curl Cheat Sheet: Helpful Commands and Exciting Hacks

February 4, 2023 / By Cassandra Lee

Listen to the article

With the command-line application curl (aka cURL, short for “client URL”), you can automate batch actions such as submitting thousands of Google Forms, flooding servers with requests in penetration testing, and accessing remote files hands-free using only **Unix bash scripting** without additional programming languages.

This curl cheat sheet aims to provide an overview of curl for beginners and a taste of hacking with curl for cybersecurity fans like you. Download this curl cheat sheet **here**.

Keep the Terminal program (on Unix/Linux systems, including macOS) at hand to try out the commands below, many of which yield meaningful results. When you’re ready, let’s dive in.

Refresher: What Is curl?

In computer networking, a client is a machine that asks for data or services, and a server is a machine that provides them. curl is a command-line program for clients to submit requests to servers.

curl is helpful for quickly and automatically checking responses from servers, its prime usage being **curl GET** and **curl POST** commands. As curl operates at the protocol level (HTTP/S, FTP, SCP, IMAP,

POP3, SMTP, etc.), you can tailor server requests and cyber attacks to complex vulnerabilities not covered by handy security tools such as BurpSuitePro.

The more familiar you are with curl, the more finely you can adjust curl operations. This curl cheat sheet will help you get started.

Note: The live websites in the commands below work at the time of writing, but URLs and technology may change anytime.

Web Browsing

The most straightforward use of curl is the command-line display of websites and files, which is also how most computer science students learn about curl in the first place.

curl options (aka **flags**) begin with a hyphen (-) or two (–), and they take arguments that are strings, URLs or file paths.

COMMAND	DESCRIPTION
`curl http://example.com`	Return the source file of a URL **http://example.com/**
`curl --list-only "http://socialdance.stanford.edu/music/"`	List contents of the directory **http://socialdance.stanford.edu/music/**
`curl -l`	Abbreviation of `curl --list-only`
`curl --location "https://aveclagare.org/mp3"`	Redirect query as specified by HTTP status response code 3xx. This URL directory, **https://aveclagare.org/mp3**, does not return the list of MP3 files using the `curl --list-only` command, but it does with `curl --location`.
`curl -L`	Abbreviation of `curl --location`
`curl --fail-early "ftp://ftp.corel.com"`	Fail quickly in resolving **ftp://ftp.corel.com**
`curl --head "https://stationx.net"`	Fetch HTTP headers of the URL **https://stationx.net**

COMMAND	DESCRIPTION
`curl -I`	Abbreviation of `curl --head`
`curl --head --show-error "http://imperial.ac.uk/podcast"`	Check whether the site **http://imperial.ac.uk/podcast** is down
`curl --head --location "https://tinyurl.com/energetic-songs" | grep Location`	Expand a shortened or disguised URL: **https://tinyurl.com/energetic-songs** redirects to a public YouTube playlist. This is also helpful when you want to unearth the actual websites behind the long, convoluted, redirect-intensive email newsletter hyperlinks.

Downloading Files

The commands below come in handy when you want to scrape websites for content. The following commands return meaningful results as of writing. Change the parameters to suit your purposes.

COMMAND	DESCRIPTION
`curl --output hello.html http://example.com`	Outputs the URL `http://example.com` to a file `hello.html`
`curl -o`	Abbreviation of `curl --output. -o` only works if placed before the target URL parameter.
`curl --remote-name "https://theory.stanford.edu/~trevisan/books/crypto.pdf"`	Download a file from **https://theory.stanford.edu/~trevisan/books/crypto.pdf**, saving the file without changing its name
`curl --remote-name "https://theory.stanford.edu/~trevisan/books/crypto.pdf" --output cryptography_notes.pdf`	Download a file from **https://theory.stanford.edu/~trevisan/books/crypto.pdf** and rename it to `cryptography_notes.pdf` Alternatively, you may replace `--output` with >. Replacing `--output` with `-o` does not work here.
`curl --remote-name --continue-at - "https://theory.sta`	Continue a partial download of a file **https://theory.stanford.edu/~trevisan/books/crypto.pdf**

