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

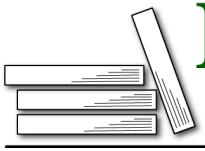
Build Websites with Hugo

Fast Web Development with Markdown

Brian P. Hogan

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

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

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

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

CHAPTER 2

Building a Basic Theme

You've built your site, but you have two nearly identical layout files with a lot of duplicate code. Your site's layout is going to get more complex as you introduce additional content types, navigation, and styles, so it's time to start organizing things. So far, you've placed all of your layout files into your Hugo site's layouts directory, but Hugo has another mechanism for controlling how things look: *themes*.

There are many Hugo themes available that you can use, but instead of using one created by someone else, you're going to build your own. Many off-the-shelf themes are fairly complex with lots of features that take time to configure properly. It's best to understand how theming works before you dive into someone else's code.

A basic Hugo theme only needs these files:

```
layouts
├── _default
│   ├── list.html
│   └── single.html
└── index.html
```

This should look pretty familiar to you from the previous chapter. The index.html file is the home page of the site. The _default directory contains the default layouts applied to single content pages (single.html) and pages that display lists of content pages (list.html). However, as you'll discover in this chapter, there are some additional files you can create that will let you share code between files.

Let's get started with your theme.

Generating the Theme

You can make a theme by creating a new directory in the themes folder of your site, but Hugo has a built-in theme generator that creates everything you need for your theme. Execute this command from the root of your Hugo site to generate a new theme named basic:

```
$ hugo new theme basic
Creating theme at ...portfolio/themes/basic
```

This command creates the themes/basic directory and several subdirectories and files:

```
themes/basic/
├── LICENSE
├── archetypes
│   └── default.md
├── layouts
│   ├── 404.html
│   ├── _default
│   │   ├── baseof.html
│   │   ├── list.html
│   │   └── single.html
│   ├── index.html
│   └── partials
│       ├── footer.html
│       ├── head.html
│       └── header.html
├── static
│   ├── css
│   └── js
└── theme.toml
```

The theme has a place for its own archetypes, as well as directories to store the layout files and static assets like stylesheets, scripts, and images. Each theme has a license file so you can open source your theme if you choose, as well as a configuration file where you can store theme-specific configuration values.

Notice that there's already a layouts/_default directory with files for single page layouts and list page layouts. There's also a layouts/index.html file that serves as the home page layout. Those files don't have any content in them, though.

Move your existing layout files into this new theme. Move the layouts/index.html file to themes/basic/layouts/index.html, and then move layouts/_default/single.html to themes/basic/layouts/_default/single.html. You can do this quickly with these commands:

```
$ mv layouts/index.html themes/basic/layouts/index.html
$ mv layouts/_default/single.html themes/basic/layouts/_default/single.html
```

Before you can use the theme, you have to tell Hugo about it by adding a line to your site's configuration. Open config.toml and add the following line to the end:

basic_theme/portfolio/config.toml

```
baseURL = "http://example.org/"
languageCode = "en-us"
title = "Brian's Portfolio"
theme = "basic"
```

Save the file and run the server again:

```
hugo server
```

Visit http://localhost:1313 in your browser. Your home page still displays, which means your theme works. Once you know it works, stop the development server with Ctrl-c.

Now, let's break the theme up and reduce some duplication by taking advantage of Hugo's layout framework.

Using Content Blocks and Partials

Your home page layout and single page layout both contain the HTML skeleton. That skeleton is going to get a lot more complex once you add a header, footer, and navigation. Instead of duplicating that code in multiple places, Hugo provides a single place for you to put your skeleton so that all other layouts can build on it. When you created the theme, it generated it for you.

Locate the file themes/basic/layouts/_default/baseof.html and open it in your editor. You'll see this code:

```
<!DOCTYPE html>
<html>
    {{- partial "head.html" . -}}
    <body>
        {{- partial "header.html" . -}}
        <div id="content">
        {{- block "main" . }}{{- end }}
        </div>
        {{- partial "footer.html" . -}}
    </body>
</html>
```

This file will be the "base" of every other layout you'll create, hence its name. And instead of including the full skeleton, it's pulling in other files, or *partials*, which contain pieces of the layout. Placing common pieces in partials makes it easier for you to reuse these across different layouts, and it helps you think

about parts of your site as components. The template generator also created these files, but it left them blank. Let's fill them in, one piece at a time.

