# Contributing

This repository contains all necessary packages of the WebdriverIO project (excluding plugins that were contributed by 3rd party developers). These packages have individual descriptions in their README files (`/packages/<package>/README.md`) providing information about their scope and responsibilities. Even though the build commands might differ from package to package the way to work with these is the same. This project uses [Lerna](https://lerna.js.org/) to manage all its subprojects in this monolith repository.

We are trying to make contributing to this project as easy and transparent as possible. If there is any information missing that prevents you from sending in a pull request, please let us know. We treat these kind of issues like actual bugs.

Also if there is anything that needs to get simplified, also please let us know.

## Set Up Project

In order to set up this project and start contributing follow this step by step guide:

\* Fork the project.

\* Clone the project somewhere on your computer

```sh

$ git clone git@github.com:<your-username>/webdriverio.git

```

\* If you need to update your fork you can do so following the steps [here](https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/syncing-a-fork)

\* Switch to Node v10 LTS (you should be able to use older/newer versions of Node but we recommend to use v10 LTS so all developers are on the same side)

\* Setup project:

```sh

$ npm install

$ npm run setup-full

```

\* Bootstraps sub-projects via ```npm run bootstrap```

Many packages depend on each other, in order to properly set up the dependency tree you need to run the [Lerna Bootstrap](https://github.com/lerna/lerna#bootstrap) command to create all necessary links. As this project also does some other house keeping tasks it is recommend to use the package bootstrap command:

\* Builds all subpackages via ```npm run build```

As the last step you need to build all sub-packages in order to resolve the internal dependencies. We also have a NPM command for that:

\* Run Tests to ensure that everything is set up correctly

```sh

$ npm run test

# run test for a specific sub project (e.g. webdriver)

$ npx jest ./packages/webdriver/tests

```

It should give you a passing result. Now you can move on to setup your development environment and start working on some code.

## Link changes to your current project

When modifying core WebdriverIO packages you can link those changes to your current project to test the changes that you made.

If you are working on a package, lets say the @wdio/cli package, you can link this in the following way from the WebdriverIO repositority.

```

$ cd packages/wdio-cli

$ npm link

```

Then in your current project you can link your changes from the the @wdio/cli package to your current project.

```

$ cd your-main-test-code

$ npm link @wdio/cli

```

## Work On Packages

If you start making changes to specific packages, make sure you listen on file changes and transpile the code everytime you press save. To do that for all packages, run:

```sh

$ npm run watch

```

If you only work on a single package you can watch only for that one by calling:

```sh

# e.g. `$ npm run watch:pkg @wdio/runner`

$ npm run watch:pkg <package-name>

```

It is also a good idea to run jest in watch mode while developing on a single package to see if changes affect any tests:

```sh

$ npx jest ./packages/<package-name>/tests --watch

```

## Create New Package

All WebdriverIO sub packages require a certain structure to work within the wdio ecosystem. To simplify the process of creating a new sub package we build a NPM script that does all the boilerplate work for you. Just run:

```sh

$ npm run create

```

It will ask you about the type and name of the new package and creates all the files for you.

## Run e2e Experience With Smoke Tests

WebdriverIO maintains a set of smoke test suites that allows to represent the full e2e experience of a user running the wdio testrunner. It is setup in a way so it doesn't require an actual browser driver since all requests are mocked using the `@wdio/webdriver-mock-service`. This offers you an opportunity to run a wdio test suite without setting up a browser driver and a test page. You can run all smoke tests via:

```sh

$ npm run test:smoke

```

There are several [test suites](https://github.com/webdriverio/webdriverio/blob/master/tests/smoke.runner.js#L359-L378) defined that run in different environments, e.g. Mocha, Jasmine and Cucumber. You can run a specific test suite by calling, e.g.:

```sh

$ npm run test:smoke mochaTestrunner

```

You can define your own scenario of mock responses in the [`@wdio/webdriver-mock-service`](https://github.com/webdriverio/webdriverio/blob/master/packages/wdio-webdriver-mock-service/src/index.js#L142-L149).

