

# Getting Started with Headless Chrome



By [Eric Bidelman](#)

(<https://developers.google.com/web/resources/contributors#ericbidelman>)

Engineer @ Google working on Lighthouse, Web Components, Chrome, and the web

## TL;DR

[Headless Chrome](https://chromium.googlesource.com/chromium/src/+lkgr/headless/README.md) (<https://chromium.googlesource.com/chromium/src/+lkgr/headless/README.md>) is shipping in Chrome 59. It's a way to run the Chrome browser in a headless environment. Essentially, running Chrome without chrome! It brings **all modern web platform features** provided by Chromium and the Blink rendering engine to the command line.

Why is that useful?

A headless browser is a great tool for automated testing and server environments where you don't need a visible UI shell. For example, you may want to run some tests against a real web page, create a PDF of it, or just inspect how the browser renders an URL.

**Caution:** Headless mode is available on Mac and Linux in **Chrome 59**. [Windows support](#) (<https://bugs.chromium.org/p/chromium/issues/detail?id=686608>) is coming in Chrome 60. To check what version of Chrome you have, open **chrome://version**.

## Starting Headless (CLI)

The easiest way to get started with headless mode is to open the Chrome binary from the command line. If you've got Chrome 59+ installed, start Chrome with the `--headless` flag:

```
chrome \
  --headless \                # Runs Chrome in headless mode.
  --disable-gpu \             # Temporarily needed for now.
  --remote-debugging-port=9222 \
  https://www.chromestatus.com # URL to open. Defaults to about:blank.
```

**Note:** Right now, you'll also want to include the `--disable-gpu` flag. That will eventually go away.

`chrome` should point to your installation of Chrome. The exact location will vary from platform to platform. Since I'm on Mac, I created convenient aliases for each version of Chrome that I have installed.

If you're on the stable channel of Chrome and cannot get the Beta, I recommend using `chrome-canary`:

```
alias chrome="/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome"
alias chrome-canary="/Applications/Google\ Chrome\ Canary.app/Contents/MacOS/Google\ Chrome Canary"
alias chromium="/Applications/Chromium.app/Contents/MacOS/Chromium"
```

Download Chrome Canary [here](https://www.google.com/chrome/browser/canary.html) (https://www.google.com/chrome/browser/canary.html).

## Command line features

In some cases, you may not need to programmatically script (#node) Headless Chrome. There are some useful command line flags

(https://cs.chromium.org/chromium/src/headless/app/headless\_shell\_switches.cc) to perform common tasks.

### Printing the DOM

The `--dump-dom` flag prints `document.body.innerHTML` to stdout:

```
chrome --headless --disable-gpu --dump-dom https://www.chromestatus.com/
```

### Create a PDF

The `--print-to-pdf` flag creates a PDF of the page:

```
chrome --headless --disable-gpu --print-to-pdf https://www.chromestatus.com/
```

### Taking screenshots

To capture a screenshot of a page, use the `--screenshot` flag:

```
chrome --headless --disable-gpu --screenshot https://www.chromestatus.com/

# Size of a standard letterhead.
chrome --headless --disable-gpu --screenshot --window-size=1280,1696 https://www

# Nexus 5x
chrome --headless --disable-gpu --screenshot --window-size=412,732 https://www.c
```

Running with `--screenshot` will produce a file named `screenshot.png` in the current working directory. If you're looking for full page screenshots, things are a tad more involved. There's a great blog post from David Schnurr that has you covered. Check out [Using headless Chrome as an automated screenshot tool](https://medium.com/@dschnr/using-headless-chrome-as-an-automated-screenshot-tool-4b07dffba79a)

(<https://medium.com/@dschnr/using-headless-chrome-as-an-automated-screenshot-tool-4b07dffba79a>)