COMMAND	DESCRIPTION
`nford.edu/~trevisan/books/crypto.pdf"`	
`curl "https://en.wikipedia.org/wiki/{Linux,Windows,OSX}" --output "file_#1.html"`	Download files from multiple locations and name them according to the format `file_(operating system).html`
`curl "https://www.gutenberg.org/files/[158-161]/[158-161]-0.{txt,zip}" --output "bk#1_#2.#3"`	Download a sequence of files and outputs `bk158_158.txt, bk158_158.zip, …, bk161_161.zip`
`curl --location http://socialdance.stanford.edu/music/ | grep '.mp4' | cut -d \" -f 8 | while read i; do curl http://socialdance.stanford.edu/music/"${i}" -o "${i##*/}"; done`	Download all MP4 files from the URL **http://socialdance.stanford.edu/music/**. Here, use `grep` to filter out the MP4 files, `cut` to find the path to the required files (the delimiter is " and the path string was at the 8th such delimiter), A `while`-loop with `curl` helps download the files recursively. You'll need to modify the `grep` and `cut` commands to download other file types and locate relevant hyperlinks in the HTML source code of the URL you specify.

curl GET Commands

Use these commands to make a GET request using curl. curl GET commands may require you to pass authorization keys via the `--header` flag.

You can also make other **HTTP requests** such as PUT and DELETE using curl and the appropriate flags.

COMMAND	DESCRIPTION
`curl --request GET "http://example.com"`	Fetch the HTML source of the URL **http://example.com/** and output it in the terminal console

COMMAND	DESCRIPTION
`curl -X`	Abbreviation of `curl --request`
`curl --request GET 'https://us-east-1.aws.data.mongodb-api.com/app/viewdata-kqgls/endpoint/view?secret=ZAEOuvuEVLF5ll3kGP8FFkAj1GMKB8xu1jRx5D7210gXiZHa5agdbSq8pzbpI8Lo' --header 'Content-Type: application/json'`	Get all MongoDB documents from the `viewdata-kqgls` app with the given secret string and content type header as query parameters. The expected result is a JSON object containing all documents. (The URL is a **custom API endpoint** I made on MongoDB.)
`curl --request GET 'https://us-east-1.aws.data.mongodb-api.com/app/viewdata-kqgls/endpoint/view?secret=ZAEOuvuEVLF5ll3kGP8FFkAj1GMKB8xu1jRx5D7210gXiZHa5agdbSq8pzbpI8Lo&id=636b5046e54ce11139fd8b96' --header 'Content-Type: application/json'`	Get a MongoDB document from the `viewdata-kqgls` app with the given ID, secret string, and content type header as query parameters. The expected result is the document, if it exists:`{"_id":"636b5046e54ce11139fd8b96","name":"Alice Bob","age":25,"greeting":"Greetings, everyone."}`

curl POST Commands

Use these commands to make a POST request using curl. curl POST commands may require the `--header` flag to pass authorization keys.

You can also make other **HTTP requests** such as PUT and DELETE using curl and the appropriate flags.

COMMAND	DESCRIPTION
`curl --header`	Pass a header to the server URL
`curl -H`	Abbreviation of `curl --header`
`curl --request POST "http://example.com" -d 'some data'`	Fetch the HTML source of the URL **http://example.com/**
`curl -X`	Abbreviation of `curl --request`

