Configuration

Improve this page

Jekyll allows you to concoct your sites in any way you can dream up, and it's thanks to the powerful and flexible configuration options that this is possible. These options can either be specified in a `_config.yml` file placed in your site's root directory, or can be specified as flags for the `jekyll` executable in the terminal.

Configuration Settings

Global Configuration

The table below lists the available settings for Jekyll, and the various `options` (specified in the configuration file) and `flags` (specified on the command-line) that control them.

SETTING	OPTIONS AND FLAGS
Site Source Change the directory where Jekyll will read files	`source: DIR` `-s, --source DIR`
Site Destination Change the directory where Jekyll will write files	`destination: DIR`

SETTING	OPTIONS AND FLAGS
	`-d, --destination DIR`
Safe Disable custom plugins, and ignore symbolic links.	`safe: BOOL` `--safe`
Exclude Exclude directories and/or files from the conversion. These exclusions are relative to the site's source directory and cannot be outside the source directory.	`exclude: [DIR, FILE, ...]`
Include Force inclusion of directories and/or files in the conversion. `.htaccess` is a good example since dotfiles are excluded by default.	`include: [DIR, FILE, ...]`
Keep files When clobbering the site destination, keep the selected files. Useful for files that are not generated by jekyll; e.g. files or assets that are generated by your build tool. The paths are relative to the `destination`.	`keep_files: [DIR, FILE, ...]`
Time Zone Set the time zone for site generation. This sets the `TZ` environment variable, which Ruby uses to handle time and date creation and manipulation. Any entry from the IANA Time Zone Database is valid, e.g. `America/New_York`. A list of all available values can be found here. The default is the local time zone, as set by your operating system.	`timezone: TIMEZONE`
Encoding Set the encoding of files by name. Only available for Ruby 1.9 or later). The default value is `utf-8` starting in 2.0.0, and `nil` before 2.0.0, which will yield the Ruby default of `ASCII-8BIT`. Available encodings can be shown by the command `ruby -e 'puts Encoding::list.join("\n")'`	`encoding: ENCODING`

SETTING	OPTIONS AND FLAGS
.	
Defaults Set defaults for YAML Front Matter variables.	see below

!! Destination folders are cleaned on site builds

The contents of `<destination>` are automatically cleaned, by default, when the site is built. Files or folders that are not created by your site will be removed. Some files could be retained by specifying them within the `<keep_files>` configuration directive. Do not use an important location for `<destination>` ; instead, use it as a staging area and copy files from there to your web server.

Build Command Options

SETTING	OPTIONS AND FLAGS
Regeneration Enable auto-regeneration of the site when files are modified.	`-w, --[no-]watch`
Configuration Specify config files instead of using `_config.yml` automatically. Settings in later files override settings in earlier files.	`--config FILE1[,FILE2,...]`
Drafts Process and render draft posts.	`--drafts`
Environment Use a specific	`JEKYLL_ENV=production`

SETTING	OPTIONS AND FLAGS
environment value in the build.	
Future Publish posts with a future date.	`future: BOOL` `--future`
LSI Produce an index for related posts.	`lsi: BOOL` `--lsi`
Limit Posts Limit the number of posts to parse and publish.	`limit_posts: NUM` `--limit_posts NUM`
Force polling Force watch to use polling.	`--force_polling`
Verbose output Print verbose output.	`-V, --verbose`
Silence Output Silence the normal output from Jekyll during a build	`-q, --quiet`
Incremental build Enable the experimental incremental build feature. Incremental build only re-builds posts and pages that have changed, resulting in significant performance improvements for large sites, but may also break site generation in certain cases.	`incremental: BOOL` `-I, --incremental`

Serve Command Options

In addition to the options below, the `serve` sub-command can accept any of the options for the `build` sub-command, which are then applied to the site build which occurs right before your site is served.

SETTING	OPTIONS AND FLAGS
Local Server Port Listen on the given port.	`port: PORT` `--port PORT`
Local Server Hostname Listen at the given hostname.	`host: HOSTNAME` `--host HOSTNAME`
Base URL Serve the website from the given base URL	`baseurl: URL` `--baseurl URL`
Detach Detach the server from the terminal	`detach: BOOL` `-B, --detach`
Skips the initial site build. Skips the initial site build which occurs before the server is started.	`--skip-initial-build`

Do not use tabs in configuration files

This will either lead to parsing errors, or Jekyll will revert to the default settings. Use spaces instead.

