

[Home](#)[Blog](#)[Search](#)

Make a Hugo Blog From Scratch

Published: Jan 13, 2019

Updated: Aug 3, 2021

Table of Contents

- [Sister Links](#)
- [Prerequisites and Notes](#)
- [Create the Site](#)
- [Homepage Layout](#)
- [The `static` Folder](#)
- [single Page Layout](#)
- [Create the First Blog Post](#)
- [baseof Layout](#)
- [Partials](#)

- [list Page Layout](#)
- [Menu](#)
- [Tweak Blog List Layout Title](#)
- [Date and Tags Partial](#)
- [terms List Layout](#)
- [Style Tweaks](#)
- [Better Title Logic](#)
- [Wrap Up](#)
- [Emails I've Received](#)

In my opinion, Hugo's current [quick start](#) is ample. It does just what the name says, *gets you started you quickly*.

Still, there have many requests on the [forums](#) for a tutorial that dives deeper than the quick start, and gets into some templating basics. The thing you're reading is my go at that. Starting from scratch, we'll build a Hugo blog.

Sister Links

- **GitHub:** <https://github.com/zwbetz-gh/make-a-hugo-blog-from-scratch>
- **Demo:** <https://make-a-hugo-blog-from-scratch.netlify.app>

Prerequisites and Notes

- Use Hugo version `0.58.3` or higher

- Basic knowledge of HTML, CSS, and Hugo templates is nice-to-have, but not necessary
- File paths will be given in Linux/Mac format (/), so adapt them accordingly to Windows (\)
- When I say “restart hugo server”, that means doing a `Control-C` then re-running `hugo server`
- The words “templates” and “layouts” will be used interchangeably, AKA I’m talking about the same thing
- In normal hugo site development workflow, it’s common to check the file tree of the `public` folder to verify the generated site is what you expect. To keep this tutorial concise, I won’t be doing that, but it’s a habit I encourage you to start

Create the Site

Okay, let’s generate a site skeleton:

```
hugo new site hugo-blog
cd hugo-blog
```

The file tree should look like:

```
├─ archetypes
│   └─ default.md
├─ config.toml
```

```
├─ content
├─ data
├─ layouts
├─ static
└─ themes
```

We won’t be using the `data` or `themes` folders in this tutorial, so go ahead and delete them.

At this point, if you run `hugo server`, you should get output like this:

```
Building sites ... WARN 2019/09/11 23:57:44 found no layout
WARN 2019/09/11 23:57:44 found no layout file for "HTML"
WARN 2019/09/11 23:57:44 found no layout file for "HTML"
```

	EN
Pages	3
Paginator pages	0
Non-page files	0
Static files	0
Processed images	0
Aliases	0
Sitemaps	1
Cleaned	0

Total in 18 ms

Watching for changes in /home/zwbetz/tmp/tmp-site/{arche

```
Watching for config changes in /home/zwbetz/tmp/tmp-site
Environment: "development"
Serving pages from memory
Running in Fast Render Mode. For full rebuilds on change
Web Server is available at http://localhost:1313/ (bind
Press Ctrl+C to stop
```

Don't let the warnings scare you. They're good, as they let us know what needs fixin'. By the end, they'll be gone.

Homepage Layout

Let's make the [homepage template](#). Create file `layouts/index.html`, with this content:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, i
    <link rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/4
      integrity="sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9Dk
      crossorigin="anonymous">
    <title>{{ .Title }}</title>
  </head>
  <body>
    <div class="container">
```

```
<main id="main">
  <div id="home-jumbotron" class="jumbotron text-c
    <h1>{{ .Title }}</h1>
    <p class="font-125">{{ .Site.Params.homeText |
  </div>
</main>
</div>
</body>
</html>
```

Then edit `config.toml`. Change the `title` and add the first param:

```
baseUrl = "http://example.org/"
languageCode = "en-us"
title = "Hugo Blog"

[params]
  homeText = "You just made a Hugo blog from scratch."
```

Run `hugo server`. The *Found no layout for "home"* warning should be gone. Then navigate to `http://localhost:1313/` and you'll see the beginnings of your homepage.

Notice how we grabbed data – `.Title` and `.Site.Params.homeText` in this case – from our [config](#) file into our template? This will be a pattern throughout the tutorial.

Also, we're piping the home text through the [markdownify function](#), which runs it through the markdown processor.

The static Folder

I usually prefer to save off CSS and JS locally instead of getting it from a CDN, so let's create file

`static/css/bootstrap.min.css` by pasting in [this CSS](#).

In the homepage template, replace this code:

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1
integrity="sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKmp6
crossorigin="anonymous">
```

