# How to Contribute to Bookshelf.js

\* Before sending a pull request for a feature or bug fix, be sure to have

[tests](https://github.com/bookshelf/bookshelf/tree/master/test).

\* Use the same coding style as the rest of the codebase. There is a pre-commit hook that formats

code automatically for you using [prettier](https://prettier.io/), so this should be taken care of

already.

\* Fill in the issue or pull request templates when creating your entry. This will help clarify the

scope of your proposal or the subject of your issue.

\* All pull requests should be made to the `master` branch.

## Development Environment Setup

You'll need to have `git` installed obviously. Begin by forking the

[main repository](https://github.com/bookshelf/bookshelf) and then getting the code from your forked copy:

```sh

git clone git@github.com:yourusername/bookshelf.git

```

Afterwards go to the bookshelf directory that was just created and install the dependencies:

```sh

npm install

```

At this point the only thing missing are the databases that will be used for running some of the tests of the automated

test suite.

There are three options for setting up this part:

\* The first one is to use docker containers for the database servers. This is explained below.

\* Alternatively you can provide configuration options of the database servers and set them up manually. This is also

explained further down.

\* Last option is to use a config file in case you already have your servers configured and don't want to change any of

their configurations.

### Using Docker Containers

You can install [Docker](https://docs.docker.com/engine/installation/#supported-platforms) and `docker-compose` easily

on any Operating System. After picking the correct operating system installer (via package manager or download) just

run the following command on the root of your cloned Bookshelf repository:

```sh

# This starts the test databases

docker-compose up -d

```

You can also teardown the databases with:

```sh

docker-compose down --remove-orphans

```

When using Docker you may run the tests many times against the same DB instances. The tests reset DB state on each run.

\*\*Note:\*\* admin privileges are needed, so you should act according to your chosen Operating System.

### Manual Database Servers Setup

The two sections below deal with manually setting up the database servers needed for running tests. If you prefer an

easier alternative you can use Docker containers instead, as explained in the previous section.

#### MySQL

You can install [MySQL](https://www.mysql.com/) easily on most linux distros by using their package manager. With Ubuntu

this should do it:

```sh

sudo apt-get install mysql-server mysql-client

```

On OSX you can download a disk image directly from the [MySQL Downloads page](http://dev.mysql.com/downloads/mysql/), or

use one of the popular package managers like [homebrew](http://brew.sh/) or [MacPorts](https://www.macports.org/).

To run the test suite you will need to make sure it is possible to connect as the user `root` without the need for a

password.

It is strongly recommended that you use the command line `mysql` client to access your MySQL instance since there can be

problems connecting as the root user with some graphical clients like `phpMyAdmin`. To check if you can connect as root

without needing a password use the following command:

```sh

mysql -u root

```

If you see an error like:

```sh

ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)

```

that means you can't login as root without a password. If you do know the root user's password just login with the known

password like this:

```sh

mysql -u root -p

```

and enter the password when asked. Then just set an empty password for root like so:

```SQL

USE mysql;

UPDATE user SET authentication\_string = "" WHERE User = "root";

FLUSH PRIVILEGES;

QUIT;

```

If you see an error like:

```sh

ERROR 1054 (42S22): Unknown column 'authentication\_string' in 'field list'

```

It's because you are using an MySQL version older than 5.7. The `authentication\_string` column has replaced the

`password` column in this newer version. If you are using an older version just use `password` instead of

`authentication\_string` in the example above.

Note that you'll probably need to set the password to `NULL` instead of an empty string in MySQL versions 5.5 and older.

The above example should work with versions 5.7 and newer.

If you have forgotten the root password you'll need to take some extra steps to reset it. Take a look at

[this Stack Overflow answer](http://stackoverflow.com/a/7825212/504930) for further details.

If you still can't access the MySQL server without a root password after all this trouble then

[this answer on Ask Ubuntu](https://askubuntu.com/a/784347/18193) has another alternative approach to the problem.

