
shot-scraper documentation

Release 1.12

Simon Willison

Sep 13, 2026

CONTENTS

1	Installation	3
1.1	shot-scraper install --help	3
2	Taking a screenshot	5
2.1	Adjusting the browser width and height	5
2.2	Screenshotting a specific area with CSS selectors	5
2.3	Specifying elements using JavaScript filters	6
2.4	Waiting for a delay	6
2.5	Waiting until a specific condition	7
2.6	Executing custom JavaScript	7
2.7	Using JPEGs instead of PNGs	8
2.8	Using WebP	8
2.9	Specifying the image format	8
2.10	Device scale factor	9
2.11	Retina images	9
2.12	Transparent background	9
2.13	Interacting with the page	10
2.14	Logging all requests	10
2.15	Browser arguments	11
2.16	Taking screenshots of local HTML files	11
2.17	Tips for executing JavaScript	11
2.18	Viewing console.log() output	12
2.19	shot-scraper shot --help	12
3	Websites that need authentication	15
3.1	shot-scraper auth --help	15
4	Taking multiple screenshots	17
4.1	Recording to an HTTP Archive	19
4.2	Running a server for the duration of the session	20
4.3	Running custom code between steps	21
4.4	shot-scraper multi --help	21
5	Scraping pages using JavaScript	23
5.1	Setting the browser viewport	23
5.2	Running more than one statement	24
5.3	Using async/await	24
5.4	Bypassing Content Security Policy headers	25
5.5	Using this for automated tests	25
5.6	Running JavaScript from a file	26

5.7	Running JavaScript from GitHub	26
5.8	Example: Extracting page content with Readability.js	26
5.9	shot-scraper javascript --help	27
6	Recording videos	29
6.1	Storyboard structure	30
6.2	Cursor and click visualization	32
6.3	Running a server for the duration of the storyboard	33
6.4	Scenes	34
6.5	Running custom code between steps	35
6.6	Actions	36
6.6.1	click	36
6.6.2	type	37
6.6.3	fill	37
6.6.4	press	37
6.6.5	scroll	37
6.6.6	pause	38
6.6.7	wait_for	38
6.6.8	wait_for_url	38
6.6.9	open	38
6.6.10	screenshot	38
6.6.11	sh	39
6.6.12	python	39
6.6.13	javascript	39
6.7	Complete example	40
6.8	Command options	41
6.9	shot-scraper video --help	41
7	Saving a web page to PDF	45
7.1	shot-scraper pdf --help	45
8	Dumping the HTML of a page	47
8.1	Retrieving the HTML for a specific element	47
8.2	shot-scraper html --help	47
9	Saving a web page to an HTTP Archive	49
9.1	Extracting resources from HAR files	50
9.2	shot-scraper har --help	50
10	Dumping out an accessibility tree	53
10.1	shot-scraper accessibility --help	53
11	Using shot-scraper with GitHub Actions	55
11.1	shot-scraper-template	55
11.2	Building a workflow from scratch	55
11.3	Optimizing PNGs using Oxipng	56
12	Contributing	57
12.1	Documentation	57
12.2	Publishing the release notes	57
13	shot-scraper	59
13.1	Documentation	59
13.2	Get started with GitHub Actions	59
13.3	Quick installation	59

13.4 Taking your first screenshot 60

13.5 Examples 60

A command-line utility for taking automated screenshots of websites

Quick start:

```
pip install shot-scraper
shot-scraper install
shot-scraper https://github.com/simonw/shot-scraper -h 900
```

Produces this screenshot in a file called `github-com-simonw-shot-scraper.png`:

Contents

INSTALLATION

Install this tool using pip:

```
pip install shot-scraper
```

This tool depends on Playwright, which first needs to install its own dedicated Chromium browser.

Run `shot-scraper install` once to install that:

```
shot-scraper install
```

Which outputs:

```
Downloading Playwright build of chromium v965416 - 117.2 Mb [=====] 100%  
→ 0.0s  
Playwright build of chromium v965416 downloaded to /Users/simon/Library/Caches/ms-  
→ playwright/chromium-965416  
Downloading Playwright build of ffmpeg v1007 - 1.1 Mb [=====] 100% 0.0s  
Playwright build of ffmpeg v1007 downloaded to /Users/simon/Library/Caches/ms-playwright/  
→ ffmpeg-1007
```

If you want to use other browsers such as Firefox you should install those too:

```
shot-scraper install -b firefox
```

1.1 shot-scraper install --help

Full `--help` for the `shot-scraper install` command:

```
Usage: shot-scraper install [OPTIONS]

Install the Playwright browser needed by this tool.

Usage:

    shot-scraper install

Or for browsers other than the Chromium default:

    shot-scraper install -b firefox
```

(continues on next page)

(continued from previous page)

Options:

-b, --browser	[chromium firefox webkit chrome chrome-beta]	
		Which browser to install
--help		Show this message and exit.

TAKING A SCREENSHOT

To take a screenshot of a web page and write it to `datasette-io.png` run this:

```
shot-scraper https://datasette.io/
```

If a file called `datasette-io.png` already exists the filename `datasette-io.1.png` will be used instead.

You can use the `-o` option to specify a filename:

```
shot-scraper https://datasette.io/ -o datasette.png
```

Use `-o -` to write the PNG image to standard output:

```
shot-scraper https://datasette.io/ -o - > datasette.png
```

If you omit the protocol `http://` will be added automatically, and any redirects will be followed:

```
shot-scraper datasette.io -o datasette.png
```

2.1 Adjusting the browser width and height

The browser window used to take the screenshots defaults to 1280px wide and 780px tall.

You can adjust these with the `--width` and `--height` options (`-w` and `-h` for short):

```
shot-scraper https://datasette.io/ -o small.png --width 400 --height 800
```

If you provide both options, the resulting screenshot will be of that size. If you omit `--height` a full page length screenshot will be produced (the default).

2.2 Screenshotting a specific area with CSS selectors

To take a screenshot of a specific element on the page, use `--selector` or `-s` with its CSS selector:

```
shot-scraper https://simonwillison.net/ -s '#bighead'
```

This produces `simonwillison-net.png` containing this image:

When using `--selector` the height and width, if provided, will set the size of the browser window when the page is loaded but the resulting screenshot will still be the same dimensions as the element on the page.

You can pass `--selector` multiple times. The resulting screenshot will cover the smallest area of the page that contains all of the elements you specified, for example:

```
shot-scraper https://simonwillison.net/ \  
-s '#bighead' -s .overband \  
-o bighead-multi-selector.png
```

To capture a rectangle around every element that matches a CSS selector, use `--selector-all`:

```
shot-scraper https://simonwillison.net/ \  
--selector-all '.day' \  
-o just-the-day-boxes.png
```

You can add `--padding 20` to add 20px of padding around the elements when the shot is taken.

2.3 Specifying elements using JavaScript filters

The `--js-selector` and `--js-selector-all` options can be used to use JavaScript expressions to select elements that cannot be targetted just using CSS selectors.

The options should be passed JavaScript expression that operates on the `el` variable, returning `true` if that element should be included in the screenshot selection.

To take a screenshot of the first paragraph on the page that contains the text “shot-scraper” you could run the following:

```
shot-scraper https://github.com/simonw/shot-scraper \  
--js-selector 'el.tagName == "P" && el.innerText.includes("shot-scraper")'
```

The `el.tagName == "P"` part is needed here because otherwise the `<html>` element on the page will be the first to match the expression.

The generated JavaScript that will be executed on the page looks like this:

```
Array.from(document.getElementsByTagName('*')).find(  
  el => el.tagName == "P" && el.innerText.includes("shot-scraper")  
)<div data-bbox="112 640 889 670" data-label="Text">

The --js-selector-all option will select all matching elements, in a similar fashion to the --selector-all option described above.


```

2.4 Waiting for a delay

Sometimes a page will not have completely loaded before a screenshot is taken. You can use `--wait X` to wait the specified number of milliseconds after the page load event has fired before taking the screenshot:

```
shot-scraper https://simonwillison.net/ --wait 2000
```

2.5 Waiting until a specific condition

In addition to waiting a specific amount of time, you can also wait until a JavaScript expression returns true using the `--wait-for expression` option.

This example takes the screenshot the moment a `<div>` with an ID of content becomes available in the DOM:

```
shot-scraper https://.../ \
  --wait-for 'document.querySelector("div#content")'
```

Here's an example that waits for a specific element to become available (in this case a cookie consent overlay) and then removes it before the screenshot is taken:

```
shot-scraper -h 800 'https://www.spiegel.de/international/' \
  --wait-for "() => {
    const div = document.querySelector('[id^="sp_message_container"]');
    if (div) {
      div.remove();
      return true;
    }
  }"
```

If the expression takes longer than 30 seconds to resolve `shot-scraper` will exit with an error.

2.6 Executing custom JavaScript

You can use custom JavaScript to modify the page after it has loaded (after the 'onload' event has fired) but before the screenshot is taken using the `--javascript` option:

```
shot-scraper https://simonwillison.net/ \
  -o simonwillison-pink.png \
  --javascript "document.body.style.backgroundColor = 'pink';"
```

For larger scripts that would be awkward to include on the command line, use `--js-file` to read the JavaScript from a file instead:

```
shot-scraper https://simonwillison.net/ \
  -o simonwillison-modified.png \
  --js-file remove-banners.js
```

Use `--js-file -` to read the script from standard input, or `--js-file gh:username/script` to load it from `github.com/username/shot-scraper-scripts/script.js` on GitHub.

The `--js-file` option is also available for the `pdf`, `html`, `accessibility` and `har` commands.

2.7 Using JPEGs instead of PNGs

Screenshots default to PNG. Use a `-o` filename ending in `.jpg` or `.jpeg` to save as JPEG, or `.webp` to save as WebP.

