

class="heading"

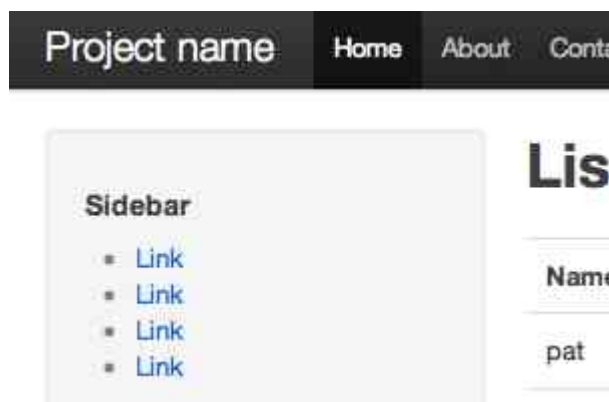
# Twitter Bootstrap, Less, and Sass: Understanding Your Options for Rails 3.1

Reddit  348  734 Email  35 [Hackernews](#)

This entry is part 1 of 3 in the series [Twitter Bootstrap and Rails](#)

## [Twitter Bootstrap and Rails](#)

- [Twitter Bootstrap, Less, and Sass: Understanding Your Options for Rails 3.1](#)
- [Too good to be true! Twitter Bootstrap meets Formtastic and Tabulous](#)
- [How to Customize Twitter Bootstrap's Design in a Rails app](#)



*Twitter Bootstrap is a great way to quickly build a very polished web site.*

By now, we've all seen [Twitter Bootstrap](#) – it's a great CSS and Javascript library open sourced by Twitter that makes it easy to produce a very polished looking site, with fantastic support for layout, navigation, typography, and much more. Twitter Bootstrap is based on [Less.js](#), the popular dynamic CSS scripting language written by [Alexis Sellier or @cloudhead](#). Less.js, like [Node.js](#), is implemented completely with Javascript. While Less is based on Javascript and not Ruby, some great work has been done just in the last couple of months to make it easy to set up Twitter Bootstrap in your Rails 3.1 app using a variety of different approaches.

Today I'm going to review the basics of Twitter Bootstrap, and then take a close look at the following gems and libraries: [less-rails-bootstrap](#), [sass-twitter-bootstrap](#), [bootstrap-sass](#) and [bootstrap-rails](#). With all of these different options available, it's hard to know exactly how to get started using Twitter Bootstrap with Rails. Before you dive in and start building the next killer Twitter Bootstrap Rails 3.1 app, be sure to understand how these different gems work under the hood so you can decide which one is right for you and your app.

## Twitter Bootstrap basics

On the surface, Twitter Bootstrap is simply a single CSS file, `bootstrap.css`, that provides all of the style and layout support, along with a few Javascript files that implement various dynamic features. This means the fastest and simplest way to get started using it is just to copy `bootstrap.css` into your Rails

3.1 app/assets/stylesheets folder like this:

[view source](#)

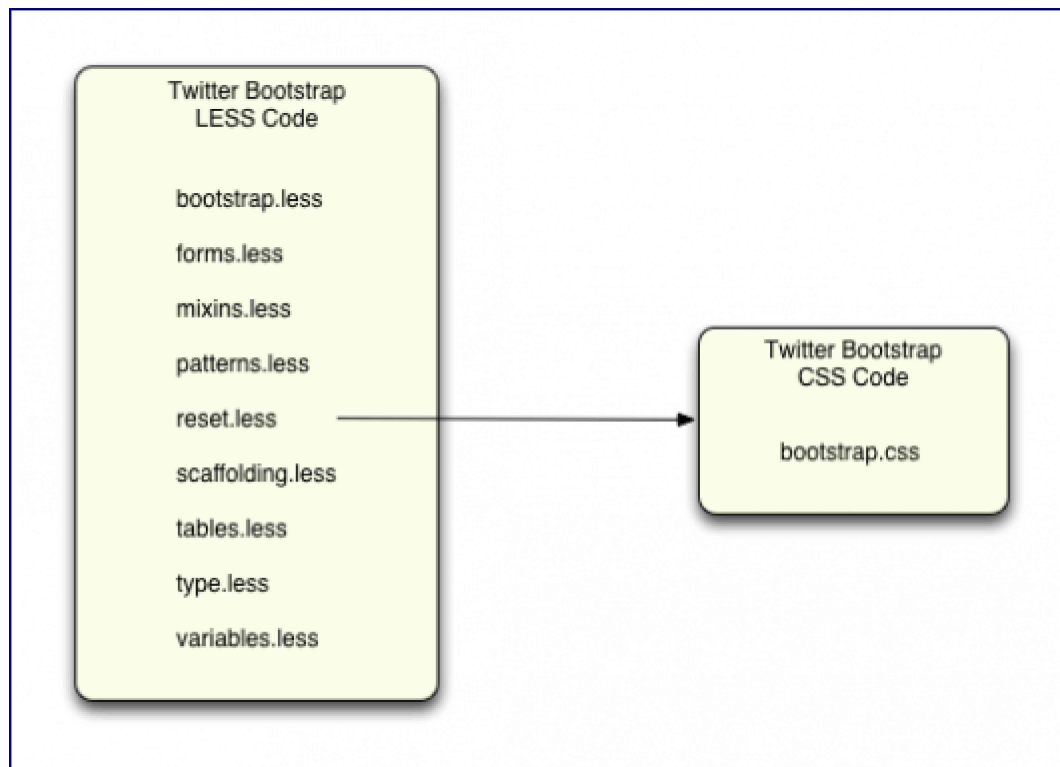
[print?](#)

```
1 $ git clone https://github.com/twitter/bootstrap.git
```

```
2 $ cp bootstrap/bootstrap.css path/to/app/assets/stylesheets/.
```

...and you're off to the races! Use their [online demo page](#) as a guide to get started; you can even use "Inspect Element" or "View Source" right on that page to get a sense of what styles they're using for each feature.

However, as I mentioned above, Twitter Bootstrap is based on the Less.js framework, and the bootstrap.css file is actually a compiled version of the Less code contained in a series of ".less" code files:



Twitter Bootstrap files

Looking in the Twitter Bootstrap github repo, the Less code files are contained in the [lib folder](#), while the compiled bootstrap.css file is right in the [root folder](#).

If you're not familiar with Less, it's very similar to Sass: it provides an enhanced, more powerful language for authoring CSS style code. As with Sass, you can use variables, mixins, nesting, etc. Think of Less as Sass implemented with Javascript instead of Ruby. If you're interested in learning more about Less take a look at [lesscss.org](#); there's also a great article out there by Jeremy Hixon, [An Introduction To LESS, And Comparison To Sass](#), that compares the two languages.

Therefore, as explained on the [Twitter Bootstrap introduction page](#), the best way to use Twitter Bootstrap in a Rails site is to use the Less source code written by the Twitter team directly, and not just the compiled CSS output. This allows you take advantage or override the styles provided by Twitter Bootstrap in a very straightforward way. But since Less isn't supported by the Rails 3.1 asset pipeline, this is a bit of a problem...

## Less-rails-bootstrap

Fortunately, [Ken Collins](#) did some great work during the past few months to solve this problem; he wrote a new gem called [less-rails-bootstrap](#) that adds support for Less into the Rails 3.1 asset pipeline. You can read his [blog post](#) or check out his [Github readme page](#) for the details. Another gem called [twitter-bootstrap-rails](#) by [Seyhun Akyürek](#) uses the same approach, although at first glance twitter-bootstrap-rails doesn't appear to have any tests in it, while Ken's less-rails-bootstrap does have an effective MiniTest::Spec test suite.

I'll repeat the setup steps from Ken's blog post here; first just add less-rails-bootstrap (or twitter-bootstrap-rails) to your Gemfile in the :assets group:

[view source](#)

[print?](#)

```
1 group :assets do
2   gem 'less-rails-bootstrap'
3 end
```

And install it using "bundle install." Then all you need to do is require the Twitter CSS code from your app/assets/stylesheets/application.css file like this:

[view source](#)

[print?](#)

```
1 /*
2  *= require twitter/bootstrap
3 */
```

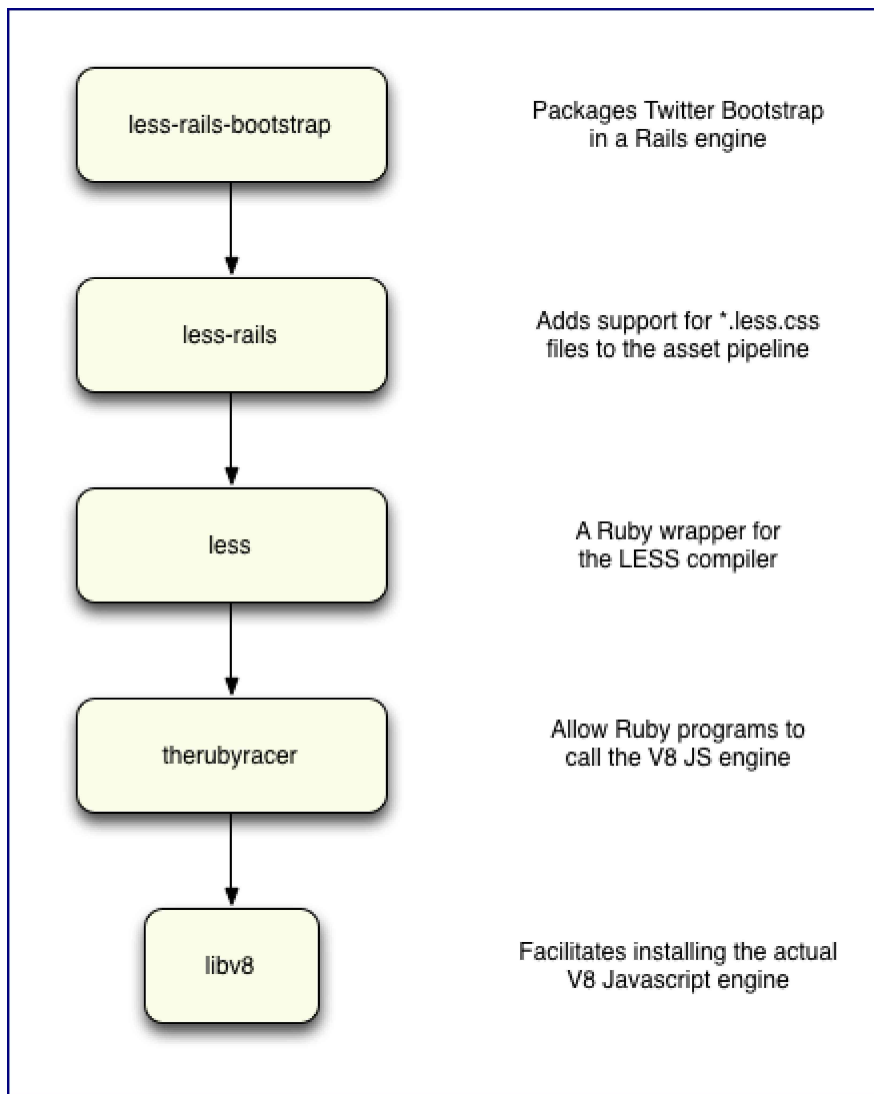
As Ken explains, now you're free to override and manipulate the Twitter Bootstrap Less code directly; for example if you add this code to a new file with a .css.less extension:

[view source](#)

[print?](#)

```
1 @import "twitter/bootstrap";
2 #foo {
3   .border-radius(4px);
4 }
```

... you now have a cross-browser compatible CSS style that will apply a rounded border with a radius of 4 pixels. But how does this actually work? Let's take a closer look:



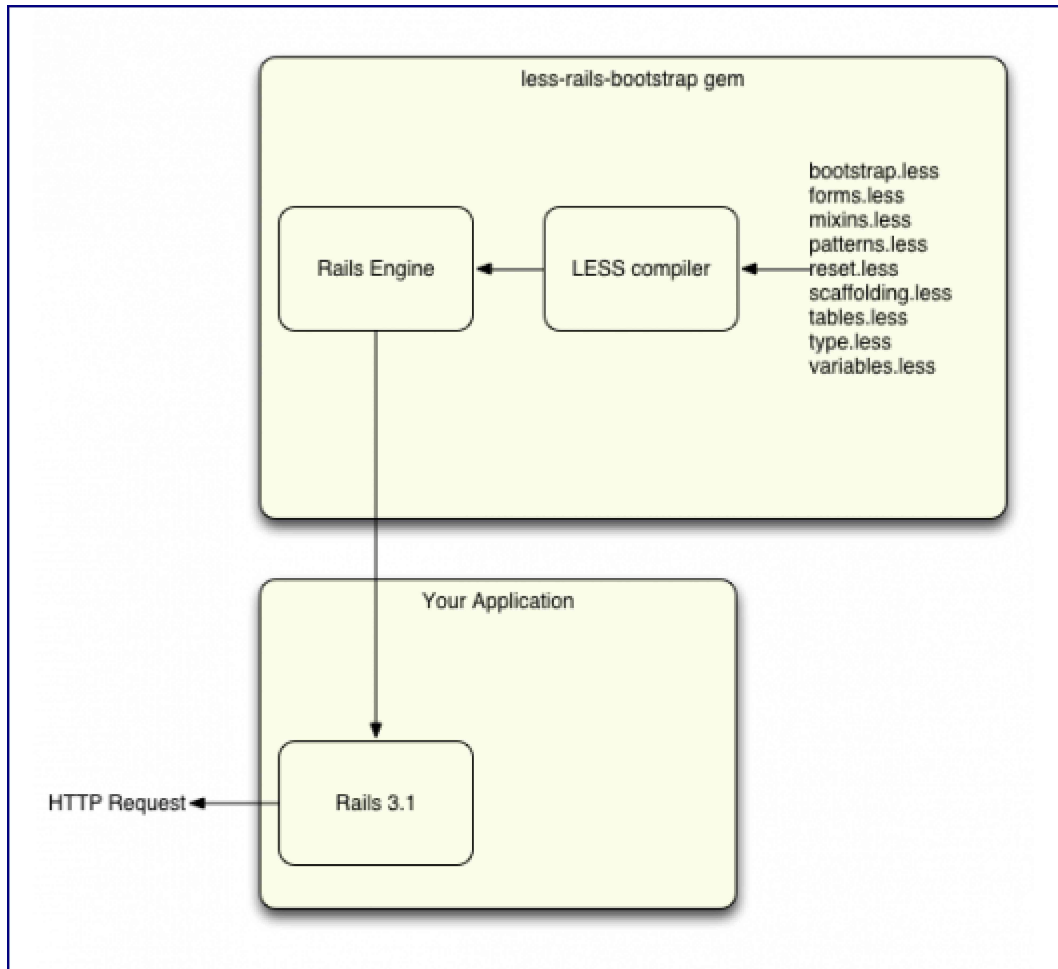
### What Gem does what?

In true Rails fashion, it turns out there's a lot of "magic" going on here behind the scenes. Before using this approach in your application, it's important to understand which gems are involved and what all of these gems are actually doing for you. Taking it from the top and diving down, here's how less-rails-bootstrap works:

- [less-rails-bootstrap](#): Ken's gem actually includes the Twitter Bootstrap Less code files (in the vendor/assets folder) and provides a Rails engine to make them accessible to your app. More on this in a minute...
- [less-rails](#): Ken wrote also wrote this gem, based on an earlier version by Karst Hammer, that helps integrate Less with the Rails 3.1 asset pipeline, providing extensions that Rails developers would expect. See Ken's clarification on this in the comments.
- [less.rb](#): This gem, written by Charles Lowell, is a thin wrapper around the Less language compiler, allowing your Rails app to call it.
- [therubyracer](#): Also written by Charles Lowell, this gem provides Ruby programs, in this case your Rails 3.1 app, the ability to call the V8 Javascript engine.
- [libv8](#): This gem makes it easy to install or compile from source the actual V8 Javascript library, which itself is implemented mostly in C.

With so much code involved, you might ask: Will less-rails-bootstrap slow my application to a crawl? The good news here is that you only need the Javascript V8 engine and all of the other code above it to compile the Less code files into CSS. That is, you'll only need all of this for development purposes while you're writing or modifying the CSS styles. Before you deploy to production you'll precompile your .css.less asset files, the same way you already do with your other CSS or Javascript files – and none of these gems will actually be used on your production web server.

Another important piece of magic that Ken has implemented is the way he packaged up Twitter's Less code file using a Rails engine. Here's how that works:



less-rails-bootstrap gem

If you're not familiar with Rails engines, take a few minutes to watch [Ryan Bates explain how they're implemented in Rails 3.1](#). Less-rails-bootstrap implements the simplest kind of Rails engine, providing your Rails 3.1 app access to the Twitter code as additional, static asset files. When an HTTP request comes in from the user – in this case you, since you only use this code during development – it's forwarded on by Rails to the engine in the less-rails-bootstrap gem. In practice, you don't really need to worry about how this works, but it is good to know where the Twitter Less code files are located, in case you need to find a specific style definition or search for something else:

[view source](#)  
[print?](#)

```
01 $ cd `bundle show less-rails-bootstrap`
02 $ find vendor
```

```
03 vendor
04 vendor/assets
05 vendor/assets/javascripts
06 vendor/assets/javascripts/twitter
07 vendor/assets/javascripts/twitter/bootstrap
08 vendor/assets/javascripts/twitter/bootstrap/alerts.js
09 vendor/assets/javascripts/twitter/bootstrap/buttons.js
10 vendor/assets/javascripts/twitter/bootstrap/dropdown.js
11 vendor/assets/javascripts/twitter/bootstrap/modal.js
12 vendor/assets/javascripts/twitter/bootstrap/popover.js
13 vendor/assets/javascripts/twitter/bootstrap/scrollspy.js
14 vendor/assets/javascripts/twitter/bootstrap/tabs.js
15 vendor/assets/javascripts/twitter/bootstrap/twipsy.js
16 vendor/assets/javascripts/twitter/bootstrap.js
17 vendor/assets/stylesheets
18 vendor/assets/stylesheets/twitter
19 vendor/assets/stylesheets/twitter/bootstrap.css.less
20 vendor/frameworks
21 vendor/frameworks/twitter
22 vendor/frameworks/twitter/bootstrap
23 vendor/frameworks/twitter/bootstrap/bootstrap.less
24 vendor/frameworks/twitter/bootstrap/forms.less
25 vendor/frameworks/twitter/bootstrap/mixins.less
26 vendor/frameworks/twitter/bootstrap/patterns.less
27 vendor/frameworks/twitter/bootstrap/reset.less
28 vendor/frameworks/twitter/bootstrap/scaffolding.less
29 vendor/frameworks/twitter/bootstrap/tables.less
30 vendor/frameworks/twitter/bootstrap/type.less
31 vendor/frameworks/twitter/bootstrap/variables.less
32 vendor/frameworks/twitter/bootstrap.less
```

## Sass-twitter-bootstrap

While Ken's done a great job with less-rails-bootstrap, you may prefer to use Sass instead of Less, because it's already supported by Rails 3.1 out of the box, or because your application already contains a large amount of Sass code, or possibly just because you're more familiar with it. At first glance, this seems to be a serious problem for Twitter Bootstrap: it's not appropriate for a large portion of the Rails community that prefers Sass, since it was implemented with the Javascript-centric Less.js technology.

Don't worry – some other great developers (John Long, Jeremy Hinegardner and others) ran into this dilemma already and came up with a solution: they translated Twitter Bootstrap's Less code into Sass, and released that as a separate library on Github, called [Sass-twitter-bootstrap](#).

Sass-twitter-bootstrap is not a gem, but instead is just a github repo containing Twitter's translated code. To use it in your Rails 3.1 app, just clone the repo and copy the bootstrap css file into your app, like this:

[view source](#)

[print?](#)

```
1 $ git clone https://github.com/jlong/sass-twitter-bootstrap.git
2 $ cp sass-twitter-bootstrap/bootstrap.css
   path/to/app/assets/stylesheets/.
```

Of course, this isn't really any different than copying the bootstrap.css file directly from the Twitter Bootstrap repo; instead what you should do is copy the Sass source files right into your application like this:

[view source](#)

[print?](#)

```
1 $ rm path/to/app/assets/stylesheets/bootstrap.css
2 $ cp -r sass-twitter-bootstrap/lib path/to/app/assets/stylesheets/twitter
```

And now since the Rails 3.1 asset pipeline supports Sass out of the box, we're good to go... almost! If you run your app now, you'll see an error:

[view source](#)

[print?](#)

```
1 ActionView::Template::Error (Undefined variable: "$baseline".
2   (in /path/to/app/assets/stylesheets/twitter/forms.scss))
```

Thanks to [Brent Collier](#), there's a simple solution for this problem: the code in application.css by default uses "require\_tree" to include all of the code under app/assets/stylesheets:

[view source](#)

[print?](#)

```
1 /*
2  * This is a manifest file that'll automatically include all the
3  * stylesheets available in this directory
4  * and any sub-directories. You're free to add application-wide styles to
5  * this file and they'll appear at
6  * the top of the compiled file, but it's generally better to create a
7  * new file per style scope.
8  *= require_self
9  *= require_tree .
10 */
```

As Brent explains, the problem with this is that Twitter Bootstrap's Less code (and the translated Sass version) was designed to be included once using the bootstrap.scss file, which in turns includes all of the other files. Brent's solution works perfectly: just remove "require\_tree" and require bootstrap.scss directly:

[view source](#)

