What's on this Page

CONTENT MANAGEMENT    FUNDAMENTALS

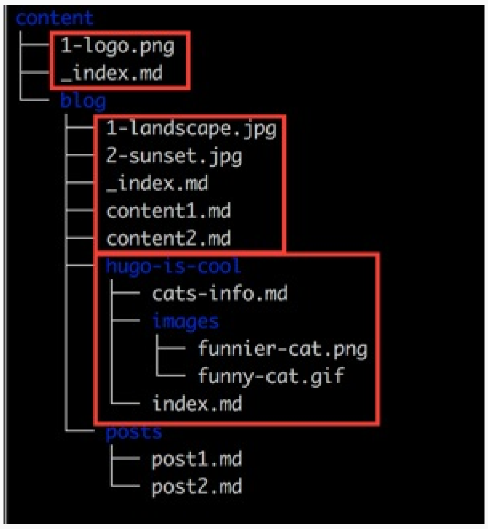
# Content Organization

Hugo assumes that the same structure that works to organize your source content is used to organize the rendered site.

## Page Bundles

Hugo `0.32` announced page-relative images and other resources packaged into `Page Bundles`.

These terms are connected, and you also need to read about Page Resources and Image Processing to get the full picture.



```
content
├── 1-logo.png
├── _index.md
└── blog
    ├── 1-landscape.jpg
    ├── 2-sunset.jpg
    ├── _index.md
    ├── content1.md
    ├── content2.md
    ├── hugo-is-cool
    │   ├── cats-info.md
    │   ├── images
    │   │   ├── funnier-cat.png
    │   │   └── funny-cat.gif
    │   └── index.md
    └── posts
        ├── post1.md
        └── post2.md
```

The illustration shows 3 bundles. Note that the home page bundle cannot contain other content pages, but other files (images etc.) are fine.

> The bundle documentation is **work in progress**. We will publish more comprehensive docs about this soon.

## Organization of Content Source

In Hugo, your content should be organized in a manner that reflects the rendered website.

While Hugo supports content nested at any level, the top levels (i.e. content/<DIRECTORIES>) are special in Hugo and are considered the content type used to determine layouts etc. To read more about sections, including how to nest them, see sections.

Without any additional configuration, the following will just work:

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

## Path Breakdown in Hugo

The following demonstrates the relationships between your content organization and the output URL structure for your Hugo website when it renders. These examples assume you are using pretty URLs, which is the default behavior for Hugo. The examples also assume a key-value of `baseURL = "https://example.com"` in your site's configuration file.

### Index Pages: `_index.md`

`_index.md` has a special role in Hugo. It allows you to add front matter and content to your list templates. These templates include those for section templates, taxonomy templates, taxonomy terms templates, and your homepage template.

> **Tip:** You can get a reference to the content and metadata in `_index.md` using the .Site.GetPage function.

You can keep one `_index.md` for your homepage and one in each of your content sections, taxonomies, and taxonomy terms. The following shows typical placement of an `_index.md` that would contain content and front matter for a `posts` section list page on a Hugo website:

```
.         url
.       ⊢--^-⊣
.        path    slug
.       ⊢--^-⊣⊢---^---⊣
.           filepath
.       ⊢------^------⊣
content/posts/_index.md
```

At build, this will output to the following destination with the associated values:

```

                     url ("/posts/")
                    ⊢-^-⊣
       baseurl      section ("posts")
⊢--------^---------⊣⊢-^-⊣
        permalink
⊢----------^-------------⊣
https://example.com/posts/index.html
```

The sections can be nested as deeply as you need. The important part to understand is, that to make the section tree fully navigational, at least the lower-most section needs a content file. (i.e. _index.md).

## Single Pages in Sections

Single content files in each of your sections are going to be rendered as single page templates. Here is an example of a single post within posts:

```
                   path ("posts/my-first-hugo-post.md")
.       ⊢-----------^------------⊣
.      section        slug
.       ⊢-^-⊣⊢--------^----------⊣
content/posts/my-first-hugo-post.md
```

When Hugo builds your site, the content will be outputted to the following destination:

```
                       url ("/posts/my-first-hugo-post/")
                   ⊢------------^----------⊣
        baseurl      section      slug
⊢--------^--------⊣⊢-^--⊣⊢-------^---------⊣
                  permalink
⊢-------------------^---------------------⊣
https://example.com/posts/my-first-hugo-post/index.html
```

## Paths Explained

The following concepts will provide more insight into the relationship between your project's organization and the default behaviors of Hugo when building the output website.

### section

A default content type is determined by a piece of content's section. `section` is determined by the location within the project's `content` directory. `section` *cannot* be specified or overridden in front matter.

### slug

A content's `slug` is either `name.extension` or `name/`. The value for `slug` is determined by

- the name of the content file (e.g., `lollapalooza.md`) OR
- front matter overrides

### path

A content's `path` is determined by the section's path to the file. The file `path`

- is based on the path to the content's location AND
- does not include the slug

### url

The `url` is the relative URL for the piece of content. The `url`

- is based on the content's location within the directory structure OR
- is defined in front matter and *overrides all the above*

## Override Destination Paths via Front Matter

Hugo believes that you organize your content with a purpose. The same structure that works to organize your source content is used to organize the rendered site. As displayed above, the organization of the source content will be mirrored in the destination.

There are times where you may need more control over your content. In these cases, there are fields that can be specified in the front matter to determine the destination of a specific piece of content.

The following items are defined in this order for a specific reason: items explained further down in the list will override earlier items, and not all of these items can be defined in front matter:

### filename

This isn't in the front matter, but is the actual name of the file minus the extension. This will be the name of the file in the destination (e.g., `content/posts/my-post.md` becomes `example.com/posts/my-post/`).

### slug

When defined in the front matter, the `slug` can take the place of the filename for the destination.

content/posts/old-post.md

```
---
title: A new post with the filename old-post.md
slug: "new-post"
---
```

This will render to the following destination according to Hugo's default behavior:

```
example.com/posts/new-post/
```

### section

`section` is determined by a content's location on disk and *cannot* be specified in the front matter. See sections for more information.

### type

A content's `type` is also determined by its location on disk but, unlike `section`, it *can* be specified in the front matter. See types. This can come in especially handy when you want a piece of content to render using a different layout. In the following example, you can create a layout at `layouts/new/mylayout.html` that Hugo will use to render this piece of content, even in the midst of many other posts.

content/posts/my-post.md

```
---
title: My Post
type: new
layout: mylayout
---
```

### url

A complete URL can be provided. This will override all the above as it pertains to the end destination. This must be the path from the baseURL (starting with a `/`). `url` will be used exactly as it provided in the front matter and will ignore the `--uglyURLs` setting in your site configuration:

content/posts/old-url.md

```
---
title: Old URL
url: /blog/new-url/
---
```

Assuming your `baseURL` is configured to `https://example.com`, the addition of `url` to the front matter will make `old-url.md` render to the following destination:

```
https://example.com/blog/new-url/
```

You can see more information on how to control output paths in URL Management.

## See Also

- Comments
- Page Resources
- Content Sections
- Content Types
- Related Content

“Content Organization” was last updated: March 25, 2021: Fix header level (4c75c2422)

IMPROVE THIS PAGE