curl
by example

Anton Zhiyanov

Table of contents

5. Handling URLs

6. Additional features

Preface

Curl (**c**lient for **URL**s) is a tool for client-side internet transfers (uploads and downloads) using a specific protocol (such as HTTP, FTP or IMAP), where the endpoint is identified by a URL. Curl runs on 92 operating systems and has over 20 billion installations worldwide.

Curl has extensive reference documentation and even a 500-page book devoted entirely to it. I wanted something lighter, so I made this interactive step-by-step guide to essential curl operations. You can read it from start to finish to (hopefully) learn more about curl, or jump to a specific use case that interests you.

The guide is available in the following formats:

- Interactive online cookbook
- Bookmarkable playground
- PDF minibook

The guide is partially based on the 3.5-hour workshop Mastering the curl command line by Daniel Stenberg, the author of curl.

1. HTTP concepts

Curl is mostly used to work with HTTP, so let's talk about it. I'll try to keep the theory short and simple.

HTTP protocol

HTTP/1.x is a plain-text protocol that describes the communication between the client and the server. The client sends messages like this:

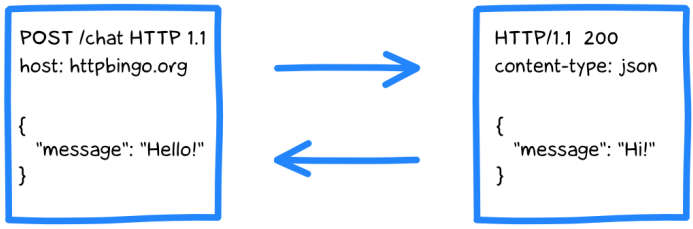
```
POST /anything/chat HTTP/1.1
host: httpbingo.org
content-type: application/json
user-agent: curl/7.87.0

{
    "message": "Hello!"
}
```

And receives messages like this in response:

```
HTTP/1.1 200 OK
date: Mon, 28 Aug 2023 07:51:49 GMT
content-type: application/json

{
    "message": "Hi!"
}
```



It's easy to read requests and responses once you get used to it.

HTTP/2, the successor to HTTP/1.1, is a binary protocol. However, curl displays HTTP/2 messages in plain text (just like HTTP/1.1), so we can safely ignore this fact for our purposes.

HTTP request

HTTP request consists of three main sections:

1. Request line:

```
POST /anything/chat HTTP/1.1
```

- The *method* (`POST`) defines the operation the client wants to perform.
- The *path* (`/anything/chat`) is the URL of the requested resource (without the protocol, domain and port).
- The *version* (`HTTP/1.1`) indicates the version of the HTTP protocol.

2. Request headers:

```
host: httpbingo.org
content-type: application/json
user-agent: curl/7.87.0
```

Each header is a key-value pair that tells the server some useful information about the request. In our case it's the hostname of the server (`httpbingo.org`), the type of the content (`application/json`) and the client's self-identification (`user-agent`).

3. Request body:

```
{
    "message": "Hello!"
}
```

The actual data that the client sends to the server.

The HTTP protocol is stateless, so any state must be contained within the request itself, either in the headers or in the body.

HTTP response

HTTP response also consists of three main sections:

1. Status line:

```
HTTP/1.1 200 OK
```

- The *version* (`HTTP/1.1`) indicates the version of the HTTP protocol.
- The *status code* (`200`) tells whether the request was successful or not, and why (there are many status codes for different situations).
- The *status message* is a human-readable description of the status code. HTTP/2 does not have it.

2. Response headers:

```
date: Mon, 28 Aug 2023 07:51:49 GMT
content-type: application/json
```

Similar to request headers, these provide useful information about the response to the client.

3. Response body:

```
{
    "message": "Hi!"
}
```

The actual data that the server sends to the client.

There is much more to the HTTP protocol, but this basic knowledge should be enough for our purposes. So let's move on.

2. Basic HTTP

Now let's see how to work with HTTP in curl.

Methods

Curl supports all HTTP methods (sometimes called *verbs*).

GET (the default one, requires no options):

```
curl http://httpbingo.org/get
```

```
{
    "method": "GET",
    "url": "http://httpbingo.org/get"
}
```

> Httpbin (`httpbingo.org`) is an HTTP debugging service. Its `/get` handle returns all the details of the incoming GET request as a JSON object. So the JSON you see in the response comes from Httpbin, not from curl itself.
>
> Httpbin provides many similar handles for various HTTP features. We'll use it extensively throughout this guide.

HEAD (`-I` / `--head` , returns headers only):

```
curl --head http://httpbingo.org/head
```

```
{
    "method": "HEAD",
    "url": "http://httpbingo.org/head"
}
```

POST (`-d` / `--data` for data or `-F` / `--form` for HTTP form):

```
curl --data "name=alice" http://httpbingo.org/post
```


```
{
    "args": {},
    "headers": {
        "Content-Type": ["application/x-www-form-urlencoded"]
    },
    "method": "POST",
    "url": "http://httpbingo.org/post",
    "data": "name=alice",
    "form": {
        "name": ["alice"]
    }
}
```


Or any other method with `-X` (`--request`):

```
curl --request PATCH --data "name=alice" \
  http://httpbingo.org/patch
```


```
{
    "args": {},
    "headers": {
        "Content-Type": ["application/x-www-form-urlencoded"]
    },
    "method": "PATCH",
    "url": "http://httpbingo.org/patch",
    "data": "name=alice",
    "form": {
        "name": ["alice"]
    }
}
```

Response code

Typically, status codes 2xx (specifically 200) are considered “success”, while 4xx are treated as client-side errors and 5xx as server-side errors. But curl doesn’t care about codes: to it, every HTTP response is a success:

```
curl http://httpbingo.org/status/503 && echo OK
```

```
OK
```

To make curl treat 4xx and 5xx codes as errors, use `-f` (`--fail`):

```
curl --fail http://httpbingo.org/status/503 && echo OK
```

```
curl: (22) The requested URL returned error: 503 (exit status 22)
```

To print the response code, use `-w` (`--write-out`) with the `response_code` variable:

```
curl --write-out "%{response_code}" \
  http://httpbingo.org/status/200
```

```
200
```

The `--write-out` option extracts specific parts of the response. We’ll talk about it in more detail in the *Advanced HTTP* chapter.

Response headers

To display response headers along with the body, use `-i` (`--include`):

```
curl --include http://httpbingo.org/uuid
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Date: Mon, 25 Mar 2024 09:04:02 GMT
Content-Length: 53

{
  "uuid": "eb5e41e1-4a4b-4358-9665-53857b399c1a"
}
```

Or save them to a file using `-D` (`--dump-header`):

```
curl --dump-header /tmp/headers \
  http://httpbingo.org/uuid >/dev/null

cat /tmp/headers
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Date: Mon, 25 Mar 2024 09:04:02 GMT
Content-Length: 53
```

Response body

Response body, sometimes called *payload*, is what curl outputs by default:

```
curl http://httpbingo.org/get
```

```
{
    "headers": {
        "Accept": ["*/*"]
    },
    "method": "GET",
    "url": "http://httpbingo.org/get"
}
```

You can ask the server to compress the data with `--compressed`, but curl will still show it as uncompressed:

```
curl --compressed http://httpbingo.org/get
```

```
{
    "headers": {
        "Accept": ["*/*"],
        "Accept-Encoding": ["deflate, gzip, br"]
    },
    "method": "GET",
    "url": "http://httpbingo.org/get"
}
```

(note the Accept-Encoding request header added to the request)

POST

POST sends data to the server. By default, it's a set of key-value pairs encoded in a single string with a `application/x-www-form-urlencoded` Content-Type header.

Use `-d` (`--data`) to specify individual key-value pairs (or the entire string):

```
curl --data name=alice --data age=25 \
  http://httpbingo.org/post
```


```
{
    "headers": {
        "Content-Type": ["application/x-www-form-urlencoded"]
    },
    "method": "POST",
    "url": "http://httpbingo.org/post",
    "data": "name=alice&age=25",
    "form": {
        "age": ["25"],
        "name": ["alice"]
    }
}
```


To send data from a file, use `@` with a file path. Use `-H` (`--header`) to change the Content-Type header with according to the file contents:

```
echo "Alice, age 25" > /tmp/data.txt

curl --data @/tmp/data.txt \
  --header "content-type: text/plain" \
  http://httpbingo.org/post
```

```
{
    "headers": {
        "Content-Type": ["text/plain"]
    },
    "method": "POST",
    "url": "http://httpbingo.org/post",
    "data": "Alice, age 25"
}
```

`--data-raw` posts data similar to `--data`, but without the special interpretation of the `@` character.

To post JSON data, use `--json`. It automatically sets the Content-Type and Accept headers accordingly:

```
curl --json '{"name": "alice"}' http://httpbingo.org/post
```

```
{
    "args": {},
    "headers": {
        "Accept": ["application/json"],
        "Content-Type": ["application/json"]
    },
    "method": "POST",
    "url": "http://httpbingo.org/post",
    "data": "{\"name\": \"alice\"}",
    "json": {
        "name": "alice"
    }
}
```

To URL-encode data (escape all symbols not allowed in URLs), use `--data-urlencode`:

```
curl --data-urlencode "name=Barton, Alice" \
  http://httpbingo.org/post
```

```
{
    "args": {},
    "headers": {
        "Content-Type": ["application/x-www-form-urlencoded"]
    },
    "method": "POST",
    "url": "http://httpbingo.org/post",
    "data": "name=Barton%2C+Alice",
    "form": {
        "name": ["Barton, Alice"]
    }
}
```

PUT

The PUT method is often used to send files to the server. Use `-T` (`--upload-file`) for this:

```
echo hello > /tmp/hello.txt
curl --upload-file /tmp/hello.txt http://httpbingo.org/put
```

```
{
    "method": "PUT",
    "url": "http://httpbingo.org/put",
    "data": "data:application/octet-stream;base64,aGVsbG8K"
}
```

Sometimes PUT is used for requests in REST APIs. For these, use `-X` (`--request`) to set the method and `-d` (`--data`) to send the data:

```
curl --request PUT \
  --header "content-type: application/json" \
  --data '{"name": "alice"}' \
  http://httpbingo.org/put
```

```
{
    "headers": {
        "Content-Type": ["application/json"]
    },
    "method": "PUT",
    "url": "http://httpbingo.org/put",
    "data": "{\"name\": \"alice\"}",
    "json": {
        "name": "alice"
    }
}
```

3. Advanced HTTP

The HTTP protocol has lots of features. We won't cover them all, but let's take a look at some of the more advanced ones.

Response variables

To extract specific information about the response, use `-w` (`--write-out`). It supports over 50 variables. For example, here we extract the status code and response content type:

```
curl --write-out "\HTTP %{response_code}\n %{content_type}" \
  http://httpbingo.org/status/429
```

```
HTTP 429
text/plain; charset=utf-8
```

Or some response headers:

```
curl --write-out "\n%header{date}\n%header{content-length} B" \
  http://httpbingo.org/uuid
```

```
{
  "uuid": "e047eb1c-4ada-484e-a98e-0bf4fb978b89"
}

Sun, 24 Mar 2024 12:11:41 GMT
53 B
```

Redirects

A *redirect* is when the server, instead of returning the requested resource, tells the client that the resource is located elsewhere (as indicated by the Location header). A redirect always has a 3xx response code.

Curl does not follow redirects by default, it returns the response as is:

```
curl --include http://httpbingo.org/redirect/1
```

```
HTTP/1.1 302 Found
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Location: /get
Date: Sun, 24 Mar 2024 12:47:39 GMT
Content-Length: 0
```

Note the *302 Found* status and the *Location* response header, which points to the redirected location.

To make curl follow redirects, use `-L` (`--location`):

```
curl --location http://httpbingo.org/redirect/1
```

```
{
    "method": "GET",
    "url": "http://httpbingo.org/get"
}
```

To protect against endless loop redirects, use `--max-redirs`:

```
curl --location --max-redirs 3 http://httpbingo.org/redirect/10
```

```
curl: (47) Maximum (3) redirects followed (exit status 47)
```

Ranges

To ask the server for a piece of data instead of the whole thing, use the `r` (`--range`) option. This will cause curl to request the specified byte range.

For example, here we request 50 bytes starting with the 100th byte:

```
curl --range 100-150 http://httpbingo.org/range/1024
```

```
wxyzabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstu
```

Note that the server may ignore the ask and return the entire response.

If you are downloading data from a server, you can also use `-C` (`--continue-at`) to continue the previous transfer at the specified offset:

```
curl --continue-at 1000 http://httpbingo.org/range/1024
```

```
mnopqrstuvwxyzabcdefghij
```

Conditional requests

Conditional requests are useful when you want to avoid downloading already downloaded data (assuming it is not stale). Curl supports two different conditions: file timestamp and etag.

To set a timestamp condition, use `-z` (`--time-cond`).

Download the data only if the remote resource is newer (condition holds):

```
curl --time-cond "Aug 30, 2023" http://httpbingo.org/etag/etag
```

```
{
    "headers": {
        "If-Modified-Since": ["Wed, 30 Aug 2023 00:00:00 GMT"]
    },
    "method": "GET",
    "url": "http://httpbingo.org/etag/etag"
}
```

Or older (condition fails):

```
curl --include --time-cond "-Aug 30, 2023" \
  http://httpbingo.org/etag/etag
```

```
HTTP/1.1 412 Precondition Failed
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Content-Type: text/plain; charset=utf-8
Etag: "etag"
Last-Modified: Sun, 24 Mar 2024 12:36:04 GMT
Date: Sun, 24 Mar 2024 12:36:04 GMT
Content-Length: 0
```

Etag conditions are a bit more involved. An *etag* is a value returned by the server that uniquely identifies the current version of the requested resource. It is often a hash of the data.

To check an etag, curl must first to save it with `--etag-save`:

```
curl --etag-save /tmp/etags \
  http://httpbingo.org/etag/20230830 >/dev/null
```

```
{
    "method": "GET",
    "url": "http://httpbingo.org/etag/etag"
}
```

```
cat /tmp/etags
# 20230830 is the etag value
```

```
"20230830"
```

And use `--etag-compare` in subsequent requests:

```
curl --include --etag-compare /tmp/etags \
  http://httpbingo.org/etag/20230830
```

```
HTTP/1.1 304 Not Modified
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Etag: "etag"
Date: Sun, 24 Mar 2024 12:41:14 GMT
```

Timestamp conditions rely on the *Last-Modified* response header, so if the server does not provide it, the resource will always be considered newer. The same goes for etag conditions and the *Etag* response header.

Multipart formpost

POST can send data as a sequence of “parts” with a `multipart/form-data` content type. It’s often used for HTML forms that contain both text fields and files.

Each part has a name, headers, and data. Parts are separated by a “mime boundary”.

To construct multipart requests with curl, use `F` (`--form`). Each of these options adds a part to the request:

```
touch /tmp/alice.png

curl --form name=Alice --form age=25 \
  --form photo=@/tmp/alice.png \
  http://httpbingo.org/dump/request
```

```
POST /dump/request HTTP/1.1
Host: httpbingo.org
Accept: */*
Content-Length: 403
Content-Type: multipart/form-data; boundary=d74496d66958873e
User-Agent: curl/8.5.0

--d74496d66958873e
Content-Disposition: form-data; name="name"

Alice
--d74496d66958873e
Content-Disposition: form-data; name="age"

25
--d74496d66958873e
Content-Disposition: form-data; name="photo"; filename="alice.png"
Content-Type: image/png


--d74496d66958873e--
```

Cookies

The HTTP protocol is stateless. Cookies are an ingenious way around this:

1. The server wants to associate some state with the client session.
2. The server returns that state in the Set-Cookie response header.
3. The client recognizes the cookies and sends them back with each request in the Cookie request header.

Each cookie has an expiration date — either explicit one or "end of session" one (for browser clients, this is often when the user closes the browser).

Curl ignores cookies by default. To enable them, use the `-b` (`--cookie`) option. To make curl persist cookies between calls, use `-c` (`--cookie-jar`).

