Using Markdown and Liquid in GitHub Docs

In this article

You can use Markdown and Liquid to format content, create reusable content, and write content for different versions on GitHub Docs.

Articles in the "Contributing to GitHub Docs" section refer to the documentation itself and are a resource for GitHub staff and open source contributors.

About using Markdown and Liquid in GitHub Docs

GitHub Docs are written using Markdown, which is a human-friendly syntax for formatting plain text. We use the variant of Markdown called GitHub Flavored Markdown and ensure that it is compliant with CommonMark. For more information, see "About writing and formatting on GitHub."

We use Liquid syntax to expand the functionality to provide accessible tables, maintainable links, versioning, variables, and chunks of reusable content. For more information about Liquid, see the Liquid documentation.

The content on this site uses Markdown rendering powered by `/src/content-render`, which is in turn built on the `remark` Markdown processor.

Lists

In a list item, the general rules for additional content after the first paragraph are:

- Images and subsequent paragraphs should each be on their own line and separated by a blank line.

- All subsequent lines in a list item must match up with the first text after the list marker.

Example usage of a list

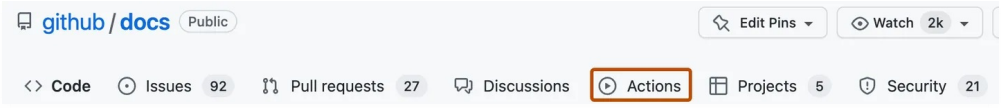
This example shows the correct way to align list items with multiple paragraphs or objects.

```
1. Under your repository name, click **Actions**.

   ![Screenshot of the tabs for the "github/docs" repository. The "Actions" tab is highlighted with an orange outline.](/assets/images/help/repository/actions-tab.png)

   This is another paragraph in the list.

1. This is the next item.
```

This content is displayed on the GitHub Docs site with the content under the first list item correctly aligned.

Example list rendered on GitHub Docs

1. Under your repository name, click **Actions**.

   This is another paragraph in the list.

2. This is the next item.

Callout tags

Callouts highlight important information that users need to know. We use standard formatting and colors for different types of callouts: notes, warnings, and danger notices. Use Liquid tags before and after the text you'd like included in the callout box.

For information on when to use callout tags, see "Style guide."

Example usage of a callout

```
{% note %}

**Note:** Owners and administrators can add outside collaborators to a repository.

{% endnote %}
```

Example callout rendered on GitHub Docs

Note: Owners and administrators can add outside collaborators to a repository.

Code sample syntax highlighting

To render syntax highlighting in command line instructions and code samples, we use triple backticks followed by the language of the sample. For a list of all supported languages, see `code-languages.yml`.

Example usage of code syntax highlighting

````
```bash
git init YOUR_REPOSITORY
```
````

Within the code sample syntax, use all uppercase text to indicate placeholder text or content that varies for each user, such as a user or repository name. By default, codeblocks will escape the content within the triple backticks. If you need to write sample code that parses the content (for example, to italicize text within `<em>` tags instead of passing the tags through literally), wrap the codeblock in `<pre>` tags.

Code blocks with a copy button

You can also add a header that includes the name of the language and a button to copy the contents of the code block.

For example, the following code adds syntax highlighting for JavaScript and a copy button for the code sample.

Example usage of a copy button

````
```javascript copy
const copyMe = true
```
````

Example code rendered on GitHub Docs

JavaScript

```javascript
const copyMe = true
```

Code sample annotations

Code sample annotations help explain longer code examples by rendering comments as annotations next to the sample code. This lets us write longer explanations of code without cluttering the code itself. Code samples with annotations are rendered in a two pane layout with the code sample on the left and the annotations on the right. The annotations are visually emphasized when someone hovers their cursor over the code example.

Code annotations only work in articles with the `layout: inline` frontmatter property. For more information on how to write and style code annotations, see "Annotating code examples."

