# Contribution

Nuxt is a community project - and so we love contributions of all kinds! ❤️

There is a range of different ways you might be able to contribute to the Nuxt ecosystem.

## Ecosystem

The Nuxt ecosystem includes many different projects and organizations. For example:

- nuxt/ - core repositories for the Nuxt framework itself. **nuxt/nuxt** contains the Nuxt framework (both versions 2 and 3).
- nuxt-modules/ - community-contributed and maintained modules and libraries. There is a process to migrate a module to `nuxt-modules` . While these modules have individual maintainers, they are not dependent on a single person.
- unjs/ - many of these libraries are used throughout the Nuxt ecosystem. They are designed to be universal libraries that are framework- and environment-agnostic. We welcome contributions and usage by other frameworks and projects.

## How To Contribute

### Triage Issues and Help Out in Discussions

Check out the issues and discussions for the project you want to help. For example, here are the issues board and discussions for Nuxt 3. Helping other users, sharing workarounds, creating reproductions, or even poking into a bug a little bit and sharing your findings makes a huge difference.

## Creating an Issue

Thank you for taking the time to create an issue! ❤️

- **Reporting bugs**: Check out our guide for some things to do before opening an issue.
- **Feature requests**: Check that there is not an existing issue or discussion covering the scope of the feature you have in mind. If the feature is to another part of the Nuxt ecosystem (such as a module), please consider raising a feature request there first. If the feature you have in mind is general or the API is not entirely clear, consider opening a discussion in the **Ideas** section to discuss with the community first.

We'll do our best to follow our internal issue decision making flowchart when responding to issues.

## Send a Pull Request

We always welcome pull requests! ❤️

### Before You Start

Before you fix a bug, we recommend that you check whether **there's an issue that describes it**, as it's possible it's a documentation issue or that there is some context that would be helpful to know.

If you're working on a feature, then we ask that you **open a feature request issue first** to discuss with the maintainers whether the feature is desired - and the design of those features. This helps save time for both the maintainers and the contributors and means that features can be shipped faster. The issue **should be confirmed** by a framework team member before building out a feature in a pull request.

For typo fixes, it's recommended to batch multiple typo fixes into one pull request to maintain a cleaner commit history.

For bigger changes to Nuxt itself, we recommend that you first create a Nuxt module and implement the feature there. This allows for quick proof-of-concept. You can then create an RFC in the form of a discussion. As users adopt it and you gather feedback, it can then be refined and either added to Nuxt core or continue as a standalone module.

### Commit Conventions

We use Conventional Commits for commit messages, which allows a changelog to be auto-generated based on the commits. Please read the guide through if you aren't familiar with it already.

Note that `fix:` and `feat:` are for **actual code changes** (that might affect logic). For typo or document changes, use `docs:` or `chore:` instead:

- ~~`fix: typo`~~ `-> docs: fix typo`

If you are working in a project with a monorepo, like `nuxt/nuxt`, ensure that you specify the main scope of your commit in brackets. For example: `feat(nuxi): add 'do-magic' command`.

## Making the Pull Request

If you don't know how to send a pull request, we recommend reading the guide.

When sending a pull request, make sure your PR's title also follows the Commit Convention.

If your PR fixes or resolves existing issues, please make sure you mention them in the PR description.

It's ok to have multiple commits in a single PR; you don't need to rebase or force push for your changes as we will use `Squash and Merge` to squash the commits into one commit when merging.

We do not add any commit hooks to allow for quick commits. But before you make a pull request, you should ensure that any lint/test scripts are passing.

In general, please also make sure that there are no *unrelated* changes in a PR. For example, if your editor has made any changes to whitespace or formatting elsewhere in a file that you edited, please revert these so it is more obvious what your PR changes. And please avoid including multiple unrelated features or fixes in a single PR. If it is possible to separate them, it is better to have multiple PRs to review and merge separately. In general, a PR should do *one thing only*.

## Once You've Made a Pull Request

Once you've made a pull request, we'll do our best to review it promptly.

If we assign it to a maintainer, then that means that person will take special care to review it and implement any changes that may be required.

If we request changes on a PR, please ignore the red text! It doesn't mean we think it's a bad PR - it's just a way of easily telling the status of a list of pull requests at a glance.

