cURL Command Tutorial with Examples

by Anirban Das - Edited by Ioannis Karydis

cURL is a command line tool and a library which can be used to receive and send data between a client and a server or any two machines connected over the internet. It supports a wide range of protocols like HTTP, FTP, IMAP, LDAP, POP3, SMTP and many more.

Due to its versatile nature, cURL is used in many applications and for many use cases. For example, the command line tool can be used to download files, testing APIs and debugging network problems. In this article, we shall look at how you can use the cURL command line tool to perform various tasks.

Contents

Install cURL

Linux

Most Linux distributions have cURL installed by default. To check whether it is installed on your system or not, type `curl` in your terminal window and press enter. If it isn't installed, it will show a "command not found" error. Use the commands below to install it on your system.

For Ubuntu/Debian based systems use:

```
sudo apt update
sudo apt install curl
```

For CentOS/RHEL systems, use:

```
sudo yum install curl
```

On the other hand, for Fedora systems, you can use the command:

```
sudo dnf install curl
```

MacOS

MacOS comes with cURL preinstalled, and it receives updates whenever Apple releases updates for the OS. However, in case you want to install the most recent version of cURL, you can install the `curl` Homebrew package. Once you install Homebrew, you can install it with:

```
brew install curl
```

Windows

For Windows 10 version 1803 and above, cURL now ships by default in the Command Prompt, so you can use it directly from there. For older versions of Windows, the cURL project has Windows binaries. Once you download the ZIP file and extract it, you will find a folder named `curl-<version number>-mingw`. Move this folder into a directory of your choice. In this article, we will assume our folder is named `curl-7.62.0-win64-mingw`, and we have moved it under `C:\`.

Next, you should add cURL's `bin` directory to the Windows `PATH` environment variable, so that Windows can find it when you type `curl` in the command prompt. For this to work, you need to follow these steps:

- Open the "Advanced System Properties" dialog by running `systempropertiesadvanced` from the Windows Run dialog (Windows key + R).
- Click on the "Environment Variables" button.
- Double-click on "Path" from the "System variables" section, and add the path `C:\curl-7.62.0-win64-mingw\bin`. For Windows 10, you can do this with the "New" button on the right. On older versions of Windows, you can type in `;C:\curl-7.62.0-win64-mingw\bin` (notice the semicolon at the beginning) at the end of the "Value" text box.

Once you complete the above steps, you can type `curl` to check if this is working. If everything went well, you should see the following output:

```
C:\Users\Administrator>curl
curl: try 'curl --help' or 'curl --manual' for more information
```

cURL basic usage

The basic syntax of using cURL is simply:

```
curl <url>
```

This fetches the content available at the given URL, and prints it onto the terminal. For example, if you run `curl example.com`, you should be able to see the HTML page printed, as shown below:

```
~ $ curl example.com
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif
;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 50px;
        background-color: #fff;
        border-radius: 1em;
    }
```

This is the most basic operation cURL can perform. In the next few sections, we will look into the various command line options accepted by cURL.

Downloading Files with cURL

As we saw, cURL directly downloads the URL content and prints it to the terminal. However, if you want to save the output as a file, you can specify a filename with the `-o` option, like so:

```
curl -o vlc.dmg http://ftp.belnet.be/mirror/videolan/vlc/3.0.4/macosx/vlc-3.0.4.dmg
```

In addition to saving the contents, cURL switches to displaying a nice progress bar with download statistics, such as the speed and the time taken:

```
~ $ curl http://ftp.belnet.be/mirror/videolan/vlc/3.0.4/macosx/vlc-3.0.4.dmg -o
vlc.dmg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 45.9M  100 45.9M    0     0  10.0M      0  0:00:04  0:00:04 --:--:-- 11.3M
```

Instead of providing a file name manually, you can let cURL figure out the filename with the `-O` option. So, if you want to save the above URL to the file `vlc-3.0.4.dmg`, you can simply use:

```
curl -O http://ftp.belnet.be/mirror/videolan/vlc/3.0.4/macosx/vlc-3.0.4.dmg
```

Bear in mind that when you use the `-o` or the `-O` options and a file of the same name exists, cURL will overwrite it.