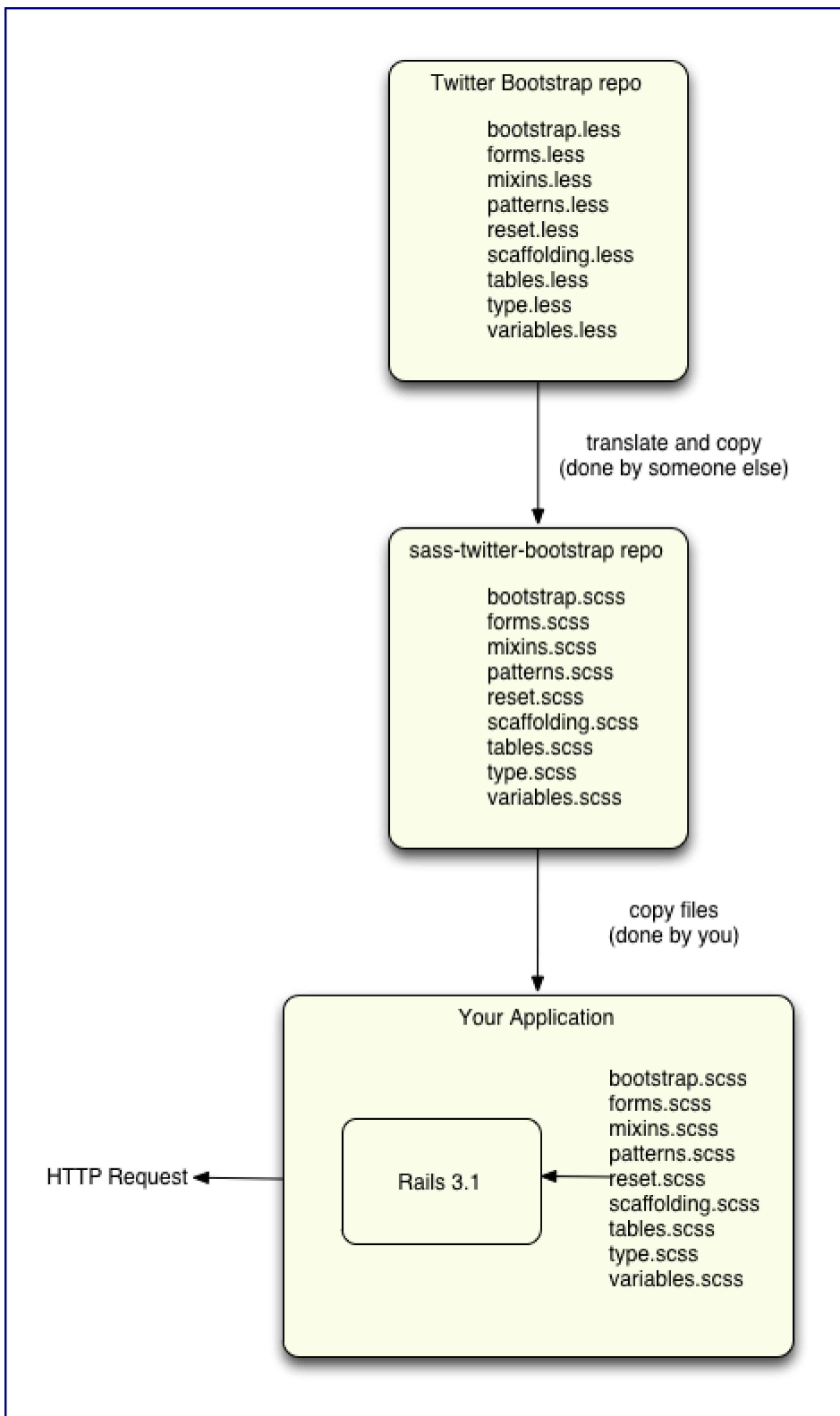
[print?](#)

```
1 /*
2  * This is a manifest file that'll automatically include all the
```

```
stylesheets available in this directory
3 * and any sub-directories. You're free to add application-wide styles to
  this file and they'll appear at
4 * the top of the compiled file, but it's generally better to create a
  new file per style scope.
5 *= require_self
6 *= require twitter/bootstrap
7 */
```

Conceptually, here's what this setup looks like:





### Copying sass files

The benefit here is that now you have Sass code implementing Twitter Bootstrap directly inside your Rails 3.1 app! This means you can go ahead and use, override or modify the Sass to your heart's content.

The obvious drawback here is the manual copy process involved. You'll need to copy the Sass code files once into your Rails app to get started, but then you'll have to repeat the copy each time Twitter releases a newer, better version of Twitter Bootstrap. While this won't happen every day, why set yourself up for an ongoing maintenance problem?

Another drawback is that you're using a second-hand, translated version of Twitter's code, and you'll have to trust the people who maintain sass-twitter-bootstrap to accurately translate the Less into Sass.

## Bootstrap-sass and bootstrap-rails

Like everything in the Rails world, there are always other good solutions out there you can take a look at, in this case the [bootstrap-sass gem](#) by Thomas McDonald and the [bootstrap-rails gem](#) by [AnjilLab](#). Both of these gems combine the Sass translation approach with a Rails engine to avoid the manual copy maintenance headache.

Using bootstrap-sass (or bootstrap-rails) is as simple as adding it to your Gemfile:

[view source](#)

[print?](#)

```
1 group :assets do
2   gem 'bootstrap-sass'
3 end
```

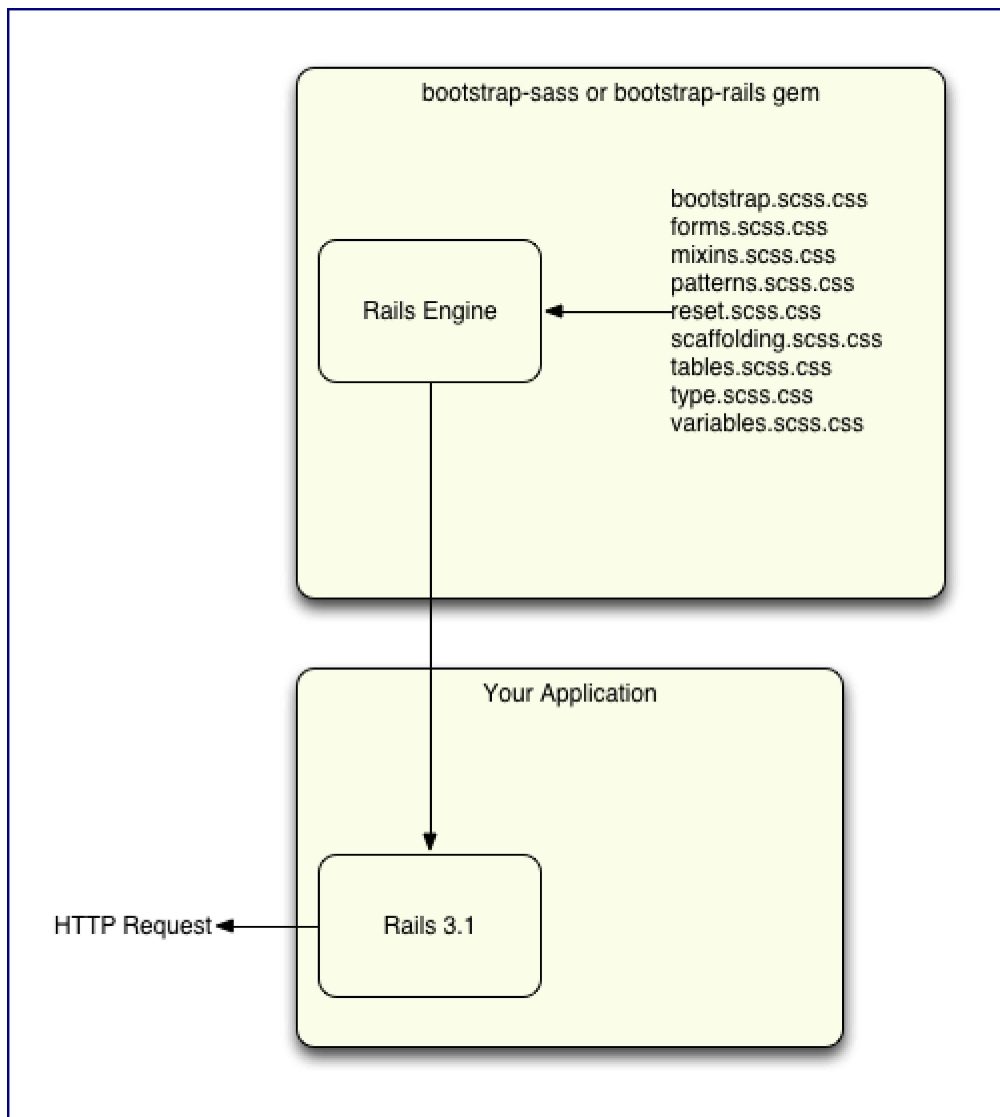
Run “bundle install” and then add a require statement to app/assets/stylesheets/application.css:

[view source](#)

[print?](#)

```
1 /*
2  *= require bootstrap
3 */
```

Similar to less-rails-bootstrap, this works by having Rails 3.1 load the Sass code from a Rails engine inside the gem:



## Bootstrap sass and rails

So there's no need to download and copy either the Twitter Bootstrap Less code, or its translated Sass version. As new versions of Twitter Bootstrap are released, presumably Thomas McDonald will re-translate the Less code into Sass, and re-release his gem. Then you'll be able to just run "bundle update" to get the new Twitter code into your application!

## Other options

Amazingly, there are even more options out there for integrating Rails 3.1 and Twitter Bootstrap that I don't have time to cover today:

- [compass-twitter-bootstrap](#) is similar to bootstrap-sass and bootstrap-rails, using a Rails engine to provide a translated version of the Twitter Bootstrap code, but is geared instead to the [Compass CSS framework](#).
- [twitter\\_bootstrap\\_form\\_for](#) implements a custom Form Builder object, the object yielded by form\_for in your view, designed specifically for Twitter Bootstrap.
- [css-bootstrap-rails](#) is similar to bootstrap-sass and bootstrap-rails, but uses a Rails engine to include only bootstrap.css without any Less or Sass files.

## Six of one, half dozen of the other

All of these approaches will work equally well:

- Using the original Less code via a Rails engine (less-rails-bootstrap or twitter-bootstrap-rails)
- Copying in translated Sass code directly into your application (sass-twitter-bootstrap), or
- Using a translated Sass version via a Rails engine (bootstrap-sass or bootstrap-rails).

Less and Sass are very similar, and using one language or the other is really a matter of personal preference. Similarly, using a Rails engine is a convenient way to include the Twitter Bootstrap code in your Rails 3.1 app and to manage upgrades smoothly. However, using a Rails engine adds some additional “magic” to your app and can be somewhat confusing when you need to find, inspect or search against the Sass code. There’s a simple elegance to the approach of just copying the Sass code right into your app, and you may not often need to update to a newer version of Twitter Bootstrap.

