Documentation

Here you should be able to find everything you need to know to accomplish the most common tasks when blogging with Hydejack. Should you think something is missing, please let me know (mailto:mail@qwtel.com) . Should you discover a mistake in the docs (or a bug in general) feel free to open an issue (https://github.com/qwtel/hydejack/issues) on GitHub.

NOTE: While this manual tries to be beginner-friendly, as a user of Jekyll it is assumed that you are comfortable running shell commands and editing text files.

Buyers of the PRO version can jump straight to installation for pro buyers (#pro-version) ,) or upgrades for pro buyers (#pro-version) .)

NOTE: This document was created using Hydejack's print layout. If you prefer to read it the documentation in your browser, you can find it here (/hydejack/docs/7.0.0/) .

Table of Contents

Install

There are multiple ways of installing Hydejack. The easiest and recommended way is via the Ruby gem (#via-gem) . If you've downloaded the zip, you'll want to install via the zip file (#via-zip) . If you know what you are doing, you can fork the git repository (#via-git) .

Buyers of the PRO version should follow these steps (#pro-version) .

Via gem

Using the gem-based theme has the advantage of not cluttering your blog repository. It's also easier to upgrade, so it is especially recommended for beginners.

If you haven't already, create a new Jekyll site first:

```
$ jekyll new <PATH>
```

Your site's root dir should look something like this

```
├── _posts
│   └── 2017-04-07-welcome-to-jekyll.markdown
├── _config.yml
├── about.md
├── Gemfile
├── Gemfile.lock
└── index.md
```

NOTE: Hydejack works with Jekyll's default `config.yml`, but it is recommended that you replace it with Hydejack's default config file (https://github.com/qwtel/hydejack/blob/master/_data/_config.yml). It contains the names of all config options known to Hydejack and provides sensible defaults (like minifying HTML and CSS in production builds).

Next, you'll want to add `jekyll-theme-hydejack` as a dependency by adding the following line to the `Gemfile`.

```
gem "jekyll-theme-hydejack"
```

(You can also remove the old theme `jekyll-theme-minima` from the Gemfile)

Now you want to edit the `_config.yml` of your Jekyll site and set Hydejack as the theme. Look for the `theme` key (or add it when missing) and set its value to `jekyll-theme-hydejack`.

```
theme: jekyll-theme-hydejack
```

For more information on gem-based themes, see the Jekyll Documentation (http://jekyllrb.com/docs/themes/).

You can now continue with running locally (#running-locally).

Via zip

If you downloaded the zip, extract the contents somewhere on your machine. The high-level folder structure will look something like.

```
├── _data
├── _featured_categories
├── _featured_tags
├── _includes
├── _js
├── _layouts
├── _posts
├── _sass
├── assets
├── _config.yml
├── 404.md
├── about.md
├── index.html
└── posts.md
```

`cd` into the directory where `_config.yml` is located and follow the steps in Running locally (#running-locally) .

Via git

If you are familiar with using git, you can add the Hydejack repository (https://github.com/qwtel/hydejack) as a remote, and merge its master branch into your working branch.

```
$ git remote add hydejack git@github.com:qwtel/hydejack.git
$ git pull hydejack master
```

You can also update Hydejack this way. The master branch will not contain work in progress, but will contain major (breaking) changes. This approach is recommended if you intend to customize Hydejack.

You can now continue with running locally (#running-locally) .

PRO Version

If you bought the PRO version, you've received a zip archive with the following contents:

```
├── hydejack-docs-7.0.0.pdf
├── install
├── upgrade
├── favicons.psd
└── sidebar-bg.psd
```

The following list describes what each of those are

`hydejack-docs-7.0.0.pdf`
: This documentation in PDF form.

`install`
: Contains all files and folders needed to create a new blog.

`upgrade`
: Contains only the files and folders needed for upgrading form an earlier version of Hydejack (6.0.0 or above). See the **Upgrade** (#upgrade) for more.

`favicon.psd`
: A Photoshop template to help with generating the favicon, apple touch icon, etc.

`sidebar-bg.psd`
: A Photoshop template for blurred sidebar backgrounds.

`*.patch`
: These are git patches that you can apply to your repository via git-apply (https://git-scm.com/docs/git-apply) . Use these if you are using git and you are worried about overwriting changes. This is for advanced users.

For new installations only the `install` folder is interesting. Unzip the archive somewhere on your machine, then `cd` *into* the `install` folder, e.g.

```
$ cd ~/Downloads/hydejack-pro-7.0.0/install/
```

You can now continue with:

Running locally

Make sure you've `cd` ed into the directory where `_config.yml` is located. Before running for the first time, dependencies need to be fetched from RubyGems (https://rubygems.org/) :

```
$ bundle install
```

NOTE: If you are missing the `bundle` command, you can install Bundler by running `gem install bundler`.

Now you can run Jekyll on your local machine:

```
$ bundle exec jekyll serve
```

and point your browser to http://localhost:4000 (http://localhost:4000) to see Hydejack in action.

Upgrade

NOTE: Before upgrading from v6 to v7, make sure you've read the **CHANGELOG (/hydejack/CHANGELOG/)**, especially the part about the license change (/hydejack/CHANGELOG/#license-v7).

Via gem

Upgrading the the gem-based theme is as easy as running

```
bundle update jekyll-theme-hydejack
```

Via zip

Upgrading via zip is a bit of a dark art, specially if you've made changes to any source files, and the prime reason why I suggest using the gem-based version of the theme.

Generally, you'll want to copy these files and folders:

- `_includes/`
- `_layouts/`
- `_sass/`
- `assets/`
- `Gemfile`
- `Gemfile.lock`

and merge them with your existing folder. However, you'll also want to check out `_data` and `_config.yml` for any changes and read latest entries to the **CHANGELOG (/hydejack/CHANGELOG/)**.

NOTE: If you've modified any of Hydejack's internal files, your changes will most likely be overwritten and you have to apply them again. Make sure you've made a backup before overwriting any files.

Via git

The latest version sits on the `master` branch of qwtel/hydejack (https://github.com/qwtel/hydejack). To apply them to your repository run

```
$ git remote add hydejack git@github.com:qwtel/hydejack.git
$ git pull hydejack master
```

PRO Version

Buyers of the PRO version will find the files necessary for an upgrade in the `upgrade` folder of the downloaded zip archive.

NOTE: If you've modified any of Hydejack's internal files, your changes will most likely be overwritten and you have to apply them again. Make sure you've made a backup before overwriting any files.

The archive also contains `.patch` files, that you can apply to your repository via git-apply (https://git-scm.com/docs/git-apply). Using this method, git will generate merge conflicts when

changes in the patch conflict with any of your changes.

Legacy

Unfortunately, upgrading form v5 and earlier is not straightforward. A lot of patterns and names have changed, motivated by a variety of reasons, including better integration with the rest of the Jekyll ecosystem and simplified workflows enabled by Jekyll Collections.

Updating the folder structure

Copy the the following folders and files from Hydejack v6 into your existing repository. Make sure you merge the folder contents.

