Extracted from:

Build Websites with Hugo

Fast Web Development with Markdown

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Build Websites with Hugo

Fast Web Development with Markdown

Brian P. Hogan

edited by Tammy Coron

Build Websites with Hugo

Fast Web Development with Markdown

Brian P. Hogan

The Pragmatic Bookshelf

Raleigh, North Carolina

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Tammy Coron
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

ISBN-13: 978-1-68050-726-3
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—May 2020

CHAPTER 5

Adding a Blog

When you want to share your thoughts with the world, using your own site is one of the best ways to do it. While there are platforms you can use to do this, choosing to instead host from your own domain offers certain benefits. You can measure its effects, control how the information is presented, and more importantly, build your own brand with content you own. When you publish content elsewhere, you have to opt in to their terms of service, and sometimes hand over control over your content and community.

To simplify the process of hosting your own content, you can use Hugo to add a blog to your site. To create a blog in Hugo, you'll create a new content type, named "Post", and you'll create layouts to support displaying those posts. A lot of what you'll do in this chapter will build on what you've done previously, but you'll apply it in new ways. In addition to creating the content type, you'll incorporate a third-party commenting system into your static site, and you'll implement pagination so you can support future content growth.

A post on a blog has a title, an author, and some content. It might have a summary as well. You might want to put your posts into categories and tag them. You can manage most of this from the front matter of each piece of content much like you did with projects.

Start by creating an archetype for a post so you have a blueprint to follow. Create the file archetypes/posts.md by copying the default archetype file:

blog/portfolio/archetypes/posts.md

```
---
title: "{{ replace .Name "-" " " | title }}"
date: {{ .Date }}
draft: false
---
```

Then, add an author field and add your name:

blog/portfolio/archetypes/posts.md

```
author: Brian Hogan
```

Next, add some default content to this file. Any placeholder text will do, like Lorem Ipsum:[1]

blog/portfolio/archetypes/posts.md

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.

<!--more-->

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat.
```

The <!--more--> line lets you specify where the content summary ends. As you learned in Pulling Data from Remote Sources, on page ?, when you use {{ .Summary }} in a layout, Hugo will pull the summary from the front matter or by generating it from the content. Sometimes that auto-generated summary ends in an awkward place. To control where the summary should end, add <!--more--> to the content.

Save the archetype and use it to generate an initial post to test things out:

```
$ hugo new posts/first-post.md
/Users/brianhogan/portfolio/content/posts/first-post.md created
```

Visit http://localhost:1313/posts/ and you'll see your post listed. This list is generated using the default list layout, which you'll customize later. Let's flesh out the layout for an individual post.

Creating the Post's Layout

The page for an individual post is bare because it's using the default single page layout. Like with project pages, there's information in each post's front matter you can use on the page. To do that, create a new single page layout for posts.

Create the directory themes/basic/layouts/posts/ to hold the layout. You can use your editor or use the following command in your terminal:

```
$ mkdir themes/basic/layouts/posts
```

Then create the file themes/basic/layouts/posts/single.html, which will hold the layout for an individual post.

Create the main content block and place the title and post content in their own sectioning elements with classes; this will help you style them later:

1. https://www.lipsum.com/

blog/portfolio/themes/basic/layouts/posts/single.html

```
{{ define "main" }}
<article class="post">
  <header>
    <h2>{{ .Title }}</h2>
  </header>

  <section class="body">
    {{ .Content }}
  </section>

</article>
{{ end }}
```

Within the header section, add the byline, which will contain the publication date and author name. The .Date field will fetch the date, and the .Format function will let you format it in a similar fashion to how you formatted the date in the footer.

blog/portfolio/themes/basic/layouts/posts/single.html