COMMAND	DESCRIPTION
`curl --request POST 'https://data.mongodb-api.com/app/data-meetp/endpoint/data/v1/action/insertOne' --header 'Content-Type: application/json' --header 'api-key: ZAEOuvuEVLF5ll3kGP8FFkAj1GMKB8xu1jRx5D7210gXiZHa5agdbSq8pzbpI8Lo' --data-raw '{"dataSource": "Cluster0","database": "curlhacks","collection": "curlhacks","document": { "name": "Alice Bob", "age": 25, "greeting": "Greetings, everyone." }}'`	Upload via the **MongoDB Data API** the given Javascript object to a database and collection both named `curlhacks`. The expected output: `{"insertedId":"636b5046e54ce11139fd8b96"}` This means `curlhacks` has registered the new Javascript object as a MongoDB document with the given ID.
`curl --request POST 'https://data.mongodb-api.com/app/data-meetp/endpoint/data/v1/action/findOne' --header 'Content-Type: application/json' --header 'api-key: ZAEOuvuEVLF5ll3kGP8FFkAj1GMKB8xu1jRx5D7210gXiZHa5agdbSq8pzbpI8Lo' --data-raw '{"dataSource": "Cluster0","database": "curlhacks","collection": "curlhacks","filter": { "name": "Alice Bob" }}'`	Enquire via the **MongoDB Data API** the database and collection, both named `curlhacks`, for a document with the key-value pair `{"name": "Alice Bob"}`. The expected output is the requested document: `{"document":{"_id":"636b5046e54ce11139fd8b96","name":"Alice Bob","age":25,"greeting":"Greetings, everyone."}}`
`curl --request POST 'https://data.mongodb-api.com/app/data-meetp/endpoint/data/v1/action/deleteOne' --header 'Content-Type: application/json' --header 'api-key: ZAEOuvuEVLF5ll3kGP8FFkAj1GMKB8xu1jRx5D7210gXiZHa5agdbSq8pzbpI8Lo' --data-raw '{"dataSource": "Cluster0","database": "curlhacks","collection": "curlhacks","filter": { "_id": { "$oid": "636b4f88fd82bd55d90962c6" } }}'`	Delete via the **MongoDB Data API** a document with the given ID from the database and collection, both named `curlhacks`. The expected output: `{"deletedCount":1}` This means `curlhacks` has deleted a MongoDB document, namely the one specified.

API Interaction

The following commands can help you automate web query requests, such as Google Form submissions. The examples below are chock-full of Google Form URLs because of a real-life hack so egregious the full source code must remain private.

I wanted a news organization to win an award so badly, I generated 12,000+ submissions to the award nomination Google Form over two months using temporary email addresses and mix-and-match reasons. I was sad the media company didn't win, but if it did, it'd face the conundrum of having reported on voting fraud yet having voting fraud seal its victory.

Identifying the various fields in the award submission Google Form

Example of a Google Form URL which I could have submitted through curl, blanking out the Google Form ID and identifying information about the news company.

Go to **curl GET** and **curl POST** commands for GET- and POST-specific API interactions using curl.

COMMAND	DESCRIPTION
`curl "https://gitlab.com/api/v4/projects"`	Query an API endpoint
`curl --header "Auth-Token:$DB_APP_TOKEN" "https://example.com/api/v3/endpoint"`	Pass a header to a server URL. A header is a field of an HTTP request or response that passes additional context and metadata about

COMMAND	DESCRIPTION
	the request or response. In this example, the header is an authorization token.
`curl -H`	Abbreviation of `curl --header`
`curl --data "ABC 123" "https://docs.google.com/forms/d/e/[GoogleFormID]/formResponse"`	Send URL-encoded raw data `"ABC 123"` to an API endpoint, in this case a Google Form.
`curl -d`	Abbreviation of `curl --data`
`curl --data "ABC 123" "https://docs.google.com/forms/d/e/[GoogleFormID]/formResponse" > output.html`	Send URL-encoded raw data `"ABC 123"` to an API endpoint, in this case a Google Form, and output to output.html data returned from the server
`curl --form "emailAddress=test@myemail.com" --form "submit=Submit" "https://docs.google.com/forms/d/e/[GoogleFormID]/formResponse" > output.html`	Emulate sending an email address to an API endpoint (Google Form here) followed by pressing the Submit button. The output file, `output.html`, will have a filled email address field.
`curl -F`	Abbreviation of `curl --form`
`curl --form "entry.123456789=</Users/user1/Downloads/playlist.m3u" "https://docs.google.com/forms/d/e/[GoogleFormID]/formResponse" > output.html`	Send to an API endpoint (Google Form here) the file contents of `/Users/user1/Downloads/playlist.m3u` to the parameter `entry.123456789`. The symbol < here means you're sending data to the server, as opposed to > for data you receive from the server. You can find the parameters of the form `entry.123456789` (the number may not be nine digits long) using your browser's **Inspector**. On Chrome-based browsers, right-click the page and select "Inspect" to see the Inspector.