## REPL mode (read-eval-print loop)

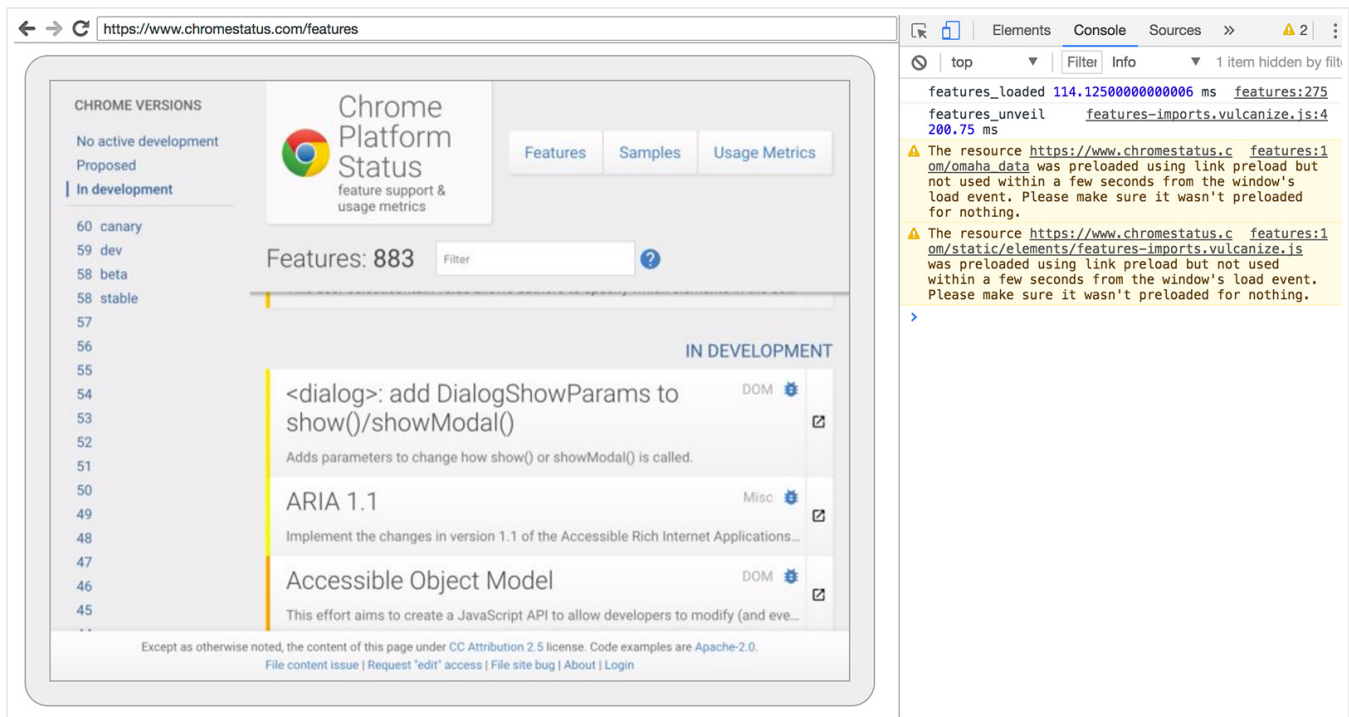
The `--repl` flag runs Headless in a mode where you can evaluate JS expressions in the browser, right from the command line:

```
$ chrome --headless --disable-gpu --repl https://www.chromestatus.com/
[0608/112805.245285:INFO:headless_shell.cc(278)] Type a Javascript expression to
>>> location.href
{"result":{"type":"string","value":"https://www.chromestatus.com/features"}}
>>> quit
$
```

## Debugging Chrome without a browser UI?

When you run Chrome with `--remote-debugging-port=9222`, it starts an instance with the [DevTools protocol](https://chromedevtools.github.io/devtools-protocol/) (<https://chromedevtools.github.io/devtools-protocol/>) enabled. The protocol is used to communicate with Chrome and drive the headless browser instance. It's also what tools like Sublime, VS Code, and Node use for remote debugging an application. #synergy

Since you don't have browser UI to see the page, navigate to `http://localhost:9222` in another browser to check that everything is working. You'll see a list of inspectable pages where you can click through and see what Headless is rendering:



DevTools remote debugging UI

From here, you can use the familiar DevTools features to inspect, debug, and tweak the page as you normally would. If you're using Headless programmatically, this page is also a powerful debugging tool for seeing all the raw DevTools protocol commands going across the wire, communicating with the browser.

## Using programmatically (Node)

### The Puppeteer API

Puppeteer (<https://github.com/GoogleChrome/puppeteer>) is a Node library developed by the Chrome team. It provides a high-level API to control headless (or full) Chrome. It's similar to other automated testing libraries like Phantom and NightmareJS, but it only works with the latest versions of Chrome.