Here the server sets the cookie `sessionid` to `123456` and curl stores it in the cookie jar `/tmp/cookies`:

```
curl --cookie "" --cookie-jar /tmp/cookies \
  http://httpbingo.org/cookies/set?sessionid=123456
```

```
cat /tmp/cookies
```

```
# Netscape HTTP Cookie File
# https://curl.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

#HttpOnly   FALSE   /cookies/   FALSE   0   sessionid   123456
```

Subsequent curl calls with `--cookie` pointing to the cookie jar (`/tmp/cookies`) will send the `sessionid` cookie back to the server:

```
curl --cookie /tmp/cookies http://httpbingo.org/cookies
```

```
{
    "sessionid": "123456"
}
```

Curl automatically discards cookies from the cookie jar when they expire (this requires an explicit expiration date set by the server). To discard session-based cookies, use `-j` (`--junk-session-cookies`):

```
curl --junk-session-cookies --cookie /tmp/cookies \
  http://httpbingo.org/cookies
```

```
{}
```

HTTP versions

By default, curl uses HTTP/1.1 for the `http` scheme and HTTP/2 for `https` . You can change this with flags:

```
--http0.9
--http1.0
--http1.1
--http2
--http3
```

To find out which version the server supports, use the `http_version` response variable:

```
curl --write-out "%{http_version}" http://httpbingo.org/status/200
```

```
1.1
```

4. Controlling transfers

Curl is a tool for internet transfers — uploads and downloads. Let's see how to control them.

Downloads

To write the response to the specified file instead of stdout, use `-o` (`--output`):

```
curl --output /tmp/uuid.txt http://httpbingo.org/uuid
cat /tmp/uuid.txt
```

```
{ "uuid": "5f8fdedd-0956-40e2-bbae-99e5861991f5" }
```

If you are only interested in specific parts of the response, extract them with `--write-out` and discard the rest with `--output /dev/null` :

```
curl --write-out "HTTP %{response_code}\n" \
  --output /dev/null \
  http://httpbingo.org/get
```

```
HTTP 200
```

`-O` (`--remote-name`) tells curl to save the output to a file specified by the URL (specifically, by the part after the last `/`). It's often used together with `--output-dir` , which tells curl where exactly to save the file:

```
curl --remote-name --output-dir /tmp \
  http://httpbingo.org/uuid

cat /tmp/uuid
```

```
{ "uuid": "5f8fdedd-0956-40e2-bbae-99e5861991f5" }
```

If the directory does not exist, `--output-dir` won't create it for you. Use `--create-dirs` for this:

```
curl --remote-name \
  --output-dir /tmp/some/place --create-dirs \
  http://httpbingo.org/uuid

cat /tmp/some/place/uuid
```

```
{ "uuid": "3daff6d5-3cfc-4218-84eb-68de2f50601b" }
```

Uploads

Curl is often used to download data from the server, but you can also upload it. Use the `-T` (`--upload-file`) option:

```
echo hello > /tmp/hello.txt
curl --upload-file /tmp/hello.txt http://httpbingo.org/put
```

```
{
    "method": "PUT",
    "url": "http://httpbingo.org/put",
    "data": "data:application/octet-stream;base64,aGVsbG8K"
}
```

For HTTP uploads, curl uses the `PUT` method.

Name resolving

By default, curl uses your DNS server to resolve hostnames to IP addresses. But you can force it to resolve to a specific IP with `--resolve` (using the same port):

```
curl --resolve httpbingo.org:8080:127.0.0.1 \
  http://httpbingo.org:8080/get
```

```
curl: (7)
Failed to connect to httpbingo.org port 8080 after 0 ms:
Couldn't connect to server (exit status 7)
```

(this one fails because no one is listening on 127.0.0.1:8080)

Or you can even map a `hostname:port` pair to another `hostname:port` pair with `--connect-to`:

```
curl --connect-to httpbingo.org:8080:httpbingo.org:80 \
  http://httpbingo.org:8080/get
```

```
{
    "method": "GET",
    "url": "http://httpbingo.org:8080/get"
}
```

(this one works fine, because Httpbin answers on port 80)

Transfer controls

To stop slow transfers, set the minimum allowed download speed (in bytes per second) with `--speed-limit`. By default, curl checks the speed in 30 seconds intervals, but you can change this with `--speed-time`.

