CHAPTER 1

Kicking the Tires

The best way to get comfortable with Hugo is to start building something with it. Throughout this book, you'll use Hugo to build a portfolio site with a blog. In this chapter, you'll develop a basic understanding of Hugo, layouts, content, and configuration, which form the foundation for working with Hugo as you build a basic site.

But first, you need to install Hugo.

Installing Hugo

Hugo is available as a single binary. You can install Hugo two ways: using a package manager, or by downloading the binary manually and placing it in a global location so you can run it anywhere.

Hugo comes in two versions: a regular version and an extended version that has additional support for asset management, which you'll want in Chapter 7, Managing Assets with Pipes, on page 91.

Package managers are utilities that let you install and remove programs and utilities. Linux systems often have a built-in package manager. If you're running macOS, you can install the Homebrew[1] package manager. If you're using Windows, you can install Chocolatey.[2] Finally, Linux users can install Linuxbrew,[3] a version of Homebrew for Linux that offers more up-to-date packages than the package manager that comes with their system.

Installation via a package manager is the easiest method, although you might not always get the most recent version.

1. https://brew.sh
2. https://chocolatey.org
3. https://docs.brew.sh/Homebrew-on-Linux

If you're using a Mac and you have Homebrew installed, you can install the extended version of Hugo with:

```
$ brew install hugo
```

On Windows, if you have Chocolatey installed, you can install Hugo with this command:

```
$ choco install -y hugo-extended
```

To install Hugo manually, find the extended version's binary for your operating system on the Releases page[4] on GitHub. For example, if you're on Windows, you're looking for a filename like hugo-extended-x.y.z-Windows-64bit.zip. Download the file.

Now that you've manually downloaded the file, you'll want to add it to your PATH environment variable, which your command-line interface uses to determine where it can find executable programs. That way you can execute it without specifying the full directory location.

On macOS or Linux, copy the executable to /usr/local/bin, which is already included in your PATH environment variable by default.

On Windows 10, create the directory C:\hugo\bin. Copy the hugo.exe file you extracted to C:\hugo\bin, and then add that folder to your PATH environment variable. To do this, use the Windows Search and type "environment". Choose the option for Edit Environment Variables for My Account. On the screen that appears, press the Environment Variables button, locate PATH in the System Variables section, and press the Edit button. Add c:\hugo\bin. Press OK to save the settings, and then press OK on the rest of the dialogs to close them.

Once Hugo is installed, run the hugo version command from your command-line interface to ensure that Hugo is available in any directory on your system, and that you've installed the extended version:

```
$ hugo version
Hugo Static Site Generator v0.68.3/extended darwin/amd64 BuildDate: unknown
```

The hugo command has several subcommands that you'll use as you build your site. You can see a list of all commands with hugo help.

Hugo is installed and ready. Let's use it to build a basic site.

4. https://github.com/gohugoio/hugo/releases

Creating Your Site

Hugo has a built-in command that generates a website project for you; this includes all of the files and directories you need to get started.

Execute the following command to tell Hugo to create a new site named portfolio:

```
$ hugo new site portfolio
```

This creates the portfolio directory, with the following files and directories within:

```
portfolio/
├── archetypes
│   └── default.md
├── config.toml
├── content
├── data
├── layouts
├── static
└── themes
```

Each of these directories has a specific purpose:

- The archetypes directory is where you place Markdown templates for various types of content. An "archetype" is an original model or pattern that you use as the basis for other things of the same type. Hugo uses the files in the archetypes folder as models when it generates new content pages. There's a default one that places a title and date in the file and sets the draft status to true. You'll create new ones later.
- The config.toml file holds configuration variables for your site that Hugo uses internally when constructing your pages, but you can also use values from this file in your themes. For example, you'll find the site's title in this file, and you can use that in your layout.
- The content directory holds all of the content for your site. You can organize your content in subdirectories like posts, projects, and videos. Each directory would then contain a collection of Markdown or HTML documents.
- The data directory holds data files in YAML, JSON, or TOML. You can load these data files and extract their data to populate lists or other aspects of your site.
- The layouts folder is where you define your site's look and feel.
- The static directory holds your CSS, JavaScript files, images, and any other assets that aren't generated by Hugo.