```
    <header>
      <h2>{{ .Title }}</h2>
➤     <p>
➤       By {{ .Params.Author }}
➤     </p>
➤     <p>
➤       Posted {{ .Date.Format "January 2, 2006" }}
➤     </p>
```

Many blogs will display the amount of time it takes to read the content at the top of an article. Let's add that to our blog page template.

The average person reads anywhere from 200 to 250 words per minute[2] depending on several factors. If you count the number of words in a page's content and divide it by 200, you'll get a conservative estimate of the number of minutes it'll take to read your content.

Hugo has built-in functions for counting words and doing math, so in your template, add the following code to your byline to determine and display the reading time:

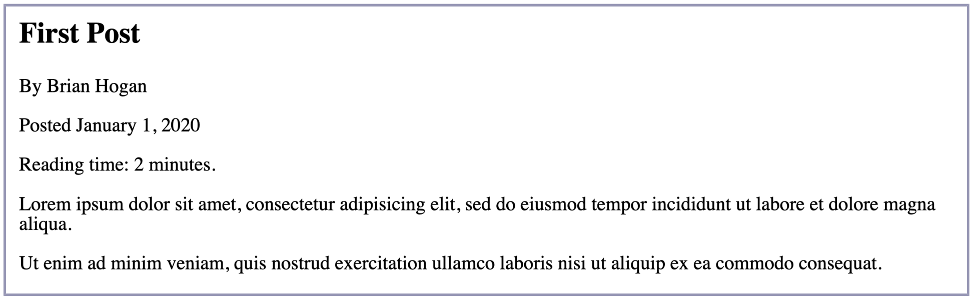
blog/portfolio/themes/basic/layouts/posts/single.html

```
      <p>
        Posted {{ .Date.Format "January 2, 2006" }}
      </p>
➤     <p>
➤       Reading time: {{ math.Round (div (countwords .Content) 200.0) }} minutes.
➤     </p>
```

2. https://www.irisreading.com/what-is-the-average-reading-speed/

To make sure this works, add a significant amount of content to your first blog post so there's some text to count.

Save the file and visit http://localhost:1313/posts/first-post.md. The author, date, and reading time display above the post:



First Post

By Brian Hogan

Posted January 1, 2020

Reading time: 2 minutes.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

As you build out more content, you'll probably want to organize and group it so it's easier for people to find.

Organizing Content with Taxonomies

Many blogs and content systems let you place your posts in categories and apply tags to your posts. This logical grouping of content is known as a *taxonomy*. Hugo supports categories and tags out of the box and can generate category and tag list pages automatically. All you have to do is add categories and tags to your front matter.

Open the posts archetype at archetypes/post.md and add some default categories and tags:

blog/portfolio/archetypes/posts.md

```
categories:
- Personal
- Thoughts
tags:
- software
- html
```

Adding these defaults to the archetype won't affect the content you've already created, so open content/posts/first-post.md and add categories and tags there too:

blog/portfolio/content/posts/first-post.md

```
categories:
- Personal
- Thoughts
tags:
- software
- html
```

Joe asks:

Does Hugo Support Syntax Highlighting for Code?

If you're publishing posts, you might want to include snippets of code from time to time. Hugo supports syntax highlighting using a library named Chroma,[a] which is compatible with the popular Pygments[b] syntax highlighter.

To configure this, tell Hugo you want it to use Pygments-style classes when it highlights your code. Add this line to config.toml:

```
pygmentsUseClasses = true
```

Then, generate a stylesheet to highlight your code using one of the available Chroma styles:[c]

```
$ hugo gen chromastyles --style=github &gt; syntax.css
```

Add the syntax.css style to your head.html partial.

Now, when you write your posts, use code fences and specify the appropriate language:

~~~
```javascript
let x = 25;
let y = 30;
```
~~~

Hugo will apply the styles to your code.

a. https://github.com/alecthomas/chroma
b. https://pygments.org/
c. https://xyproto.github.io/splash/docs/

Both categories and tags are lists of items, so you need to use the correct notation in YAML. Hugo supports TOML and JSON front matter as well, so you could use those for your posts instead if you find those easier to manage.

By adding the tags and categories to your post, Hugo will generate pages for those using your default list layout. Visit http://localhost:1313/tags to see the tags list: