shot-scraper documentation

Release 1.2

Simon Willison

Apr 27, 2023

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 7
- 2.8 Retina images 7
- 2.9 Transparent background 8
- 2.10 Interacting with the page 8
- 2.11 Logging all requests 8
- 2.12 Taking screenshots of local HTML files 9
- 2.13 Tips for executing JavaScript 9
- 2.14 Viewing console.log() output 10
- 2.15 `shot-scraper shot --help` 10

3 Websites that need authentication **13**
- 3.1 `shot-scraper auth --help` 13

4 Taking multiple screenshots **15**
- 4.1 `shot-scraper multi --help` 17

5 Scraping pages using JavaScript **19**
- 5.1 Running more than one statement 20
- 5.2 Using async/await 20
- 5.3 Running JavaScript from a file 20
- 5.4 Using this for automated tests 21
- 5.5 Example: Extracting page content with Readability.js 21
- 5.6 shot-scraper javascript –help 22

6 Saving a web page to PDF **23**
- 6.1 `shot-scraper pdf --help` 23

7 Dumping the HTML of a page **25**
- 7.1 Retrieving the HTML for a specific element 25
- 7.2 `shot-scraper html --help` 25

8 Dumping out an accessibility tree 27
8.1 shot-scraper accessibility --help 27

9 Using shot-scraper with GitHub Actions 29
9.1 shot-scraper-template 29
9.2 Building a workflow from scratch 29
9.3 Optimizing PNGs using Oxipng 30

10 Contributing 31
10.1 Documentation 31
10.2 Tweeting the release notes 32

11 shot-scraper 33
11.1 Documentation 33
11.2 Get started with GitHub Actions 33
11.3 Quick installation 33
11.4 Taking your first screenshot 34
11.5 Examples 34

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

CHAPTER ONE

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
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

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

(continues on next page)

(continued from previous page)

```
                                  Which browser to install
  --help                          Show this message and exit.
```

CHAPTER

TWO

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")
).classList.add("js-selector-a1f5ba0fc4a4317e58a3bd11a0f16b96");
```

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")'
```

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';"
```

2.7 Using JPEGs instead of PNGs

Screenshots default to PNG. You can save as a JPEG by specifying a `-o` filename that ends with `.jpg`.

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 Retina images

The `--retina` option sets 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.

2.9 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. Does not work with JPG images or when `quality` is set.

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

2.10 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'
```

2.11 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": 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`.

2.12 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.

2.13 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
  // ...
  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.

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

2.15 `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:

      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:
```

(continues on next page)

(continued from previous page)

```
      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:
  -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 and shot - defaults
                                  to the full height of the page
  -o, --output FILE
  -s, --selector TEXT             Take shot of first element matching this CSS
                                  selector
  --selector-all TEXT             Take shot of all elements matching this CSS
                                  selector
  --js-selector TEXT              Take shot of first element matching this JS
                                  (el) expression
  --js-selector-all TEXT          Take shot of all elements matching this JS
                                  (el) expression
  -p, --padding INTEGER           When using selectors, add this much padding in
                                  pixels
  -j, --javascript TEXT           Execute this JS prior to taking the shot
  --retina                        Use device scale factor of 2
  --omit-background               Omit the default browser background from the
                                  shot, making it possible take advantage of
                                  transparence. Does not work with JPEGs or when
                                  using --quality.
  --quality INTEGER               Save as JPEG with this quality, e.g. 80
  --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
  --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
  --silent                        Do not output any messages
  --help                          Show this message and exit.
```

CHAPTER

THREE

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

CHAPTER

FOUR

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

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

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:

```yaml
- 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:

```yaml
- 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:

```yaml
- 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:

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

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

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

4.1 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/

  https://shot-scraper.datasette.io/en/stable/multi.html

Options:
  -a, --auth FILENAME             Path to JSON authentication context file
  --retina                        Use device scale factor of 2
  --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
  --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
  --help                          Show this message and exit.
```

CHAPTER

FIVE

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
```

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"
}
```

5.1 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);
}"
```

5.2 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 Skypack 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.skypack.dev/@mozilla/readability');
  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
));"
```

5.3 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
```

5.4 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';
      }"
```

5.5 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 Skypack 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.skypack.dev/@mozilla/readability');
  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",
    "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.

5.6 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 FILENAME            Read input JavaScript from this file
  -a, --auth FILENAME             Path to JSON authentication context file
  -o, --output FILENAME           Save output JSON to this file
  -r, --raw                       Output JSON strings as raw text
  -b, --browser [chromium|firefox|webkit|chrome|chrome-beta]
                                  Which browser to use
  --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
  --help                          Show this message and exit.
```

CHAPTER SIX

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
```

6.1 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:
  -a, --auth FILENAME             Path to JSON authentication context file
  -o, --output FILE
  -j, --javascript TEXT           Execute this JS prior to creating the PDF
  --wait INTEGER                  Wait this many milliseconds before taking the
```

(continues on next page)

(continued from previous page)

```
                                  screenshot
  --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
  --silent                        Do not output any messages
  --help                          Show this message and exit.
```

CHAPTER

SEVEN

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!'"
```

7.1 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>
  <img class="datasette-logo" src="/static/datasette-logo.svg" alt="Datasette">
</h1>
```

7.2 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:
  -a, --auth FILENAME             Path to JSON authentication context file
  -o, --output FILE
  -j, --javascript TEXT           Execute this JS prior to saving the HTML
  -s, --selector TEXT             Return outerHTML of first element matching
                                  this CSS selector
  --wait INTEGER                  Wait this many milliseconds before taking the
                                  snapshot
  --log-console                   Write console.log() to stderr
  -b, --browser [chromium|firefox|webkit|chrome|chrome-beta]
                                  Which browser to use
  --user-agent TEXT               User-Agent header to use
  --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
  --help                          Show this message and exit.
```

CHAPTER

EIGHT

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.

8.1 `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:
  -a, --auth FILENAME    Path to JSON authentication context file
  -o, --output FILENAME
  -j, --javascript TEXT  Execute this JS prior to taking the snapshot
  --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
  --help                 Show this message and exit.
```

CHAPTER

NINE

USING SHOT-SCRAPER WITH GITHUB ACTIONS

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

9.1 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.

9.2 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:

```yaml
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.

9.3 Optimizing PNGs using Oxipng

You can losslessy 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.

CHAPTER

TEN

CONTRIBUTING

To contribute to this tool, first checkout the code. Then create a new virtual environment:

```
cd shot-scraper
python -m venv venv
source venv/bin/activate
```

Or if you are using `pipenv`:

```
pipenv shell
```

Now install the dependencies and test dependencies:

```
pip install -e '.[test]'
```

Then you'll need to install the Playwright browsers too:

```
shot-scraper install
```

To run the tests:

```
pytest
```

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

```
tests/run_examples.sh
```

10.1 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
pip install -r 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:

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

10.2 Tweeting the release notes

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

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

Example tweet.

CHAPTER

ELEVEN

SHOT-SCRAPER

A command-line utility for taking automated screenshots of websites

For background on this project see shot-scraper: automated screenshots for documentation, built on Playwright.

11.1 Documentation

- Full documentation for shot-scraper
- Tutorial: Automating screenshots for the Datasette documentation using shot-scraper
- Release notes

11.2 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.

11.3 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
```

11.4 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.

11.5 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.