- The themes directory holds any themes you download or create. You can use the layouts folder to override or extend a theme you've downloaded.

In your terminal, switch to the newly created portfolio directory:

```
$ cd portfolio
```

Take a look at the site's configuration file. Open the config.toml file in your text editor. You'll see the following text:

```
baseURL = "http://example.org/"
languageCode = "en-us"
title = "My New Hugo Site"
```

This file is written in TOML,[5] a configuration format designed to be easy to read and modify. The default configuration file only has a handful of data, but you'll add more as you build out your site.

Hugo's internal functions use the baseURL value to build absolute URLs. If you have a domain name for your site, you should change this value to reflect that domain. In this book, you'll use relative URLs, so you can leave this value alone until you're ready to deploy your site to production.

The title value is where you'll store the site's title. You'll use this value in your layouts, so change it from its default value:

kicking_tires/portfolio/config.toml

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

Save the file and exit the editor. You're ready to start working on the site itself.

Building the Home Page

A typical web page has a skeleton that looks like this:

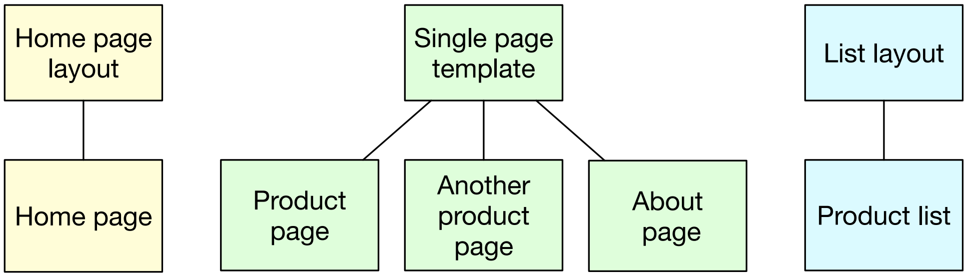
```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>Title</title>
  </head>
  <body>
    <!-- some content goes here -->
  </body>
</html>
```

5. https://github.com/toml-lang/toml

Every time you create a page, you start with a skeleton such as this and then fill in the `body` section. Your site will likely have a lot more common elements like navigation, a banner, and a footer. As you build out your site, you end up duplicating all of this content on every page, which is difficult to manage if you do it by hand. That's why dynamic sites are so popular; they provide mechanisms to reduce duplication by separating the content from the layout.

In a Hugo site, you define *layouts* that contain this skeleton, so you can keep this boilerplate code in a central location. Your content pages contain only the content, and Hugo applies a layout based on the type of content.

Hugo needs a layout for the home page of your site, a layout for other content pages, and a layout that shows a list of pages, like an archive index or a product list. The following figure illustrates this relationship:



As you can see in the figure, the "home page" of the site has its own layout. The "Product list" page has a "list layout" associated with it. However, the two product pages and the "About" page all share a layout.

In this chapter, you'll create a layout for the home page and another layout for single pages, and they'll have nearly identical content. You'll build a layout for list pages in the next chapter.

Let's start with the home page layout. Hugo uses a separate layout for the home page because it assumes you'll want your home page to have a different look than other content pages. For example, your home page might contain a list of recent blog posts or articles, or show previews of other content. Or it might have a larger banner image than other pages. For now, you'll keep the home page simple.