- `_data/`
- `_includes/`
- `_layouts/`
- `_sass/`
- `assets/`
- `index.html` (`index.md` *)
- `Gemfile`
- `Gemfile.lock`

Note that the `public` folder has been renamed to `assets`. You'll want to move your static assets there.

Updating the configuration

`_config.yml` has changed considerably. Open it and make the following changes.

1. Rename the following keys
   - `font_accent` => `font_heading`
   - `load_google_fonts` => `google_fonts`
   - `google_analytics_id` => `google_analytics`
2. Enable Jekyll Collections for categories and tags by adding

```
collections:
  featured_categories:
    permalink: /category/:name/
    output:    true
  featured_tags:
    permalink: /tag/:name/
    output:    true
```

3. Delete `photo` and `photo2x` form the author key and add a `picture` hash instead that looks like

```
picture:
  path: <photo>
  srcset:
    1x: <photo>
    2x: <photo2x>
```

   If you have only one photo, you can just provide the URL directly, e.g. `picture: <url>`.

   For more information, see Adding an author (#adding-an-author).)

4. Rename `gems` to `plugins` and make sure the list contains `jekyll-seo-tag`.

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

> **NOTE**: When making changes to `_config.yml`, it is necessary to restart the Jekyll process for the changes to take effect.

Restoring the tags

1. Delete the `tag` folder.
2. Create a top-level folder called `_featured_tags`.
3. For each entry in `_data/tags.yml`, create a markdown file in `_features_tags` with the name of the tag as filename, e.g. `hyde.md` for tag “hyde”.
4. For each tag, copy its contents from `_data/tags.yml` into the new file’s front matter, e.g.

```
---
layout: list
name: Hyde
description: >
  Hyde is a brazen two-column Jekyll theme...
accent_image: /hydejack/public/img/hyde.jpg
accent_color: '#949667'
---
```

   Be aware that `image` has been renamed to `accent_image` and `color` has been renamed to `accent_color`.

5. Add `layout: list` to the front matter.
6. Once you've copied all tags into their own files, delete `_data/tags.yml`.

Restoring the sidebar entries

Hydejack can now link to any kind of page in the sidebar.

1. Delete `sidebar_tags` in `_config.yml`.
2. Open a file who's page you would like to add to the sidebar. If you want to add a tag, open `_featured_tags/<tagname>.md`.
3. Add `menu: true` to its front matter.
4. (Optional) Set `order: <number>`, where `<number>` is the number at which you would like the link to appear.

Restoring the RSS feed

The feed is now provided by the `jekyll-feed` plugin instead of a custom solution.

1. Delete `atom.xml`
2. Add `- jekyll-feed` to `gems` in `_config.yml`, e.g.

```
gems:
  - jekyll-seo-tag
  - jekyll-feed
```

3. (Optional) Add the following to `_config.yml` to make the feed appear at the same URL as the old `atom.xml`.

```
feed:
  path: atom.xml
```

Restoring the comments

The way comments are enabled has changed slightly. You now have to enable them per page by adding `comments: true` to the front matter (this is what the Disqus integration guide (https://disqus.com/admin/install/platforms/jekyll/) suggests). To enable them for all posts, add to the config file

```
defaults:
  - scope:
      type: posts
    values:
      comments: true
```

Restoring the about page

Hydejack now has a dedicated layout for about pages. To use it, open `about.md` and change the `layout` in the front matter to `about` and delete `{\% include about-short.html author=site.author %\}`.

Config

Once Jekyll is running, you can start with basic configuration by adding various entries to `_config.yml`. Besides the documentation here, the default `_config.yml` (https://github.com/qwtel/hydejack/blob/master/_config.yml) is also documented extensively.

NOTE: When making changes to `_config.yml`, it is necessary to restart the Jekyll process for the changes to take effect.

Setting `url` and `baseurl`

The first order of business should be to set the correct `url` and `baseurl` values in `_config.yml`.

The `url` is the domain of your site, including the protocol (`http` or `https`). For this site, it is

```
url: https://qwtel.com
```

If your entire Jekyll blog is hosted in a subdirectory of your page, provide the path in `baseurl` with a leading `/`, but no trailing `/`, e.g.

```
baseurl: /hydejack
```

Otherwise, provide the empty string `''`

GitHub Pages

When hosting on GitHub Pages (https://pages.github.com/) (unless you are using a custom domain), the `url` is

```
url: https://<username>.github.io
```

The `baseurl` depends on the kind of page you are hosting.

- When hosting a *user or organization page*, use the empty string.
- When hosting *project page*, use `/<reponame>`.

For for information on the types of pages you can host on GitHub, see the GitHub Help article (https://help.github.com/articles/user-organization-and-project-pages/).

Changing accent colors and sidebar images

Hydejack allows you to choose the background image of the sidebar, as well as the accent color (color of the links, selection and focus outline, etc…) on a per-page, per-category, per-tag, per-author and global basis.

Set the fallback values in `_config.yml`, which are used should no other rule (page, category, tag, author) apply:

```yaml
accent_image: /hydejack/assets/img/sidebar-bg.jpg
accent_color: '#A85641'
```

> **NOTE**: I recommend using a blurred image in order for the text to remain readable. If you save a blurred image as JPG, it will also drastically reduce its file size.

The `accent_image` is optional. If you leave it out, Hydejack will use the `accent_color` as background (slightly darkened). You can also provide a single color instead of an image like this:

```yaml
accent_image:
  background: '#202020' # provide a valid CSS background value
```