Which approach to take is up to you!

Series Navigation [Too good to be true! Twitter Bootstrap meets Formtastic and Tabulous >>](#)

Tagged with: [bootstrap](#), [twitter](#)



## Pat Shaughnessy

Pat Shaughnessy is a Ruby developer working at a global management consulting firm. Pat also writes a weekly blog column about Ruby development at [patshaughnessy.net](http://patshaughnessy.net). Pat's tutorials and articles are geared towards Ruby beginners but contain enough information and detail to be interesting to more advanced developers also. Pat's articles and presentations have been featured multiple times on the Ruby Weekly newsletter, the Ruby5 podcast and the Ruby Show.

<http://patshaughnessy.net> [Twitter](#)

← [Crafting Rubies: Best Practices While Cutting Gems](#)  
[Class Variables – A Ruby Gotcha](#) →

## 15 Comments



1. [Décio Ferreira](#) 16 Nov 11 | 9:54 am

Great article Pat. A bit of shameless self promotion, but could I just suggest one more option, my bootstrap-generators gem that provides Twitter Bootstrap generators for Rails 3.1.

The gem is usable but still under development, so feedback or contributions are very welcome.

[Reply](#)

• [Pat Shaughnessy](#) 17 Nov 11 | 8:33 am



Thanks Décio! I'll definitely check out your gem; using generators is one approach I hadn't thought of before...

[Reply](#)

2. [Ken Collins](#) 18 Nov 11 | 5:18 am



Pat, another great article! Some feedback:

First, I have never used RSpec, both the less-rails and less-rails-bootstrap projects use MiniTest::Spec. I tend to use bare bones testing frameworks for my projects so I can stay away from complex test systems for the gems I author. I like that MiniTest is the TU replacement and part of ruby-core.


Also, technically Tilt, when used with Charles' less.rb gem, is enough to render css.less templates in the asset pipeline. I made the less-rails gem to solve two critical problems (a) a standard way to augment less.rb's paths to when used with the asset pipeline, critical for gems like less-rails-bootstrap to hook into and (b) provide extensions to less.rb that rails developers would expect, like the asset url helpers. My favorite of which is the asset-data-url for base64 encoded images.

Another key reason that I made less-rails-bootstrap was to solve the lack of name-spacing I witnessed in other packages including the bootstrap file. I predict that as the asset pipeline gains more adoption, people will learn that there assets need to be namespaced (via directories) just like we are used to our gems and models. So you are free to happenstantially make your own variables.less file in your assets path and never worry about it conflicting with twitter/bootstrap/variables.less. No other gems that I saw are thinking about this potential problem we will face and eventually gem authors will start to understand the importance of

namespacing their assets. Because of this the recent less-rails-bootstrap tries to follow a convention I found in Sass/Compass where the path to the raw source files are in a vendor/frameworks directory, which I use less-rails to make sure is in the renderable paths for less.

Again, great article!

[Reply](#)

- [Pat Shaughnessy 18 Nov 11 | 8:48 am](#) 

Thanks for your kind comments, Ken! And thanks so much for the corrections. Yea it's funny the MiniTest::Spec code resembles RSpec at first glance, actually... I'll have to try MiniTest myself soon. And sorry to get the purpose of the less.rb gem wrong; thanks for the thorough clarification. I've updated the article text above for both mistakes.

You make a good point about namespacing - that sounds like a really great idea.

Maybe what I'll do is write a follow up article someday soon about more of the details of how Less and the Rails asset pipeline work together. I was covering so much ground in this article I really didn't have time to get into the real details. ...or maybe I should let you write that one :)

Keep up the great work on less-rails-bootstrap!

[Reply](#)

3. [oz 19 Nov 11 | 1:48 pm](#) 

“Less.js, like Node.js, is implemented completely with Javascript.” Hm nope. Node.js is a good deal of Javascript, but it wouldn't exist without all the C++ around it. ;)

Nonetheless, thanks for a great article.

[Reply](#)

4. [Bill Christian 22 Nov 11 | 9:13 am](#) 

Which option is best for overriding the Bootstrap variables? I get a little lost on how I can change the base link color (in less or sass versions) without editing the source files.

[Reply](#)

- [Pat Shaughnessy 22 Nov 11 | 3:50 pm](#) 

Hi Bill, If you're using Less and have the less-rails-bootstrap gem installed, then just create an custom\_variables.css.less file anywhere under app/assets/stylesheets and add this Less code to it, for example:

```
@import "twitter/bootstrap";  
@linkColor: #0F0;
```

If you're using Sass, then you don't need any gem installed since Rails 3.1 supports it by default. So you just need to create custom\_variables.css.scss file instead, and then use this Sass code inside it:

\$linkColor: #F00;

But for Sass it looks like you'll have to modify the bootstrap.css.scss file - either the one you copied from sass-twitter-bootstrap or else the one inside of bootstrap-sass, for example - and import your new custom\_variables.css.scss file like this:

```
@import "bootstrap/variables";  
@import "custom_variables";  
@import "bootstrap/mixins";
```

This last part seems very ugly to me; I'll look into it some more and write here again if I find a cleaner way to override Sass variables.


[Reply](#)

- [Bill Christian 22 Nov 11 | 8:28 pm](#) 

So if I understand correctly, I can override the \*existing\* LESS variables defined in the original sources by using the import and the new LESS statement. Do I need to modify the require statement in the application.css to specify a cascading order? Also, should i just import the twitter/bootstrap/variables file or do I need to bring in everything in my override file?

Thanks for your help. If you have a gist or github showing an example, I'll stop with the questions.

[Reply](#)

- [Thomas McDonald 02 Dec 11 | 3:47 pm](#) 

Indeed it is, and the update to bootstrap-sass I've just pushed means that if you define the variables you want to change before you @import "bootstrap" then SASS will not overwrite your previously defined variables with the bootstrap defaults.

[Reply](#)

- [Pat Shaughnessy 03 Dec 11 | 7:18 am](#) 

Cool! Thanks for letting us know, Thomas. I'll get and try out your changes very soon...

[Reply](#)

5. [Pat Shaughnessy 23 Nov 11 | 1:45 pm](#) 

Questions are no problem at all! I'm happy to try to help. And sorry for the slow reply - this is actually a complicated issue.

So after some more research, I now think the best way to override Twitter Bootstrap variables is to:

1. Remove application.css and instead use application.css.less or application.css.scss, and:
2. Import (not require) each of the TB files individually, including your override values after

including the TB variables file.

Application.css.less example (using less-rails-bootstrap): <https://gist.github.com/1389826>

Application.css.scss example (using bootstrap-sass for example):  
<https://gist.github.com/1389831>

Both Less and Sass seem to have the same issue, and work equally well.

Notes:

- If you instead used `@import` on the entire TB tree, like I show above in the article, then all the TB css code might be included more than once in your generated css file!
- In either case you should not use `require_tree`, but just `require_self`.

See: <http://stackoverflow.com/questions/6420460/in-rails-3-1-is-it-really-impossible-to-avoid-including-duplicate-copies-of-sty> and <http://railscasts.com/episodes/268-sass-basics> for more details on this.

Sorry for the long comment - I should have covered this in the article :)

class="heading"

## Too good to be true! Twitter Bootstrap meets Formtastic and Tabulous

Reddit  413  451 Email  1 [Hackernews](#)

This entry is part 2 of 3 in the series [Twitter Bootstrap and Rails](#)

[Twitter Bootstrap and Rails](#)

- [Twitter Bootstrap, Less, and Sass: Understanding Your Options for Rails 3.1](#)
- Too good to be true! Twitter Bootstrap meets Formtastic and Tabulous
- [How to Customize Twitter Bootstrap's Design in a Rails app](#)



## New origami

Name	<input type="text" value="Divine Dragon"/>
Artist	<input type="text" value="Satoshi Kamiya"/>
<input type="button" value="Create Origami"/>	

This simple 5 step tutorial will create a working Rails 3.1 app using Twitter Bootstrap

[Last month I discussed](#) all of the different gems that are available for integrating [Twitter Bootstrap](#) with a Rails 3.1 app. These gems make the Twitter Bootstrap CSS styles available in your Rails 3.1 asset pipeline as Less or Sass code. However, getting the Twitter CSS code is just the first step – you also need to know how to use it in a Rails app.

Today I'll take the next step. This article is a detailed, step by step tutorial that will take you through the process of writing a new Rails 3.1 app that uses Twitter Bootstrap with [Formtastic](#) and [Tabulous](#), two great gems that make it easier to implement web forms and tab based navigation in Rails – two major features of Twitter Bootstrap. We'll also use a third gem called [Formtastic-Bootstrap](#) that modifies the HTML produced by Formtastic to play nicely with the Twitter Bootstrap files.

Read on to learn more... in just five easy steps we'll have a simple web site up and running that illustrates how to implement Twitter Bootstrap forms and tab navigation using Rails 3.1!

### Step 1: Create a new Rails app

Let's get started by creating a brand new Rails 3.1.3 app, the latest version of Rails available as I write this:

[view source](#)

[print?](#)

```
1 $ rails -v
2 Rails 3.1.3
3 $ rails new origami_hub
4 $ cd origami_hub
5 $ rm public/index.html
6 $ rails generate controller welcome index
```

You can see we're going to create a web site to keep track of Origami artwork! You can also see I deleted the default, static home page, public/index.html. And the last command generates a controller called "Welcome" that will handle displaying the web site's home page.

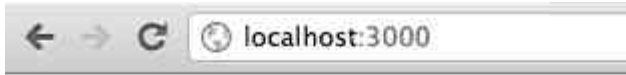
The next step is to edit the config/routes.rb file. Go ahead and paste this into the routes file:

[view source](#)

[print?](#)

```
1 OrigamiHub::Application.routes.draw do
2   root :to => 'welcome#index'
3 end
```

Finally, let's start up our Rails server (type "rails server") and open <http://localhost:3000>:



## Welcome#index

Find me in `app/views/welcome/index.html.erb`

Default page created by rails controller generator

You should see this text in the site's new home page; if you don't then double check the commands above carefully.

