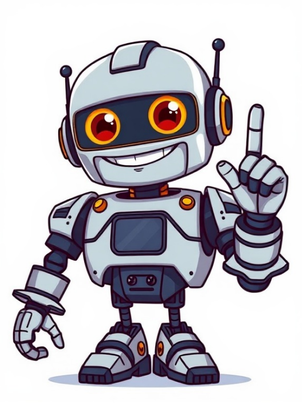
I'm not a bot



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 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, _data, _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 , replacing 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 data, 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: 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. 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/3.4.1/gems/minima-2.5.1 # # then invoke explorer with above path, substituting `/` with `\` explorer C:\Ruby26-x64\lib\ruby\gems\3.4.1\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 /_data /_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. Themes with _data directory4.3.0 Starting with version 4.3.0, Jekyll also takes into account the _data directory of themes. This allows data to be distributed across themes. A typical example is text used within design elements. Imagine a theme provides the include file testimonials.html. This design element creates a new section on the page, and puts a h3 heading over the list of testimonials. A theme developer will probably formulate the heading in English and put it directly into the HTML source code. Consumers of the theme can copy the included file into their project and replace the heading there. With the consideration of the _data directory there is another solution for this standard task. Instead of entering the text directly into the design template, the designer adds a reference to a text catalog (e.g. site.data.i18n.testimonials.header) and create a file _data/i18n/testimonials.yml in the data directory of the theme. In this file the header is put under the key header and Jekyll takes care of the rest. For theme developers, this, at first sight, is of course a bigger effort than before. However, for the consumers of the theme, the customization is greatly simplified. Imagine the theme is used by a customer from Germany. In order for her to get the translated header for the testimonials design element in, she just has to create a data file in her project directory with the key site.data.i18n.testimonials.header, put the German translation or a header of her choice on top of it and the design element is already customized. She no longer has to copy the included file into her project directory, customize it there and, what weighs heaviest, waiver all updates of the theme, simply because the theme developer offered her the possibility to make changes to text modules centrally via text files. Data files provide a high degree of flexibility. The place where theme developers put text modules may differ from that of the consumer of the theme which can cause unforeseen troubles! Related to above example the overriding key site.data.i18n.testimonials.header from the theme’s _data/i18n/testimonials.yml file on the consumer site can be located in three different locations: _data/i18n.yml with key testimonials.header _data/i18n/testimonials.yml with key header (which mirrors the layout of the given example) _data/i18n/testimonials/header.yml without any key, the headline can go straight into the file Theme developers should have this ambiguity in mind, when supporting consumers that feel lost in setting their text modules for the design elements the theme provides. When using the data feature ask yourself, is the key that you introduce something that changes the behaviour of the theme when present or not, or is it just data that’s displayed anyway. If it’s changing the behaviour of the theme it should go into site.config otherwise it’s fine to be provided via site.data. Bundling data that modifies the behavior of a theme is considered an anti-pattern whose use is strongly discouraged. It is solely up to the author of the theme to ensure that every provided data can be easily overridden by the consumer of the theme if they desire to. 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 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: 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" Install the theme: Add the following to your site’s _config.yml to activate the theme: theme: jekyll-theme-minimal Build your site: 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. First, you need to have it in a git repository: git init # Only the first time git add -A git commit -m "Init commit" 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 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 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. Jekyll’s growing community produces wide variety of themes, plugins, tutorials and other resources that can be helpful. Below is a collection of links to some of the most popular Jekyll resources. Themes See also: docs/themes. Plugins jekyll-plugin topic on GitHub Planet Jekyll Guides Integrations Use a SaaS service as a backend for functionality on your Jekyll site Staticman: Add user-generated content to a Jekyll site (free and open source) Talkyard: Embedded comments for Jekyll and others (free and open source, or hosted serverless) Content Management CloudCannon: The Cloud CMS for Jekyll Contentful: Content infrastructure for digital teams TinaCMS.io: Tina is an open-source, Git-backed headless content management system (CMS) that empowers both developers and content creators to collaborate seamlessly on a single platform Decap CMS: Decap CMS (formerly Netlify CMS) is an open source content management system for your Git workflow that enables you to provide editors with a friendly UI and intuitive workflows Siteleaf: Built for developers, Loved by everyone Kentico Kontent: A headless CMS with full control over content presentation E-commerce MemberSpace: Add memberships and paywall functionality to a Jekyll site Snipcart: Add a shopping cart to a Jekyll site Forms Search Algolia: Add a powerful instant search to your Jekyll site Elastic Site Search: Another option for adding search to your Jekyll site, built on Elasticsearch Bonsai Search: The easiest way to use Elasticsearch for your Jekyll site CloudSh: Website search with a few lines of JavaScript Editors plugins Visual Studio Code has various jekyll related plugins and supports autocompletion for