## TypeScript definitions

WebdriverIO provides their own type definitions for project that use TypeScript. Given the large amount of commands it would make it unmaintainable to \_\_not\_\_ automate the process of generating these. However there are certain edge cases where manual work is required.

All type definitions are located in the sub package folders. Some of them are autogenerated and should not be modified as changes might be overwritten. Each package defines its typings differently:

- the `webdriver` and `devtools` package define all commands and their respective types in the json file of the `@wdio/protocols` package

- the `webdriverio` package has the types defined in the comments above every command in `/packages/webdriverio/src/commands/\*\*/\*`

- the `@wdio/allure-reporter` has its interface defined in comments within the service (`/packages/wdio-allure-reporter/src/index.js`)

- all other packages have their type definitions directly in the `\*.d.ts` in the root directory of that package (e.g. `/packages/wdio-cucumber-framework/cucumber-framework.d.ts`) - these are \_\_not\_\_ autogenerated and you can make modifications directly in these files

You can find all files responsible for the generating the typings here: https://github.com/webdriverio/webdriverio/tree/master/scripts/type-generation. You can trigger the process by calling:

```sh

$ npm run generate:typings

```

This will run the scripts in the directory shown above and generate the typings various packages located in `/packages/\*`. Whenever you change the types in these packages, make sure you re-generate the types with the command shown above. In order to ensure that packages where type definitions have changed are released we keep the generated type definition in source control.

### Adding Types for WebdriverIO

If you add a new command or extend an existing with a special option, follow this guide to ensure that types are generated properly. As mentioned above every WebdriverIO command contains a comment section that is used to generate the documentation and type definition. For example the `newWindow` command:

```js

/\*\*

\*

\* Open new window in browser. This command is the equivalent function to `window.open()`. This command does not

\* work in mobile environments.

\*

\* \_\_Note:\_\_ When calling this command you automatically switch to the new window.

\*

\* <example>

:newWindowSync.js

it('should open a new tab', () => {

browser.url('http://google.com')

console.log(browser.getTitle()) // outputs: "Google"

browser.newWindow('https://webdriver.io', 'WebdriverIO window', 'width=420,height=230,resizable,scrollbars=yes,status=1')

console.log(browser.getTitle()) // outputs: "WebdriverIO · Next-gen browser and mobile automation test framework for Node.js"

browser.closeWindow()

console.log(browser.getTitle()) // outputs: "Google"

});

\* </example>

\*

\* @param {String} url website URL to open

\* @param {NewWindowOptions=} options newWindow command options

\* @param {String=} options.windowName name of the new window

\* @param {String=} options.windowFeatures features of opened window (e.g. size, position, scrollbars, etc.)

\*

\* @return {String} id of window handle of new tab

\*

\* @uses browser/execute, protocol/getWindowHandles, protocol/switchToWindow

\* @alias browser.newWindow

\* @type window

\*/

```

WebdriverIO follows the pattern that every command property that is required for the command execution (e.g. in the example above the `url` parameter) is a parameter of the command. Every optional parameter that just modifies the way the command is being executed is being attached as a command option object (e.g. in the example above `windowName` and `windowFeatures`). To properly create typings for the options object we need to give this interface a new name (e.g. `NewWindowOptions`) and:

- define this new interface in the [type definition template](/scripts/templates/webdriverio.tpl.d.ts)

- add custom interfaces to the list of [`CUSTOM\_INTERFACES`](/scripts/type-generation/constants.js)

- write some type tests

### Testing Type Definitions

To make sure that we don't accidently change the types and cause users test to break we run some simple typescript checks. You can run all type definition tests by running:

```sh

$ npm run test:typings

```

This will run all tests for all type definitions WebdriverIO provides. These tests just check if TypeScript can compile them according to the generated type definitions. All type checks are located in `/webdriverio/tests/typings`. If you extend a WebdriverIO command or interfaces for other type definitions please ensure that you have used it in these files. The directory contains tests for the asynchronous usage of WebdriverIO as well as for using it synchronously with `@wdio/sync`.