With this code:

```
{{ $css := "css/bootstrap.min.css" | relURL }}
<link rel="stylesheet" href="{{ $css }}">
```

Keep in mind that everything under the [static folder](#) gets copied as-is to the root of our site. So:

```
static/css/bootstrap.min.css
```

Becomes:

```
css/bootstrap.min.css
```

In the new CSS code, we declare a variable with the path to the CSS file, then pipe it to `relURL`, which creates a [relative URL](#).

With hugo server still running, check the homepage and confirm it looks the same.

single Page Layout

Before creating our first blog post, we need a layout for [single pages](#), else our blog post will not “know” how to display itself.

Create file `layouts/_default/single.html` :

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, i
    {{ $css := "css/bootstrap.min.css" | relURL }}
    <link rel="stylesheet" href="{{ $css }}">
    <title>{{ .Title }}</title>
  </head>
  <body>
```

```
<div class="container">
  <main id="main">
    <h1>{{ .Title }}</h1>
    {{ .Content }}
  </main>
</div>
</body>
</html>
```

Create the First Blog Post

A little more prep work: let's edit the default [archetype](#) file, which lives at `archetypes/default.md`. To keep it from tripping us up, let's replace `draft: true` with `draft: false`. Then add a line above for tags, `tags: []`, which we'll use later. The file should now look like:

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

By default, a blog post [permalink](#) will be `/blog/:filename:`. Let's change this to be just the filename. Our `config.toml`

should now look like:

```
baseUrl = "http://example.org/"
languageCode = "en-us"
title = "Hugo Blog"

[permalinks]
  blog = "/:filename/"

[params]
  homeText = "You just made a Hugo blog from scratch."
```

Okay cool. Now we're ready to create our first post. Stop hugo server, then run:

```
hugo new blog/if-by-rudyard-kipling.md
```

Open the newly created post, located at `content/blog/if-by-rudyard-kipling.md`, and add `tags: ["poetry", "life"]`. It won't do anything now, but will be used later.

Then paste in the famous Rudyard Kipling poem, **If**. It should now look like (your `date` will be different):

```
---
title: "If by Rudyard Kipling"
```

date: 2019-01-12T22:14:57-06:00

tags: ["poetry", "life"]

draft: false

If you can keep your head when all about you
Are losing theirs and blaming it on you,
If you can trust yourself when all men doubt you,
But make allowance for their doubting too;
If you can wait and not be tired by waiting,
Or being lied about, don't deal in lies,
Or being hated, don't give way to hating,
And yet don't look too good, nor talk too wise:

If you can dream—and not make dreams your master;
If you can think—and not make thoughts your aim;
If you can meet with Triumph and Disaster
And treat those two impostors just the same;
If you can bear to hear the truth you've spoken
Twisted by knaves to make a trap for fools,
Or watch the things you gave your life to, broken,
And stoop and build 'em up with worn-out tools:

If you can make one heap of all your winnings
And risk it on one turn of pitch-and-toss,
And lose, and start again at your beginnings
And never breathe a word about your loss;
If you can force your heart and nerve and sinew
To serve your turn long after they are gone,
And so hold on when there is nothing in you
Except the Will which says to them: 'Hold on!'

If you can talk with crowds and keep your virtue,
Or walk with Kings—nor lose the common touch,
If neither foes nor loving friends can hurt you,
If all men count with you, but none too much;
If you can fill the unforgiving minute
With sixty seconds' worth of distance run,
Yours is the Earth and everything that's in it,
And—which is more—you'll be a Man, my son!

Restart hugo server then navigate to

`http://localhost:1313/if-by-rudyard-kipling/` to see the
blog post.

baseof Layout

Our homepage and single page layouts repeat a lot of code. Wouldn't it be nice if there was a way to share the repeated code, then only have the unique code in each layout? Well, there is, and it's known as the [baseof](#) layout.

So let's create one at `layouts/_default/baseof.html`:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, i
```

```

{{ $css := "css/bootstrap.min.css" | relURL }}
<link rel="stylesheet" href="{{ $css }}">
<title>{{ .Title }}</title>
</head>
<body>
  <div class="container">
    <main id="main">
      {{ block "main" . }}{{ end }}
    </main>
  </div>
</body>
</html>

```

The important piece above is `{{ block "main" . }}{{ end }}`, which basically says “unique code from other layouts will go here”. Notice the dot (`.`)? It’s important. [The dot is how you pass context around](#) in Hugo. Also see this [article](#) by Regis Philibert.

