Learn

Hugo Templates for WordPress Designers

Leon Barnard

writes on September 29, 2017

Last year I wrote an article introducing the general concepts behind static site generators. This time around I'd like to dive into some specific (yet still basic) details about a popular static site generator called **Hugo**, comparing it to its most well-known "dynamic" ancestor, WordPress.

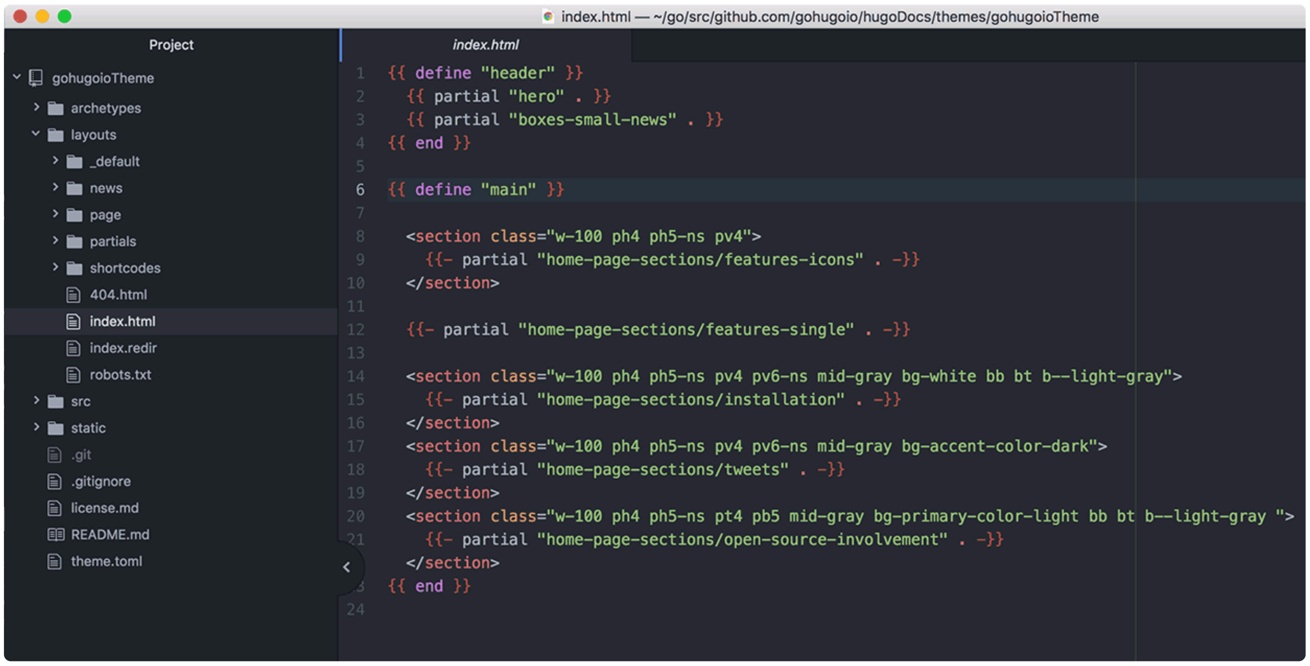
Introduction

Before we get started, let me clarify that I won't be getting into how to migrate from WordPress to Hugo. There is an export tool available, and numerous blog posts describing how others did it.

Instead, I'll be focusing on the theme templates that each one uses, since that's one of the biggest adjustments when diving into Hugo, whether you've migrated from WordPress or are starting from scratch.

The good news is that there is a lot of overlap between Hugo and WordPress in terms of how the templates work and the features they offer.

