Jekyll • Simple, blog-aware, static sites

jekyllrb.com/docs/themes

October 10, 2021

Improve this page

Themes

Jekyll has an extensive theme system that allows you to leverage community-maintained templates and styles to customize your site's presentation. Jekyll themes specify plugins and package up assets, layouts, includes, and stylesheets in a way that can be overridden by your site's content.

Pick up a theme

You can find and preview themes on different galleries:

See also: resources.

Understanding gem-based themes

When you create a new Jekyll site (by running the `jekyll new <PATH>` command), Jekyll installs a site that uses a gem-based theme called Minima.

With gem-based themes, some of the site's directories (such as the `assets`, `_layouts`, `_includes`, and `_sass` directories) are stored in the theme's gem, hidden from your immediate view. Yet all of the necessary directories will be read and processed during Jekyll's build process.

In the case of Minima, you see only the following files in your Jekyll site directory:

```
.
├── Gemfile
├── Gemfile.lock
├── _config.yml
├── _posts
│   └── 2016-12-04-welcome-to-jekyll.markdown
├── about.markdown
└── index.markdown
```

The `Gemfile` and `Gemfile.lock` files are used by Bundler to keep track of the required gems and gem versions you need to build your Jekyll site.

Gem-based themes make it easier for theme developers to make updates available to anyone who has the theme gem. When there's an update, theme developers push the update to RubyGems.

If you have the theme gem, you can (if you desire) run `bundle update` to update all gems in your project. Or you can run `bundle update <THEME>`, replacing `<THEME>` with the theme name, such as `minima`, to just update the theme gem. Any new files or updates the theme developer has made (such as to stylesheets or includes) will be pulled into your project automatically.

The goal of gem-based themes is to allow you to get all the benefits of a robust, continually updated theme without having all the theme's files getting in your way and over-complicating what might be your primary focus: creating content.

Overriding theme defaults

Jekyll themes set default layouts, includes, and stylesheets. However, you can override any of the theme defaults with your own site content.

To replace layouts or includes in your theme, make a copy in your `_layouts` or `_includes` directory of the specific file you wish to modify, or create the file from scratch giving it the same name as the file you wish to override.

For example, if your selected theme has a `page` layout, you can override the theme's layout by creating your own `page` layout in the `_layouts` directory (that is, `_layouts/page.html`).

To locate a theme's files on your computer:

1. Run `bundle info --path` followed by the name of the theme's gem, e.g., `bundle info --path minima` for Jekyll's default theme.

   This returns the location of the gem-based theme files. For example, the Minima theme's files might be located in `/usr/local/lib/ruby/gems/2.6.0/gems/minima-2.5.1` on macOS.

2. Open the theme's directory in Finder or Explorer:

```
# On MacOS
open $(bundle info --path minima)

# On Windows
# First get the gem's installation path:
#
#   bundle info --path minima
#   => C:/Ruby26-x64/lib/ruby/gems/3.0.0/gems/minima-2.5.1
#
# then invoke explorer with above path, substituting `/` with `\`
explorer C:\Ruby26-x64\lib\ruby\gems\3.0.0\gems\minima-2.5.1

# On Linux
xdg-open $(bundle info --path minima)
```

   A Finder or Explorer window opens showing the theme's files and directories. The Minima theme gem contains these files:

```
.
├── LICENSE.txt
├── README.md
├── _includes
│   ├── disqus_comments.html
│   ├── footer.html
│   ├── google-analytics.html
│   ├── head.html
│   ├── header.html
│   ├── icon-github.html
│   ├── icon-github.svg
│   ├── icon-twitter.html
│   └── icon-twitter.svg
├── _layouts
│   ├── default.html
│   ├── home.html
│   ├── page.html
│   └── post.html
├── _sass
│   ├── minima
│   │   ├── _base.scss
│   │   ├── _layout.scss
│   │   └── _syntax-highlighting.scss
│   └── minima.scss
└── assets
    └── main.scss
```

With a clear understanding of the theme's files, you can now override any theme file by creating a similarly named file in your Jekyll site directory.

Let's say, for a second example, you want to override Minima's footer. In your Jekyll site, create an `_includes` folder and add a file in it called `footer.html`. Jekyll will now use your site's `footer.html` file instead of the `footer.html` file from the Minima theme gem.

To modify any stylesheet you must take the extra step of also copying the main sass file (`_sass/minima.scss` in the Minima theme) into the `_sass` directory in your site's source.

Jekyll will look first to your site's content before looking to the theme's defaults for any requested file in the following folders:

- `/assets`
- `/_layouts`
- `/_includes`
- `/_sass`

Note that making copies of theme files will prevent you from receiving any theme updates on those files. An alternative, to continue getting theme updates on all stylesheets, is to use higher specificity CSS selectors in your own additional, originally named CSS files.