For example to test the `touchActions` properties we have it tested in `/tests/typings/webdriverio/async.ts`:

```ts

// touchAction

const ele = await $('')

const touchAction: WebdriverIO.TouchAction = {

action: "longPress",

element: await $(''),

ms: 0,

x: 0,

y: 0

}

await ele.touchAction(touchAction)

await browser.touchAction(touchAction)

```

as well as in `/tests/typings/sync/sync.ts`:

```ts

const ele = $('')

const touchAction: WebdriverIO.TouchAction = {

action: "longPress",

element: $(''),

ms: 0,

x: 0,

y: 0

}

ele.touchAction(touchAction)

browser.touchAction(touchAction)

```

## Test Your Changes

In order to test certain scenarios this project has a test directory that allows you to run predefined test. It allows you to check your code changes while you are working on it. You find all these files in `/examples`. You find all necessary information [in there](https://github.com/webdriverio/webdriverio/tree/master/examples/README.md). When submitting a PR, WebdriverIO runs the following checks:

- \*Dependency Checks\*

We automatically check if every sub-package has all dependencies installed in the package.json.

You can manually trigger this check by calling:

```sh

$ npm run test:depcheck

```

- \*EsLint\*

Common EsLint test to align code styles and detecting syntax errors early.

You can manually trigger this check by calling:

```sh

$ npm run test:eslint

```

- \*TypeScript Definition Tests\*

As we generate our type definitions we want to be cautious that the generated definitions actually define the interface as expected. Read more

on that at [Testing Type Definitions](#testing-type-definitions).

You can manually trigger this check by calling:

```sh

$ npm run test:typings

```

- \*Unit Tests\*

Like every project we unit test our code and ensure that new patches are properly tested. The coverage treshold is pretty high so ensure that

your changes cover all necessary code paths. We are using [Jest](https://jestjs.io/) as unit test framework here.

You can manually trigger this check by calling:

```sh

$ npm run test:coverage

```

- \*Smoke Tests\*

While unit tests already cover a lot cases, we run in addition to that smoke tests that simulate test scenarios which are difficult to test

on a unit level as they include functionality of dependencies that are stubbed out in unit tests. Such scenarios are for example: proper

test retries or failure handling. Smoke test run actual e2e test where the driver is being stubbed (via [`@wdio/smoke-test-service`](https://github.com/webdriverio/webdriverio/blob/master/packages/wdio-smoke-test-service/package.json)) to return fake results.

You can manually trigger this check by calling:

```sh

$ npm run test:smoke

```

- \*e2e Tests\*

Last but not least we run actual e2e tests with real browser to ensure that our WebDriver DevTools implementation is working as expected. These

test spin up headless Chrome and Firefox browser to test the commands implemented in the `devtools` package. Given that WebDriver functionality

is already tested with [WPT](https://github.com/web-platform-tests/wpt) we don't need to do it there.

You can manually trigger this check by calling:

```sh

$ npm run test:e2e

```

## Back-Porting Bug Fixes

Starting from v6 the WebdriverIO team tries to backport all features that would be still backwards compatible with older versions. The team tries to release a new major version every year (usually around December/January). With a new major version update (e.g. v6) we continue to maintain the last version (e.g. v5) and depcrecate the previous maintained version (e.g. v4). With that the team commits to always support 2 major versions.

### As Triager

Everyone triaging or reviewing a PR should label it with `backport-requested` if the changes can be applied to the maintained (previous) version. Generally every PR that would not be a breaking change for the previous version should be considered to be ported back. If a change relies on features or code pieces that are only available in the current version then a back port can still be considered if you feel comfortable making the necessary adjustments. That said, don't feel forced to back port code if the time investment and complexity is too high. Backporting functionality is a reasonable contribution that can be made by any contributor.