If you have a partially downloaded file, you can resume the file download with the `-C -` option, as shown below:

```
curl -O -C - http://ftp.belnet.be/mirror/videolan/vlc/3.0.4/macosx/vlc-3.0.4.dmg
```

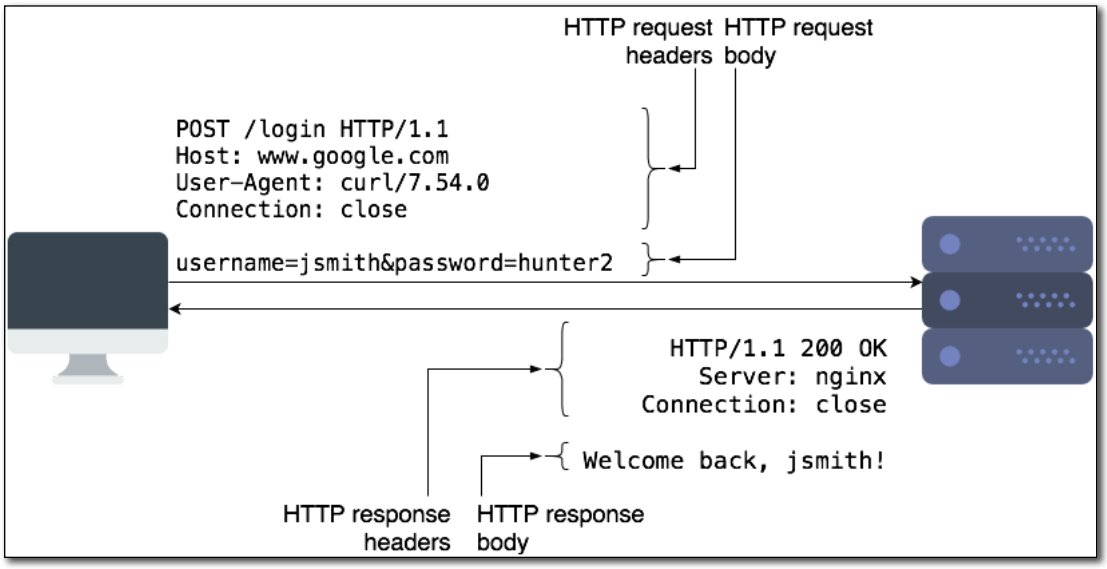
```
~ $ curl -O -C - http://ftp.belnet.be/mirror/videolan/vlc/3.0.4/macosx/vlc-3.0.4
.dmg
** Resuming transfer from byte position 3784704
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  3 42.3M    3 1380k    0     0   249k      0  0:02:53  0:00:05  0:02:48  281k
```

Like most other command line tools, you can combine different options together. For example, in the above command, you could combine `-O -C -` and write it as `-OC -`.

Anatomy of a HTTP request/response

Before we dig deeper into the features supported by cURL, we will discuss a little bit about HTTP requests and responses. If you are familiar with these concepts, you directly skip to the other sections.

To request a resource such as a webpage, or to submit some data to a server, a HTTP client (such as a browser or cURL) makes a HTTP request to the server The server responds back with a HTTP response, which contains the "contents" of that page.



HTTP requests contain the request method, URL, some headers, and some optional data as part of the "request body". The request method controls how a certain request should be processed. The most common types of request methods are "GET" and "POST". Typically, we use "GET" requests to retrieve a resource from the server, and "POST" to submit data to the server for processing. "POST" requests typically contain some data in the request body, which the server can use.

HTTP responses are similar and contain the status code, some headers, and a body. The body contains the actual data that clients can display or save to a file. The status code is a 3-digit code which tells the client if the request succeeded or failed, and how it should proceed further. Common status codes are 2xx (success), 3xx (redirect to another page), and 4xx/5xx (for errors).

HTTP is an "application layer protocol", and it runs over another protocol called TCP. It takes care of retransmitting any lost data, and ensures that the client and server transmit data at an optimal rate. When you use HTTPS, another protocol called SSL/TLS runs between TCP and HTTP to secure the data.

Most often, we use domain names such as `google.com` to access websites. Mapping the domain name to an IP address occurs through another protocol called DNS.

You should now have enough background to understand the rest of this article.

Following redirects with cURL

By default, when cURL receives a redirect after making a request, it doesn't automatically make a request to the new URL. As an example of this, consider the URL `http://www.facebook.com`. When you make a request using to this URL, the server sends a HTTP 3XX redirect to `https://www.facebook.com/`. However, the response body is otherwise empty. So, if you try this out, you will get an empty output:

```
~ $ curl http://www.facebook.com/
~ $
```

If you want cURL to follow these redirects, you should use the `-L` option. If you repeat make a request for `http://www.facebook.com/` with the `-L` flag, like so:

```
curl -L http://www.facebook.com/
```