Refer to your selected theme's documentation and source repository for more information on which files you can override.

Converting gem-based themes to regular themes

Suppose you want to get rid of the gem-based theme and convert it to a regular theme, where all files are present in your Jekyll site directory, with nothing stored in the theme gem.

To do this, copy the files from the theme gem's directory into your Jekyll site directory. (For example, copy them to `/myblog` if you created your Jekyll site at `/myblog`. See the previous section for details.)

Then you must tell Jekyll about the plugins that were referenced by the theme. You can find these plugins in the theme's gemspec file as runtime dependencies. If you were converting the Minima theme, for example, you might see:

```
spec.add_runtime_dependency "jekyll-feed", "~> 0.12"
spec.add_runtime_dependency "jekyll-seo-tag", "~> 2.6"
```

You should include these references in the `Gemfile` in one of two ways.

You could list them individually in both `Gemfile` and `_config.yml`.

```
# ./Gemfile

gem "jekyll-feed", "~> 0.12"
gem "jekyll-seo-tag", "~> 2.6"

# ./_config.yml

plugins:
  - jekyll-feed
  - jekyll-seo-tag
```

Or you could list them explicitly as Jekyll plugins in your Gemfile, and not update `_config.yml`, like this:

```
# ./Gemfile

group :jekyll_plugins do
  gem "jekyll-feed", "~> 0.12"
  gem "jekyll-seo-tag", "~> 2.6"
end
```

Either way, don't forget to `bundle update`.

If you're publishing on GitHub Pages you should update only your `_config.yml` as GitHub Pages doesn't load plugins via Bundler.

Finally, remove references to the theme gem in `Gemfile` and configuration. For example, to remove `minima`:

- Open `Gemfile` and remove `gem "minima", "~> 2.5"`.
- Open `_config.yml` and remove `theme: minima`.

Now `bundle update` will no longer get updates for the theme gem.

Installing a gem-based theme

The `jekyll new <PATH>` command isn't the only way to create a new Jekyll site with a gem-based theme. You can also find gem-based themes online and incorporate them into your Jekyll project.

For example, search for jekyll theme on RubyGems to find other gem-based themes. (Note that not all themes are using `jekyll-theme` as a convention in the theme name.)

To install a gem-based theme:

1. Add the theme gem to your site's `Gemfile`:

   ```
   # ./Gemfile

   # This is an example, declare the theme gem you want to use here
   gem "jekyll-theme-minimal"
   ```

   Or if you've started with the `jekyll new` command, replace `gem "minima", "~> 2.0"` with the gem you want, e.g:

   ```
   # ./Gemfile

   - gem "minima", "~> 2.5"
   + gem "jekyll-theme-minimal"
   ```

2. Install the theme:

   ```
   bundle install
   ```

3. Add the following to your site's `_config.yml` to activate the theme:

   ```
   theme: jekyll-theme-minimal
   ```

4. Build your site:

   ```
   bundle exec jekyll serve
   ```

You can have multiple themes listed in your site's `Gemfile`, but only one theme can be selected in your site's `_config.yml`.

If you're publishing your Jekyll site on GitHub Pages, note that GitHub Pages supports only some gem-based themes. GitHub Pages also supports using any theme hosted on GitHub using the `remote_theme` configuration as if it were a gem-based theme.

Creating a gem-based theme

If you're a Jekyll theme developer (rather than a consumer of themes), you can package up your theme in RubyGems and allow users to install it through Bundler.

If you're unfamiliar with creating Ruby gems, don't worry. Jekyll will help you scaffold a new theme with the `new-theme` command. Run `jekyll new-theme` with the theme name as an argument.

Here is an example:

```
jekyll new-theme jekyll-theme-awesome
    create /path/to/jekyll-theme-awesome/_layouts
    create /path/to/jekyll-theme-awesome/_includes
    create /path/to/jekyll-theme-awesome/_sass
    create /path/to/jekyll-theme-awesome/_layouts/page.html
    create /path/to/jekyll-theme-awesome/_layouts/post.html
    create /path/to/jekyll-theme-awesome/_layouts/default.html
    create /path/to/jekyll-theme-awesome/Gemfile
    create /path/to/jekyll-theme-awesome/jekyll-theme-awesome.gemspec
    create /path/to/jekyll-theme-awesome/README.md
    create /path/to/jekyll-theme-awesome/LICENSE.txt
    initialize /path/to/jekyll-theme-awesome/.git
    create /path/to/jekyll-theme-awesome/.gitignore
Your new Jekyll theme, jekyll-theme-awesome, is ready for you in /path/to/jekyll-
theme-awesome!
For help getting started, read /path/to/jekyll-theme-awesome/README.md.
```

Add your template files in the corresponding folders. Then complete the `.gemspec` and the README files according to your needs.