Now let’s update our other layouts. The homepage layout at `layouts/index.html` should now look like:

```

{{ define "main" }}

<div id="home-jumbotron" class="jumbotron text-center">
  <h1>{{ .Site.Title }}</h1>
  <p class="font-125">{{ .Site.Params.homeText | markdown

```

```

</div>

{{ end }}

```

And the single page layout at `layouts/_default/single.html` should now look like:

```

{{ define "main" }}

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

{{ end }}

```

Everything within `{{ define "main" }}` and the closing `{{ end }}` will be the unique code for a given layout. Much cleaner, eh?

Partials

Since we’re cleaning house, let’s talk about [partials](#). The docs sum them up nicely:

Partials are smaller, context-aware components in your list and page templates that can be used economically to keep your templating DRY (Don’t Repeat Yourself).

Partials are useful for factoring out repeated code blocks. They're also useful for keeping your code organized and easy-to-read.

So let's make one for our `<head>` code. Create file `layouts/partials/head.html` :

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, ini
  {{ $css := "css/bootstrap.min.css" | relURL }}
  <link rel="stylesheet" href="{{ $css }}">
  <title>{{ .Title }}</title>
</head>
```

Then update `layouts/_default/baseof.html` to reference it:

```
<!doctype html>
<html lang="en">
  {{ partial "head.html" . }}
  <body>
    <div class="container">
      <main id="main">
        {{ block "main" . }}{{ end }}
      </main>
    </div>
  </body>
```

```
</html>
```

list Page Layout

Since a list with one thing would be sad, let's create two more blog posts. Stop hugo server, then run:

```
hugo new blog/that-it-will-never-come-again-by-emily-dic
```

Paste the poem **That it will never come again** by Emily

Dickinson, then add the `poetry` and `time` tags. Your `date` will be different:

```
---
title: "That it will never come again by Emily Dickinson"
date: 2019-01-12T23:24:54-06:00
tags: ["poetry", "time"]
draft: false
---
```

That it will never come again
Is what makes life so sweet.
Believing what we don't believe
Does not exhilarate.

That if it be, it be at best


```
An ablative estate --  
This instigates an appetite  
Precisely opposite.
```

And another:

```
hugo new blog/trees-by-joyce-kilmer.md
```

Paste the poem **Trees** by Joyce Kilmer, then add the `poetry` and `trees` tags. Your `date` will be different:

```
---  
title: "Trees by Joyce Kilmer"  
date: 2019-01-13T20:28:42-06:00  
tags: ["poetry", "trees"]  
draft: false  
---
```

I think that I shall never see
A poem lovely as a tree.

A tree whose hungry mouth is prest
Against the earth's sweet flowing breast;

A tree that looks at God all day,
And lifts her leafy arms to pray;

A tree that may in Summer wear
A nest of robins in her hair;

Upon whose bosom snow has lain;
Who intimately lives with rain.

Poems are made by fools like me,
But only God can make a tree.

Okay cool, now let's make the [list](#) layout. Create file `layouts/_default/list.html`:

```
{{ define "main" }}  
  
<h1>{{ .Title }}</h1>  
{{ range .Pages.ByPublishDate.Reverse }}  
<p>  
    <a class="font-125" href="{{ .RelPermalink }}">{{ .Title }}</a>  
</p>  
{{ end }}  
  
{{ end }}
```

In the above, we list all pages by publish date in reverse, then make the title into a hyperlink. Start hugo server then navigate to `http://localhost:1313/blog/` and test it out.

Notice how the first usage of `.Title`, which generates the title `Blogs`, is inferred from the content path of `content/blog` (we will change this behavior in a later section). The second usage of `.Title` is inside a `range` statement, so it will grab the title *from each page* in the list.

Menu

It's inconvenient to manually type the URL of the page we want, so let's make a [menu](#), also known as a *nav* or *navigation*. We'll use a partial for this.

Create file `layouts/partials/nav.html` :