Now, you will be able to see the HTML content of the page, similar to the screenshot below. In the next section, we will see how we can verify that there is a HTTP 3XX redirect.

```
~ $ curl -L http://www.facebook.com/
<!DOCTYPE html>
<html lang="kn" id="facebook" class="no_js">
<head><meta charset="utf-8" /><meta name="referrer" content="default" id="meta_r
eferrer" /><script>window._cstart=+new Date();</script><script>function envFlush
(a){function b(b){for(var c in a)b[c]=a[c]}window.requireLazy?window.requireLazy
(["Env"],b):(window.Env=window.Env||{},b(window.Env))}envFlush({"ajaxpipe_token"
:"AXiGVMq4Kc21CV_g","timeslice_heartbeat_config":{"pollIntervalMs":33,"idleGapTh
resholdMs":60,"ignoredTimesliceNames":{"requestAnimationFrame":true,"Event liste
nHandler mousemove":true,"Event listenHandler mouseover":true,"Event listenHandl
er mouseout":true,"Event listenHandler scroll":true},"isHeartbeatEnabled":true,"
isArtilleryOn":false},"shouldLogCounters":true,"timeslice_categories":{"react_re
nder":true,"reflow":true},"sample_continuation_stacktraces":true,"dom_mutation_f
```

Please bear in mind that cURL can only follow redirects if the server replied with a "HTTP redirect", which means that the server used a 3XX status code, and it used the "Location" header to indicate the new URL. cURL cannot process Javascript or HTML-based redirection methods, or the "Refresh header".

If there is a chain of redirects, the `-L` option will only follow the redirects up to 500 times. You can control the number of maximum redirects that it will follow with the `--max-redirs` flag.

```
curl -L --max-redirs 700 example.com
```

If you set this flag to `-1`, it will follow the redirects endlessly.

```
curl -L --max-redirs -1 example.com
```

Viewing response headers with cURL

When debugging issues with a website, you may want to view the HTTP response headers sent by the server. To enable this feature, you can use the `-i` option.

Let us continue with our previous example, and confirm that there is indeed a HTTP 3XX redirect when you make a HTTP request to `http://www.facebook.com/`, by running:

```
curl -L -i http://www.facebook.com/
```

Notice that we have also used `-L` so that cURL can follow redirects. It is also possible to combine these two options and write them as `-iL` or `-Li` instead of `-L -i`.

Once you run the command, you will be able to see the HTTP 3XX redirect, as well as the page HTTP 200 OK response after following the redirect:

```
~ $ curl -Li http://www.facebook.com/
HTTP/1.1 302 Found
Location: https://www.facebook.com/
Content-Type: text/html; charset="utf-8"
X-FB-Debug: eWOW1Dpquwgm/zj4YT2Cht4rrhEGSm1OydUHjczvaxr1g6atZQU4RxUwoI0jh8sjLhog
h51o8txIDadyNqkYMA==
Date: Sat, 15 Dec 2018 11:31:30 GMT
Content-Length: 0
Connection: keep-alive

HTTP/2 200
cache-control: private, no-cache, no-store, must-revalidate
pragma: no-cache
strict-transport-security: max-age=15552000; preload
vary: Accept-Encoding
x-content-type-options: nosniff
x-frame-options: DENY
x-xss-protection: 0
expires: Sat, 01 Jan 2000 00:00:00 GMT
set-cookie: fr=1HG7G888ZSheZu24U..BcFOYT.zh.AAA.0.0.BcFOYT.AWUsNnOa; expires=Fri
, 15-Mar-2019 11:31:31 GMT; Max-Age=7776000; path=/; domain=.facebook.com; secur
e; httponly
set-cookie: sb=E-YUXKQlJqPp-J4U8bn7eWsG; expires=Mon, 14-Dec-2020 11:31:31 GMT;
Max-Age=63072000; path=/; domain=.facebook.com; secure; httponly
content-type: text/html; charset="utf-8"
x-fb-debug: 0YE5YXuBd7SZ6xnwcS8l3lw7tVcpV0vGMvROvnlESbLEiFHMWIQuJ7P3FTkQtLirLpbn
l2bGRdSAhQdqZi3l9Q==
date: Sat, 15 Dec 2018 11:31:31 GMT

<!DOCTYPE html>
<html lang="kn" id="facebook" class="no_js">
<head><meta charset="utf-8" /><meta name="referrer" content="default" id="meta_r
```

If you use the `-o`/`-O` option in combination with `-i`, the response headers and body will be saved into a single file.