Create the file `layouts/index.html` and add the following code, which defines an HTML skeleton for the home page:

kicking_tires/portfolio/layouts/index.html

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8">
    <title>{{ .Site.Title }}</title>
  </head>
  <body>

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

    {{ .Content }}

  </body>
</html>
```

To pull in data and content, Hugo uses Go's http/templates library[6] in its layouts. While you don't need a deep understanding of how the templating language works, it's worth exploring when you want to build more complex templates.

In addition to the HTML code, there are some spots between double curly braces. This is how you define where content goes, and the content can come from many places. The {{ .Site.Title }} lines pull the title field out of the config.toml file you just edited and place it into the <title> and <h1> tags. The .Site prefix tells Hugo to get the value from the site's data rather than from the page's data. As you'll see shortly, pages have their own data that you can use.

The {{ .Content }} line displays the content for the page, which will come from an associated Markdown document in your site's content folder. Note that this doesn't use the .Site prefix, because this data is related to the page. When you're working with a Hugo layout, there's a "scope", or "context" that contains the data you want to access. The current context in a template is represented by the dot. In a Hugo layout, the context is set to the Page context, which looks something like this:

```
Context (.)
├── Site.
│   └── Title
├── Title
└── Content
```

For convenience, Hugo makes all of the site's data available in the Page context under the Site key. That's why you can do .Site.Title to get the site's title. To get the page title, you'd use .Title. There are many more pieces of data available to you within this context, and you'll use many of them later in the book.

6. https://golang.org/pkg/text/template/

You've added the {{ .Content }} statement to the home page layout, but you're probably wondering where the content comes from. For the site's home page, Hugo will look for the content in a file named _index.md in the content directory. This special filename is how Hugo finds content for all index pages, like the home page and lists of content like tags, categories, or other collections you'll create.

Create the content/_index.md file and open it in your editor. You can place any valid Markdown content into this file. For this example, add a couple sentences and a list of what's on the site:

kicking_tires/portfolio/content/_index.md

```
This is my portfolio.

On this site, you'll find

* My biography
* My projects
* My résumé
```

After you've added your content, save the file.

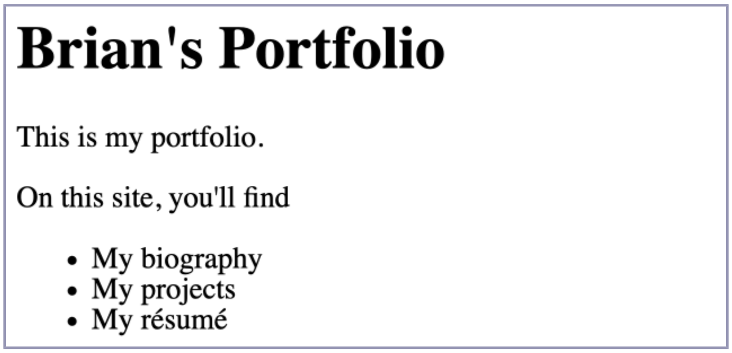
Time to test things out. Hugo has a built-in development server that you can use to preview your site while you build it. Run the following command to start the server:

```
$ hugo server
```

Hugo builds your site and displays the following output in your console letting you know that the server is running:

```
...
Web Server is available at http://localhost:1313/ (bind address 127.0.0.1)
Press Ctrl+C to stop
```

Visit http://localhost:1313 in your web browser and you'll see your home page:



Brian's Portfolio

This is my portfolio.

On this site, you'll find

- My biography
- My projects
- My résumé

Use your browser's View Source feature and you'll see the layout and your content combined:

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta name="generator" content="Hugo 0.68.3" />
    <script src="/livereload.js?port=1313&mindelay=10&v=2"
            data-no-instant defer></script>
    <meta charset="utf-8">
    <title>Brian&#39;s Portfolio</title>
  </head>
  <body>
    <h1>Brian&#39;s Portfolio</h1>
    <p>This is my portfolio.</p>
    <p>On this site, you&rsquo;ll find</p>
    <ul>
    <li>My biography</li>
    <li>My projects</li>
    <li>My re´sume´</li>
    </ul>
  </body>
</html>
```

Hugo's development server automatically reloads files when they change. Open the content/_index.md file in your text editor and make some changes to the content. When you save the file, the changes appear in your browser automatically. There's no need to install a separate server or browser extension. The single Hugo binary handles it all for you by injecting a little bit of JavaScript at the bottom of each page which handles reloading the page.