### As A Merger

Once a PR with a `backport-requested` label got merged, you are responsible for backporting the patch to the older version. To do so, pull the latest code from GitHub:

```sh

$ git pull

$ git fetch --all

$ git checkout v5

```

Before you can start please export an `GITHUB\_AUTH` token into your environment in order to allow the executing script to fetch data about pull requests and set proper labels. Go to your [personal access token](https://github.com/settings/tokens) settings page and generate such token with only having the `public\_repo` field enabled. Then export it into your environment and run the backport script. It fetches all commits connected with PRs that are labeled with `backport-requested` and cherry-picks them into the maintainance branch. Via an interactive console you can get the chance to review the PR again and whether you want to backport it or not. To start the process, just execute:

```sh

$ npm run backport

```

If during the process a cherry-pick fails you can always abort and manually troubleshoot. If you are not able to resolve the problem, create an issue in the repo and include the author of that PR. A successful backport of two PRs will look like this:

```

$ npm run backport

> webdriverio-monorepo@ backport /path/to/webdriverio/webdriverio

> node ./scripts/backport.js

? You want to backport "Backporting Test PR" by christian-bromann?

(See PR https://github.com/webdriverio/webdriverio/pull/4890) Yes

Backporting sha 264b7bc49dfc3fe8f1ed8b58d681ce562aaf1544 from v6 to v5

[cb-backport-process e47c5527] Backporting Test PR (#4890)

Author: Christian Bromann <mail@christian-bromann.com>

Date: Mon Dec 16 14:54:02 2019 +0100

1 file changed, 2 insertions(+)

? You want to backport "Second backport test" by christian-bromann?

(See PR https://github.com/webdriverio/webdriverio/pull/4891) Yes

Backporting sha 77dc52fdb86c51242b92f299998d2904004cb347 from v6 to v5

[cb-backport-process 35a3ad41] Second backport test (#4891)

Author: Christian Bromann <mail@christian-bromann.com>

Date: Mon Dec 16 16:01:46 2019 +0100

1 file changed, 2 deletions(-)

Successfully backported 2 PRs ?!

Please now push them to master and make a new v5.x release!

```

You can always reach out to the `webdriverio/ProjectCommitters` channel on Gitter for questions.

## Release New Version

Package releases are made using Lerna's release capabilities and executed by the [technical steering committee](https://github.com/webdriverio/webdriverio/blob/master/GOVERNANCE.md#the-technical-committee) only. Before you can start please export an `GITHUB\_AUTH` token into your environment in order to allow [`lerna-changelog`](https://www.npmjs.com/package/lerna-changelog#github-token) to gather data about the upcoming release and autogenerate the [CHANGELOG.md](/CHANGELOG.md). Go to your [personal access token](https://github.com/settings/tokens) settings page and generate such token with only having the `public\_repo` field enabled. Then export it into your environment:

```sh

$ export GITHUB\_AUTH=...

```

You are now all set and just need to call:

```sh

# ensure to have pulled the latest code

$ git pull origin master --tags

# release using Lerna

$ npm run release

```

and choose the appropriate version upgrade based on the [Semantic Versioning](https://semver.org/). To help choose the right release type, here are some general guidelines:

- \_\_Breaking Changes\_\_: never do these by yourself! A major release is always a collaborative effort between all TSC members. It requires consensus from all of them.

- \_\_Minor Release\_\_: minor releases are always required if a new, user focused feature was added to one of the packages. For example if a command was added to WebdriverIO or a service provides a new form of integration a minor version bump would be appropiate. However if an internal package like `@wdio/local-runner` exposes a new interface that is solely used internally we can consider that as a patch release.

- \_\_Patch Release\_\_: everytime a bug was fixed, documentation (this includes TypeScript definitions) got updated or existing functionality was improved we should do a patch release.

If you are unsure about which release type to pick, reach out in the TSC Gitter channel.