Example of an annotated code sample

```yaml annotate
# The name of the workflow as it will appear in the "Actions" tab of the GitHub
repository.
name: Post welcome comment
# The `on` keyword lets you define the events that trigger when the workflow is
run.
on:
  # Add the `pull_request` event, so that the workflow runs automatically
  # every time a pull request is created.
  pull_request:
    types: [opened]
# Modifies the default permissions granted to `GITHUB_TOKEN`.
permissions:
  pull-requests: write
# Defines a job with the ID `build` that is stored within the `jobs` key.
jobs:
  build:
    name: Post welcome comment
    # Configures the operating system the job runs on.
    runs-on: ubuntu-latest
    # The `run` keyword tells the job to execute the [`gh pr comment`]
(https://cli.github.com/manual/gh_pr_comment) command on the runner.
    steps:
      - run: gh pr comment $PR_URL --body "Welcome to the repository!"
        env:
          GITHUB_TOKEN: $
          PR_URL: $
```

For an example of an article that uses code annotations on GitHub Docs, see "Using scripts to test your code on a runner."

Octicons

Octicons are icons used across GitHub's interface. We reference Octicons when documenting the user interface and to indicate binary values in tables. Find the name of specific Octicons on the Octicons site.

If you're referencing an Octicon that appears in the UI, identify whether the Octicon is the entire label of the UI element (for example, a button that is labeled only with "+") or whether it's only decorative, in addition to another label (for example, a button is labeled "+ Add message").

- If the Octicon is the entire label, use your browser's developer tools to inspect the Octicon and determine what screen reader users will hear instead. Then, use that text for the `aria-label` (for example, `<svg version="1.1" width="16" height="16" viewBox="0 0 16 16" class="octicon octicon-plus" aria-label="Add file" role="img"><path d="M7.75 2a.75.75 0 0 1 .75.75V7h4.25a.75.75 0 0 1 0 1.5H8.5v4.25a.75.75 0 0 1-1.5 0V8.5H2.75a.75.75 0 0 1 0-1.5H7V2.75A.75.75 0 0 1 7.75 2Z"></path></svg>`). Occasionally, in the UI, the Octicon itself will not have an `aria-label`, but a surrounding element such as a `<summary>` or `<div>` tag will.
  - Some Octicons used as labels have dynamic `aria-label` elements that change based on the state of the UI element or a user input. For example, when someone has two security policies-`Policy A` and `Policy B`-their UI will show two trash Octicons labelled `<svg version="1.1" width="16" height="16" viewBox="0 0 16 16" class="octicon octicon-trash" aria-label="Delete Policy A" role="img"><path d="M11 1.75V3h2.25a.75.75 0 0 1 0 1.5H2.75a.75.75 0 0 1 0-1.5H5V1.75C5 .784 5.784 0 6.75 0h2.5C10.216 0 11 .784 11 1.75ZM4.496 6.675l.66 6.6a.25.25 0 0 0 .249.225h5.19a.25.25 0 0 0 .249-.225l.66-6.6a.75.75 0 0 1 1.492.149l-.66 6.6A1.748 1.748 0 0 1 10.595 15h-5.19a1.75 1.75 0 0 1-1.741-1.575l-.66-6.6a.75.75 0 1 1 1.492-.15ZM6.5 1.75V3h3V1.75a.25.25 0 0 0-.25-.25h-2.5a.25.25 0 0 0-.25.25Z"></path></svg>` and `<svg version="1.1"`

