Contributing to Create React App

Loving Create React App and want to get involved? Thanks! There are plenty of ways you can help.

Please take a moment to review this document in order to make the contribution process straightforward and effective for everyone involved.

Following these guidelines helps to communicate that you respect the time of the developers managing and developing this open source project. In return, they should reciprocate that respect in addressing your issue or assessing patches and features.

Core Ideas

As much as possible, we try to avoid adding configuration and flags. The purpose of this tool is to provide the best experience for people getting started with React, and this will always be our first priority. This means that sometimes we sacrifice additional functionality (such as server rendering) because it is too hard to solve it in a way that wouldn't require any configuration.

We prefer **convention, heuristics, or interactivity** over configuration. Here are a few examples of them in action.

Convention

Instead of letting the user specify the entry filename, we always assume it to be `src/index.js`. Rather than letting the user specify the output bundle name, we generate it, but make sure to include the content hash in it. Whenever possible, we want to leverage convention to make good choices for the user, especially in cases where it's easy to misconfigure something.

Heuristics

Normally, `npm start` runs on port `3000`, and this is not explicitly configurable. However, some environments like cloud IDEs want the programs to run on a specific port to serve their output. We want to play well with different environments, so Create React App reads `PORT` environment variable and prefers it when it is specified. The trick is that we know cloud IDEs already specify it automatically, so there is no need for the user to do anything. Create React App relies on heuristics to do the right thing depending on environment.

Another example of this is how `npm test` normally launches the watcher, but if the `CI` environment variable is set, it will run tests once. We know that popular CI environments set this variable, so the user doesn't need to do anything. It just works.

Interactivity

We prefer to add interactivity to the command line interface rather than add configuration flags. For example, `npm start` will attempt to run with port `3000` by default, but it may be busy. Many other tools fail in this case and ask that you pass a different port, but Create React App will display a prompt asking if you'd like to run the app on the next available port.

Another example of interactivity is `npm test` watcher interface. Instead of asking people to pass command line flags for switching between test runner modes or search patterns, we print a hint with keys that you can press during the test session to instruct watcher what to do. Jest supports both flags and interactive CLI but Create React App prefers long-running sessions to keep user immersed in the flow over short-running sessions with different flags.

Breaking the Rules

No rules are perfect. Sometimes we may introduce flags or configuration if we believe the value is high enough to justify the complexity. For example, we know that apps may be hosted paths different from the root, and we need to support this use case. However, we still try to fall back to heuristics when possible. In this example, we ask that you specify `homepage` in `package.json`, and infer the correct path based on it. We also nudge the user to fill out the `homepage` after the build, so the user becomes aware that the feature exists.

Submitting a Pull Request

Good pull requests, such as patches, improvements, and new features, are a fantastic help. They should remain focused in scope and avoid containing unrelated commits.

Please **ask first** if somebody else is already working on this or the core developers think your feature is in-scope for Create React App. Generally always have a related issue with discussions for whatever you are including.

Please also provide a **test plan**, i.e. specify how you verified that your addition works.

Folder Structure of Create React App

`create-react-app` is a monorepo, meaning it is divided into independent sub-packages. These packages can be found in the `packages/` directory.

Overview of directory structure

```
packages/
  babel-plugin-named-asset-import/
  babel-preset-react-app/
```

```
confusing-browser-globals/
cra-template/
cra-template-typescript/
create-react-app/
eslint-config-react-app/
react-app-polyfill/
react-dev-utils/
react-error-overlay/
react-scripts/
```

Package Descriptions

babel-preset-react-app This package is a babel preset intended to be used with `react-scripts`. It targets platforms that React is designed to support (IE 11+) and enables experimental features used heavily at Facebook. This package is enabled by default for all `create-react-app` scaffolded applications.

create-react-app The global CLI command code can be found in this directory, and shouldn't often be changed. It should run on Node 0.10+.

eslint-config-react-app This package contains a conservative set of rules focused on making errors apparent and enforces no style rules. This package is enabled by default for all `create-react-app` scaffolded applications.

react-dev-utils This package contains utilities used for `react-scripts` and sibling packages. Its main purpose is to conceal code which the user shouldn't be burdened with upon ejecting.

react-scripts This package is the heart of the project, which contains the scripts for setting up the development server, building production builds, configuring all software used, etc. All functionality must be retained (and configuration given to the user) if they choose to eject.

