Annotating code examples

You can annotate longer code examples to explain how they work and how people can customize them for other uses.

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

In this article

About code annotations

Code annotations help explain longer code examples by describing what a code example does and why. The annotations render next to the code example in a two pane layout, so we can write longer annotations without making the code itself difficult to read. We only annotate full code examples, not snippets. Code annotations are not required for every code example and should only be used if there is a clear need for them.

Code annotations can be helpful for a variety of audiences. Often, code annotations will be used to explain key concepts to new users or specific choices to more experienced users.

For new users, code annotations are a way to go beyond the high level overview of a code example and explain what each line of code does so that someone can understand the code as if a friend or coworker were guiding them through it.

For more experienced users, code annotations can help them understand a code example and then tailor it to their specific needs. Annotations can explain why code was written a certain way so that the fundamentals are clear.

You can annotate multiple code examples in a single article, but keep in mind that each annotation increases the complexity of an article and adds repetitive navigation tasks for people using screen readers. If you have multiple code examples in an article, consider whether they can be combined into a single example.

Enabling and adding code annotations

1. Specify the `layout: inline` frontmatter property for the article.
2. Create a code example using triple backticks.
3. Specify a language for the code example after the triple backtick, followed by `annotate`. For example, ```` ```yaml annotate ```` or ```` ```ruby annotate ````.

4. Add annotations using comment tags (`#`, `//`, `<&#33--`, `%%`) within the code example. You must use the comment tag for the language that the code sample is written in. For example, `#` for YAML and `//` for JavaScript.
    - An annotated code example must start with a single line annotation. You can start with a blank annotation if you do not want to add an annotation to the first line of code.
    - Annotations apply to the code from the line below the comment tag to the next comment tag or the end of the code block.

Annotation rules

The following rules apply to all code annotations.

- Multiline-style comments, such as `/*` are not supported.
- You can include any number of spaces before the comment tag starts.
- You can include any number of spaces after the comment tag ends.
- To create a blank annotation, insert a comment tag with no text after it. Blank annotations are useful if some lines of a sample don't require an annotation.
- Anything after the comment tag will be parsed with Markdown. Links, versioning, and other styling will render as if they were written in Markdown.
- Multiple sequential comments will create a single annotation.
- Lines that do not start with a comment tag and are empty or only contain spaces will be ignored.
- You must start the code section with a single line comment. If the first line (or section) of the code does not need an annotation, you can use a comment tag with no text to create a blank annotation.
- For HTML style, you should include a closing tag, ``, after your annotations to maintain syntax highlighting.

Code annotations best practices

Introduce the overall purpose of a code example with an introduction before the code block and use annotations to explain what specific lines of code do and why they do it.

Prioritize clarity in code annotations while trying to keep them as short as possible. People use code samples as a foundation for their own work, so annotations should help people understand the sample as it is written and how they might adapt the sample for other uses.

Consider your audience when writing code annotations and do not assume people will know why an example is written a certain way.

Annotations can be used to show the expected outcomes for the code that they annotate, but the results for the entire code example should be in whichever way best serves the audience: either the introduction for the code example or discussed after the example.

If a code example is changed, check that all annotations are still valid.

Example of an annotated code example

The following examples show what rendered code annotations look like and the raw code that creates them.

Rendered example

The following code example shows a workflow that posts a welcome comment on a pull request when it is opened.

YAML Beside Inline

```yaml
name: Post welcome comment
```

The name of the workflow as it will appear in the "Actions" tab of the GitHub repository.

```yaml
on:
```

The `on` keyword lets you define the events that trigger when the workflow is run.

```yaml
  pull_request:
    types: [opened]
```

Add the `pull_request` event, so that the workflow runs automatically every time a pull request is created.

```yaml
permissions:
  pull-requests: write
```

Modifies the default permissions granted to `GITHUB_TOKEN`.

```yaml
jobs:
  build:
    name: Post welcome comment
```

Defines a job with the ID `build` that is stored within the `jobs` key.

```yaml
    runs-on: ubuntu-latest
```

Configures the operating system the job runs on.

```yaml
    steps:
      - run: gh pr comment $PR_URL --body "Welcome to the repository!"
        env:
          GITHUB_TOKEN: $
          PR_URL: $
```

The `run` keyword tells the job to execute a command on the runner.

Raw code example

```
The following code example shows a workflow that posts a welcome comment on a
pull request when it is opened.

```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 a command on the runner.
    steps:
      - run: gh pr comment $PR_URL --body "Welcome to the repository!"
        env:
          GITHUB_TOKEN: $
          PR_URL: $
```