Viewing request headers and connection details

In the previous section, we have seen how you can view HTTP response headers using cURL. However, sometimes you may want to view more details about a request, such as the request headers sent and the connection process. cURL offers the `-v` flag (called "verbose mode") for this purpose, and it can be used as follows:

```
curl -v https://www.booleanworld.com/
```

The output contains request data (marked with `>`), response headers (marked with `<`) and other details about the request, such as the IP used and the SSL handshake process (marked with `*`). The response body is also available below this information. (However, this is not visible in the screenshot below).

```
~ $ curl -v https://www.booleanworld.com/
*   Trying 2606:4700:30::6818:60a7...
* TCP_NODELAY set
* Connected to www.booleanworld.com (2606:4700:30::6818:60a7) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/cert.pem
  CApath: none
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-ECDSA-CHACHA20-POLY1305
* ALPN, server accepted to use h2
* Server certificate:
*  subject: OU=Domain Control Validated; OU=PositiveSSL Multi-Domain; CN=sni200397.cloudflaressl.com
*  start date: Nov 28 00:00:00 2018 GMT
*  expire date: Jun  6 23:59:59 2019 GMT
*  subjectAltName: host "www.booleanworld.com" matched cert's "*.booleanworld.com"
*  issuer: C=GB; ST=Greater Manchester; L=Salford; O=COMODO CA Limited; CN=COMODO ECC Domain Validation Secure Server CA 2
*  SSL certificate verify ok.
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Using Stream ID: 1 (easy handle 0x7f9eda806600)
> GET / HTTP/2
> Host: www.booleanworld.com
> User-Agent: curl/7.54.0
> Accept: */*
> Referer:
>
* Connection state changed (MAX_CONCURRENT_STREAMS updated)!
< HTTP/2 200
< date: Sat, 15 Dec 2018 17:26:53 GMT
< content-type: text/html; charset=UTF-8
```

Most often, we aren't interested in the response body. You can simply hide it by "saving" the output to the null device, which is `/dev/null` on Linux and MacOS and `NUL` on Windows:

```
curl -vo /dev/null https://www.booleanworld.com/ # Linux/MacOS
curl -vo NUL https://www.booleanworld.com/ # Windows
```

Silencing errors

Previously, we have seen that cURL displays a progress bar when you save the output to a file. Unfortunately, the progress bar might not be useful in all circumstances. As an example, if you hide the output with `-vo /dev/null`, a progress bar appears which is not at all useful.

You can hide all these extra outputs by using the `-s` header. If we continue with our previous example but hide the progress bar, then the commands would be:

```
curl -svo /dev/null https://www.booleanworld.com/ # Linux/MacOS
curl -svo NUL https://www.booleanworld.com/ # Windows
```

The `-s` option is a bit aggressive, though, since it even hides error messages. For your use case, if you want to hide the progress bar, but still view any errors, you can combine the `-S` option.

So, if you are trying to save cURL output to a file but simply want to hide the progress bar, you can use:

```
curl -sSvo file.html https://www.booleanworld.com/
```

Setting HTTP request headers with cURL

When testing APIs, you may need to set custom headers on the HTTP request. cURL has the `-H` option which you can use for this purpose. If you want to send the custom header `X-My-Custom-Header` with the value of `123` to `https://httpbin.org/get`, you should run:

```
curl -H 'X-My-Custom-Header: 123' https://httpbin.org/get
```

(httpbin.org is a very useful website that allows you to view details of the HTTP request that you sent to it.)

The data returned by the URL shows that this header was indeed set:

```
~ $ curl -H 'X-My-Custom-Header: 123' https://httpbin.org/get
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Connection": "close",
    "Host": "httpbin.org",
    "Referer": "",
    "User-Agent": "curl/7.54.0",
    "X-My-Custom-Header": "123"
  },
  "origin": "[redacted]",
  "url": "https://httpbin.org/get"
}
```

You can also override any default headers sent by cURL such as the "User-Agent" or "Host" headers. The HTTP client (in our case, cURL) sends the "User-Agent" header to tell the server about the type and version of the client used. Also, the client uses the "Host" header to tell the server about the site it should serve. This header is needed because a web server can host multiple websites at a single IP address.

Also, if you want to set multiple headers, you can simply repeat the `-H` option as required.

```
curl -H 'User-Agent: Mozilla/5.0' -H 'Host: www.google.com' ...
```

However, cURL does have certain shortcuts for frequently used flags. You can set the "User-Agent" header with the `-A` option:

```
curl -A Mozilla/5.0 http://httpbin.org/get
```

The "Referer" header is used to tell the server the location from which they were referred to by the previous site. It is typically sent by browsers when requesting Javascript or images linked to a page, or when following redirects. If you want to set a "Referer" header, you can use the `-e` flag:

```
curl -e http://www.google.com/ http://httpbin.org/get
```

Otherwise, if you are following a set of redirects, you can simply use `-e ';auto'` and cURL will take care of setting the redirects by itself.

