Introduction

GitHub Pages are free static web sites hosted directly from your GitHub repository. Using standard technologies like YAML and Markdown, anyone can build and maintain a site in minutes.

Suppose you want to set up a basic web site to share information about the project you're working on. It might be a personal site, like as resume or portfolio. Or it might be a professional site, like a user guide or developer reference. With GitHub Pages, you can spin the site up in minutes and enable anyone with a basic understanding of Markdown to contribute to it. You also get all the benefits of GitHub for source control, including branches and pull requests.

In this module, you learn how to set up and use GitHub Pages to host both personal and professional sites.

Learning objectives

In this module, you will:

- Enable GitHub Pages
- Choose a theme with Jekyll
- Use YAML front matter
- Customize your site
- Create and edit blog posts

Prerequisites

- A GitHub account
- The ability to navigate and edit files in GitHub

It is recommended that you complete Communicate effectively on GitHub by using Markdown before beginning this module.

Next unit: What is GitHub Pages?

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue.

What is GitHub Pages?

docs.microsoft.com/en-us/learn/modules/create-host-web-sites-github-pages/2-what-is-github-pages

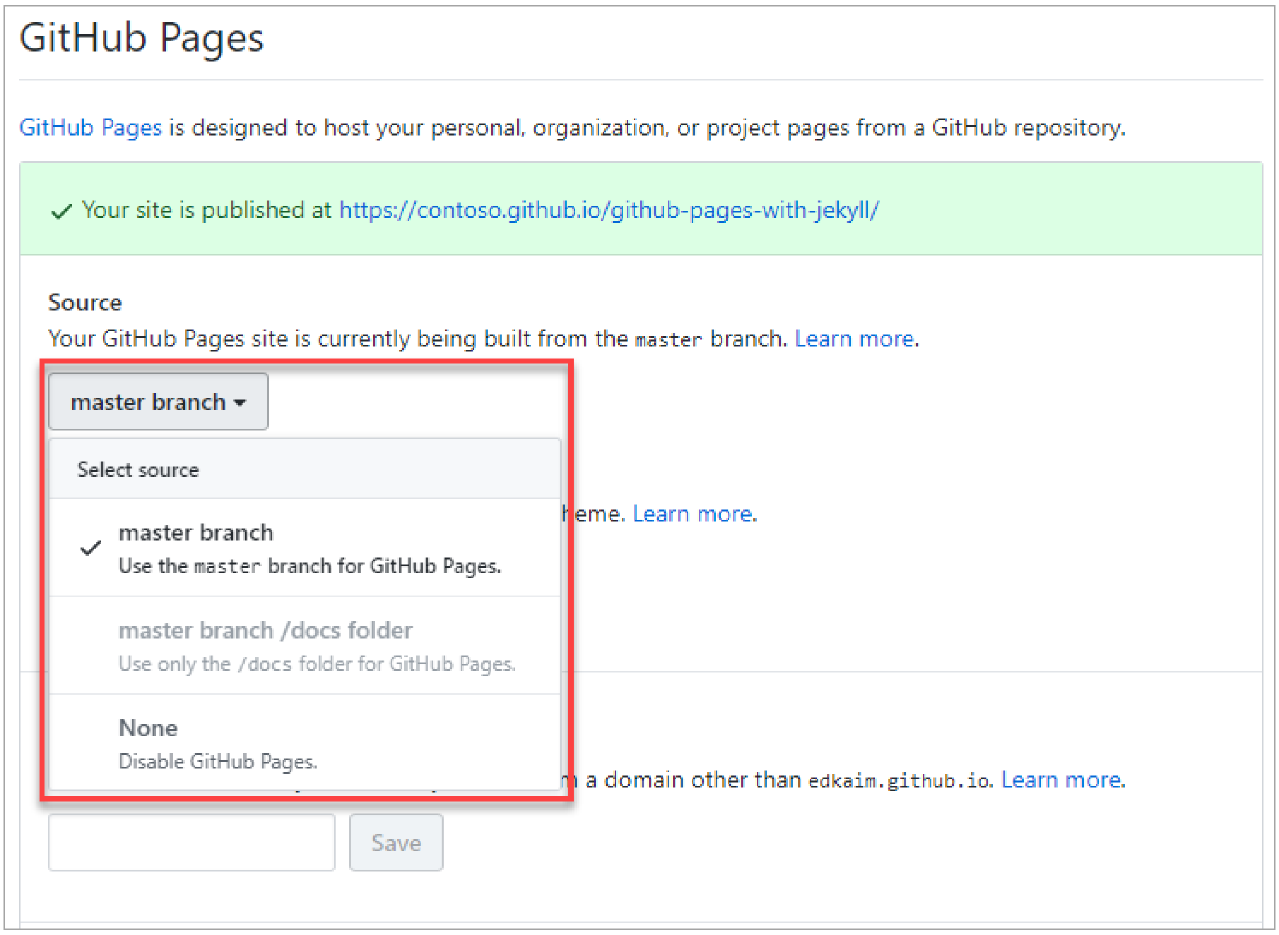
Here, we discuss the process of creating and maintaining a GitHub Pages web site.

GitHub Pages are static sites hosted directly from your GitHub repository. But they're more than just a collection of static files. By making use of site generation technologies like Jekyll and Liquid, developers define dynamic templates that are processed into complete static web sites. Every time a change is committed to the source branch associated with the site, it is re-generated using the latest updates and automatically published to the target URL.

Learn more about Publishing sources for GitHub Pages sites.

Enabling GitHub Pages

The first step in using GitHub Pages is to enable it from your repository's **Settings** tab. You can opt to use the `master` branch, or specify the `docs` folder within it. If you ever want to disable GitHub Pages, you can do so here.