The first partial listed in the baseof.html file is the head.html file, which you'll find in themes/basic/layouts/partials/head.html. This file will contain the code that would normally appear in the head section of a website. Open it in your editor.

Add the following code to the file to define the head element, with one meta tag that specifies the content type, another meta tag that specifies the viewport, so your page will scale properly on mobile devices, and finally, the title of the site:

basic_theme/portfolio/themes/basic/layouts/partials/head.html

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>{{ .Site.Title }}</title>
</head>
```

Save the file when you've made those changes.

The next partial reference is the header.html file, so open the file themes/basic/layouts/partials/header.html. This file will hold the header of your page. It's where you'll put the banner and navigation. Add the following code to add an HTML header section and a nav section with links to the root of the site and the static pages you created in the previous chapter:

basic_theme/portfolio/themes/basic/layouts/partials/header.html

```
<header>
  <h1>{{ .Site.Title }}</h1>
</header>

<nav>
  <a href="/">Home</a>
  <a href="/about">About</a>
  <a href="/resume">Résumé</a>
  <a href="/contact">Contact</a>
</nav>
```

Hugo has support for a more complex menu system, where you can configure your menu titles and destinations in your site's configuration file, but for small sites like this, it's less work to hard-code the URLs in the navigation. Since the navigation is in a partial, you only have to maintain it in a single file, so there's no advantage to doing it in a data-driven fashion like you would with a dynamic site.

Finally, create the footer of the site by opening themes/basic/layouts/_default/partials/footer.html, the last partial referenced in the baseof.html file. Add a footer element and a copyright date to the file with the following code:

basic_theme/portfolio/themes/basic/layouts/partials/footer.html
```
<footer>
  <small>Copyright {{now.Format "2006"}} Me.</small>
</footer>
```

You're using Go's date formatting to print out the current year. Go uses Mon Jan 2 15:04:05 MST 2006 as a reference time for its formatters. Instead of having to use special characters to parse out a current time, Go lets you use this reference time to format the specific dates and times. In this case, you only want the year, so you can use "2006" as the formatting string.

All of the partials for the base template are in place, but before moving on, let's look at the syntax for partials in the baseof.html file. Partials in the baseof.html template that Hugo generated for you look like this:

```
{{- partial "head.html" . -}}
```

Previously, when you've used those curly braces to inject the title, you used {{. But this code uses {{-. The dash suppresses whitespace characters in the output, such as spaces and newline characters. Placing a dash after the opening braces removes all whitespace in front of the expression, and placing the dash in front of the closing braces removes whitespace after the expression. In most cases, it's up to you whether or not you want to use them, but you'll often see dashes used to reduce the number of blank lines in the files Hugo generates.

In addition to the dashes, there's a dot after the name of the partial. The partial function takes a filename and a context for the data. In order to access data like the page title and content, you have to pass the context to the partial function, and a period means "the current context." Remember that the default context for a layout is the Page context, so when you pass the dot to the partial function, you're making the Page context available to the partial.

To use the new base template, replace the existing layouts you've used with code that defines a layout "block". In the baseof.html file, you'll find this line:

```
{{- block "main" . }}{{- end }}
```

This line looks for a block named "main" and pulls it in. Those blocks are what you'll define in your actual layout pages like index.html and single.html. Notice it's also passing the current context, so you'll be able to access it in the layout pages.

Define this block in your home page layout first. Open themes/basic/layouts/index.html and replace the contents with this code, which defines the main block:

basic_theme/portfolio/themes/basic/layouts/index.html

```
{{ define "main" }}
  {{ .Content }}
{{ end }}
```

When Hugo builds the site, it'll construct the home page by combining the content file (content/_index.md) with the layout file, which will then use the baseof file. The partials and content are all assembled, creating the full page.

The index.html layout only affects your site's home page, so modify the code in themes/basic/layouts/_default/single.html so the single page layout works the same way:

basic_theme/portfolio/themes/basic/layouts/_default/single.html

```
{{ define "main" }}
  <h2>{{ .Title }}</h2>
  {{ .Content }}
{{ end }}
```

This code looks just like the code in the home page layout, except you're also displaying the title of the page.

With the changes in place, make sure all your files are saved and then fire up the development server with hugo server. Visit http://localhost:1313 and test your pages. The home page displays with the current year displayed in the footer.

Use the navbar you created to jump between pages to ensure they all work.

You now have a working theme that's organized and maintainable. Let's add some CSS to make it look nice.