Versioning documentation

In this article

GitHub Docs uses YAML frontmatter and liquid operators to support multiple versions of GitHub with a single-source approach.

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

On GitHub Docs, we provide versions of our documentation that reflect the differences in UI and functionality across GitHub's major product offerings. Contributors can use versioning syntax to scope content to a specific product offering.

Versioning syntax allows the reader to manually choose the version of the documentation that applies to the product they're using. GitHub Docs' URLs can also include versioning information, which allows links from GitHub.com, GitHub Enterprise Server, and GitHub AE to send the reader directly to documentation for the product they're using.

How and where to version

Versioning for content on GitHub Docs is single-source to avoid repetition and keep prose DRY. For articles, you apply versioning to an individual Markdown file with YAML metadata, then use conditional statements within the file's prose to instruct the site which text to display depending on the version the reader selects. Single-sourcing contrasts to the creation of separate files that reflect each version of the content.

There are two types of versioning syntax for GitHub Docs.

- YAML: Used most often in YAML front matter within Markdown files in `content/`, but also in many types of YAML files in `data/`. Indicates the versioning for an entire piece of content.

  ```
  versions:
    PRODUCT: 'VERSIONS'
    PRODUCT: 'VERSIONS'
    ...
  ```

  The following example shows content versioned for GitHub.com, and all versions of

GitHub Enterprise Server.

```
versions:
  fpt: *
  ghes: *
```

- Liquid: Used within Markdown files in `content/` and `data/reusables/`, variable strings within YAML files in `data/variables/`, or strings within `data/glossaries/external.yml`. Indicates which text should appear when a reader chooses a version for content that has multiple versions defined by YAML front matter.
  - Product-based versioning:

    ```
    {% ifversion SHORT-PRODUCT-NAME %} ... {% endif %}
    ```

  - Feature-based versioning:

    ```
    {% ifversion FEATURE-NAME %} ... {% endif %}
    ```

About the different versions of GitHub

We provide versioned documentation for users of GitHub.com plans including GitHub Enterprise Cloud, GitHub Enterprise Server, and GitHub AE. If multiple versions of a page exist on the site, readers can choose the version from the version picker at the top of the page.

GitHub.com

Documentation for GitHub.com has two possible versions:

Free, Pro, or Team plans

For Free, Pro, or Team plans on GitHub.com, use `free-pro-team@latest`. The short name is `fpt`.

GitHub Enterprise Cloud

For GitHub Enterprise Cloud, use `enterprise-cloud@latest`. The short name is `ghec`.

GitHub Enterprise Server

Documentation for GitHub Enterprise Server has multiple versions and can be divided into two types: documentation for *supported releases* (we support four at any one time), and documentation for *deprecated releases* (we do not link to these on the Docs site but we support a "frozen" snapshot of these docs in perpetuity, so they can still be accessed if you know the URLs). See `lib/enterprise-server-releases.js` for a list.

The versions are named `enterprise-server@<release>`. The short name is `ghes`. In Liquid conditionals, we can specify ranges, like `ghes > 3.0`. For more information, see "Versioning with Liquid conditional operators."

GitHub AE

Versioning for GitHub AE uses the `github-ae@latest` version. The short name is `ghae`.

Versioning in the YAML frontmatter

You can use the `versions` property within the file's frontmatter to define which products an entire page applies to. For example, the following YAML frontmatter will version an article for GitHub Enterprise Server 2.20 and above and Free, Pro, or Team.

```
title: About your personal dashboard
versions:
  fpt: '*'
  ghes: '>=2.20'
```

The following example will version an article for all supported versions of GitHub Enterprise Server:

```
title: Downloading your license
versions:
  ghes: '*'
```

You can also version a page for a range of releases. The following example will version the page for GitHub.com, and GitHub Enterprise Server versions 2.22 and 3.0 only:

```
versions:
  fpt: '*'
  ghec: '*'
  ghes: '>=2.22 <3.1'
```

Versioning with Liquid conditional operators

We use the Liquid template language (specifically, this Node.js port) and a custom `{% ifversion ... %}` tag to create versions of our documentation.

If you define multiple products in the `versions` key within a page's YAML frontmatter, you can use the conditional operators `ifversion` / `else` (or `ifversion` / `elsif` / `else`) in the Markdown to control how the site renders content on the page for a particular product. For example, a feature may have more options on GitHub.com than on GitHub Enterprise Server, so you can version the content appropriately via the `versions` frontmatter, and use Liquid conditionals to describe the additional options for GitHub.com.

Notes:

- Use `ifversion` for product-based versioning and feature-based versioning.
- Do not use `if` or `unless`.
- Make sure to use `elsif` and not `else if`. Liquid does not recognize `else if` and will not render content inside an `else if` block.

