# Contributing to Cypress

Thanks for taking the time to contribute! :smile:

\*\*Once you learn how to use Cypress, you can contribute in many ways:\*\*

- Join the [Cypress Gitter chat](https://on.cypress.io/chat) and answer questions. Teaching others how to use Cypress is a great way to learn more about how it works.

- Blog about Cypress. We display blogs featuring Cypress on our [Examples](https://on.cypress.io/examples) page. If you'd like your blog featured, [open a PR to add it to our docs](https://github.com/cypress-io/cypress-documentation/blob/develop/CONTRIBUTING.md#adding-examples).

- Write some documentation or improve our existing docs. Know another language? You can help us translate them. See our [guide to contributing to our docs](https://github.com/cypress-io/cypress-documentation/blob/master/CONTRIBUTING.md).

- Give a talk about Cypress. [Contact us](mailto:support@cypress.io) ahead of time and we'll send you some swag. :shirt:

\*\*Want to dive deeper into how Cypress works? There are several ways you can help with the development of Cypress:\*\*

- [Report bugs](https://github.com/cypress-io/cypress/issues/new) by opening an issue.

- [Request features](https://github.com/cypress-io/cypress/issues/new) by opening an issue.

- [Help triage existing issue](#triaging-issues).

- Write code to address an issue. We have some issues labeled as [`first-timers-only`](https://github.com/cypress-io/cypress/labels/first-timers-only) that are good place to start. [Please thoroughly read our writing code guide](#writing-code).

## Table of Contents

- [CI Status](#ci-status)

- [Code of Conduct](#code-of-conduct)

- [Opening Issues](#opening-issues)

- [Triaging Issues](#triaging-issues)

- [Writing Documentation](#writing-documentation)

- [Writing Code](#writing-code)

- [What you need to know before getting started](#what-you-need-to-know-before-getting-started)

- [Requirements](#requirements)

- [Getting Started](#getting-started)

- [Coding Style](#coding-style)

- [Adding links within code](#Adding-links-within-code)

- [Tests](#tests)

- [Packages](#packages)

- [Committing Code](#committing-code)

- [Branches](#branches)

- [Pull Requests](#pull-requests)

- [Testing](#testing)

- [Dependencies](#dependencies)

- [Reviewing Code](#reviewing-code)

- [Some rules about Code Review](#Some-rules-about-Code-Review)

- [Steps to take during Code Review](#Steps-to-take-during-Code-Review)

- [Code Review Checklist](#Code-Review-Checklist)

- [Code Review of Dependency Updates](#Code-Review-of-Dependency-Updates)

- [Deployment](#deployment)

## CI status

| Build status | Description |

| :------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------ | :----------------------------------------------------------------------------------------- |

| [![CircleCI](https://circleci.com/gh/cypress-io/cypress-test-node-versions.svg?style=svg&circle-token=6a7c4e7e7ab427e11bea6c2af3df29c4491d2376)](https://circleci.com/gh/cypress-io/cypress-test-node-versions) | [cypress-test-node-versions](https://github.com/cypress-io/cypress-test-node-versions) |

| [![CircleCI](https://circleci.com/gh/cypress-io/cypress-test-ci-environments.svg?style=svg&circle-token=66a4d36c3966cbe476f13e7dfbe3af0693db3fb9)](https://circleci.com/gh/cypress-io/cypress-test-ci-environments) | [cypress-test-ci-environments](https://github.com/cypress-io/cypress-test-ci-environments) |

| [![CircleCI](https://circleci.com/gh/cypress-io/cypress-test-module-api.svg?style=svg&circle-token=317f79ae796e0ffd6cc7dd90859c0f67e5a306e7)](https://circleci.com/gh/cypress-io/cypress-test-module-api) | [cypress-test-module-api](https://github.com/cypress-io/cypress-test-module-api) |

| [![CircleCI](https://circleci.com/gh/cypress-io/cypress-test-nested-projects.svg?style=svg)](https://circleci.com/gh/cypress-io/cypress-test-nested-projects) | [cypress-test-nested-projects](https://github.com/cypress-io/cypress-test-nested-projects) |

| [![CircleCI](https://circleci.com/gh/cypress-io/cypress-on.svg?style=svg&circle-token=51ba85f5720654ee58212f45f6b9afc56d55d52a)](https://circleci.com/gh/cypress-io/cypress-on) | [cypress-on](https://github.com/cypress-io/cypress-on) |