Specifying a Jekyll environment

at build time

In the build (or serve) arguments, you can specify a Jekyll environment and value. The build will then apply this value in any conditional statements in your content.

For example, suppose you set this conditional statement in your code:

```
{% if jekyll.environment == "production" %}
   {% include disqus.html %}
{% endif %}
```

When you build your Jekyll site, the content inside the `if` statement won't be run unless you also specify a `production` environment in the build command, like this:

```
JEKYLL_ENV=production jekyll build
```

Specifying an environment value allows you to make certain content available only within specific environments.

The default value for `JEKYLL_ENV` is `development`. Therefore if you omit `JEKYLL_ENV` from the build arguments, the default value will be `JEKYLL_ENV=development`. Any content inside `{% if jekyll.environment == "development" %}` tags will automatically appear in the build.

Your environment values can be anything you want (not just `development` or `production`). Some elements you might want to hide in development environments include Disqus comment forms or Google Analytics. Conversely, you might want to expose an "Edit me in GitHub" button in a development environment but not include it in production environments.

By specifying the option in the build command, you avoid having to change values in your configuration files when moving from one environment to another.

Front Matter defaults

Using YAML Front Matter is one way that you can specify configuration in the pages and posts for your site. Setting things like a default layout, or customizing the title, or specifying a more precise date/time for the post can all be added to your page or post front matter.

Often times, you will find that you are repeating a lot of configuration options. Setting the same layout in each file, adding the same category - or categories - to a post, etc. You can even add custom variables like author names, which might be the same for the majority of posts on your blog.

Instead of repeating this configuration each time you create a new post or page, Jekyll provides a way to set

these defaults in the site configuration. To do this, you can specify site-wide defaults using the `defaults` key in the `_config.yml` file in your projects root directory.

The `defaults` key holds an array of scope/values pairs that define what defaults should be set for a particular file path, and optionally, a file type in that path.

Let's say that you want to add a default layout to all pages and posts in your site. You would add this to your `_config.yml` file:

```
defaults:
  -
    scope:
      path: "" # an empty string here means all f
    values:
      layout: "default"
```

Here, we are scoping the `values` to any file that exists in the scopes path. Since the path is set as an empty string, it will apply to **all files** in your project. You probably don't want to set a layout on every file in your project - like css files, for example - so you can also specify a `type` value under the `scope` key.

```
defaults:
  -
    scope:
      path: "" # an empty string here means all f
      type: "posts" # previously `post` in Jekyll
    values:
      layout: "default"
```

Now, this will only set the layout for files where the type is `posts`. The different types that are available to you are `pages`, `posts`, `drafts` or any collection in your site. While `type` is optional, you must specify a value for `path` when creating a `scope/values` pair.

As mentioned earlier, you can set multiple scope/values pairs for `defaults`.

```
defaults:
  -
    scope:
      path: ""
      type: "posts"
    values:
      layout: "my-site"
  -
    scope:
      path: "projects"
      type: "pages" # previously `page` in Jekyll
    values:
      layout: "project" # overrides previous defa
      author: "Mr. Hyde"
```

With these defaults, all posts would use the `my-site` layout. Any html files that exist in the `projects/` folder will use the `project` layout, if it exists. Those files will also have the `page.author` liquid variable set to `Mr. Hyde` as well as have the category for the page set to `project`.

```
collections:
  - my_collection:
    output: true

defaults:
  -
    scope:
      path: ""
      type: "my_collection" # a collection in you
    values:
      layout: "default"
```

In this example the `layout` is set to `default` inside the collection with the name `my_collection`.

Precedence

Jekyll will apply all of the configuration settings you specify in the `defaults` section of your `_config.yml` file. However, you can choose to override settings from other scope/values pair by specifying a more specific path for the scope.

You can see that in the last example above. First, we

set the default layout to `my-site`. Then, using a more specific path, we set the default layout for files in the `projects/` path to `project`. This can be done with any value that you would set in the page or post front matter.