`width="16" height="16" viewBox="0 0 16 16" class="octicon octicon-trash" aria-label="Delete Policy B" role="img"><path d="M11 1.75V3h2.25a.75.75 0 0 1 0 1.5H2.75a.75.75 0 0 1 0-1.5H5V1.75C5 .784 5.784 0 6.75 0h2.5C10.216 0 11 .784 11 1.75ZM4.496 6.675l.66 6.6a.25.25 0 0 0 .249.225h5.19a.25.25 0 0 0 .249-.225l.66-6.6a.75.75 0 0 1 1.492.149l-.66 6.6A1.748 1.748 0 0 1 10.595 15h-5.19a1.75 1.75 0 0 1-1.741-1.575l-.66-6.6a.75.75 0 1 1 1.492-.15ZM6.5 1.75V3h3V1.75a.25.25 0 0 0-.25-.25h-2.5a.25.25 0 0 0-.25.25Z"></path></svg>`. For dynamic `aria-label` elements, since we can't document the exact `aria-label` that people will encounter, describe the Octicon and a placeholder example of the label (for example, `"<svg version="1.1" width="16" height="16" viewBox="0 0 16 16" class="octicon octicon-trash" aria-label="The trash icon" role="img"><path d="M11 1.75V3h2.25a.75.75 0 0 1 0 1.5H2.75a.75.75 0 0 1 0-1.5H5V1.75C5 .784 5.784 0 6.75 0h2.5C10.216 0 11 .784 11 1.75ZM4.496 6.675l.66 6.6a.25.25 0 0 0 .249.225h5.19a.25.25 0 0 0 .249-.225l.66-6.6a.75.75 0 0 1 1.492.149l-.66 6.6A1.748 1.748 0 0 1 10.595 15h-5.19a1.75 1.75 0 0 1-1.741-1.575l-.66-6.6a.75.75 0 1 1 1.492-.15ZM6.5 1.75V3h3V1.75a.25.25 0 0 0-.25-.25h-2.5a.25.25 0 0 0-.25.25Z"></path></svg>"`). This will help people identify both the Octicon and how it is labelled, and give context for collaborating with people who are visually describing the Octicon.

- If the Octicon is decorative, it's likely hidden to screen readers with the `aria-hidden=true` attribute. If so, for consistency with the product, use `aria-hidden="true"` in the Liquid syntax for the Octicon in the docs as well (for example, `"<svg version="1.1" width="16" height="16" viewBox="0 0 16 16" class="octicon octicon-plus" aria-hidden="true"><path d="M7.75 2a.75.75 0 0 1 .75.75V7h4.25a.75.75 0 0 1 0 1.5H8.5v4.25a.75.75 0 0 1-1.5 0V8.5H2.75a.75.75 0 0 1 0-1.5H7V2.75A.75.75 0 0 1 7.75 2Z"></path></svg> Add message"`).

If you're using the Octicon in another way, such as using the "check" and "x" icons to reflect binary values in tables, use the `aria-label` to describe the meaning of the Octicon, not its visual characteristics. For example, if you're using a "x" icon in the "Supported" column of a table, use "Not supported" as the `aria-label`. For more information, see "Style guide."

Example usage of Octicons

```
{% octicon "<name of Octicon>" %}
{% octicon "plus" %}
{% octicon "plus" aria-label="Add file" %}
"{% octicon "plus" aria-hidden="true" %} Add file"
```

Operating system tags

We occasionally need to write documentation for different operating systems. Each operating system may require a different set of instructions. We use operating system tags to demarcate information for each operating system.

Example usage of operating system tags

```
{% mac %}

These instructions are pertinent to Mac users.

{% endmac %}
```

```
{% linux %}

 These instructions are pertinent to Linux users.

{% endlinux %}
```

```
{% windows %}

These instructions are pertinent to Windows users.

{% endwindows %}
```

You can define a default platform in an article's YAML frontmatter. For more information, see "Using YAML frontmatter."

Tool tags

We occasionally need to write documentation that has different instructions for different tools. For example, the GitHub UI, GitHub CLI, GitHub Desktop, GitHub Codespaces, and Visual Studio Code might be able to accomplish the same task using different steps. We use tool tags to control what information is displayed for each tool.

GitHub Docs maintains tool tags for GitHub products and selected third-party extensions. See the `all-tools.js` object in the `github/docs` repository for a list of all supported tools.