| [![CircleCI](https://circleci.com/gh/cypress-io/cypress-test-node-versions.svg?style=svg&circle-token=6a7c4e7e7ab427e11bea6c2af3df29c4491d2376)](https://circleci.com/gh/cypress-io/cypress-test-node-versions) | [cypress-test-example-repos](https://github.com/cypress-io/cypress-test-example-repos) |

| [![CircleCI](https://circleci.com/gh/cypress-io/docsearch-scraper.svg?style=svg&circle-token=8087137233788ec1eab4f79d4451392ca53183b2)](https://circleci.com/gh/cypress-io/docsearch-scraper) | [docsearch-scraper](https://github.com/cypress-io/docsearch-scraper) |

| [![Docker Build Status](https://img.shields.io/docker/build/cypress/base.svg)](https://hub.docker.com/r/cypress/base/) | [cypress-docker-images](https://github.com/cypress-io/cypress-docker-images) |

| [![Build status](https://ci.appveyor.com/api/projects/status/ln8tg3dv42nk916c?svg=true)](https://ci.appveyor.com/project/cypress-io/cypress) | Windows CI |

## Code of Conduct

All contributors are expecting to abide by our [Code of Conduct](./CODE\_OF\_CONDUCT.md).

## Opening Issues

\*\*The most important things to do are:\*\*

- Search existing [issues](https://github.com/cypress-io/cypress/issues) for your problem.

- Understand our [roadmap](https://on.cypress.io/roadmap).

- [Update Cypress](#update-cypress).

- [Gather debugging information](#getting-more-information).

- [Fill out the provided issue template](#fill-out-our-issue-template).

- [Describe your problem, not your solution](#describe-problems)

- [Explain how to reproduce the issue](#reproducibility).

Finally, if you are up to date, supported, have collected information about the problem, and have the best reproduction instructions you can give, you are ready to [open an issue](https://github.com/cypress-io/cypress/issues/new).

### Update Cypress

Before filing a bug, make sure you are up to date. Your issue may have already been fixed. Even if you do not see the issue described as resolved in a newer version, a newer version may help in the process of debugging your issue by giving more helpful error messages.

[See our document on installing cypress](https://on.cypress.io/installing-cypress)

### Getting more information

For some issues, there are places you can check for more information. This may help you resolve the issue yourself. Even if it does not, this information can help us figure out and resolve an issue.

- For issues in the web browser, check the JavaScript console and your Network tab in your DevTools.

- Click on any command in the Command Log where the failure occurred, this will log more information about the error to the JavaScript console.

- Use Cypress [`debug`](https://on.cypress.io/debug) or [`pause`](https://on.cypress.io/pause) commands to step through your commands.

- Ask other Cypress users for help in our [chat](https://on.cypress.io/chat).

- Try more advanced troubleshooting from [troubleshooting Cypress](https://on.cypress.io/debugging#Troubleshooting-Cypress) doc.

### Fill out our Issue Template

When opening an issue, there is a provided [issue template](./.github/ISSUE\_TEMPLATE.md). Fill out the information according to the template. This is information needed for Cypress to continue forward with your problem. Any issues that do not follow the issue template will be closed.

### Describe Problems

When you file a feature request or bug, we need you to \*\*describe the problem you are facing first\*\*, not just your desired solution.

Often, your problem may have a lot in common with other similar problems. If we understand your use case, we can compare it to other use cases and sometimes find a more powerful or more general solution which solves several problems at once. Understanding the root issue can let us merge and contextualize things. Sometimes there's already a way to solve your problem that might just not be obvious.

Also, your proposed solution may not be compatible with the direction we want to take the product, but we may be able to come up with another solution which has approximately the same effect and does fit into the product direction.

### Reproducibility

\*\*It is nearly impossible for us to resolve many issues if we can not reproduce them. Your best chance of getting a bug looked at quickly is to provide a repository with a reproducible bug that can be cloned and run.\*\*

## Triaging Issues

When an issue is opened in [cypress](https://github.com/cypress-io/cypress), we need to evaluate the issue to determine what steps should be taken next. So, when approaching new issues, there are some steps that should be taken.

### Is this a question?

Some opened issue are questions, not bug reports or feature requests. Issues are reserved for potential bugs or feature requests \*only\*. If this is the case, you should:

- Explain that issues in our GitHub repo are reserved for potential bugs or feature requests and that the issue will be closed since it appears to be neither a bug nor a feature request.