Finally, if you set defaults in the site configuration by adding a `defaults` section to your `_config.yml` file, you can override those settings in a post or page file. All you need to do is specify the settings in the post or page front matter. For example:

```
# In _config.yml
...
defaults:
  -
    scope:
      path: "projects"
      type: "pages"
    values:
      layout: "project"
      author: "Mr. Hyde"
      category: "project"
...
```

```
# In projects/foo_project.md
---
author: "John Smith"
layout: "foobar"
---
The post text goes here...
```

The `projects/foo_project.md` would have the `layout` set to `foobar` instead of `project` and the `author` set to `John Smith` instead of `Mr. Hyde` when the site is built.

Default Configuration

Jekyll runs with the following configuration options by default. Alternative settings for these options can be explicitly specified in the configuration file or on the command-line.

!! There are two unsupported kramdown options

Please note that both `remove_block_html_tags` and `remove_span_html_tags` are currently unsupported in Jekyll due to the fact that they are not included within the kramdown HTML converter.

```
# Where things are
source:       .
destination:  ./_site
plugins_dir:  ./_plugins
layouts_dir:  ./_layouts
data_dir:     ./_data
includes_dir: ./_includes
collections:  null

# Handling Reading
safe:         false
include:      [".htaccess"]
exclude:      []
keep_files:   [".git", ".svn"]
encoding:     "utf-8"
markdown_ext: "markdown,mkdown,mkdn,mkd,md"

# Filtering Content
show_drafts: null
limit_posts: 0
future:      false
unpublished: false

# Plugins
whitelist: []
gems:      []

# Conversion
markdown:    kramdown
highlighter: rouge
lsi:         false
excerpt_separator: "\n\n"
incremental: false
```

Markdown Options

The various Markdown renderers supported by Jekyll sometimes have extra options available.

Redcarpet

Redcarpet can be configured by providing an `extensions` sub-setting, whose value should be an array of strings. Each string should be the name of one of the `Redcarpet::Markdown` class's extensions; if present in the array, it will set the corresponding extension to `true`.

Jekyll handles two special Redcarpet extensions:

- `no_fenced_code_blocks` — By default, Jekyll sets the `fenced_code_blocks` extension (for delimiting code blocks with triple tildes or triple backticks) to `true`, probably because GitHub's eager adoption of them is starting to make them inescapable. Redcarpet's normal `fenced_code_blocks` extension is inert when used with Jekyll; instead, you can use this inverted version of the extension for disabling fenced code.

Note that you can also specify a language for highlighting after the first delimiter:

```ruby
# ...ruby code
```

With both fenced code blocks and highlighter enabled, this will statically highlight the code; without any syntax highlighter, it will add a `class="LANGUAGE"` attribute to the `<code>` element, which can be used as a hint by various JavaScript code highlighting libraries.

- `smart` — This pseudo-extension turns on SmartyPants, which converts straight quotes to curly quotes and runs of hyphens to em (`---`) and en (`--`) dashes.

All other extensions retain their usual names from Redcarpet, and no renderer options aside from `smart` can be specified in Jekyll. A list of available extensions can be found in the Redcarpet README file. Make sure you're looking at the README for the right version of Redcarpet: Jekyll currently uses v3.2.x. The most commonly used extensions are:

- `tables`
- `no_intra_emphasis`
- `autolink`

Kramdown

In addition to the defaults mentioned above, you can also turn on recognition of Github Flavored

Markdown by passing an `input` option with a value of "GFM".

For example, in your `_config.yml`:

```
kramdown:
  input: GFM
```

Custom Markdown Processors

If you're interested in creating a custom markdown processor, you're in luck! Create a new class in the `Jekyll::Converters::Markdown` namespace:

```
class Jekyll::Converters::Markdown::MyCustomProce
  def initialize(config)
    require 'funky_markdown'
    @config = config
  rescue LoadError
    STDERR.puts 'You are missing a library requir
    STDERR.puts '  $ [sudo] gem install funky_mar
    raise FatalException.new("Missing dependency:
  end

  def convert(content)
    ::FunkyMarkdown.new(content).convert
  end
end
```

Once you've created your class and have it properly setup either as a plugin in the `_plugins` folder or as a gem, specify it in your `_config.yml`:

```
markdown: MyCustomProcessor
```

❮ BACK NEXT ❯