On rare occasions, we will add new tools. Before adding a new tool, read "Creating tool switchers in articles." To add a new tool, add an entry to the `allTools` object in `lib/all-tools.js` as a key-value pair. The key is the tag you'll use to refer to the tool in the article, and the value is how the tool will be identified on the tool picker at the top of the article.

You can define a default tool for an article in the YAML frontmatter. For more information, see "Using YAML frontmatter."

Example usage of tool tags

```
{% api %}

These instructions are pertinent to API users.

{% endapi %}
```

```
{% bash %}

These instructions are pertinent to Bash shell commands.

{% endbash %}
```

```
{% cli %}

These instructions are pertinent to GitHub CLI users.

{% endcli %}
```

```
{% codespaces %}
```

```
These instructions are pertinent to Codespaces users. They are mostly used
outside the Codespaces docset, when we want to refer to how to do something
inside Codespaces. Otherwise `webui` or `vscode` may be used.

{% endcodespaces %}
```

```
{% curl %}

These instructions are pertinent to curl commands.

{% endcurl %}
```

```
{% desktop %}

 These instructions are pertinent to GitHub Desktop.

{% enddesktop %}
```

```
{% importer_cli %}

These instructions are pertinent to GitHub Enterprise Importer CLI users.

{% endimporter_cli %}
```

```
{% javascript %}

These instructions are pertinent to javascript users.

{% endjavascript %}
```

```
{% jetbrains %}

These instructions are pertinent to users of JetBrains IDEs.

{% endjetbrains %}
```

```
{% powershell %}

These instructions are pertinent to `pwsh` and `powershell` commands.

{% endpowershell %}
```

```
{% vscode %}

These instructions are pertinent to VS Code users.

{% endvscode %}
```

```
{% webui %}

These instructions are pertinent to GitHub UI users.

{% endwebui %}
```

Reusable and variable strings of text

Reusable strings (commonly called content references or conrefs) contain content that is used in more than one place in our documentation. Creating these allows us to update the content in a single location rather than every place the string appears.

For longer strings, we use reusables, and for shorter strings, we use variables. For more information about reusables and variables, see "Creating reusable content."

Table pipes

Every row of a table in the GitHub Docs must start and end with a pipe, `|`.

```
| Where is the table located? | Does every row end with a pipe? |
| --- | --- |
| GitHub Docs | Yes |
```

Table row headers

If you create a table where the first column contains headers for the table rows, wrap your table in the Liquid tag `{% rowheaders %} {% endrowheaders %}`. For more information on using markup for tables, see "Style guide."

Example table with row headers

```
{% rowheaders %}

|              | Mona | Tom    | Hobbes |
|--------------|------|--------|--------|
|Type of cat   | Octo | Tuxedo | Tiger  |
|Likes to swim| Yes  | No     | No     |

{% endrowheaders %}
```

Example table without row headers

```
| Name   | Vocation         |
| ------ | ---------------- |
| Mona   | GitHub mascot    |
| Tom    | Mouse antagonist |
| Hobbes | Best friend      |
```

Tables with codeblocks

Although using tables to contain block items, such as code blocks, is generally discouraged, occasionally it may be appropriate.

Because tables in GitHub Flavored Markdown cannot contain any line breaks or block-level structures, you must use HTML tags to write the table structure.

When HTML tables contain code blocks, the width of the table might exceed the regular width of page content, and then overflow into the area normally containing the mini table of contents.

If this happens, add the following CSS style to the `<table>` HTML tag:

```
<table style="table-layout: fixed;">
```

For a current example of this usage, see "Examples."

Links

Links to docs in the `docs` repository must start with a product ID (like `/actions` or `/admin`) and contain the entire filepath, but not the file extension. For example, `/actions/creating-actions/about-custom-actions`.

Image paths must start with `/assets` and contain the entire filepath including the file extension. For example, `/assets/images/help/settings/settings-account-delete.png`.

The links to Markdown pages undergo some transformations on the server side to match the current page's language and version. The handling for these transformations lives in `lib/render-content/plugins/rewrite-local-links`.

