# Questions

This is the issue tracker for Ember.js. The Ember.js community uses this site

to collect and track bugs and discussions of new features. If you are having

difficulties using Ember.js or have a question about usage, please ask a

question on Stack Overflow: http://stackoverflow.com/questions/ask?tags=ember.js, or by joining [the community chat](https://discord.gg/emberjs).

The Ember.js community is very active on Stack Overflow and most questions

receive attention the same day they're posted:

http://stackoverflow.com/questions/tagged/ember.js

# Issue Labeling

Ember uses [StandardIssueLabels](https://github.com/wagenet/StandardIssueLabels) for Github Issues.

# Issues

Think you've found a bug or have a new feature to suggest? Let us know!

## Reporting a Bug

1. Update to the most recent master release if possible. We may have already

fixed your bug.

2. Search for similar issues. It's possible somebody has encountered

this bug already.

3. Provide [Ember Twiddle](http://ember-twiddle.com/) demo that specifically

shows the problem. This demo should be fully operational with the exception

of the bug you want to demonstrate. The more pared down, the better.

If it is not possible to produce a fiddle, please make sure you provide very

specific steps to reproduce the error. If we cannot reproduce it, we will

close the ticket.

4. Your issue will be verified. The provided example will be tested for

correctness. The Ember team will work with you until your issue can

be verified.

5. Keep up to date with feedback from the Ember team on your ticket. Your

ticket may be closed if it becomes stale.

6. If possible, submit a Pull Request with a failing test. Better yet, take

a stab at fixing the bug yourself if you can!

The more information you provide, the easier it is for us to validate that

there is a bug and the faster we'll be able to take action.

### Triaging policy

\* You might be requested to provide a reproduction or extra information. In that

case, the issue will be labeled as \_Needs Submitter Response\_. If we did not

get any response after seven days, we will ping you to remind you about it. We

might close the issue if we do not hear from you after two weeks since the

original notice.

\* If you submit a feature request as an issue, you will be invited to follow the

[instructions in this document](https://github.com/emberjs/ember.js/blob/master/CONTRIBUTING.md#requesting-a-feature)

and the issue will be closed

\* Issues that become inactive will be labeled accordingly

to inform the original poster and Ember contributors that the issue

should be closed since the issue is no longer actionable. The issue

can be reopened at a later time if needed, e.g. becomes actionable again.

\* If possible, issues will be labeled to indicate the status or priority.

For example, labels may have a prefix for `Status: X`, or `Priority: X`.

Statuses may include: `In Progress`, `On Hold`. Priorities may include:

`High` or `Low`.

## Requesting a Feature

1. Ember has an RFC process for feature requests. To begin the discussion either

[gather feedback](https://github.com/emberjs/rfcs/blob/master/README.md#gathering-feedback-before-submitting)

on the emberjs/rfcs repository. Or, draft an [Ember RFC](https://github.com/emberjs/rfcs#ember-rfcs)

- Use RFC pull request for well formed ideas.

- Use RFC issues to propose a rough idea, basically a great place to test

the waters.

2. Provide a clear and detailed explanation of the feature you want and why

it's important to add. Keep in mind that we want features that will be useful

to the majority of our users and not just a small subset. If you're just

targeting a minority of users, consider writing an add-on library for Ember.

3. If the feature is complex, consider writing an Ember RFC document. If we do

end up accepting the feature, the RFC provides the needed documentation for

contributors to develop the feature according the specification accepted by the core team.

4. After discussing the feature you may choose to attempt a Pull Request. If

you're at all able, start writing some code. We always have more work to do

than time to do it. If you can write some code then that will speed the process

along.