For example, allow no less than 10 bytes/sec during a 3-second interval:

```
curl --speed-limit 10 --speed-time 3 http://httpbingo.org/get
```

```
{
    "method": "GET",
    "url": "http://httpbingo.org/get"
}
```

To limit bandwidth usage, set `--limit-rate`. It accepts anything from bytes to petabytes:

```
curl --limit-rate 3 http://httpbingo.org/get
curl --limit-rate 3k http://httpbingo.org/get
curl --limit-rate 3m http://httpbingo.org/get
curl --limit-rate 3g http://httpbingo.org/get
curl --limit-rate 3t http://httpbingo.org/get
curl --limit-rate 3p http://httpbingo.org/get
```

Another thing to limit is the number of concurrent requests (e.g. if you download a lot of files). Use `--rate` for this. It accepts seconds, minutes, hours or days:

```
curl --rate 3/s http://httpbingo.org/anything/[1-9].txt
curl --rate 3/m http://httpbingo.org/anything/[1-9].txt
curl --rate 3/h http://httpbingo.org/anything/[1-9].txt
curl --rate 3/d http://httpbingo.org/anything/[1-9].txt
```

Timeouts

To limit the maximum amount of time curl will spend interacting with a single URL, use `--max-time` (in fractional seconds):

```
curl --max-time 0.5 http://httpbingo.org/delay/1
```

```
curl: (28)
Operation timed out after 502 milliseconds
with 0 bytes received (exit status 28)
```

(this one fails)

Instead of limiting the total time, you can use `--connect-timeout` to limit only the time it takes to establish a network connection:

```
curl --connect-timeout 0.5 http://httpbingo.org/delay/1
```

```
{
    "method": "GET",
    "url": "http://httpbingo.org/delay/1"
}
```

(this one works fine)

Retries

Sometimes the remote host is temporarily unavailable. To deal with these situations, curl provides the `--retry [num]` option. If a request fails, curl will try it again, but no more than `num` times:

```
curl --include --retry 3 http://httpbingo.org/unstable
```

```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Content-Type: text/plain; charset=utf-8
Date: Sun, 24 Mar 2024 12:16:38 GMT
Content-Length: 0

HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Content-Type: text/plain; charset=utf-8
Date: Sun, 24 Mar 2024 12:16:39 GMT
Content-Length: 0
```

(this URL fails 50% of the time)

You can set the maximum time curl will spend retrying with `--retry-max-time` (in seconds) or the delay between retries with `--retry-delay` (also in seconds):

```
curl --include \
  --retry 3 --retry-max-time 2 --retry-delay 1 \
  http://httpbingo.org/unstable
```

```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Content-Type: text/plain; charset=utf-8
Date: Sun, 24 Mar 2024 12:17:16 GMT
Content-Length: 0

HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Content-Type: text/plain; charset=utf-8
Date: Sun, 24 Mar 2024 12:17:17 GMT
Content-Length: 0
```

For curl, “request failed” means one of the following HTTP codes: 408, 429, 500, 502, 503 or 504. If the request fails with a “connection refused” error, curl will not retry. But you can change this with `--retry-connrefused`, or even enable retries for all kinds of problems with `--retry-all-errors`.

5. Handling URLs

Curl supports URLs (URIs, really) similar to how RFC 3986 defines them:

```
scheme://user:password@host:port/path?query#fragment
```

- `scheme` defines a protocol (like `https` or `ftp`). If omitted, curl will try to guess one.
- `user` and `password` are authentication credentials (passing credentials in URLs is generally not used anymore for the security reasons).
- `host` is the hostname, domain name or IP address of the server.
- `port` is the port number. If omitted, curl will use the default port associated with the scheme (such as 80 for `http` or 443 for `https`).
- `path` is the path to the resource on the server.
- `query` is usually a sequence of `name=value` pairs separated by `&`.

For curl, anything starting with `-` or `--` is an option, and everything else is a URL.

