Introduction

What is Jekyll?

Jekyll is a parsing engine bundled as a ruby gem used to build static websites from dynamic components such as templates, partials, liquid code, markdown, etc. Jekyll is known as "a simple, blog aware, static site generator".

Example Jekyll websites (https://github.com/mojombo/jekyll/wiki/Sites).

What does Jekyll do?

Jekyll is installed as a ruby gem local computer. Once installed you can call `jekyll serve` in the terminal in a directory and provided that directory is setup in a way jekyll expects, it will do magic stuff like parse markdown/textile files, compute categories, tags, permalinks, and construct your pages from layout templates and partials.

Once parsed, Jekyll stores the result in a self-contained static `_site` folder. The intention here is that you can serve all contents in this folder statically from a plain static web-server.

You can think of Jekyll as a normalish dynamic blog but rather than parsing content, templates, and tags on each request, Jekyll does this once *beforehand* and caches the *entire website* in a folder for serving statically.

Jekyll is Not Blogging Software.

Jekyll is a parsing engine.

Jekyll does not come with any content nor does it have any templates or design elements. This is a common source of confusion when getting started. Jekyll does not come with anything you actually use or see on your website - you have to make it.

Why Should I Care?

Jekyll is very minimalistic and very efficient. The most important thing to realize about Jekyll is that it creates a static representation of your website requiring only a static web-server. Traditional dynamic blogs like Wordpress require a database and server-side code. Heavily trafficked dynamic blogs must employ a caching layer that ultimately performs the same job Jekyll sets out to do; serve static content.

Therefore if you like to keep things simple and you prefer the command-line over an admin panel UI then give Jekyll a try.

Developers like Jekyll because we can write content like we write code:

- Ability to write content in markdown or textile in your favorite text-editor.
- Ability to write and preview your content via localhost.
- No internet connection required.
- Ability to publish via git.
- Ability to host your blog on a static web-server.
- Ability to host freely on GitHub Pages.
- No database required.

How Jekyll Works

> **Heads up!** The following is a complete but concise outline of exactly how Jekyll works. Core concepts are introduced in rapid succession without code examples. This information is not intended to specifically teach you how to do anything, rather it is intended to give you the *full picture* relative to what is going on in Jekyll-world. Learning these core concepts should help you avoid common frustrations and ultimately help you better understand the code examples contained throughout Jekyll-Bootstrap.

Initial Setup

After installing jekyll (http://jekyllrb.com/docs/installation/) you'll need to format your website directory in a way jekyll expects. Jekyll-bootstrap conveniently provides the base directory format.

The Jekyll Application Base Format

Jekyll expects your website directory to be laid out like so:

- **_config.yml**
- **_includes**
- **_layouts**
  - default.html
  - post.html
- **_posts**
  - 20011-10-25-open-source-is-good.markdown
  - 20011-04-26-hello-world.markdown
- **_site**
- **index.html**
- **assets**
  - css
    - style.css
  - javascripts

_config.yml Stores configuration data.

_includes This folder is for partial views.

_layouts This folder is for the main templates your content will be inserted into. You can have different layouts for different pages or page sections.

_posts This folder contains your dynamic content/posts. the naming format is required to be @YEAR-MONTH-DATE-title.MARKUP@ .

_site This is where the generated site will be placed once Jekyll is done transforming it.

assets This folder is not part of the standard jekyll structure. The assets folder represents *any generic* folder you happen to create in your root directory. Directories and files not properly formatted for jekyll will be left untouched for you to serve normally.

(read more: http://jekyllrb.com/docs/usage/ (http://jekyllrb.com/docs/usage/))

Jekyll Configuration

Jekyll supports various configuration options that are fully outlined here: http://jekyllrb.com/docs/configuration/ (http://jekyllrb.com/docs/configuration/)

Content in Jekyll

Content in Jekyll is either a post or a page. These content "objects" get inserted into one or more templates to build the final output for its respective static-page.

Posts and Pages

Both posts and pages should be written in markdown, textile, or HTML and may also contain Liquid templating syntax. Both posts and pages can have meta-data assigned on a per-page basis such as title, url path, as well as

arbitrary custom meta-data.

Working With Posts

Creating a Post

Posts are created by properly formatting a file and placing it the `_posts` folder.

Formatting

A post must have a valid filename in the form `YEAR-MONTH-DATE-title.MARKUP` and be placed in the `_posts` directory. If the data format is invalid Jekyll will not recognize the file as a post. The date and title are automatically parsed from the filename of the post file. Additionally, each file must have YAML Front-Matter (http://jekyllrb.com/docs/frontmatter/) prepended to its content. YAML Front-Matter is a valid YAML syntax specifying meta-data for the given file.

Order

Ordering is an important part of Jekyll but it is hard to specify a custom ordering strategy. Only reverse chronological and chronological ordering is supported in Jekyll.