You can also use `--quality X` to save as a JPEG with the specified quality, in order to reduce the filesize. 80 is a good value to use here:

```
shot-scraper https://simonwillison.net/ \
-h 800 -o simonwillison.jpg --quality 80
```

```
% ls -lah simonwillison.jpg
-rw-r--r--@ 1 simon staff 168K Mar  9 13:53 simonwillison.jpg
```

2.8 Using WebP

Save a screenshot as WebP by specifying a filename ending in `.webp`:

```
shot-scraper https://simonwillison.net/ -h 800 -o simonwillison.webp
```

WebP screenshots are lossless by default, equivalent to `--quality 100`. Use a lower quality to apply lossy compression:

```
shot-scraper https://simonwillison.net/ \
-h 800 -o simonwillison.webp --quality 80
```

WebP also works with selectors:

```
shot-scraper https://simonwillison.net/ -s '#bighead' -o bighead.webp
```

WebP images are limited to 16,383 pixels in either dimension, so use `--height` or a selector when capturing a page that exceeds that limit.

2.9 Specifying the image format

Use `--format png`, `--format jpeg` or `--format webp` to select the format explicitly. This takes precedence over the output filename extension.

Without `-o`, the automatic filename uses the selected format. This writes to `datasette-io.webp`:

```
shot-scraper https://datasette.io/ --format webp
```

Use `--format` with `-o -` to write WebP or JPEG to standard output:

```
shot-scraper https://datasette.io/ --format webp -o - > datasette.webp
```

The `--quality` option accepts values from 0 to 100 for JPEG and WebP. Combining `--format png` with `--quality` is an error because PNG does not use a quality setting.

2.10 Device scale factor

The `--scale-factor` option sets a specific device scale factor, which effectively simulates different device pixel ratios. This setting is useful for testing high-definition displays or emulating screens with various pixel densities.

For example, setting `--scale-factor 3` results in an image with a CSS pixel ratio of 3, which is ideal for emulating a high-resolution display, such as Apple iPhone 12 screens.

To take a screenshot with a scale factor of 3 (tripled resolution), run the following command:

```
shot-scraper https://simonwillison.net/ -o simon.png \  
  --width 390 --height 844 --scale-factor 3
```

This will multiply both the width and height of the screenshot by 3, resulting in an image that is 1170px wide and 2532px high, matching the iPhone 12's screen.

The `--scale-factor` option takes a positive float as input. For example, setting `--scale-factor 2.625` simulates the Google Pixel 6's CSS pixel ratio.

2.11 Retina images

The `--retina` option is a shortcut to set a device scale factor of 2. This means that an image will have its resolution effectively doubled, emulating the display of images on [retina](#) or higher pixel density screens.

```
shot-scraper https://simonwillison.net/ -o simon.png \  
  --width 400 --height 600 --retina
```

This example will produce an image that is 800px wide and 1200px high.

Note: The `--retina` option should not be used in conjunction with the `--scale-factor` flag as they are mutually exclusive. If both are provided, the command will raise an error to prevent conflicts.

2.12 Transparent background

The `--omit-background` option instructs the browser to ignore the default background, allowing for the capture of a page with a transparent background. The default Chromium browser supports this for PNG and WebP images. JPEG does not support transparency.

```
shot-scraper https://simonwillison.net/ -o simon.png \  
  --width 400 --height 600 --omit-background
```

WebP can preserve transparency when using `--quality`:

```
shot-scraper https://simonwillison.net/ -o simon.webp \  
  --width 400 --height 600 --omit-background --quality 80
```

2.13 Interacting with the page

Sometimes it's useful to be able to manually interact with a page before the screenshot is captured.

Add the `--interactive` option to open a browser window that you can interact with. Then hit `<enter>` in the terminal when you are ready to take the shot and close the window.

```
shot-scraper https://simonwillison.net/ -o after-interaction.png \  
--height 800 --interactive
```

This will output:

```
Hit <enter> to take the shot and close the browser window:  
# And after you hit <enter>...  
Screenshot of 'https://simonwillison.net/' written to 'after-interaction.png'
```

2.14 Logging all requests

It can sometimes be useful to see a list of all of the requests that the browser made while it was rendering a page.

Use `--log-requests` to output newline-delimited JSON representing each request, including requests for images and other assets.

Pass `-` to output the list to standard output, or use a filename to write to a file on disk.

The output looks like this:

```
shot-scraper http://datasette.io/ --log-requests -
```

```
{"method": "GET", "url": "http://datasette.io/", "status": 302, "size": null, "timing": {  
  ↪ "startTime": 1663211674984.7068, "domainLookupStart": 0.698, "domainLookupEnd": 1.897,  
  ↪ "connectStart": 1.897, "secureConnectionStart": -1, "connectEnd": 18.726, "requestStart  
  ↪ ": 18.86, "responseStart": 99.75, "responseEnd": 101.75000000162981}}  
{"method": "GET", "url": "https://datasette.io/", "status": 200, "size": 34592, "timing"  
  ↪ ": {"startTime": 1663211675085.51, "domainLookupStart": 0.187, "domainLookupEnd": 0.  
  ↪ 197, "connectStart": 0.197, "secureConnectionStart": 15.719, "connectEnd": 63.854,  
  ↪ "requestStart": 64.098, "responseStart": 390.231, "responseEnd": 399.268}}  
{"method": "GET", "url": "https://datasette.io/static/site.css", "status": 200, "size": 3  
  ↪ 3952, "timing": {"startTime": 1663211675486.027, "domainLookupStart": -1,  
  ↪ "domainLookupEnd": -1, "connectStart": -1, "secureConnectionStart": -1, "connectEnd": -  
  ↪ 1, "requestStart": 0.408, "responseStart": 99.407, "responseEnd": 100.433}}  
...
```

Note that the `size` field here will be the size of the response in bytes, but in some circumstances this will not be available and it will be returned as `"size": null`.

2.15 Browser arguments

Additional arguments to pass to the browser instance. The list of Chromium flags can be found [here](#).

For example, to remove font render hinting:

```
shot-scraper https://simonwillison.net/ -o no-hinting.png \
  --height 800 --browser-arg "--font-render-hinting=none"
```

To add multiple arguments, add `--browser-arg` for each argument:

```
shot-scraper https://simonwillison.net/ -o no-hinting-no-gpu.png \
  --height 800 --browser-arg "--font-render-hinting=none" --browser-arg "--disable-gpu"
```

2.16 Taking screenshots of local HTML files

You can pass the path to an HTML file on disk to take a screenshot of that rendered file:

```
shot-scraper index.html -o index.png
```

CSS and images referenced from that file using relative paths will also be included.

2.17 Tips for executing JavaScript

If you are using the `--javascript` option to execute code, that code will be executed after the page load event has fired but before the screenshot is taken.

You can use that code to do things like hide or remove specific page elements, click on links to open menus, or even add annotations to the page such as this [pink arrow example](#).

This code hides any element with a `[data-ad-rendered]` attribute and the element with `id="ensNotifyBanner"`:

```
document.querySelectorAll(
  '[data-ad-rendered],#ensNotifyBanner'
).forEach(el => el.style.display = 'none')
```

You can execute that like so:

```
shot-scraper https://www.latimes.com/ -o latimes.png --javascript "
document.querySelectorAll(
  '[data-ad-rendered],#ensNotifyBanner'
).forEach(el => el.style.display = 'none')
"
```

In some cases you may need to add a pause that executes during your custom JavaScript before the screenshot is taken - for example if you click on a button that triggers a short fading animation.

You can do that using the following pattern:

```
new Promise(takeShot => {
  // Your code goes here
  // ...
```

(continues on next page)

(continued from previous page)

```

setTimeout(() => {
  // Resolving the promise takes the shot
  takeShot();
}, 1000);
});

```

If your custom code defines a `Promise`, `shot-scraper` will wait for that promise to complete before taking the screenshot. Here the screenshot does not occur until the `takeShot()` function is called.

If you see errors relating to CSP headers such as “Failed to fetch dynamically imported module” you can work around them using *the `-bypass-csp` option*.

2.18 Viewing console.log() output

Almost all of the `shot-scraper` commands accept a `--log-console` option, which will cause them to output any calls to `console.log()` to standard error while the command is running.

This includes both `console.log()` calls in the existing page JavaScript, as well as any calls to that method that you include in your own custom JavaScript.

For example, running `--log-console` while taking a screenshot of the Facebook homepage will show this warning message, which Facebook logs to the developer tools console to help protect people from social engineering attacks:

```

% shot-scraper shot facebook.com --log-console

.d8888b. 888 888
d88P Y88b 888 888
Y88b. 888 888
"Y888b. 888888 .d88b. 88888b. 888 This is a browser feature intended for
"Y88b. 888 d88""88b 888 "88b 888 developers. If someone told you to copy-paste
"888 888 888 888 888 888 Y8P something here to enable a Facebook feature
Y88b d88P Y88b. Y88..88P 888 d88P or "hack" someone's account, it is a
"Y8888P" "Y888 "Y88P" 88888P" 888 scam and will give them access to your
888 Facebook account.
888
888
888

```

See <https://www.facebook.com/selfxss> for more information.

Screenshot of `'http://facebook.com'` written to `'facebook-com.png'`

2.19 shot-scraper shot --help

Full `--help` for this command:

```
Usage: shot-scraper shot [OPTIONS] URL
```

Take a single screenshot of a page or portion of a page.