In your terminal, press CTRL+C to stop the Hugo server. You're ready to add another page to the site, but this time you will do it with Hugo's content generator.

Creating Content Using Archetypes

When you created content/_index.md, you created the file manually. You can tell Hugo to create content pages that contain placeholder content. This way, you never have to start with a blank slate when creating content. Hugo uses archetypes to define this default content.

The file archetypes/default.md—which was generated when you ran the hugo new site command—contains the following code:

kicking_tires/portfolio/archetypes/default.md

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

This is a Markdown file with YAML front matter and a little bit of templating code that will generate the title from the filename you specify, and fill in the date the file is generated. Hugo uses the front matter of each file when it generates pages. You'll dig into front matter a lot more in Populating Page Content Using Data in Front Matter, on page 36. Front matter isn't required in either content pages or archetypes, but you'll find it useful in many places as you build out your site. In this case, there's a draft field set to true. When Hugo generates pages, it will skip pages marked as drafts.

The hugo new command uses this file as a template to create a new Markdown file. Try it out. Create a new "About" page with this command:

```
$ hugo new about.md
/Users/brianhogan/portfolio/content/about.md created
```

This generates the file content/about.md. Open the file in your editor, and you'll see this code:

```
---
title: "About"
date: 2020-01-01T12:40:44-05:00
draft: true
---
```

The title is filled in, and the front matter also includes the date and time the file was created. It also has a draft status of true. Modify the draft status to false or remove the line entirely so that Hugo will generate this page. Then, below the front matter, add some additional content. When you're done, your file will look like this:

kicking_tires/portfolio/content/about.md

```
---
title: "About"
date: 2020-01-01T12:40:44-05:00
draft: false
---

This is my About page.
```

In order for Hugo to generate this page, you need another layout file. Remember that the layout you created, index.html, is only for the home page of the site. The "About" page you just created is an example of what Hugo calls a "single" page. As such, you need a "single page" layout to display it.

You can create different single page layouts for each content type. These get stored in subdirectories of the layouts directory. You're not going to do anything that complex yet. To create a default single page layout that every content page will use, store it in the layouts/_default directory. Create that directory now, either in your text editor or on the command line like this:

```
$ mkdir layouts/_default/
```

Now, create the file layouts/_default/single.html by copying the existing layouts/index.html file. Again, you can do this in your editor or the CLI:

```
$ cp layouts/index.html layouts/_default/single.html
```

Let's make a slight modification to this file; you'll have it display the page title in addition to the content. Open the single.html file in your editor and add the title like this:

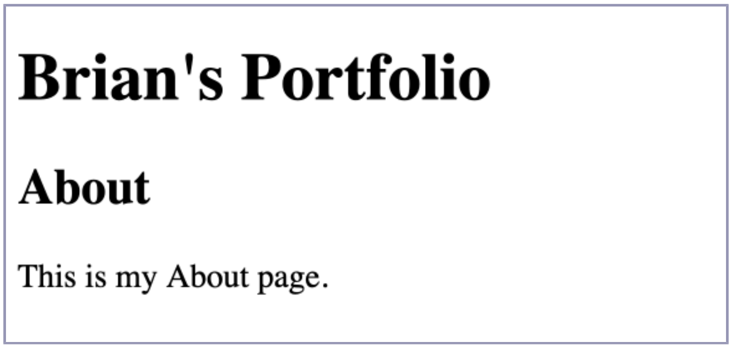
kicking_tires/portfolio/layouts/_default/single.html

```
<body>
  <h1>{{ .Site.Title}}</h1>
➤ <h2>{{ .Title }}</h2>
  {{ .Content }}
</body>
```

Save the file and exit the editor. Then, start up the development server again:

```
$ hugo server
```

Visit http://localhost:1313/about and you'll see your new page:



Brian's Portfolio

About

This is my About page.

