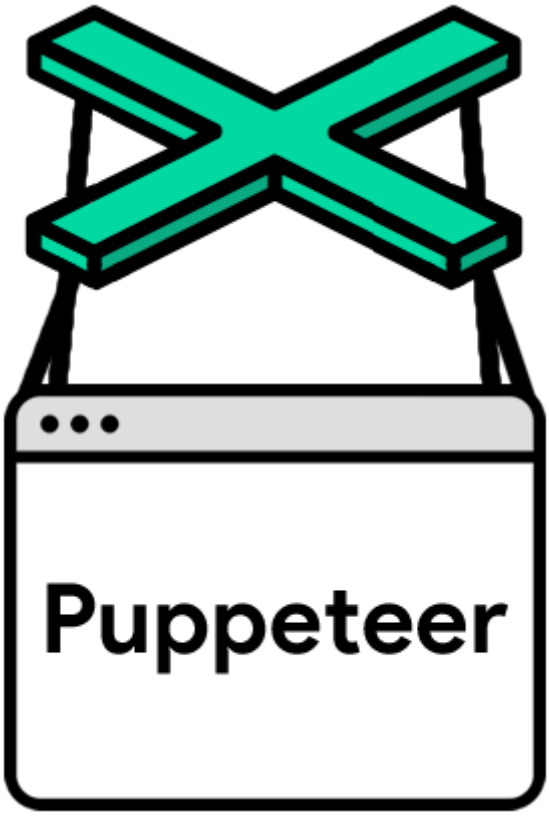
Version: 19.6.3

Puppeteer

Guides | **API** | **FAQ** | **Contributing** | **Troubleshooting**



> Puppeteer is a Node.js library which provides a high-level API to control Chrome/Chromium over the DevTools Protocol. Puppeteer runs in headless mode by default, but can be configured to run in full (non-headless) Chrome/Chromium.

What can I do?

Most things that you can do manually in the browser can be done using Puppeteer! Here are a few examples to get you started:

- Generate screenshots and PDFs of pages.
- Crawl a SPA (Single-Page Application) and generate pre-rendered content (i.e. "SSR" (Server-Side Rendering)).
- Automate form submission, UI testing, keyboard input, etc.
- Create an automated testing environment using the latest JavaScript and browser features.
- Capture a timeline trace of your site to help diagnose performance issues.
- Test Chrome Extensions.

Getting Started

Installation

To use Puppeteer in your project, run:

```
npm i puppeteer
# or `yarn add puppeteer`
```

```
# or `pnpm i puppeteer`
```

When you install Puppeteer, it automatically downloads a recent version of Chromium (~170MB macOS, ~282MB Linux, ~280MB Windows) that is guaranteed to work with Puppeteer. For a version of Puppeteer without installation, see `puppeteer-core`.

Configuration

Puppeteer uses several defaults that can be customized through configuration files.

For example, to change the default cache directory Puppeteer uses to install browsers, you can add a `.puppeteerrc.cjs` (or `puppeteer.config.cjs`) at the root of your application with the contents

```
const {join} = require('path');

/**
 * @type {import("puppeteer").Configuration}
 */
module.exports = {
  // Changes the cache location for Puppeteer.
  cacheDirectory: join(__dirname, '.cache', 'puppeteer'),
};
```

After adding the configuration file, you will need to remove and reinstall `puppeteer` for it to take effect.

See the configuration guide for more information.

`puppeteer-core`

Every release since v1.7.0 we publish two packages:

- `puppeteer`
- `puppeteer-core`

`puppeteer` is a *product* for browser automation. When installed, it downloads a version of Chromium, which it then drives using `puppeteer-core`. Being an end-user product, `puppeteer` automates several workflows using reasonable defaults that can be customized.

`puppeteer-core` is a *library* to help drive anything that supports DevTools protocol. Being a library, `puppeteer-core` is fully driven through its programmatic interface implying no defaults are assumed and `puppeteer-core` will not download Chromium when installed.

You should use `puppeteer-core` if you are connecting to a remote browser or managing browsers yourself. If you are managing browsers yourself, you will need to call `puppeteer.launch` with an an explicit `executablePath` (or `channel` if it's installed in a standard location).

When using `puppeteer-core`, remember to change the import:

```
import puppeteer from 'puppeteer-core';
```

Usage

Puppeteer follows the latest maintenance LTS version of Node.

Puppeteer will be familiar to people using other browser testing frameworks. You launch/connect a browser, create some pages, and then manipulate them with Puppeteer's API.

For more in-depth usage, check our guides and examples.

Example

The following example searches developer.chrome.com for blog posts with text "automate beyond recorder", click on the first result and print the full title of the blog post.

```
import puppeteer from 'puppeteer';

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();

  await page.goto('https://developer.chrome.com/');

  // Set screen size
  await page.setViewport({width: 1080, height: 1024});

  // Type into search box
  await page.type('.search-box__input', 'automate beyond recorder');

  // Wait and click on first result
  const searchResultSelector = '.search-box__link';
  await page.waitForSelector(searchResultSelector);
  await page.click(searchResultSelector);

  // Locate the full title with a unique string
  const textSelector = await page.waitForSelector(
    'text/Customize and automate'
```

```
  );
  const fullTitle = await textSelector.evaluate(el => el.textContent);

  // Print the full title
  console.log('The title of this blog post is "%s".', fullTitle);

  await browser.close();
})();
```

Default runtime settings

1. Uses Headless mode

Puppeteer launches Chromium in headless mode. To launch a full version of Chromium, set the `headless` option when launching a browser:

```
const browser = await puppeteer.launch({headless: false}); // default is true
```

2. Runs a bundled version of Chromium

By default, Puppeteer downloads and uses a specific version of Chromium so its API is guaranteed to work out of the box. To use Puppeteer with a different version of Chrome or Chromium, pass in the executable's path when creating a `Browser` instance:

```
const browser = await puppeteer.launch({executablePath: '/path/to/Chrome'});
```

You can also use Puppeteer with Firefox Nightly (experimental support). See `Puppeteer.launch` for more information.

See `this article` for a description of the differences between Chromium and Chrome. `This article` describes some differences for Linux users.

3. Creates a fresh user profile

Puppeteer creates its own browser user profile which it **cleans up on every run**.

Using Docker

See our Docker guide.

Using Chrome Extensions

See our Chrome extensions guide.

Resources

- API Documentation
- Guides
- Examples
- Community list of Puppeteer resources

Contributing

Check out our contributing guide to get an overview of Puppeteer development.

FAQ

Our FAQ has migrated to our site.