```
<div id="nav-border" class="container">
  <nav id="nav" class="nav justify-content-center">
    {{ range .Site.Menus.main }}
      {{ $icon := printf "<i data-feather=\"%s\"></i>" .Pr
      {{ $text := print $icon " " .Name | safeHTML }}
      <a class="nav-link" href="{{ .URL }}">{{ $text }}</a
    {{ end }}
  </nav>
</div>
```

The above code loops through menu data specified in the config file. It grabs the icon name, link text, and URL, then builds each

nav link element. The [print](#) and [printf](#) functions are great for [string manipulation](#).

Let's not forget to add a reference to `layouts/_default/baseof.html` :

```
<!doctype html>
<html lang="en">
  {{ partial "head.html" . }}
  <body>
    {{ partial "nav.html" . }}
    <div class="container">
      <main id="main">
        {{ block "main" . }}{{ end }}
      </main>
    </div>
  </body>
</html>
```

The nav is built from config file data, so update your `config.toml` to be:

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

[permalinks]
```

```

blog = "[:filename/"

# See https://feathericons.com/
# The value of pre is the icon name
[menu]
  [[menu.main]]
    name = "Home"
    pre = "home"
    url = "/"
    weight = 1
  [[menu.main]]
    name = "Blog"
    pre = "edit"
    url = "/blog/"
    weight = 2
  [[menu.main]]
    name = "Tags"
    pre = "tag"
    url = "/tags/"
    weight = 3

[params]
  homeText = "You just made a Hugo blog from scratch."

```

Each nav link has an associated icon. The `pre` value is the icon name. See [feather icons](https://feathericons.com/) for the full list of icons – which, as the site says, are indeed simply beautiful.

The icons are sourced from a JS file, so we'll need that. Create file `static/js/feather.min.js` then paste in [this JS](#). We'll need to reference this JS, so let's make a partial just for it.

Create file `layouts/partials/script.html` :

```

{{ $js := "js/feather.min.js" | relURL }}
<script src="{{ $js }}"></script>
<script>
  feather.replace();
</script>

```

Then reference it in `layouts/_default/baseof.html` :

```

<!doctype html>
<html lang="en">
  {{ partial "head.html" . }}
  <body>
    {{ partial "nav.html" . }}
    <div class="container">
      <main id="main">
        {{ block "main" . }}{{ end }}
      </main>
    </div>
    {{ partial "script.html" . }}
  </body>
</html>

```

Navigate to `http://localhost:1313/`, then give the menu a test run.

Tweak Blog List Layout Title

Currently, the title of `http://localhost:1313/blog/` is `Blogs`. This is intended behavior because Hugo pluralizes the list title according to a set of common English pluralization rules. We don't want this, though, so let's change it to `Blog`.

Create file `content/blog/_index.md`:

```
---
title: Blog
---
```

The leading underscore in `_index.md` is important. It differentiates a branch bundle vs a page bundle. Read more about this in the [page bundle docs](#).

Now navigate to `http://localhost:1313/blog/` to see the change.

Date and Tags Partial

Let's add the date and tags to each post. We want it to show on the blog list page, and the single page for each post, so we'll

use a partial.

Create file `layouts/partials/date-and-tags.html`:

```
{{ $dateTime := .PublishDate.Format "2006-01-02" }}
{{ $dateFormat := .Site.Params.dateFormat | default "Jan" }}
<i data-feather="calendar"></i> <time datetime="{{ $dateTime }}">{{ $dateTime }}</time>
{{ with .Params.tags }}
  <br>
  <i data-feather="tag"></i>
  {{ range . }}
  {{ $href := print (relURL "tags/") (urlize .) }}
  <a class="btn btn-sm btn-outline-dark tag-btn" href="{{ $href }}">{{ . }}</a>
  {{ end }}
{{ end }}
```

By default, hugo uses these [taxonomies](#):

```
[taxonomies]
  category = "categories"
  tag = "tags"
```

We only want to use tags, so add this to `config.toml`:

```
[taxonomies]
  tag = "tags"
```

The `date-and-tags.html` partial pulls a date [format](#) from the config file – if the date format value is not specified, it [defaults](#) to `Jan 2, 2006`. So update `config.toml` again with the `dateFormat` param.

```
[params]
dateFormat = "Jan 2, 2006"
homeText = "You just made a Hugo blog from scratch."
```

Also, you may be wondering why we used `.PublishDate` instead of `.Date`. We did this because it gives us the [best of both worlds](#): if `.PublishDate` is not set in the page front matter, then it falls back to `.Date`.