- Guide them to existing resources where their questions can be asked like our [community chat](https://on.cypress.io/chat), our [documentation](https://docs.cypress.io), or [Stack Overflow](https://stackoverflow.com/questions/tagged/cypress).

- Cypress offers support via email when signing up for any of our our [paid plans](https://www.cypress.io/pricing/), so remind them that this is an option. Cypress also offers screen sharing and workshops with our [premium support options](https://www.cypress.io/support/) if they would like something higher-touch.

- Add the `type: question` label to the issue.

- Close the issue.

### Does this issue belong in this repository?

#### Other open source repos

Issues may be opened about wanting changes to our [documentation](https://github.com/cypress-io/cypress-documentation), our [example-kitchensink app](https://github.com/cypress-io/cypress-example-kitchensink), or [another repository](https://github.com/cypress-io). In this case you should:

- Thank them for their contribution.

- Explain that this repo is only for bugs or feature requests of the Cypress product.

- If you have permission to 'Transfer the issue', do so. If not, explain that they can open an issue in our other repository and link to the repository.

- Close the issue (if not already transferred).

#### Our Dashboard Service

Issues may be opened about wanting features in our Dashboard Service. In this case you should:

- Thank them for expressing interest in a new feature.

- Refer them to the Dashboard ProductBoard: "You can express interest and see progress for this feature on our Roadmap from our Dashboard's product board here: https://portal.productboard.com/cypress-io/1-cypress-dashboard All related work for the Dashboard features is handled in that ProductBoard and will be handled by the Dashboard team directly when you comment there."

- Close the issue

- Close the issue to comments

### Is this already an open issue?

Search [all issues](https://github.com/cypress-io/cypress/issues) for keywords from the issue to ensure there isn't already an issue open for this. GitHub has some [search tips](https://help.github.com/articles/searching-issues-and-pull-requests/) that may help you better find the relevant issue.

If an issue already exists you should:

- Thank them for their contribution.

- Explain that this issue if a duplicate of another issue, linking to the relevant issue (`#1234`).

- Add the `type: duplicate` label to the issue.

- Close the issue.

### Does the issue provide all the information from our issue template?

When opening an issue, there is a provided [issue template](./.github/ISSUE\_TEMPLATE.md). If the opened issue does not provide enough information asked from the issue template you should:

- Explain that we require new issues follow our provided [issue template](./.github/ISSUE\_TEMPLATE.md) and that issues that are opened without this information are automatically closed per our [contributing guidelines](#fill-out-our-issue-template).

- Close the issue.

### Are they running the current version of Cypress?

If they listed an older version of Cypress in their issue. We don't want to spend the time to set up a reproducible project (which can be time consuming) only to find that bumping the Cypress version fixes it. You should:

- Ask them to update to the newest version of Cypress and comment about the results.

- Add the `stage: awaiting response` label to the issue.

### Is the fix or feature within our vision for Cypress?

There will inevitably be suggestions that will not fit within the scope of Cypress's vision for our product. If an issue or pull request falls under this category you should:

- Thank them for their contribution.

- Explain why it doesnt fit into the scope of Cypress, and offer clear suggestions for improvement, if youre able. Be kind, but firm.

- Link to relevant documentation, if there is any. If you notice repeated requests for things you dont want to accept, add them into the [documentation](https://github.com/cypress-io/cypress-documentation) to avoid repeating yourself.

- Add the `stage: wontfix` label to the issue.

- Close the issue/pull request

### Is what they are describing actually happening?

The best way to determine the validity of a bug is to recreate it yourself. Follow the directions or information provided to recreate the bug that is described. Did they provide a repository that demonstrates the bug? Great - fork it and run the project and steps required. If they did not provide a repository, the best way to reproduce the issue is to have a 'sandbox' project up and running locally for Cypress. This is just a simple project with Cypress installed where you can freely edit the application under test and the tests themselves to recreate the problem.

\*\*Attempting to recreate the bug will lead to a few scenarios:\*\*

#### 1. You can not recreate the bug

If you cannot recreate the situation happening you should:

- Thank them for their contribution.

- Explain that there is not enough information to reproduce the bug. Provide information on how you went about recreating the scenario, if youre able. Note your OS, Browser, Cypress version and any other information.

- Link them to our contributing guideline for [opening issues](#opening-issues).

- Note that if no reproducible example is provided, we will unfortunately have to close the issue.

- Add the `stage: needs information` label to the issue.