When I wrote about Hugo last year it lacked many of the features that make WordPress so popular. But it has evolved tremendously since then (while still remaining incredibly fast) and now offers "grown up" features like nested templates and related posts, with new features being added all the time.



```
{{ define "header" }}
  {{ partial "hero" . }}
  {{ partial "boxes-small-news" . }}
{{ end }}

{{ define "main" }}

  <section class="w-100 ph4 ph5-ns pv4">
    {{- partial "home-page-sections/features-icons" . -}}
  </section>

  {{- partial "home-page-sections/features-single" . -}}

  <section class="w-100 ph4 ph5-ns pv4 pv6-ns mid-gray bg-white bb bt b--light-gray">
    {{- partial "home-page-sections/installation" . -}}
  </section>
  <section class="w-100 ph4 ph5-ns pv4 pv6-ns mid-gray bg-accent-color-dark">
    {{- partial "home-page-sections/tweets" . -}}
  </section>
  <section class="w-100 ph4 ph5-ns pt4 pb5 mid-gray bg-primary-color-light bb bt b--light-gray ">
    {{- partial "home-page-sections/open-source-involvement" . -}}
  </section>
{{ end }}
```

Template Basics

The first thing to know about Hugo is that it is written in the **Go** programming language. Go is not as well-known as many other languages on the web, but it is gaining rapidly in popularity and Treehouse now offers a Go Language Overview course.

If the thought of learning a new programming language makes switching to a new content system a no-Go (pun intended 😉), **don't give up just yet!** I don't know how to program in Go and yet I've been able to write sophisticated Hugo templates for the Balsamiq documentation sites mostly just by knowing HTML and perusing the Hugo documentation.

Let me try to convince you by giving you the first few lessons in Hugo template creation.

Different Syntax, Same Functions

In WordPress, when you want to display the content of a blog post or page, you write this line in a template

```
<?php the_content(); ?>
```

In Hugo, it's

```
{{ .Content }}
```

Not too bad, right? You might even say it's an improvement. (You're going to have to get used to those double curly brackets and dots, but you get to give up those weird question marks, which always made me unsure of what I was doing.)

Let's keep it going by mentioning that, like PHP, Hugo code can be used right alongside HTML, either next to it or inside it.

If you've customized a WordPress template before, you've probably written a mix of HTML and PHP like this before.

```
<a href="/"><?php bloginfo('name'); ?></a>
```

This code locates the name of the site and creates a link to the home page around it. WordPress has a function called `bloginfo` with a parameter of `name` to get it.

Hugo, on the other hand, uses a variable called `.Site` with a property called `.Title`. The same code in Hugo would be written like this.

```
<a href="/">{{ .Site.Title }}</a>
```

Hey, that's also pretty intuitive, no?

I hope you've calmed back down by now. 🙂

Let's keep going.

The last basic example is showing a list of posts or pages. The general structure is the same in both systems: loop through the posts and add a list item for each one, using a link to the post and its title.

In WordPress...

```
<ul>
    <?php while (have_posts()) : the_post(); ?>
        <li><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></li>
    <?php endwhile; ?>
</ul>
```

WordPress uses a PHP `while` loop, which is common in many programming languages. Hugo uses a simple function called `range` for this same purpose.

In Hugo...

```
<ul>
    {{ range .Data.Pages }}
        <li><a href="{{ .Permalink }}">{{ .Title }}</a></li>
    {{ end }}
</ul>
```

The list items themselves look very similar between the two languages, but, again, the Hugo one looks a little neater. And it requires less text overall, making it easier to read.

Zooming Out

So now that we've established some of the basic syntactical differences between WordPress and Hugo, let's look at the different kinds of templates.

Includes, a.k.a. Partials

One of the main reasons to use a website generator over, say, writing straight HTML, is the ability to re-use code across pages. It would be silly in this day and age to copy and paste the same HTML for the header and footer, for example, into each page that you create.

The PHP `include()` function alone is one of the main reasons why it became so popular as a web language early on and why WordPress uses PHP.

You can use the `include()` function in WordPress, although they now have their own functions that make it easier to retrieve specific commonly-used parts of the page. To include the global page footer, you can write

```
<?php get_footer(); ?>
```

Hugo does not make assumptions about the structure of your page the way that WordPress does and uses a function like the original PHP `include()` called `partial` to insert content from one file inside another.