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

✓ Your site is published at https://contoso.github.io/github-pages-with-jekyll/

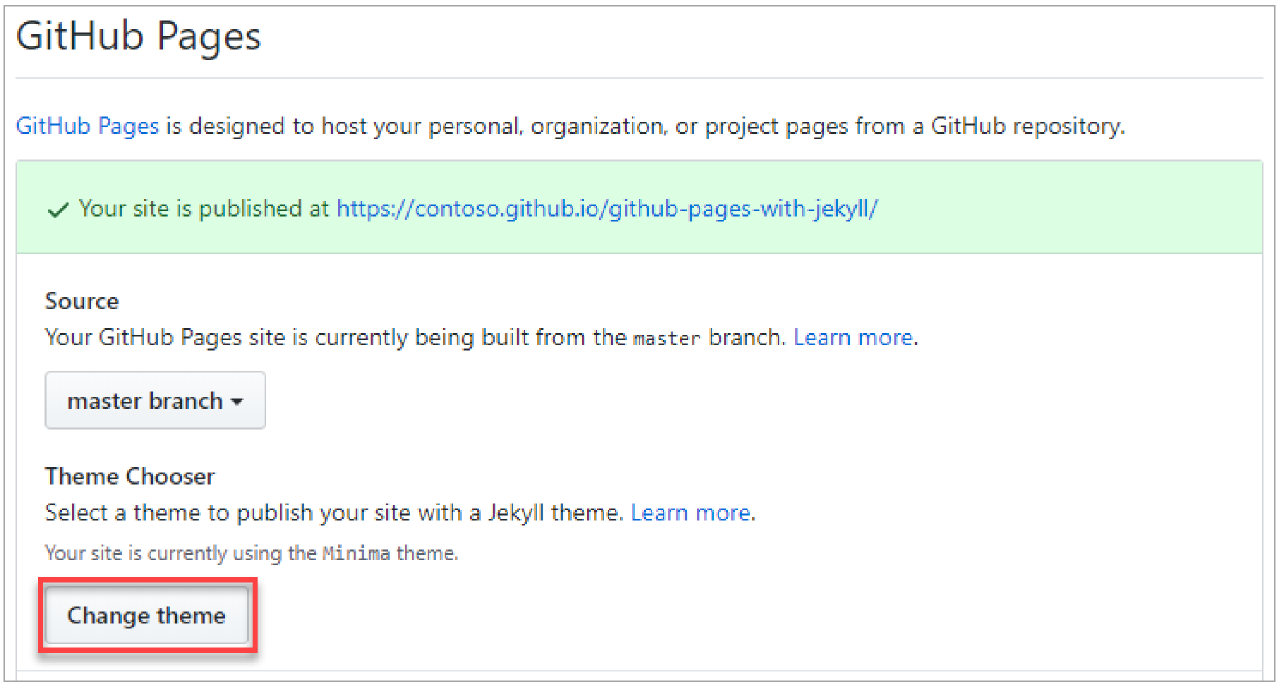
Source

Your GitHub Pages site is currently being built from the `master` branch. Learn more.

master branch ▾
Select source
✓ master branch
Use the master branch for GitHub Pages.
master branch /docs folder
Use only the /docs folder for GitHub Pages.
None
Disable GitHub Pages.
heme. Learn more.
n a domain other than edkaim.github.io. Learn more.
Save

Choosing a theme with Jekyll

Jekyll is the static site generator used by GitHub to build your web site from the contents of your repository. In addition to providing great content convenience, it also conforms to a standard design convention. This style standardization allows for swappable themes, which you can select from the **GitHub Pages** configuration.



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

✓ Your site is published at https://contoso.github.io/github-pages-with-jekyll/

Source

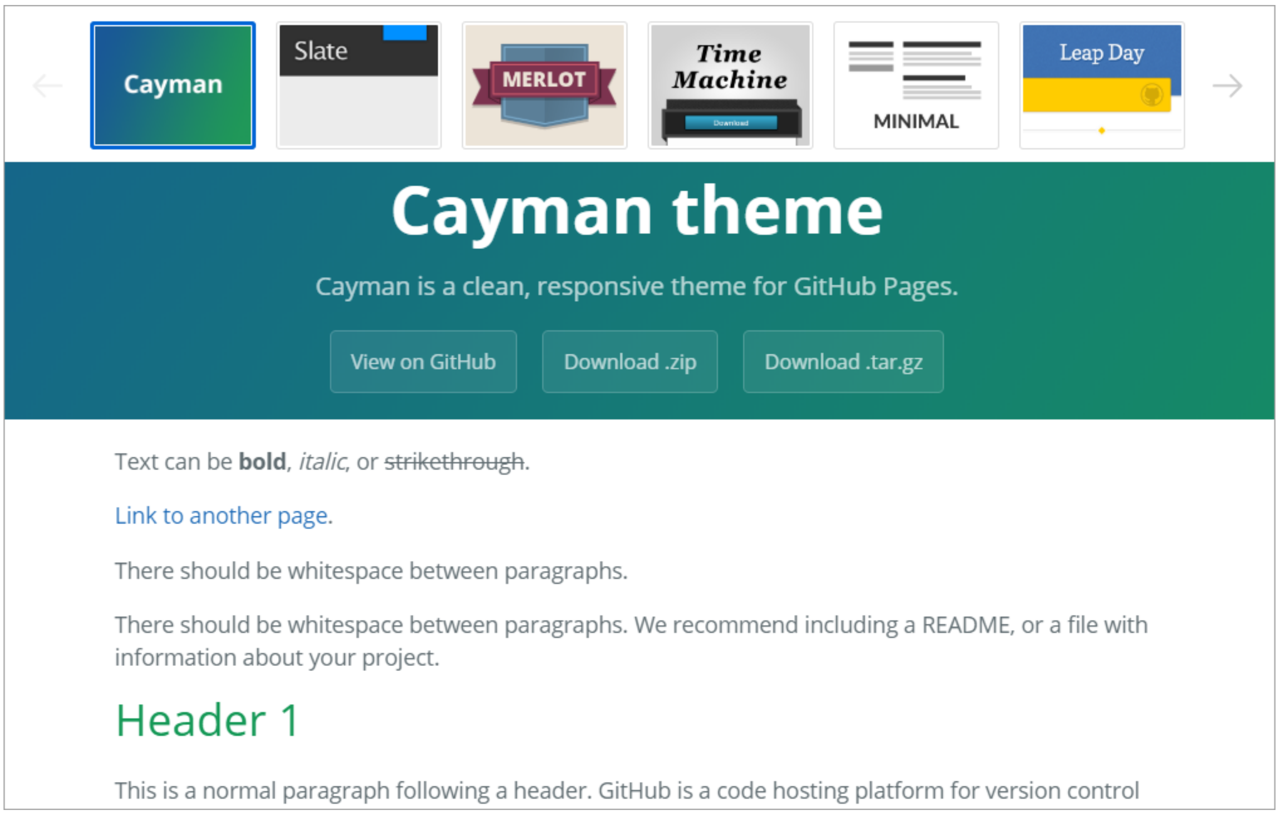
Your GitHub Pages site is currently being built from the master branch. Learn more.

