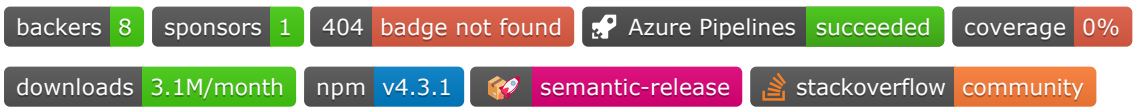
commitizen

4.3.1 • Public • Published 4 months ago

Commitizen for contributors

When you commit with Commitizen, you'll be prompted to fill out any required commit fields at commit time. No more waiting until later for a git commit hook to run and reject your commit (though **that** can still be helpful). No more digging through **CONTRIBUTING.md** to find what the preferred format is. Get instant feedback on your commit message formatting and be prompted for required fields.



Installing the command line tool

Commitizen is currently tested against Node.js 12, 14, & 16, although it may work in older versions of Node.js. You should also have npm 6 or greater.

Installation is as simple as running the following command (if you see `EACCES` error, reading **fixing npm permissions** may help):

```
npm install -g commitizen
```

Using the command line tool

If your repo is Commitizen friendly:

Simply use `git cz` or just `cz` instead of `git commit` when committing. You can also use `git-cz`, which is an alias for `cz`.

Alternatively, if you are using **npm 5.2+** you can use `npx` instead of installing globally:

```
npx cz
```

or as an npm script:

```
...
"scripts": {
  "commit": "cz"
}
```

When you're working in a Commitizen-friendly repository, you'll be prompted to fill in any required fields, and your commit messages will be formatted according to the standards defined by project maintainers.

```
➜ ng-poopy  master ✗  git add .
➜ ng-poopy  master ✗  git cz

All commit message lines will be cropped at 100 characters.

? Select the type of change that you're committing: (Use arrow keys)
❯ feat:     A new feature
  fix:      A bug fix
  docs:     Documentation only changes
  style:    Changes that do not affect the meaning of the code
            (white-space, formatting, missing semi-colons, etc)
  refactor: A code change that neither fixes a bug or adds a feature
  perf:     A code change that improves performance
  test:     Adding missing tests
  chore:    Changes to the build process or auxiliary tools
            and libraries such as documentation generation
```

If your repo is NOT Commitizen friendly:

If you're **not** working in a Commitizen-friendly repository, then `git cz` will work just the same as `git commit`, but `npx cz` will use the **streamich/git-cz** adapter. To fix this, you need to first **make your repo Commitizen friendly**

Making your repo Commitizen friendly

For this example, we'll be setting up our repo to use **AngularJS's commit message convention**, also known as **conventional-changelog**.

First, install the Commitizen CLI tools:

```
npm install commitizen -g
```

Next, initialize your project to use the cz-conventional-changelog adapter by typing:

```
# npm
commitizen init cz-conventional-changelog --save-dev --save-exact

# yarn
commitizen init cz-conventional-changelog --yarn --dev --exact

# pnpm
commitizen init cz-conventional-changelog --pnpm --save-dev --save-e
```

Note that if you want to force install over the top of an old adapter, you can apply the `--force` argument. For more information on this, just run `commitizen help`.

The above command does three things for you:

1. Installs the cz-conventional-changelog adapter npm module
2. Saves it to `package.json`'s `dependencies` or `devDependencies`
3. Adds the `config.commitizen` key to the root of your `package.json` file as shown here:

```
...
  "config": {
```

```
    "commitizen": {
      "path": "cz-conventional-changelog"
    }
  }
```

Alternatively, Commitizen configs may be added to a `.czrc` file:

```
{
  "path": "cz-conventional-changelog"
}
```

This just tells Commitizen which adapter we actually want our contributors to use when they try to commit to this repo.

`commitizen.path` is resolved via **require.resolve** and supports:

- npm modules
- directories relative to `process.cwd()` containing an `index.js` file
- file base names relative to `process.cwd()` with `.js` extension
- full relative file names
- absolute paths

Please note that in the previous version of Commitizen we used czConfig. **czConfig has been deprecated**, and you should migrate to the new format before Commitizen 3.0.0.

Optional: Install and run Commitizen locally

Installing and running Commitizen locally allows you to make sure that developers are running the exact same version of Commitizen on every machine.

Install Commitizen with `npm install --save-dev commitizen`.

On **npm 5.2+** you can **use `npx`** to initialize the conventional changelog adapter:

```
npx commitizen init cz-conventional-changelog --save-dev --save-exact
```

For **previous versions of npm (< 5.2)** you can execute `./node_modules/.bin/commitizen` or `./node_modules/.bin/cz` in order to

actually use the commands.

You can then initialize the conventional changelog adapter using:
`./node_modules/.bin/commitizen init cz-conventional-changelog --save-dev --save-exact`

And you can then add some nice npm scripts in your `package.json` file pointing to the local version of Commitizen:

```
...
"scripts": {
  "commit": "cz"
}
```

This will be more convenient for your users because then if they want to do a commit, all they need to do is run `npm run commit` and they will get the prompts needed to start a commit!

> **NOTE:** If you are using `precommit` hooks thanks to something like `husky`, you will need to name your script something other than `"commit"` (e.g. `"cm": "cz"`). The reason is because npm scripts has a "feature" where it automatically runs scripts with the name *prexxx* where *xxx* is the name of another script. In essence, npm and husky will run `"precommit"` scripts twice if you name the script `"commit"`, and the workaround is to prevent the npm-triggered *precommit* script.

Optional: Running Commitizen on `git commit`

This example shows how to incorporate Commitizen into the existing `git commit` workflow by using git hooks and the `--hook` command-line option. This is useful for project maintainers who wish to ensure the proper commit format is enforced on contributions from those unfamiliar with Commitizen.