In short, if you have an idea that would be nice to have, create an issue on the

emberjs/rfcs repo. If you have a question about requesting a feature, start a

discussion at [discuss.emberjs.com](https://discuss.emberjs.com)

# Building Ember.js

Building Ember.js is a quick process:

```sh

# clone the latest ember.js directory from github

- git clone https://github.com/emberjs/ember.js.git

# cd to the cloned ember.js directory

- cd ember.js

# ensure Node.js and yarn are installed

# build ember.js

- yarn install

- yarn build

```

## Using Custom Builds in an Ember CLI App

While testing custom behavior (maybe that you'd like to write an RFC for...), here's how you'd use a local custom build with an Ember app to test out the custom build:

```sh

# cd to the directory from Building Ember.js (above)

cd ember.js

yarn link

yarn start

# in a new terminal

cd ../your-app-directory/

yarn link ember-source

```

# How to Run Unit Tests

Pull requests should pass the Ember.js unit tests. Do the following to run these tests.

1. Follow the setup steps listed above under [Building Ember.js](#building-emberjs).

2. To start the development server, run `yarn start`.

3. To run all tests, visit <http://localhost:4200/>.

4. To test a specific package, visit `http://localhost:4200/tests/index.html?package=PACKAGE\_NAME`. Replace

`PACKAGE\_NAME` with the name of the package you want to test. For

example:

\* [Ember.js Internals](http://localhost:4200/tests/index.html?package=@ember/-internals)

To test multiple packages, you can separate them with commas.

You can also pass `jquery=VERSION` in the test URL to test different

versions of jQuery. You can also pass `jquery=none` to run tests without jQuery

integration.

## From the CLI

Run `yarn test` to run a basic test suite or run `TEST\_SUITE=all yarn test` to

run a more comprehensive suite.

## From ember-cli

1. `ember test --server`

2. Connect the browsers you want.

To run a specific browser, you can use the `--launch` flag

\* `ember test --server --launch SL\_Firefox\_Current`

\* `ember test --launch SL\_Firefox\_Current`

\* `ember test --launch SL\_Firefox\_Current,Chrome`

To test multiple launchers, you can separate them with commas.

# Pull Requests

We love pull requests. Here's a quick guide:

1. Fork the repo.

2. Run the tests. We only take pull requests with passing tests, and it's great

to know that you have a clean slate: `yarn install && yarn test`.

(To see tests in the browser, run `yarn start` and open `http://localhost:4200/tests/index.html`.)

3. Add a test for your change. Only refactoring and documentation changes

require no new tests. If you are adding functionality or fixing a bug, we need

a test! If your change is a new feature, please

[wrap it in a feature flag](https://guides.emberjs.com/release/contributing/adding-new-features/).

4. Make sure to check out the

[JavaScript Style Guide](https://github.com/emberjs/ember.js/blob/master/STYLEGUIDE.md) and

ensure that your code complies with the rules. If you missed a rule or two, don't worry, our

tests will warn you.

5. Make the test pass.

6. Commit your changes. Please use an appropriate commit prefix.

If your pull request fixes an issue specify it in the commit message. Some examples:

```

[DOC beta] Update CONTRIBUTING.md for commit prefixes

[FEATURE query-params-new] Message

[BUGFIX beta] Message

[SECURITY CVE-111-1111] Message

```

For more information about commit prefixes see [the appendix](#commit-tagging).

7. Push to your fork and submit a pull request. Please provide us with some

explanation of why you made the changes you made. For new features make sure to

explain a standard use case to us.

We try to be quick about responding to tickets but sometimes we get a bit

backlogged. If the response is slow, try to find someone on IRC (#emberjs) to

give the ticket a review.

Some things that will increase the chance that your pull request is accepted,

taken straight from the Ruby on Rails guide:

\* Use Ember idioms and helpers

\* Include tests that fail without your code, and pass with it

\* Update the documentation, the surrounding one, examples elsewhere, guides,

whatever is affected by your contribution

Syntax:

\* Two spaces, no tabs.

\* No trailing whitespace. Blank lines should not have any space.

\* a = b and not a=b.

\* Follow the conventions you see used in the source already.