Since the date is hard-coded into the filename format, to change the order, you must change the dates in the filenames.

Tags

Posts can have tags associated with them as part of their meta-data. Tags may be placed on posts by providing them in the post's YAML front matter. You have access to the post-specific tags in the templates. These tags also get added to the sitewide collection.

Categories

Posts may be categorized by providing one or more categories in the YAML front matter. Categories offer more significance over tags in that they can be reflected in the URL path to the given post. Note categories in Jekyll work in a specific way. If you define more than one category you are defining a category hierarchy "set". Example:

```
1. ---
2. title : Hello World
3. categories : [lessons, beginner]
4. ---
```

This defines the category hierarchy "lessons/beginner". Note this is *one category* node in Jekyll. You won't find "lessons" and "beginner" as two separate categories unless you define them elsewhere as singular categories.

Working With Pages

Creating a Page

Pages are created by properly formatting a file and placing it anywhere in the root directory or subdirectories that do *not* start with an underscore.

Formatting

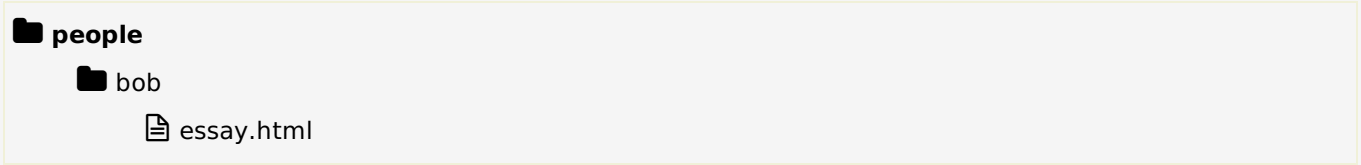
In order to register as a Jekyll page the file must contain YAML Front-Matter (http://jekyllrb.com/docs/frontmatter/). Registering a page means 1) that Jekyll will process the page and 2) that the page object will be available in the `site.pages` array for inclusion into your templates.

Categories and Tags

Pages do not compute categories nor tags so defining them will have no effect.

Sub-Directories

If pages are defined in sub-directories, the path to the page will be reflected in the url. Example:



This page will be available at `http://yourdomain.com/people/bob/essay.html`

Recommended Pages

- **index.html**

- **index.html**
  You will always want to define the root index.html page as this will display on your root URL.
- **404.html**
  Create a root 404.html page and GitHub Pages will serve it as your 404 response.
- **sitemap.html**
  Generating a sitemap is good practice for SEO.
- **about.html**
  A nice about page is easy to do and gives the human perspective to your website.

Templates in Jekyll

Templates are used to contain a page's or post's content. All templates have access to a global site object variable: `site` as well as a page object variable: `page` . The site variable holds all accessible content and metadata relative to the site. The page variable holds accessible data for the given page or post being rendered at that point.

Create a Template
Templates are created by properly formatting a file and placing it in the `_layouts` directory.

Formatting
Templates should be coded in HTML and contain YAML Front Matter. All templates can contain Liquid code to work with your site's data.

Rending Page/Post Content in a Template
There is a special variable in all templates named : `content` . The `content` variable holds the page/post content including any sub-template content previously defined. Render the content variable wherever you want your main content to be injected into your template:

```
<body>
  <div id="sidebar"> ... </div>
  <div id="main">
    {{content}}
  </div>
</body>
```

Sub-Templates

Sub-templates are exactly templates with the only difference being they define another "root" layout/template within their YAML Front Matter. This essentially means a template will render inside of another template.

Includes

In Jekyll you can define include files by placing them in the `_includes` folder. Includes are NOT templates, rather they are just code snippets that get included into templates. In this way, you can treat the code inside includes as if it was native to the parent template.

Any valid template code may be used in includes.

Using Liquid for Templating

Templating is perhaps the most confusing and frustrating part of Jekyll. This is mainly due to the fact that Jekyll templates must use the Liquid Templating Language.

What is Liquid?

Liquid (https://github.com/Shopify/liquid) is a secure templating language developed by Shopify (http://shopify.com). Liquid is designed for end-users to be able to execute logic within template files without imposing any security risk on the hosting server.

Jekyll uses Liquid to generate the post content within the final page layout structure and as the primary interface for working with your site and post/page data.

Why Do We Have to Use Liquid?

GitHub uses Jekyll to power GitHub Pages (http://pages.github.com/). GitHub cannot afford to run arbitrary code on their servers so they lock developers down via Liquid.

Liquid is Not Programmer-Friendly.

The short story is liquid is not real code and its not intended to execute real code. The point being you can't do jackshit in liquid that hasn't been allowed explicitly by the implementation. What's more you can only access data-structures that have been explicitly passed to the template.

In Jekyll's case it is not possible to alter what is passed to Liquid without hacking the gem or running custom plugins. Both of which cannot be supported by GitHub Pages.