#### 2. You can recreate the bug

If you can recreate the bug you should:

- Thank them for their contribution.

- Explain that you were able to recreate the bug. Provide the exact test code ran and the versions of Cypress, OS, and browser you used to recreate it.

- If you know where the code is that could possibly fix this issue - link to the file or line of code from the [cypress](https://github.com/cypress-io/cypress) repo and remind the user that we are open source and that we gladly accept PRs, even if they are a work in progress.

- Add the `stage: ready for work` label to the issue.

#### 3. You can tell the problem is a user error

In recreating the issue, you may realize that they had a typo or used the Cypress API incorrectly, etc. In this case you should:

- Leave a comment informing the user of their error.

- Link to relevant documentation, if there is any. If you notice repeated user errors for the same situation, add them into the [documentation](https://github.com/cypress-io/cypress-documentation) to avoid repeating yourself.

- Close the issue.

### Has the issue gone stale?

Some issues are opened and sadly forgotten about by the person originally opening the issue.

#### Not enough information ever provided

Sometimes we request more information to be provided (label `stage: needs information`) for an open issue, but no one is able to provide a reproducible example or they simply never respond. \*\*This does not mean that we don't believe that there is a bug!\*\* We just, unfortunately, do not have a path forward to fix it without this information. In this case you should:

- Add a comment reminding them or our request for more information and that the issue will be closed if it is not provided. Sometimes issues get forgotten about, and all the person needs is a gentle reminder.

- If there is still no response after a weeks time, explain that you are closing the issue due to not enough information or inactivity and that they can comment in the issue with a reproducible example and we will reopen the issue.

- Close the issue.

#### They already solved their issue

Some issues are resolved by the community, by giving some guidance or a workaround, but the original opener of the issue forgets to close the issue. In this case you should:

- Explain that you are closing the issue as resolved and that they can comment if they are still having the issue and we will consider reopening it.

- Close the issue.

## Writing Documentation

Cypress documentation lives in a separate repository with its own dependencies and build tools.

See [Documentation Contributing Guideline](https://github.com/cypress-io/cypress-documentation/blob/master/CONTRIBUTING.md).

## Writing code

Working on your first Pull Request? You can learn how from this free series [How to Contribute to an Open Source Project on GitHub](https://egghead.io/courses/how-to-contribute-to-an-open-source-project-on-github)

### What you need to know before getting started

#### Cypress and Packages

Cypress is a large open source project. When you want to contribute to Cypress, you may be unsure which part of the project to work within.

This repository is made up of various packages. They are discrete modules with different responsibilities, but each is necessary for the Cypress app and is not necessarily useful outside of the Cypress app.

Here is a list of the core packages in this repository with a short description, located within the [`packages`](./packages) directory:

| Folder Name | Package Name | Purpose |

| :------------------------------------ | :---------------------- | :--------------------------------------------------------------------------- |

| [cli](./cli) | `cypress` | The command-line tool that is packaged as an `npm` module. |

| [desktop-gui](./packages/desktop-gui) | `@packages/desktop-gui` | The front-end code for the Cypress Desktop GUI. |

| [driver](./packages/driver) | `@packages/driver` | The code that is used to drive the behavior of the API commands. |

| [electron](./packages/electron) | `@packages/electron` | The Cypress implementation of Electron. |

| [example](./packages/example) | `@packages/example` | Our example kitchen-sink application. |

| [extension](./packages/extension) | `@packages/extension` | The Cypress Chrome browser extension |

| [https-proxy](./packages/https-proxy) | `@packages/https-proxy` | This does https proxy for handling http certs and traffic. |

| [launcher](./packages/launcher) | `@packages/launcher` | Finds and launches browsers installed on your system. |

| [reporter](./packages/reporter) | `@packages/reporter` | The reporter shows the running results of the tests (The Command Log UI). |

| [root](./packages/root) | `@packages/root` | Dummy package pointing at the root of the repository. |

| [runner](./packages/runner) | `@packages/runner` | The runner is the minimal "chrome" around the user's application under test. |

| [server](./packages/server) | `@packages/server` | The <3 of Cypress. This orchestrates everything. The backend node process. |

| [socket](./packages/socket) | `@packages/socket` | A wrapper around socket.io to provide common libraries. |

| [static](./packages/static) | `@packages/static` | Serves static assets used in the Cypress GUI. |

| [ts](./packages/ts) | `@packages/ts` | A centralized version of typescript. |

We try to tag all issues with a `pkg/` tag describing the appropriate package the work is required in. For example, the [`pkg/driver`](https://github.com/cypress-io/cypress/labels/pkg%2Fdriver) label is tagged on issues that require work in the `driver` package.