Okay, now update `layouts/_default/single.html` to reference the partial:

```
{{ define "main" }}

<h1>{{ .Title }}</h1>
{{ partial "date-and-tags.html" . }}
<br><br>
{{ .Content }}

{{ end }}
```

And update `layouts/_default/list.html` to reference the partial as well:

```
{{ define "main" }}

<h1>{{ .Title }}</h1>
{{ range .Pages.ByPublishDate.Reverse }}
<p>
  <a class="font-125" href="{{ .RelPermalink }}">{{ .Title }}</a>
  <br>
  {{ partial "date-and-tags.html" . }}
</p>
{{ end }}

{{ end }}
```

Now navigate to `http://localhost:1313/blog/` to see the changes.

terms List Layout

Let's customize the way our tags (terms) are displayed on `http://localhost:1313/tags/`. They currently show a date, which we don't want. So instead we'll display a "count" next to each tag, showing how many times it's been used. The tags will be ordered alphabetically.

Create file `layouts/_default/terms.html` :

```
{{ define "main" }}

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

{{ range .Data.Terms.Alphabetical }}
<p>
  <a class="btn btn-outline-dark font-125" href="{{ .PageURL }}">
    <span class="badge badge-dark">{{ .Count }}</span> {{ .PageURL }}
  </a>
</p>
{{ end }}

{{ end }}
```

Navigate to `http://localhost:1313/tags/` to checkout the new tags listing.

Style Tweaks

Let's add a few style tweaks with some custom CSS. Create file `layouts/partial/style.html` :

```
<style>
.container {
  max-width: {{ .Site.Params.containerMaxWidth | default 1000px }};
}
```

```
}
#nav a {
  font-weight: bold;
  color: inherit;
}
#nav-border {
  border-bottom: 1px solid #212529;
}
#main {
  margin-top: 1em;
  margin-bottom: 4em;
}
#home-jumbotron {
  background-color: inherit;
}
.font-125 {
  font-size: 125%;
}
.tag-btn {
  margin-bottom: 0.3em;
}
img {
  max-width: 100%;
}
</style>
```

Then add a reference to it in `layouts/partial/head.html` :

```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, ini
  {{ $css := "css/bootstrap.min.css" | relURL }}
  <link rel="stylesheet" href="{{ $css }}">
  <title>{{ .Title }}</title>
  {{ partial "style.html" . }}
</head>
```

If you would rather not use an internal stylesheet, Hugo's templating features can also be used with an [external stylesheet](#).

Now navigate to any page to see the style changes.

Better Title Logic

Currently, we're just showing the page title in the `<title>` element. Let's make it so that the homepage only shows the site title, then other pages show both the page title and the site title.

Also, let's add `{{ hugo.Generator }}` so that the hugo version used to generate the site shows in the HTML source.

Update `layouts/partials/head.html` to be:

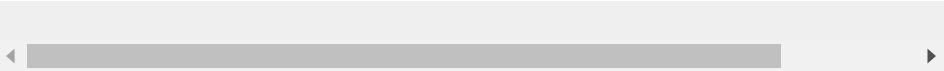
```
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, ini
  {{ hugo.Generator }}
  {{ $css := "css/bootstrap.min.css" | relURL }}
  <link rel="stylesheet" href="{{ $css }}">
  {{ $title := print .Title " | " .Site.Title }}
  {{ if .IsHome }}{{ $title = .Site.Title }}{{ end }}
  <title>{{ $title }}</title>
  {{ partial "style.html" . }}
</head>
```

Navigate to a few pages and glance at the browser tab to see the changes.

Wrap Up

If you stop hugo server and just run `hugo`, which will generate your site to the `public` folder, your full project file tree should now look like:

```
├─ archetypes
│   └─ default.md
├─ config.toml
├─ content
│   └─ blog
│       ├── if-by-rudyard-kipling.md
│       └─ _index.md
```

There you go, you just made a Hugo blog from scratch. Don't forgot to refer back to the [finished product source code](#) if you missed anything along the way.

I hope this tutorial was helpful. Hugo can do *many* things, so it's near impossible to cover all the possibilities in one tutorial. Well then, what's next? I recommend a deep dive into the [docs](#). As for where to deploy your site, since Hugo is a static site generator, you can host the generated files just about anywhere. I personally use Netlify, but do checkout the [hosting and deployment](#) docs as well.