Changing fonts

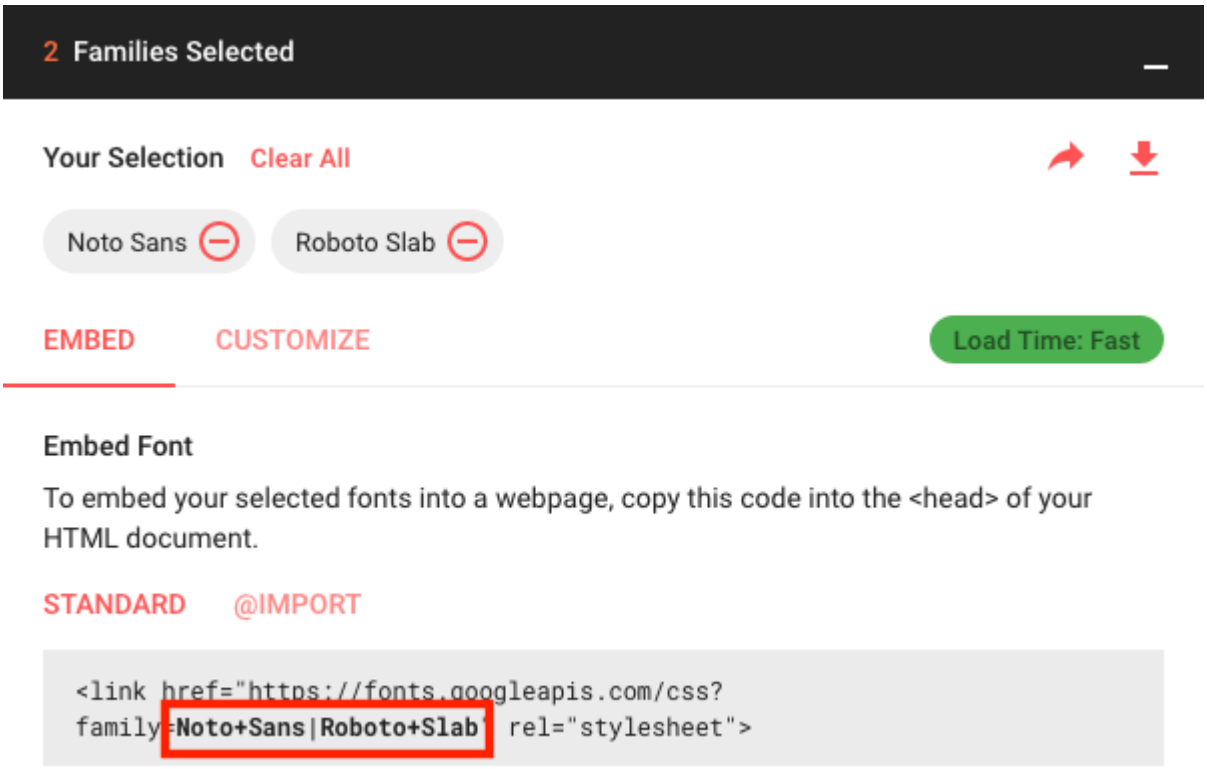
Hydejack lets you configure the font of regular text and headlines and it has built-in support for Google Fonts. There are three keys in `_config.yml` associated with it: `font`, `font_heading` and `google_fonts`. The defaults are:

```yaml
font:         "'Noto Sans', Helvetica, Arial, sans-serif"
font_heading: "'Roboto Slab', Helvetica, Arial, sans-serif"
google_fonts: "Roboto+Slab:700|Noto+Sans:400,400i,700,700i"
```

`font` and `font_heading` must be valid CSS `font-family` values. When using Google Fonts make sure they consist of at least two fonts (everything except the first entry will be used as a fallback until the fonts have completed loading).

The `google_fonts` key is the string necessary to fetch the fonts from Google. You can get it from the download page at Google Fonts (https://fonts.google.com) after you've selected one

or more fonts:



Using safe web fonts

If you prefer not to use Google Fonts and use safe web fonts (http://www.cssfontstack.com/) instead, set `no_google_fonts` to `true`:

```
hydejack:
  no_google_fonts: true
```

In this case, `font` and `font_heading` do not have to contain more than one font. You may also remove the `google_fonts` key in this case.

Choosing a blog layout

Hydejack features two layouts for showing your blog posts.

- The `list` layout (https://qwtel.com/hydejack/posts/) only shows the title and groups the posts by year of publication.
- The `blog` layout (https://qwtel.com/hydejack/blog/) is a traditional paginated layout and shows the title and an excerpt of each post.

In order to use the `list` layout add the following front-matter to a new markdown file:

```
---
layout: list
title:  Home
---
```

If you want to use the `blog` layout, you need to add the `paginate` and `paginate_path` keys to your config file, e.g.

```
paginate:      5
paginate_path: '/page-:num/'
```

The `blog` layout needs to be applied to a file with the `.html` file extension and the `paginate_path` needs to match the path to the `index.html` file, i.e. if you want the blog to appear at `/blog/`, put a `index.html` in the `blog` dir and set `paginate_path` to be `/blog/page-:num/`.

For more information see Pagination (https://jekyllrb.com/docs/pagination/).

Adding an author

As a bare minimum, you should add an `author` key with a `name` and `email` sub-key (used by the feed plugin (https://github.com/jekyll/jekyll-feed)) to to your config file:

```
author:
  name:  Florian Klampfer
  email: mail@qwtel.com
```

If you would like the author to be displayed in the about section below a post or project*, as well as the top of about and welcome* pages, add an `about` key and provide some markdown content. I recommend using the YAML pipe `|` syntax, so you can include multiple paragraphs:

```
author:
  name:  Florian Klampfer
  email: mail@qwtel.com
  about: |
    Hi, I'm Florian or @qwtel...

    This is another paragraph.
```

Adding an author's picture

If you'd like for the author's picture to appear in addition the the about text (see above), you have to provide an URL to the `picture` key:

```
author:
  picture:  /hydejack/assets/img/me.jpg
```

If you'd like to provide multiple versions of the picture for screens with different pixel densities, you can provide `path` and `srcset` keys instead.

```
author:
  picture:
    path:   /hydejack/assets/img/me.jpg
    srcset:
      1x:   /hydejack/assets/img/me.jpg
      2x:   /hydejack/assets/img/me@2x.jpg
```

The `path` key is a fallback image for browsers that don't support the `srcset` attribute.

The keys of the `srcset` hash will be used as image descriptors. For more information on `srcset`, see the documentation at MDN (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img#attr-srcset), or this article from CSS-Tricks (https://css-tricks.com/responsive-images-youre-just-changing-resolutions-use-srcset/).

Adding social media icons

Hydejack supports a variety of social media icons out of the box. These are defined on a per-author basis, so make sure you've followed the steps in Adding an author (#adding-an-author).

NOTE: If you are using the gem-based version of Hydejack, download `social.yml` (https://github.com/qwtel/hydejack/blob/master/_data/social.yml) and put it into `_data` in the root directory. This is necessary because gem-based themes do not support including `_data`.

You can add a link to a social network by adding an entry to the `social` key in to an author. It consists of the name of the social network as key and your username within that network as value, e.g.

```yaml
author:
  social:
    twitter: qwtel
    github:  qwtel
```

Check out `authors.yml` (https://github.com/qwtel/hydejack/blob/master/_data/authors.yml) to see which networks are available. You can also follow the steps here (#advanced) to add your own social media icons.

You can change the order in which the icons appear by moving lines up or down, e.g.

```yaml
author:
  social:
    github:  qwtel # now github appears first
    twitter: qwtel
```

To get an overview of which networks are available and how a typical username in that network looks like, see the included `authors.yml` (https://github.com/qwtel/hydejack/blob/master/_data/authors.yml).

Should providing a username not produce a correct link for some reason, you can provide a complete URL instead, e.g.

```yaml
social:
  youtube: https://www.youtube.com/channel/UCu0PYX_kVANdmgIZ4bw6_kA
```

NOTE: You can add any platform, even if it's not defined in `social.yml` (https://github.com/qwtel/hydejack/blob/master/_data/social.yml), by providing a complete URL.

However, a fallback icon will be used when no icon is available. Supplying your own icons is an advanced topic (#advanced) .