Theme Chooser

Select a theme to publish your site with a Jekyll theme. Learn more.

Your site is currently using the Minima theme.

There are a variety of themes provided by GitHub. There is also an array of commercial and open source themes available from the Jekyll community.



Cayman theme

Cayman is a clean, responsive theme for GitHub Pages.

Text can be **bold**, *italic*, or ~~strikethrough~~.

Link to another page.

There should be whitespace between paragraphs.

There should be whitespace between paragraphs. We recommend including a README, or a file with information about your project.

Header 1

This is a normal paragraph following a header. GitHub is a code hosting platform for version control

Learn more about Jekyll Themes.

Using YAML front matter

The term *front matter* refers to YAML metadata that prepends the content of a file. For Jekyll, this includes generator instructions to indicate the layout style of a Markdown page (`post` , `page` , and so on). It may also include page metadata, such as the document title, or page content variables, such as a blog post's author.

Below is a simple example that would use the `post` layout. This assumes there is a `_layouts/post.html` file that defines the container HTML. Other layout options may be offered by adding their respective HTML files in the `_layouts` folder.

yml

```
---
layout: post
title: This is set as the document title.
---

This is visible body content, which may use Markdown, HTML, and Liquid templating.
```

Learn more about Front Matter.

Customizing your site

Once your site is up and running, you can customize details about your site via `_config.yml` . This file includes virtually all site-wide configuration options, including site metadata, navigation menus, theme colors, compiler options, and more.

Learn more about _config.yml Configuration.

Creating and editing content

Creating and editing pages on your site follows the standard GitHub experience. The files you use for your GitHub Pages web site enjoy all of the same benefits as other files in your GitHub repository, so you can edit them with any tool, create and merge branches, and link with issues or pull requests.

In addition to Markdown and HTML, Jekyll supports the **Liquid** template language syntax. Liquid provides the ability for users to dynamically insert variables and basic logic flow constructs into their content files. When compiled, the final product is standard HTML.

The example below shows a combination of `for` looping and variable insertion.

markdown

```
<ul>
  {% for post in site.posts %}
    <li>
      <h2><a href="{{ post.url }}">{{ post.title }}</a></h2>
      {{ post.excerpt }}
    </li>
  {% endfor %}
</ul>
```

Learn more about Liquid template language.

Working with blog posts

Despite not having a database to work with, Jekyll still supports the concept of blogging using a specific convention: `_posts/2020-06-25-blog-post-name.md`. As you can likely infer, all blog posts are stored in the `_posts` folder and use the date and name convention as shown. During compilation, Jekyll processes the files in this folder to produce a list of HTML blog posts.

The example below illustrates the structure of a simple blog post. It includes metadata for `subtitle`, `tags`, and `comments`, which may or may not be supported by the theme you choose.

markdown

```
---
layout: post
title: Blog post title rendered by theme
subtitle: Blog post subtitle rendered by theme
tags: welcoming
comments: true
---

This is the first line of rendered content in the post.
```

Learn more about Adding content to your GitHub Pages site using Jekyll.

Next unit: Exercise - Enable, create, and update a GitHub Pages web site

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue.

About GitHub Pages

You can use GitHub Pages to host a website about yourself, your organization, or your project directly from a GitHub repository.

> GitHub Pages is available in public repositories with GitHub Free and GitHub Free for organizations, and in public and private repositories with GitHub Pro, GitHub Team, GitHub Enterprise Cloud, and GitHub Enterprise Server. For more information, see "GitHub's products."

GitHub Pages is a static site hosting service that takes HTML, CSS, and JavaScript files straight from a repository on GitHub, optionally runs the files through a build process, and publishes a website. You can see examples of GitHub Pages sites in the GitHub Pages examples collection.

You can host your site on GitHub's `github.io` domain or your own custom domain. For more information, see "Using a custom domain with GitHub Pages."

If your project site is published from a private or internal repository owned by an organization using GitHub Enterprise Cloud, you can manage access control for the site. In an enterprise with managed users, all GitHub Pages sites are privately published. For more information, see "Changing the visibility of your GitHub Pages site."

To get started, see "Creating a GitHub Pages site."

Organization owners can disable the publication of GitHub Pages sites from the organization's repositories. For more information, see "Managing the publication of GitHub Pages sites for your organization."