URL parameters

If you pass a lot of URL parameters, the query part can become quite long. The `--url-query` option allows you to specify query parts separately:

```
curl --url-query "name=Alice" --url-query "age=25" \
  http://httpbingo.org/get
```

```
{
    "args": { "age": ["25"], "name": ["Alice"] },
    "method": "GET",
    "url": "http://httpbingo.org/get?name=Alice&age=25"
}
```

Multiple URLs

Curl accepts any number of URLs, each of which requires a destination — stdout or a file. For example, this command saves the first UUID to `/tmp/uuid1.json` and the second UUID to `/tmp/uuid2.json`:

```
curl \
  --output /tmp/uuid1.json http://httpbingo.org/uuid \
  --output /tmp/uuid2.json http://httpbingo.org/uuid
cat /tmp/uuid1.json
cat /tmp/uuid2.json
```

```
{ "uuid": "4e5ee0e3-4bed-467c-b523-e9bdc202d79d" }
{ "uuid": "e3cd85d2-e6c3-4570-bc78-f752710eec96" }
```

The `-O` (`--remote-name`) derives the filename from the URL:

```
curl --output-dir /tmp \
  --remote-name http://httpbingo.org/anything/one \
  --remote-name http://httpbingo.org/anything/two
ls /tmp
```

```
one
two
```

To write both responses to the same file, you can use redirection:

```
curl http://httpbingo.org/uuid \
  http://httpbingo.org/uuid > /tmp/uuid.json
cat /tmp/uuid.json
```

```
{ "uuid": "a086befd-2d85-4133-a53e-05fd41ba8f6c" }
{ "uuid": "a752166d-452b-4d72-ad03-2bc8ddbeb77e" }
```

URL globbing

Curl automatically expands glob expressions in URLs into multiple specific URLs.

For example, this command requests three different paths (`al`, `bt`, `gm`), each with two different parameters (`num=1` and `num=2`), for a total of six URLs:

```
curl --output-dir /tmp --output "out_#1_#2.txt" \
  http://httpbingo.org/anything/{al,bt,gm}?num=[1-2]

ls /tmp
```

```
out_al_1.txt
out_al_2.txt
out_bt_1.txt
out_bt_2.txt
out_gm_1.txt
out_gm_2.txt
```

You can disable globbing with the `--globoff` option if `[]{}` characters are valid in your URLs. Then curl will treat them literally.

State reset

When you set options, they apply to all URLs curl processes. For example, here both headers are sent to both URLs:

```
curl \
  -H "x-num: one" http://httpbingo.org/headers?1 \
  -H "x-num: two" http://httpbingo.org/headers?2
```

```
{
  "headers": { "X-Num": [ "one", "two" ] }
}
{
  "headers": { "X-Num": [ "one", "two" ] }
}
```

Sometimes that's not what you want. To reset the state between URL calls, use the `--next` option:

```
curl \
  -H "x-num: one" http://httpbingo.org/headers?1 \
  --next \
  -H "x-num: two" http://httpbingo.org/headers?2
```

```
{
  "headers": { "X-Num": [ "one" ] }
}
{
  "headers": { "X-Num": [ "two" ] }
}
```

6. Additional features

Let's talk about some additional curl features you might find useful.

Verbose

`-v` (`--verbose`) makes curl verbose, which is useful for debugging:

```
# curl prints the debug information to stderr,
# so I have to redirect it to stdout with `2>&1`
curl --verbose http://httpbingo.org/uuid 2>&1
```

```
* Host httpbingo.org:80 was resolved.
* IPv6: (none)
* IPv4: 172.19.0.4
*   Trying 172.19.0.4:80 ...
* Connected to httpbingo.org (172.19.0.4) port 80
> GET /uuid HTTP/1.1
> Host: httpbingo.org
> User-Agent: curl/8.5.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Access-Control-Allow-Credentials: true
< Access-Control-Allow-Origin: *
< Content-Type: application/json; charset=utf-8
< Date: Sun, 24 Mar 2024 12:09:50 GMT
< Content-Length: 53
<
{ [53 bytes data]
{
  "uuid": "2afb5271-c403-43e7-8785-4f852dc706a7"
}
* Connection #0 to host httpbingo.org left intact
```

