# Contributing

The PostGraphile project welcomes all contributors, we want to invest in you so

you can invest back into PostGraphile. Together we can make great software that

enables developers to build powerful applications in much less time then would

previously have been taken. This document aims to help you get started

contributing to the project.

## Development environment

First of all, our development environment is focussed around Unix tools. If

you are on Windows, you may need to use something like Docker or a VM to help

you develop.

Since PostGraphile is mostly powered by Graphile Engine the best way to

develop it is to do it in the context of a Graphile Engine build so that you

may dig into the depths if you need to.

### Yarn

We use yarn workspaces, so it's very important that you use the `yarn`

package manager rather an npm whilst developing PostGraphile.

```

npm install -g yarn

```

Before we start, check that you're running up to date versions of the relevant

tools, you should be running at least:

```

$ node --version

v10.13.0

$ yarn --version

1.12.1

$ git --version

git version 2.18.0

$ watchman --version # optional

4.9.0

```

(If you don't have `watchman` that's fine, but it helps when watching large

numbers of files. On Mac you can install it using homebrew:

`brew install watchman`)

To get started, we're going to check out `graphile-engine`, and then we're

going to check out `postgraphile` \_inside of\_ `graphile-engine`. With a few

tweaks this will enable `postgraphile` to use the development version of the

modules in `graphile-engine` without needing to perform `yarn link` etc.

It should be safe to copy and paste these commands into your terminal (one at

a time), but if you're using zsh you may want to run this first so that comments are ignored:

```

set -k

```

### Setup

Okay, here's the setup:

```bash

# Clone graphile-engine:

git clone git@github.com:graphile/graphile-engine.git graphile-engine

# Clone postgraphile \_inside of\_ graphile-engine:

git clone git@github.com:graphile/postgraphile.git graphile-engine/postgraphile

########################################

# Initial setup for graphile-engine

cd graphile-engine

# Install deps:

yarn

# Monorepo stuff:

yarn lerna bootstrap

# Run initial build:

yarn prepack:all

########################################

# Initial setup for postgraphile

cd postgraphile

# Install deps:

yarn

# Remove deps that should be served by the parent graphile-engine:

./rmlocal.sh

# Builds GraphiQL, images, etc so they can be require()d:

yarn make-assets

```

### Database setup

For testing, PostGraphile of course requires a Postgres database. [Download and install](https://www.postgresql.org/download/) it, e.g. with:

```

sudo apt-get update

sudo apt-get install postgresql postgresql-contrib postgresql-client-common

```

Then we need to set up a superuser (who can install extensions) and create test databases for the individual test suites.

You can use the [command line utilities](https://www.postgresql.org/docs/11/reference-client.html) or go with `psql`:

```

sudo -u postgres psql

CREATE ROLE me WITH LOGIN SUPERUSER; -- PASSWORD 'mypassword'

SET ROLE me;

CREATE DATABASE postgraphile\_test;

COMMENT ON DATABASE postgraphile\_test IS 'https://github.com/graphile/postgraphile';

CREATE DATABASE graphileengine\_test;

COMMENT ON DATABASE graphileengine\_test IS 'https://github.com/graphile/graphile-engine';

```

You may use the default `postgres` superuser instead of your own one if you prefer.

Using your OS login for naming the user `me` requires fewer customisations down the road, as on many systems it will be the default.

The [`lds` package](./lds/README.md) uses another database `lds\_test`, this should be created automatically by its `yarn db:init` script.

To allow login without a password (please evaluate your own security requirements), you may change the [`pg\_hba.conf`](https://www.postgresql.org/docs/11/auth-pg-hba-conf.html) so that the user is always accepted from the local machine:

```

# for postgraphile testing

local postgraphile\_test,graphileengine\_test,lds\_test all trust

host postgraphile\_test,graphileengine\_test,lds\_test me 127.0.0.1/32 trust

host postgraphile\_test,graphileengine\_test,lds\_test me ::1/128 trust

```

You don't need to install Postgres locally, running it on an external server or [in a Docker container](https://github.com/graphile/graphile-engine/wiki/Development-with-docker-databases) works as well.

See below for how to make your database connections known to the test runner.