Types of GitHub Pages sites

There are three types of GitHub Pages sites: project, user, and organization. Project sites are connected to a specific project hosted on GitHub, such as a JavaScript library or a recipe collection. User and organization sites are connected to a specific GitHub account.

To publish a user site, you must create a repository owned by your user account that's named `<username>.github.io`. To publish an organization site, you must create a repository owned by an organization that's named `<organization>.github.io`. Unless you're using a custom domain, user and organization sites are available at `http(s)://<username>.github.io` or `http(s)://<organization>.github.io`.

The source files for a project site are stored in the same repository as their project. Unless you're using a custom domain, project sites are available at `http(s)://<username>.github.io/<repository>` or `http(s)://<organization>.github.io/<repository>`.

If you publish your site privately, the URL for your site will be different. For more information, see "Changing the visibility of your GitHub Pages site."

For more information about how custom domains affect the URL for your site, see "About custom domains and GitHub Pages."

You can only create one user or organization site for each account on GitHub. Project sites, whether owned by an organization or a user account, are unlimited.

Publishing sources for GitHub Pages sites

The publishing source for your GitHub Pages site is the branch and folder where the source files for your site are stored.

> **Warning**: GitHub Pages sites are publicly available on the internet by default, even if the repository for the site is private or internal. If your project site is published from a private or internal repository owned by an organization using GitHub Enterprise Cloud, you can manage access control for the site. In an enterprise with managed users, all GitHub Pages sites are privately published. Otherwise, if you have sensitive data in your site's repository, you may want to remove the data before publishing. For more information, see "About repositories" and "Changing the visibility of your GitHub Pages site."

If the default publishing source exists in your repository, GitHub Pages will automatically publish a site from that source. The default publishing source for user and organization sites is the root of the default branch for the repository. The default publishing source for project sites is the root of the `gh-pages` branch.

If you want to keep the source files for your site in a different location, you can change the publishing source for your site. You can publish your site from any branch in the repository, either from the root of the repository on that branch, `/`, or from the `/docs` folder on that branch. For more information, see "Configuring a publishing source for your GitHub Pages site."

If you choose the `/docs` folder of any branch as your publishing source, GitHub Pages will read everything to publish your site, including the *CNAME* file, from the `/docs` folder. For example, when you edit your custom domain through the GitHub Pages settings, the custom domain will write to `/docs/CNAME`. For more information about *CNAME* files, see "Managing a custom domain for your GitHub Pages site."

Static site generators

GitHub Pages publishes any static files that you push to your repository. You can create your own static files or use a static site generator to build your site for you. You can also customize your own build process locally or on another server. We recommend Jekyll, a static site generator with built-in support for GitHub Pages and a simplified build process. For more information, see "About GitHub Pages and Jekyll."

GitHub Pages will use Jekyll to build your site by default. If you want to use a static site generator other than Jekyll, disable the Jekyll build process by creating an empty file called `.nojekyll` in the root of your publishing source, then follow your static site generator's instructions to build your site locally.

GitHub Pages does not support server-side languages such as PHP, Ruby, or Python.

Guidelines for using GitHub Pages

- GitHub Pages sites created after June 15, 2016 and using `github.io` domains are served over HTTPS. If you created your site before June 15, 2016, you can enable HTTPS support for traffic to your site. For more information, see "Securing your GitHub Pages with HTTPS."
- GitHub Pages sites shouldn't be used for sensitive transactions like sending passwords or credit card numbers.
- Your use of GitHub Pages is subject to the GitHub Terms of Service, including the prohibition on reselling.

Usage limits

GitHub Pages sites are subject to the following usage limits:

- GitHub Pages source repositories have a recommended limit of 1GB. For more information, see "What is my disk quota?"

- Published GitHub Pages sites may be no larger than 1 GB.

- GitHub Pages sites have a *soft* bandwidth limit of 100GB per month.

- GitHub Pages sites have a *soft* limit of 10 builds per hour.

If your site exceeds these usage quotas, we may not be able to serve your site, or you may receive a polite email from GitHub Support suggesting strategies for reducing your site's impact on our servers, including putting a third-party content distribution network (CDN) in front of your site, making use of other GitHub features such as releases, or moving to a different hosting service that might better fit your needs.

Prohibited uses

GitHub Pages is not intended for or allowed to be used as a free web hosting service to run your online business, e-commerce site, or any other website that is primarily directed at either facilitating commercial transactions or providing commercial software as a service (SaaS).

In addition, GitHub does not allow GitHub Pages to be used for certain purposes or activities. For a list of prohibited uses, see "GitHub's Additional Product Terms for GitHub Pages."

MIME types on GitHub Pages

A MIME type is a header that a server sends to a browser, providing information about the nature and format of the files the browser requested. GitHub Pages supports more than 750 MIME types across thousands of file extensions. The list of supported MIME types is generated from the mime-db project.

While you can't specify custom MIME types on a per-file or per-repository basis, you can add or modify MIME types for use on GitHub Pages. For more information, see the mime-db contributing guidelines.

Further reading

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.

Front Matter | Jekyll • Simple, blog-aware, static sites

jekyllrb.com/docs/front-matter

October 10, 2021

Improve this page

Front Matter

Any file that contains a YAML front matter block will be processed by Jekyll as a special file. The front matter must be the first thing in the file and must take the form of valid YAML set between triple-dashed lines. Here is a basic example:

```
---
layout: post
title: Blogging Like a Hacker
---
```

Between these triple-dashed lines, you can set predefined variables (see below for a reference) or even create custom ones of your own. These variables will then be available for you to access using Liquid tags both further down in the file and also in any layouts or includes that the page or post in question relies on.

UTF-8 Character Encoding Warning

If you use UTF-8 encoding, make sure that no `BOM` header characters exist in your files or very, very bad things will happen to Jekyll. This is especially relevant if you're running Jekyll on Windows.