## Step 2: Add Bootstrap-Sass

Today I'll use the [Bootstrap-Sass](#) gem by [Thomas McDonald](#) to import the Twitter Bootstrap code. To learn more about Bootstrap-Sass and how it works, check out [my post from November](#). Adding it to our new Rails app is simple; first edit your Gemfile and add it to the "assets" group:

[view source](#)

[print?](#)

```
1 # Gems used only for assets and not required
2 # in production environments by default.
3 group :assets do
4   gem 'bootstrap-sass'
5 etc...
```

Then you need to install it with Bundler:

[view source](#)

[print?](#)

```
1 $ bundle install
```

Next, we need to create a new Sass code file to hold our CSS style code. Create a file called "origami\_hub.css.scss" in the `app/assets/stylesheets` folder with this one line of code in it. This imports the Twitter Bootstrap CSS styles into our app:

[view source](#)

[print?](#)

```
1 @import 'bootstrap';
```

Now if you stop, re-run your Rails server and refresh the page you should see the same thing, but with subtle changes in the font and layout:



# Welcome#index

Find me in `app/views/welcome/index.html.erb`

Default page with Twitter Bootstrap styles

This may not look much different, but actually there are subtle changes in the fonts, and also the text is flush against the top left corner now. These changes mean we have successfully loaded Twitter Bootstrap into our Rails app! If you ran View → Source in your browser and inspected the code returned by the `origami_hub.css` request, you'd see all of the Twitter styles.

## Step 3: Add Scaffolding

So far, so good; now we have our site's navigation bar appearing correctly. As a next step, let's create an Origami model along with pages for displaying and editing them – Rails scaffolding will be perfect. For today, I'll just use two string attributes; the name of the origami artwork and the name of the artist. Here are the commands:

[view source](#)

[print?](#)

```
1 $ rails generate scaffold origami name:string artist:string
```

```
2 $ rake db:migrate
```

```
3 $ rm app/assets/stylesheets/scaffolds.css.scss
```

```
4 $ rm app/assets/stylesheets/origamis.css.scss
```

Notice that I deleted two generated files called “`scaffolds.css.scss`” and “`origamis.css.scss`” which interfere with Twitter Bootstrap's styles. Now if you open the origami index page, `http://localhost:3000/origamis`, you'll see this:



# Listing origamis

Name	Artist
------	--------

[New Origami](#)

Index page from Rails scaffold generator with Twitter Bootstrap styles

This should look familiar if you've ever used Rails scaffolding before, but here we also see the Twitter Bootstrap table styling.

## Step 4: Add Formtastic using Formtastic-Bootstrap

Our next step today will be to introduce [Formtastic](#), written by [Justin French](#), into our sample app. If

you're not familiar with it yet, Formtastic produces cleaner HTML than the standard Rails "form\_for" and related methods, making it easier to create a nice looking form. It's also easier to use while writing Rails view code. Finally, Formtastic's HTML is also easier to work with while writing Javascript code or Cucumber tests, since there are CSS styles added to make finding specific form elements easier.

Unfortunately, Formtastic and Twitter Bootstrap don't play well together: Twitter's styles don't refer to the HTML tags produced by Formtastic. But we can get these two components to work together by using a great new gem called [Formtastic-Bootstrap](#), by [Matthew Bellantoni](#) – a Boston Rubyist like me. Take a look at the github readme page for more details on how Formtastic-Bootstrap actually works.

To put it into our application, I need to first add it to my Gemfile:

[view source](#)



[print?](#)

```
1 gem 'formtastic-bootstrap'
```

And now as Matthew explains on his readme page, I also need to create a new file called `config/initializers/formtastic.rb`, which should contain this one line of code:

[view source](#)



[print?](#)

```
1 Formtastic::Helpers::FormHelper.builder =  
  FormtasticBootstrap::FormBuilder
```

This instructs Formtastic to use Matthew's new form builder object, instead of Formtastic's standard form builder. Matthew's form builder is what generates the HTML Twitter expects.

Next, I need to rewrite the form produced by the Rails scaffold generator to use Formtastic's `semantic_form_for` function – paste this code into `app/views/origamis/_form.html.erb`, overwriting what was there before:

[view source](#)

[print?](#)

```
01 <%= semantic_form_for @origami do |f| %>  
02   <%= f.semantic_errors %>  
03   <%= f.inputs do %>  
04     <%= f.input :name, :hint => "Enter the origami artwork's name" %>  
05     <%= f.input :artist, :hint => "Enter the origami artist's name" %>  
06   <% end %>  
07   <%= f.buttons do %>  
08     <%= f.commit_button :button_html => {:class => "primary"} %>  
09   <% end %>  
10 <% end %>
```

A couple of details to note here:

- I've passed in the Formtastic "hint" option for each of my fields to add some help text on the screen, and
- I've passed in the "button\_html" option to use the Twitter Bootstrap blue button style

We need to run `bundle install` again to install Matthew's gem; bundler will also install Formtastic

itself since it's a dependency for Formtastic-Bootstrap:


[view source](#)



[print?](#)

```
1 $ bundle install
```

After restarting the server and clicking on the “New Origami” link I get a nice looking form that uses the Twitter Bootstrap styles:



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/origamis/new'. The page title is 'New origami'. Below the title, there are two input fields: 'Name' with the placeholder text 'Enter the origami artwork's name' and 'Artist' with the placeholder text 'Enter the origami artist's name'. At the bottom of the form is a blue button labeled 'Create Origami'.

[Back](#)

Form rendered by FormtasticBootstrap::FormBuilder

## Step 5: Adding tab navigation with Tabulous

Another one of the great features of Twitter Bootstrap is the nice looking black navigation bar along the top of the page. How can we get that into our sample app? We could just copy/paste the HTML code in from one of the Twitter examples pages, but instead I'll show you how to use a great new gem called [Tabulous](#). [Wyatt Greene](#), another fellow Boston Rubyist, wrote Tabulous earlier this year and it's tremendously easy to use. We're lucky to have such a creative Ruby community here: two of the four gems I'm using today with Twitter Bootstrap were written in Boston! For more information on Tabulous, be sure to read two great articles Wyatt wrote when he introduced the gem in March: [Introducing Tabulous: Tabs in Rails](#) and [Tutorial for Adding Tabs to Rails Using Tabulous](#).

Once again, let's first add the gem to our app by editing the Gemfile:

[view source](#)



[print?](#)

```
1 gem 'tabulous'
```

...and running bundle install:

[view source](#)



[print?](#)

```
1 $ bundle install
```

Now as Wyatt explains in his tutorial, we need to run this command:

[view source](#)

[print?](#)

```
1 $ rails generate tabs
```

This will create a new file called `app/tabs/tabulous.rb`, where we declare all of the tabs we want to display in our site. I won't explain all of the syntax here; again refer to [Wyatt's tutorial](#) for more information on exactly what's in `tabulous.rb`.

For our sample app, we'll just need to make a minor edit to the tabs table:

[view source](#)

[print?](#)

```
1 config.tabs do
2 [
3   [ :welcome_tab, 'Welcome', root_path, true, true ],
4   [ :origamis_tab, 'Origamis', origamis_path, true, true ],
5 ]
6 end
```

It's amazing how Wyatt's generator already did most of the work for us! All I did here was reorder the two lines. For the sake of readability and to save space I removed some comments that normally appear in the `tabulous.rb` file explaining the purpose of each column in the table; you'll see those if you open the `tabulous.rb` file on your machine.

And we're almost done! The last thing I need to do is add the HTML for displaying the Twitter Bootstrap navigation bar. To do that, I'll just copy/paste some HTML from their "fluid.html" example page – you can find this HTML in the [Twitter Bootstrap github repo](#). The HTML should go into the `app/views/layouts/application.html.erb` file, between the `<body>` tag and the call to `yield`:

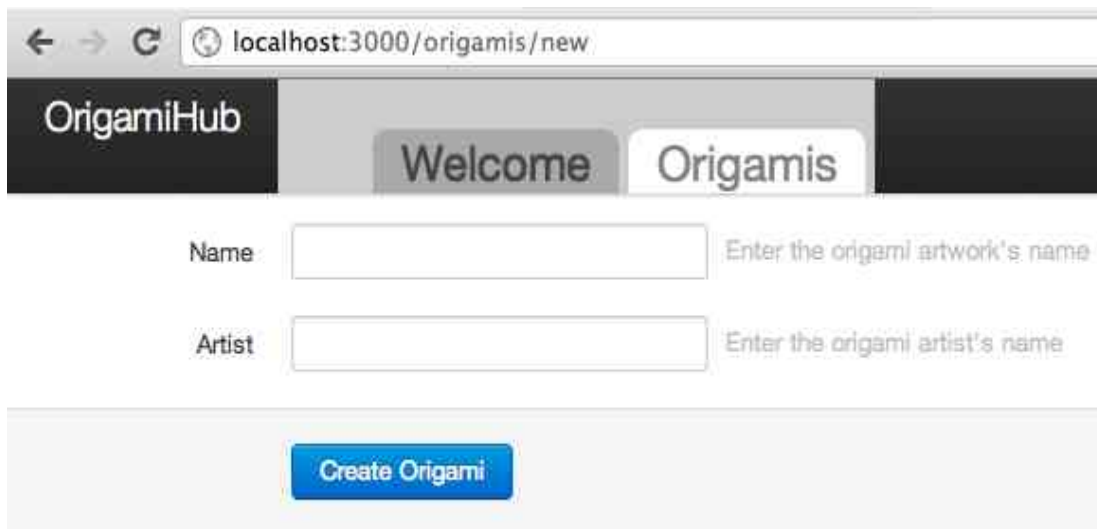
[view source](#)

[print?](#)

```
01 <body>
02 <div class="topbar">
03   <div class="topbar-inner">
04     <div class="container-fluid">
05       <a class="brand" href="#">OrigamiHub</a>
06       <%= tabs %>
07     </div>
08   </div>
09 </div>
10 <%= yield %>
```

Notice I replaced the `<ul>... </ul>` HTML from Twitter's example with a single call to `<%= tabs %>`. This will display the tabs that I've declared in the tab table above!

Now restarting my server again – remember I ran `bundle install` earlier – let's refresh the page:



[Back](#)

Twitter Bootstrap navigation bar displayed with Tabulous scaffold styles

Almost there... but this is still not quite right. By default, Tabulous will include some scaffolding tab CSS code to help you get started. However, since we're using Twitter Bootstrap we don't need any of that. To remove it, all we need to do is change the "scaffolding" setting to `false` in `app/tabs/tabulous.rb`:

[view source](#)



[print?](#)

```
1 config.css.scaffolding = false
```

We also need to change the "active\_tab\_clickable" setting to be true – this will tell Tabulous to always generate an `<a>` tag for each tab, which is what Twitter Bootstrap expects:

[view source](#)



[print?](#)

```
1 config.active_tab_clickable = true
```

Now let's refresh our page once more:

localhost:3000/origamis/new

OrigamiHub Welcome **Origamis**

Name:  Enter the origami artwork's name

Artist:  Enter the origami artist's name

Create Origami

[Back](#)

Tabulous tabs in Twitter Bootstrap navigation barbut missing body padding style

And it works! Now we see the tabs appear properly, and not only that, they actually work too! You can click on “Welcome” to see the Welcome index placeholder page from above, and on “Origamis” to get the Origami index page, since we entered those paths into the tab table in app/tabs/tabulous.rb.