Usage:

(continues on next page)

(continued from previous page)

```
shot-scraper www.example.com
```

This will write the screenshot to `www-example-com.png`

Use `"-o"` to write to a specific file:

```
shot-scraper https://www.example.com/ -o example.png
```

You can also **pass** a path to a local file on disk:

```
shot-scraper index.html -o index.png
```

Using `"-o -"` will output to standard out:

```
shot-scraper https://www.example.com/ -o - > example.png
```

Use `-s` to take a screenshot of one area of the page, identified using one **or** more CSS selectors:

```
shot-scraper https://simonwillison.net -s '#bighead'
```

Options:

<code>-a, --auth FILENAME</code>	Path to JSON authentication context file
<code>-w, --width INTEGER</code>	Width of browser window, defaults to 1280
<code>-h, --height INTEGER</code>	Height of browser window and shot - defaults to the full height of the page
<code>-o, --output FILE</code>	
<code>-s, --selector TEXT</code>	Take shot of first element matching this CSS selector
<code>--selector-all TEXT</code>	Take shot of all elements matching this CSS selector
<code>--js-selector TEXT</code>	Take shot of first element matching this JS (el) expression
<code>--js-selector-all TEXT</code>	Take shot of all elements matching this JS (el) expression
<code>-p, --padding INTEGER</code>	When using selectors, add this much padding in pixels
<code>-j, --javascript TEXT</code>	Execute this JS prior to taking the shot
<code>--js-file TEXT</code>	Read JavaScript to execute from this file, use - for stdin or <code>gh:username/script</code> to load from github.com/username/shot-scraper-scripts/script.js
<code>--scale-factor FLOAT</code>	Device scale factor. Cannot be used together with <code>'--retina'</code> .
<code>--retina</code>	Use device scale factor of 2. Cannot be used together with <code>'--scale-factor'</code> .
<code>--omit-background</code>	Omit the default browser background from the shot, making it possible take advantage of transparency. Does not work with JPEGs.
<code>--format [png jpeg webp]</code>	Image format . Overrides the output extension; use with <code>-o - for</code> stdout.

(continues on next page)

(continued from previous page)

```

--quality INTEGER RANGE      JPEG or WebP quality, e.g. 80. Defaults to
                             JPEG unless WebP is selected. WebP defaults to
                             100 (lossless).  [0<=x<=100]

--wait INTEGER               Wait this many milliseconds before taking the
                             screenshot

--wait-for TEXT              Wait until this JS expression returns true

--timeout INTEGER            Wait this many milliseconds before failing

-i, --interactive            Interact with the page in a browser before
                             taking the shot

--devtools                  Interact mode with developer tools

--log-requests FILENAME      Log details of all requests to this file

--log-console                Write console.log() to stderr

-b, --browser [chromium|firefox|webkit|chrome|chrome-beta]
                             Which browser to use

--browser-arg TEXT           Additional arguments to pass to the browser

--user-agent TEXT            User-Agent header to use

--reduced-motion              Emulate 'prefers-reduced-motion' media feature

--fail                       Fail with an error code if a page returns an
                             HTTP error

--skip                       Skip pages that return HTTP errors

--bypass-csp                 Bypass Content-Security-Policy

--silent                     Do not output any messages

--auth-password TEXT          Password for HTTP Basic authentication

--auth-username TEXT          Username for HTTP Basic authentication

--help                       Show this message and exit.

```

WEBSITES THAT NEED AUTHENTICATION

If you want to take screenshots of a site that has some form of authentication, you will first need to authenticate with that website manually.

You can do that using the `shot-scraper auth` command:

```
shot-scraper auth \
  https://datasette-auth-passwords-demo.datasette.io/-/login \
  auth.json
```

(For this demo, use username = `root` and password = `password!`)

This will open a browser window on your computer showing the page you specified.

You can then sign in using that browser window - including 2FA or CAPTCHAs or other more complex form of authentication.

When you are finished, hit <enter> at the `shot-scraper` command-line prompt. The browser will close and the authentication credentials (usually cookies) for that browser session will be written out to the `auth.json` file.

To take authenticated screenshots you can then use the `-a` or `--auth` options to point to the JSON file that you created:

```
shot-scraper https://datasette-auth-passwords-demo.datasette.io/ \
  -a auth.json -o authed.png
```

3.1 shot-scraper auth --help

Full `--help` for `shot-scraper auth`:

```
Usage: shot-scraper auth [OPTIONS] URL CONTEXT_FILE
```

Open a browser so user can manually authenticate **with** the specified site, then save the resulting authentication context to a file.

Usage:

```
shot-scraper auth https://github.com/ auth.json
```

Options:

```
-b, --browser [chromium|firefox|webkit|chrome|chrome-beta]
```

Which browser to use

```
--browser-arg TEXT
```

Additional arguments to **pass** to the browser

(continues on next page)

(continued from previous page)

<code>--user-agent TEXT</code>	User-Agent header to use
<code>--devtools</code>	Open browser DevTools
<code>--log-console</code>	Write <code>console.log()</code> to stderr
<code>--help</code>	Show this message and exit.

TAKING MULTIPLE SCREENSHOTS

You can configure multiple screenshots using a YAML file. Create a file called `shots.yml` that looks like this:

```
- output: example.com.png
  url: http://www.example.com/
- output: w3c.org.png
  url: https://www.w3.org/
```

Then run the tool like so:

```
shot-scraper multi shots.yml
```

This will create two image files, `www-example-com.png` and `w3c.org.png`, containing screenshots of those two URLs.

Use `-` to pass in YAML from standard input:

```
echo "- url: http://www.example.com" | shot-scraper multi -
```

If you run the tool with the `-n` or `--no-clobber` option any shots where the output file already exists will be skipped.

You can specify a subset of screenshots to take by specifying output files that you would like to create. For example, to take just the shots of `one.png` and `three.png` that are defined in `shots.yml` run this:

```
shot-scraper multi shots.yml -o one.png -o three.png
```

The `url:` can be set to a path to a file on disk as well:

```
- output: index.png
  url: index.html
```

Use the `--scale-factor` option to capture all screenshots at a specific scale factor, which effectively simulates different device pixel ratios. This setting is useful for testing high-definition displays or emulating screens with various pixel densities.

For example, setting `--scale-factor 3` results in screenshots with a CSS pixel ratio of 3, which is ideal for emulating a high-resolution display, such as Apple's iPhone 12 screens.

To take screenshots with a scale factor of 3 (tripled resolution), run the following command:

```
shot-scraper multi shots.yml --scale-factor 3
```

This will multiply both the width and height of all screenshots by 3, resulting in images with a higher level of detail, suitable for scenarios where you need to capture the screen as it would appear on a high-DPI display.

Use `--retina` to take all screenshots at retina resolution instead, doubling the dimensions of the files:

```
shot-scraper multi shots.yml --retina
```

Note: The `--retina` option should not be used in conjunction with the `--scale-factor` flag as they are mutually exclusive. If both are provided, the command will raise an error to prevent conflicts.

To take a screenshot of just the area of a page defined by a CSS selector, add `selector` to the YAML block:

```
- output: bighead.png
  url: https://simonwillison.net/
  selector: "#bighead"
```

You can pass more than one selector using a `selectors:` list. You can also use `padding:` to specify additional padding:

```
- output: bighead-multi-selector.png
  url: https://simonwillison.net/
  selectors:
    - "#bighead"
    - ".overband"
  padding: 20
```

You can use `selector_all:` to capture every element matching a selector, or `selectors_all:` to pass a list of such selectors:

```
- output: selectors-all.png
  url: https://simonwillison.net/
  selectors_all:
    - ".day"
    - ".entry:nth-of-type(1)"
  padding: 20
```

The `--js-selector` and `--js-selector-all` options can be provided using the `js_selector:`, `js_selectors:`, `js_selector_all:` and `js_selectors_all:` keys:

```
- output: js-selector-all.png
  url: https://github.com/simonw/shot-scraper
  js_selector: |-
    el.tagName == "P" && el.innerText.includes("shot-scraper")
  padding: 20
```

To execute JavaScript after the page has loaded but before the screenshot is taken, add a `javascript` key:

```
- output: bighead-pink.png
  url: https://simonwillison.net/
  selector: "#bighead"
  javascript: |
    document.body.style.backgroundColor = 'pink'
```

Use `js_file` in place of `javascript` to load the JavaScript from a file - useful for scripts that are too large to include directly in the YAML:

```
- output: bighead-modified.png
  url: https://simonwillison.net/
  js_file: remove-banners.js
```

`js_file` also accepts `gh:username/script` to load a script from `github.com/username/shot-scraper-scripts/script.js`.

You can include desired `height`, `width`, `quality`, `wait` and `wait_for` options on each item as well:

```
- output: simon-narrow.jpg
  url: https://simonwillison.net/
  width: 400
  height: 800
  quality: 80
  wait: 500
  wait_for: document.querySelector('#bighead')
```

To save a WebP image, use an output filename ending in `.webp`. WebP is lossless by default; add `quality` from 0 to 99 for lossy compression:

```
- output: simon.webp
  url: https://simonwillison.net/
  height: 800
- output: bighead.webp
  url: https://simonwillison.net/
  selector: "#bighead"
  quality: 80
```

You can also specify `format`: `png`, `format: jpeg` or `format: webp` on each item. This takes precedence over the output filename extension:

```
- output: screenshot
  url: https://simonwillison.net/
  format: webp
  quality: 80
```

Without `format` or a `.webp` output filename, `quality` selects JPEG. Combining `format: png` with `quality` is an error. See [Taking a screenshot](#) for more about image formats and quality.

4.1 Recording to an HTTP Archive

Similar to the [shot-scraper har command](#), `shot-scraper multi` can optionally record HTTP Archive files of the requests made during a session.

Add the `--har` flag to record all requests and responses to a `trace.har` JSON file, or `--har-zip` for a `trace.har.zip` file. Use `--har-file filename.har` to provide a path to a custom filename - this will be recorded as JSON or zip depending on the file extension.

If you are running this against a larger number of pages you should use `--har-zip` to avoid the JSON HAR file growing so large that it causes Playwright to crash.