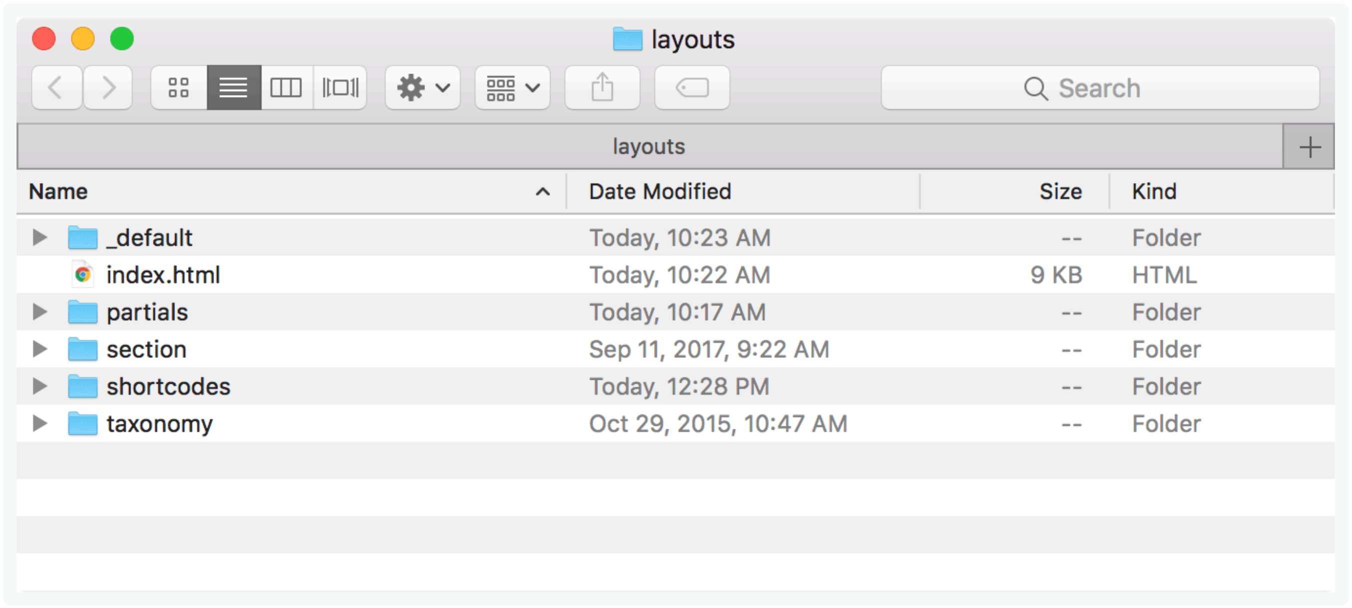
It works like this

```
{{ partial "footer.html" . }}
```

(Don't forget the dot ("."), called the context, at the end.)

Note that the name of the partial file must be located in the "partials" directory, which is one of the default folders inside the "layouts" directory, where your template files live.

This is a good time to introduce one of the main differences between WordPress and Hugo in terms of templates. The locations of things in WordPress are mostly **obscured from view**, with pretty much everything residing in a database (as shown in this image). But static sites are just **copies of files on your computer**, allowing you to preview and manipulate the structure of your site.



Name	Date Modified	Size	Kind
_default	Today, 10:23 AM	--	Folder
index.html	Today, 10:22 AM	9 KB	HTML
partials	Today, 10:17 AM	--	Folder
section	Sep 11, 2017, 9:22 AM	--	Folder
shortcodes	Today, 12:28 PM	--	Folder
taxonomy	Oct 29, 2015, 10:47 AM	--	Folder

A typical "layouts" folder in Hugo.