Layouts and includes

Theme layouts and includes work just like they work in any Jekyll site. Place layouts in your theme's `/_layouts` folder, and place includes in your themes `/_includes` folder.

For example, if your theme has a `/_layouts/page.html` file, and a page has `layout: page` in its front matter, Jekyll will first look to the site's `_layouts` folder for the `page` layout, and if none exists, will use your theme's `page` layout.

Assets

Any file in `/assets` will be copied over to the user's site upon build unless they have a file with the same relative path. You can ship any kind of asset here: SCSS, an image, a webfont, etc. These files behave like pages and static files in Jekyll:

- If the file has front matter at the top, it will be rendered.
- If the file does not have front matter, it will simply be copied over into the resulting site.

This allows theme creators to ship a default `/assets/styles.scss` file which their layouts can depend on as `/assets/styles.css`.

All files in `/assets` will be output into the compiled site in the `/assets` folder just as you'd expect from using Jekyll on your sites.

Stylesheets

Your theme's stylesheets should be placed in your theme's `_sass` folder, again, just as you would when authoring a Jekyll site.

```
_sass
└── jekyll-theme-awesome.scss
```

Your theme's styles can be included in the user's stylesheet using the `@import` directive.

```
@import "{{ site.theme }}";
```

Theme-gem dependencies3.5.0

Jekyll will automatically require all whitelisted `runtime_dependencies` of your theme-gem even if they're not explicitly included under the `plugins` array in the site's config file. (Note: whitelisting is only required when building or serving with the `--safe` option.)

With this, the end-user need not keep track of the plugins required to be included in their config file for their theme-gem to work as intended.

Pre-configuring Theme-gems4.0

Jekyll will read-in a `_config.yml` at the root of the theme-gem and merge its data into the site's existing configuration data.

But unlike other entities loaded from within the theme, loading the config file comes with a few restrictions, as summarized below:

- Jekyll's default settings cannot be overridden by a theme-config. That *ball is still in the user's court*.
- The theme-config-file cannot be a symlink, irrespective of `safe mode` and whether the file pointed to by the symlink is a legitimate file within the theme-gem.
- The theme-config should be a set of key-value pairs. An empty config file, a config file that simply *lists items* under a key, or a config file with just a simple string of text will simply be ignored silently. Users will not get a warning or any log output regarding this discrepancy.
- Any settings defined by the theme-config can be overridden by the user.

While this feature is to enable easier adoption of a theme, the restrictions ensure that a theme-config cannot affect the build in a concerning manner. Any plugins required by the theme will have to be listed manually by the user or provided by the theme's `gemspec` file.

This feature will let the theme-gem to work with *theme-specific config variables* out-of-the-box.

Documenting your theme

Your theme should include a `/README.md` file, which explains how site authors can install and use your theme. What layouts are included? What includes? Do they need to add anything special to their site's configuration file?

Adding a screenshot

Themes are visual. Show users what your theme looks like by including a screenshot as `/screenshot.png` within your theme's repository where it can be retrieved programmatically. You can also include this screenshot within your theme's documentation.

Previewing your theme

To preview your theme as you're authoring it, it may be helpful to add dummy content in, for example, `/index.html` and `/page.html` files. This will allow you to use the `jekyll build` and `jekyll serve` commands to preview your theme, just as you'd preview a Jekyll site.

If you do preview your theme locally, be sure to add `/_site` to your theme's `.gitignore` file to prevent the compiled site from also being included when you distribute your theme.

Publishing your theme

Themes are published via RubyGems.org. You will need a RubyGems account, which you can create for free.

1. First, you need to have it in a git repository:

   ```
   git init # Only the first time
   git add -A
   git commit -m "Init commit"
   ```

2. Next, package your theme, by running the following command, replacing `jekyll-theme-awesome` with the name of your theme:

   ```
   gem build jekyll-theme-awesome.gemspec
   ```

3. Finally, push your packaged theme up to the RubyGems service, by running the following command, again replacing `jekyll-theme-awesome` with the name of your theme:

   ```
   gem push jekyll-theme-awesome-*.gem
   ```

4. To release a new version of your theme, update the version number in the gemspec file, (`jekyll-theme-awesome.gemspec` in this example), and then repeat Steps 1 - 3 above. We recommend that you follow Semantic Versioning while bumping your theme-version.