This example:

```
shot-scraper multi shots.yml --har
```

Will output something like this:

```
Screenshot of 'http://www.example.com/' written to 'example.com.png'
Screenshot of 'https://www.w3.org/' written to 'w3c.org.png'
Wrote to HAR file: trace.har
```

When writing to a HAR you can omit the `output:` key in a YAML file to skip taking a screenshot of that file. This `shots.yml` file for example:

```
- url: https://example.com/
- url: https://datasette.io/
```

When run like this:

```
shot-scraper multi shots.yml --har-zip
```

Will produce this output, recording a HAR without taking any screenshots:

```
Skipping screenshot of 'https://example.com/'
Skipping screenshot of 'https://datasette.io/'
Wrote to HAR file: trace.har.zip
```

4.2 Running a server for the duration of the session

If you need to run a server for the duration of the `shot-scraper multi` session you can specify that using a `server:` block, like this:

```
- server: python -m http.server 8000
```

The `server:` key also accepts a list of arguments:

```
- server:
  - python
  - -m
  - http.server
  - 8000
```

With that server configured, you can now take screenshots of `http://localhost:8000/` and any other URLs hosted by that server:

```
- output: index.png
  url: http://localhost:8000/
```

Before taking the first screenshot after starting a server, `shot-scraper` will wait for up to 30 seconds for that screenshot's URL to start accepting connections - so servers that are slow to start up will still work. If the server process exits with an error before it starts listening the command will fail with a message describing what happened.

The server process will be automatically terminated when the `shot-scraper multi` command completes, unless you pass the `--leave-server` option to `shot-scraper multi` in which case it will be left running - you can terminate it using `kill PID` with the PID displayed in the console output.

4.3 Running custom code between steps

If you are taking screenshots of a single application, you may find it useful to run additional steps between shots that modify that application in some way.

You can do that using the `sh:` or `python:` keys. These can specify shell commands or Python code to run before taking the screenshot:

```
- sh: echo "Hello from shell" > index.html
  output: from-shell.png
  url: http://localhost:8000/
```

You can also specify a list of shell arguments like this:

```
- sh:
  - curl
  - -o
  - index.html
  - https://www.example.com/
  output: example.png
  url: http://localhost:8000/
```

If you specify these steps without a `url:` key they will still execute as individual task executions, without also taking a screenshot:

```
- sh: echo "hello world" > index.html
- python: |
    content = open("index.html").read()
    open("index.html", "w").write(content.upper())
```

If a `sh:` or `python:` command exits with a non-zero status, `shot-scraper multi` stops and exits with an error.

For multi-line `sh:` | blocks, use `set -e` if you want the shell to stop at the first failing command.

4.4 shot-scraper multi --help

Full `--help` for this command:

```
Usage: shot-scraper multi [OPTIONS] CONFIG

Take multiple screenshots, defined by a YAML file

Usage:

    shot-scraper multi config.yml

Where config.yml contains configuration like this:

    - output: example.png
      url: http://www.example.com/

For full YAML syntax documentation, see:
https://shot-scraper.datasette.io/en/stable/multi.html
```

(continues on next page)

(continued from previous page)

Options:

-a, --auth FILENAME	Path to JSON authentication context file
--scale-factor FLOAT	Device scale factor. Cannot be used together with '--retina'.
--retina	Use device scale factor of 2. Cannot be used together with '--scale-factor'.
--timeout INTEGER	Wait this many milliseconds before failing
-n, --no-clobber	Skip images that already exist
-o, --output TEXT	Just take shots matching these output files
-b, --browser [chromium firefox webkit chrome chrome-beta]	Which browser to use
--browser-arg TEXT	Additional arguments to pass to the browser
--user-agent TEXT	User-Agent header to use
--reduced-motion	Emulate 'prefers-reduced-motion' media feature
--log-console	Write console.log() to stderr
--fail	Fail with an error code if a page returns an HTTP error
--skip	Skip pages that return HTTP errors
--silent	Do not output any messages
--auth-password TEXT	Password for HTTP Basic authentication
--auth-username TEXT	Username for HTTP Basic authentication
--leave-server	Leave servers running when script finishes
--har	Save all requests to trace.har file
--har-zip	Save all requests to trace.har.zip file
--har-file FILE	Path to HAR file to save all requests
--help	Show this message and exit.

SCRAPING PAGES USING JAVASCRIPT

The `shot-scraper javascript` command can be used to execute JavaScript directly against a page and return the result as JSON.

This command doesn't produce a screenshot, but has interesting applications for scraping.

To retrieve a string title of a document:

```
shot-scraper javascript https://datasette.io/ "document.title"
```

This returns a JSON string:

```
"Datasette: An open source multi-tool for exploring and publishing data"
```

To return a raw string instead, use the `-r` or `--raw` options:

```
shot-scraper javascript https://datasette.io/ "document.title" -r
```

Output:

```
Datasette: An open source multi-tool for exploring and publishing data
```

5.1 Setting the browser viewport

The page used to run JavaScript defaults to a 1280x720 viewport. You can change that using `--width` and `--height`:

```
shot-scraper javascript https://datasette.io/ \  
  --width 390 \  
  --height 844 \  
  "({width: window.innerWidth, height: window.innerHeight})"
```

To return a JSON object, wrap an object literal in parenthesis:

```
shot-scraper javascript https://datasette.io/ "{\  
  title: document.title,  
  tagline: document.querySelector('.tagline').innerText  
}"
```

This returns:

```
{
  "title": "Datasette: An open source multi-tool for exploring and publishing data",
  "tagline": "An open source multi-tool for exploring and publishing data"
}
```

5.2 Running more than one statement

You can use `() => { ... }` function syntax to run multiple statements, returning a result at the end of your function.

This example raises an error if no paragraphs are found.

```
shot-scraper javascript https://www.example.com/ "
() => {
  var paragraphs = document.querySelectorAll('p');
  if (paragraphs.length == 0) {
    throw 'No paragraphs found';
  }
  return Array.from(paragraphs, el => el.innerText);
}"
```

5.3 Using async/await

You can pass an `async` function if you want to use `await`, including to import modules from external URLs. This example loads the [Readability.js](#) library from [jsdelivr](#) and uses it to extract the core content of a page:

```
shot-scraper javascript \
  https://simonwillison.net/2022/Mar/14/scraping-web-pages-shot-scraper/ "
async () => {
  const readability = await import('https://cdn.jsdelivr.net/npm/@mozilla/readability@0.
↪6.0/+esm');
  return (new readability.Readability(document)).parse();
}"
```

To use functions such as `setInterval()`, for example if you need to delay the shot for a second to allow an animation to finish running, return a promise:

```
shot-scraper javascript datasette.io "
new Promise(done => setInterval(
  () => {
    done({
      title: document.title,
      tagline: document.querySelector('.tagline').innerText
    });
  }, 1000
));"
```

5.4 Bypassing Content Security Policy headers

Some websites use [Content Security Policy](#) (CSP) headers to prevent additional JavaScript from executing on the page, as a security measure.

When using `shot-scraper` this can prevent some JavaScript features from working. You might see error messages that look like this:

```
shot-scraper javascript github.com "
  async () => {
    await import('https://cdn.jsdelivr.net/npm/left-pad/+esm');
    return 'content-security-policy ignored' }
"
```

Output:

```
Error: TypeError: Failed to fetch dynamically imported module:
https://cdn.jsdelivr.net/npm/left-pad/+esm
```

You can use the `--bypass-csp` option to have `shot-scraper` run the browser in a mode that ignores these headers:

```
shot-scraper javascript github.com "
  async () => {
    await import('https://cdn.jsdelivr.net/npm/left-pad/+esm');
    return 'content-security-policy ignored' }
" --bypass-csp
```

Output:

```
"content-security-policy ignored"
```

5.5 Using this for automated tests

If a JavaScript error occurs, a stack trace will be written to standard error and the tool will terminate with an exit code of 1.

This can be used to run JavaScript tests in continuous integration environments, by taking advantage of the `throw "error message"` JavaScript statement.

This example [uses GitHub Actions](#):

```
- name: Test page title
  run: |-
    shot-scraper javascript datasette.io "
      if (document.title !== 'Datasette') {
        throw 'Wrong title detected';
      }"
```

5.6 Running JavaScript from a file

You can also save JavaScript to a file and execute it like this:

```
shot-scraper javascript datasette.io -i script.js
```

Or read it from standard input like this:

```
echo "document.title" | shot-scraper javascript datasette.io
```

Or read it from standard input like this:

```
echo "document.title" | shot-scraper javascript datasette.io
```

5.7 Running JavaScript from GitHub

A special `gh:` prefix can be used to load scripts from GitHub.

You can use this with a full path to a `script.js` file in a public GitHub repository like this:

```
shot-scraper javascript datasette.io -i gh:simonw/shot-scraper-scripts/readability.js
```

Or by convention if the script lives in a repo called `shot-scraper-scripts` you can omit that (and the `.js` extension too) like this:

```
shot-scraper javascript datasette.io -i gh:simonw/readability
```

Both of these examples will execute `readability.js`, explained in the next section.

5.8 Example: Extracting page content with Readability.js

`Readability.js` is “a standalone version of the readability library used for Firefox Reader View.” It lets you parse the content on a web page and extract just the title, content, byline and some other key metadata.

The following recipe imports the library from the `jsdelivr` CDN, runs it against the current page and returns the results to the console as JSON:

```
shot-scraper javascript https://simonwillison.net/2022/Mar/24/datasette-061/ "
async () => {
  const readability = await import('https://cdn.jsdelivr.net/npm/@mozilla/readability@0.
  ↪6.0/+esm');
  return (new readability.Readability(document)).parse();
}"
```

The output looks like this:

```
{
  "title": "Datasette 0.61: The annotated release notes",
  "byline": null,
  "dir": null,
  "lang": "en-gb",
```

(continues on next page)

(continued from previous page)

```

"content": "<div id=\"readability-page-1\" class=\"page\"><div id=\"primary\">\n\n\n↪\n\n<p>I released ... <this is a very long string>",
"length": 8625,
"excerpt": "I released Datasette 0.61 this morning\u2014closely followed by 0.61.1,
↪to fix a minor bug. Here are the annotated release notes. In preparation for Datasette,
↪1.0, this release includes two potentially \u2026",
"siteName": null
}

```

See [Extracting web page content using Readability.js](#) and [shot-scraper](#) for more.

