Automatically lint and format code before commits

- The main command: bin/lint
  - What each mode does
- What gets linted
  - Ruby
  - JavaScript, GJS, CSS, and SCSS formatting
  - JavaScript and GJS linting
  - SCSS linting
  - YAML and locale checks
  - Type checking
- What can be auto-fixed
- Relationship to git hooks
- Practical workflow

Discourse uses lefthook for git hooks, and `bin/lint` as the main CLI entry point for running the same checks manually.

If you are working in a local clone, install the hooks once:

```
pnpm install
pnpm lefthook install
```

After that, staged files will be checked automatically on `git commit`.

The main command: `bin/lint`

Use `bin/lint` when you want to run the repo's configured linters yourself instead of waiting for the pre-commit hook.

Common examples:

```
bin/lint
bin/lint path/to/file.rb path/to/file.gjs
bin/lint --recent
bin/lint --staged
bin/lint --unstaged
bin/lint --wip
bin/lint --fix path/to/file.rb
bin/lint --fix --recent
bin/lint --fix
```

What each mode does

- `bin/lint`: lint all supported files in the repository
- `bin/lint path/to/file ...`: lint only the given files
- `bin/lint --recent`: lint files changed in the last 50 commits, plus untracked files
- `bin/lint --staged`: lint only staged files
- `bin/lint --unstaged`: lint only unstaged files
- `bin/lint --wip`: lint staged files, unstaged files, and files changed since `main`
- `bin/lint --fix ...`: run the auto-fixers for the selected files
- `bin/lint --fix`: run all available auto-fixers across the repository
- `bin/lint --verbose`: print the underlying lefthook commands

When you pass explicit files, `bin/lint` filters them to supported lintable file types before invoking lefthook.

> :information_source: Markdown files are only linted under `docs/developer-guides/`. Running `bin/lint path/to/doc.md` on any other Markdown file will report that nothing was linted.

What gets linted

The exact configuration lives in `lefthook.yml`. At the time of writing, `bin/lint` covers:

Ruby

- `**/*.{rb,rake,thor}`

- Ruby scripts under `bin/**/*`
- `Gemfile`

Checks:

- `rubocop`
- `syntax_tree` (`stree check`)

JavaScript, GJS, CSS, and SCSS formatting

- `app/assets/stylesheets/**/*.{css,scss}`
- `frontend/**/*.{js,gjs,scss,css,cjs,mjs}`
- matching plugin and theme asset files

Checks:

- `prettier`/`pprettier`

JavaScript and GJS linting

- `frontend/**/*.{js,gjs}`
- matching plugin and theme JS files

Checks:

- `eslint` (with `eslint-plugin-ember`'s `template-*` rules covering the template portion of `.gjs` files)

SCSS linting

- `app/assets/stylesheets/**/*.scss`
- matching plugin and theme SCSS files

Checks:

- `stylelint`

YAML and locale checks

- `**/*.{yaml,yml}` except `config/database.yml`
- `**/{client,server}.en.yml`

Checks:

- `yaml-lint`
- `script/i18n_lint.rb`

Type checking

When you run `bin/lint` with no file arguments, the full-repo lint also runs:

- `pnpm lint:types`

This is the Glint/TypeScript-style check for Discourse's type information. It runs in TypeScript's `strict` mode and also generates the published declarations.

> :information_source: `bin/lint path/to/file` and the pre-commit hook do **not** run the full type check. Use plain `bin/lint` when you want the complete repo-wide lint pass.

What can be auto-fixed

`bin/lint --fix` can automatically fix a lot of issues, but not all of them.

Auto-fix is configured for:

- `prettier --write`
- `eslint --fix`
- `stylelint --fix`
- `rubocop -A`
- `syntax_tree` (`stree write`)

In practice, that means `--fix` can reformat and rewrite:

- Ruby
- JavaScript / GJS
- CSS / SCSS

These checks are **not** auto-fixed by `bin/lint --fix`:

- YAML syntax validation
- i18n linting for `client.en.yml` / `server.en.yml`
- Glint/type checking

Relationship to git hooks

The pre-commit hook uses the same lefthook configuration as `bin/lint`, but it runs only against staged files.

That means:

- a commit can fail because staged files do not pass linting
- `bin/lint --staged` is the closest manual equivalent to the pre-commit hook
- `bin/lint --fix --staged` is a good way to repair exactly what you are about to commit

Practical workflow

For day-to-day development, these are the most useful commands:

```
# Before committing a couple of changed files
bin/lint --fix path/to/file1.rb path/to/file2.gjs

# Check exactly what the pre-commit hook will check
bin/lint --staged

# Clean up all current in-progress work
bin/lint --fix --wip

# Run the full repo lint suite, including type checks
bin/lint
```

□□□ #2
□ □□□ □□□ 2026-10-31 13:27:49 UTC
□ □□□ □□□ 2026-10-31 13:27:50 UTC