Front Matter Variables Are Optional

If you want to use Liquid tags and variables but don't need anything in your front matter, just leave it empty! The set of triple-dashed lines with nothing in between will still get Jekyll to process your file. (This is useful for things like CSS and RSS feeds!)

Predefined Global Variables

There are a number of predefined global variables that you can set in the front matter of a page or post.

Variable	Description

Variable	Description
`layout`	If set, this specifies the layout file to use. Use the layout file name without the file extension. Layout files must be placed in the `_layouts` directory. • Using `null` will produce a file without using a layout file. This is overridden if the file is a post/document and has a layout defined in the front matter defaults. • Starting from version 3.5.0, using `none` in a post/document will produce a file without using a layout file regardless of front matter defaults. Using `none` in a page will cause Jekyll to attempt to use a layout named "none".
`permalink`	If you need your processed blog post URLs to be something other than the site-wide style (default `/year/month/day/title.html`), then you can set this variable and it will be used as the final URL.
`published`	Set to false if you don’t want a specific post to show up when the site is generated.

Render Posts Marked As Unpublished

To preview unpublished pages, run `jekyll serve` or `jekyll build` with the `--unpublished` switch. Jekyll also has a handy drafts feature tailored specifically for blog posts.

Custom Variables

You can also set your own front matter variables you can access in Liquid. For instance, if you set a variable called `food`, you can use that in your page:

```
---
food: Pizza
---

<h1>{{ page.food }}</h1>
```

Predefined Variables for Posts

These are available out-of-the-box to be used in the front matter for a post.

Variable	Description
`date`	A date here overrides the date from the name of the post. This can be used to ensure correct sorting of posts. A date is specified in the format `YYYY-MM-DD HH:MM:SS +/-TTTT`; hours, minutes, seconds, and timezone offset are optional.

Variable	Description
`category` `categories`	Instead of placing posts inside of folders, you can specify one or more categories that the post belongs to. When the site is generated the post will act as though it had been set with these categories normally. Categories (plural key) can be specified as a YAML list or a space-separated string.
`tags`	Similar to categories, one or multiple tags can be added to a post. Also like categories, tags can be specified as a YAML list or a space-separated string.

Don't repeat yourself

If you don't want to repeat your frequently used front matter variables over and over, define defaults for them and only override them where necessary (or not at all). This works both for predefined and custom variables.

Configuration

Improve this page

Jekyll gives you a lot of flexibility to customize how it builds your site. These options can either be specified in a `_config.yml` or `_config.toml` file placed in your site's root directory, or can be specified as flags for the `jekyll` executable in the terminal.

- Incremental Regeneration

Adding content to your GitHub Pages site using Jekyll

docs.github.com/en/pages/setting-up-a-github-pages-site-with-jekyll/adding-content-to-your-github-pages-site-using-jekyll

You can add a new page or post to your Jekyll site on GitHub Pages.

GitHub Pages is available in public repositories with GitHub Free and GitHub Free for organizations, and in public and private repositories with GitHub Pro, GitHub Team, GitHub Enterprise Cloud, and GitHub Enterprise Server. For more information, see "GitHub's products."

People with write permissions for a repository can add content to a GitHub Pages site using Jekyll.

About content in Jekyll sites

Before you can add content to a Jekyll site on GitHub Pages, you must create a Jekyll site. For more information, see "Creating a GitHub Pages site with Jekyll."

The main types of content for Jekyll sites are pages and posts. A page is for standalone content that isn't associated with a specific date, such as an "About" page. The default Jekyll site contains a file called `about.md`, which renders as a page on your site at `YOUR-SITE-URL/about`. You can edit the contents of that file to personalize your "About" page, and you can use the "About" page as a template to create new pages. For more information, see "Pages" in the Jekyll documentation.

A post is a blog post. The default Jekyll site contains a directory named `_posts` that contains a default post file. You can edit the contents of that post, and you can use the default post as a template to create new posts. For more information, see "Posts" in the Jekyll documentation.

Your theme includes default layouts, includes, and stylesheets that will automatically be applied to new pages and posts on your site, but you can override any of these defaults. For more information, see "About GitHub Pages and Jekyll."

To set variables and metadata, such as a title and layout, for a page or post on your site, you can add YAML front matter to the top of any Markdown or HTML file. For more information, see "Front Matter" in the Jekyll documentation.

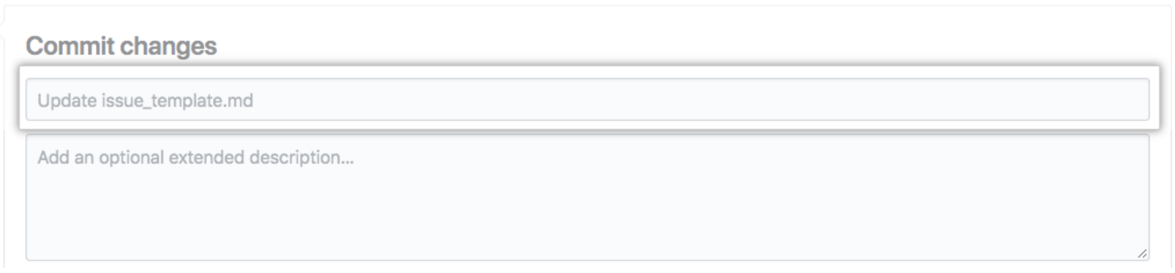
Changes to your site are published automatically when the changes are merged into your site's publishing source. If you want to preview your changes first, you can make the changes locally instead of on GitHub. Then, test your site locally. For more information, see "Testing your GitHub Pages site locally with Jekyll."