Comparison operators

For versions that don't have numbered releases (like `fpt` and `ghae`), you have two options:

- `{% ifversion ghae %}`
- `{% ifversion not ghae %}`

For versions that have numbered releases (currently only `ghes`), you can do the same for content that is either available in all of the releases or not available in any of the releases:

- `{% ifversion ghes %}`
- `{% ifversion not ghes %}`

If you need to denote content that is only available (or not available) in certain releases, you can use the following operators:

Operator	Meaning	Example
`=`	Equal to	`{% ifversion ghes = 3.0 %}`
`>`	Newer than	`{% ifversion ghes > 3.0 %}`
`<`	Older than	`{% ifversion ghes < 3.0 %}`
`!=`	Not equal to	`{% ifversion ghes != 3.0 %}` (don't use `not` in ranges)

The Liquid operators `==`, `>=`, and `<=` are not supported in the GitHub Docs.

Logical operators

When all operands must be true for the condition to be true, use the operator `and`:

```
{% ifversion ghes > 2.21 and ghes < 3.1 %}
```

When at least one operand must be true for the condition to be true, use the operator `or`:

```
{% ifversion fpt or ghes > 2.21 %}
```

Do not use the operators `&&` or `||`. Liquid does not recognize them, and the content will not render in the intended versions.

Whitespace control

When using Liquid conditionals in lists or tables, you can use whitespace control characters to prevent the addition of newlines and other whitespace that would break the list or table rendering.

You can add a hyphen (`-`) on either the left, right, or both sides to indicate that there should be no newline or other whitespace on that side.

```
{%- ifversion fpt %}
```

For example, to version a table row, instead of adding liquid versioning for the row starting at the end of the previous row, like this:

```
Column A | Column B | Column C
---------|----------|---------
This row is for all versions | B1 | C1{% ifversion ghes %}
This row is for GHES only | B2 | C2{% endif %}
This row is for all versions | B3 | C3
```

You can include the liquid versioning on its own line and use whitespace control to strip the newline to the left of the liquid tag. This makes reading the source much easier, without breaking the rendering of the table:

```
Column A | Column B | Column C
---------|----------|---------
This row is for all versions | B1 | C1
{%- ifversion ghes %}
This row is for GHES only | B2 | C2
{%- endif %}
This row is for all versions | B3 | C3
```

About feature-based versioning

When you document any new change or feature, use feature-based versioning.

A small minority of features and changes will only ever apply to one product. The majority of features come to GitHub.com and eventually reach all products. In general, changes "flow" from GitHub.com (including GitHub Enterprise Cloud) to GitHub Enterprise Server, and then to GitHub AE.

Feature-based versioning provides named "feature flags" that simplify the maintenance and versioning of documentation. You can use a single feature name (or "flag") to group and version prose throughout content. When a feature comes to additional products, you only need to make a change to the YAML versioning in the file within `data/features/`.

Managing features

Each feature is managed through individual YAML files in `data/features/`.

Note: Do not delete `data/features/placeholder.yml` because it is used by tests.

To create a new feature, first create a new YAML file with the feature name you want to use in this directory. For a feature named `meow`, that would be `data/features/meow.yml`.

Add a `versions` block to the YAML file with the short names of the versions the feature is available in. For example:

```
versions:
  fpt: '*'
  ghec: '*'
  ghes: '>3.1'
  ghae: '*'
```

The format and allowed values are the same as the frontmatter versions property. For more information, see "Versions" in the `github/docs` repository README.

Liquid conditionals

Now you can use `{% ifversion meow %} ... {% endif %}` in content files!

Frontmatter

You can also use the feature in frontmatter in content files:

```
versions:
  feature: 'meow'
```

You can only use one `feature` entry under `versions`, and the value of `feature` can only contain one feature name.

You can combine feature-based versioning and standard versioning in frontmatter. When you do this the article will be included in the superset of all of the versions specified in the feature-based versioning file and directly in the Markdown file. For example, you might have a feature that is currently only available in GHEC, and this is specified in the feature-based versioning. However, you want the "About" article for this feature to also be visible in the FPT docs. In this case you could add `fpt` and `feature` to the `versions` block in the front matter:

```
versions:
  fpt: '*'
  feature: 'some-new-feature'
```

Schema enforcement

The schema for validating the feature versioning lives in [`src/content-linter/lib/feature-versions-schema.js`] and is exercised by [`tests/linting/lint-versioning.js`].

Versioning for GitHub AE

GitHub Docs shows only the documentation for the latest version of GitHub AE. For more information, see "About the displayed version of GitHub AE documentation."

You can version documentation to appear in the current GitHub AE documentation set in two ways:

- In Liquid syntax, use `ghae` or `ghae > VERSION`.
  - For example, if you want documentation to be visible for GitHub AE 3.6 and later, use `ghae > 3.5`.
  - If you're confident that the change is available in the latest public release of GitHub AE and you want documentation to appear immediately on GitHub Docs, you can always just use `ghae`.
- In YAML front matter or feature-based versioning files, use `ghae: '> VERSION'`.
  - For example, if you want documentation to be visible for GitHub AE 3.6 and later, use `ghae: '> 3.5'`.
  - If you're confident that the change is available in the latest public release of GitHub AE and you want documentation to appear immediately on GitHub Docs, you can always use `ghae: '*'`.

