Oh, hello there! You're probably reading this because you are interested in

contributing to nteract. That's great to hear! This document will help you

through your journey of open source. Love it, cherish it, take it out to

dinner, but most importantly: read it thoroughly!

## What do I need to know to help?

### The JavaScript/TypeScript

You'll need knowledge of JavaScript (ES6), React, RxJS, Redux, and TypeScript to

help out with this project. That's a whole lot of cool stuff! But don't worry,

we've got some resources to help you out.

- [Building a voting app with Redux and React](http://teropa.info/blog/2015/09/10/full-stack-redux-tutorial.html)

- [Introduction to Reactive Programming](https://gist.github.com/staltz/868e7e9bc2a7b8c1f754)

- [Examples, Explanations, and Resources for RxJS 5](https://github.com/btroncone/learn-rxjs)

- [TypeScript in 5 Minutes](https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html)

### Jupyter and ZeroMQ (Optional)

While not a strict pre-requisite, familiarity with the protocol that Jupyter

provides for creating rich notebooks like nteract (and other consoles/REPLs) is

advised to understand the overall system.

- [Jupyter Messaging](http://jupyter-client.readthedocs.org/en/latest/messaging.html)

- [ZeroMQ](http://zguide.zeromq.org/page:all)

If you want a gentle guide to Rx + Jupyter messaging at the same time, we have

a [build your own REPL with enchannel](https://github.com/nteract/docs-old/blob/master/enchannel/build-your-own-repl.md)

tutorial. This allows you to work without React while learning concepts,

leading to implementing a light version of [ick](https://github.com/nteract/ick),

an interactive console.

## What NodeJS version do I need?

It's best to use the latest LTS version of [NodeJS](https://nodejs.org) (currently: ![unknown to this

user agent. Please check the NodeJS website linked previously](https://img.shields.io/npm/v/node/lts.svg?label=)).

Some dependencies can break when using newer releases.

To manage multiple versions of NodeJS, take a look at the [n Package](https://www.npmjs.com/package/n). You

can switch to the latest LTS version using:

```

$ n lts

```

## How do I make a contribution?

Never made an open source contribution before? Wondering how contributions work

in the nteract world? Here's a quick rundown!

1. Find an issue that you are interested in addressing or a feature that you would like to address.

2. Fork the repository associated with the issue to your local GitHub organization.

3. Install the `ipykernel` on your machine using the instructions outlined [on the kernels guide](https://nteract.io/kernels).

4. Clone the repository to your local machine using:

```

$ git clone https://github.com/github-username/repository-name.git

```

5. `cd` into your new clone and run the following scripts:

- `yarn install` or just `yarn` to install dependencies

- `yarn build:all` to compile typescript

6. Create a new branch for your fix using:

```

$ git checkout -b branch-name-here

```

7. Make the appropriate changes for the issue you are trying to address or the feature that you want to add. Validate your changes by following the steps in the "How do I validate my changes" segment below.

8. Confirm that unit tests still pass successfully with:

```

$ yarn test

```

If tests fail, don't hesitate to ask for help.

9. Add and commit the changed files using `git add` and `git commit`.

10. Push the changes to the remote repository using:

```

$ git push origin branch-name-here

```

11. Submit a pull request to the upstream repository.

12. Title the pull request per the requirements outlined in the section below.

13. Set the description of the pull request with a brief description of what you did and any questions you might have about what you did.

14. Wait for the pull request to be reviewed by a maintainer.

15. Make changes to the pull request if the reviewing maintainer recommends them.

16. Celebrate your success after your pull request is merged! :tada:

## How do I validate my changes to nteract?

### Validating in the nteract desktop app

In addition to writing tests, you will want to validate your changes by testing them in the nteract desktop and web applications. These applications are a great test harness for manual tests and QA. To set up the nteract desktop application for development, follow the steps below.

1. Ensure that you have installed the monorepo dependencies by running `yarn` in the root directory.

2. Open a terminal window and run the following command. This will start a progressive build of the nteract desktop app. Whenever you make a code change, the build will automagically update.

```

$ yarn build:desktop:watch

```

3. In a separate window, run the following command. This will spawn the Electron app running the desktop app.

```

$ yarn spawn

```

You will need to reload the nteract page to fetch the latest changes from the build. You can do so by clicking `View > Reload` in the menu.

\*\*Helpful Tip:\*\* You can enter the debugger in the Electron app by placing a `debugger;` statement in the desired location in your source code.

### Validating in the nteract web app

The nteract web app is a great place to validate changes made to the nteract source that affect interactions with a Jupyter server. To setup the nteract web app for development, please follow the instructions in [its README](./applications/jupyter-extension).

## How should I write my commit messages and PR titles?

Good commit messages serve at least three important purposes:

- To speed up the reviewing process.

- To help us write a good release note.

- To help the future maintainers of nteract/nteract (it could be you!), say

five years into the future, to find out why a particular change was made to

the code or why a specific feature was added.

Structure your commit message like this:

```

> Short (50 chars or less) summary of changes

>

> More detailed explanatory text, if necessary. Wrap it to about 72

> characters or so. In some contexts, the first line is treated as the

> subject of an email and the rest of the text as the body. The blank

> line separating the summary from the body is critical (unless you omit

> the body entirely); tools like rebase can get confused if you run the

> two together.

>

> Further paragraphs come after blank lines.

>

> - Bullet points are okay, too

>

> - Typically a hyphen or asterisk is used for the bullet, preceded by a

> single space, with blank lines in between, but conventions vary here

>

```

\_Source: http://git-scm.com/book/ch5-2.html\_

### DO

- Write the summary line and description of what you have done in the

imperative mode, that is as if you were commanding. Start the line

with "Fix", "Add", "Change" instead of "Fixed", "Added", "Changed".

- Always leave the second line blank.

- Line break the commit message (to make the commit message readable

without having to scroll horizontally in gitk).

### DON'T

- Don't end the summary line with a period.

### Tips

- If it seems difficult to summarize what your commit does, it may be because it

includes several logical changes or bug fixes, and are better split up into

several commits using `git add -p`.

### References

The following blog post has a nice discussion of commit messages:

- "On commit messages" http://who-t.blogspot.com/2009/12/on-commit-messages.html

## How fast will my PR be merged?

Your pull request will be merged as soon as there are maintainers to review it

and after tests have passed. You might have to make some changes before your

PR is merged but as long as you adhere to the steps above and try your best,

you should have no problem getting your PR merged.

That's it! You're good to go!