# Contributing

This document is going to take you through the process of contributing to the Mock Service Worker library. Please make sure you read it before landing your first contribution. Thank you.

## Getting started

### Fork the repository

Please use the GitHub UI to \*\*fork this repository\*\*. Mock Service Worker has a forked builds enabled on the CI, so you will see the test status of your fork's branch.

### Install

```bash

$ cd msw

$ yarn install

```

> Please use the [\*\*Yarn\*\*][yarn-url] package manager while working on this project. Thank you for respecting our internal decisions.

## Git workflow

```bash

# Navigate to the `master` branch and ensure it's up-to-date

$ git checkout master

$ git pull --rebase

# Create a feature branch

$ git checkout -b feature/gql-subscriptions

# Commit the changes

$ git add .

$ git commit -m 'Adds support for GraphQL subscriptions'

# Push

$ git push -u origin feature/rest-api-support

```

Once you have pushed the changes to your remote feature branch, [Create a pull request](https://github.com/open-draft/msw/compare) to the `msw` repository in GitHub. Undergo the process of code review, where the maintainers of the library will help you to get the changes from great to awesome, and enjoy your implementation merged to `master`.

## Tests

Make sure to cover the introduced changes with relevant tests.

- [Jest][jest-url], a test runner.

### Unit tests

#### Writing a unit test

Create a test suite next to the function you would like to test:

```bash

$ touch src/multiply.test

```

> In this example we are creating a unit test for the existing `multiply` function.

Write a test file and provide your assertions:

```ts

import { multiply } from './multiply'

describe('multiply', () => {

describe('given two numbers' () => {

let result: ReturnType<typeof multiply>

beforeAll(() => {

result = multiply(3, 4)

})

it('should return the result of their multiplication', () => {

expect(result).toBe(12)

})

})

})

```

#### Running all unit tests

```bash

$ yarn test:unit

```

#### Running a single unit test

```bash

$ yarn test:unit path/to/suite.test.ts

```

### Integration tests

- [Jest][jest-url], a test runner.

- [Puppeteer][puppeteer-url], a Chromium automation tool.

#### Writing an integration test

Each integration test consists of two parts:

- `test-name.mocks.ts`, a client-side usage scenario. This is a mock definition file that showcases how the library is used in this test scenario.

- `test-name.test.ts`, an actual test suite. This is where you interact with a test scenario and write your assertions.

> Please refer to the [existing integration tests](https://github.com/open-draft/msw/tree/master/test) for more details on how a test suite must be structured and what interaction commands are available.

#### Running all integration tests

```bash

$ yarn test:integration

```

#### Running a single integration test

```bash

$ yarn test:integration test/rest-api/basic.mock.ts

```

#### Running a usage scenario

When working on a feature or a bug fix it's useful to interact with your usage example. There's a dedicated command that can run a given usage scenario in a local server for you to work with:

```bash

$ yarn test:focused test/rest-api/basic.mock.ts

```

> Navigate to the URL in the terminal to preview and interact with the usage scenario.

## Build

To perform a production build of the library run the following command:

```bash

$ yarn build

```

[yarn-url]: https://classic.yarnpkg.com/en/

[jest-url]: https://jestjs.io

[puppeteer-url]: https://pptr.dev