### Requirements

You must have [`node`](https://nodejs.org/en/) and [`yarn`](https://yarnpkg.com/en/docs/install) installed to run the project. Currently, Cypress should be developed using the Node.js version specified in the root [.node-version](.node-version) file. You can use [avn](https://github.com/wbyoung/avn) to automatically use the right version of Node.js for this repo.

### Getting Started

The project utilizes [yarn workspaces](https://yarnpkg.com/lang/en/docs/workspaces/) and leverages [lerna](https://lerna.js.org/) to orchestrate running within the context of one or more packages. While it is not as important to understand lerna or yarn workspaces, it \*\*is important\*\* to note that running scripts or installing new dependencies should always happen from the repo's root directory.

> \*\*? Running on Windows?\*\*

>

> Many of the NPM scripts used during development use commands designed for a Linux-like shell.If you are running a Windows operating system, you may encounter many commands that are not working. To fix this behavior, you have to set a Linux-like shell as the default `npm` script shell. If you have Git for Windows installed, you can set Git Bash as the default script shell by using the following command:

> ```bash

> yarn config set script-shell "C:\\Program Files (x86)\\git\\bin\\bash.exe"

> ```

> Git Bash may be installed in `Program Files`, if so, use the following command:

>```bash

> yarn config set script-shell "C:\\Program Files\\git\\bin\\bash.exe"

>```

\*\*Install all dependencies:\*\*

```bash

yarn

```

This will install all the dependencies for the repo and perform a preliminary build.

\*\*Next, start the app:\*\*

```bash

yarn start

```

If there are errors building the packages, prefix the commands with `DEBUG=cypress:\*` to see more details.

This outputs a lot of debugging lines. To focus on an individual module, run with `DEBUG=cypress:launcher` for instance.

When running `yarn start` this routes through the CLI and eventually calls `yarn dev` with the proper arguments. This enables Cypress day-to-day development to match the logic of the built binary + CLI integration.

If you want to bypass the CLI entirely, you can use the `yarn dev` task and pass arguments directly. For example, to headlessly run a project in a given folder, while trying to record to the Dashboard

```text

yarn dev --run-project /project/folder --record --key <key>

```

#### Adding new Dependencies

?? There is a [bug in yarn](https://github.com/yarnpkg/yarn/issues/7734) that may cause issues adding a new dependency to a workspace. You can avoid this by downgrading yarn to 1.19.1 (temporarily downgrade using `npx yarn@1.19.1 workspace @packages/server add my-new-dep1`).

```shell

# add a new dep to the root of the repo

$ yarn add -W my-new-dep1

# add a new dep to a specific package

$ yarn workspace @packages/server add my-new-dep1

$ yarn workspace @packages/server add --dev my-new-dep1

```

#### Tasks

> Scripts are intended to be \*\*run\*\* from the \*\*root of the repo\*\*. \*\*Do not install dependencies or run scripts from within a sub-directory.\*\*

##### Common Top Level Tasks

By default, top level tasks will execute for all packages. However, most scripts can be provided one or more scopes. Providing a scope will execute tasks within the provided packages. Scope values are based on \*\*package names\*\* and not the directory structure.

| Task | Purpose |

| :----------------- | :--------------------------------------------------------------- |

| `build` | Compile non-node code (coffeescript/typescript) |

| `start` | Open Cypress in dev and global mode |

| `watch` | Auto-rebuild on file changes |

| `clean` | Remove build artifacts |

| `clean-deps` | Remove all installed dependencies (in root and in every package) |

| `test` | Run the default set of tests (may be package dependent) |

| `test-debug` | Run unit/integration tests with inspect node CLI flags |

| `test-unit` | Run unit tests |

| `test-integration` | Run integration tests |

| `test-e2e` | Run end-to-end tests |

| `test-watch` | Run unit tests and rebuild/rerun on file changes |

> Most of the time you will only want to run a task within a specific package; this can be done by providing the package name as a scope to the top level task.

```shell

# Run test-unit only within cypress package (./cli)

$ yarn test-unit --scope cypress

# Run test-unit only within the cypress and server packages (./cli & ./packages/server)

$ yarn test-unit --scope cypress --scope @packages/server

# Run test-unit in all packages with the name starting with `@packages/`

$ yarn test-unit --scope @packages/\*

```

##### Package-Level Scripts

> Although scripts are meant to be run from the root of the repo; they typically delegate to the scripts within the packages.