#### PostgreSQL

You can install [PostgreSQL](http://www.postgresql.org/) easily on most linux distros by using their package manager.

With Ubuntu this should do it:

```sh

sudo apt-get install postgresql postgresql-client

```

On OSX the easiest way is probably by using [PosgresApp](http://postgresapp.com/). It should also be available to

install via [homebrew](http://brew.sh/) or [MacPorts](https://www.macports.org/) if you prefer.

In the case of PostgreSQL the requirement is to be able to connect as the `postgres` user on localhost also without the

need for a password. This can be achieved by editing or adding the following line in the `pg\_hba.conf` file:

```

host all all 127.0.0.1/32 trust

```

This file can be found in `/etc/postgresql/9.4/main/` on most linux systems. The `9.4` part could be different depending

on the version that is available in your distro. On OSX the location of this file will depend on the installation method

chosen, but for the recommended PostgresApp install it will be in `/Users/[yourusername]/Library/Application

Support/Postgres/var-9.3/`. Again, the `var-9.3` part may be different depending on the version you installed.

The `trust` in the example above tells the locally running PostgreSQL server to ignore user passwords and always grant

access on clients connecting locally. Do not use this setting in a production environment.

After editing the `pg\_hba.conf` file you'll need to restart the PostgreSQL server for the changes to take effect.

### Using a Configuration File

If you don't want to go to the trouble of performing the changes explained in the previous two sections you can instead

use a config file that tells the test suite about your current database setup.

The tests will look for a `BOOKSHELF\_TEST` environment variable that points to a `config.js` file with the connection

details for each database server. This file must not be the same database config file you use for any other application,

otherwise you risk data loss in that application.

Example config file:

```javascript

module.exports = {

mysql: {

database: 'bookshelf\_test',

user: 'root',

encoding: 'utf8'

},

postgres: {

user: 'myusername',

database: 'bookshelf\_test',

password: 'secretpassword',

host: 'localhost',

port: 5432,

charset: 'utf8',

ssl: false

},

sqlite3: {

filename: ':memory:'

}

};

```

This file can be placed anywhere on your system and can have any name that you like, as long as the environment variable

is pointing correctly to it. For convenience you can put it in your home directory and add the following line to your

`.bashrc` or `.zshrc`:

```

export BOOKSHELF\_TEST='/home/myusername/.bookshelf\_config.js'

```

#### Database Creation

After having ensured the test suite can access both database servers just create a new database on each that will be

used exclusively by Bookshelf.js:

```SQL

CREATE DATABASE bookshelf\_test;

```

Note that this step isn't necessary when using Docker.

### Running the Tests

The easiest way to run the tests is by using docker & docker-compose as [explained above](#using-docker-containers).

The test suite requires that both MySQL and PostgreSQL servers have a database named `bookshelf\_test`. See the sections

above for further instructions.

Once you have your development environment properly setup, you can run the tests:

```sh

npm test

```

Always make sure all the tests are passing before sending a pull request.

## Publishing a New Release

The release process is fairly well automated. You do need publishing rights to the npmjs.com

[bookshelf package](https://www.npmjs.com/package/bookshelf) and write access to the GitHub repository. Once that's set

just make sure to follow the process explained below in the correct order.

1. You should draft a new release on GitHub. This isn't strictly necessary, but is highly recommended. At this time you

shouldn't publish it yet, but just save it as a draft instead.

2. Make sure you're in the `master` branch with all the latest changes: `git pull`.

2. Update the `CHANGELOG.md` file and update the version number of `package.json`. For the changelog just follow the

format of the previous update. In general you should link to PRs instead of issues when mentioning changes. If the PRs'

descriptions are well written they should already include any associated issues. At this point there is no need to

commit and/or push these changes since that is taken care of automatically by the release scripts.

3. Just run `npm publish` and sit back.