5.9 shot-scraper javascript -help

Full --help for this command:

```
Usage: shot-scraper javascript [OPTIONS] URL [JAVASCRIPT]
```

Execute JavaScript against the page **and return** the result **as** JSON

Usage:

```
shot-scraper javascript https://datasette.io/ "document.title"
```

To **return** a JSON **object**, use this:

```
"({title: document.title, location: document.location})"
```

To use setInterval() **or** similar, **pass** a promise:

```

"new Promise(done => setInterval(
  () => {
    done({
      title: document.title,
      h2: document.querySelector('h2').innerHTML
    });
  }, 1000
));"

```

If a JavaScript error occurs an exit code of 1 will be returned.

Options:

-i, --input TEXT	Read input JavaScript from this file or use gh:username/script to load from github.com/username/shot-scraper-scripts/script.js
-a, --auth FILENAME	Path to JSON authentication context file
-w, --width INTEGER	Width of browser window, defaults to 1280
-h, --height INTEGER	Height of browser window, defaults to 720
-o, --output FILENAME	Save output JSON to this file
-r, --raw	Output JSON strings as raw text
--timeout INTEGER	Wait this many milliseconds before failing

(continues on next page)

(continued from previous page)

```
-b, --browser [chromium|firefox|webkit|chrome|chrome-beta]
                                Which browser to use
--browser-arg TEXT              Additional arguments to pass to the browser
--user-agent TEXT               User-Agent header to use
--reduced-motion                Emulate 'prefers-reduced-motion' media feature
--log-console                   Write console.log() to stderr
--fail                          Fail with an error code if a page returns an
                                HTTP error
--skip                          Skip pages that return HTTP errors
--bypass-csp                    Bypass Content-Security-Policy
--auth-password TEXT            Password for HTTP Basic authentication
--auth-username TEXT            Username for HTTP Basic authentication
--help                          Show this message and exit.
```

RECORDING VIDEOS

The `shot-scraper video` command records a WebM video from a YAML storyboard.

Storyboards describe the video as a sequence of scenes. Each scene can open a page, wait for content, perform actions and pause between steps.

Create a file called `storyboard.yml` like this:

```
output: demo.webm
url: https://shot-scraper.datasette.io/en/stable/

viewport:
  width: 1280
  height: 720

cursor: true
wait_for: "text=Quick start"

scenes:
- name: Documentation home
  do:
    - pause: 1

- name: Open installation docs
  do:
    - click: ".sidebar-tree a[href='installation.html']"
    - wait_for: 'h1:has-text("Installation")'
    - screenshot: installation.png
    - pause: 1

- name: Search the docs
  do:
    - click: "input.sidebar-search"
    - type:
        into: "input.sidebar-search"
        text: "authentication"
        delay_ms: 25
    - press:
        selector: "input.sidebar-search"
        key: Enter
    - wait_for: "text=Search Results"
    - pause: 2
```

Then run:

```
shot-scraper video storyboard.yml
```

This opens the starting URL, records the scenes and writes the video to `demo.webm`.

Use `-o` or `--output` to override the output filename:

```
shot-scraper video storyboard.yml -o alternate.webm
```

Use `--mp4` to also convert the recorded WebM video to MP4 using `ffmpeg`. The WebM is still written first, then the MP4 is written using the same filename with the extension replaced by `.mp4`:

```
shot-scraper video storyboard.yml --mp4
```

If `ffmpeg` is not installed, the WebM file is still created but the command exits with a non-zero status and an error explaining that the MP4 was not created.

6.1 Storyboard structure

A storyboard file is a YAML mapping with these keys:

output

Filename for the recorded WebM video. This can be omitted if `-o` is used.

```
output: demo.webm
```

url

Starting URL for the video. This can be an `http://` or `https://` URL, a bare domain or a path to a local HTML file.

```
url: https://shot-scraper.datasette.io/en/stable/
```

sh

Optional shell command to run before `server:` starts and before the browser opens. If both top-level `sh:` and `python:` are present, `sh:` runs first. This can be a string, which is run through the shell, or a list of arguments, which is run directly.

```
sh: |
  set -e
  echo "Preparing storyboard files"
  date > /tmp/storyboard-started.txt
```

The shell process must exit with status `0`. For multi-line `sh: |` blocks, use `set -e` if you want the shell to stop at the first failing command.

python

Optional Python code to run before `server:` starts and before the browser opens. If both top-level `sh:` and `python:` are present, `python:` runs after `sh:`.

```
python: |
  from pathlib import Path
  root = Path("/tmp/demo-root")
  root.mkdir(parents=True, exist_ok=True)
  (root / "index.html").write_text("<h1>Local demo</h1>")
```

If a `sh:` or `python:` command exits with a non-zero status, `shot-scraper` video stops and exits with an error.

server

Optional command to run as a server for the duration of the storyboard recording. This can be a string, which is run through the shell, or a list of arguments, which is run directly. See [Running a server for the duration of the storyboard](#) for more details.

```
server: python -m http.server 8000
```

```
server:
- python
- -m
- http.server
- 8000
```

viewport

Optional browser viewport size. Defaults to 1280 by 720. Use a mapping with `width` and `height` values:

```
viewport:
  width: 1440
  height: 900
```

cursor

Set to `true` to show a cursor dot and click rings in the video. Set to `false` or omit it to leave the cursor hidden. Use a mapping to configure the cursor:

```
cursor:
  visible: true
  clicks: true
  color: "#ff4f00"
  size: 18
  click_size: 44
```

`visible` shows or hides the cursor dot. `clicks` shows or hides click rings. `color` is a CSS color for the cursor and rings. `size` is the cursor dot diameter in pixels. `click_size` is the click ring diameter in pixels.

wait

Seconds to pause after the starting page has loaded and before recording scenes. Use this when the page needs a fixed amount of time before the first scene starts.

```
wait: 0.5
```

wait_for

Selector to wait for after the starting page has loaded and before recording scenes. This uses [Playwright locator syntax](#), so CSS selectors and selectors such as `text=Quick start` are supported.

```
wait_for: "text=Quick start"
```

wait_for_url

URL string or glob pattern to wait for after the starting page has loaded and before recording scenes.

```
wait_for_url: "**/dashboard"
```

javascript

Optional JavaScript to run in the initial page after `url:`, `wait:`, `wait_for:` and `wait_for_url:` have completed, before scenes start. This runs inside the current Playwright page context.

```
javascript: |
  localStorage.setItem("theme", "dark");
  document.documentElement.dataset.storyboard = "true";
```

scenes

Required list of scenes to record. If you omit the top-level `url`, the first scene must define `open`.

```
scenes:
- name: Open docs
  open: https://shot-scraper.datasette.io/en/stable/
  wait_for: "text=Quick start"
  do:
  - pause: 1
```

6.2 Cursor and click visualization

Playwright videos do not show the system cursor. Add `cursor: true` to inject a visible cursor dot and click rings into the page while recording:

```
output: demo.webm
url: https://shot-scraper.datasette.io/en/stable/
cursor: true

scenes:
- name: Click installation link
  do:
  - click: ".sidebar-tree a[href='installation.html']"
  - wait_for: 'h1:has-text("Installation")'
  - pause: 1
```

You can also configure the cursor using these fields:

```
cursor:
  visible: true
  clicks: true
  color: "#ff4f00"
  size: 18
  click_size: 44
```

`visible` controls whether the cursor dot is shown. `clicks` controls whether click rings are shown. `color` is any CSS color value. `size` is the cursor dot diameter in pixels. `click_size` is the click ring diameter in pixels. Set `visible: false` to show click rings without the cursor dot.

6.3 Running a server for the duration of the storyboard

If you need to run a server for the duration of the `shot-scraper` video session, specify it using `server::`

```
output: demo.webm
python: |
    from pathlib import Path
    root = Path("/tmp/demo-root")
    root.mkdir(parents=True, exist_ok=True)
    (root / "index.html").write_text("<h1>Local demo</h1>")
server: python -m http.server 8000 --directory /tmp/demo-root
url: http://localhost:8000/
wait_for: h1

scenes:
- name: Home page
  do:
  - pause: 1
```

The `server:` key also accepts a list of arguments:

```
output: demo.webm
python: |
    from pathlib import Path
    root = Path("/tmp/demo-root")
    root.mkdir(parents=True, exist_ok=True)
    (root / "index.html").write_text("<h1>Local demo</h1>")
server:
- python
- -m
- http.server
- 8000
- --directory
- /tmp/demo-root
url: http://localhost:8000/
wait_for: h1

scenes:
- name: Home page
  do:
  - pause: 1
```

After starting the server, `shot-scraper` will wait for up to 30 seconds for the storyboard's `url:` to start accepting connections before recording begins - so servers that are slow to start up will still work. If the server process exits with an error before it starts listening the command will fail with a message describing what happened.

The server process will be automatically terminated when the video command completes, unless you pass `--leave-server`. In that case it will be left running, and the process ID will be displayed in the console output.

6.4 Scenes

Each scene can use these keys:

name

Optional label used in progress messages.

```
name: Search the docs
```

open

Navigate to a URL at the start of the scene. Relative URLs are resolved against the current page URL.

```
open: installation.html
```

wait_for

Wait for a selector before running the scene actions. This uses [Playwright locator syntax](#), so CSS selectors and selectors such as `text=Welcome` are both supported.

```
wait_for: 'h1:has-text("Installation")'
```

wait_for_url

Wait for the page URL to match a string or glob pattern supported by Playwright.

```
wait_for_url: "**/installation.html"
```

sh

Shell command to run before the scene opens a page or runs actions. This can be a string, which is run through the shell, or a list of arguments, which is run directly.

```
sh: echo "scene" > scene.txt
```

python

Python code to run before the scene opens a page or runs actions.

```
python: |
    open("scene.txt", "w").write("ok")
```

do

A list of [actions](#) to run. Actions run in the order listed, after `sh:`, `python:`, `open:`, `wait_for:` and `wait_for_url:` for the scene. Use a `pause` action at the end of this list to keep recording the final frame for a moment.

```
do:
- click: ".sidebar-tree a[href='installation.html']"
- wait_for: 'h1:has-text("Installation")'
- pause: 1
```

Example:

```
scenes:
- name: Search the docs
  open: https://shot-scraper.datasette.io/en/stable/
  wait_for: "input.sidebar-search"
  do:
  - type:
```

(continues on next page)

(continued from previous page)

```

    into: "input.sidebar-search"
    text: "authentication"
    delay_ms: 40
  - press:
    selector: "input.sidebar-search"
    key: Enter
  - wait_for: "text=Search Results"
  - pause: 1.5

```

6.5 Running custom code between steps

Storyboard scenes support the same `sh:` and `python:` keys as `shot-scraper multi`. These commands run before the scene opens a page or runs actions:

```

scenes:
- name: Build local page
  sh: echo "Hello from shell" > index.html
  open: index.html
  do:
  - pause: 1

```

If a scene-level or action-level `sh:` or `python:` command exits with a non-zero status, `shot-scraper video` stops and exits with an error.