configuration file. jekyll-atom: A collection of snippets and tools for Jekyll in Atom markdown-writer: An Atom package for Jekyll. It can create new posts/drafts, manage tags/categories, insert link/images and add many useful key mappings. sublime-jekyll: A Sublime Text package for Jekyll static sites. This package should help creating Jekyll sites and posts easier by providing access to key template tags and filters, as well as common completions and a current date/datetime command (for dating posts). You can install this package manually via GitHub, or via Package Control. vim-jekyll: A vim plugin to generate new posts and run jekyll build all without leaving vim. WordPress2Jekyll: A WordPress plugin that allows you to use WordPress as your editor and (automatically) export content into Jekyll. WordPress2Jekyll attempts to marry these two systems together in order to make a site that can be easily managed from all devices. Posts Forks Time to Visit Bridgetown Creating a Faster Jekyll Jekyll is a Ruby Gem that can be installed on most systems. Requirements Ruby version 2.7.0 or higher, including all development headers (check your Ruby version using ruby -v) RubyGems (check your Gems version using gem -v) GCC and Make (check versions using gcc -v,g++ -v, and make -v) Guides For detailed install instructions, follow the guide for your operating system. macOS Ubuntu FreeBSD Other Linux Windows Quickstart Installation Ruby 101 Community Step by Step Tutorial Command Line Usage Configuration Rendering Process Pages Posts Front Matter Collections Data Files Assets Static Files Plugins Blog Migrations Upgrading Deployment GitHub Pages are public web pages for users, organizations, and repositories, that are freely hosted on GitHub’s github.io domain or on a custom domain name of your choice. GitHub Pages are powered by Jekyll behind the scenes, so they’re a great way to host your Jekyll-powered website for free. Your site is automatically generated by GitHub Pages when you push your source files. Note that GitHub Pages works equally well for regular HTML content, simply because Jekyll treats files without front matter as static assets. So if you only need to push generated HTML, you’re good to go without any further setup. The GitHub Pages Documentation is comprehensive and includes a a guide to setting up a GitHub Pages site using Jekyll. We recommend following this guide. This page contains some additional information which may be useful when working on GitHub Pages sites with Jekyll. Project Page URL Structure Sometimes it’s nice to preview your Jekyll site before you push your gh-pages branch to GitHub. The subdirectory-like URL structure GitHub uses for Project Pages complicates the proper resolution of URLs. In order to assure your site builds properly, use the handy URL filters: [{{ page.title }}]("{{ page.url | relative_url }}") This way you can preview your site locally from the site root on localhost, but when GitHub generates your pages from the gh-pages branch all the URLs will resolve properly. Deploying Jekyll to GitHub Pages GitHub Pages work by looking at certain branches of repositories on GitHub. There are two basic types available: user/organization and project pages. The way to deploy these two types of sites are nearly identical, except for a few minor details. User and Organization Pages User and organization pages live in a special GitHub repository dedicated to only the GitHub Pages files. This repository must be named after the account name. For example, @mojombo’s user page repository has the name mojombo.github.io. Content from the master branch of your repository will be used to build and publish the GitHub Pages site, so make sure your Jekyll site is stored there. Project Pages Unlike user and organization Pages, Project Pages are kept in the same repository as the project they are for, except that the website content is stored in a specially named gh-pages branch or in a docs folder on the master branch. The content will be rendered using Jekyll, and the output will become available under a subpath of your user pages subdomain, such as username.github.io/project (unless a custom domain is specified). The Jekyll project repository itself is a perfect example of this branch structure—the master branch contains the actual software project for Jekyll, and the Jekyll website that you’re looking at right now is contained in the docs folder of the same repository. Please refer to GitHub official documentation on user, organization and project pages to see more detailed examples. GitHub Pages overrides the “Site Source” configuration value, so if you locate your files anywhere other than the root directory, your site may not build correctly. Running and Testing Locally Once the project is configured with the github-pages environment, it’s quite hard to switch back and forth with the local settings and the production-level settings. For that we can use certain CLI options to make the workflow hassle-free. bundle exec jekyll serve --baseurl="" This will run the jekyll server on your local machine i.e. on . Refer server options for available options. The Jekyll gem makes a jekyll executable available to you in your terminal.jekyll command [argument] [option] [argument_to_option] Examples: jekyll new site/ --blank jekyll serve --config _alternative_config.yml Typically you’ll use jekyll serve while developing locally and jekyll build when you need to generate the site for production.To change Jekyll’s default build behavior have a look through the configuration options. Welcome to Jekyll’s step-by-step tutorial. This tutorial takes you from having some front-end web development experience to building your first Jekyll site from scratch without relying on the default gem-based theme. Installation Jekyll is a Ruby gem. First, install Ruby on your machine. Go to Installation and follow the instructions for your operating system. With Ruby installed, install Jekyll from the terminal: gem install jekyll bundler Create a new Gemfile to list your project’s dependencies: Edit the Gemfile in a text editor and add jekyll as a dependency: Run bundle to install jekyll for your project. You can now prefix all jekyll