Adding an email or RSS icon

If you'd like to add email or RSS to the list, add the `email` and `rss` keys, e.g.:

```
social:
  email: mailto:mail@qwtel.com
  rss:   https://qwtel.com/hydejack/feed.xml # make sure you provide an absolute U
```

Enabling comments

Hydejack supports comments via Disqus (https://disqus.com/) . Before you can add comments to a page you need to register and add your site to Disqus' admin console. Once you have obtained your "Disqus shortname", you include it in your config file:

```
disqus: <Disqus shortname>
```

Now comments can be enabled by adding `comments: true` to the front matter.

```
---
layout:   post
title:    Hello World
comments: true
---
```

You can enable comments for entire classes of pages by using front matter defaults (https://jekyllrb.com/docs/configuration/#front-matter-defaults) . E.g. to enable comments on all posts, add to your config file:

```
defaults:
  - scope:
      type: posts
    values:
      comments: true
```

Enabling Google Analytics

Enabling Google Analytics is as simple as setting the `google_analytics` key.

```
google_analytics: UA-XXXXXXXX-X
```

Conversely, if you want to disable it, you only have to remove the key and no GA code will be part of the generated pages.

Changing built-in strings

You can change the wording of built-in strings like “Related Posts” or “Read more” in `_data/strings.yml`. If you are using the gem-based version, you can get the default file here (https://github.com/qwtel/hydejack/blob/master/_data/strings.yml).

You will frequently find markers like `<!--post_title-->`. You can place them freely within your string and they will be replaced with the content they refer to.

You may also use this feature to translate the theme into different languages. In this case you should also set the `lang` key to your config file, e.g.

```
lang: cc-ll
```

where `cc` is the 2-letter country code and `ll` specifies a 2-letter location code, e.g.: `de-at`.

You may also change the strings used for formatting dates and times (look out for the `date_formats` key), but be aware that the values you provide need to be valid Ruby format directives (http://ruby-doc.org/core-2.4.1/Time.html#method-i-strftime).

Basics

Adding a category or tag

Hydejack allows you to use the `list` layout to show all posts of a particular tag or category.

Before you start, make sure your config files contains the `featured_tags` and `features_categories` collections:

```
collections:
  featured_categories:
    permalink:          /category/:name/
    output:             true
  featured_tags:
    permalink:          /tag/:name/
    output:             true
```

Recap: Tags and categories in Jekyll

Posts in Jekyll can belong to one or more categories, as well as one or more tags. They are defined in a post's front matter:

```
---
layout:     post
title:      Welcome to Jekyll
categories: [jekyll, update]
tags:       [jekyll, update]
---
```

Posts can also be assigned to a category based on their position within the folder structure, e.g.

```
├── jekyll
│   └── update
│       └── _posts
│           └── 2017-04-07-welcome-to-jekyll.markdown
```

would place "Welcome to Jekyll" in the categories `jekyll` and `update`. Whether you use this method or not, categories will always be part of a posts URL, while tags will not.

Categories	`/jekyll/update/2017/04/07/welcome-to-jekyll/`
Tags	`/2017/04/07/welcome-to-jekyll/`

As far as Jekyll is concerned, these are the only differences.

Tags and categories in Hydejack

Categories and tags are displayed by Hydejack below the title, after the date. Categories are displayed with the preposition “in”, while tags are displayed with the preposition “on”, e.g.

Categories	Welcome to Jekyll¬ 07 Apr 2017 **in** Jekyll / Update
Tags	Welcome to Jekyll¬ 07 Apr 2017 **on** Jekyll, Update
Both	Welcome to Jekyll¬ 07 Apr 2017 **in** Jekyll / Update **on** Jekyll, Update

Adding a new category or tag

Be default, categories and tags are rendered as plain text. Further steps are necessary if you want them to link to a page that contains a list of all posts that belong to that category or tag.

For each “featured” category or tag, a file called `<categoryname>.md` or `<tagname>.md` has to be created in `_featured_tags` or `_featured_categories`, respectively. Each file in these folders is part of a Jekyll Collection (https://jekyllrb.com/docs/collections/).

The the data of a category or tag is set in the files front matter, e.g.

```
---
layout: list
title:  Hyde
slug:   hyde
description: >
  Hyde is a brazen two-column Jekyll](http://jekyllrb.com) theme
  that pairs a prominent sidebar with uncomplicated content.
  It's based on [Poole](http://getpoole.com), the Jekyll butler.
---
```

`layout`
: Must be `list`

`title`
: Used as title of the page, as well as name of the category or tag as part of the line below a blog post’s title. Can be different from the name of the tag or category, as long as `slug` is identical to the name.

`slug`

Must be identical to the key used in the blog's front matter, i.e. if you use `categories: [jekyll]` or `tags: [jekyll]` the `slug` must be `jekyll`. Normally the slug is derived from the title, but it is recommended that you set it explicitly.

`description`
: A medium-length description, used on the tag or category's detail page as meta description and shown in a message box below the title.

`accent_image`
: URL. Will be used as fallback for all pages that belong to that category or tag.

`accent_color`
: Color code. Will be used as fallback for all pages that belong to that category or tag.

`menu`
: Set to to `true` if you want the category or tag to appear in the sidebar. For more information, see Adding an entry to the sidebar (#adding-an-entry-to-the-sidebar).

Once the file is created, the page can be found at `/category/<categoryname>/` or `/tag/<tagname>/`.

Adding a page

You can add generic pages that support markdown content but aren't blog posts. For example, this documentation is written in markdown, consisting of several generic pages.

To add a page, create a new markdown file and put `layout: page` in a front matter

```
---
layout: page
title:  Documentation
---
```

Now you can add content as you would in a blog post.

Adding an entry to the sidebar

Hydejack's sidebar can add links to any page within the site. In order for a page to appear in the sidebar, it needs to have a truthy `menu` value defined in its front matter. The page

also needs to have a `title`, otherwise the entry in the sidebar will be blank.

If you want the link to appear at a particular position, you can set a numeric value to the `order` key. However, the page is not guaranteed to appear in the 5th position when you set a value of `5`, since it will only use that number to sort the pages, i.e. the position of a page also depends on the `order` of all other pages in the sidebar.

Adding an about page

About pages are a frequent use case, so Hydejack has a special layout for it, which is a slight modification of the `page` layout. Demo (https://qwtel.com/hydejack/about/). The main difference is that it will display an author's `about` text and `picture` above the regular content.

To create an about page, make sure `layout` is set to `about`, and that the `author` key is set to an author defined in `_data/authors.yml`. For more on authors, see Adding an author (#adding-an-author).)

```
---
layout: about
title:  About
author: qwtel
---
```

Adding a welcome page*

If you bought the PRO version of Hydejack you have access to the `welcome` layout. It is intended to showcase your projects and blog posts in a compact way. Technically, it is a modified version of the `about` layout, so it will also show author information at the top. Demo (https://qwtel.com/hydejack/).

For reference, the layout/order of content on the welcome page looks like:

- Title
- Author's about text
- Content (before `content_separator`)
- Latest/Selected Projects

- Latest/Selected Posts
- Content after `content_separator` (if any)

You can create a welcome page by creating a new markdown file and setting the layout to `welcome` in the front matter.

```
---
layout: welcome
title:  Welcome
author: qwtel
---
```

Without further configuration, the welcome page will show the two most recent projects and five most recent blog posts. However, the welcome layout supports selecting specific projects and posts, by adding to the front matter, e.g.:

```
---
layout:            welcome
title:             Welcome
selected_projects:
  - _projects/hydejack-v6.md
  - _projects/hyde-v2.md
selected_posts:
  - _posts/2017-05-03-javascripten.md
  - _posts/2012-02-07-example-content.md
more_projects:     projects.md
more_posts:        posts.md
big_project:       false
content_separator: <!-- more -->
---
```

`layout`
: Must be `welcome`.