Adding a new page to your site

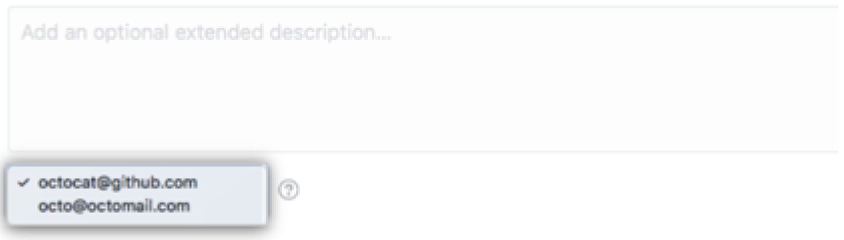
1. On GitHub, navigate to your site's repository.
2. Navigate to the publishing source for your site. For more information about publishing sources, see "About GitHub Pages."
3. In the root of your publishing source, create a new file for your page called *PAGE-NAME.md*, replacing *PAGE-NAME* with a meaningful filename for the page.
4. Add the following YAML frontmatter to the top of the file, replacing *PAGE TITLE* with the page's title and *URL-PATH* with a path you want for the page's URL. For example, if the base URL of your site is `https://octocat.github.io` and your *URL-PATH* is `/about/contact/`, your page will be located at `https://octocat.github.io/about/contact`.

```
layout: page
title: "PAGE TITLE"
permalink: /URL-PATH/
```

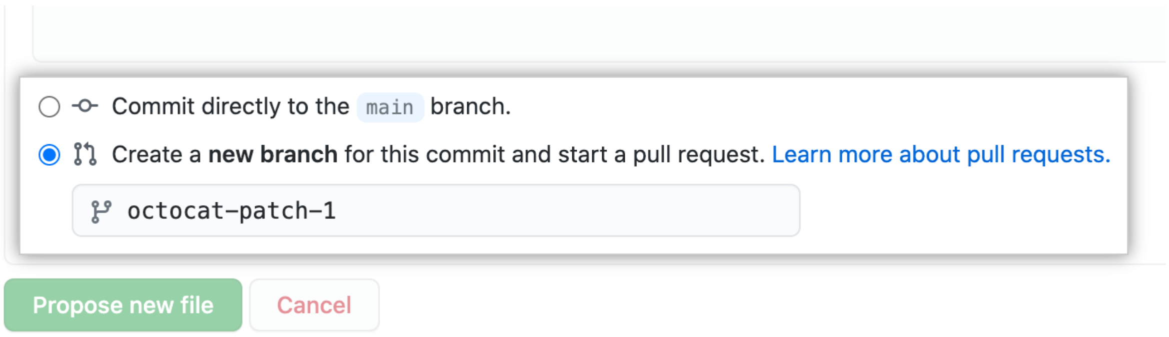
5. Below the frontmatter, add content for your page.
6. At the bottom of the page, type a short, meaningful commit message that describes the change you made to the file. You can attribute the commit to more than one author in the commit message. For more information, see "Creating a commit with multiple co-authors."



7. If you have more than one email address associated with your GitHub account, click the email address drop-down menu and select the email address to use as the Git author email address. Only verified email addresses appear in this drop-down menu. If you enabled email address privacy, then `<username>@users.noreply.github.com` is the default commit author email address. For more information, see "Setting your commit email address."

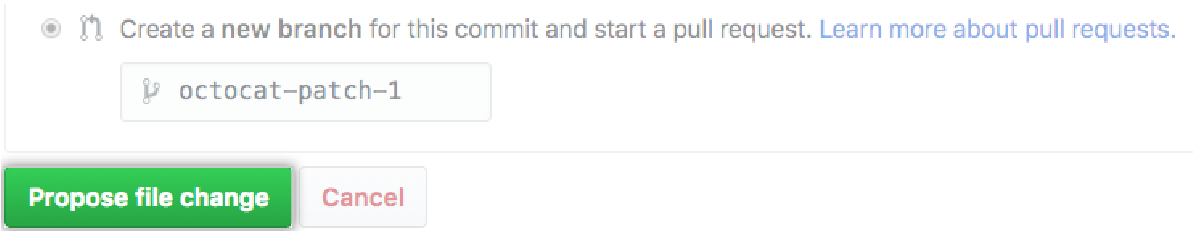


8. Below the commit message fields, decide whether to add your commit to the current branch or to a new branch. If your current branch is the default branch, you should choose to create a new branch for your commit and then create a pull request. For more information, see "Creating a new pull request."



Commit directly to the main branch.
Create a new branch for this commit and start a pull request. Learn more about pull requests.
octocat-patch-1
Propose new file
Cancel

9. Click **Propose file change**.



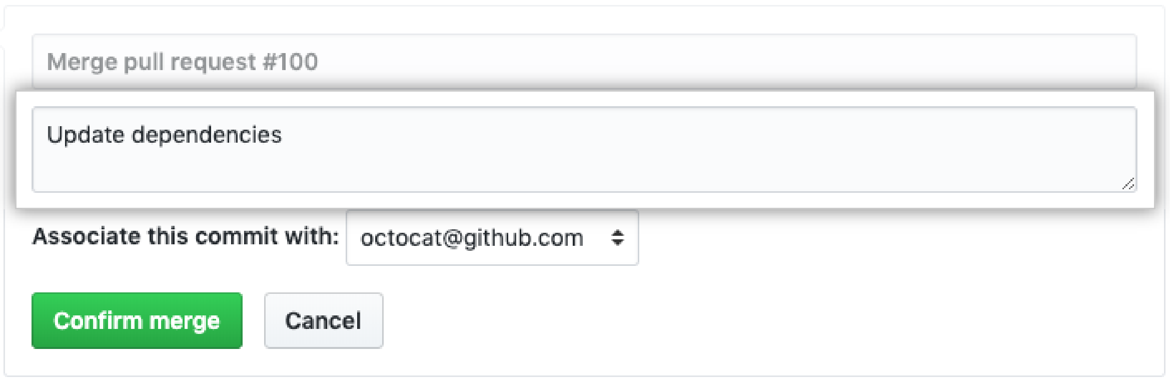
Create a new branch for this commit and start a pull request. Learn more about pull requests.
octocat-patch-1
Propose file change
Cancel