## Developing

### Graphile Engine

Graphile Engine is built in a mixture of TypeScript and Flow (we're slowly

migrating to TypeScript); which means it has a compile step. For developer

productivity we don't incur this compilation cost every time we run the

development version of PostGraphile, instead we require the compiled code in

most places. This means that we must watch the source code for changes so

that when we run the compiled code it's not out of date. To do this, in the

root of graphile-engine, run `yarn watch`:

```

yarn watch

```

This will compile everything, and then monitor for changes and compile just

the changed files. Every time a file is compiled it will be listed in the

output - \*\*be careful to check for errors\*\*!

You should leave this watch process running, so open another terminal to do

further work.

### PostGraphile

First, change into the postgraphile directory (`graphile-engine/postgraphile/`).

To run PostGraphile in development, you can use the `scripts/dev` command.

This command emulates the `postgraphile` command, and it has two modes. If

you want the `scripts/dev` command to exit when `postgraphile` would, add two

hyphens, e.g.

```bash

# Will run once and exit on error or `-X` option

scripts/dev -- -c postgres://localhost/my\_db -s my\_schema --watch --enhance-graphiql

```

The `--` makes `scripts/dev` act more like a regular build - i.e. exiting on

the `-X` commands or when an error occurs.

If, however, you want `scripts/dev` to automatically restart whenever you

change the PostGraphile or Graphile Engine source code, run it without the

`--`. To manually restart the server type in `rs` and hit enter while

`scripts/dev` is running.

```bash

# Will monitor the source for changes and restart automatically

scripts/dev -c postgres://localhost/my\_db -s my\_schema --watch --enhance-graphiql

```

## Updating or changing branches

Due to this peculiar setup, when you want to update or switch branches,

you have to do an extra couple steps than you might expect.

### Graphile Engine

(In the `graphile-engine` folder.)

Thanks to yarn workspaces, updating the `graphile-engine` folder

is straightforward:

```bash

git pull --rebase # or checkout a branch, or whatever

yarn # update deps

```

If you're running `yarn watch` in the `graphile-engine` folder then it's

probably a good idea to restart it incase any of the dependencies have

changed.

### PostGraphile

(In the `graphile-engine/postgraphile` folder.)

```

git pull --rebase # or checkout a branch, or whatever

yarn # update deps

./rmlocal.sh # Remove deps that should be served by the parent graphile-engine

```

The `./rmlocal.sh` script is the one you must remember to run again -

otherwise yarn will restore the release versions of `postgraphile-core`,

`graphile-build-pg` and `graphile-build`, thus your local changes won't be represented.

## Tests

PostGraphile uses [Jest](http://facebook.github.io/jest/) for testing to take

advantage of Jest’s snapshot feature. We test against a local database, so make sure [it is set up](#Database\_setup).

PostgreSQL is by default running on `localhost:5432` (if that port isn't already used).

### Graphile Engine

Graphile Engine uses a user-configurable test database, for example `graphileengine\_test`:

```

createdb graphileengine\_test

```

The tests of the `postgraphile-core`, `graphile-utils` and `pg-pubsub` packages use the `TEST\_DATABASE\_URL` environment variable, which is mandatory.

Its format is a [connection URL](https://www.postgresql.org/docs/11/libpq-connect.html#id-1.7.3.8.3.6), passed to [node-postgres](https://node-postgres.com/features/connecting#Connection URI).

The `lds` and `subscriptions-lds` packages' test commands will create and use a new `"lds\_test"` database using the `createdb` command. If you cannot make `createdb` run directly without options (e.g. if you're not using `trust` authentication) then you may set the `LDS\_TEST\_DATABASE\_URL` environment variable and seed the DB and run the tests manually (see the relevant `package.json` to see what is done by the `yarn test` command). Alternatively, set the [environment variables for `createdb`](https://www.postgresql.org/docs/current/app-createdb.html#id-1.9.4.4.7) to direct it your server.

Note: before you can run those tests, you'll need to configure your PostgreSQL server to support logical decoding for the live queries tests.

See [the @graphile/lds README](packages/lds/README.md#postgresql-configuration) for how to enable `wal\_level = logical`.

We must then export this `TEST\_DATABASE\_URL` environment variable so the tests

know where to install. \*\*WARNING\*\*: this database will be overwritten!

```

# assuming passwordless login is set up

export TEST\_DATABASE\_URL="postgres://localhost/graphileengine\_test"

```

Then you can run the tests with

```

yarn test

```

This takes a while; I'd advise that you focus on the integration tests in

`postgraphile-core` in most cases; and since we're using jest you can pass a

filter such as `queries` to only run tests with a file name that contains the

word `queries`:

```

cd packages/postgraphile-core

yarn test queries

```

### PostGraphile

To run PostGraphile tests you will need to first create the

`postgraphile\_test` database:

```bash

createdb postgraphile\_test

```

The PostGraphile tests use the `TEST\_PG\_URL` environment variable as a connection URL, which can be overwritten. (This differs from the Graphile Engine `TEST\_DATABASE\_URL` to avoid conflicts between these two independent test suites.)

The default value is `'postgres:///postgraphile\_test'`.

Then run the test suite with:

```bash

yarn test

```

When developing PostGraphile, we recommend using the Jest watch mode feature.

So instead you would run tests like so:

```bash

yarn test --watch

```

(If you get an error about too many open files, consider installing

`watchman` as mentioned above.)

Now, only the tests in the files you have changed will be run. There are some

slow tests in the PostGraphile suite so hopefully this should make your

development time faster. Once you are in watch mode, Jest will present you with

some options you can use to better configure your testing experience.

### Snapshots

We make use of the Jest snapshot feature. Even when you change small things

the snapshot tests are likely to fail, this is expected. To update the

snapshots so that the tests pass again, you can press `u` if you're running

in watch mode, or you can run `yarn test -u`. You should carefully review the

changes to the snapshots to ensure they're what you intended. Commit the

changes to the snapshots and the changed snapshots will be reviewed along

with the rest of your changes in the PR review process.

### Linting

PostGraphile uses ESLint and Travis CI to test builds and enforce lint rules:

[travis-ci.org/graphile/postgraphile](https://travis-ci.org/graphile/postgraphile).

## GraphiQL

The instance of GraphiQL used by PostGraphile is a

[`create-react-app`](https://github.com/facebookincubator/create-react-app)

located in `postgraphiql`. When developing PostGraphile (running

`scripts/dev` only), GraphiQL will run on a different port to take advantage

of the `create-react-app` developer experience.

Note that `postgraphiql` has it's own `package.json` and `yarn.lock` because

it depends on a specific version of GraphQL which is different from the wide

range supported by PostGraphile/Graphile Engine. When we build PostGraphile

before publishing (with `scripts/build`), GraphiQL is built into a resources

served by the PostGraphile middleware people import into their projects.

## Commit messages

PostGraphile team use

[karma-style](http://karma-runner.github.io/1.0/dev/git-commit-msg.html) commit

messages: a type (`feat`/`fix`/`chore`/`docs`/etc.), a scope

(`graphql`/`postgraphile`/`examples`/`tests`) and then the commit message.

Commit messages are written in the imperative tense.

Here’s a few examples:

```

feat(ci): run against multiple postgres versions

fix(postgraphile): fix opaque error messages

chore(docs): rename anonymous role to default role

```

When committing to a branch or a PR you do not need to adhere to this format.

However, all commits to the `master` branch \_must\_ be in this format. Often all

of the commits in a PR will be squashed and a commit message of this format

will be written to summarize the changes.

You must use one of the following types:

- `chore`

- `docs`

- `feat`

- `fix`

- `refactor`

- `style`

- `test`

Common scopes are as follows. You are not required to use any of the following

scopes and may instead invent your own. These are just a few that get commonly

used.

- `\*`

- `postgraphile`

- `graphql`

- `interface`

- `postgres`

- `package`

- `scripts`

- `examples`

- `ci`

## Resources

Here are some resources that will help you learn more about Postgres and

GraphQL so that you may understand more of what is going on inside

PostGraphile.

- [The Anatomy of a GraphQL Query.](https://dev-blog.apollodata.com/the-anatomy-of-a-graphql-query-6dffa9e9e747) This article provides the vocabulary you need to talk about a GraphQL query technically.