Making POST requests with cURL

By default, cURL sends GET requests, but you can also use it to send POST requests with the `-d` or `--data` option. All the fields must be given as `key=value` pairs separated by the ampersand (`&`) character. As an example, you can make a POST request to httpbin.org with some parameters:

```
curl --data "firstname=boolean&lastname=world" https://httpbin.org/post
```

From the output, you can easily tell that we posted two parameters (this appears under the "form" key):

```
~ $ curl --data "firstname=boolean&lastname=world" https://httpbin.org/post
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "firstname": "boolean",
    "lastname": "world"
  },
  "headers": {
    "Accept": "*/*",
    "Connection": "close",
    "Content-Length": "32",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.54.0"
  },
  "json": null,
  "origin": "[redacted]",
  "url": "https://httpbin.org/post"
}
```

Any special characters such as `@`, `%`, `=` or spaces in the value should be URL-encoded manually. So, if you wanted to submit a parameter "email" with the value "test@example.com", you would use:

```
curl --data "email=test%40example.com" https://httpbin.org/post
```

Alternatively, you can just use `--data-urlencode` to handle this for you. If you wanted to submit two parameters, `email` and `name`, this is how you should use the option:

```
curl --data-urlencode "email=test@example.com" --data-urlencode "name=Boolean World" https://httpbin.org/post
```

If the `--data` parameter is too big to type on the terminal, you can save it to a file and then submit it using `@`, like so:

```
curl --data @params.txt example.com
```

So far, we have seen how you can make POST requests using cURL. If you want to upload files using a POST request, you can use the `-F` ("form") option. Here, we will submit the file `test.c`, under the parameter name `file`:

```
curl -F file=@test.c https://httpbin.org/post
```

This shows the content of the file, showing that it was submitted successfully:

```
~ $ curl -F file=@test.c https://httpbin.org/post
{
  "args": {},
  "data": "",
  "files": {
    "file": "int main() {\n\tprintf(\"Hello world!\\n\");\n\treturn 0;\n}"
  },
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Connection": "close",
    "Content-Length": "250",
    "Content-Type": "multipart/form-data; boundary=------------------------69a45
7f9a3be53e2",
    "Expect": "100-continue",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.54.0"
  },
  "json": null,
  "origin": "[redacted]",
  "url": "https://httpbin.org/post"
}
```

Submitting JSON data with cURL

In the previous section, we have seen how can submit POST requests using cURL. You can also submit JSON data using the `--data` option. However, most servers expect to receive a POST request with key-value pairs, similar to the ones we have discussed previously. So, you need to add an additional header called 'Content-Type: application/json' so that the server understands it's dealing with JSON data and handles it appropriately. Also, you don't need to URL-encode data when submitting JSON.

So if you had the following JSON data and want to make a POST request to https://httpbin.org/post:

```
{
  "email": "test@example.com",
  "name": ["Boolean", "World"]
}
```

Then, you can submit the data with:

```
curl --data '{"email":"test@example.com", "name": ["Boolean", "World"]}' -H 'Content-Type: application/json' ht
```

In this case, you can see the data appear under the `json` value in the httpbin.org output:

```
~ $ curl --data '{"email":"test@example.com", "name": ["Boolean", "World"]}' -H
'Content-Type: application/json' https://httpbin.org/post
{
  "args": {},
  "data": "{\"email\":\"test@example.com\", \"name\": [\"Boolean\", \"World\"]}"
,
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Connection": "close",
    "Content-Length": "58",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.54.0"
  },
  "json": {
    "email": "test@example.com",
    "name": [
      "Boolean",
      "World"
    ]
  },
  "origin": "[redacted]",
  "url": "https://httpbin.org/post"
}
```

You can also save the JSON file, and submit it in the same way as we did previously:

```
curl --data @data.json https://httpbin.org/post
```

Changing the request method

Previously, we have seen how you can send POST requests with cURL. Sometimes, you may need to send a POST request with no data at all. In that case, you can simply change the request method to POST with the `-X` option, like so:

```
curl -X POST https://httpbin.org/post
```

You can also change the request method to anything else, such as PUT, DELETE or PATCH. One notable exception is the HEAD method, which cannot be set with the `-X` option. The HEAD method is used to check if a document is present on the server, but without downloading the document. To use the HEAD method, use the