Each package is responsible for building itself and testing itself and can do so using whatever tools are appropriate, but each conforms to a standard set of scripts so that building, watching, testing, etc. can be orchestrated from the root of this repo. Here are the scripts supported and what they mean:

| Task | Purpose |

| :----------------- | :------------------------------------------------------------------------------------------------------------------------------------------------------- |

| `build` | Build the package |

| `build-prod` | Build all assets for production (if makes sense) |

| `start` | Run a server for serving files |

| `watch` | Watch source files and build development assets when they are saved. This may also run a server for serving files and run tests related to a saved file. |

| `clean` | Remove any assets created by `build-dev` or `build-prod` |

| `clean-deps` | Remove any dependencies installed (usually by `yarn`) |

| `test` | Runs all tests once (this usually means running unit tests; via `yarn test-unit`) |

| `test-unit` | Run all unit tests within the package; `exit 0` if N/A |

| `test-integration` | Run all integration tests within the package; `exit 0` if N/A |

| `test-e2e` | Run all e2e tests within the package; `exit 0` if N/A |

| `test-watch` | Run all unit tests in the package in watch mode |

#### Debugging

Some packages use [debug](https://github.com/visionmedia/debug#readme) to

log debug messages to the console. The naming scheme should be

`cypress:<package name>`; where package name is without the `@packages` scope. For example to see launcher messages during unit

tests start it using

```bash

$ DEBUG=cypress:launcher yarn test --scope @packages/launcher

```

If you want to see log messages from all Cypress projects use wild card

```bash

$ DEBUG=cypress:\*

```

Or for an individual package:

```bash

DEBUG=cypress:cli

DEBUG=cypress:server

DEBUG=cypress:launcher

```

### Coding Style

We use [eslint](https://eslint.org/) to lint all JavaScript code and follow rules specified in

[@cypress/eslint-plugin-dev](https://github.com/cypress-io/eslint-plugin-cypress) plugin.

When you edit files, you can quickly fix all changed files before you commit using

```bash

$ yarn lint-changed --fix

```

When committing files, we run a Git pre-commit hook to lint the staged JS files. See the [`lint-staged` project](https://github.com/okonet/lint-staged).

If this command fails, you may need to run `yarn lint-changed --fix` and commit those changes.

We \*\*DO NOT\*\* use Prettier to format code. You can find [.prettierignore](.prettierignore) file that ignores all files in this repository. To ensure this file is loaded, please always open \_the root repository folder\_ in your text editor, otherwise your code formatter might execute, reformatting lots of source files.

### Adding links within code

When adding links to outside resources within the Cypress Test Runner (including links to Cypress's own documentation), we utilize our [`cypress-on`](https://github.com/cypress-io/cypress-services/tree/develop/packages/on) service for all links.

This is to ensure that links do not go dead in older versions of Cypress when the location of the link has changed. To add a new link:

- Make up a new slug for the linked resource like `https://on.cypress.io/my-special-link`.

- Open a PR adding the new slug in [links.yml](https://github.com/cypress-io/cypress-services/blob/develop/packages/on/data/links.yml) with the href of the resource it should redirect to. \*Note: this requires access to the internal [cypress-services](https://github.com/cypress-io/cypress-services) repo which is only granted to employees. If you're an outside contributor and need a link reroute added, please comment in the relevant PR asking for assistance.\*

- Wait for the PR to be reviewed and \*\*deployed\*\* from [cypress-services](https://github.com/cypress-io/cypress-services). This is required before your changes can be merged into the `cypress` project.

### Tests

For most packages there are typically unit and some integration tests.

Our true e2e tests are in [`packages/server`](packages/server), which test the full stack all together.

Please refer to each packages' `README.md` which documents how to run tests. It is not feasible to try to run all of the tests together. We run our entire test fleet across over a dozen containers in CI.

If you're curious how we manage all of these tests in CI check out our [`circle.yml`](circle.yml) file found in the root `cypress` directory.

#### Docker

Sometimes tests pass locally, but fail in CI. Our CI environment is dockerized. In order to run the image used in CI locally:

1. [Install Docker](https://docs.docker.com/install/) and get it running on your machine.