Setting Up a Local Copy

You will need `npm@7` and `yarn@1` in order to bootstrap and test a local copy of this repo.

1. Clone the repo with `git clone https://github.com/facebook/create-react-app`
2. Run `npm install` in the root `create-react-app` folder.

Once it is done, you can modify any file locally and run `npm start`, `npm test` or `npm run build` like you can in a generated project. It will serve the application from the files located in `packages/cra-template/template`.

If you want to try out the end-to-end flow with the global CLI, you can do this too:

```
npx create-react-app my-app
cd my-app
```

and then run `npm start` or `npm run build`.

Contributing to E2E (end to end) tests

TL;DR use the command `npm run e2e:docker` to run unit and e2e tests.

More detailed information are in the dedicated README.

CI testing with private packages

create-react-app relies on main registry to fetch all dependencies, but, if you are in the need to usage of custom private packages that need to be fetch while running E2E test you might need a different configuration.

Customizing E2E registry configuration We use verdaccio to emulate packages publishing in a registry using a default configuration. You might modify the current behaviour by editing the file `task/verdaccio.yaml`.

For more information about the configuration check out the Verdaccio documentation.

Tips for contributors using Windows

The scripts in tasks folder and other scripts in `package.json` will not work in Windows out of the box. However, using Bash on windows makes it easier to use those scripts without any workarounds. The steps to do so are detailed below:

Install Bash on Ubuntu on Windows

A good step by step guide can be found here

Install Node.js and yarn

Even if you have node and yarn installed on your windows, it would not be accessible from the bash shell. You would have to install it again. Installing via `nvm` is recommended.

Line endings

By default git would use `CRLF` line endings which would cause the scripts to fail. You can change it for this repo only by setting `autocrlf` to false by running `git config core.autocrlf false`. You can also enable it for all your repos by using the `--global` flag if you wish to do so.

Cutting a Release

1. Tag all merged pull requests that go into the release with the relevant milestone. Each merged PR should also be labeled with one of the labels named `tag: ...` to indicate what kind of change it is. **Make sure all breaking changes are correctly labelled with `tag: breaking change`.**
2. Close the milestone and create a new one for the next release.
3. In most releases, only `react-scripts` needs to be released. If you don't have any changes to the `packages/create-react-app` folder, you don't need to bump its version or publish it (the publish script will publish only changed packages).
4. Note that files in `packages/create-react-app` should be modified with extreme caution. Since it's a global CLI, any version of `create-react-app` (global CLI) including very old ones should work with the latest version of `react-scripts`.
5. Pull the latest changes from GitHub, run `npm ci`.
6. Create a change log entry for the release:

- You'll need an access token for the GitHub API. Save it to this environment variable: `export GITHUB_AUTH="..."`
- Run `npm run changelog`. The command will find all the labeled pull requests merged since the last release and group them by the label and affected packages, and create a change log entry with all the changes and links to PRs and their authors. Copy and paste it to `CHANGELOG.md`.
- Add a four-space indented paragraph after each non-trivial list item, explaining what changed and why. For each breaking change also write who it affects and instructions for migrating existing code.
- Maybe add some newlines here and there. Preview the result on GitHub to get a feel for it. Changelog generator output is a bit too terse for my taste, so try to make it visually pleasing and well grouped.

7. Make sure to include "Migrating from ..." instructions for the previous release. Often you can copy and paste them.
8. Run `npm run publish`. (It has to be `npm run publish` exactly, not `npm publish` or `yarn publish`.)
9. Wait for a long time, and it will get published. Don't worry that it's stuck. In the end the publish script will prompt for versions before publishing the packages.
10. After publishing, create a GitHub Release with the same text as the changelog entry. See previous Releases for inspiration.

Make sure to test the released version! If you want to be extra careful, you can publish a prerelease by running `npm run publish -- --canary --exact --preid next --dist-tag=next --force-publish=* minor` instead of `npm run publish`.

Releasing the Docs

1. Go to the `docusaurus/website` directory
2. Run `npm ci`
3. Run `npm run build`
4. You'll need an access token for the GitHub API. Save it to this environment variable: `export GITHUB_AUTH="..."`
5. Run `GIT_USER=<GITHUB_USERNAME> CURRENT_BRANCH=main USE_SSH=true npm run deploy`

Many thanks to h5bp for the inspiration with this contributing guide