`-I` option:

```
curl -I https://www.booleanworld.com/
```

When you make a HEAD request, cURL displays all the request headers by default. Servers do not send any content when they receive a HEAD request, so there is nothing after the headers:

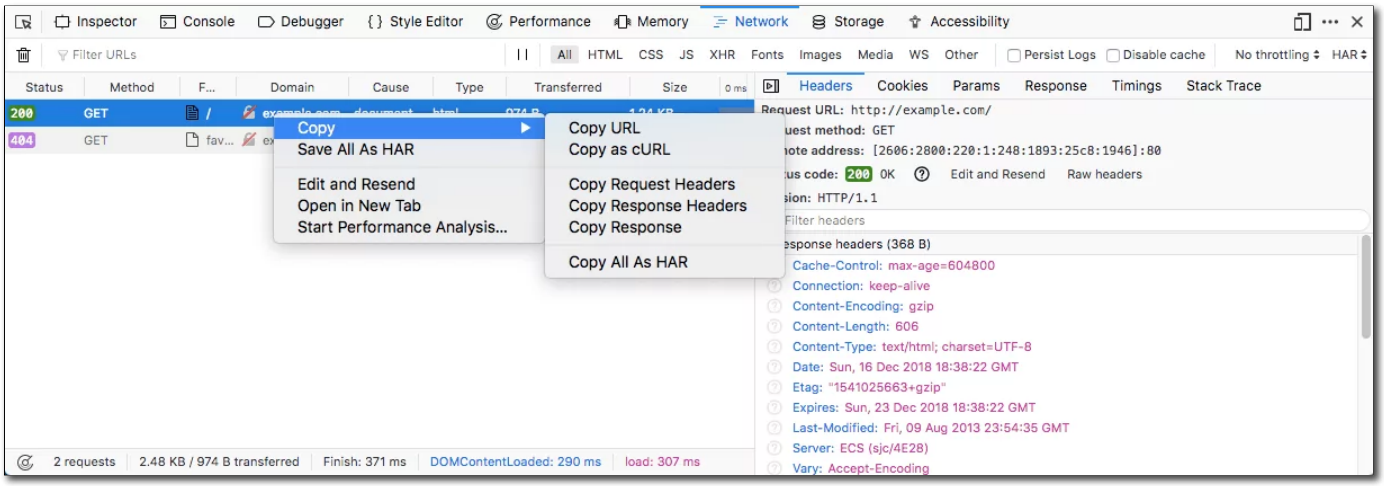
```
~ $ curl -I https://www.booleanworld.com/
HTTP/2 200
date: Sun, 16 Dec 2018 10:27:46 GMT
content-type: text/html; charset=UTF-8
set-cookie: __cfduid=d444b7b0d4663cae187aedb7c0ec60db51544956065; expires=Mon, 1
6-Dec-19 10:27:45 GMT; path=/; domain=.booleanworld.com; HttpOnly; Secure
vary: Accept-Encoding, Cookie
cache-control: max-age=3, must-revalidate
last-modified: Sun, 16 Dec 2018 10:19:58 GMT
x-frame-options: sameorigin
x-xss-protection: 1; mode=block
strict-transport-security: max-age=15552000; includeSubDomains; preload
x-content-type-options: nosniff
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi
/beacon/expect-ct"
server: cloudflare
cf-ray: 48a075927c977014-SIN
```

Replicating browser requests with cURL

If you want to replicate a request made through your browser through cURL, you can use the Chrome, Firefox and Safari developer tools to get a cURL command to do so.

The steps involved are the same for all platforms and browsers:

- Open developer tools in Firefox/Chrome (typically F12 on Windows/Linux and Cmd+Shift+I on a Mac)
- Go to the network tab
- Select the request from the list, right click it and select “Copy as cURL”



The copied command contains all the headers, request methods, cookies etc. needed to replicate the exact same request. You can paste the command in your terminal to run it.

Making cURL fail on HTTP errors

Interestingly, cURL doesn’t differentiate between a successful HTTP request (2xx) and a failed HTTP request (4xx/5xx). So, it always returns an exit status of 0 as long as there was no problem connecting to the site. This makes it difficult to write shell scripts because there is no way to check if the file could be downloaded successfully.