If we mark a PR as 'pending', that means we likely have another task to do in reviewing the PR - it's an internal note-to-self, and not necessarily a reflection on whether the PR is a good idea or not. We will do our best to explain via a comment the reason for the pending status.

We'll do our best to follow our PR decision making flowchart when responding and reviewing to pull requests.

## Create a Module

If you've built something with Nuxt that's cool, why not extract it into a module, so it can be shared with others? We have many excellent modules already, but there's always room for more.

If you need help while building it, feel free to check in with us.

## Make an RFC

We highly recommend creating a module first to test out big new features and gain community adoption.

If you have done this already, or it's not appropriate to create a new module, then please start by creating a new discussion. Make sure it explains your thinking as clearly as possible. Include code examples or function signatures for new APIs. Reference existing issues or pain points with examples.

If we think this should be an RFC, we'll change the category to RFC and broadcast it more widely for feedback.

An RFC will then move through the following stages:

- `rfc: active` - currently open for comment
- `rfc: approved` - approved by the Nuxt team
- `rfc: ready to implement` - an issue has been created and assigned to implement
- `rfc: shipped` - implemented
- `rfc: archived` - not approved, but archived for future reference

## Conventions Across Ecosystem

The following conventions are *required* within the `nuxt/` organization and recommended for other maintainers in the ecosystem.

## Module Conventions

Modules should follow the Nuxt module template. See module guide for more information.

## Use Core `unjs/` Libraries

We recommend the following libraries which are used throughout the ecosystem:

- pathe - universal path utilities (replacement for node `path` )
- ufo - URL parsing and joining utilities
- unbuild - rollup-powered build system
- ... check out the rest of the unjs/ organization for many more!

## Use ESM Syntax and Default to `type: module`

Most of the Nuxt ecosystem can consume ESM directly. In general we advocate that you avoid using CJS-specific code, such as `__dirname` and `require` statements. You can read more about ESM.

## What's Corepack

Corepack makes sure you are using the correct version for package manager when you run corresponding commands. Projects might have `packageManager` field in their `package.json` .

Under projects with configuration as shown below, Corepack will install `v7.5.0` of `pnpm` (if you don't have it already) and use it to run your commands.

package.json
```json
{
  "packageManager": "pnpm@7.5.0"
}
```

## Use ESLint

We use ESLint for both linting and formatting with `@nuxt/eslint-config`.

### IDE Setup

We recommend using VS Code along with the ESLint extension. If you would like, you can enable auto-fix and formatting when you save the code you are editing:

```json
settings.json
{
  "editor.codeActionsOnSave": {
    "source.fixAll": false,
    "source.fixAll.eslint": true
  }
}
```

## No Prettier

Since ESLint is already configured to format the code, there is no need to duplicate the functionality with Prettier. To format the code, you can run `yarn lint --fix` , `pnpm lint --fix` , or `bun run lint --fix` or referring the ESLint section for IDE Setup.

If you have Prettier installed in your editor, we recommend you disable it when working on the project to avoid conflict.

**Note**: we are discussing enabling Prettier in future.

## Package Manager

For libraries, we recommend `pnpm` . For modules, we still recommend `yarn` but we may well switch this recommendation to `pnpm` in future once we support plug and play mode with Nuxt itself.

It is important to enable Corepack to ensure you are on the same version of the package manager as the project. Corepack is built-in to new node versions for seamless package manager integration.

To enable it, run

```
> corepack enable
```

You only need to do this one time, after Node.js is installed on your computer.

# Writing Documentation

Documentation is an essential part of Nuxt. We aim to be an intuitive framework - and a big part of that is making sure that both the developer experience and the docs are perfect across the ecosystem. 👌

Here are some tips that may help improve your documentation:

- Avoid subjective words like *simply*, *just*, *obviously...* when possible.
  Keep in mind your readers can have different backgrounds and experiences. Therefore, these words don't convey meaning and can be harmful.

  ❌ **Avoid:** "Simply make sure the function returns a promise."

  ✅ **Prefer:** "Make sure the function returns a promise."

- Prefer active voice.

  ❌ **Avoid:** "An error will be thrown by Nuxt."

  ✅ **Prefer:** "Nuxt will throw an error."

---

Edit on Github