You can also specify a list of shell arguments:

```

scenes:
- name: Fetch page
  sh:
  - curl
  - -L
  - -o
  - index.html
  - https://shot-scraper.datasette.io/en/stable/installation.html
  open: index.html

```

Use `python:` to run Python code before a scene:

```

scenes:
- name: Rewrite page
  python: |
    content = open("index.html").read()
    open("index.html", "w").write(content.upper())
  open: index.html

```

For commands between individual browser actions, use `sh:` or `python:` inside the `do:` list:

```

output: demo.webm
python: |
  from pathlib import Path
  root = Path("/tmp/demo-root")

```

(continues on next page)

(continued from previous page)

```

    root.mkdir(parents=True, exist_ok=True)
    (root / "index.html").write_text("<h1>First version</h1>")
server: python -m http.server 8000 --directory /tmp/demo-root
url: http://localhost:8000/
wait_for: 'h1:has-text("First version")'

scenes:
- name: Update then reload
  do:
    - sh: echo "<h1>Updated</h1>" > /tmp/demo-root/index.html
    - open: http://localhost:8000/
    - wait_for: 'h1:has-text("Updated")'

```

Use `javascript:` or `js:` inside `do:` to run code in the current Playwright page context. Unlike `sh:` and `python:`, this executes in the browser page, so it can read and modify the DOM, `localStorage` and other browser APIs:

```

scenes:
- name: Highlight the installation heading
  open: https://shot-scraper.datasette.io/en/stable/installation.html
  wait_for: 'h1:has-text("Installation")'
  do:
    - javascript: |
        document.querySelector("h1").style.outline = "4px solid red";
        localStorage.setItem("storyboard-mode", "demo");
    - screenshot: highlighted-installation.png

```

There is no scene-level `javascript:` key. To run page JavaScript during a scene, put it inside the scene's `do:` list.

6.6 Actions

Actions are single-key mappings in a scene's `do` list.

6.6.1 click

Click a selector. The string form is shorthand for a mapping with `selector:`.

```
- click: "button[aria-label='Menu']"
```

You can also provide click options. `button` can be `left`, `right` or `middle`. `count` is the number of clicks.

```
- click:
  selector: "button[aria-label='Menu']"
  button: left
  count: 2

```

6.6.2 type

Type text into an input, textarea or focused editable element. Use `into:` or `selector:` to identify the target; both names mean the same thing. `delay_ms` is optional and sets the milliseconds between keystrokes.

```
- type:
  into: "#search"
  text: "datasette"
  delay_ms: 50
```

6.6.3 fill

Fill a field immediately. Use `into:` or `selector:` to identify the target; both names mean the same thing.

```
- fill:
  into: "#email"
  text: "demo@example.com"
```

6.6.4 press

Press a key. The string form presses the key using the page keyboard, so it acts on whichever element is currently focused.

```
- press: Enter
```

Use the mapping form to send the key press to a specific selector:

```
- press:
  selector: "#search"
  key: Enter
```

6.6.5 scroll

Scroll by a number of pixels. The numeric shorthand scrolls vertically by that many pixels:

```
- scroll: 800
```

Use the mapping form for `x`, `y`, `to` and `duration`. `duration` is in seconds and enables smooth scrolling.

```
- scroll:
  y: 800
  duration: 1.2
```

Use `to` to scroll an element into view:

```
- scroll:
  to: "#pricing"
  duration: 1
```

6.6.6 pause

Pause for a number of seconds:

```
- pause: 0.5
```

6.6.7 wait_for

Wait for a selector using [Playwright locator syntax](#). CSS selectors and text selectors such as `text=Search Results` are supported.

```
- wait_for: ".loaded"
```

6.6.8 wait_for_url

Wait for the current URL to match a string or glob pattern:

```
- wait_for_url: "**/pricing"
```

6.6.9 open

Navigate during a scene. Relative URLs are resolved against the current page URL.

```
- open: /pricing
```

6.6.10 screenshot

Take a screenshot during the storyboard. The string form writes a viewport screenshot to that path.

```
- screenshot: step-2.png
```

Use the mapping form for output, selector and full_page. selector captures just that element.

```
- screenshot:  
  output: form.png  
  selector: "#signup-form"
```

Use full_page to capture the full page instead of just the current viewport:

```
- screenshot:  
  output: full-page.png  
  full_page: true
```

6.6.11 sh

Run a shell command. The string form is run through the shell.

```
- sh: echo "Updated" > index.html
```

Provide a list of arguments to run without a shell:

```
- sh:
  - touch
  - updated.html
```

6.6.12 python

Run Python code using the same Python executable that is running shot-scraper:

```
- python: |
  content = open("index.html").read()
  open("index.html", "w").write(content.upper())
```

6.6.13 javascript

Run JavaScript in the current Playwright page context. This can read and modify the DOM, localStorage and other browser APIs:

```
- javascript: |
  document.querySelector("h1").style.outline = "4px solid red";
```

The shorter js key is also supported:

```
- js: window.scrollTo(0, 0)
```

Use top-level javascript: for JavaScript that should run once after the initial page has loaded and before scenes start:

```
output: demo.webm
url: https://shot-scraper.datasette.io/en/stable/
javascript: |
  document.documentElement.dataset.storyboard = "true";
  document.body.style.backgroundColor = "#fffd7";

scenes:
- name: Page with prepared browser state
  wait_for: "text=Quick start"
  do:
  - js: document.querySelector("h1").textContent = "Storyboard demo";
  - pause: 1
```

6.7 Complete example

This example records a short walkthrough of the shot-scraper documentation site:

```
output: shot-scraper-docs-demo.webm
url: https://shot-scraper.datasette.io/en/stable/

viewport:
  width: 1280
  height: 720

cursor:
  visible: true
  clicks: true
  color: "#ff4f00"
  size: 18
  click_size: 44

wait_for: "text=Quick start"

scenes:
- name: Documentation home
  do:
    - pause: 1
- name: Open installation docs
  do:
    - click: ".sidebar-tree a[href='installation.html']"
    - wait_for: 'h1:has-text("Installation")'
    - screenshot: installation.png
    - pause: 1
- name: Search the docs
  do:
    - click: "input.sidebar-search"
    - type:
        into: "input.sidebar-search"
        text: "authentication"
        delay_ms: 25
    - press:
        selector: "input.sidebar-search"
        key: Enter
    - wait_for: "text=Search Results"
    - js: |
        document.body.style.outline = "4px solid #ff4f00";
    - screenshot: search-results.png
    - pause: 2
```

6.8 Command options

shot-scraper video supports the same browser selection, authentication, console logging, timeout, CSP bypass and HTTP Basic authentication options as the other browser-based commands.

Use `--silent` to hide progress messages. Use `--leave-server` to leave a configured server: process running after the command finishes.

Use `--mp4` to create an MP4 copy of the recorded WebM video. This requires `ffmpeg` to be installed. The command will then create both a `filename.webm` and `filename.mp4` file.

6.9 shot-scraper video --help

Full `--help` for this command:

Usage: shot-scraper video [OPTIONS] STORYBOARD_FILE

Record a WebM video from a YAML storyboard.

Common usage:

```
shot-scraper video storyboard.yml
shot-scraper video storyboard.yml -o demo.webm --mp4
```

A storyboard is a YAML mapping with an output filename, a starting URL (or an opening scene), and a list of scenes. Each scene can wait, run commands, run browser actions, and pause between steps.

Example storyboard.yml:

```
output: demo.webm
url: https://shot-scraper.datasette.io/en/stable/
viewport:
  width: 1280
  height: 720
cursor: true
wait_for: "text=Quick start"
scenes:
- name: Documentation home
  do:
    - pause: 1
- name: Open installation docs
  do:
    - click: ".sidebar-tree a[href='installation.html']"
    - wait_for: 'h1:has-text("Installation")'
    - screenshot: installation.png
    - pause: 1
- name: Search the docs
  do:
    - click: "input.sidebar-search"
    - type:
      into: "input.sidebar-search"
```

(continues on next page)

(continued from previous page)

```

    text: "authentication"
    delay_ms: 25
  - press:
    selector: "input.sidebar-search"
    key: Enter
  - wait_for: "text=Search Results"
  - pause: 2

```

Top-level YAML keys:

output: WebM filename. `-o/--output` overrides this. With `--mp4`, an MP4 **is** also written using the same filename **with** the suffix replaced by `.mp4`.

url: Starting URL, bare domain, **or** local HTML path. Omit this only **if** the first scene has **open**.

sh: Shell command string **or** argument **list** to run before python: **and** server:.

python: Python code to run after sh: **and** before server:.

server: Optional command string **or** argument **list** to run **while** recording.

viewport: Mapping **with** width: **and** height:. Defaults to 1280 by 720.

cursor: true, false, **or** a mapping **with** visible, clicks, color, size **and** click_size.

wait: Seconds to pause after the starting page loads.

wait_for: Selector **or** Playwright text selector to wait **for**.

wait_for_url: URL pattern to wait **for**.

javascript: JavaScript to run before scene recording starts.

scenes: Required **list** of scenes.