Among other things, Puppeteer can be used to easily take screenshots, create PDFs, navigate pages, and fetch information about those pages. I recommend the library if you want to

quickly automate browser testing. It hides away the complexities of the DevTools protocol and takes care of redundant tasks like launching a debug instance of Chrome.

Install it:

```
yarn add puppeteer
```

### Example - print the user agent

```
const puppeteer = require('puppeteer');

(async() => {
  const browser = await puppeteer.launch();
  console.log(await browser.version());
  browser.close();
})();
```

### Example - taking a screenshot of the page

```
const puppeteer = require('puppeteer');

(async() => {

  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://www.chromestatus.com', {waitUntil: 'networkidle'});
  await page.pdf({path: 'page.pdf', format: 'A4'});

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

Check out [Puppeteer's documentation](https://github.com/GoogleChrome/puppeteer/blob/master/docs/api.md)

(<https://github.com/GoogleChrome/puppeteer/blob/master/docs/api.md>) to learn more about the full API.

## The CRI library

[chrome-remote-interface](https://www.npmjs.com/package/chrome-remote-interface) (<https://www.npmjs.com/package/chrome-remote-interface>) is a lower-level library than Puppeteer's API. I recommend it if you want to be close to the metal and use the [DevTools protocol](https://chromedevtools.github.io/devtools-protocol/) (<https://chromedevtools.github.io/devtools-protocol/>) directly.

## Launching Chrome

chrome-remote-interface doesn't launch Chrome for you, so you'll have to take care of that yourself.

In the CLI section, we started Chrome manually (#cli) using `--headless --remote-debugging-port=9222`. However, to fully automate tests, you'll probably want to spawn Chrome *from* your application.

One way is to use `child_process`:

```
const execFile = require('child_process').execFile;

function launchHeadlessChrome(url, callback) {
  // Assuming MacOSx.
  const CHROME = '/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome
  execFile(CHROME, ['--headless', '--disable-gpu', '--remote-debugging-port=9222
}

launchHeadlessChrome('https://www.chromestatus.com', (err, stdout, stderr) => {
  ...
});
```

But things get tricky if you want a portable solution that works across multiple platforms. Just look at that hard-coded path to Chrome :(

## Using ChromeLauncher

Lighthouse (<https://developers.google.com/web/tools/lighthouse/>) is a marvelous tool for testing the quality of your web apps. A robust module for launching Chrome was developed within Lighthouse and is now extracted for standalone use. The chrome-launcher NPM module (<https://www.npmjs.com/package/chrome-launcher>) will find where Chrome is installed, set up a debug instance, launch the browser, and kill it when your program is done. Best part is that it works cross-platform thanks to Node!

By default, **chrome-launcher will try to launch Chrome Canary** (if it's installed), but you can change that to manually select which Chrome to use. To use it, first install from npm:

```
yarn add chrome-launcher
```

## Example - using chrome-launcher to launch Headless

```
const chromeLauncher = require('chrome-launcher');

// Optional: set logging level of launcher to see its output.
// Install it using: yarn add lighthouse-logger
// const log = require('lighthouse-logger');
// log.setLevel('info');

/**
 * Launches a debugging instance of Chrome.
 * @param {boolean=} headless True (default) launches Chrome in headless mode.
 *     False launches a full version of Chrome.
 * @return {Promise<ChromeLauncher>}
 */
function launchChrome(headless=true) {
  return chromeLauncher.launch({
    // port: 9222, // Uncomment to force a specific port of your choice.
    chromeFlags: [
      '--window-size=412,732',
      '--disable-gpu',
      headless ? '--headless' : ''
    ]
  });
}

launchChrome().then(chrome => {
  console.log(`Chrome debuggable on port: ${chrome.port}`);
  ...
  // chrome.kill();
});
```

Running this script doesn't do much, but you should see an instance of Chrome fire up in the task manager that loaded `about:blank`. Remember, there won't be any browser UI. We're headless.

To control the browser, we need the DevTools protocol!

## Retrieving information about the page

**Warning:** The DevTools protocol can do a ton of interesting stuff, but it can be a bit daunting at first. I recommend spending a bit of time browsing the [DevTools Protocol Viewer](#)

(<https://chromedevtools.github.io/devtools-protocol/>), first. Then, move on to the **chrome-remote-interface** API docs to see how it wraps the raw protocol.

Let's install the library:

```
yarn add chrome-remote-interface
```

## Examples

### Example - print the user agent

```
const CDP = require('chrome-remote-interface');  
  
...  
  
launchChrome().then(async chrome => {  
  const version = await CDP.Version({port: chrome.port});  
  console.log(version['User-Agent']);  
});
```

Results in something like: **HeadlessChrome/60.0.3082.0**

### Example - check if the site has a web app manifest

(<https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/>)

```
const CDP = require('chrome-remote-interface');  
  
...  
  
(async function() {  
  
  const chrome = await launchChrome();  
  const protocol = await CDP({port: chrome.port});  
  
  // Extract the DevTools protocol domains we need and enable them.  
  // See API docs: https://chromedevtools.github.io/devtools-protocol/  
  const {Page} = protocol;  
  await Page.enable();  
  
  Page.navigate({url: 'https://www.chromestatus.com/'});  
  
  // Wait for window.onload before doing stuff.
```



```

Page.loadEventFired(async () => {
  const manifest = await Page.getAppManifest();

  if (manifest.url) {
    console.log('Manifest: ' + manifest.url);
    console.log(manifest.data);
  } else {
    console.log('Site has no app manifest');
  }

  protocol.close();
  chrome.kill(); // Kill Chrome.
});

})();

```

**Example** - extract the `<title>` of the page using DOM APIs.

```

const CDP = require('chrome-remote-interface');

...

(async function() {

  const chrome = await launchChrome();
  const protocol = await CDP({port: chrome.port});

  // Extract the DevTools protocol domains we need and enable them.
  // See API docs: https://chromedevtools.github.io/devtools-protocol/
  const {Page, Runtime} = protocol;
  await Promise.all([Page.enable(), Runtime.enable()]);

  Page.navigate({url: 'https://www.chromestatus.com/'});

  // Wait for window.onload before doing stuff.
  Page.loadEventFired(async () => {
    const js = "document.querySelector('title').textContent";
    // Evaluate the JS expression in the page.
    const result = await Runtime.evaluate({expression: js});

    console.log('Title of page: ' + result.result.value);

    protocol.close();
    chrome.kill(); // Kill Chrome.
  });
})();

```

```
});  
  
})();
```

## Using Selenium, WebDriver, and ChromeDriver

Right now, Selenium opens a full instance of Chrome. In other words, it's an automated solution but not completely headless. However, Selenium can be configured to run headless Chrome with a little work. I recommend [Running Selenium with Headless Chrome](https://intoli.com/blog/running-selenium-with-headless-chrome/) (<https://intoli.com/blog/running-selenium-with-headless-chrome/>) if you want the full instructions on how to set things up yourself, but I've dropped in some examples below to get you started.

### Using ChromeDriver

[ChromeDriver](https://sites.google.com/a/chromium.org/chromedriver/) (<https://sites.google.com/a/chromium.org/chromedriver/>) 2.3.0 supports Chrome 59 and later and works with headless Chrome. In some cases, you may need Chrome 60 to work around bugs. For example, there are known issues with taking screenshots in Chrome 59.

Install:

```
yarn add selenium-webdriver chromedriver
```

Example:

```
const fs = require('fs');  
const webdriver = require('selenium-webdriver');  
const chromedriver = require('chromedriver');  
  
// This should be the path to your Canary installation.  
// I'm assuming Mac for the example.  
const PATH_TO_CANARY = '/Applications/Google Chrome Canary.app/Contents/MacOS/Go  
  
const chromeCapabilities = webdriver.Capabilities.chrome();  
chromeCapabilities.set('chromeOptions', {  
  binary: PATH_TO_CANARY // Screenshots require Chrome 60. Force Canary.  
  'args': [  
    '--headless',  
  ]  
});
```

```
const driver = new webdriver.Builder()
  .forBrowser('chrome')
  .withCapabilities(chromeCapabilities)
  .build();

