How to Open a Pull Request

A big part of contributing to open source is submitting changes to a project: improvements to source code or tests, updates to docs content, even typos or broken links. This doc will cover what you need to know to **open a pull request** in Gatsby.

What is a Pull Request (PR)?

In case you aren't familiar, here's how the folks at GitHub define a pull request:

> Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.

Gatsby uses the PR process to review and test changes before they're added to Gatsby's GitHub repository. Anyone can open a pull request. The same process is used for all contributors, whether this is your first open source contribution or you're a core member of the Gatsby team.

When someone wants to contribute to Gatsby, they open a request to *pull* their code into the repo. Depending on the type of change, PRs are categorized into:

- Documentation
- Code

Recommendations for different kinds of contributions will follow in this guide and throughout the contributing docs.

Things to know before opening a PR

We typically recommend opening an issue before a pull request if there isn't already an issue for the problem you'd like to solve. This helps facilitate a discussion before deciding on an implementation.

For some changes, such as typo fixes or broken links, it may be appropriate to open a small PR by itself.

Opening PRs in Gatsby

For any kind of change to files in the Gatsby repo, you can follow the below steps. Be sure to check out additional tips for contributing to various parts of the repo later in this doc, such as docs changes, blog posts, starters, or code improvements and tests.

To test changes locally against the Gatsby site and project files, fork the repo and install parts of it to run on your local machine.

- Fork and clone the Gatsby repo.
- Follow the instructions for the part of the project you want to change. (See specific sections below.)
- Create a branch in Git to isolate your changes:

  ```
  git checkout -b some-change
  ```

- Once you have changes in Git you want to push, add them and create a commit. For information on how to structure your commits, check out the Managing PRs doc.
  - Using a dot character `.` will add all untracked files in the current directory and subdirectories.

  ```
  git add .
  ```

  - Using a visual tool like GitHub Desktop or GitX can help for choosing which files and lines to commit.
- Committing code will run the automated linter using Prettier. To run the linter manually, run an npm script in the project's base directory:

  ```
  npm run format
  ```

- Commit any linting changes before pushing by amending the previous commit or by adding a new commit. For more on linting and tests, visit the Managing PRs doc.

  ```
  git commit --amend
  ```

- Push your changes to your fork, assuming it is set up as `origin`:

  ```
  git push origin head
  ```

- To open a PR with your changes against the Gatsby repo, you can use the GitHub Pull Request UI. Alternatively, you can use the command line: we recommend hub for that.

Documentation PRs

The Gatsby docs site lives in docs directories on GitHub, including docs and tutorial content. There are also some examples in the Gatsby repo referenced in

the docs.

Additional docs PR steps:

- For docs-only changes, consider using `git checkout -b docs/some-change` or `git checkout -b docs-some-change`, as this will short circuit the CI process and only run linting tasks.

Further instructions can be found on the docs contributions page.

Code changes

Instructions for making changes to the Gatsby source code, tests, internals, APIs, packages, and more can be found in the contributing docs on setting up your local dev environment.

Preparing a PR for review

When preparing and merging a PR for the monorepo you'll need to follow a couple of conventions:

- A bot automatically adds a `status: triage needed` label to your PR. A team member will remove this label and add appropriate ones
- Format the PR title to follow Conventional Commits format (more details below)
- Make sure that all required tests pass — if you think that a test is flaky and you're not sure, ask about it in the pull request
- Put the PR into *Draft* mode if it's still a work in progress
- Fill out the PR template (Description, Docs, Related Issues)

Conventional Commits Examples

When referring to folders/folder structures the root of the monorepo is assumed.

- If solely something inside `/docs` is changed, e.g. someone fixes a typo in one of our docs or you're updating the tutorial, the PR title should be `chore(docs): $TEXT`
- If you're updating something in `/.github` or `.circleci` it'll be `chore: $TEXT`
- If you're only updating something in one package, e.g. in `/packages/gatsby-plugin-utils` the scope is the package name `gatsby-plugin-utils`
  - If only docs or README is updated inside the package, the title should be `chore(gatsby-plugin-utils: $TEXT`
  - If any other *chore* changes like bumping a patch version of a dependency is done, it's `chore(gatsby-plugin-utils): $TEXT`
  - If you're fixing a bug: `fix(gatsby-plugin-utils): $TEXT`
  - Adding a new feature: `feat(gatsby-plugin-utils): $TEXT`
  - Making a breaking change for the package: `BREAKING CHANGE(gatsby-plugin-utils): $TEXT` — please note that in this case you'll need to coordinate this

with team members as this requires a closer look & a special release process
  - Improving the performance: `perf(gatsby-plugin-utils): $TEXT`
- If you're updating two packages, you could combine it like so: `feat(gatsby,gatsby-plugin-utils): $TEXT`
- If you're updating multiple packages but the main gist of the change is in one package (e.g. you add a feature to `gatsby` and had to update some signatures in other packages) the rules of the "one package" example still apply, so e.g. `feat(gatsby): $TEXT`
- If you're updating multiple packages and there's not a clear package that it all relates to, you can skip the *scope* and only do `fix: $TEXT` , `feat: $TEXT`, etc.
- If you're updating dependencies in multiple packages you can use `chore(deps): $TEXT`

Follow up with reviews and suggestions

After a PR is sent to the Gatsby GitHub repo, the Gatsby core team and the community may suggest modifications to the changes that your PR introduces.

The Gatsby teams reviews and approves every PR that the community sends to make sure that it meets the contribution guidelines of the repo, and to find opportunities for improvement to your PR changes.

These suggestions may also be called "request changes" by the GitHub UI. When a change request is added to your PR, this and the rest of the change requests will appear on the GitHub page for your PR. From this page you can use the suggestions UI to:

- Review the suggested changes using the "View changes" button.
- Commit the suggestions.
- Discuss suggestions to ask questions about the suggested changes.
- Add suggestions to a batch so they can be pushed in a single commit.

For suggestions that may not be resolved using the GitHub UI, remember that you can keep adding related commits to your PR before it is merged and those commits will also be a part of such PR.

After all your questions have been resolved and the requested changes have been committed, you can mark the conversation as solved.

This process helps both the Gatsby team and the community to contribute with improvements for your changes before they are merged into the Gatsby GitHub repo.

Update your fork with the latest Gatsby changes

The Gatsby GitHub repo is very active, so it's likely you'll need to update your fork with the latest changes to be able to merge in your code. This requires

adding Gatsby as an upstream remote:

- Set Gatsby's repo URL as a remote source. The name of the remote is arbitrary; this example uses `upstream`.

  ```
  git remote add upstream git@github.com:gatsbyjs/gatsby.git
  ```

  - *Note: this syntax uses SSH and keys: you can also use `https` and your username/password.*

- You can verify the remote name and URL at any time:

  ```
  git remote -v
  ```

- Fetch the latest changes from Gatsby:

  ```
  git fetch upstream master
  ```

- In the branch you want to update, merge any changes from Gatsby into your fork:

  ```
  git merge upstream/master
  ```

  - If there are any merge conflicts, you'll want to address those to get a clean merge.

- Once your branch is in good working order, push the changes to your fork:

  ```
  git push origin head
  ```

For more information on working with upstream repos, visit the GitHub docs.

Additional resources

- CSS Tricks: How to Contribute to an Open Source Project
- Creating a pull request from GitHub
- Configuring a remote for a fork
- Which remote URL should I use?
- Git Branching and Merging
- Feature Branching and Workflows
- Resolving merge conflicts
- Guide on Markdown Syntax