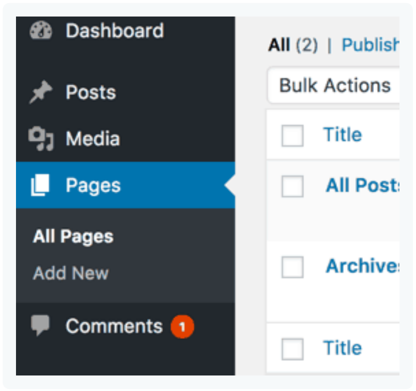
It is a lot like mobile vs. desktop operating systems. On desktop operating systems you can look into the file structure (via Finder, Explorer, etc.), but mobile operating systems try to hide this hierarchy and simply associate files with applications, freeing you up from having to worry about the internal structure.

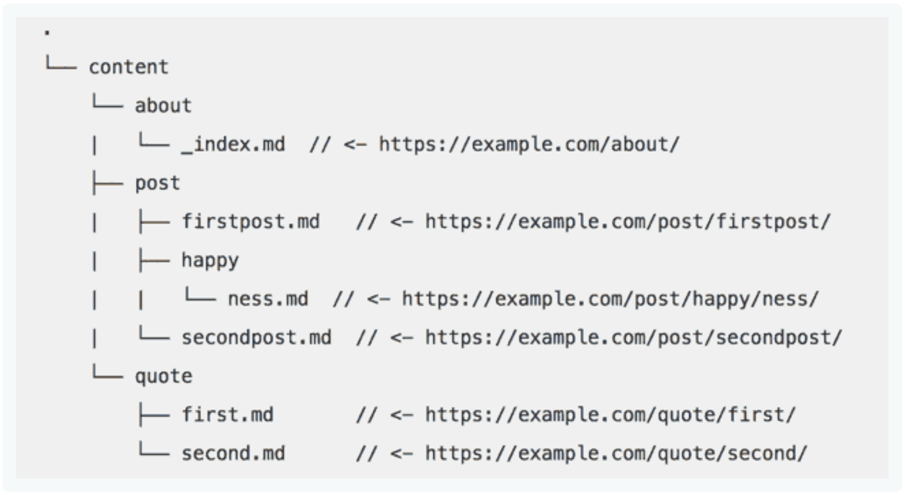
In our case, Hugo behaves like the desktop operating system and WordPress is more like a mobile OS. This is not necessarily a bad thing, as it gives you more clarity into the navigation and structure of your website. But it does hand you the burden of making sure that you put things in the right place.

Luckily, Hugo does a good job of keeping its names consistent. It's handy that the function is called `partial`, which is a good reminder to put your partial files in the "partials" folder. 🙂

Section Templates

Continuing this thread on the philosophical differences between Hugo and WordPress, Hugo is less prescriptive about the organization of your site. While WordPress uses a rigid model of posts and pages, Hugo is crafted in terms of generic "content" and folders.





```
.
└── content
    └── about
    |   └── _index.md  // <- https://example.com/about/
    ├── post
    |   ├── firstpost.md   // <- https://example.com/post/firstpost/
    |   ├── happy
    |   |   └── ness.md  // <- https://example.com/post/happy/ness/
    |   └── secondpost.md  // <- https://example.com/post/secondpost/
    └── quote
        ├── first.md       // <- https://example.com/quote/first/
        └── second.md      // <- https://example.com/quote/second/
```

Posts and Pages in WordPress (top) vs. content folders in Hugo (bottom)

The primary assumption that Hugo makes about your content is that you have organized it deliberately. One effect of this is that the top-level folder structure allows you to define different behavior for the content inside each folder.

The Hugo documentation says "[w]hile Hugo supports content nested at any level, the **top levels are special**." To reinforce this idea, the top-level folders have their own name in Hugo, called a **content section**, or section for short. The closest analogy, in WordPress parlance, is to think of a Hugo section like a WordPress category.