// Navigate to google.com, enter a search.
driver.get('https://www.google.com/');
driver.findElement({name: 'q'}).sendKeys('webdriver');
driver.findElement({name: 'btnG'}).click();
driver.wait(webdriver.until.titleIs('webdriver - Google Search'), 1000);

// Take screenshot of results page. Save to disk.
driver.takeScreenshot().then(base64png => {
  fs.writeFileSync('screenshot.png', new Buffer(base64png, 'base64'));
});

driver.quit();
```

## Using WebDriverIO

WebDriverIO (<http://webdriver.io/>) is a higher level API on top of Selenium WebDriver.

Install:

```
yarn add webdriverio chromedriver
```

Example: filter CSS features on chromestatus.com

```
const webdriverio = require('webdriverio');
const chromedriver = require('chromedriver');

// This should be the path to your Canary installation.
// I'm assuming Mac for the example.
const PATH_TO_CANARY = '/Applications/Google Chrome Canary.app/Contents/MacOS/Go
const PORT = 9515;

chromedriver.start([
  '--url-base=wd/hub',
  `--port=${PORT}`,
  '--verbose'
]);
```

```
(async () => {

const opts = {
  port: PORT,
  desiredCapabilities: {
    browserName: 'chrome',
    chromeOptions: {
      binary: PATH_TO_CANARY // Screenshots require Chrome 60. Force Canary.
      args: ['--headless']
    }
  }
};

const browser = webdriverio.remote(opts).init();

await browser.url('https://www.chromestatus.com/features');

const title = await browser.getTitle();
console.log(`Title: ${title}`);

await browser.waitForText('.num-features', 3000);
let numFeatures = await browser.getText('.num-features');
console.log(`Chrome has ${numFeatures} total features`);

await browser.setValue('input[type="search"]', 'CSS');
console.log('Filtering features...');
await browser.pause(1000);

numFeatures = await browser.getText('.num-features');
console.log(`Chrome has ${numFeatures} CSS features`);

const buffer = await browser.saveScreenshot('screenshot.png');
console.log('Saved screenshot...');

chromedriver.stop();
browser.end();

