning an object means you have the right to destroy it
- Values have one owner
- If you need an object temporarily, you borrow it
- Objects can be borrowed either mutably or immutably

Ruby, JavaScript, and Python are dynamically typed

Your code still has to reason about types

These languages do not have an explicit concept of ownership

Your code still has to reason about ownership

Semantics still exist, regardless of whether the language gives you a way to express or enforce them

Enough of this Rust nonsense.
Let's look at some examples.

Ownership in Active Record

```ruby
class User < ActiveRecord::Base
  attribute :name, Type::String.new
end

name = "Sean"
user = User.new(name: name)
# Active Record has taken ownership of `name`

name << " Griffin"
# ^-- This would cause bugs!
```

Ownership in Active Record

```
class User < ActiveRecord::Base
  attribute :name, Type::String.new
end


name = "Sean"
user = User.new(name: name)
user.name << " Griffin"
```

Ownership in Active Record

```
class User < ActiveRecord::Base
  attribute :name, Type::String.new
end


user = User.new(name: "Sean")
user.name << " Griffin"
```

Let's look at a Rails bug!

(I kid, I kid)

```ruby
# Returns a hash with the \parameters used to form the \path of the request.
# Returned hash keys are strings:
#
#   {'action' => 'my_action', 'controller' => 'my_controller'}
def path_parameters
  @env[PARAMETERS_KEY] ||= {}
end
```

```
index 277207a..4defb7f 100644
--- a/actionpack/lib/action_dispatch/http/parameters.rb
+++ b/actionpack/lib/action_dispatch/http/parameters.rb
@@ -32,7 +29,7 @@ module ActionDispatch
       #
       #   {'action' => 'my_action', 'controller' => 'my_controller'}
       def path_parameters
-        @env[PARAMETERS_KEY] ||= {}
+        get_header(PARAMETERS_KEY) || {}
       end

     private
```

use get / set header to avoid depending on the ***env*** *ivar*

```
controller.request.path_parameters[:controller] =
  _controller_path
```

```
fn path_parameters(&mut self) -> &mut HashMap<String, String>;
```

```ruby
# Returns a hash with the \parameters used to form the \path of the request.
# Returned hash keys are strings:
#
#   {'action' => 'my_action', 'controller' => 'my_controller'}
def path_parameters
  get_header(PARAMETERS_KEY) || {}
end
```

Who owns the empty hash?

What do we expect the caller to do with the result of this method?

What do we expect the caller to do with the result of this method?

- Are they allowed to mutate it?

What do we expect the caller to do with the result of this method?

- Are they allowed to mutate it?
- Are they allowed to keep a strong reference to it?

What do we expect the caller to do with the result of this method?

- Are they allowed to mutate it?
- Are they allowed to keep a strong reference to it?
- Are we allowed to mutate it out from under them?

What do we expect the caller to do with the result of this method?

- Are they allowed to mutate it?
- Are they allowed to keep a strong reference to it?
- Are we allowed to mutate it out from under them?
- Are we allowed to replace the hash on ourselves? Does the caller expect this hash to always be associated with **self**?

What was the intention of the change?

call the
path_parameters=
setter rather than
rely on mutations

```
fn path_parameters(&mut self) -> &mut HashMap<String, String>;
```

```
fn path_parameters(&self) -> &HashMap<String, String>;
```

How can we make our ownership contract more explicit?

```ruby
def changed_attributes
  @changed_attributes ||=
    ActiveSupport::HashWithIndifferentAccess.new
end
```

```ruby
def changed_attributes
  super.reverse_merge(mutation_tracker.changed_values)
end
```

The return value isn't owned by Active Record, your mutations won't be seen by it

```
index 0bcfa5f..d40f352 100644
--- a/activerecord/lib/active_record/attribute_methods/dirty.rb
+++ b/activerecord/lib/active_record/attribute_methods/dirty.rb
@@ -80,7 +80,7 @@ module ActiveRecord
       def changed_attributes
         super.reverse_merge(mutation_tracker.changed_values)
+          .freeze
       end
```