There’s one other CSS bug here you may not have noticed: the “New Origami” text at the top of the form was obscured by the navigation bar – this is because the way Twitter’s CSS code works is that it assumes the HTML body will have padding on it. To fix this, we just need to manually add the 60px padding on the body style in our origami\_hub.css.scss file like this – be sure to put the padding style after the import or it won’t work:

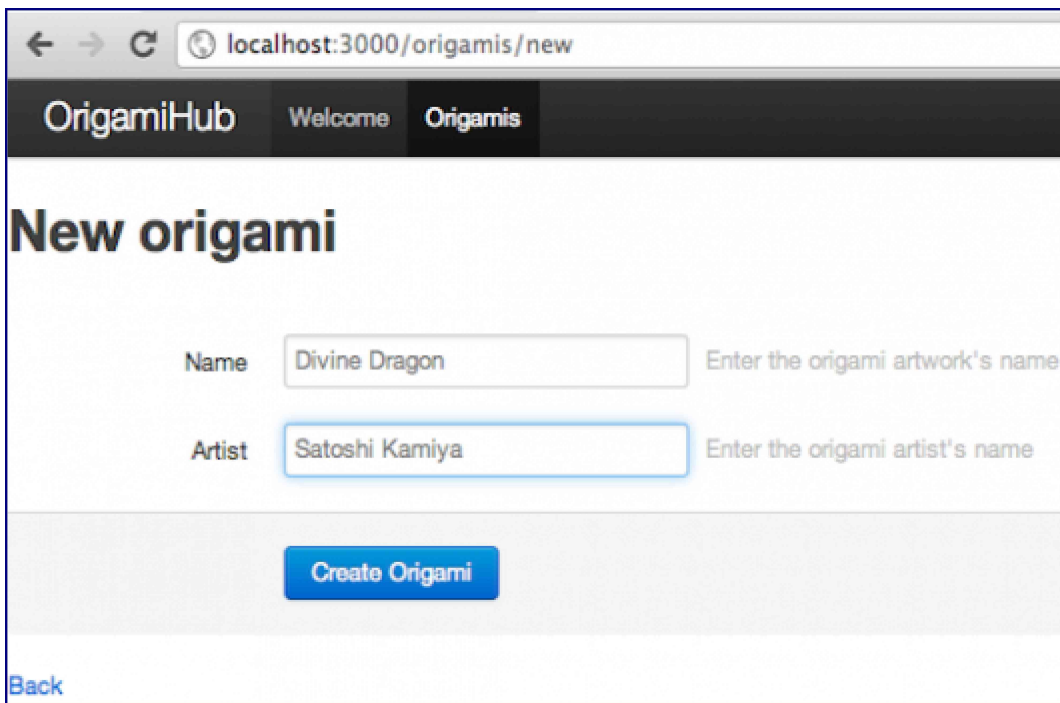
[view source](#)

[print?](#)

```
1 @import 'bootstrap';
2 body {
3   padding-top: 60px;
4 }
```

And refreshing the page once again we get:





Tabulous tabs in Twitter Bootstrap navigation bar with proper body padding

Wyatt's Tabulous gem will also automatically display the currently active tab properly, since Tabulous and Twitter Bootstrap both use the "active" CSS style to indicate that. It all just works! Now as I add more pages to my app, I just need to add entries in the tabs table, and I'm also free to re-order them.

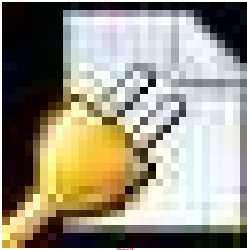
## Conclusion

Twitter Bootstrap is a fantastic way to build a nice looking web site very quickly; however, having a sophisticated CSS library will only take you so far. You still need to build a web application of some kind to bring the Twitter design elements to life. Using Rails on the back end can be a perfect complement to Twitter Bootstrap, but, as we've seen today, using Rails can also be a challenge since the two frameworks weren't designed to work together perfectly. Before you dive into building your next Rails 3.1 application with Twitter Bootstrap, be sure to do your homework – there are a lot of gem authors actively working in this area and as Twitter Bootstrap becomes more and more popular expect to see even more new gems and changes to existing gems.

Series Navigation << [Twitter Bootstrap, Less, and Sass: Understanding Your Options for Rails 3.1](#) [How to Customize Twitter Bootstrap's Design in a Rails app](#) >>

Tagged with: [twitter bootstrap](#)





## [Pat Shaughnessy](#)

Pat Shaughnessy is a Ruby developer working at a global management consulting firm. Pat also writes a weekly blog column about Ruby development at [patshaughnessy.net](http://patshaughnessy.net). Pat's tutorials and articles are geared towards Ruby beginners but contain enough information and detail to be interesting to more advanced developers also. Pat's articles and presentations have been featured multiple times on the Ruby Weekly newsletter, the Ruby5 podcast and the Ruby Show.

<http://patshaughnessy.net> [Twitter](#)

← [NET to Ruby: Learning How to Write Tests, Part II](#)

[Occasions: Giving Events a Backbone](#) →

## 18 Comments

1. [Gady](#) [14 Dec 11 | 10:08 am](#)



Excellent work!

Can't wait for the next twitter bootstrap articles !

[Reply](#)

2. [Alex](#) [14 Dec 11 | 5:23 pm](#)



and if you are a simple\_form fan then you can use:

[https://github.com/rafaelfranca/simple\\_form-bootstrap](https://github.com/rafaelfranca/simple_form-bootstrap)


[Reply](#)

• [Pat Shaughnessy](#) [14 Dec 11 | 6:07 pm](#)



Thanks for the link, Alex! I'll have to try that one out...

[Reply](#)

3. [Todor Grudev 15 Dec 11 | 12:34 am](#) 

pretty usefull thank you :)

[Reply](#)


4. [Antonio 15 Dec 11 | 3:49 pm](#) 

Hey, thanks for the great article.

I had to change the tabs table to root\_path instead of welcome\_index\_path though.

Antonio

[Reply](#)

- [Pat Shaughnessy 16 Dec 11 | 3:45 am](#) 

Yes, you're right welcome\_index\_path does not work. Earlier I had a different routes.rb file and forget to update that in the tabs table.

Article corrected now... Thanks a lot, Antonio!

[Reply](#)

5. [Leandro Facchinetti 16 Dec 11 | 11:32 am](#) 

Great article, Pat!

I found myself using bootstrap with rails before, but wasn't aware of all these neat tricks about integration with other gems that generate html.

I must warn that the last image, the one that shows the final product of the tutorial, is broken! There's nothing at <http://cdn.rubysource.com/files/2011/12/working-tabs-fixed.png>.

[Reply](#)

- [Pat Shaughnessy 16 Dec 11 | 2:28 pm](#) 

Thanks Leandro! We've fixed the image; let me know if you still have trouble seeing it.

[Reply](#)

6. [ouyang 20 Dec 11 | 8:28 pm](#) 

Thanks !

gem bootstrap\_helper :

[https://github.com/xdite/bootstrap\\_helper](https://github.com/xdite/bootstrap_helper)

[Reply](#)

7. [Anand 15 Jan 12 | 1:49 pm](#) 

Pat, I followed your instructions and things worked great, the only thing I am perplexed is why my form input area height is almost half as that as what you have on the example in this article.

Not sure how to go about fixing this.

Thank you

[Reply](#)

- [Anand 16 Jan 12 | 3:50 pm](#) 

My application.html.erb had , as per some other posts changing it to fixed this. Hope this help somebody.

[Reply](#)

- [Anand 16 Jan 12 | 3:52 pm](#) 

I changed my application.html.erb from "" to "" and this fixed the form input height problem.

[Reply](#)

8. [Antonio 27 Jan 12 | 3:00 pm](#) 

Hey Pat,

This works great in Rails 3.2 too :)

[Reply](#)

9. [Wyatt Greene 07 Feb 12 | 4:17 pm](#) 

I just released version 1.2.0 of the tabulous gem which now supports Twitter Bootstrap version 2. See the gem's README file for details on how to use with Twitter Bootstrap.

[Reply](#)

10. [Pavel Varela 17 Feb 12 | 12:16 pm](#) 

With rails -v 3.2.1 i all ends up completely messed up :(

[Reply](#)

11. [Ben Matthews 18 Feb 12 | 2:25 am](#) 

For those using Rails 3.2 and Twitter Bootstrap 2.0, Sam Pointer has published a great tutorial on updating the gems and code to make this example app work:

<http://blog.sam-pointer.com/2012/02/12/formtastic-bootstrap-with-rails-3-2-and-twitter-bootstrap-2>

[Reply](#)

- [Pat Shaughnessy 19 Feb 12 | 8:06 am](#) 

I just saw this myself - thanks Sam! Your update will be really helpful for people, since I wrote my articles at the worst possible time: just before Rails 3.2 and TB 2.0 were released, lol.

[Reply](#)

- [Sam Pointer 19 Feb 12 | 10:17 am](#) 

Hey Pat,

My pleasure - thanks for a great series. At the pace this stuff moves the window of "right time" is greatly diminishing. However, this is a very nice problem to have in some respects!

Thanks Ben for posting the reference to my post; I've extended the comment period on it.

[Reply](#)

class="heading"

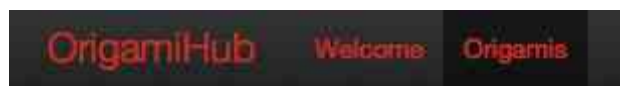
## How to Customize Twitter Bootstrap's Design in a Rails app

Reddit  1  237 Email  0 [Hackernews](#)

This entry is part 3 of 3 in the series [Twitter Bootstrap and Rails](#)

[Twitter Bootstrap and Rails](#)

- [Twitter Bootstrap, Less, and Sass: Understanding Your Options for Rails 3.1](#)
- [Too good to be true! Twitter Bootstrap meets Formtastic and Tabulous](#)
- How to Customize Twitter Bootstrap's Design in a Rails app



### New origami

Name

Artist

Create Origami

Maybe customizing Twitter Bootstrap's design like this wasn't such a good idea!

Back in November [I discussed various options](#) for integrating [Twitter Bootstrap](#) into a Rails 3.1 app, including using the [less-rails-bootstrap](#) and [bootstrap-sass](#) gems. Then last month [I wrote a follow up tutorial](#) showing how to quickly create a working web site using Twitter Bootstrap, Formtastic and Tabulous. Today I'd like to continue this series by discussing how to customize the Twitter Bootstrap design itself.

You might ask: Why customize Twitter Bootstrap at all? After all, their design looks great out of the box – why second guess their design decisions? Well, you might decide you like a certain color or font

better, or more likely you want to distinguish your web site from all the other sites that are using Twitter Bootstrap also. It's so easy to use that many developers have adopted it, and more and more web sites are beginning to have that same look and feel.