COMMAND	DESCRIPTION
	The output file, `output.html`, will show the file contents in the corresponding field.
`curl --form "entry.123456789=</Users/user1/Downloads/playlist.m3u" --form "emailAddress=test@myemail.com" "https://docs.google.com/forms/d/e/[GoogleFormID]/formResponse"`	Send more than one piece of data to the given API endpoint. This command sends over the email and playlist file specified. The output for this command will be in the terminal.
`curl --data "entry.123456789=</Users/user1/Downloads/playlist.m3u&emailAddress=test@myemail.com" "https://docs.google.com/forms/d/e/[GoogleFormID]/formResponse"`	Similarly as above, send more than one piece of data to the given API endpoint. This command sends over the email and the raw data string"`</Users/user1/Downloads/playlist.m3u`". The output for this command will be in the terminal.
`curl --form "input=@pic1.jpg" "https://www.iloveimg.com/resize-image" > output.html` `curl --form "input=/Users/user1/Downloads/pic1.jpg" "https://www.iloveimg.com/resize-image" > output.html`	Send a file `/Users/user1/Downloads/pic1.jpg` as form data to the given API endpoint. Both commands are equivalent. They send an image file to `https://www.iloveimg.com/resize-image`. – Use @ if the file is in the current working directory (obtained via `pwd`); – Don't use @ if you provide the full directory path of the file. The output file, `output.html`, will show the image-resizing options returned by the API.

Cookies

It appears that the sole action of sending cookies to the target website doesn't affect the HTML layout of the website. Nevertheless, curl supports the following methods:

COMMAND	DESCRIPTION
`curl --cookie "registered=yes"`	Send `"registered=yes"` as cookie
`curl --cookie "name=alice;email=test@myemail.com"`	Send “name=alice” and `"email=test@myemail.com"` as cookies
`curl --cookie import_cookies.txt`	Send the contents of `import_cookies.txt` as cookie(s). As most browsers no longer support the `“Set-Cookie:”` prefix, format your cookies in the file as: `key1=value1;key2=value2`
`curl -b`	Abbreviation of `--cookie`
`curl --cookie-jar mycookies.txt`	Write cookies to `mycookies.txt` after executing the curl operation on other flags
`curl -c`	Abbreviation of `--cookie-jar`
`curl --dump-header headers_and_cookies.txt http://example.com`	Output HTTP headers and cookie data of **http://example.com** to `headers_and_cookies.txt`
`curl -D`	Abbreviation of `curl --dump-header`

curl Script

You can use curl commands in bash scripts. Here are some example scripts involving curl commands:

EXAMPLE	DESCRIPTION
curl-install-package.sh	Install packages with curl
curl-url-time.sh	Check a website response time
curl-format-json.sh	Beautify json output for curl response
curl-remote-scripts.sh	curl run remote scripts

curl Advanced

Here are some commands for fine-tuning your curl operations.