Our contract becomes explicit

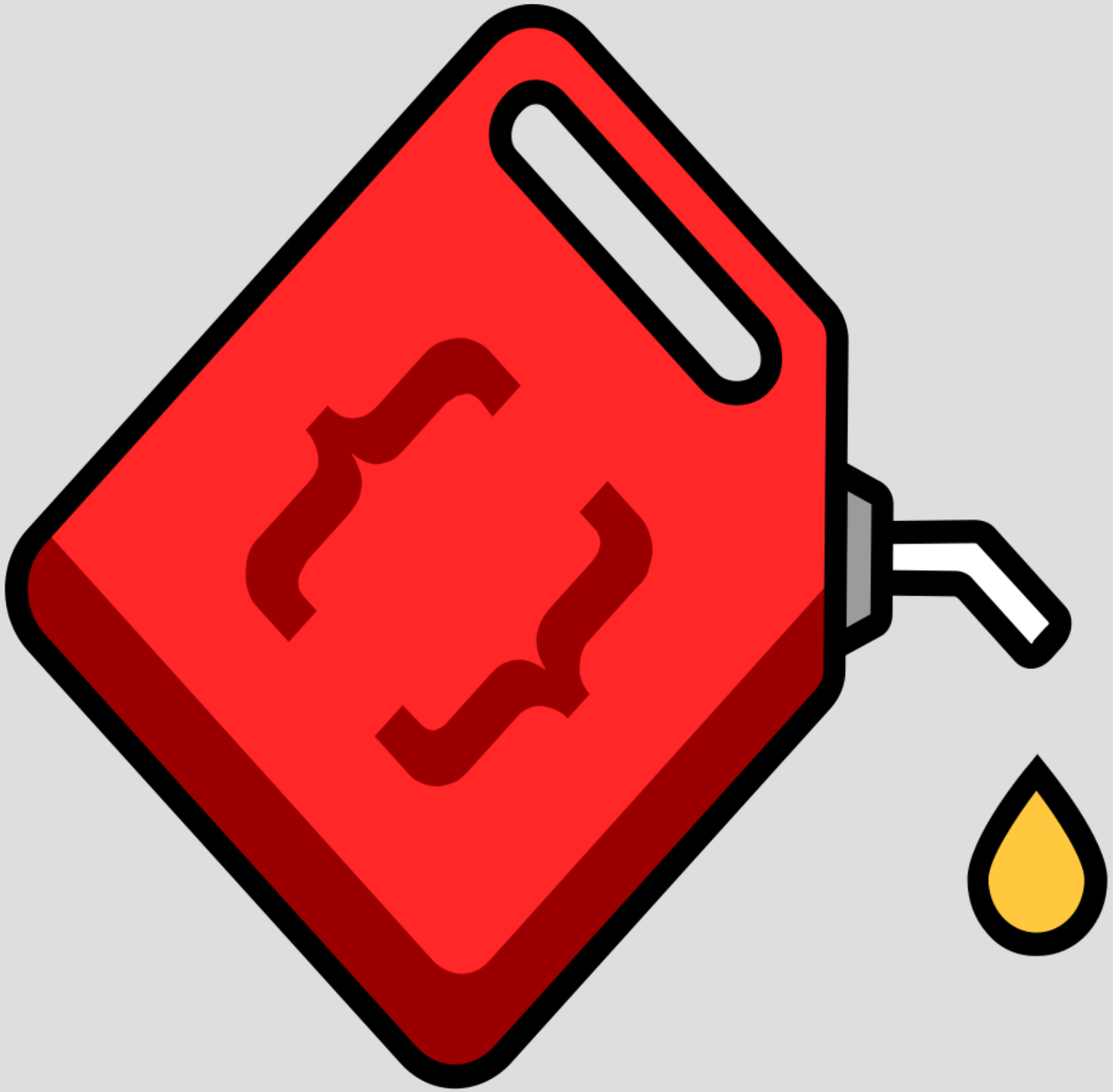
The bug was eventually resolved

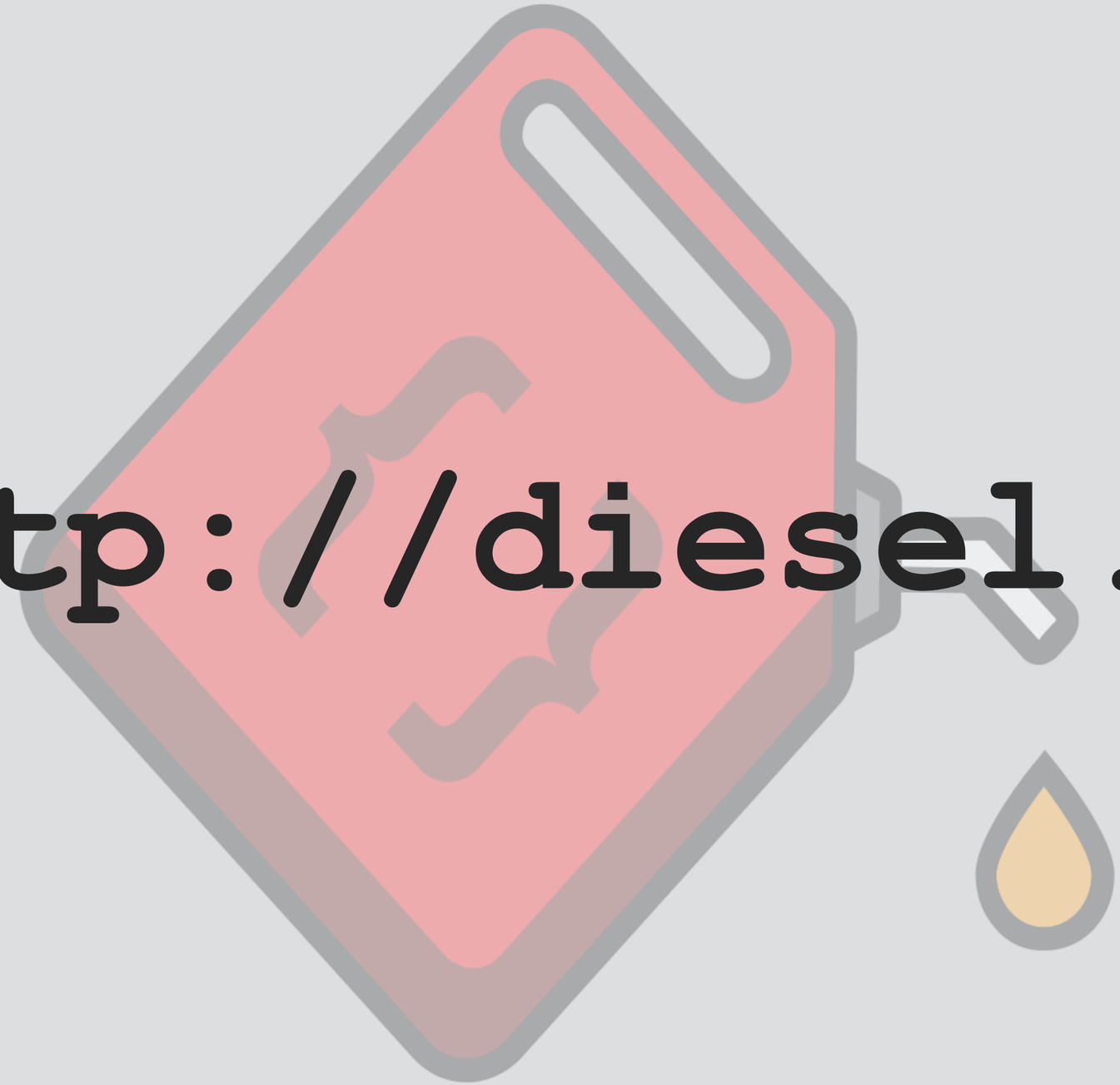
```
index c9df787..cca7376 100644
--- a/actionpack/lib/action_dispatch/http/parameters.rb
+++ b/actionpack/lib/action_dispatch/http/parameters.rb
@@ -43,7 +43,7 @@ module ActionDispatch
       #
       #   {'action' => 'my_action', 'controller' => 'my_controller'}
       def path_parameters
-        get_header(PARAMETERS_KEY) || {}
+        get_header(PARAMETERS_KEY) || set_header(PARAMETERS_KEY, {})
       end

       private
```

Remember the parameter hash we return. Callers expect to be able to manipulate it.

— The commit message resolving the bug

Let's start thinking about ownership in our code





http://diesel.rs

Thank you!

- Email: sean@seantheprogrammer.com
- Twitter: @sgrif
- Github: @sgrif
- http://diesel.rs
- http://bikeshed.fm