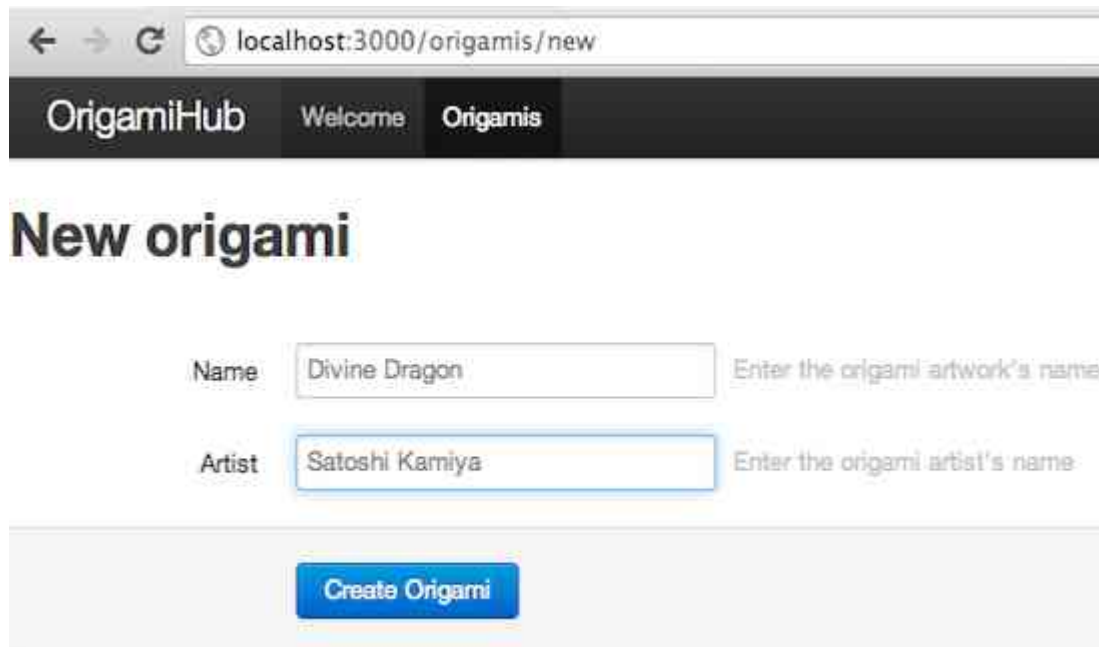
Or you might ask the opposite question: Why not just discard Twitter Bootstrap and write your own design that sets your site apart from all the others out there using Twitter Bootstrap? The reason is that Twitter Bootstrap provides a tremendous amount of useful CSS support that would take a long time to reimplement. Just look at all of the features on their [home page](#) – discarding all of these just to make your site look a little different would be a mistake.

One interesting option I came across recently is the concept of “Twitter Bootstrap themes,” which will allow you to pick from a series of different designs, while keeping all of the reset, typography, form, table and other styling support. While there are no themes right now, the Paris designer and entrepreneur [Sacha Greif](#) is currently working on the first theme to provide an alternative to Twitter's design, which he calls “Fuzzy.” You can [sign up here](#) to have Sacha send you an email when it's ready.

However, since Twitter Bootstrap themes are still not available, today I'm going to discuss the technical mechanics of how to go about customizing Twitter Bootstrap's Less and/or Sass code yourself – assuming you are or work with a good designer: Where is the Twitter code? How do I change their font and color settings? How do I make more substantial changes to their design? Read on to learn more...

## Customizing Twitter Bootstrap variables with Bootstrap-Sass

[Last month I showed how to build](#) a simple Rails 3.1 scaffolding site called “OrigamiHub” that ended up looking like this:



The screenshot shows a web browser window with the address bar displaying `localhost:3000/origamis/new`. The page has a dark navigation bar with the text "OrigamiHub", "Welcome", and "Origamis". Below the navigation bar, the main heading is "New origami". There are two input fields: "Name" with the value "Divine Dragon" and "Artist" with the value "Satoshi Kamiya". Each input field has a placeholder text: "Enter the origami artwork's name" and "Enter the origami artist's name" respectively. At the bottom of the form is a blue button labeled "Create Origami".

[Back](#)

Origami Hub app from last month

To build OrigamiHub I used a Ruby gem called [Bootstrap-Sass](#). I imported the Twitter code into the site by writing a file called `app/assets/stylesheets/origami_hub.css.sass` containing this code:

[view source](#)  
[print?](#)

```
1 @import 'bootstrap';
2 body {
3   padding-top: 60px;
4 }
```

Although I didn't explain it this way last month, this is already the simplest possible example of how to customize the Twitter design: the "padding-top" style overrides or adds to the standard body style Twitter chose. By declaring your styles after the "@import 'bootstrap'" line, you can override their CSS style code with your own styles. This is easy enough to understand: the browser processes all of the CSS code in the order it receives it; the code received towards the end will override the CSS styles declared earlier. By including your styles second, you can easily override anything Twitter's CSS code did.

However, the developers behind Twitter Bootstrap anticipated that designers may want to tweak their color, font or layout selections, and created a code file called "variables" containing commonly used global values. Since I used Bootstrap-Sass for OrigamiHub, this file will be called variables.css.scss, and I'll be able to find it as follows using Bundler:

[view source](#)  
[print?](#)

```
01 $ cd `bundle show bootstrap-sass`
02 $ find vendor/assets/stylesheets
03 vendor/assets/stylesheets
04 vendor/assets/stylesheets/bootstrap
05 vendor/assets/stylesheets/bootstrap/forms.css.scss
06 vendor/assets/stylesheets/bootstrap/mixins.css.scss
07 vendor/assets/stylesheets/bootstrap/patterns.css.scss
08 vendor/assets/stylesheets/bootstrap/reset.css.scss
09 vendor/assets/stylesheets/bootstrap/scaffolding.css.scss
10 vendor/assets/stylesheets/bootstrap/tables.css.scss
11 vendor/assets/stylesheets/bootstrap/type.css.scss
12 vendor/assets/stylesheets/bootstrap/variables.css.scss
13 vendor/assets/stylesheets/bootstrap.css.scss
```

See [November's post](#) for more details. Now let's take a look at what's inside variables.css.scss:

[view source](#)  
[print?](#)

```
01 /* Variables.less
02  * Variables to customize the look and feel of Bootstrap
03  * ----- */
04 // Links
05 $linkColor:          #0069d6 !default;
06 $linkColorHover:    darken($linkColor, 15) !default;
07 // Grays
```

```
08 $black:           #000 !default;
09 $grayDark:        lighten($black, 25%) !default;
10 $gray:            lighten($black, 50%) !default;
11 $grayLight:       lighten($black, 75%) !default;
12 ...etc...
```

It's not hard to figure out that by changing one of these values we can quickly change the appearance of the web site. However, it's not a good idea to edit this code directly since it's located inside the Bootstrap-Sass gem. Instead, as Thomas McDonald shows on the [Bootstrap-Sass Readme page](#), you can change one of these values by specifying the value you want BEFORE you import bootstrap, like this:

[view source](#)

[print?](#)

```
1 $linkColor: #FF69d6;
2 @import 'bootstrap';
3 body {
4   padding-top: 60px;
5 }
```

Now reloading the page I get a pink “back” link instead:



The screenshot shows a web form with two input fields labeled 'Name' and 'Artist'. Below the fields is a blue button with the text 'Create Origami'.

[Back](#)

Pink Back Link

If this doesn't work for you, be sure you have the latest version of Bootstrap-Sass, at least v1.4.1 or later:

[view source](#)



[print?](#)

```
1 $ bundle update bootstrap-sass
```

It turns out there's a subtle but important detail in the variables.css.scss file above – the odd use of “!default” at the end of each variable declaration:

[view source](#)



[print?](#)

```
1 $linkColor:           #0069d6 !default;
```

Here “!default” is a Sass language feature that means: if there was already a value declared for this



variable (“linkColor” in this example) then leave that unchanged and use it. If not, then use the specified value by default (“#0069d6” here). Practically speaking this means that you need to be sure to declare custom variable settings before and not after the “import bootstrap” line. This use of “!default” was added very recently to Bootstrap-Sass, so be sure to update the gem before trying to change a setting from variables.css.scss.

## Customizing Twitter Bootstrap variables with Less-Rails-Bootstrap

As I explained back in November, another option for including Twitter Bootstrap into a Rails app is to use the Less language instead of Sass, using the [Less-Rails-Bootstrap](#) gem. While not included in a Rails app by default like Sass, Less can be a good choice since it’s the language originally used by Twitter to developer Twitter Bootstrap.

If I had used Less-Rails-Bootstrap to build OrigamiHub, I would have a file called origami\_hub.css.less instead, with very similar code:

[view source](#)

[print?](#)

```
1 @import 'twitter/bootstrap';
2 body {
3   padding-top: 60px;
4 }
```

The only difference here is that the “import” command contains a different path to the Less code, since Less-Rails-Bootstrap uses a slightly different directory structure:

[view source](#)

[print?](#)

```
1 $ cd `bundle show less-rails-bootstrap`
2 $ find vendor/assets/stylesheets
3 vendor/assets/stylesheets
4 vendor/assets/stylesheets/twitter
5 vendor/assets/stylesheets/twitter/bootstrap.css.less
```

You can see only the top-level bootstrap.css.less file is located under vendor/assets/stylesheets. The other Less code files are here:

[view source](#)

[print?](#)

```
01 $ find vendor/frameworks
02 vendor/frameworks
03 vendor/frameworks/twitter
04 vendor/frameworks/twitter/bootstrap
05 vendor/frameworks/twitter/bootstrap/bootstrap.less
06 vendor/frameworks/twitter/bootstrap/forms.less
07 vendor/frameworks/twitter/bootstrap/mixins.less
08 vendor/frameworks/twitter/bootstrap/patterns.less
```

```
09 vendor/frameworks/twitter/bootstrap/reset.less
10 vendor/frameworks/twitter/bootstrap/scaffolding.less
11 vendor/frameworks/twitter/bootstrap/tables.less
12 vendor/frameworks/twitter/bootstrap/type.less
13 vendor/frameworks/twitter/bootstrap/variables.less
14 vendor/frameworks/twitter/bootstrap.less
```

Here we can see all the same code files, but using the “less” file extension instead. And you’ll find the same set of variables and values in variables.less that we saw above in variables.css.scss:

[view source](#)  
[print?](#)

```
1 // Links
2 @linkColor:          #0069d6;
3 @linkColorHover:    darken(@linkColor, 15);
4 // Grays
5 @black:              #000;
6 @grayDark:          lighten(@black, 25%);
7 @gray:               lighten(@black, 50%);
8 @grayLight:         lighten(@black, 75%);
9 ...etc...
```

Less uses a slightly different syntax than Sass: “@linkColor” instead of “\$linkColor”. Also we don’t see the “!default” directive we had earlier. Most importantly, changing the value of a variable in Less works differently than it does in Sass; there’s no concept of a default value and overriding it. In fact, in Less variables are actually implemented as constants. See [lesscss.org](http://lesscss.org) for more details. This means once you define a value for a variable it cannot be changed.