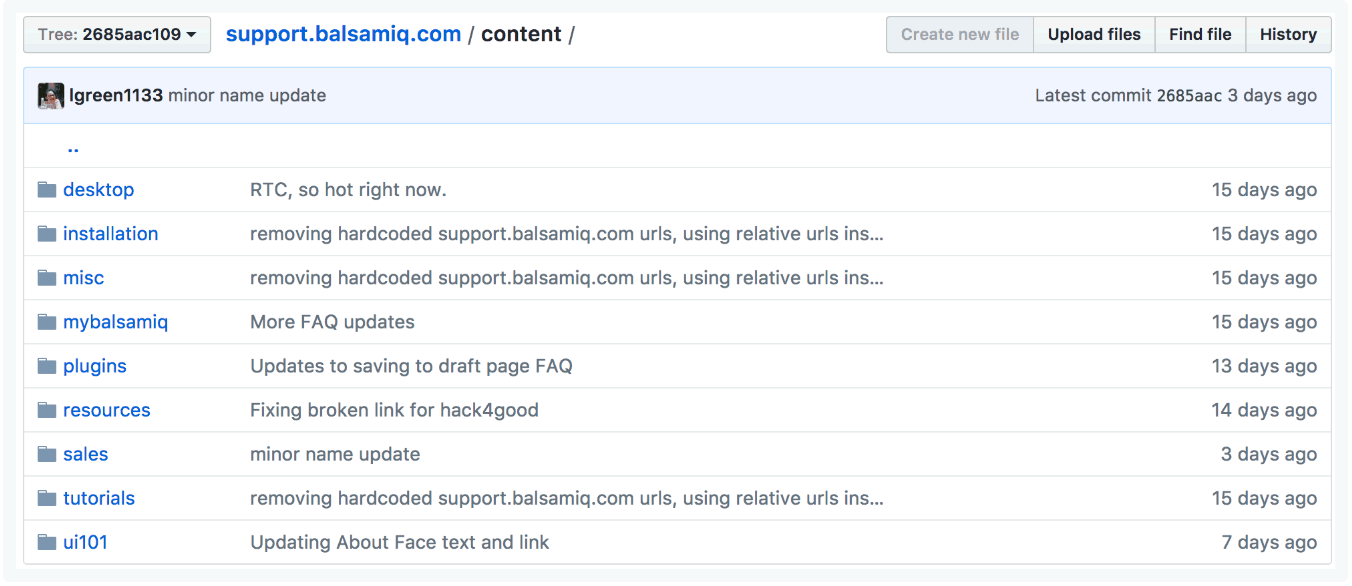
What this means for you is that Hugo makes it easy to create different templates for each folder of your content ("section").

In terms of file structure, we're jumping from the "partials" folder in your Hugo "layouts" directory to one of its siblings, the "section" folder. And, again, names are important.

To make a section template **create a template file whose name is the same as the folder it applies to**. So, if you have a top-level folder called "features", you would create a section template called "features.html". That's it.

The nice thing about section templates is that you don't have do anything to get them to work (unlike partials, which have to be explicitly referenced in a template). If a file exists with the same name as a content folder, Hugo will automatically use that file as the template for that section. If not, it will just use a default template.

To see how this looks in practice, take a look at the structure of the Balsamiq support site, which has content folders called "plugins", "tutorials", "sales", and others.

