tweaking octopress

Peter Humburg

Published at *Genomic Campfire* on Feb 13 2015

Contents

This is the second post an a short series describing my experience with setting up Octopress for this blog. The first post, dealing with installation and basic customisation, can be found here.

Once I started writing posts for the blog I noticed a few things that I thought could use some tweaking. This post documents several changes that can be achieved without too much work. I have also made one or two larger changes that probably warrant their own posts.

Descriptions and keywords

Octopress can add *description* and *keywords* meta tags to the header of generated pages, but support is somewhat half-hearted in the default theme[1]. If you want this, you'll have to add *description* and *keywords* fields to every post. To simplify this process I modified the `new_post` rake task. The `Rakefile` in the root directory of the Octopress installation contains all the rake tasks for Octopress. The task for generating new posts contains the following code to produce the YAML header:

```
  open(filename, 'w') do |post|
    post.puts "---"
    post.puts "layout: post"
    post.puts "title: \"#{title.gsub(/&/,'&amp;')}\""
    post.puts "date: #{Time.now.strftime('%Y-%m-%d %H:%M:%S %z')}"
    post.puts "comments: true"
    post.puts "categories: "
    post.puts "---"
  end
```

Simply add

```
    post.puts "description: "
    post.puts "keywords: "
```

after the *categories* entry. The *foxslide* theme adds all of this, except the site wide keywords, to the generated pages. If you use the default theme (or another theme that has less support for this) you may need to make some changes to `source/_includes/head.html`. There are instructions on how to do this at the sweatmeat blog. I only added a short piece of code to get keywords from `_config.yml` added to the blog's index page. This goes somewhere close to line that adds *page.keywords*.

```
{% if index %}
    {% if site.keywords  %}<meta name="keywords" content="{{ site.keywords}}">{% endif %}
{% endif %}
```

Rename posts on publication

Next, I wanted the posts generated by the `new_post` rake task to be marked as drafts so that they don't get published prematurely. For that we go back to the code that generates the meta data block and add *published: false* to the output.

One more thing that seems helpful is to automatically adjust file names for posts when they are published to include the current date[2]. This isn't strictly necessary because the date inside the meta data block overrides the date in the file name anyway, but it makes sense to keep things consistent. Following the description on ewal.net I added the following rake task to the Octopress `Rakefile`.

```
desc "Rename files in the posts directory if the filename does not match the post date in the YAML fron
task :rename_posts do
  Dir.chdir("#{source_dir}/#{posts_dir}") do
    Dir['*.markdown'].each do |post|
```

[1] The *foxslide* theme I use has improved support for these, making my life a bit easier.
[2] thanks to Ewal.net for the idea

```
      post_date = ""
      File.open( post ) do |f|
        f.grep( /^date: / ) do |line|
          post_date = line.gsub(/date: /, "").gsub(/\s.*$/, "")
          break
        end
      end
      # Get the post title from the currently processed post
      post_title = post.to_s.gsub(/\d{4}-\d{2}-\d{2}/, "")
      # determine the correct filename
      new_post_name = post_date + post_title
      is_draft = false
      File.open( post ) do |f|
          f.grep( /^published: false/ ) do |line|
            is_draft = true
            break
          end
      end
      if !is_draft && post != new_post_name
          puts "renaming #{post} to #{new_post_name}"
          FileUtils.mv(post, new_post_name)
      end
    end
  end
end
```

Now running `rake rename_posts` will check all published posts and adjust their file name if it is inconsistent with the date listed in the YAML block. If you want you can add this to the list of tasks executed by `rake gen_deploy` to further automate the process.

Adding a category list

I expect to publish posts in a number of categories and it would be nice to have links to the category index pages. That will hopefully help readers to find the posts they are interested in. I'm using the category list plugin by alswl. The `category_list.rb` script can generate plain lists as well as tag clouds. I opted for a simple list in the footer of the page (next to the recent post list). When using the default theme you should add one of the custom asides that come with the plugin to the *default_asides* list in `_config.yml`. With the *foxslide* theme I actually had to modify `source/source/_includes/custom/footer_widgets.html`, adding

```
<div class="span3">
  <h2>categories</h2>
  <ul id="category-list">{% category_list counter:true %}</ul>
</div>
```

in an appropriate place.