})();
```

## Further resources

---

Here are some useful resources to get you started:

## Docs

- [DevTools Protocol Viewer](https://chromedevtools.github.io/devtools-protocol/) (https://chromedevtools.github.io/devtools-protocol/) - API reference docs

## Tools

- [chrome-remote-interface](https://www.npmjs.com/package/chrome-remote-interface) (https://www.npmjs.com/package/chrome-remote-interface) - node module that wraps the DevTools protocol
- [Lighthouse](https://github.com/GoogleChrome/lighthouse) (https://github.com/GoogleChrome/lighthouse) - automated tool for testing web app quality; makes heavy use of the protocol
- [chrome-launcher](https://github.com/GoogleChrome/lighthouse/tree/master/chrome-launcher) (https://github.com/GoogleChrome/lighthouse/tree/master/chrome-launcher) - node module for launching Chrome, ready for automation

## Demos

- ["The Headless Web"](https://paul.kinlan.me/the-headless-web/) (https://paul.kinlan.me/the-headless-web/) - Paul Kinlan's great blog post on using Headless with api.ai.

## FAQ

---

### Do I need the `--disable-gpu` flag?

Yes, for now. The `--disable-gpu` flag is a temporary requirement to work around a few bugs. You won't need this flag in future versions of Chrome. See <https://crbug.com/546953#c152> (https://bugs.chromium.org/p/chromium/issues/detail?id=546953#c152) and <https://crbug.com/695212> (https://bugs.chromium.org/p/chromium/issues/detail?id=695212) for more information.

### So I still need Xvfb?

No. Headless Chrome doesn't use a window so a display server like Xvfb is no longer needed. You can happily run your automated tests without it.

What is Xvfb? Xvfb is an in-memory display server for Unix-like systems that enables you to run graphical applications (like Chrome) without an attached physical display. Many people use Xvfb to run earlier versions of Chrome to do "headless" testing.

## How do I create a Docker container that runs Headless Chrome?

Check out [lighthouse-ci](https://github.com/ebidel/lighthouse-ci) (<https://github.com/ebidel/lighthouse-ci>). It has an [example Dockerfile](https://github.com/ebidel/lighthouse-ci/blob/master/builder/Dockerfile.headless) (<https://github.com/ebidel/lighthouse-ci/blob/master/builder/Dockerfile.headless>) that uses Ubuntu as a base image, and installs + runs Lighthouse in an App Engine Flexible container.

## Can I use this with Selenium / WebDriver / ChromeDriver?

Yes. See [Using Selenium, WebDriver, or ChromeDriver](#) (#drivers).

## How is this related to PhantomJS?

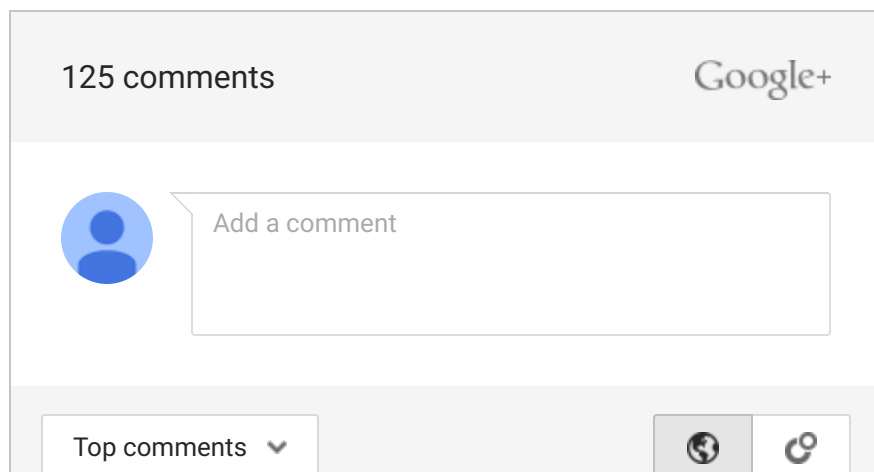
Headless Chrome is similar to tools like [PhantomJS](http://phantomjs.org/) (<http://phantomjs.org/>). Both can be used for automated testing in a headless environment. The main difference between the two is that Phantom uses an older version of WebKit as its rendering engine while Headless Chrome uses the latest version of Blink.

At the moment, Phantom also provides a higher level API than the [DevTools protocol](https://chromedevtools.github.io/devtools-protocol/) (<https://chromedevtools.github.io/devtools-protocol/>).

## Where do I report bugs?

For bugs against Headless Chrome, file them on [crbug.com](https://bugs.chromium.org/p/chromium/issues/entry?components=Blink&blocking=705916&cc=skyostil%40chromium.org&Proj=Headless) (<https://bugs.chromium.org/p/chromium/issues/entry?components=Blink&blocking=705916&cc=skyostil%40chromium.org&Proj=Headless>)

For bugs in the DevTools protocol, file them at [github.com/ChromeDevTools/devtools-protocol](https://github.com/ChromeDevTools/devtools-protocol) (<https://github.com/ChromeDevTools/devtools-protocol/issues/new>).





**Jacob Edward** 1 week ago - Shared publicly

Not only is your code throwing an error without "google-chrome" replacing "chrome", it's also throwing a Promise error... brilliant

+1 · Reply



**Jacob Edward** 2 weeks ago - Shared publicly

just copy and paste the Example - extract the <title> of the page using DOM APIs. and I'm getting errors with node js (delete the ... of course)

+1 · Reply



**Paul Irish** 3 months ago - Shared publicly

Btw: Windows is supported as of Chrome 60!

+14 +1 · Reply



**Juan Carlos Madrigal**

3 months ago +3

Hi Paul,  
I want to know if you could help me a little.  
I'm using Chrome Canary on Windows: Version 60.0.3112.0 (Official Build) canary (64-bit).



**Jacob Edward** 1 month ago (edited) - Shared publicly

google-chrome --headless --disable-gpu --repl  
<https://www.chromestatus.com/>