`selected_projects`
: A list of paths to project files that should be displayed below the main content of the page. The paths are relative to the main directory with no leading `./`. If no paths are provided, the two most recent projects will be used.

`selected_projects`
: A list of paths to blog posts that should be featured on the welcome page. The paths are relative to the main directory with no leading `./`. If no paths are provided, the five most recent posts will be used.

`more_projects`
: The path to the main projects page. The path is relative to the main directory with no leading `./`.

`more_posts`
: The path to the main posts page. The path is relative to the main directory with no leading `./`.

`big_project`
: Optional. When `true`, project thumbnails will span the full width instead of half. This setting takes precedence over the `big_project` value of individual projects, i.e. it will apply to the entire page.

`content_separator`
: Optional. Defines a marker that will be used to split the content in two parts. The first part will go before the "Selected/Latest Projects" and "Selected/Latest Posts" section, the second part will go below it.

Adding a projects page*

The projects page will show all projects in a particular collection. First, you need to make sure that you have the `projects` collection defined in `_config.yml`:

```
collections:
  projects:
    permalink: /projects/:path/
    output:    true
```

Next, add a `projects.md` to in the root (you can adjust the name/location to match the the `permalink` of the collection). This file has the `projects` layout (mind the "s" at the end) and should have a `show_collection` key, with the name of the collection as a value, e.g.:

```
---
layout:          projects
title:           Projects*
show_collection: projects
big_project:     true
---
```

`layout`

Must be `projects`.

`title`

The title of the page. Note that this name is reused as part of each individual project page (for the link that directs back to the projects page).

`show_collection`

The name of the collection you want display on this page. Defaults to `projects`.

`big_project`

Optional. When `true`, project thumbnails will span the full width, instead of only half. This setting takes precedence over the `big_project` value of individual projects, i.e. it will apply to the entire page.

Adding a project*

Projects are organized using Jekyll Collections (https://jekyllrb.com/docs/collections/). Each project generates an entry on the projects layout (Demo (https://qwtel.com/hydejack/projects/)) as well as its own detail page (Demo (https://qwtel.com/hydejack/projects/hydejack-v6/)).

Each project is defined by a file in the `_projects` directory. The project's meta information is defined in the file's front matter. You can also add markdown content. A project's front matter may look like:

```yaml
---
layout:      project
title:       Hyde v2*
date:        2 Jan 2014
screenshot:
  src:       /hydejack/assets/img/projects/hyde-v2@0,25x.jpg
  srcset:
    1920w:   /hydejack/assets/img/projects/hyde-v2.jpg
    960w:    /hydejack/assets/img/projects/hyde-v2@0,5x.jpg
    480w:    /hydejack/assets/img/projects/hyde-v2@0,25x.jpg
caption:     Hyde is a brazen two-column Jekyll theme.
description: >
  Hyde is a brazen two-column [Jekyll](http://jekyllrb.com) theme that pairs a pro
  It's based on [Poole](http://getpoole.com), the Jekyll butler.
links:
  - title:   Demo
    url:     http://hyde.getpoole.com
  - title:   Source
    url:     https://github.com/poole/hyde
author:      mdo
big_project: true
---
```

`layout`

Must be set to `project`

`date`

Providing a year is the minimum requirement. Used to sort the projects.

`screenshot`

A 16:9 screenshot of the project. You can pass a URL to an image, but it is recommended that you provide an entire `srcset` (see above). Hydejack will show the screenshot in various sizes, depending on the screen width, so that no specific size will fit all. Instead it is recommended that you use a mipmap (https://en.wikipedia.org/wiki/Mipmap) -like approach, providing the image in multiple sizes, each image half the width of the previous one. The `src` key is a fallback image for browsers that don't support the `srcset` attribute. The keys of the `srcset` hash will be used as descriptors. For more information on `srcset`, see the documentation at MDN (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img#attr-srcset), or this article from CSS-Tricks (https://css-tricks.com/responsive-images-youre-just-changing-resolutions-use-srcset/).

`caption`

A short description, shown as part of each "project card" in the `projects` layout.

`description`

A medium-length description, used on the project's detail page as meta description and shown as message box below he screenshot.

`links`

A list of `title` - `url` pairs that link to external resources related to this project.

`author`

Shown below the project, similar to posts.

`big_project`

Optional. When `true`, the project preview will span the full content width. You can use this for projects that you want to direct additional attention to. You can set/override this for an entire page, by setting `big_project` in the front matter (applies to the `projects` and `welcome` layout).

Adding a resume*

Hydejack's PRO version features a generalized resume layout. Demo (https://qwtel.com/hydejack/resume/).

It generates the resume page from a valid JSON Resume (https://jsonresume.org/), which is good news if you already have a JSON resume. Otherwise, there are various ways of obtaining one:

- You can use the visual JSON Resume Editor (http://registry.jsonresume.org/).
- If you have a LinkedIn profile, you can try LinkedIn to Json Résumé (https://jmperezperez.com/linkedin-to-json-resume/).
- You can edit the example `resume.json` (https://github.com/qwtel/hydejack/blob/v6/_data/resume.json) in the `_data` directly. It contains example entries for each type of entry.

Once you have a JSON Resume, place it into `_data`.

If you prefer editing YAML files, there is an example `_resume.yml` (https://github.com/qwtel/hydejack/blob/v6/_data/_resume.yml) file in `_data`. In order to use it, rename it to `resume.yml` and delete `resume.json`.

To render the resume page, create a new markdown file and set the layout to `resume` in the front matter:

```
---
layout: resume
title:  Resume
---
```

NOTE: You can download the final `resume.json` (minified) from the assets folder. When running locally, you can find it at `_site/assets/resume.json`.

Writing

Hydejack offers a few additional features to markup your markdown. Don't worry, these are merely CSS classes added via the standard `{:.my-class}` syntax, so that your posts remain compatible with other Jekyll themes.

NOTE: For an introduction to markdown in general, see Mastering Markdown (https://guides.github.com/features/mastering-markdown/) and kramdown Syntax (https://kramdown.gettalong.org/syntax.html).

A word on building speeds

If building speeds are a problem, try using the `--incremental` flag, e.g.

```
bundle exec jekyll serve --incremental
```

From the Jekyll docs (https://jekyllrb.com/docs/configuration/#build-command-options) (emphasis mine):

> 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*.

The breakage occurs when you create new files or change filenames. Also, changing the title, category, tags, etc. of a page or post will not be reflected in pages other then the page

or post itself. This makes it ideal for writing new posts and previewing changes, but not setting up new content.

Adding a table of contents

You can add a generated table of contents to any page by adding `{:toc}` below a list.

Example: see above

Markdown:

```
* this unordered seed list will be replaced by toc as unordered list
{:toc}
```

Adding message boxes

You can add a message box by adding the `message` class to a paragraph.

Example:

NOTE: You can add a message box.