At first glance, this might mean that the only way to change the Twitter Bootstrap settings would be to edit the Twitter Less code directly, right inside of Less-Rails-Bootstrap. But this would be very ugly: every time I updated Less-Rails-Bootstrap I would lose my changes. While it might be possible to track my changes in a branch using git somehow, re-merging every time I got a newer version, this would be an obvious maintenance problem and an ongoing headache.

Actually, it turns out there’s a simpler way to do it. Because of a bug in the Less compiler, you can override the “constant” value of a variable by changing it after it is initially declared. That is, the value of the variable constant will be whatever its last assigned value is. It seems that the Less compiler first evaluates the values of all constants, and then evaluates the rest of the Less script, substituting the value for each variable.

What this means for customizing Less-Rails-Bootstrap is that you need to declare your custom variable values AFTER the import line, not before it like with Bootstrap-Sass. Here’s an example changing the gray scale colors to use a shade of red:

[view source](#)  
[print?](#)

```
1 @import 'twitter/bootstrap';
2 body {
3   padding-top: 60px;
```

4 }

5 @black: #200;

“#200” refers to a color that is not quite black, but has some red in it; remember “RGB” = “Red-Green-Blue.” Now my site looks like this:



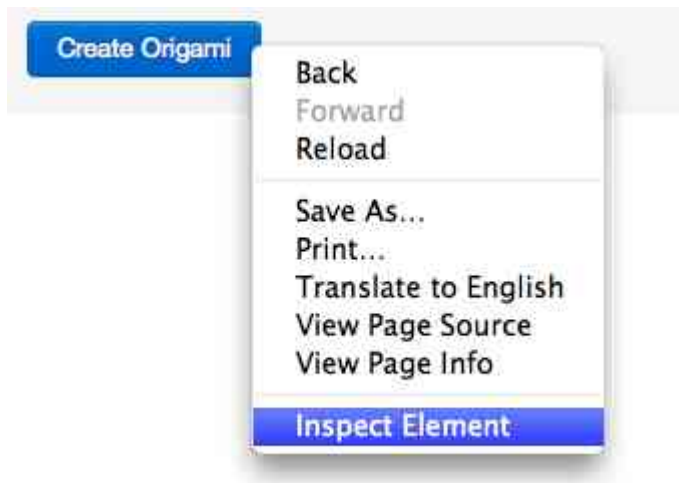
@black variable changed to be red

## Variables aren't enough!

Sadly, there just aren't enough customizable settings in the variables.less file – aside from linkColor, the only other settings there have to do with font size, colors and grid sizes. And as you can see from the screen shot above, the color settings don't actually even work properly, since many of the Twitter Less code files refer to hard coded color values directly: here the navigation bar is still black even though I've changed the value of “black” to be more red. Also, it's very odd to change the value of the “black” variable to be a shade of red in the first place... very confusing!

Unfortunately, the only good way to customize the Twitter Bootstrap design is to look closely at the HTML you are using in your site, find where the styles you are actually using are defined, and then override the Sass or Less code as needed.

Let's take an example: suppose my designer or I decided to make the “Create Origami” button a shade of red, instead of blue. The only effective way to do this would be to first use the Chrome “Inspect Element” command (or a similar command from your favorite browser) like this:



Chrome Inspect->Element right click menu item

... and then find the element's CSS style:

```
><div style="margin:0;padding:0;display:
  inline">...</div>
><fieldset class="inputs">...</fieldset>
▼<div class="actions">
  <input class="primary btn commit create"
    name="commit" type="submit" value="Create
    Origami">
  </div>
</form>
```

```
}
.btn.primary {
  color: white;
  background-color: #0064CD;
  background-repeat: repeat-x;
  background-image: khtml: linear g
  background-image: moz: linear g
  background-image: ms: linear gr
  background-image: webkit: gradi
```

Finding a style in the Chrome developer tools window

Here I can see that the button's blue background color is set by the "btn.primary" class. Searching through the Less code inside of Less-Rails-Bootstrap for this definition:

[view source](#)  
[print?](#)

```
01 $ cd `bundle show less-rails-bootstrap`
02 $ cd vendor/frameworks/twitter/bootstrap
03 $ ack btn
04 patterns.less
05 512:.btn,
06 545:// Base .btn styles
07 546:.btn {
08 619::root .btn {
09 624:button.btn,
10 625:input[type=submit].btn {
11 697: .btn {
12 871: .btn {
```

... I can see that the button related styles are defined in patterns.less. Looking through the file, I find the primary button style code:

[view source](#)

[print?](#)

```
01 // Base .btn styles
02 .btn {
03   // Button Base
04   cursor: pointer;
05   display: inline-block;
06   ... etc ...
07   // Primary Button Type
08   &.primary {
09     color: @white;
10     .gradientBar(@blue, @blueDark)
11   }
12   ... etc...
13 }
```

Here you can see how the primary button is defined: using a white font color and a gradient blue background (“blue” → “blueDark”). And you can also see the blue value is hard coded into the patterns.less file – there’s no variable in variables.less called “\$primaryButtonColor” or something similar, although there is a comment indicating this variable might be added soon.

To change the form buttons to be red instead, we need to copy/paste this Less code out into our application’s Less file (origami\_hub.css.less) and make the desired changes:

[view source](#)

[print?](#)

```
01 @import 'twitter/bootstrap';
02 body {
03   padding-top: 60px;
04 }
05 // Custom colors:
06 @black: #200;
07 @redDark: darken(@red, 5);
08 @linkColor: @redDark;
09 @titleColor: lighten(@red, 10);
10 // Copied from vendor/frameworks/twitter/bootstrap/patterns.less in
11 less-rails-bootstrap
12 .btn {
13   // Primary Button Type
14   &.primary {
15     color: @white;
16     .gradientBar(@red, @redDark)
17   }
18 }
```

```

18 .topbar {
19   // Hover and active states
20   ul .active &gt; a {
21     color: @titleColor;
22   }
23   a {
24     color: @titleColor;
25   }
26   // Website name
27   .brand {
28     color: @titleColor;
29   }
30 }

```

Here I've defined a couple of new shades of red, and used them to override a few other style definitions related to the navigation bar text, along with the primary button style, leading to a more red version of my site:

The screenshot shows a dark navigation bar with the text 'OrigamiHub', 'Welcome', and 'Origamis' in a red color. Below the navigation bar is a large red heading 'New origami'. Underneath the heading are two input fields: 'Name' and 'Artist', both with placeholder text 'Enter the origami name'. At the bottom of the form is a red button labeled 'Create Origami'.

[Back](#)

OrigamiHub with a customized version of the Twitter Bootstrap design

## Conclusion

This seems like a lot of work to change something from blue to red! Why bother? Or why not just discard Twitter Bootstrap entirely and start from scratch?

The reason is obvious: if you browse for a few minutes around the Twitter Bootstrap Less or Sass code files in one of the gems I've mentioned here, you'll see a tremendous number of useful styles and features. Here are just a few examples:

- `reset.less` – contains the reset styles to create a clean, cross-browser foundation to use, based on

[Eric Meyer's work from 2007.](#)

- mixins.less – contains a series of useful functions that can be reused by other styles.
- type.less – useful typography utilities.
- etc..., etc...

And that's just the beginning; there are a number of Twitter Bootstrap features I haven't even mentioned here. Plus a newer version of Twitter Bootstrap, v2.0, is underway and will soon add even more Less/Sass code files and CSS features to the list.

The way I like to think about Twitter Bootstrap is that it's a CSS coding platform. The same way that Rails makes it a lot easier to build a web site by implementing commonly needed functionality that all web sites need, Twitter Bootstrap, on a much smaller scale, implements many of the style features that any web site would need. The problem is that to use it effectively you really do need to take the time to learn how the Less, or translated Sass, code works, since there are many hard coded design details. To use it with a custom, unique design that you or a designer you are working with has written will require you to manually copy, paste and customize the portions of Twitter Bootstrap you want to change. Hopefully this process will become easier in future versions, as they add more variables and other ways to customize their design!

Series Navigation << [Too good to be true! Twitter Bootstrap meets Formtastic and Tabulous](#)

Tagged with: [twitter bootstrap](#)



## Pat Shaughnessy

Pat Shaughnessy is a Ruby developer working at a global management consulting firm. Pat also writes a weekly blog column about Ruby development at [patshaughnessy.net](http://patshaughnessy.net). Pat's tutorials and articles are geared towards Ruby beginners but contain enough information and detail to be interesting to more advanced developers also. Pat's articles and presentations have been featured multiple times on the Ruby Weekly newsletter, the Ruby5 podcast and the Ruby Show.

<http://patshaughnessy.net> [Twitter](#)

← [Loccasions: Getting to Occasions](#)

[Smelly Cucumbers](#) →

## 5 Comments

1. [Simon Hamp 11 Jan 12 | 1:11 pm](#) 

Great tutorial, Pat! Just want to let you and others know that anything they build that uses even parts of Bootstrap can get a slot on Built With Bootstrap (<http://builtwithbootstrap.tumblr.com/>), our little showcase. Just submit a short, descriptive post with a screenshot :)

[Reply](#)

- [Pat Shaughnessy 11 Jan 12 | 3:26 pm](#) 

Cool idea Simon! I already told one of my developer friends to submit his site there :)

[Reply](#)

2. [Oscar 11 Jan 12 | 7:12 pm](#) 

Awesome post pat! Ive been playing a bit with bootstrap and I can see why developers are loving it. Will have to apply your less tutorial on it. Thanks!

[Reply](#)

3. [Bob Walsh 26 Jan 12 | 2:56 pm](#) 

Fantastic series of well-writtern articles, but now I have to choose:

With Twitter-bootstrap 2.0 out in a week, if you were starting a Rails 3.2 project today, would you a) less-rails-bootstrap or b) the bootstrap-sass gem by Thomas McDonald?

[Reply](#)

- [Pat Shaughnessy 27 Jan 12 | 3:24 pm](#) 

Hi Bob, Thanks!

First of all, I'm not sure either of those two gems have been updated with TB 2.0 yet - although I expect both of them would be rapidly once the new version of bootstrap is released.

If you have a preference for Less vs. Sass, then choose based on that. TB was originally written in Less, but I also trust that Thomas will translate it to Sass properly.

Finally if you have no other reason to choose one way or the other, I might go with



bootstrap-sass just because Rails already supports Sass (.scss files) by default in the asset pipeline, while it does not support Less. I assume that's still true in Rails 3.2.

But really they are both excellent gems and either will do the job for you.

[Reply](#)

**Leave a Reply**