Creating reusable content

In this article

You can create reusable content that can be referenced in multiple content files.

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 reusables

Reusables are long strings of reusable text, such as paragraphs or procedural lists, that can be referenced in multiple content files.

We use Markdown (instead of YAML) for reusables. Markdown makes it possible for our localization pipeline to split the strings into smaller translatable segments, leading to fewer translation errors and less churn when the source English content changes.

Each reusable lives in its own Markdown file.

The path and filename of each Markdown file determines what its path will be in the data object. For example, a file named `/data/reusables/foo/bar.md` will be accessible as `{% data reusables.foo.bar %}` in pages.

Reusable files are divided generally into directories by task. For example, if you're creating a reusable string for articles about GitHub notifications, you'd add it in the directory `data/reusables/notifications/`, in a file named `data/reusables/notifications/YOUR-REUSABLE-NAME.md`. The content reference you'd add to the source would look like `{% data reusables.notifications.YOUR-REUSABLE-NAME %}`.

Applying versioning to reusables

Reusables can include Liquid conditionals to conditionally render content depending on the current version being viewed.

About variables

Variables are short strings of reusable text.

We use YAML files for variables.

The path, filename, and keys within each YAML file determine what its path will be in the data object.

For example, this YAML file, `data/variables/foo/bar.yml`, contains two variables:

```yaml
# the YAML file can contain multiple short strings in one file
meaning_of_life: 42

# the strings can also be nested if needed
nested:
  values:
    too: Yes!
```

The values would be accessible as `{% data foo.bar.meaning_of_life %}` and `{% data foo.bar.nested.values.too %}`.