Troubleshooting your environment

In this article

Learn about troubleshooting problems in your local environment and the GitHub Docs staging platform.

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

Troubleshooting tests that fail locally but pass in CI

If you run tests locally and get failures in `tests/rendering/server.js` around static assets, stylesheets, or the client-side JavaScript bundle, but the same tests pass in CI on a PR, run the command `npm run build`. This is a one-time command that creates static assets locally.

For more information, see "Creating a local environment."

Troubleshooting stalled staging deployments

If a staging deployment is pending for more than ten minutes, try closing your pull request (without deleting the branch) and reopening it. This will trigger a new staging deployment. It won't break anything.

If that doesn't work, use the commands below to trigger a new staging deployment by pushing an empty commit on the command line.

```
git commit --allow-empty -m 'empty commit to redeploy staging'
git push
```

Troubleshooting stalled or stuck CI

If your tests are stuck on "In progress" or "Pending" for more than an hour, use the commands below to rerun CI by pushing an empty commit on the command line.

```
git commit --allow-empty -m 'empty commit to rerun CI'
git push
```

Troubleshooting local server problems

If you are running `script/server` and get a `Cannot find module` error, try the following command before restarting the server.

```
npm install
```

If that doesn't fix the problem, use the following command to remove the `node_modules` directory and reinstall.

```
rm -rf node_modules
npm install
```

Troubleshooting staging problems

If you are having trouble with the staging server, you should see more information about the error in your browser or on the command line if you run the site locally. Check out your branch locally and use the following command to launch the local server.

```
script/server
```

When the server is running, navigate to the problematic article on `https://localhost:4000` in your browser. The staging server will just show an "Oops" error, but the local server should show a stack trace for debugging.

If you see an error resembling the one below, make sure single quotes are properly escaped in the frontmatter. Also, check the formatting in `redirect_from` blocks. For more information, see "Using YAML frontmatter."

```
error parsing file:
/Users/z/git/github/docs/content/dotcom/articles/troubleshooting-custom-domains-
and-github-pages.md
(node:89324) UnhandledPromiseRejectionWarning: YAMLException: can not read a
block mapping entry; a multiline key may not be an implicit key at line 4, column
14:
    redirect_from:
                 ^
```

Checking internal links

The "Link Checker: On PR" test reports broken links on the site, including images. If there are broken links, the test will fail and the test's details view will show either `TitleFromAutotitleError` errors, which just report the broken link URL, or a more descriptive report which also lists the page that contains the broken link.

If the error does not include the location of the broken link, you will need to search the `docs` repository for the broken link to find the file.

When you locate the broken link, make sure the link is versioned correctly. For example, if the article only exists for GHES version 3.8+, make sure the link is versioned for 3.8+.

If an article that is available for GitHub Enterprise Server links to a GitHub.com-only

article, include the version in the path to prevent the URL from automatically converting to include a GitHub Enterprise Server version number. The following example demonstrates how to link from a GitHub Enterprise Server article to a GitHub.com-only article.

```
[{{ data variables.product.prodname_github_connect }} Addendum to the {{ data
variables.product.prodname_enterprise }} License Agreement](/free-pro-
team@latest/articles/github-connect-addendum-to-the-github-enterprise-license-
agreement/)"
```

Debugging locally

During development, you can visit any page on `http://localhost:4000` and add `?json=page` to the end of the path to show some underlying information that may be helpful for debugging. In addition to basic info like title and intro, these are a few fields that may be useful.

Field	Description
`productVersions`	Shows what the site is parsing from the `productVersions` frontmatter.
`permalinks`	Shows all permalinks that the site is generating for the page.
`redirect_from`	Shows the hardcoded redirects in the `redirect_from` frontmatter.
`redirects`	Shows all redirects that the site is generating for the page.
`includesPlatformSpecificContent`	Shows whether the site detects any platform-specific content on the page.

Working with liquid processing

If your text or code example includes content between curly brackets (`{` and `}`), you need to wrap it between `&#123% raw %}` and `&#123% raw %}` tags to disable Liquid processing for that section. For example:

- **Use**:

  ```
  GITHUB_TOKEN: {% raw %}${{ secrets.GITHUB_TOKEN }}{% endraw %}
  ```

- **Avoid**:

  ```
  GITHUB_TOKEN: $${{ secrets.GITHUB_TOKEN }}$
  ```