About the displayed version of GitHub AE documentation

We display one version of GitHub AE documentation on GitHub Docs: the latest. We define "latest" in the site's source, within `all-versions.js`. In almost all cases, this value is a safe place to determine the latest public release of GitHub AE.

"Latest" corresponds with the latest version of GitHub AE that we've deployed to the first customer using the product in production. Generally, GitHub upgrades all production customers on GitHub AE to the latest version around the same time, so Product considers only showing the latest documentation an acceptable compromise.

Product notifies Docs Content of new releases via release issues in `github/releases`.

Best practices

Versioned content impacts the reader, but also impacts anyone who contributes to or reviews the content. Here are a few tips to improve the writing, reading, and reviewing experience for versioning syntax. None of these practices are mandatory and you will find edge and corner cases, but they're intended as useful heuristics to help you think through versioning.

Avoid unnecessary versioning

For the reader, gaining a general understanding is more important than reading details that precisely reflect the differences between particular products or plans. In conceptual or procedural content, try to describe features or portions of the UI in a general way that doesn't require versioning syntax. In addition to being easier for us to maintain, this strengthens understanding for readers who refer to documentation for multiple products.

- Do ask yourself, "can I write this content in a way that applies to all products without any versioning?"
- Do try to avoid versioning screenshots if you can, given the effort required to create them. Minor differences between UI copy may not affect understanding. If product-specific text or UI elements exist, but the screenshot still provides helpful context, ask yourself whether versioning the screenshots would affect understanding to a meaningful degree.
- Don't version prose if you can explain a concept or walk the reader through a procedure without versioning for specific products.

When modifying an existing content file, review existing versioning early and often

Staying cognizant of existing versioning will help ensure that you write relevant versioning statements, and can help remind you to version new content accurately.

- Do review the entire page's versioning in the front matter as soon as you begin editing.
- Do review the versioning around content that you're editing.
- Do review the rendered version of changes that you're making, and switch to each available version for the page as part of your self-review.

Avoid repetition as much as possible

Use versioning syntax within a sentence or paragraph to differentiate prose for two different plans or products. A contributor can edit just one paragraph with versioning statements, instead of needing to consider larger blocks of versioned text and modify similar but differently versioned prose in two places. A reviewer can suggest a change once, instead of needing to leave the same suggestion in multiple places. But if the behavior differs dramatically or versioning within the sentence or paragraph becomes complicated or difficult for a contributor to parse, consider repeating yourself to make the prose easier to maintain.

- Do use versioning syntax inline within paragraphs to avoid repeating sentences or entire paragraphs.

  > You can do {% ifversion fpt %}something{% elsif ghec %}something else{% endif %}.

- Do use your judgment: for prose that would be complicated to write or read without lots of versioning syntax within a sentence or paragraph, consider repeating the entire paragraph in a version block for each relevant product.

  > {% ifversion fpt %}
  >
  > If you use a Free, Pro, or Team plan, you can do something. Here's more

> information about the things you can do with a Free, Pro, or Team plan...
>
> {% elsif ghec %}
>
> If you use GitHub Enterprise Cloud, you can do something else. Here's more information about the things you can do with GitHub Enterprise Cloud...
>
> {% endif %}

Be explicit, not implicit

If you know exactly which products the content describes, version explicitly for those products. Syntax like `not`, and `else` in particular, can be imprecise. The end result of `not` and `else` depend on each article's front matter, so a contributor must do more investigation to understand prose with this versioning. This creates the potential for errors. The complexity of implicit versioning increases in reusables, where articles that reference the reusable may have different versioning, and thus different evaluations of `not` or `else`. We also occasionally introduce a new version to GitHub Docs when GitHub introduces a new product, which changes the end result of `not` and `else` when we add the new version to existing articles.

- Do remember that GitHub offers four products, and remember that GitHub Docs can display documentation for eight total versions at any given time.
- Do review an entire article's versioning in the front matter when you begin editing, as this can help you understand how `not` and `else` will behave in Liquid statements, or change when you enable new versions in the front matter.

Verify and communicate versioning as you work through content design and creation

Sometimes a change isn't included in the release it was originally intended for. You can save time for reviewers and ensure more accurate content by confirming versioning throughout content design and creation, for both releases and improvements.

- Do consider versioning in content design, and do double-check the versioning when you request stakeholder reviews for content creation.
- Do make the review easier for other writers and stakeholders: point out differences between versions in your request for review, linking to specific rendered versions of the content if necessary.
- Do trust, but verify.

Test, test, and test again

Whether you're writing the content or reviewing the content, pay attention to the content design plan and affected products, and check the rendered content in a staging or development environment to ensure that the content describes each product accurately.