As a programmer - this is very frustrating.

But rather than look a gift horse in the mouth we are going to suck it up and view it as an opportunity to work around limitations and adopt client-side solutions when possible.

> My personal stance is to not invest time trying to hack liquid. It's really unnecessary *from a programmer's* perspective. That is to say if you have the ability to run custom plugins (i.e. run arbitrary ruby code) you are better off sticking with ruby. Toward that end I've built Mustache-with-Jekyll (http://github.com/plusjade/mustache-with-jekyll) which is now abandoned =/. You should use http://ruhoh.com (http://ruhoh.com) instead =D.

Static Assets

Static assets are any file in the root or non-underscored subfolders that are not pages. That is they have no valid YAML Front Matter and are thus not treated as Jekyll Pages. Static assets should be used for images, css, and javascript files.

How Jekyll Parses Files

Remember Jekyll is a processing engine. There are two main types of parsing in Jekyll.

- **Content parsing.**
  This is done with textile or markdown.
- **Template parsing.**
  This is done with the liquid templating language.

And thus there are two main types of file formats needed for this parsing.

- **Post and Page files.**
  All content in Jekyll is either a post or a page so valid posts and pages are parsed with markdown or textile.
- **Template files.**
  These files go in `_layouts` folder and contain your blogs **templates**. They should be made in HTML with the help of Liquid syntax. Since include files are simply injected into templates they are essentially parsed as if they were native to the template.

Arbitrary files and folders.
Files that *are not* valid pages are treated as static content and pass through Jekyll untouched and reside on your blog in the exact structure and format they originally existed in.

Formatting Files for Parsing.

We've outlined the need for valid formatting using **YAML Front Matter**. Templates, posts, and pages all need to provide valid YAML Front Matter even if the Matter is empty. This is the only way Jekyll knows you want the file processed.

YAML Front Matter must be prepended to the top of template/post/page files:

```
---
layout: post
category : pages
tags : [how-to, jekyll]
---

... contents ...
```

Three hyphens on a new line start the Front Matter block and three hyphens on a new line end the block. The

Three hyphens on a new line start the Front-Matter block and three hyphens on a new line end the block. The data inside the block must be valid YAML.

Configuration parameters for YAML Front-Matter is outlined here: A comprehensive explanation of YAML Front Matter (http://jekyllrb.com/docs/frontmatter/)

Defining Layouts for Posts and Templates Parsing.

The `layout` parameter in the YAML Front Matter defines the template file for which the given post or template should be injected into. If a template file specifies a layout parameter, it is effectively being used as a `sub-template`. That is to say loading a post file into a template file that refers to another template file will work in the way you'd expect; as a nested sub-template.

How Jekyll Generates the Final Static Files.

Ultimately, Jekyll's job is to generate a static representation of your website. The following is an outline of how that's done:

1. **Jekyll collects data.**
   Jekyll scans the posts directory and collects all posts files as post objects. It then scans the layout assets and collects those and finally scans other directories in search of pages.
2. **Jekyll computes data.**
   Jekyll takes these objects, computes metadata (permalinks, tags, categories, titles, dates) from them and constructs one big `site` object that holds all the posts, pages, layouts, and respective metadata. At this stage your site is one big computed ruby object.
3. **Jekyll liquifies posts and templates.**
   Next jekyll loops through each post file and converts (through markdown or textile) and **liquifies** the post inside of its respective layout(s). Once the post is parsed and liquified inside the the proper layout structure, the layout itself is "liquified".
   Liquification is defined as follows: Jekyll initiates a Liquid template, and passes a simpler hash representation of the ruby site object as well as a simpler hash representation of the ruby post object. These simplified data structures are what you have access to in the templates.
4. **Jekyll generates output.**
   Finally the liquid templates are "rendered", thereby processing any liquid syntax provided in the templates and saving the final, static representation of the file.

Notes.
Because Jekyll computes the entire site in one fell swoop, each template is given access to a global `site` hash that contains useful data. It is this data that you'll iterate through and format using the Liquid tags and filters in order to render it onto a given page.

Remember, in Jekyll you are an end-user. Your API has only two components:

1. The manner in which you setup your directory.
2. The liquid syntax and variables passed into the liquid templates.

All the data objects available to you in the templates via Liquid are outlined in the **API Section** of Jekyll-Bootstrap. You can also read the original documentation here: http://jekyllrb.com/docs/variables/ (http://jekyllrb.com/docs/variables/)

Conclusion

I hope this paints a clearer picture of what Jekyll is doing and why it works the way it does. As noted, our main programming constraint is the fact that our API is limited to what is accessible via Liquid and Liquid only.

Jekyll-bootstrap is intended to provide helper methods and strategies aimed at making it more intuitive and easier to work with Jekyll =)

Thank you for reading this far.

Next Step: Jekyll Bootstrap API → (/api/bootstrap-api.html)