You can check this by making a request manually:

```
curl https://www.booleanworld.com/404 -sSo file.txt
```

You can see that curl doesn’t print any errors, and the exit status is also zero:

```
~ $ curl https://www.booleanworld.com/404 -sSo file.txt
~ $ echo $?
0
```

If you want to consider these HTTP errors as well, you can use the `-f` option, like so:

```
curl https://www.booleanworld.com/404 -fsSo file.txt
```

Now, you can see that cURL prints an error and also sets the status code to 22 to inform that an error occured:

```
~ $ curl https://www.booleanworld.com/404 -fsSo file.txt
curl: (22) The requested URL returned error: 404
~ $ echo $?
22
```

Making authenticated requests with cURL

Some webpages and APIs require authentication with an username and password. There are two ways to do this. You can mention the username and password with the `-u` option:

```
curl -u boolean:world https://example.com/
```

Alternatively, you can simply add it to the URL itself, with the `<username>:<password>@<host>` syntax, as shown:

```
curl https://boolean:world@example.com/
```

In both of these methods, curl makes a "Basic" authentication with the server.

Testing protocol support with cURL

Due to the wide range of protocols supported by cURL, you can even use it to test protocol support. If you want to check if a site supports a certain version of SSL, you can use the `--sslv<version>` or `--tlsv<version>` flags. For example, if you want to check if a site supports TLS v1.2, you can use:

```
curl -v --tlsv1.2 https://www.booleanworld.com/
```

The request takes place normally, which means that the site supports TLSv1.2. Now, let us check if the site supports SSL v3:

```
curl -v --sslv3 https://www.booleanworld.com/
```

This command throws a `handshake_failed` error, because the server doesn't support this version of SSL.

```
~ $ curl -v --sslv3 https://www.booleanworld.com/
*   Trying 104.24.97.167...
* Connected to www.booleanworld.com (104.24.97.167) port 443 (#0)
* found 148 certificates in /etc/ssl/certs/ca-certificates.crt
* found 592 certificates in /etc/ssl/certs
* ALPN, offering http/1.1
* gnutls_handshake() failed: Handshake failed
* Closing connection 0
curl: (35) gnutls_handshake() failed: Handshake failed
```

Please note that, depending on your system and the library version/configuration, some of these version options may not work. The above output was taken from Ubuntu 16.04's cURL. However, if you try this with cURL in MacOS 10.14, it gives an error:

```
~ $ curl -v --sslv3 https://www.booleanworld.com/
*   Trying 2606:4700:30::6818:60a7...
* TCP_NODELAY set
* Connected to www.booleanworld.com (2606:4700:30::6818:60a7) port 443 (#0)
* LibreSSL was built without SSLv3 support
* Closing connection 0
curl: (4) LibreSSL was built without SSLv3 support
```

You can also test for HTTP protocol versions in the same way, by using the flags `--http1.0`, `--http1.1` or `--http2`.

Setting the Host header and cURL's `--resolve` option

Previously, we have discussed about how a web server chooses to serve different websites to visitors depending upon the "Host" header. This can be very useful to check if your website has virtual hosting configured correctly, by changing the "Host" header. As an example, say you have a local server at `192.168.0.1` with two websites configured, namely `example1.com` and `example2.com`. Now, you can test if everything is configured correctly by setting the Host header and checking if the correct contents are served:

```
curl -H 'Host: example1.com' http://192.168.0.1/
curl -H 'Host: example1.com' http://192.168.0.1/
```

Unfortunately, this doesn't work so well for websites using HTTPS. A single website may be configured to serve multiple websites, with each website using its own SSL/TLS certificate. Since SSL/TLS takes place at a lower level than HTTP, this means clients such as cURL have to tell the server which website we're trying to access at the SSL/TLS level, so that the server can pick the right certificate. By default, cURL always tells this to the server.

However, if you want to send a request to a specific IP like the above example, the server may pick a wrong certificate and that will cause the SSL/TLS verification to fail. The `Host` header only works at the HTTP level and not the SSL/TLS level.

To avoid the problem described above, you can use the `--resolve` flag. The resolve flag will send the request to the port and IP of your choice but will send the website name at both SSL/TLS and HTTP levels correctly.

Let us consider the previous example. If you were using HTTPS and wanted to send it to the local server `192.168.0.1`, you can use:

```
curl https://example1.com/ --resolve example1.com:192.168.0.1:443
```

It also works well for HTTP. Suppose, if your HTTP server was serving on port 8080, you can use either the `--resolve` flag or set the Host header and the port manually, like so:

```
curl http://192.168.0.1:8080/ -H 'Host: example1.com:8080'
curl http://example.com/ --resolve example1.com:192.168.0.1:8080
```

The two commands mentioned above are equivalent.

Resolve domains to IPv4 and IPv6 addresses

Sometimes, you may want to check if a site is reachable over both IPv4 or IPv6. You can force cURL to connect to either the IPv4 or over IPv6 version of your site by using the `-4` or `-6` flags.