Better category pages

The default category pages generated by Octopress, located at */blog/categories/(category)* are rather bland. It would be much nicer if they looked more like the front page with previews of the posts and pagination. One approach to achieve this has been described by Prateek Gianchandani. This basically involves creating a copy of the index page, restricting the listing of posts to a single category and copying it to a directory with the same name. The result of this is that much nicer category pages are available at */(category)*. I started with the same process but made some adjustments to suit my needs.

Most importantly, I don't want to be forced to create category pages manually whenever I use a new category for the first time. The first step in automating this is to create a template that will form the basis of all category pages (in `source/_templates/category.html`).

```
<div id="category-wrapper">
  <div class="container">
    <div class="row" id="post-container">
    {% assign index = true %}
    {% for post in site.categories._CATEGORY_ %}
    {% assign content = post.content %}
      <article class="span4">
        {% include article.html %}
      </article>
    {% endfor %}
    </div>
      <div class="row">
        <ul class="pager">
          {% if paginator.next_page %}
            <li class="previous">
              <a href="{{paginator.next_page}}">&larr; Older</a>
            </li>
          {% endif %}
          {% if paginator.previous_page %}
            <li class="next">
              <a class="next" href="{{paginator.previous_page}}">Newer &rarr;</a>
            </li>
          {% endif %}
        </ul>
      </div>
  </div>
</div>
```

To create a new category page from this template it is copied to `source/categories/<category>/index.html` together with an appropriate meta data block. All this is accomplished by a new rake task I defined:

```
desc "Create a new category page in #{source_dir}/categories/(filename)/index.html"
task :new_category, [:filename, :force] do |t, args|
  raise "### You haven't set anything up yet. First run `rake install` to set up an Octopress theme." u
  args.with_defaults(:force => false)
  category = args.filename
  page_dir = [source_dir, "categories", category]
  title = "Category: #{category}"
  filename = "index"
  extension = "html"
  page_dir = page_dir.map! { |d| d = d.to_url }.join('/')                # Sanitize path
  mkdir_p page_dir
  file = "#{page_dir}/#{filename}.#{extension}"
  if !args.force and File.exist?(file)
    abort("rake aborted!") if ask("#{file} already exists. Do you want to overwrite?", ['y', 'n']) == '
  end
  puts "Creating category page: #{file}"
  open(file, 'w') do |page|
    page.puts "---"
    page.puts "layout: category"
    page.puts "title: \"#{title}\""
    page.puts "sharing: false"
    page.puts "footer: true"
    page.puts "category: #{category}"
    page.puts "---"
    open("source/_templates/category.html", "r+") do |template|
      template.each_line do |line|
        line = line.gsub! "_CATEGORY_", "#{category}" if line =~ /_CATEGORY_/
        page.puts line
      end
```

```
    end
  end
end
```

This reduces the manual intervention required when a new category is introduced to calling an additional rake task. That is better than having to mess with files directly but I really would prefer if this happened automatically. Here is another task to facilitate that.

```
desc "Create index pages for all categories."
task :categories, :force do |t, args|
  raise "### You haven't set anything up yet. First run `rake install` to set up an Octopress theme." u
  raise "### No categories found. Try running 'rake generate' first." unless File.directory?("#{public_
  args.with_defaults(:force => true)
  categories = Dir["#{public_dir}/blog/categories/*/"].map { |d| File.basename(d) }
  categories.each do |cat|
    Rake::Task[:new_category].invoke("#{cat}", args.force)
    Rake::Task[:new_category].reenable
  end
end
```

This task inspects the category pages generated by Octopress to obtain a list of all categories and then creates custom category pages as described above. This can be added to the list of tasks executed by *gen_deploy* to ensure that category pages are up to date.

