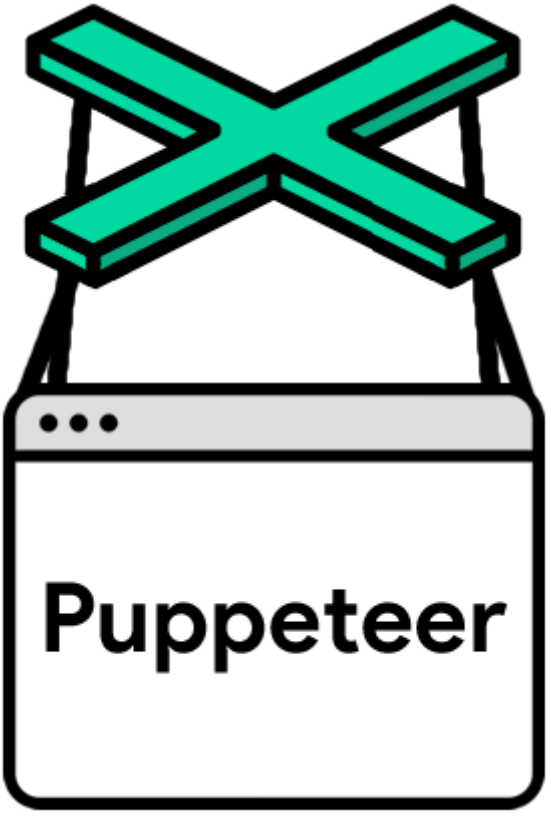
Version: 21.1.0

Puppeteer

CI passing npm v21.1.0

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 ("headful") 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 using yarn
yarn add puppeteer
# or using pnpm
pnpm i puppeteer
```

When you install Puppeteer, it automatically downloads a recent version of Chrome for Testing (~170MB macOS, ~282MB Linux, ~280MB Windows) that is guaranteed to work with Puppeteer. The browser is downloaded to the `$HOME/.cache/puppeteer` folder by default (starting with Puppeteer v19.0.0).

If you deploy a project using Puppeteer to a hosting provider, such as Render or Heroku, you might need to reconfigure the location of the cache to be within your project folder (see an example below) because not all hosting providers include `$HOME/.cache` into the project's deployment.

For a version of Puppeteer without the browser installation, see `puppeteer-core`.

If used with TypeScript, the minimum supported TypeScript version is `4.7.4`.

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 Chrome, 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 Chrome 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 () => {
  // Launch the browser and open a new blank page
  const browser = await puppeteer.launch();
  const page = await browser.newPage();

  // Navigate the page to a URL
  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

By default Puppeteer launches Chrome in old Headless mode.

```
const browser = await puppeteer.launch();
// Equivalent to
const browser = await puppeteer.launch({headless: true});
```

Chrome 112 launched a new Headless mode that might cause some differences in behavior compared to the old Headless implementation. In the future Puppeteer will start defaulting to new implementation. We recommend you try it out before the switch:

```
const browser = await puppeteer.launch({headless: 'new'});
```

To launch a "headful" version of Chrome, set the `headless` to `false` option when launching a browser:

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

2. Runs a bundled version of Chrome

By default, Puppeteer downloads and uses a specific version of Chrome 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. See status of cross-browser support 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.