10. Create a pull request for your proposed changes.

11. In the "Pull Requests" list, click the pull request you would like to merge.

12. Click **Merge pull request**. For more information, see "Merging a pull request."

13. If prompted, type a commit message, or accept the default message.



Merge pull request #100

Update dependencies

Associate this commit with: octocat@github.com

14. Click **Confirm merge**.

15. Optionally, delete the branch. For more information, see "Creating and deleting branches within your repository."

Adding a new post to your site

1. On GitHub, navigate to your site's repository.

2. Navigate to the publishing source for your site. For more information about publishing sources, see "About GitHub Pages."

3. Navigate to the `_posts` directory.

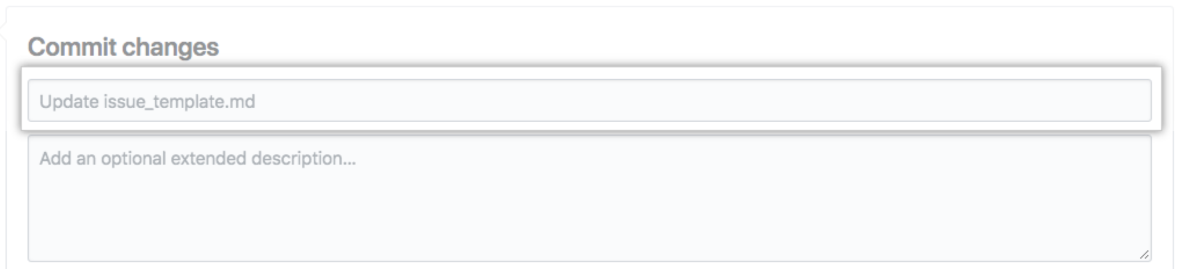
4. Create a new file called *YYYY-MM-DD-NAME-OF-POST.md*, replacing *YYYY-MM-DD* with the date of your post and *NAME-OF-POST* with the name of your post.

5. Add the following YAML frontmatter to the top of the file, replacing *POST TITLE* with the post's title, *YYYY-MM-DD hh:mm:ss -0000* with the date and time for the post, and *CATEGORY-1* and *CATEGORY-2* with as many categories you want for your post.

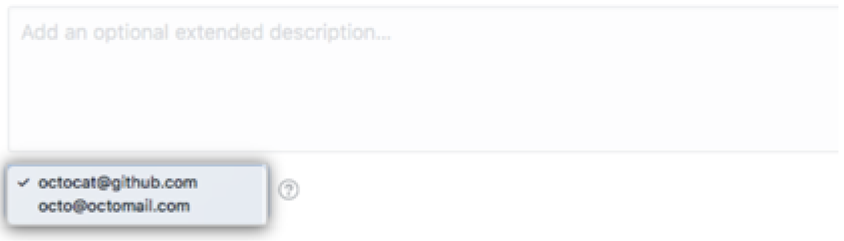
```
layout: post
title: "POST TITLE"
date: YYYY-MM-DD hh:mm:ss -0000
categories: CATEGORY-1 CATEGORY-2
```

6. Below the frontmatter, add content for your post.

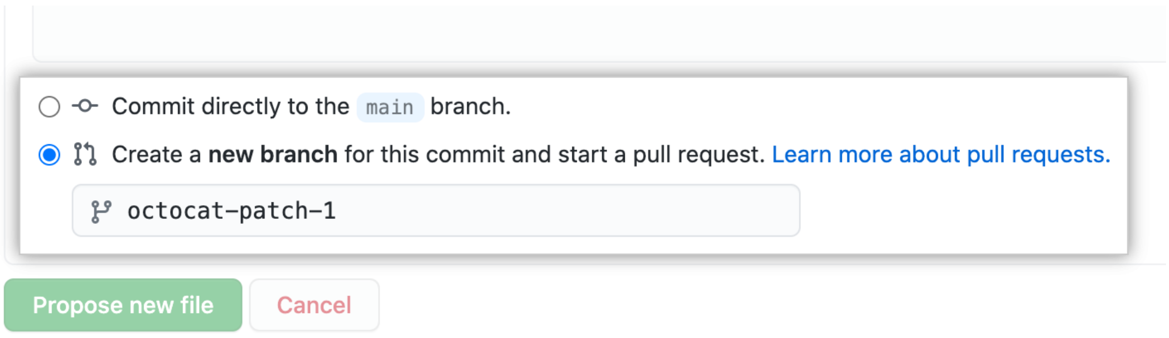
7. At the bottom of the page, type a short, meaningful commit message that describes the change you made to the file. You can attribute the commit to more than one author in the commit message. For more information, see "Creating a commit with multiple co-authors."



8. If you have more than one email address associated with your GitHub account, click the email address drop-down menu and select the email address to use as the Git author email address. Only verified email addresses appear in this drop-down menu. If you enabled email address privacy, then `<username>@users.noreply.github.com` is the default commit author email address. For more information, see "Setting your commit email address."