commands listed in this tutorial with bundle exec to make sure you use the jekyll version defined in your Gemfile. Create a site It’s time to create a site! Create a new directory for your site and name it whatever you want. Through the rest of this tutorial we’ll refer to this directory as root. You can also initialize a Git repository here. One of the great things about Jekyll is there’s no database. All content and site structure are files that a Git repository can version. Using a repository is optional but is recommended. You can learn more about using Git by reading the Git Handbook. Let’s add your first file. Create index.html in root with the following content: Home Hello World! Build Since Jekyll is a static site generator, it has to build the site before we can view it. Run either of the following commands to build your site: jekyll build - Builds the site and outputs a static site to a directory called _site. jekyll serve - Does jekyll build and runs it on a local web server at , rebuilding the site any time you make a change. When you’re developing a site, use jekyll serve. To force the browser to refresh with every change, use jekyll serve --livereload. If there’s a conflict or you’d like Jekyll to serve your development site at a different URL, use the --host and --port arguments, as described in the serve command options. The version of the site that jekyll serve builds in _site is not suited for deployment. Links and asset URLs in sites created with jekyll serve will use or the value set with command-line configuration, instead of the values set in your site’s configuration file. To learn about how to build your site when it’s ready for deployment, read the Deployment section of this tutorial. Run jekyll serve and go to in your browser. You should see “Hello World!”. At this point, you might be thinking, “So what?”. The only thing that happened was that Jekyll copied an HTML file from one place to another. Patience, young grasshopper, there’s still much to learn! Next. you’ll learn about Liquid and templating. Jekyll uses the Liquid templating language to process templates. Generally in Liquid you output content using two curly braces e.g. {{ variable }} and perform logic statements by surrounding them in a curly brace percentage sign e.g. {% if statement %}. To learn more about Liquid, check out the official Liquid Documentation. Jekyll provides a number of useful Liquid additions to help you build your site: Quickstart Installation Ruby 101 Community Step by Step Tutorial Command Line Usage Configuration Rendering Process Pages Posts Front Matter Collections Data Files Assets Static Files Plugins Blog Migrations Upgrading Deployment Jekyll is a static site generator. It takes text written in your favorite markup language and uses layouts to create a static website. You can tweak the site’s look and feel, URLs, the data displayed on the page, and more. Prerequisites Jekyll requires the following: Ruby version 2.7.0 or higher RubyGems GCC and Make See Requirements for guides and details. Instructions Install all prerequisites. Install the jekyll and bundler gems. gem install jekyll bundler Create a new Jekyll site at ./myblog. Change into your new directory. Build the site and make it available on a local server. Browse to If you are using Ruby version 3.0.0 or higher, step 5 may fail. You may fix it by adding webrick to your dependencies: bundle add webrick Pass the --livereload option to serve to automatically refresh the page with each change you make to the source files: bundle exec jekyll serve --livereload If you encounter any errors during this process, check that you have installed all the prerequisites in Requirements. If you still have issues, see Troubleshooting. Installation varies based on your operating system. See our guides for OS-specific instructions. Sick of dealing with hosting companies? GitHub Pages is powered by Jekyll, so you can easily deploy your site using GitHub for free—custom domain name and all. Learn more about GitHub Pages → In contrast to Docs, Tutorials provide more detailed, narrative instruction that cover a variety of Jekyll topics and scenarios. Tutorials might contain the following: Step-by-step processes through particular scenarios or challenges Full walk-throughs using sample data, showing inputs and results from the sample data Detailed explanation about the pros and cons for different Jekyll strategies End-to-end instruction in developing a complete feature on a Jekyll site Instruction that combines various techniques from across the docs In short, tutorials aren’t the core reference information in docs. They walk users through processes from beginning to end. The Tutorials section is new, so there aren’t many tutorials yet. You can add a tutorial here to help populate this section. Some of these techniques might even guide you through a supporting tool, script, service, or other hack used with your Jekyll site. Feel free to include tutorials involving external services with Jekyll as well. However, note that Jekyll in no way endorses any third-party tools mentioned in tutorials. How to contribute a tutorial We welcome your tutorial contributions. To add your tutorial: Fork the Jekyll project by clicking the Fork button in the upper-right corner of the jekyll/jekyll project GitHub repo. Add your tutorial in the _tutorials collection. Make sure your tutorial has the same front matter items as other tutorial items. Add a reference to your tutorial filename in _data/tutorials.yml. This allows your tutorial to appear in the Tutorials sidebar. Follow the regular git workflow to submit the pull request. When you submit your pull request, the Jekyll documentation team will review your contribution and either merge it or suggest edits.

- http://novastal.pl/userfiles/file/c73d2981-fcd1-4143-8386-10745ce04191.pdf
- what are physiological functions
- national guidelines for newborn care sri lanka
- https://nguoixunghekiev.vn/userfiles/file/13235341492.pdf
- https://funbugs.ie/userfiles/file/13672673463.pdf
- roots (1977 full movie free)
- what is a formal invitation meaning
- how to play pokemon fan games on chromebook
- james dyson famous inventions
- febepalaso