Markdown:

```
**NOTE**: You can add a message box.
{:.message}
```

Adding large text

You can add large text by adding the `lead` class to the paragraph.

Example:

You can add large text.

Markdown:

```
You can add large text.
{:.lead}
```

Adding large images

You can make an image span the full width by adding the `lead` class.

Example:

Markdown:

```
![Full-width image](https://placehold.it/800x100){:.lead}
```

Adding large quotes

You can make a quote “pop out” by adding the `lead` class.

Example:

> You can make a quote “pop out”.

Markdown:

```
> You can make a quote "pop out".
{:.lead}
```

Adding faded text

You can gray out text by adding the `faded` class.

Use this sparingly and for information that is not essential — or you don't want viewers to read at all, like when you pull a line form a dirty rap song..

Example:

I'm faded, faded, faded.

Markdown:

```
I'm faded, faded, faded.
{:.faded}
```

Adding tables

Adding tables is straightforward and works just as described in the kramdown docs (https://kramdown.gettalong.org/syntax.html#tables) , e.g.

Default aligned	Left aligned	Center aligned	Right aligned
First body part	Second cell	Third cell	fourth cell

Markdown:

```
| Default aligned |Left aligned| Center aligned  | Right aligned  |
|-----------------|:-----------|:---------------:|---------------:|
| First body part |Second cell | Third cell      | fourth cell    |
```

However, it gets tricker when adding large tables. In this case, Hydejack will break the layout and grant the table the entire available screen width to the right:

Default aligned	Left aligned	Center aligned	Right aligned	Default aligned	Left aligned	Center aligned	Right aligned	Default aligned
First body part	Second cell	Third cell	fourth cell	First body part	Second cell	Third cell	fourth cell	First body part
Second line	foo	**strong**	baz	Second line	foo	**strong**	baz	Second line

Default aligned	Left aligned	Center aligned	Right aligned	Default aligned	Left aligned	Center aligned	Right aligned	Default aligned
Third line	quux	baz	bar	Third line	quux	baz	bar	Third line
Second body				Second body				Second body
2 line				2 line				2 line
Footer row				Footer row				Footer row

Scroll table

If the extra space still isn't enough, the table will receive a scrollbar. It is browser default behavior to break the lines inside table cells to fit the content on the screen. By adding the `scroll-table` class on a table, the behavior is changed to never break lines inside cells, e.g:

Default aligned	Left aligned	Center aligned	Right aligned	Default aligned	Le
First body part	Second cell	Third cell	fourth cell	First body part	Se
Second line	foo	**strong**	baz	Second line	fo
Third line	quux	baz	bar	Third line	qu
Second body				Second body	
2 line				2 line	
Footer row				Footer row	

Flip table

Alternatively, you can "flip" (transpose) the table. Unlike the other approach, this will keep the table head (now the first column) fixed in place.

You can enable this behavior by adding `flip-table` or `flip-table-small` to the CSS classes of the table. The `-small` version will only enable scrolling on "small" screens (<

1080px wide).

> **NOTE**: This approach only works on simple tables that have a single `tbody` and an optional `thead`.

Example:

Default aligned	First body part	Second line	Third line	4th line	5th line	6th line
Left aligned	Second cell	foo	quux	quux	quux	quux
Center aligned	Third cell	**strong**	baz	baz	baz	baz
Right aligned	fourth cell	baz	bar	bar	bar	bar
Default aligned	First body part	Second line	Third line	4th line	5th line	6th line
Left aligned	Second cell	foo	quux	quux	quux	quux
Center aligned	Third cell	**strong**	baz	baz	baz	baz
Right aligned	fourth cell	baz	bar	bar	bar	bar
Default aligned	First body part	Second line	Third line	4th line	5th line	6th line
Left aligned	Second cell	foo	quux	quux	quux	quux
Center aligned	Third cell	**strong**	baz	baz	baz	baz
Right aligned	fourth cell	baz	bar	bar	bar	bar
Default aligned	First body part	Second line	Third line	4th line	5th line	6th line
Left aligned	Second cell	foo	quux	quux	quux	quux
Center aligned	Third cell	**strong**	baz	baz	baz	baz
Right aligned	fourth cell	baz	bar	bar	bar	bar

Small tables

If a table is small enough to fit the screen even on small screens, you can add the `stretch-table` class to force a table to use the entire available content width. Note that stretched tables can no longer be scrolled.

Default aligned	Left aligned	Center aligned	Right aligned

Default aligned	Left aligned	Center aligned	Right aligned
First body part	Second cell	Third cell	fourth cell

Adding code blocks

To add a code block without syntax highlighting, simply indent 4 spaces (regular markdown). For code blocks with code highlighting, use `~~~<language>` . This syntax is also supported by GitHub. For more information and a list of supported languages, see Rouge (http://rouge.jneen.net/) .

Example:

```
// Example can be run directly in your JavaScript console

// Create a function that takes two arguments and returns the sum of those
// arguments
var adder = new Function("a", "b", "return a + b");

// Call the function
adder(2, 6);
// > 8
```

Markdown:

```
~~~js
// Example can be run directly in your JavaScript console

// Create a function that takes two arguments and returns the sum of those
// arguments
var adder = new Function("a", "b", "return a + b");

// Call the function
adder(2, 6);
// > 8
~~~
```

NOTE: DO NOT use Jekyll's `{ % highlight % } ... { % endhighlight % }` syntax, especially together with the `linenos` option. The generated `table` to render the line numbers does not have a CSS class or any other way of differentiating it from regular tables, so that the styles above apply, resulting in a broken page. What's more, the output

from `highlight` tags isn't even valid HTML, nesting `pre` tags inside `pre` tags, which will in break the site during minification. You can read more about it here (https://github.com/penibelst/jekyll-compress-html/issues/71) and here (https://github.com/jekyll/jekyll/issues/4432) .

Adding math

Hydejack supports math blocks (https://kramdown.gettalong.org/syntax.html#math-blocks) via KaTeX (https://khan.github.io/KaTeX/) .

Why KaTeX instead of MathJax? KaTeX is faster and more lightweight at the cost of having less features, but for the purpose of writing blog posts, this should be a favorable tradeoff.

Before you add math content, make sure you have the following in your config file:

```
kramdown:
  math_engine:          mathjax # this is not a typo
  math_engine_opts:
    preview:            true
    preview_as_code:    true
```

Inline

Example:

Lorem ipsum `f(x) = x^2` $f(x) = x^2$.