Once either of these methods is implemented, users running `git commit` will be presented with an interactive Commitizen session that helps them write useful commit messages.

> **NOTE:** This example assumes that the project has been set up to **use Commitizen locally**.

Traditional git hooks

Update `.git/hooks/prepare-commit-msg` with the following code:

```
#!/bin/bash
exec < /dev/tty && node_modules/.bin/cz --hook || true
```

Husky

For `husky` users, add the following configuration to the project's `package.json` file:

```
"husky": {
  "hooks": {
    "prepare-commit-msg": "exec < /dev/tty && npx cz --hook || true'
  }
}
```

> **Why `exec < /dev/tty` ?** By default, git hooks are not interactive. This command allows the user to use their terminal to interact with Commitizen during the hook.

Congratulations! Your repo is Commitizen friendly. Time to flaunt it!

Add the "Commitizen friendly" badge to your README using the following markdown:

```
[![Commitizen friendly](https://img.shields.io/badge/commitizen-frienc
```

Your badge will look like this:

It may also make sense to change your `README.md` or `CONTRIBUTING.md` files to include or link to the Commitizen project so that your new contributors may learn more about installing and using Commitizen.

Conventional commit messages as a global utility

Install `commitizen` globally, if you have not already.

```
npm install -g commitizen
```

Install your preferred `commitizen` adapter globally (for example `cz-conventional-changelog`).

```
npm install -g cz-conventional-changelog
```

Create a `.czrc` file in your `home` directory, with `path` referring to the preferred, globally-installed, `commitizen` adapter

```
echo '{ "path": "cz-conventional-changelog" }' > ~/.czrc
```

You are all set! Now `cd` into any `git` repository and use `git cz` instead of `git commit`, and you will find the `commitizen` prompt.

Pro tip: You can use all the `git commit` options with `git cz`. For example: `git cz -a`.

> If your repository is a **Node.js** project, making it **Commitizen friendly** is super easy.

If your repository is already **Commitizen friendly**, the local `commitizen` adapter will be used, instead of globally installed one.

Commitizen for multi-repo projects

As a project maintainer of many projects, you may want to standardize on a single commit message format for all of them. You can create your own node module which acts as a front-end for Commitizen.

1. Create your own entry point script

```
// my-cli.js

#!/usr/bin/env node
"use strict";

const path = require('path');
const bootstrap = require('commitizen/dist/cli/git-cz').bootstrap;

bootstrap({
  cliPath: path.join(__dirname, '../../node_modules/commitizen'),
  // this is new
  config: {
    "path": "cz-conventional-changelog"
  }
});
```

2. Add the script to your `package.json` file

```
// package.json

{
  "name": "company-commit",
  "bin": "./my-cli.js",
  "dependencies": {
    "commitizen": "^2.7.6",
    "cz-conventional-changelog": "^1.1.5"
  }
}
```

3. Publish it to npm and use it!

```
npm install --save-dev company-commit

./node_modules/.bin/company-commit
```

Adapters

We know that every project and build process has different requirements, so we've tried to keep Commitizen open for extension. You can do this by choosing from any of the pre-built adapters or even by building your own. Here are some of the great adapters available to you:

- cz-conventional-changelog
- cz-conventional-changelog-for-jira
- cz-conventional-changelog-with-jiraid-detection
- cz-jira-smart-commit
- @endemolshinegroup/cz-jira-smart-commit
- @endemolshinegroup/cz-github
- rb-conventional-changelog
- @mapbox/cz-mapbox-changelog
- cz-customizable
- cz-commitlint
- commitlint
- vscode-commitizen
- cz-emoji
- cz-adapter-eslint
- commitiquette
- cz-format-extension
- cz-emoji-conventional
- cz-git
- cz-vinyl

To create an adapter, just fork one of these great adapters and modify it to suit your needs. We pass you an instance of **Inquirer.js**, but you can capture input using whatever means necessary. Just call the `commit` callback with a string and we'll be happy. Publish it to npm, and you'll be all set!

Retrying failed commits

As of version 2.7.1, you may attempt to retry the last commit using the `git cz --retry` command. This can be helpful when you have tests set up to run via a git precommit hook. In this scenario, you may have attempted a Commitizen commit, painstakingly filled out all of the commitizen fields, but your tests fail. In previous Commitizen versions, after fixing your tests, you would be forced to fill out all of the fields again. Enter the retry command. Commitizen will retry the last commit that you attempted in this repo without you needing to fill out the fields again.

Please note that the retry cache may be cleared when upgrading Commitizen versions, upgrading adapters, or if you delete the `commitizen.json` file in your home or temp directory. Additionally, the commit cache uses the filesystem path of the repo, so if you move a repo or change its path, you will not be able to retry a commit. This is an edge case but might be confusing if you have scenarios where you are moving folders that contain repos.

It is important to note that if you are running `cz` from an npm script (let's say it is called `commit`) you will need to do one of the following:

- Pass `-- --retry` as an argument for your script. i.e: `npm run commit -- -- retry`
- Use npx to find and call the `cz` executable directly. i.e: `npx cz --retry`

Note that the last two options **do not** require you to pass `--` before the args but the first **does**.

Commitizen for project maintainers

As a project maintainer, making your repo Commitizen friendly allows you to select pre-existing commit message conventions or to create your own custom commit message convention. When a contributor to your repo uses Commitizen, they will be prompted for the correct fields at commit time.

Go further

Commitizen is great on its own, but it shines when you use it with some other amazing open source tools. Kent C. Dodds shows you how to accomplish this in his Egghead.io series, **How to Write an Open Source JavaScript Library**. Many of the concepts can be applied to non-JavaScript projects as well.