For example, if you include the following link in a content file:

```
/github/writing-on-github/creating-a-saved-reply
```

When viewed on GitHub.com docs, the link gets rendered with the language code:

```
/en/github/writing-on-github/creating-a-saved-reply
```

and when viewed on GitHub Enterprise Server docs, the version is included as well:

```
/en/enterprise-server@2.20/github/writing-on-github/creating-a-saved-reply
```

For more information about links, see "Style guide."

Permalinks

Because the site is dynamic, it does not build HTML files for each different version of an article. Instead it generates a "permalink" for every version of the article. It does this based on the article's `versions` frontmatter.

> **Note**: As of early 2021, the `free-pro-team@latest` version is not included in URLs. A helper function called `lib/remove-fpt-from-path.js` removes the version from URLs.

For example, an article that is available in currently supported versions will have permalink URLs like the following:

- `/en/get-started/quickstart/set-up-git`
- `/en/enterprise-cloud@latest/get-started/quickstart/set-up-git`
- `/en/enterprise-server@3.10/get-started/quickstart/set-up-git`
- `/en/enterprise-server@3.9/get-started/quickstart/set-up-git`
- `/en/enterprise-server@3.8/get-started/quickstart/set-up-git`
- `/en/enterprise-server@3.7/get-started/quickstart/set-up-git`
- `/en/enterprise-server@3.6/get-started/quickstart/set-up-git`

An article that is not available in GitHub Enterprise Server will have just one permalink:

- `/en/get-started/quickstart/set-up-git`

> **Note:** If you are a content contributor, you don't need to worry about supported versions when adding a link to a document. Following the examples above, if you want to reference an article, you can just use its relative location: `/github/getting-started-with-github/set-up-git`.

Internal links with AUTOTITLE

When linking to another GitHub Docs page, use standard Markdown syntax like `[]()`, but type `AUTOTITLE` instead of the page title. The GitHub Docs application will replace `AUTOTITLE` with the title of the linked page during rendering. This special keyword is case-sensitive, so take care with your typing or the replacement will not work.

Example usage of internal links with AUTOTITLE

- `For more information, see "[AUTOTITLE](/path/to/page)."`
- `For more information, see "[AUTOTITLE](/path/to/page#section-link)."`
- `For more information, see the TOOLNAME documentation in "[AUTOTITLE](/path/to/page?tool=TOOLNAME)."`

> **Note:** Same-page section links do not work with this keyword. Type out the full header text instead.

Linking to the current article in a different version of the docs

Sometimes you may want to link from an article to the same article in a different product version. For example:

- You mention some functionality that is not available for free, pro, or team plans and you want to link to the GitHub Enterprise Cloud version of the same page.
- The GitHub Enterprise Server version of an article describes a feature that shipped with that version, but site administrators can upgrade to the latest version of the feature that's in use on GitHub Enterprise Cloud.

You can link directly to a different version of the page using the `currentArticle` property. This means that the link will continue to work directly even if the article URL changes.

```
{% ifversion fpt %}For more information, see the [{% data
variables.product.prodname_ghe_cloud %} documentation](/enterprise-cloud@latest{{
currentArticle }}).{% endif %}
```

Preventing transformations

Sometimes you want to link to a Dotcom-only article in Enterprise content and you don't want the link to be Enterprise-ified. To prevent the transformation, you should include the preferred version in the path.

```
"[GitHub's Terms of Service](/free-pro-team@latest/github/site-policy/github-
terms-of-service)"
```

Sometimes the canonical home of content moves outside the docs site. None of the links included in `src/redirects/lib/external-sites.json` get rewritten. See `contributing/redirects.md` for more info about this type of redirect.

Legacy filepaths and redirects for links

Our docs contain links that use legacy filepaths such as `/article/article-name` or `/github/article-name`. Our docs also contain links that refer to articles by past names. Both of these link types function properly because of redirects, but they are bugs.

When you add a link to an article, use the current filepath and article name.