2. Run the following command from the root of the project:

```shell

$ yarn docker

```

There is a script [scripts/run-docker-local.sh](scripts/run-docker-local.sh) that runs the cypress image (see [circle.yml](circle.yml) for the current image name).

The image will start and will map the root of the repository to `/cypress` inside the image. Now you can modify the files using your favorite environment and rerun tests inside the docker environment.

##### Troubleshooting

Sometimes building inside the image has problems with `node-sass` library. This generally only happens when installing packages locally (in a non-linux environment), and then trying to use these packages in the docker container (linux). The same can happen if packages are installed via the docker container and then trying to run locally; i.e. installed node\_modules were for the docker linux environment, but cypress is running in a non-linux environment.

```text

Error: Missing binding /cypress/packages/desktop-gui/node\_modules/node-sass/vendor/linux-x64-48/binding.node

Node Sass could not find a binding for your current environment: Linux 64-bit with Node.js 6.x

Found bindings for the following environments:

- OS X 64-bit with Node.js 6.x

This usually happens because your environment has changed since running `npm install`.

Run `npm rebuild node-sass` to build the binding for your current environment.

```

In order to resolve this issue, remove all node\_modules via `yarn clean-deps` and then reinstall them in the correct environment via `yarn`. If using/running from a docker container, this is done from within the container; however if running locally, then run from your local shell.

```bash

$ yarn docker

yarn clean-deps

yarn

```

#### Docker for built binary

You can also use Docker to simulate and debug the built binary. In a temporary folder (for example from the folder `/tmp/test-folder/`) start a Docker image:

```shell

$ docker run -it -w /app -v $PWD:/app cypress/base:8 /bin/bash

```

Point the installation at a specific beta binary and NPM package archive (if needed) and \_set local cache folder\_ to unzip the downloaded binary into a subfolder.

```shell

$ export CYPRESS\_INSTALL\_BINARY=https://cdn.cypress.io/beta/.../cypress.zip

$ export CYPRESS\_CACHE\_FOLDER=./cypress-cache

$ yarn add https://cdn.cypress.io/beta/npm/.../cypress.tgz

```

Note that unzipping the Linux binary inside a Docker container onto a mapped volume drive is \*slow\*. But once this is done you can modify the application resource folder in the local folder `/tmp/test-folder/node\_modules/cypress/cypress-cache/3.3.0/Cypress/resources/app` to debug issues.

### Packages

Generally when making contributions, you are typically making them to a small number of packages. Most of your local development work will be inside a single package at a time.

Each package documents how to best work with it, so consult the `README.md` of each package.

They will outline development and test procedures. When in doubt just look at the `scripts` of each `package.json` file. Everything we do at Cypress is contained there.

## Committing Code

### Branches

The repository is setup with two main (protected) branches.

- `master` is the code already published in the last Cypress version.

- `develop` is the current latest "pre-release" code. This branch is set as the default branch, and all pull requests should be made against this branch.

### Pull Requests

- When opening a PR for a specific issue already open, please name the branch you are working on using the convention `issue-[issue number]`. For example, if your PR fixes Issue #803, name your branch `issue-803`. If the PR is a larger issue, you can add more context like `issue-803-new-scrollable-area` If there is not an associated open issue, \*\*create an issue using our [Issue Template](./.github/ISSUE\_TEMPLATE.md)\*\*.

- PR's can be opened before all the work is finished. In fact we encourage this! Please create a [Draft Pull Request](https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests#draft-pull-requests) if your PR is not ready for review. [Mark the PR as \*\*Ready for Review\*\*](https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/changing-the-stage-of-a-pull-request#marking-a-pull-request-as-ready-for-review) when you're ready for a Cypress team member to review the PR.

- Prefix the title of the Pull Request using [semantic-release](https://github.com/semantic-release/semantic-release)'s format as defined [here](https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#type). For example, if your PR is fixing a bug, you should prefix the PR title with `fix:`.

- Fill out the [Pull Request Template](./.github/PULL\_REQUEST\_TEMPLATE.md) completely within the body of the PR. If you feel some areas are not relevant add `N/A` as opposed to deleting those sections. PR's will not be reviewed if this template is not filled in.

- Please check the "Allow edits from maintainers" checkbox when submitting your PR. This will make it easier for the maintainers to make minor adjustments, to help with tests or any other changes we may need.

![Allow edits from maintainers checkbox](https://user-images.githubusercontent.com/1271181/31393427-b3105d44-ada9-11e7-80f2-0dac51e3919e.png)

### Testing

This repository is exhaustively tested by [CircleCI](https://circleci.com/gh/cypress-io/cypress). Additionally we test the code by running it against various other example projects. See CI badges and links at the top of this document.