If `--verbose` is not enough, try `--trace` or `--trace-ascii` (the single `-` sends the trace output to stdout):

```
curl --trace-ascii - http://httpbingo.org/uuid
```

```
== Info: Host httpbingo.org:80 was resolved.
== Info: IPv6: (none)
== Info: IPv4: 172.19.0.4
== Info:   Trying 172.19.0.4:80 ...
== Info: Connected to httpbingo.org (172.19.0.4) port 80
=> Send header, 80 bytes (0x50)
0000: GET /uuid HTTP/1.1
0014: Host: httpbingo.org
0029: User-Agent: curl/8.5.0
0041: Accept: */*
004e:
<= Recv header, 17 bytes (0x11)
0000: HTTP/1.1 200 OK
<= Recv header, 40 bytes (0x28)
0000: Access-Control-Allow-Credentials: true
<= Recv header, 32 bytes (0x20)
0000: Access-Control-Allow-Origin: *
<= Recv header, 47 bytes (0x2f)
0000: Content-Type: application/json; charset=utf-8
<= Recv header, 37 bytes (0x25)
0000: Date: Sun, 24 Mar 2024 12:09:27 GMT
<= Recv header, 20 bytes (0x14)
0000: Content-Length: 53
<= Recv header, 2 bytes (0x2)
0000:
<= Recv data, 53 bytes (0x35)
0000: {.  "uuid": "41493d2a-de20-4e25-b747-5fbf147be98b".}.
{
  "uuid": "41493d2a-de20-4e25-b747-5fbf147be98b"
}
== Info: Connection #0 to host httpbingo.org left intact
```

Progress meters

Curl has two progress meters. The default is verbose:

```
# I have a `silent` option in my config file, so I have
# to turn it off explicitly. By default, it's not set,
# so `--no-silent` is not needed.

# Also, curl prints the progress to stderr,
# so I have to redirect it to stdout with `2>&1`.

curl --no-silent http://httpbingo.org/uuid 2>&1
```

```
% Total    % Received % Xferd  Average Speed   Time    Time     T
                                 Dload  Upload   Total   Spent

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- -
100    53  100    53    0     0  21900      0 --:--:-- --:--:-- -
```

The other is compact:

```
curl --no-silent --progress-bar http://httpbingo.org/uuid 2>&1
```

```
############################################################ 100.0%
```

The `--silent` option turns the meter off completely:

```
curl --silent http://httpbingo.org/uuid 2>&1
```

```
{
    "uuid": "9a0e4135-7f6c-4a47-b3af-949d55289b68"
}
```

Credentials

You almost never want to pass the username and password in the curl command itself. One way to avoid this is to use the `.netrc` file. It specifies hostnames and credentials for accessing them:

```
machine httpbingo.org
login alice
password cheese

machine example.com
login bob
password nuggets
```

Pass the `--netrc` option to use the `$HOME/.netrc` file, or `--netrc-file` to use a specific one:

```
echo "machine httpbingo.org" > /tmp/netrc
echo "login alice" >> /tmp/netrc
echo "password cheese" >> /tmp/netrc

curl --netrc-file /tmp/netrc \
  http://httpbingo.org/basic-auth/alice/cheese
```

```
{
    "authorized": true,
    "user": "alice"
}
```

Config file

As the number of options increases, the curl command becomes harder to decipher. To make it more readable, you can prepare a config file that lists one option per line (`–` is optional):

```
output-dir /tmp
show-error
silent
```

By default, curl reads the config from `$HOME/.curlrc`, but you can change this with the `-K` (`--config`) option:

```
curl --config /sandbox/.curlrc http://httpbingo.org/uuid
```

```
{
    "uuid": "9a0e4135-7f6c-4a47-b3af-949d55289b68"
}
```

Exit status

When curl exits, it returns a numeric value to the shell. For success, it's 0, and for errors, there are about 100 different values.

For example, here is an exit status 7 (failed to connect to host):

```
curl http://httpbingo.org:1313/get
echo "exit status = $?"
```

```
exit status = 7
```

You can access the exit status through the `$?` shell variable.

Final thoughts

That's it! We've covered a lot of ground, from making all kinds of HTTP requests, to controlling transfers, handling URLs, and configuring curl.

Hopefully your curl-fu has gotten a little better. There are two great resources if you want to improve it further: man curl (reference documentation) and Everything curl (500-page book).

If you find this book useful — please spread the word and subscribe to my other projects at antonz.org/subscribe.

Here are some final words of wisdom for you:

```
curl http://httpbingo.org/status/418
```

```
I'm a teapot!
```