Inline Documentation Guidelines:

All inline documentation is written using YUIDoc. Follow these rules when

updating or writing new documentation:

1. All code blocks must be fenced

2. All code blocks must have a language declared

3. All code blocks must be valid code for syntax highlighting

4. All examples in code blocks must be aligned

5. Use two spaces between the code and the example: `foo(); // result`

6. All references to code words must be enclosed in backticks

7. Prefer a single space between sentences

8. Reference Ember.js as Ember.

9. Wrap long markdown blocks > 80 characters

10. Don't include blank lines after `@param` definitions

Code words are:

\* `thisPropertyName`

\* `Global.Class.attribute`

\* `thisFunction()`

\* `Global.CONSTANT\_NAME`

\* `true`, `false`, `null`, `undefined` (when referring to programming values)

\* references to other properties/methods

And in case we didn't emphasize it enough: we love tests!

NOTE: Partially copied from https://raw.github.com/thoughtbot/factory\_girl\_rails/master/CONTRIBUTING.md

# CI (Github Actions) Tests

We use [GitHub Actions](https://github.com/emberjs/ember.js/actions?query=workflow%3ABuild+event%3Apull\_request) to test each PR before it is merged.

When you submit your PR (or later change that code), a CI build will automatically be kicked off. A note will be added to the PR, and will indicate the current status of the build.

Within the CI build, you can see that we (currently) run six different test suites.

\* The `each-package` test suite is closest to what you normally run locally on your machine.

\* The `build-tests EMBER\_ENV=production...` test suite runs tests against a production build.

\* The `browserstack` test suite runs tests against various supported browsers.

## Common CI Build Issues

### Production Build Failures

If your build is failing on the 'production' suite, you may be relying on a debug-only function that does not even exist in a production build (`Ember.warn`, `Ember.deprecate`, `Ember.assert`, etc.). These will pass on the 'each-package' suite (and locally) because those functions are present in development builds.

There are helpers for many of these functions, which will resolve this for you: `expectDeprecation`, `expectAssertion`, etc. Please use these helpers when dealing with these functions.

If your tests can't aren't covered a helper, one common solution is the use of `DEBUG` flag. Wrapping the debug-only dependent test in a check of this flag will cause that test to not be run in the prod test suite:

```javascript

import { DEBUG } from '@glimmer/env';

if (DEBUG) {

// Development-only test goes here

}

```

Note: before using this approach, please be certain your test is really depending on a debug-only function and not truly failing in production.

To recreate this build environment locally:

\* Run `ember serve --environment=production` in a terminal (takes much much longer than a default `ember s`)

\* Browse to `localhost:4200/tests/index.html?dist=prod&prod=true`

### Single Unexplained Test Suite Failure

Sometimes a single test suite will fail, without giving any indication of a real error.

\* Try to recreate the test environment locally (see above for production builds)

\* Restart all the test suites on CI by doing another push

\* Ask a repo collab to restart that single test suite

# Appendix

## Commit Tagging

All commits should be tagged. Tags are denoted by square brackets (`[]`) and come at the start of the commit message.

### Bug Fixes

In general bug fixes are pulled into the beta branch. As such, the prefix is: `[BUGFIX beta]`. If a bug fix is a serious regression that requires a new patch release, `[BUGFIX release]` can be used instead.

For bugs related to canary features, follow the prefixing rules for features.

The vast majority of bug fixes apply to the current stable or beta releases, so submit your PR against `emberjs:master` with one of the above mentioned BUGFIX tags. (In the unusual case of a bug fix specifically for a past release, tag for that release `[BUGFIX release-1-13]` and submit the PR against the stable branch for that release: `emberjs:stable-1-13`.)

### Cleanup

Cleanup commits are for removing deprecated functionality and should be tagged

as `[CLEANUP beta]`.

### Features

All additions and fixes for features in canary should be tagged as `[FEATURE name]` where name is the same as the flag for that feature.

### Documentation

Documentation commits are tagged as `[DOC channel]` where channel is `canary`,

`beta`, or `release`. If no release is provided `canary` is assumed. The channel should be the most stable release that this documentation change applies to.

### Security

Security commits will be tagged as `[SECURITY cve]`. Please do not submit security related PRs without coordinating with the security team. See the [Security Policy](https://emberjs.com/security/) for more information.

### Other

In general almost all commits should fall into one of these categories. In the cases where they don't please submit your PR untagged. An Ember contributor will let you know if tagging is required.