# Contributing to Expo CLI

## How to contribute

You can contribute to Expo development tools in various ways, including:

- [Reporting bugs or issues](https://github.com/expo/expo-cli/issues/new) on GitHub. Please make sure to include fill in all the details in the issue template to make sure the issue can be addressed as quickly as possible.

- [Submitting improvements to the documentation](https://github.com/expo/expo-docs). Updates, enhancements, new guides, spelling fixes...

- Helping other people on the [forums](https://forums.expo.io).

- Looking at existing [issues](https://github.com/expo/expo-cli/issues) and adding more information, particularly helping to reproduce the issues.

- [Submitting a pull request](#submitting-a-pull-request) with a bug fix or an improvement.

## Expo CLI repository

The [Expo CLI GitHub repository](https://github.com/expo/expo-cli) contains the packages that make up Expo CLI.

### Packages

- `expo-cli`: Expo CLI is the command line interface for developing, building and sharing Expo apps.

- `@expo/dev-tools`: the web-based graphical user interface included in Expo CLI for quickly viewing logs, connecting testing devices, deploying updates and more.

- `xdl`: the Expo development library is a dependency of both the CLI and Dev Tools user interfaces, doing all the heavy lifting behind the scenes.

- `@expo/schemer`: a library for validating [Expo configuration files](https://docs.expo.io/workflow/configuration/).

- `@expo/json-file`: a library for reading and writing JSON files.

- `@expo/osascript`: a library for working with `osascript` which runs AppleScript code on macOS.

- `@expo/traveling-fastlane-darwin`/`@expo/traveling-fastlane-linux`: JavaScript wrappers for managing iOS certs, based on [Fastlane](https://fastlane.tools), which is a Ruby based app automation tool.

### Branches

The `master` branch of the repository should be kept releasable at any time. This way we can continuously deploy fixes and improvements without costly managing of different branches and issues will be noticed and fixed quickly. This also ensures other contributors can check out the latest version from GitHub and work on it with minimal disruption from other features in progress.

Keeping the `master` releasable means that changes merged to it need to be:

- \*\*Backwards compatible\*\*: Expo CLI should work with \_every currently supported Expo SDK version\_. If the code you're adding depends on a feature only present in newer or unreleased SDK versions, it needs to check which SDK version is being used and not assume the latest version is being used.

- \*\*Non-breaking\*\*: If code that is unreleasable or fails the test suite ends up in `master`, it should be reverted.

- \*\*Tested\*\*: Always include a test plan in pull requests. Do not merge code that doesn't pass all automated tests.

## Setting up the repository for development

1. Clone the repository.

2. Run `yarn`. (Installs dependencies and links packages in the workspace.)

3. Run `git config commit.template .github/.COMMIT\_TEMPLATE` (Sets you up with our commit message template)

4. Run `yarn start` in the root folder. (Start watching and automatically re-building packages when there are new changes.)

## Submitting a pull request

To submit a pull request:

1. Fork the [repository](https://github.com/expo/expo-cli) and create a feature branch. (Existing contributors can create feature branches without forking. Prefix the branch name with `@your-github-username/`.)

2. Write the description of your pull request. Make sure to include a test plan and test your changes.

3. Make sure all tests pass on CircleCI.

4. Wait for a review and adjust the code if necessary.

## Publishing a release

To publish a new release, run this command (you must have two-factor authentication enabled for npm):

```

./scripts/publish.js

```

The command will bump the versions of all packages with changes since the previous release and publish them in the correct order. For each changed package, it will ask, if the changes require a new \_major\_ version (breaking changes), \_minor\_ version (new backwards compatible functionality) or just a \_patch\_ version (backwards compatible bug fixes).

### Canary release

If you wish to publish a canary version, please run:

```

./scripts/publish.js --bump prerelease

```