# Development

Thanks for contributing! We want to ensure that `urql` evolves and fulfills

its idea of extensibility and flexibility by seeing continuous improvements

and enhancements, no matter how small or big they might be.

If you're about to add a new exchange, please consider publishing it as

a separate package.

## How to contribute?

We follow fairly standard but lenient rules around pull requests and issues.

Please pick a title that describes your change briefly, optionally in the imperative

mood if possible.

If you have an idea for a feature or want to fix a bug, consider opening an issue

first. We're also happy to discuss and help you open a PR and get your changes

in!

## How do I set up the project?

Luckily it's not hard to get started. You can install dependencies using yarn.

Please don't use `npm` to respect the lockfile.

```sh

yarn

```

There are multiple commands you can run in the root folder to test your changes:

```sh

# TypeScript checks:

yarn run check

# Linting (prettier & eslint):

yarn run lint

# Jest Tests (for all packages):

yarn run test

# Builds (for all packages):

yarn run build

```

You can find the main packages in `packages/\*` and the addon exchanges in `exchanges/\*`.

Each package also has its own scripts that are common and shared between all packages.

```sh

# Jest Tests for the current package:

yarn run test

# Linting (prettier & eslint):

yarn run lint

# Builds for the current package:

yarn run build

# TypeScript checks for the current package:

yarn run check

```

While you can run `build` globally in the interest of time it's advisable to only run it

on the packages you're working on. Note that TypeScript checks don't require any packages

to be built.

## How do I test my changes?

It's always good practice to run the tests when making changes. If you're unsure which packages

may be affected by your new tests or changes you may run `yarn test --watch` in the root of

the repository.

If your editor is not set up with type checks you may also want to run `yarn run check` on your

changes.

Additionally you can head to any example in the `examples/` folder

and run them. There you'll also need to run `yarn` to install their

dependencies. All examples are started using `yarn start`.

## How do I lint my code?

We ensure consistency in `urql`'s codebase using `eslint` and `prettier`.

They are run on a `precommit` hook, so if something's off they'll try

to automatically fix up your code, or display an error.

If you have them set up in your editor, even better!

## How do I document a change for the changelog?

This project uses [changesets](https://github.com/atlassian/changesets). This means that for

every PR there must be documentation for what has been changed and which package is affected.

You can document a change by running `yarn changeset`, which will ask you which packages

have changed and whether the change is major/minor/patch. It will then ask you to write

a change entry as markdown.

This will eventually end up in the package's `CHANGELOG.md` file when we do a release.

[Read more about adding a `changeset` here.](https://github.com/atlassian/changesets/blob/master/docs/adding-a-changeset.md#i-am-in-a-multi-package-repository-a-mono-repo)

## How do I upgrade all dependencies?

It may be a good idea to keep all dependencies on the `urql` repository up-to-date every now and

then. Typically we do this by running `yarn upgrade-interactive --latest` and checking one-by-one

which dependencies will need to be bumped. In case of any security issues it may make sense to

just run `yarn upgrade [package]`.

Afterwards `yarn` may accidentally introduce duplicate packages due to some transitive dependencies

having been upgraded separately. This can be fixed by running:

```sh

npx yarn-deduplicate yarn.lock

yarn

```

## How do I add a new package?

First of all we need to know where to put the package.

- Exchanges should be added to `exchanges/` and the folder should be the plain

name of the exchange. Since the `package.json:name` is following the convention

of `@urql/exchange-\*` the folder should just be without this conventional prefix.

- All other packages should be added to `packages/`. Typically all packages should

be named `@urql/\*` and their folders should be named exactly this without the

prefix or `\*-urql`. Optionally if the package will be named `\*-urql` then the folder

can take on the same name.

When adding a new package, start by copying a `package.json` file from another project.

You may want to alter the following fields first:

- `name`

- `version` (either start at `0.1.0` or `1.0.0`)

- `description`

- `repository.directory`

- `keywords`

Make sure to also alter the `devDependencies`, `peerDependencies`, and `dependencies` to match

the new package's needs.

\*\*The `main` and `module` fields follow a convention:\*\*

All output bundles will always be output in the `./dist` folder by `rollup`, which is set up in

the `build` script. Their filenames are a "kebab case" (dash-cased) version of the `name` field with

an appropriate extension (`.esm.js` for `module` and `.cjs.js` for `main`).

If your entrypoint won't be at `src/index.ts` you may alter it. But the `types` field has to match

the same file relative to the `dist/types` folder, where `rollup` will output the TypeScript

declaration files.

When setting up your package make sure to create a `src/index.ts` file

(or any other file which you've pointed `package.json:source` to). Also don't forget to

copy over the `tsconfig.json` from another package (You won't need to change it).

The `scripts.prepare` task is set up to check your new `package.json` file for correctness. So in

case you get anything wrong, you'll get a short error when running `yarn` after setting your new

project up. Just in case! ?

Afterwards you can check whether everything is working correctly by running:

```sh

yarn

yarn run check

```

## When and how do I add a changeset?

Our changelogs and releases are maintained using `changeset`. For each PR that actively changes the

behaviour of one or more packages in our monorepo it allows you to log the change and classify it

per package as either `patch`, `minor`, or `major`.

You won't need to add a changeset if you're simply making "non-visible" changes to the docs or other

files that aren't published to `npm`.

If you are however making visible changes you can track a change by calling the `changeset` CLI.

```sh

# In the root of the urql repository call:

yarn changeset

```

This will open an interactive CLI that asks you to track your changes and enter a change message.

You should always add to the message, what you've changed (not which package if it's only for a

single package though) and what the impact is. If it's a breaking change it should also include some

instructions on how users should update their code.

This will create a new "changeset file" in the `.changeset` folder, which you should commit and

push, so that it's added to your PR.

## How do I release new versions of our packages?

The process of releasing versions is automated using `changeset`. This moves a lot of

the work of writing CHANGELOG entries away from our release process and to our PR

review process. During the release the created `changeset` entries are automatically

applied and our `CHANGELOG`s are updated.

First check what changes you're about to release with `yarn changeset status`.

Then the process is similar to using `yarn version` and `yarn publish`.

First we'll run `yarn changeset version` to bump all packages' versions and update CHANGELOG files:

```sh

# This will automatically bump versions as necessary and update CHANGELOG files:

yarn changeset version

```

> \*\*Note:\*\* This command requires you to create a [GitHub Personal Access Token](https://github.com/settings/tokens/new)

> and have it set on the `GITHUB\_TOKEN` environment variable, either globally, e.g. in your

> `~/.profile` file, or locally in a `.env` file.

Then verify that the updated `package.json` and `CHANGELOG.md` files look correct and commit the

changes:

```sh

# Commit all updated files

git commit -a -m 'Version Packages'

```

At this point we have a new commit with packages that have been updated and should be published.

Please don't push this commit before publishing so that we keep `master` in a pre-release state, in

case the publish actually fails!

```

# Then publish all new packages / versions:

yarn changeset publish

# And push the "Version Packages" commit and all tags afterwards:

git push && git push --tags

```

Publishing the packages will also create the tags, it won't however push them automatically. After

you've pushed the new tags, please make sure to update releases with the new content in each

`CHANGELOG.md`, like so: https://github.com/FormidableLabs/urql/releases/tag/%40urql%2Fexchange-graphcache%402.2.2