Scene YAML keys:

name: Label shown **in** progress output.

open: URL/path to **open** at the start of this scene.

wait_for: Selector to wait **for**.

wait_for_url: URL pattern to wait **for**.

sh: Shell command string **or** argument **list** to run before actions.

python: Python code to run before actions.

do: List of browser/page actions.

Actions **for** a scene's **do**: **list**:

```

- click: "selector"
- click: {selector: "selector", button: right, count: 2}
- fill: {into: "selector", text: "value"}
- type: {into: "selector", text: "value", delay_ms: 25}
- press: {selector: "selector", key: "ControlOrMeta+A"}
- scroll: {x: 0, y: 500, duration: 0.5}
- scroll: {to: "selector", duration: 0.5}
- pause: 1.5
- wait_for: "selector"
- wait_for_url: "**/finished"
- open: "installation.html"
- js: "document.body.dataset.demo = '1'"

```

(continues on next page)

(continued from previous page)

```

- screenshot: output.png
- screenshot: {output: heading.png, selector: "h1"}
- sh: "echo scene > scene.txt"
- python: "open('scene.txt', 'w').write('ok')"

```

For full YAML syntax documentation, see:

<https://shot-scraper.datasette.io/en/stable/video.html>

Options:

<code>-o, --output FILE</code>	Output video filename (.webm), overriding output: in the storyboard
<code>-a, --auth FILENAME</code>	Path to JSON authentication context file
<code>--timeout INTEGER</code>	Wait this many milliseconds before failing
<code>-b, --browser [chromium firefox webkit chrome chrome-beta]</code>	Which browser to use
<code>--browser-arg TEXT</code>	Additional arguments to pass to the browser
<code>--user-agent TEXT</code>	User-Agent header to use
<code>--reduced-motion</code>	Emulate ' prefers-reduced-motion ' media feature
<code>--log-console</code>	Write console.log() to stderr
<code>--fail</code>	Fail with an error code if a page returns an HTTP error
<code>--skip</code>	Skip pages that return HTTP errors
<code>--bypass-csp</code>	Bypass Content-Security-Policy
<code>--silent</code>	Do not output any messages
<code>--auth-password TEXT</code>	Password for HTTP Basic authentication
<code>--auth-username TEXT</code>	Username for HTTP Basic authentication
<code>--leave-server</code>	Leave servers running when script finishes
<code>--mp4</code>	Also convert the recorded WebM video to MP4 using ffmpeg
<code>--help</code>	Show this message and exit.

SAVING A WEB PAGE TO PDF

The `shot-scraper pdf` command saves a PDF version of a web page - the equivalent of using `Print -> Save to PDF` in Chromium.

```
shot-scraper pdf https://datasette.io/
```

This will save to `datasette-io.pdf`. You can use `-o` to specify a filename:

```
shot-scraper pdf https://datasette.io/tutorials/learn-sql \
-o learn-sql.pdf
```

You can pass the path to a local file on disk instead of a URL:

```
shot-scraper pdf invoice.html -o invoice.pdf
```

7.1 `shot-scraper pdf --help`

Full `--help` for this command:

```
Usage: shot-scraper pdf [OPTIONS] URL
```

Create a PDF of the specified page

Usage:

```
shot-scraper pdf https://datasette.io/
```

Use `-o` to specify a filename:

```
shot-scraper pdf https://datasette.io/ -o datasette.pdf
```

You can **pass** a path to a file instead of a URL:

```
shot-scraper pdf invoice.html -o invoice.pdf
```

Options:

<code>-a, --auth FILENAME</code>	Path to JSON authentication context file
<code>-o, --output FILE</code>	
<code>-j, --javascript TEXT</code>	Execute this JS prior to creating the PDF
<code>--js-file TEXT</code>	Read JavaScript to execute from this file, use

(continues on next page)

(continued from previous page)

```

- for stdin or gh:username/script to load from
  github.com/username/shot-scraper-
  scripts/script.js
--wait INTEGER          Wait this many milliseconds before taking the
                        screenshot
--wait-for TEXT        Wait until this JS expression returns true
--timeout INTEGER      Wait this many milliseconds before failing
--media-screen         Use screen rather than print styles
--landscape            Use landscape orientation
--format [letter|legal|tabloid|ledger|a0|a1|a2|a3|a4|a5|a6]
                        Which standard paper size to use
--width TEXT           PDF width including units, e.g. 10cm
--height TEXT          PDF height including units, e.g. 10cm
--scale FLOAT RANGE    Scale of the webpage rendering [0.1<=x<=2.0]
--print-background    Print background graphics
--log-console          Write console.log() to stderr
--fail                Fail with an error code if a page returns an
                        HTTP error
--skip                Skip pages that return HTTP errors
--bypass-csp           Bypass Content-Security-Policy
--silent              Do not output any messages
--auth-password TEXT   Password for HTTP Basic authentication
--auth-username TEXT   Username for HTTP Basic authentication
--help                Show this message and exit.

```

DUMPING THE HTML OF A PAGE

The `shot-scraper html` command dumps out the final HTML of a page after all JavaScript has run.

```
shot-scraper html https://datasette.io/
```

Use `-o filename.html` to write the output to a file instead of displaying it.

```
shot-scraper html https://datasette.io/ -o index.html
```

Add `--javascript SCRIPT` to execute custom JavaScript before taking the HTML snapshot.

```
shot-scraper html https://datasette.io/ \  
  --javascript "document.querySelector('h1').innerText = 'Hello, world!'"
```

8.1 Retrieving the HTML for a specific element

You can use the `-s SELECTOR` option to capture just the HTML for one specific element on the page, identified using a CSS selector:

```
shot-scraper html https://datasette.io/ -s h1
```

This outputs:

```
<h1>  
    
</h1>
```

8.2 `shot-scraper html --help`

Full `--help` for this command:

```
Usage: shot-scraper html [OPTIONS] URL  
  
  Output the final HTML of the specified page  
  
  Usage:  
  
    shot-scraper html https://datasette.io/
```

(continues on next page)

(continued from previous page)

Use `-o` to specify a filename:

```
shot-scraper html https://datasette.io/ -o index.html
```

Options:

<code>-a, --auth FILENAME</code>	Path to JSON authentication context file
<code>-o, --output FILE</code>	
<code>-j, --javascript TEXT</code>	Execute this JS prior to saving the HTML
<code>--js-file TEXT</code>	Read JavaScript to execute from this file, use - for stdin or gh:username/script to load from github.com /username/shot-scraper- scripts/script.js
<code>-s, --selector TEXT</code>	Return outerHTML of first element matching this CSS selector
<code>--wait INTEGER</code>	Wait this many milliseconds before taking the snapshot
<code>--timeout INTEGER</code>	Wait this many milliseconds before failing
<code>--log-console</code>	Write console.log() to stderr
<code>-b, --browser [chromium firefox webkit chrome chrome-beta]</code>	Which browser to use
<code>--browser-arg TEXT</code>	Additional arguments to pass to the browser
<code>--user-agent TEXT</code>	User-Agent header to use
<code>--fail</code>	Fail with an error code if a page returns an HTTP error
<code>--skip</code>	Skip pages that return HTTP errors
<code>--bypass-csp</code>	Bypass Content-Security-Policy
<code>--silent</code>	Do not output any messages
<code>--auth-password TEXT</code>	Password for HTTP Basic authentication
<code>--auth-username TEXT</code>	Username for HTTP Basic authentication
<code>--help</code>	Show this message and exit.

SAVING A WEB PAGE TO AN HTTP ARCHIVE

An HTTP Archive file captures the full details of a series of HTTP requests and responses as JSON.

The `shot-scraper har` command can save a `*.har.zip` file that contains both that JSON data and the content of any assets that were loaded by the page.

```
shot-scraper har https://datasette.io/
```

This will save to `datasette-io.har`. You can use `-o` to specify a filename:

```
shot-scraper har https://datasette.io/tutorials/learn-sql \
-o learn-sql.har
```

A `.har` file is JSON. You can view it using the [Google HAR Analyzer](#) tool.

HTTP Archives can also be created as `.har.zip` files. These have a slightly different format: the `har.har` JSON does not include the full content of the responses, which is instead stored as separate files inside the `.zip`.

To create one of these, either add the `--zip` flag:

```
shot-scraper har https://datasette.io/ --zip
```

Or specify a filename that ends in `.har.zip`:

```
shot-scraper har https://datasette.io/ -o datasette-io.har.zip
```

You can view the contents of a HAR zip file using `unzip -l`:

```
unzip -l datasette-io.har.zip
```

```
Archive:  datasette-io.har.zip
 Length   Date       Time    Name
-----
 39067   02-13-2025  10:33   41824dbd0c51f584faf0e2c4e88de01b8a5dcdcd.html
  4052   02-13-2025  10:33   34972651f161f0396c697c65ef9aaeb2c9ac50c4.css
  2501   02-13-2025  10:33   9f612e71165058f0046d8bf8fec12af7eb15f39d.css
 10916   02-13-2025  10:33   2737174596eafba6e249022203c324605f023cdd.svg
  5557   02-13-2025  10:33   427504aa6ef5a8786f90fb2de636133b3fc6d1fe.js
  1393   02-13-2025  10:33   25c68a82b654c9d844c604565dab4785161ef697.js
  1170   02-13-2025  10:33   31c073551ef5c84324073edfc7b118f81ce9a7d2.svg
  1158   02-13-2025  10:33   1e0c64af7e6a4712f5e7d1917d9555bbc3d01f7a.svg
  1161   02-13-2025  10:33   ec8282b36a166d63fae4c04166bb81f945660435.svg
  3373   02-13-2025  10:33   5f85a11ef89c0e3f237c8e926c1cb66727182102.svg
```

(continues on next page)

(continued from previous page)

1134	02-13-2025	10:33	3b9d8109b919dfe9393dab2376fe03267dadcf1.svg
31670	02-13-2025	10:33	469f0d28af6c026dcae8c81731e2b0484aeac92c.jpeg
1157	02-13-2025	10:33	b7786336bfce38a9677d26dc9ef468bb1ed45e19.svg
50494	02-13-2025	10:33	har.har
-----			-----
154803			14 files

You can record multiple pages to a single HTTP Archive using the *shot-scraper multi -har* option.