Good luck, have fun, and go build that personal site you've been meaning to do :)

Emails I've Received

The following are some of the lovely emails I've received about this post.

Your article should be included in the docs. It was exactly what I was looking for since I was struggling just to display navigation links on the page. Thank you for writing this guide. It was structured like the Jekyll quickstart guide and I really liked it for that reason.

— MrSimsek

Can't thank you enough for this! That's a lot of well organized work friend.

— theadle

Thank you so much for this. Amazing contribution!

— nnise

I've been meaning to write for a few years now, hopefully this should get me liftoff.

— lj888

Thanks for writing this tutorial. Stuff like this is very helpful for beginners.

— pawsys

I wanted to drop you a quick line to thank you for the tutorial on creating a Hugo site/theme from scratch. I've been using Hugo on and off for my site for the last three years and currently use a provided theme (Blackburn). I've searched for and worked my way through the few tutorials on creating a theme that I've come across and want to let you know that I think yours is both the most comprehensive and clearest one I've used.

— Gary

Thanks a ton for that blog post. It's helped me quite a bit.

— Rex

I found Hugo a few days ago and felt like I found just what I was looking for. Sadly, I attacked it through emacs using ox-hugo which added a good bit of abstraction hiding all the issues going on. So I took a step back and looked for more how-to's on Hugo. After rifling through much junk, I found your Hugo from Scratch article. Thank You ! GREAT write-up. Smart.

— Paul

I just clicked on a few of your links and realized that you wrote the most useful Hugo resource that I found! Thank you for writing the guide to building a Hugo blog from scratch! I learned a lot there, now I just need to wrap up this quick project. I really appreciate your assistance!

— Noah

Hey there. Thanks for this up-to-date and comprehensive tutorial. Helped me a lot to kickstart my gamedev blog!

— Gabriel

Thanks for your tutorial, it has helped me a lot.

— Andrew

Good job on this, very detailed walkthrough. Came here from the gohugo forums

— ljem

Thanks a lot for this super nice tutorial! This is what I needed :-)

— zig

Thanks for this great tutorial :)

— Jaya

Great tutorial. I can now build my own theme. Thanks!

— hallazzang

Hello, a very detailed tutorial. Thank you for sharing the same.

— Raghavan

Thanks very much for the Hugo tutorial on your site. I'm working on bootstrapping myself into this system - I actually wrote something conceptually very similar in Perl in the mid-90s, for generating static sites from template files and embedded code, though it didn't have a server mode and was nowhere NEAR as fast :P Anyway, I've also struggled with the state of some of the official documentation as has been discussed on the forums, so I know it's a concern. Your document has been the clearest and most helpful for me out of everything I've read so far, including the official docs on the Hugo site.

— Steve

I just wanted to let you know that I found your blog post this past weekend and that it was insanely helpful for me, like more than anything else I found. A lot of the other tutorials I found were more in the format “build a website with Hugo + [some web technology]” and they tend to focus more on using the tools but not really why I needed them. Put this in your config file, add this CLI tool, etc. As someone who knew absolutely nothing about the web stack I didn’t understand at all why I needed five different tools just to make a website. Anyway, I appreciated how your post was 100% laser focused on *Hugo* and was verbose enough for me to follow along and take ideas from. Thats all I wanted to say, thanks!

— Eric

Hey there, Just wanted to say that I really appreciated the Hugo tutorial. I have had a long history with Jekyll, but only ever at a basic level. I just got a new Mac M1 and have nothing but trouble getting Ruby, rbenv, RVM or asdf working well enough to start developing in Jekyll again. After a week of real frustration and having to reinstall everything (I completely borked my system after numerous attempts at getting ANY kind of Ruby working for Jekyll), I decided to look around at other SSGs. I found Hugo, but the documentation was lacking something and I was struggling to get a handle on the basics. After following your introduction, I have a solid understanding of the basics, and

much of the detail on the Hugo Docs makes a ton more sense. Appreciate the work you put into it and made it available for people like me. Cheers!

— Alistair

I appreciate your blog and it has helped me a lot. I am currently developing in Hugo right now and your post on developing hugo from scratch really got me started. Most of the other tutorials were very cryptic. You laid it out more effectively so I was able to put something out. Thank you for finding the time to write so many great things.

— Scott

Built With

License

See an Issue?