To ensure that readers of the blog get to see the improved index pages for each category I modified the category list plugin such that it generates links to the new pages. All that is left to do now is to style the new index pages. I'm using a custom layout based on the standard *page* layout. The only differences are that I changed the width of the content area and removed comments.

```
7c7
<       <article class="span8 offset2 article-format" role="article">
---
>       <article class="span12 article-format" role="category">
26,31d25
<       {% if site.disqus_short_name and page.comments == true %}
<         <section>
<           <h1>Comments</h1>
<           <div id="disqus_thread" aria-live="polite">{% include post/disqus_thread.html %}</div>
<         </section>
<       {% endif %}
```

Gravatar support

You may want to add your Gravatar image to your blog, either on the 'about' page or as part of an 'about' aside. Joe Nicosia has a plugin for that purpose on GitHub. Using that plugin I added the following code to `sources/_includes/custom/asides/about.html` to add my Gravatar to the 'about' aside of the *foxslide* theme.

```
{% if site.gravatar_email %}
<span id=gravatar>
        <img src="{% gravatar_image %}" alt="Gravatar of {{ site.author}} " title="Gravatar of {{ site.
</span>
{% endif %}
```

The only other things that are needed are a *gravatar_email* entry in `_config.yml` and a bit of CSS in `sass/custom/_styles.scss`. I chose to display a circular version of the Gravatar as a small inset at the start of the 'about' text.

```
#gravatar img {
  border-radius: 50%;
  float: left;
  margin-right: 10px;
  margin-top: 50px;
  margin-bottom: 0px;
}
```

Tell the world!

Now that the blog is up and running it is time to spread the word. Apart from announcing the existence of the blog through social media we can notify various search engines. Quite a few blog related ones can be reached through Ping-O-Matic. Since this needs to be repeated after each new post is published it is a good idea to automate the process with a rake task. The following code (again thanks to Ewal.net) accomplishes this..

```
desc 'Ping pingomatic'
task :pingomatic do
  begin
    require 'xmlrpc/client'
    puts '* Pinging ping-o-matic'
    XMLRPC::Client.new('rpc.pingomatic.com', '/').call('weblogUpdates.extendedPing', 'Genomic Campfire'
  rescue LoadError
    puts '! Could not ping ping-o-matic, because XMLRPC::Client could not be found.'
  end
end
```

Of course you'll have to customise this with your own blog's details. This doesn't include Google or Bing. To reach those as well we use two more rake tasks that send a notification about the updated site map.

```
desc 'Notify Google of the new sitemap'
task :sitemapgoogle do
  begin
    require 'net/http'
    require 'uri'
    puts '* Pinging Google about our sitemap'
    Net::HTTP.get('www.google.com', '/webmasters/tools/ping?sitemap=' + URI.escape('http://humburg.gith
  rescue LoadError
    puts '! Could not ping Google about our sitemap, because Net::HTTP or URI could not be found.'
  end
end

desc 'Notify Bing of the new sitemap'
task :sitemapbing do
  begin
    require 'net/http'
    require 'uri'
    puts '* Pinging Bing about our sitemap'
    Net::HTTP.get('www.bing.com', '/webmaster/ping.aspx?siteMap=' + URI.escape('http://humburg.github.i
  rescue LoadError
    puts '! Could not ping Bing about our sitemap, because Net::HTTP or URI could not be found.'
  end
end
```

Finally, we combine all three of these into a single task.

```
desc "Notify various services about new content"
task :notify => [:pingomatic, :sitemapgoogle, :sitemapbing] do
end
```

Conclusion

With the modifications described in this and the previous post in place the blog largely functions the way I want it to. In addition to the modifications described here I also switched the Markdown engine to pandoc. A separate post about this is in the pipeline. I'm also working on improvements to the GitHub plugin, i.e. a better list of GitHub repositories to display as an aside.

With all this in mind, how do I feel about my experience with Octopress so far? Getting a blog with the default theme up and running was pretty straightforward. The support for custom themes is good, but of course the quality of individual themes varies. How much work is required to get all the details sorted depends a lot on how well the theme fits your requirements and how particular you are about getting everything to work just so. I enjoy tweaking all the details until they are just right but have limited time available to do this sort of thing. Overall Octopress provided a fairly good balance of things just working and the ability to change the details. I even learned some *Ruby* in the process.