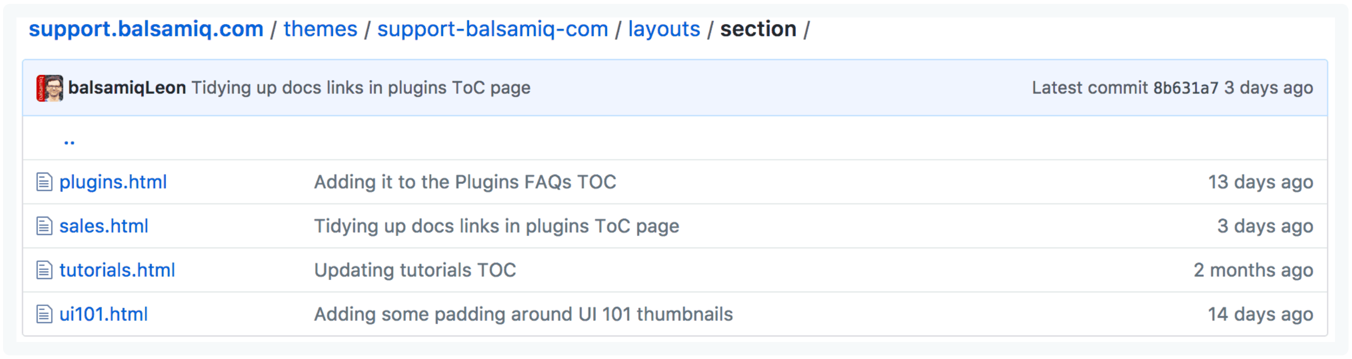

Tree: 2685aac109 ▾ support.balsamiq.com / content /

Create new file | Upload files | Find file | History

lgreen1133 minor name update — Latest commit 2685aac 3 days ago

..		
desktop	RTC, so hot right now.	15 days ago
installation	removing hardcoded support.balsamiq.com urls, using relative urls ins...	15 days ago
misc	removing hardcoded support.balsamiq.com urls, using relative urls ins...	15 days ago
mybalsamiq	More FAQ updates	15 days ago
plugins	Updates to saving to draft page FAQ	13 days ago
resources	Fixing broken link for hack4good	14 days ago
sales	minor name update	3 days ago
tutorials	removing hardcoded support.balsamiq.com urls, using relative urls ins...	15 days ago
ui101	Updating About Face text and link	7 days ago

In the section directory, there are template files for some of these, named after the corresponding content folder (e.g., “plugins.html”).


support.balsamiq.com / themes / support-balsamiq-com / layouts / section /

balsamiqLeon Tidying up docs links in plugins ToC page — Latest commit 8b631a7 3 days ago

..		
plugins.html	Adding it to the Plugins FAQs TOC	13 days ago
sales.html	Tidying up docs links in plugins ToC page	3 days ago
tutorials.html	Updating tutorials TOC	2 months ago
ui101.html	Adding some padding around UI 101 thumbnails	14 days ago

Any content in a folder whose name matches one of those section template files will automatically get that template applied.

Shortcodes

Let’s end with a feature that Hugo and WordPress are in strong agreement on. Hugo has a familiar useful feature called shortcodes, known in WordPress as…shortcodes. 🙂

A shortcode is a text snippet that’s a kind of shorthand for a longer block of code.

In WordPress you can write a shortcode by surrounding its name with square brackets and passing in parameters, like this

```
[gallery id="123" size="medium"]
```

Hugo works nearly the same way, but uses a slightly different syntax. The same shortcode would be written in Hugo as