9.1 Extracting resources from HAR files

Use the `--extract` or `-x` option to automatically extract all resources from the HAR file into a directory:

```
shot-scraper har https://datasette.io/ --extract
```

This will create both `datasette-io.har` and a `datasette-io/` directory containing all resources with meaningful filenames derived from their URLs.

The extracted files use extensions based on their content-type. For example, a request to `/api/data` that returns JSON will be saved with a `.json` extension.

You can combine this with `--zip`:

```
shot-scraper har https://datasette.io/ --extract --zip
```

This creates `datasette-io.har.zip` and extracts resources to the `datasette-io/` directory.

9.2 shot-scraper har --help

Full `--help` for this command:

```
Usage: shot-scraper har [OPTIONS] URL
```

Record a HAR file **for** the specified page

Usage:

```
shot-scraper har https://datasette.io/
```

This defaults to saving to `datasette-io.har` - use `-o` to specify a different filename:

```
shot-scraper har https://datasette.io/ -o trace.har
```

Use `--zip` to save **as** a `.har.zip` file instead, **or** specify a filename ending **in** `.har.zip`

Use `--extract` / `-x` to also extract **all** resources **from the** HAR into a directory. With `-x`, you can specify a base path **and** the `.har` extension will be added automatically:

(continues on next page)

(continued from previous page)

```
shot-scraper har https://datasette.io/ -x -o /tmp/datasette
```

This creates /tmp/datasette.har and extracts resources to /tmp/datasette/

Options:

-z, --zip	Save as a .har.zip file
-x, --extract	Extract resources from the HAR file into a directory
-a, --auth FILENAME	Path to JSON authentication context file
-o, --output FILE	HAR filename
--wait INTEGER	Wait this many milliseconds before taking the screenshot
--wait-for TEXT	Wait until this JS expression returns true
-j, --javascript TEXT	Execute this JavaScript on the page
--js-file TEXT	Read JavaScript to execute from this file, use - for stdin or gh:username/script to load from github.com/username/shot-scraper-scripts/script.js
--timeout INTEGER	Wait this many milliseconds before failing
--log-console	Write console.log() to stderr
--fail	Fail with an error code if a page returns an HTTP error
--skip	Skip pages that return HTTP errors
--bypass-csp	Bypass Content-Security-Policy
--auth-password TEXT	Password for HTTP Basic authentication
--auth-username TEXT	Username for HTTP Basic authentication
--help	Show this message and exit.

DUMPING OUT AN ACCESSIBILITY TREE

The `shot-scraper accessibility` command dumps out the Chromium accessibility tree for the provided URL, as JSON:

```
shot-scraper accessibility https://datasette.io/
```

Use `-o filename.json` to write the output to a file instead of displaying it.

Add `--javascript SCRIPT` to execute custom JavaScript before taking the snapshot.

10.1 `shot-scraper accessibility --help`

Full `--help` for this command:

```
Usage: shot-scraper accessibility [OPTIONS] URL
```

Dump the Chromium accessibility tree **for** the specifed page

Usage:

```
shot-scraper accessibility https://datasette.io/
```

Options:

<code>-a, --auth FILENAME</code>	Path to JSON authentication context file
<code>-o, --output FILENAME</code>	
<code>-j, --javascript TEXT</code>	Execute this JS prior to taking the snapshot
<code>--js-file TEXT</code>	Read JavaScript to execute from this file, use - for stdin or <code>gh:username/script</code> to load from github.com/username/shot-scraper-scripts/script.js
<code>--timeout INTEGER</code>	Wait this many milliseconds before failing
<code>--log-console</code>	Write <code>console.log()</code> to stderr
<code>--fail</code>	Fail with an error code if a page returns an HTTP error
<code>--skip</code>	Skip pages that return HTTP errors
<code>--bypass-csp</code>	Bypass Content-Security-Policy
<code>--auth-password TEXT</code>	Password for HTTP Basic authentication
<code>--auth-username TEXT</code>	Username for HTTP Basic authentication
<code>--help</code>	Show this message and exit.

USING SHOT-SCRAPER WITH GITHUB ACTIONS

shot-scraper was designed with GitHub Actions for screenshot automation in mind.

11.1 shot-scraper-template

The [shot-scraper-template](#) template repository can be used to quickly create your own GitHub repository with GitHub Actions configured to take screenshots of a page and write it back to the repository. Read [Instantly create a GitHub repository to take screenshots of a web page](#) for details.

11.2 Building a workflow from scratch

This Actions workflow can be used to install shot-scraper and its dependencies, take screenshots defined in the `shots.yml` file in that repository and then write the resulting screenshots back to the same repository:

```
name: Take screenshots

on:
  push:
  workflow_dispatch:

permissions:
  contents: write

jobs:
  shot-scraper:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python 3.10
        uses: actions/setup-python@v3
        with:
          python-version: "3.10"
      - uses: actions/cache@v3
        name: Configure pip caching
        with:
          path: ~/.cache/pip
          key: ${{ runner.os }}-pip
      - name: Cache Playwright browsers
```

(continues on next page)

(continued from previous page)

```

uses: actions/cache@v3
with:
  path: ~/.cache/ms-playwright/
  key: ${{ runner.os }}-playwright
- name: Install dependencies
  run: |
    pip install shot-scraper
    shot-scraper install
- name: Take shots
  run: |
    shot-scraper multi shots.yml
- name: Commit and push
  run: |-
    git config user.name "Automated"
    git config user.email "actions@users.noreply.github.com"
    git add -A
    timestamp=$(date -u)
    git commit -m "${timestamp}" || exit 0
    git pull --rebase
    git push

```

The actions/cache@v3 steps set up [caching](#), so your workflow will only download and install the necessary software the very first time it runs.

11.3 Optimizing PNGs using Oxipng

You can losslessly compress the PNGs generated using shot-scraper by running them through [Oxipng](#). Add the following steps to the beginning of your workflow to install Oxing:

```

- name: Cache Oxipng
  uses: actions/cache@v3
  with:
    path: ~/.cargo/
    key: ${{ runner.os }}-cargo
- name: Install Oxipng if it is not installed
  run: |
    which oxipng || cargo install oxipng

```

Then after running shot-scraper add this step to compress the images:

```

- name: Optimize PNGs
  run: |-
    oxipng -o 4 -i 0 --strip safe *.png

```

[simonw/datasette-screenshots](#) is an example of a repository that uses this pattern.

See [Optimizing PNGs in GitHub Actions using Oxipng](#) for more on how this works.

CONTRIBUTING

The GitHub repository for this project is [simonw/shot-scraper](https://github.com/simonw/shot-scraper).

To contribute to this tool, first checkout the code. You can use `uv` to run the tests like this:

```
cd shot-scraper
uv run pytest
```

You'll need to install the Playwright browsers too:

```
uv run shot-scraper install
```

Some of the tests exercise the CLI utility directly. Run those like so:

```
uv run tests/run_examples.sh
```

12.1 Documentation

Documentation for this project uses `MyST` - it is written in Markdown and rendered using Sphinx.

To build the documentation locally, run the following:

```
cd docs
uv run --with-requirements requirements.txt make livehtml
```

This will start a live preview server, using `sphinx-autobuild`.

The CLI `--help` examples in the documentation are managed using `Cog`. Update those files like this:

```
uv run cog -r docs/*.md
```

12.2 Publishing the release notes

After pushing a release, I use the following to create a screenshot of the release notes to use in social media posts:

```
shot-scraper https://github.com/simonw/shot-scraper/releases/tag/0.15 \
  --selector '.Box-body' --width 700 \
  --retina
```

[Example post](#).

SHOT-SCRAPER

A CLI utility for taking screenshots of websites, recording video demos and scraping sites using JavaScript
For background on this project see [shot-scraper: automated screenshots for documentation, built on Playwright](#).

13.1 Documentation

- [Full documentation for shot-scraper](#)
- [Tutorial: Automating screenshots for the Datasette documentation using shot-scraper](#)
- [Release notes](#)

13.2 Get started with GitHub Actions

To get started without installing any software, use the [shot-scraper-template](#) template to create your own GitHub repository which takes screenshots of a page using `shot-scraper`. See [Instantly create a GitHub repository to take screenshots of a web page](#) for details.

13.3 Quick installation

You can install the `shot-scraper` CLI tool using `pip`:

```
pip install shot-scraper
# Now install the browser it needs:
shot-scraper install
```

13.4 Taking your first screenshot

You can take a screenshot of a web page like this:

```
shot-scraper https://datasette.io/
```

This will create a screenshot in a file called `datasette-io.png`.

Many more options are available, see [Taking a screenshot](#) for details.

13.5 Examples

- The [shot-scraper-demo](#) repository uses this tool to capture recently spotted owls in El Granada, CA according to [this page](#), and to generate an annotated screenshot illustrating a Datasette feature as described in [my blog](#).
- The [Datasette Documentation](#) uses screenshots taken by `shot-scraper` running in the [simonw/datasette-screenshots](#) GitHub repository, described in detail in [Automating screenshots for the Datasette documentation using shot-scraper](#).
- Ben Welsh built [@newshomepages](#), a Twitter bot that uses `shot-scraper` and GitHub Actions to take screenshots of news website homepages and publish them to Twitter. The code for that lives in [palewire/news-homepages](#).
- [scrape-hacker-news-by-domain](#) uses `shot-scraper javascript` to scrape a web page. See [Scraping web pages from the command-line with shot-scraper](#) for details of how this works.
- Reuters uses `shot-scraper` to generate regularly updating data dashboards [for email newsletters](#).