is giving me an infinite loop where {"result": {"type": "undefined"}} is echo'd indefinitely... on ubuntu

+6 +1 · Reply



**ryan edge**

1 month ago +1

google-chrome in ubuntu is a symlink to a bash script at /opt/google/chrome/google-chrome  
This bash script calls a binary at /opt/google/chrome/chrome



**Siddarth Gajjar**

1 month ago

**+ryan edge** Hello ryan,

I need help regarding chrome headless. How to run javascript using chrome --headless cli



**Jacob Edward** 1 month ago - Shared publicly

you need to change

chrome --headless --disable-gpu ...

to

**+1** · Reply



**Akristie Powell** 1 month ago - Shared publicly

Kanskje ,best å la være en stund.

**+1**



**Андрей Лавров** 3 months ago - Shared publicly

Can Google Chrome browser windows be accessed from other computer in normal (none headless) mode via DevTools protocol?

**+4** **+1** · Reply



**可可** 3 months ago - Shared publicly

awsome

**+3** **+1** · Reply



**Siddharth Kulkarni** 3 months ago - Shared publicly

Great write up. Need more content though.

**+2** **+1** · Reply

View all 6 replies ▼



**Manu J**

2 months ago

**+Daniel Teixeira** I found that too but it's not complete. For ex. print-to-pdf is missing





**Frank Taylor**  
(paceaux)

2 months ago

The API documentation is all the content you need. It's up to us to take it from there. But, if you wanted a starting point:  
<http://blog.frankmtaylor.com/2017/06/16/seeing>



**Akristie Powell** 1 month ago - Shared publicly

Jeg synes det kan være VELDIG slitsomt å skrive,svar og tilbake meldinger,føler jeg blir satt TILBAKE,noe.Liker det,når jeg skriver mere fritt.

· Translate

+1



**Akristie Powell** 1 month ago - Shared publicly

Jeg finner ikke,samme hvor mye jeg leter,hodeløs Chrome.men takker,så mye.Ene ide er vel å oversette norsk-eller hvilket som helst språk-tema.Glad....for dette tilbudet.Klart jeg er.

· Translate

+1



**Michael Bingham** 1 week ago - Shared publicly

Is it possible to include site login credentials when using print to pdf?

+1 · Reply



**helun cao** 1 week ago - Shared publicly

When I connected the browser with WebSocket, the connection was automatically disconnected after 60 seconds. I found the browser was actively disconnected. Is there any way to extend the time or close it?

+1 +1 · Reply



**ajinkya dhamnaskar** 3 weeks ago - Shared publicly

Is there any concrete way to identify, if JS code is running on usual chrome browser or headless chrome?



+1 · Reply



**Saeed Zamani** 3 weeks ago - Shared publicly  
useful af

+1 · Reply



**Hakan Bilgin** 3 months ago (edited) - Shared publicly  
Great but as many other, I have trouble upgrading Chrome since the auto-update fails on MacOS :-(  
Could someone fix that problem first?

+1 · Reply



**Dragan Filipović**  
2 months ago  
59 has not landed yet on Mac, try Canary.



**Dan Phiffer** 3 months ago - Shared publicly  
Great article, thanks!

I made a shell script that takes screenshots, but for some reason if I run it from a cronjob it doesn't work. Any ideas why that might be?

+1 +1 · Reply



**David Jones**  
1 month ago  
The user cron runs as may not have access or permission to launch the chrome executable that you can on the command line.




**贾伟川** 3 months ago - Shared publicly  
Great!

+1 +1 · Reply



**aljoša gomilšek (papak)**  
3 months ago - Shared publicly  
will gpu acceleration be (eventually) enabled? as I can see "it is needed for now"

+1

**Zane Hitchcox** 3 weeks ago - Shared publicly  
Now getting (node:1001)  
UnhandledPromiseRejectionWarning: Unhandled  
promise rejection (rejection id: 1): Error: connect  
ECONNREFUSED 127.0.0.1:9222

+1

 · Reply

Show more

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](http://creativecommons.org/licenses/by/3.0/) (<http://creativecommons.org/licenses/by/3.0/>), and code samples are licensed under the [Apache 2.0 License](http://www.apache.org/licenses/LICENSE-2.0) (<http://www.apache.org/licenses/LICENSE-2.0>). For details, see our [Site Policies](https://developers.google.com/terms/site-policies) (<https://developers.google.com/terms/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated August 21, 2017.



### Chromium Blog

The latest news on the  
Chromium blog



### GitHub

Fork our API code samples  
and other open-source  
projects.



### Twitter

Connect with @ChromiumDev  
on Twitter



### Videos

Check out the Web Developer  
Relations team's videos



### Events

Attend a developer event and  
get hacking