The site title, page title, and page content are all visible. The page title comes from the about.md file's front matter, while the site title comes from the config.toml file. This is a small example of how you can use data from various locations to build your pages.

Stop the Hugo server with CTRL+C. Let's explore what Hugo creates for you and look at a few options to control it.

Building and Exploring Hugo's Output

When you run hugo server, it generates your content in memory. To write Hugo's output to disk, run the hugo command with no options, like this:

```
$ hugo
...
                   | EN
+------------------+----+
  Pages            |  5
  Paginator pages  |  0
  Non-page files   |  0
  Static files     |  0
  Processed images |  0
  Aliases          |  0
  Sitemaps         |  1
  Cleaned          |  0

Total in 21 ms
```

This generates the pages for your site in a new directory named public. If you look at that directory's contents, you'll see these files and directories:

```
public/
├── about
│   └── index.html
├── categories
│   └── index.xml
├── index.html
├── index.xml
├── sitemap.xml
└── tags
    └── index.xml
```

Hugo has created the HTML files for your home page and the About page, as well as the file index.xml, which is an RSS feed for your site. It's also created a sitemap file, as well as RSS feeds for your site's tags and categories. You're not doing anything with those right now, so you can ignore them.

This public folder is your entire static site. To host it, you could upload the contents of the public folder to your web server, or upload it to your CDN. You'll explore how to put your site live in Chapter 8, Deploying the Site, on page 109.

Hugo doesn't clean up the public folder. If you were to remove some pages or rename them, you would need to delete the generated versions from the public folder as well. It's much easier to delete the entire public folder whenever you generate the site:

```
$ rm -r public && hugo
```

Alternatively, use Hugo's --cleanDestinationDir argument:

```
$ hugo --cleanDestinationDir
```

You can tell Hugo to minify the files it generates. This process removes whitespace characters, resulting in smaller file sizes that will make the site download faster for your visitors. To do this, use the --minify option:

```
$ hugo --cleanDestinationDir --minify
```

The public/index.html now looks something like this:

```
<!doctype html><html lang=en-us><head><meta name=generator
content="Hugo 0.68.3"><meta charset=utf-8><title>Brian's Portfolio</title>
</head><body><h1>Brian's Portfolio</h1><p>This is my portfolio.</p>
<p>On this site, you&rsquo;ll find</p><ul><li>My biography</li>
<li>My projects</li><li>My re´sume´</li></ul></body></html>
```

All the indentation and line breaks are removed. As a result, this file contains fewer bytes, which means it'll transfer faster when you eventually deploy the files to production.

Your Turn

Before moving on, try the following things to make sure you understand how page creation and content generation works:

1. Change the draft status of the About page's content back to true and regenerate the site with the hugo command, with no additional options. Notice that the about/index.html file still exists. Use the --cleanDestinationDir option and the about/index.html file will disappear. Set the draft status back to false again and rebuild the site to restore the file.

2. Create a "Résumé" page using hugo new resume.md. The generated title in the front matter will say "Resume". Open the content/resume.md file in your editor, update the generated title from "Resume" to "Résumé", and set the draft status to false. Fill in some content, and then run the development

web server. Navigate to http://localhost:1313/resume to view the page. The single page content template you created applies to this page as well.

3. Experiment with the default archetype. Change the draft state to false in the default archetype and generate a fourth page called contact.md. This new page will now have the draft state set to false. All future pages you create with the hugo new command will now have this status set.

Wrapping Up

You built a basic Hugo site, and you created a handful of content pages using Hugo's command-line interface. You've also defined the layout those content pages will use. However, you've got some duplicate code in your layout, which makes your site more difficult to maintain. In the next chapter, you'll extract your layout into a reusable theme and organize the code to reduce duplication.