Markdown:

```
Lorem ipsum $$ f(x) = x^2 $$.
```

Block

Example:

$$
\begin{aligned}
\phi(x,y) &= \phi \left(\sum_{i=1}^n x_ie_i, \sum_{j=1}^n y_je_j \right) \\[2em]
&= \sum_{i=1}^n \sum_{j=1}^n x_i y_j \phi(e_i, e_j) \\[2em]
&= (x_1, \ldots, x_n)
\left(\begin{array}{ccc}
\phi(e_1, e_1) & \cdots & \phi(e_1, e_n) \\
\vdots & \ddots & \vdots \\
\phi(e_n, e_1) & \cdots & \phi(e_n, e_n)
\end{array}\right)
\left(\begin{array}{c}
y_1 \\
\vdots \\
y_n
\end{array}\right)
\end{aligned}
$$

Markdown:

```
$$
\begin{aligned}
  \phi(x,y) &= \phi \left(\sum_{i=1}^n x_ie_i, \sum_{j=1}^n y_je_j \right) \\[2em]
            &= \sum_{i=1}^n \sum_{j=1}^n x_i y_j \phi(e_i, e_j)            \\[2em]
            &= (x_1, \ldots, x_n)
               \left(\begin{array}{ccc}
                 \phi(e_1, e_1)  & \cdots & \phi(e_1, e_n) \\
                 \vdots          & \ddots & \vdots         \\
                 \phi(e_n, e_1)  & \cdots & \phi(e_n, e_n)
               \end{array}\right)
               \left(\begin{array}{c}
                 y_1    \\
                 \vdots \\
                 y_n
               \end{array}\right)
\end{aligned}
$$
```

NOTE: KaTeX does not support the `align` and `align*` environments. Instead, `aligned` should be used, e.g. `\begin{aligned} ... \end{aligned}`.

Scripts

There are two ways of adding third party scripts. Embedding (#embedding) is ideal for one-off scripts, e.g. `widgets.js` that is part of embedded tweets (see below). Adding global scripts (#global-scripts) is for scripts that should be loaded on every page.

Embedding

Hydejack supports embedding third party scripts directly inside markdown content. This will work in most cases, except when a script can not be loaded on a page more than once (this will occur when a user navigates to the same page twice).

NOTE: Adding "raw" script tags will make the page slow, unless they have the `async` or `defer` attribute set. For more see below (#async-vs-defer-vs-loadjsdeferred).

Example:

```
<script async src="//platform.twitter.com/widgets.js" charset="utf-8"></script>
<blockquote class="twitter-tweet" data-lang="en">
  <p lang="en" dir="ltr">
    The next version of Hydejack (v6.3.0) will allow embedding 3rd party scripts,
    like the one that comes with this tweet for example.
  </p>
  &mdash; Florian Klampfer (@qwtel)
  <a href="https://twitter.com/qwtel/status/871098943505039362">June 3, 2017</a>
</blockquote>
```

Global scripts

If you have scripts that should be included on every page you can add them globally by opening (or creating) `_includes/my-scripts.html` and adding them like you normally would:

```
<script
  src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
  integrity="sha256-k2WSCIexGzOj3Euiig+TlR8gA0EmPjuc79OEeY5L45g="
  crossorigin="anonymous"></script>
```

`my-scripts.html` will be included at the end of the `body` tag.

`async` vs. `defer` vs. `loadJSDeferred`

I highly recommended setting the `async` or `defer` attribute on your external scripts, otherwise the entire page can't finish loading until a separate HTTP request is completed, which can take a long time (this applies to the web in general, not just Hydejack).

Specific to Hydejack is the `loadJSDeferred` function, which is used to load Hydejack's own scripts. It has various advantages which are detailed in the table below.

	`async`	`defer`	`loadJSDeferred`
Download	immediately	immediately	after document `load`
Execution	asap	before document `load`	after document `load`
Ordering	none	preserves order	via callback nesting
Support	IE8+	IE9+	IE5+ (Hydejack only)

Using `loadJSDeferred`

Using `loadJSDeferred` is slightly more work than just adding `defer` to a script tag.

```
<script>
  loadJSDeferred('<script-src>', function () {
    // <callback code>
  });
</script>
```

If you have scripts that depend on other scripts, you can nest calls, e.g.

```
<script>
  loadJSDeferred('<script-src-1>', function () {
    // <callback script 1>
    loadJSDeferred('<script-src-2>', function () {
      // <callback script 1 + 2>
      loadJSDeferred('<script-src-3>', function () {
        // <callback script 1 + 2 + 3>
      });
    });
  });
</script>
```

Registering push state event listeners

When embedding scripts globally you might want to run some init code after each page load. However, the problem with push state-based page loads is that the `load` event won't fire again. Luckily, Hydejack's push state component exposes an event that you can listen to instead.

```
<script>
  document.getElementsByTagName('hy-push-state')[0].addEventListener('hy-push-stat
    // <your init code>
  });
</script>
```

Note that the above code must only run once, so include it in your `my-scripts.html`.

`hy-push-state-start`
: Occurs after clicking a link.

`hy-push-state-ready`
: Animation fished and response has been parsed, ready to swap out the content.

`hy-push-state-after`
: The old content has been replaced with the new content.

`hy-push-state-progress`
: Special case when animation is finished, but no response from server has arrived yet. This is when the loading spinner will appear.

`hy-push-state-load`

> All embedded script tags have been inserted into the document and have finished loading.

Escape hatch

If you can't make an external script work with Hydejack's push state approach to page loading, you can disable push state by adding to your config file:

```
hydejack:
  no_push_state: true
```

Build

Preparation

Before building, make sure the following is part of your config file:

```
compress_html:
  comments:  ["<!-- ", " -->"]
  clippings: all
  endings:   all
  ignore:
    envs:    [development]

sass:
  style:     compressed
```

You can check out jekyll-compress-html (https://github.com/penibelst/jekyll-compress-html) and https://jekyllrb.com/docs/assets/#sassscss (https://jekyllrb.com/docs/assets/#sassscss) for details.

Building locally

When building Hydejack it is important to set the environment variable `JEKYLL_ENV` to `production`. Otherwise the output will not be minified. Building itself happens via Jekyll's `build` command.

```
$ JEKYLL_ENV=production bundle exec jekyll build
```

This will generate the finished static files in `_site`, which can be deployed using the methods outlined in the Jekyll Documentation (https://jekyllrb.com/docs/deployment-methods/).

Building locally with latent semantic analysis

By default, related posts are simply the most recent posts. Hydejack modifies this a bit, by showing the most recent posts of the same category or tag. However, the results are still pretty "unrelated". To provide better results, Jekyll supports latent semantic analysis (https://en.wikipedia.org/wiki/Latent_semantic_analysis) via `classifier-reborn` (http://www.classifier-reborn.com/)'s Latent Semantic Indexer (http://www.classifier-reborn.com/lsi)

To use the LSI, you first have to disable Hydejack's default behavior, by setting `use_lsi: true` under the `hydejack` key in your config file.

```
hydejack:
  use_lsi: true
```

Then, you have to run `jekyll build` with the `--lsi` flag:

```
$ JEKYLL_ENV=production bundle exec jekyll build --lsi
```

Note that this may take a long time. Once it is finished, the generated static files will be located in the `_site` directory, which can be deployed using the methods outlined in the Jekyll Documentation (https://jekyllrb.com/docs/deployment-methods/).

GitHub Pages

To deploy to GitHub Pages, the steps are:

```
$ cd _site
$ git init # you only need to do this once
$ git remote add origin <github_remote_url> # you only need to do this once
$ git add .
$ git commit -m "Build"
$ git push origin master:<remote_branch>
$ cd ..
```

`github_remote_url`
: Find this on your repository's GitHub page.