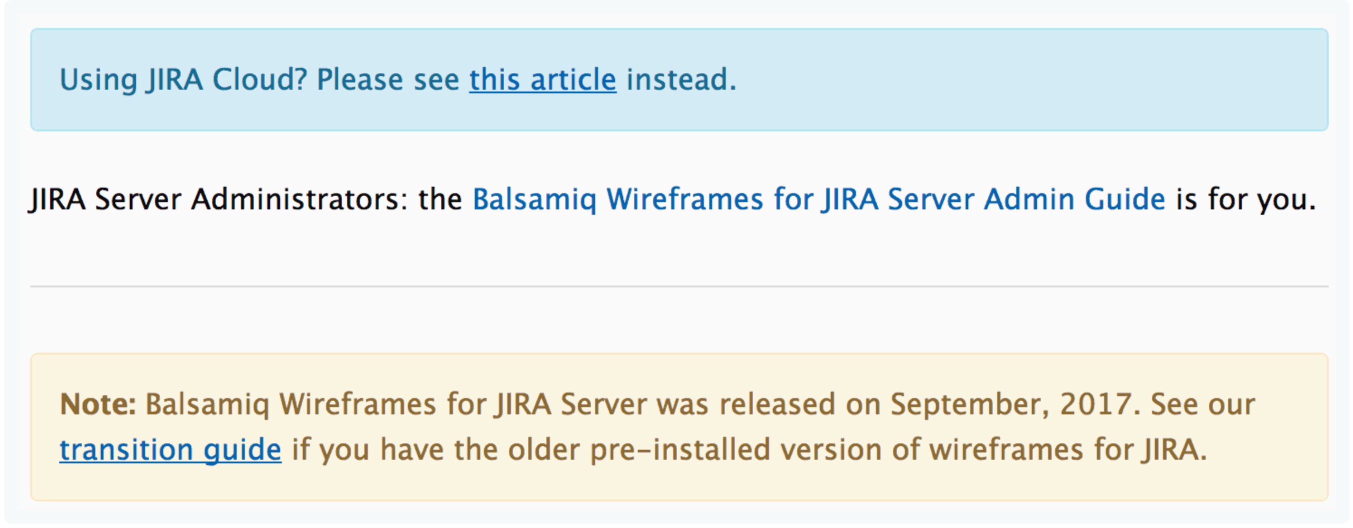
```
{{< gallery id="123" size="medium" >}}
```

In Hugo, each shortcode is an HTML template file. And where do you put those files? You guessed it, in a folder called “shortcodes”.

Writing your own custom shortcodes can be tricky – you’ll probably be writing something like `{{ .Get 0 }}` to grab the parameters – but it’s not like writing shortcodes in WordPress is a piece of cake either.

In practice, you’ll probably use (or maybe customize) an existing shortcode, just like you do in WordPress. Hugo comes with some handy shortcodes out of the box for YouTube, Instagram, Twitter, and others.

One way we use shortcodes for the Balsamiq documentation site is for alert and information messages, as shown here:


Using JIRA Cloud? Please see this article instead.

JIRA Server Administrators: the Balsamiq Wireframes for JIRA Server Admin Guide is for you.

Note: Balsamiq Wireframes for JIRA Server was released on September, 2017. See our transition guide if you have the older pre-installed version of wireframes for JIRA.

Info and Alert messages in the Balsamiq Docs.

This Hugo template also has some nice examples of these types of shortcodes in use.

The End and The Beginning

In this post I've covered some highlights of the differences and similarities between WordPress and Hugo, which is hopefully enough to help you make the paradigm shift from one to the other.

But, of course, there's a lot I didn't cover. Hugo also supports tags (like WordPress) through something called "taxonomies" and default content templates with something called "archetypes", but most of these features you don't need to understand right away (or even at all).

One thing I've observed about the evolution of Hugo is that it is becoming **less esoteric and more practical** and functional in response to the features that people are asking for. I see this as a good thing, and probably one of the big reasons that Hugo is rapidly gaining in popularity.

I remember spending days trying to understand this explanation of taxonomies early on, for example. Whereas a newer feature like sections is a lot more straight-forward.

Lastly, here's a list of excellent Hugo resources to help you on your journey. If you've made it this far you must be at least a little curious to learn more.

- The official Hugo documentation.
- The official Hugo forums – a great place to find answers to your questions or debug your code.
- Giraffe Academy Hugo tutorial course on YouTube – 23 videos!
- A Hugo "Learn" theme – Good for playing around with.
- More Hugo themes – so you don't have to start from scratch.
- One of many blog posts on migrating from WordPress to Hugo.
- Getting Started with Static Sites – my previous post on this blog about static sites.

Thanks for reading. Happy site-building!

Leon Barnard is a writer and designer at Balsamiq. He writes for the Balsamiq blog and manages their Hugo-based documentation and support sites.