COMMAND	DESCRIPTION
`curl -h`	Show help commands
`curl --version`	Show curl version
`curl -v ftp://ftp.corel.com/`	Get verbose output while connecting to the URL **ftp://ftp.corel.com/** You may use this -v flag along with other flags such as `--head`, `--location`.
`curl --trace ftp_corel.txt https://twitter.com/`	Get details of the packets captured in the connection to the URL **https://twitter.com/**
`curl -s https://twitter.com/ > twitter.html`	Download the URL **https://twitter.com/** in silent mode, not outputting the progress
`curl -L "https://twitter.com/search" --connect-timeout 0.1`	Specify the maximum time in seconds (0.1 seconds in this example) allowed to connect to the URL **https://twitter.com/search**
`curl -s -w '%{remote_ip} %{time_total} %{http_code} \n' -o /dev/null http://apkpush.io`	Return the specified parameter values as a string `'%{remote_ip} %{time_total} %{http_code} \n'` on the terminal output and suppress all other
`curl -r 0-99 http://example.com`	**http://example.com/**
`curl -r -500 http://example.com`	Get the last 500 bytes of the URL **http://example.com/**
`curl -r 0-99 ftp://ftp.corel.com`	Get the first 100 bytes of an FTP URL. curl only supports ranges with explicit start and end positions.

COMMAND	DESCRIPTION
`curl -m 0.1`	Specify maximum operation time in seconds (0.1s here)

curl Request Example

Let's conclude this article with a curl POST request hack. Proceed at your own risk.

COMMAND	DESCRIPTION	TEST RESULT
`curl -X POST https://textbelt.com/text --data-urlencode phone='+[area code][phone number]' --data-urlencode message='Please delete this message. This is a service provided by textbelt.' -d key=textbelt`	Send a free SMS text message to a phone number in E.164 format via **https://textbelt.com/** with the API key `textbelt`. If you have a custom API key, replace "`textbelt`" with it.	On the terminal:`{"success":true,"textId":"205381667028627395","quotaRemaining":0}` On the phone:

We hope this curl cheat sheet helps you to explore curl and its uses. Happy curl hacking!

Frequently Asked Questions

What are curl commands?

curl commands are commands issued using the command-line application curl, mostly requests to

How do you use the curl method?

What is a flag in curl?

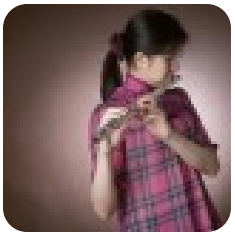
Does curl use TCP or UDP?

How is curl different from WGET?

Grow your Cyber Security Skills

FIND OUT MORE

00:00 00:00 1



Cassandra Lee

I make connections across disciplines: cyber security, writing/journalism, art/design, music, mathematics, technology, education, psychology, and more. I've been advocating for girls and women in STEM since the 2010s, having written for Huffington Post, International Mathematical Olympiad 2016, and Ada Lovelace Day, and I'm honored to join StationX. You can find me on **LinkedIn** and **Linktree**.

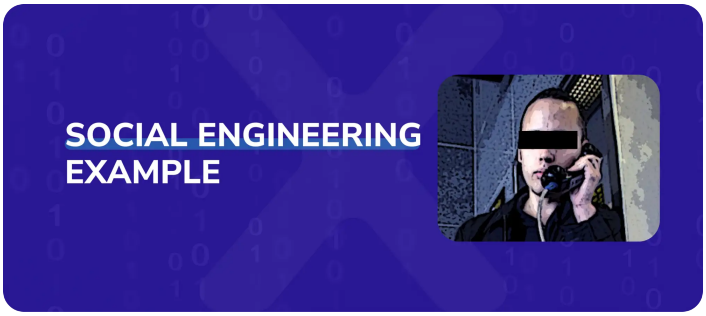
Related Articles

Nmap Cheat Sheet 2023: All the Commands, Flags & Switches

Read More »

Linux Command Line Cheat Sheet: All the Commands You Need

Read More »



00:00 00:00 1

Read More »

INFO

- Affiliates
- Legal Notices
- Privacy Policy
- Site Map

SECURITY ASSESSMENT

- Penetration Testing
- Vulnerability Scanning
- Build Reviews
- Source Code Review
- Social Engineering

CONSULTING

- Audit & Compliance
- Incident Response
- Security Architecture
- Risk Assessment
- Security Training