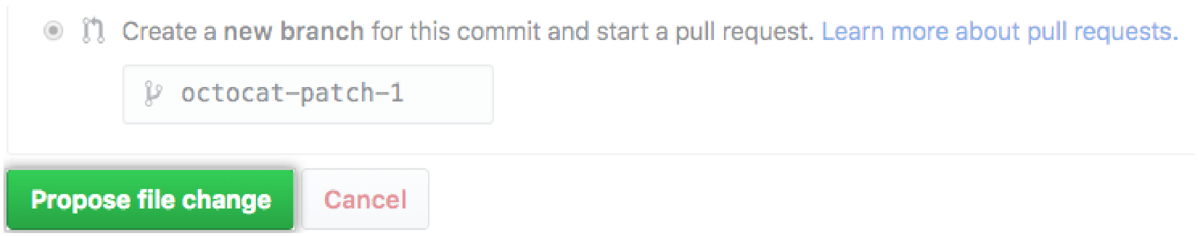
9. Below the commit message fields, decide whether to add your commit to the current branch or to a new branch. If your current branch is the default branch, you should choose to create a new branch for your commit and then create a pull request. For more information, see "Creating a new pull request."


Commit directly to the `main` branch.

Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

octocat-patch-1

10. Click **Propose file change**.


Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

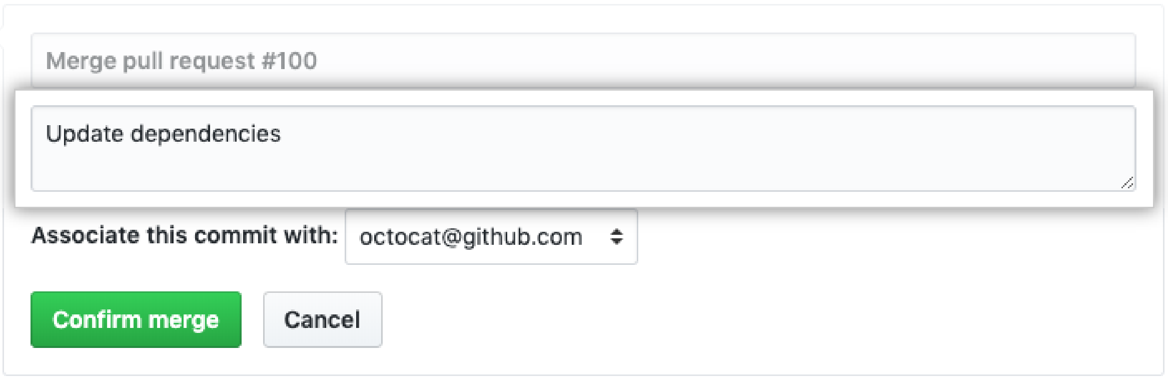
octocat-patch-1

11. Create a pull request for your proposed changes.

12. In the "Pull Requests" list, click the pull request you would like to merge.

13. Click **Merge pull request**. For more information, see "Merging a pull request."

14. If prompted, type a commit message, or accept the default message.


Merge pull request #100

Update dependencies

Associate this commit with: octocat@github.com

15. Click **Confirm merge**.

16. Optionally, delete the branch. For more information, see "Creating and deleting branches within your repository."

Your post should now be up on your site! If the base URL of your site is `https://octocat.github.io`, then your new post will be located at `https://octocat.github.io/YYYY/MM/DD/TITLE.html`.

Next steps

You can add a Jekyll theme to your GitHub Pages site to customize the look and feel of your site. For more information, see "Adding a theme to your GitHub Pages site using Jekyll."

Exercise - Enable, create, and update a GitHub Pages web site

docs.microsoft.com/en-us/learn/modules/create-host-web-sites-github-pages/3-introduction-to-github-pages

In this exercise you use GitHub Learning Lab to learn about setting up and using GitHub Pages.

GitHub Learning Lab is an integrated experience that's easy to use. You get feedback and instructions throughout the lab as you work in your GitHub repository.

Here are a few suggestions to make the Learning Lab exercise more enjoyable.

- GitHub Learning Lab is installed on your account in the first step of this lab. If you're asked, be sure to *install it on all repositories*. This won't affect the organizations that you're a member of, just the personal repositories that the lab creates for you.
- After the install, you may be returned to the main page. To get back to your lab, just use the button on the bottom of this page.
- GitHub will create a repository for you to use. Give permissions to GitHub Learning Lab.
- GitHub Learning Lab will set itself as a reviewer on your pull requests so that it can give you the next steps just in time. Sometimes reviewing your pull request will take a few minutes.
- When you're given a link for creating or editing a file or told to open a tab, **be sure to open it in another tab in your browser**. This way you can come back to the instructions without leaving the file.
- Comments and instructions will continue on your pull request or in an issue on your repository.

When you've finished the exercise in GitHub, return here for:

- A quick knowledge check
- A summary of what you've learned
- To earn a badge for completing this module

Start the learning lab on GitHub

Next unit: Knowledge check

Continue

Need help? See our troubleshooting guide or provide specific feedback by reporting an issue.

✓ 200 XP

Knowledge check

3 minutes

Choose the best response for each question. Then select "Check your answers."

Check your knowledge

1. How do you enable GitHub Pages for your repository?

- ○ Email GitHub support and ask for your free web site installation kit.
- ○ Subscribe to GitHub Enterprise and wait for further instructions in email.
- [✓ Select a **Source** from the **GitHub Pages** section of the **Settings** tab of your repository.] ✓

  Correct! After a moment or two, your site will be provisioned and ready to go.

2. Which of the following **cannot** be hosted on GitHub Pages?

- ○ Personal sites, like your resume or portfolio.
- [✓ Dynamic server-side applications.] ✓

  GitHub Pages only hosts static content. However, you can still use JavaScript to add dynamic behavior and even integrate with external services from your site. For server-side apps, step up to Azure Web Apps.
- ○ Professional sites, like product pages and marketing campaigns.

3. Which of the following would you most likely use to edit the body content of your GitHub Pages site?

- [✓ Markdown] ✓

  Markdown is the preferred syntax for creating and editing the core content of your site. You can also use HTML and the Liquid templating language.

○ Jekyll

○ YAML

Next unit: Summary

Continue >