`remote_branch`
: Either `master` for "user or organization pages", or `gh-pages` for "project pages"

More on user, organization, and project pages (https://help.github.com/articles/user-organization-and-project-pages/) .

Advanced

Adding a custom social media icon

Hydejack includes a number of social media icons by default (in fact, everything that is provided by IcoMoon (https://icomoon.io/)), but since the landscape is always changing, it is likely that a platform that is important to you will be missing at some point.

NOTE: You can add any platform by simply providing a complete URL. However, a fallback icon 🔗 will be used.

Creating the icon font

In order to add a custom social media icon you have to use the IcoMoon App (https://icomoon.io/app/) (free) to create a custom icon webfont. However, it is important that the generated font include all icons already in use by Hydejack. For this purpose, find the `selection.json` in `assets/icomoon/selection.json` (https://github.com/qwtel/hydejack/blob/v6/assets/icomoon/selection.json) and upload it to the app via "Import Icons". Then, use the app to add your icon(s). Consult the IcoMoon docs (https://icomoon.io/#docs) for additional help.

Once you've created and downloaded the icon font form IconMoon, replace the `icomoon` folder in `assets` in its entirety. Keep in mind that future updates of Hydejack will override this folder.

Adding the platform's metadata

For the second step it is necessary to add the network's metadata to `_data/social.yml`. An entry looks like:

```yaml
deviantart:
  name: DeviantArt
  icon: icon-deviantart
  prepend: "https://"
  append: ".deviantart.com"
```

`name`
: The name of the network. Used for for the title attribute and screen readers.

`icon`
: The icon CSS class. Can be chosen during the IcoMoon creation process.

`prepend`
: Optional. A string that is prepended to the username to form the link to the profile. If the final URL should be `https://<username>.deviantart.com`, this would be `https://`

`append`
: Optional. A string that is appended to the username to form the link to the profile. If the final URL should be `https://<username>.deviantart.com`, this would be `.deviantart.com`.

Building the JavaScript

In order to build the JavaScript you need to have node.js (https://nodejs.org/en/) installed. Specifically, the `npm` command needs to be available, which is part of node.js.

NOTE: Building the JavaScript is optional! Hydejack comes with a pre-built, minified `hydejack.js` file that you can find in part of the theme's `assets`.

Before you start, make sure you've copied the following files:

- `_js/`
- `package.json`
- `package-lock.json`
- `.babelrc`
- `.eslintignore`
- `.eslintrc`

When building for the first time (and after each update of Hydejack) you have to run

```
$ npm install
```

to fetch all dependencies (and put them in a local folder `node_modules`), lint the code and write the bundled and minified script into `assets/js/hydejack.js`.

You can re-build it with

```
$ npm run build:js
```

If you want to actively develop the scripts, it is better to run

```
$ npm run watch:js
```

which will build a non-minified version of `assets/js/hydejack.js` after each filechange.

How CSS is organized in Hydejack

Hydejack takes a quite unique approach to CSS, which is motivated by the ability to inline essential CSS rules in a `style` tag in the `<head/>` of a page (to increase the loading speed), while serving the rest in a separate file.

The styles are written in SCSS and are located in the `_sass` folder, which looks like

```
├── hydejack
│   ├── __inline
│   ├── __link
│   ├── _base.pre.scss
│   ├── ...
│   └── _social.pre.scss
├── pooleparty
│   ├── __inline
│   ├── __link
│   ├── _base.pre.scss
│   ├── ...
│   └── _type.pre.scss
├── mixins.scss
├── my-inline.scss
├── my-style.scss
├── syntax.scss
└── variables.scss
```

The style rules are organized alongside components (or rather, topics) like “sidebar” and “footer”. Further, there are two separate frameworks, “pooleparty” and “hydejack”, which grew out of the original Poole (http://getpoole.com/) and Hyde (http://hyde.getpoole.com/) projects. Poole/party contains more general style rules, while Hyde/jack contains those that more are specific to the theme. However, this separation has become more blurry over time.

Inside those folders, you will notice the `__inline` and `__link` folders. The unfriendly names are intentional, because their contents are generated by a script and shouldn’t be modified directly. The source files are located in the same folder and end with `.pre.scss`. They are fully valid SCSS files, but contain comments that mark which lines should be inlined and which should be fetched asynchronously.

The rules are as follows:

- Every line between `// <<< inline` and `// >>>` will be inlined
- Every line between `// <<< link` and `// >>>` will be linked
- Every line that isn’t contained in a block and ends with `// inline` will be inlined
- Every line that isn’t contained in a block and ends with `// link` will be linked
- Every line for which none of the above applies will be included in both.

The actual splitting happen with the `_scripts/build-css.sh` script (requires node.js 8+). You can run the script once by using

```
$ npm run build:css
```

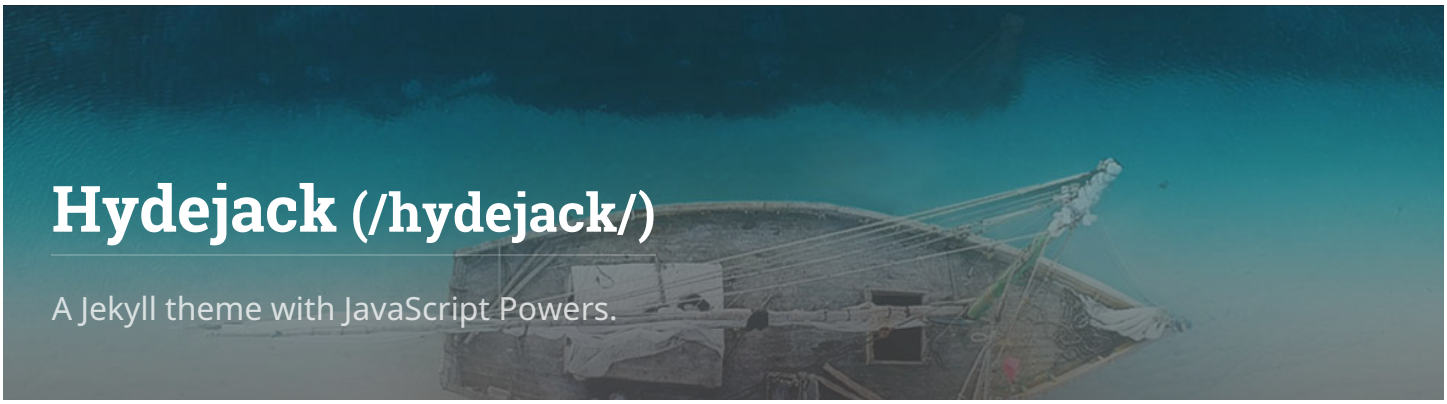
or rebuild the CSS on every file change

```
$ npm run watch:css
```

Note that `my-inline.scss` and `my-style.scss` are not affected by this. Also, since all files are valid SCSS, the splitting part is entirely optional. If you would like to build just one regular CSS file, add

```
hydejack:
  no_inline_css: true
```

to your config file.



Hydejack (/hydejack/)

A Jekyll theme with JavaScript Powers.