To run local tests, consult the `README.md` of each package.

### Dependencies

We use [RenovateBot](https://renovatebot.com/) to automatically upgrade our dependencies. The bot uses the settings in [renovate.json](renovate.json) to maintain our [Update Dependencies](https://github.com/cypress-io/cypress/issues/3777) issue and open PRs. You can manually select a package to open a PR from our [Update Dependencies](https://github.com/cypress-io/cypress/issues/3777) issue.

After a PR has been opened for a dependency update, our `cypress-bot` will comment on the PR detailing the guidelines to be used to review the dependency update. Please read these guidelines carefully and make any updates where you see the PR may not be meeting the quality of these guidelines.

## Reviewing Code

### Some rules about Code Review

1. The contributor opening the pull request may not approve their own PR.

2. The PR will not be merged if some reviewers have requested changes.

If any of the Pull Request Review guidelines can't be met, a comment should be left by the reviewer with 'Request changes'. The original contributor is responsible for making any updates and request re-review once those changes are addressed.

### Steps to take during Code Review

- Run the code and use it as the end user would.

- Double check the issue and PR description to ensure it is meeting the original requirements.

- Read through every line of changed code (Yes, we know this could be a LOT).

- If you dont understand why some piece of code is required, ask for clarification! Likely the contributor had a reason and can provide the answer quicker than investigating yourself.

### Code Review Checklist

Below are guidelines to help during code review. If any of the following requirements can't be met, leave a comment in the review selecting 'Request changes', otherwise 'Approve'.

#### User Experience

- [ ] The feature/bugfix is self-documenting from within the product.

- [ ] The change provides the end user with a way to fix their problem (no dead ends).

- [ ] If a breaking change or a change to a commonly used API, the proposed changes have been discussed and agreed upon in the weekly team meeting (or a separate meeting if a larger change).

#### Functionality

- [ ] The code works and performs its intended function with the correct logic.

- [ ] Performance has been factored in (for example, the code cleans up after itself to not cause memory leaks).

- [ ] The code guards against edge cases and invalid input and has tests to cover it.

#### Maintainability

- [ ] The code is readable (too many nested 'if's are a bad sign).

- [ ] Names used for variables, methods, etc, clearly describe their function.

- [ ] The code is easy to understood and there are relevant comments explaining.

- [ ] New algorithms are documented in the code with link(s) to external docs (flowcharts, w3c, chrome, firefox).

- [ ] There are comments containing link(s) to the addressed issue (in tests and code).

#### Quality

- [ ] The change does not reimplement code.

- [ ] There's not a module from the ecosystem that should be used instead.

- [ ] There is no redundant or duplicate code.

- [ ] There are no irrelevant comments left in the code.

- [ ] Tests are testing the codes intended functionality in the best way possible.

#### Internal

- [ ] The original issue has been tagged with a release in ZenHub.

### Code Review of Dependency Updates

Below are some guidelines Cypress uses when reviewing dependency updates.

#### Dependency Update Instructions

- Read through the entire changelog of the dependency's changes. If a changelog is not available, check every commit made to the dependency. \*\*NOTE\*\* - do not rely on semver to indicate breaking changes - every product does not follow this standard.

- Add a PR review comment noting any relevant changes in the dependency.

- If any of the following requirements cannot be met, leave a comment in the review selecting 'Request changes', otherwise 'Approve'.

#### Dependency Updates Checklist

- [ ] Code using the dependency has been updated to accommodate any breaking changes

- [ ] The dependency still supports the version of Node that the package requires.

- [ ] The PR been tagged with a release in ZenHub.

- [ ] Appropriate labels have been added to the PR (for example: label `type: breaking change` if it is a breaking change)

## Deployment

We will try to review and merge pull requests quickly. After merging we will try releasing a new version. If you want to know our build process or build your own Cypress binary, read [DEPLOY.md](./DEPLOY.md)

## Known problems

### ENFILE or EMFILE

If you get `ENFILE: file table overflow`, `ENFILE: too many open files` or any other `ENFILE` or `EMFILE` errors on Mac, that means you are doing synchronous file system operations. Cypress should \*\*NEVER\*\* do them. Instead we should use async file system operations and let `graceful-fs` retry them. Find the place where the synchronous `fs` operation is done from the stacktrace and make it async.