Please bear in mind that a website can be reached over IPv4 and IPv6 only if:

- There are appropriate DNS records for the website that links it to IPv4 and IPv6 addresses.
- You have IPv4 and IPv6 connectivity on your system.

For example, if you want to check if you can reach the website icanhazip.com over IPv6, you can use:

```
curl -6 https://icanhazip.com/
```

If the site is reachable over HTTPS, you should get your own IPv6 address in the output. This website returns the public IP address of any client that connects to it. So, depending on the protocol used, it displays an IPv4 or IPv6 address.

You can also use the `-v` option along with `-4` and `-6` to get more details.

Disabling cURL's certificate checks

By default, cURL checks certificates when it connects over HTTPS. However, it is often useful to disable the certificate checking, when you are trying to make requests to sites using self-signed certificates, or if you need to test a site that has a misconfigured certificate.

To disable certificate checks, use the `-k` certificate. We will test this by making a request to expired.badssl.com, which is a website using an expired SSL certificate.

```
curl -k https://expired.badssl.com/
```

With the `-k` option, the certificate checks are ignored. So, cURL downloads the page and displays the request body successfully. On the other hand, if you didn't use the `-k` option, you will get an error, similar to the one below:

```
~ $ curl https://expired.badssl.com/
curl: (60) SSL certificate problem: certificate has expired
More details here: https://curl.haxx.se/docs/sslcerts.html

curl performs SSL certificate verification by default, using a "bundle"
 of Certificate Authority (CA) public keys (CA certs). If the default
 bundle file isn't adequate, you can specify an alternate file
 using the --cacert option.
If this HTTPS server uses a certificate signed by a CA represented in
 the bundle, the certificate verification probably failed due to a
 problem with the certificate (it might be expired, or the name might
 not match the domain name in the URL).
If you'd like to turn off curl's verification of the certificate, use
 the -k (or --insecure) option.
HTTPS-proxy has similar options --proxy-cacert and --proxy-insecure.
```

Troubleshooting website issues with "cURL timing breakdown"

You may run into situations where a website is very slow for you, and you would like to dig deeper into the issue. You can make cURL display details of the request, such as the time taken for DNS resolution, establishing a connection etc. with the `-w` option. This is often called as a cURL "timing breakdown".

As an example, if you want to see these details for connecting to `https://www.booleanworld.com/`, run:

```
curl https://www.booleanworld.com/ -sSo /dev/null -w 'namelookup:\t%{time_namelookup}\nconnect:\t%{time_connect
```

(If you are running this from a Windows system, change the `/dev/null` to `NUL`).

You will get some output similar to this:

```
~ $ curl -sSo /dev/null -w 'namelookup:\t%{time_namelookup}\nconnect:\t%{time_co
nnect}\nappconnect:\t%{time_appconnect}\npretransfer:\t%{time_pretransfer}\nredi
rect:\t%{time_redirect}\nstarttransfer:\t%{time_starttransfer}\ntotal:\t\t%{time
_total}\n' https://www.booleanworld.com/
namelookup:     0.004659
connect:        0.036939
appconnect:     0.284737
pretransfer:    0.284957
redirect:       0.000000
starttransfer:  0.843396
total:          0.982222
```

Each of these values is in seconds, and here is what each value represents:

- **namelookup** — The time required for DNS resolution.
- **connect** — The time required to establish the TCP connection.
- **appconnect** — This is the time taken to establish connections for any layers between TCP and the application layer, such as SSL/TLS. In our case, the application layer is HTTP. Also, if there is no such intermediate layer (such as when there is a direct HTTP request), this time will always be 0.
- **pretransfer** — This is the time taken from the start to when the transfer of the file is just about to begin.
- **redirect** — This is the total time taken to process any redirects.
- **starttransfer** — Time it took from the start to when the first byte is about to be transferred.
- **total** — The total time taken for cURL to complete the entire process.

As an example, say, you are facing delays connecting to a website and you notice the "namelookup" value was too high. As this indicates a problem with your ISP's DNS server, you may start looking into why the DNS lookup is so slow, and switch to another DNS server if needed.

cURL configuration files

Sometimes, you may want to make all cURL requests use the same options. Passing these options by hand isn't a feasible solution, so cURL allows you to specify options in a configuration file.

The default configuration file is located in `~/.curlrc` in Linux/MacOS and `%appdata%\_curlrc` in Windows. Inside this file, you can specify any options that you need, such as:

```
# Always use IPv4
-4
# Always show verbose output
-v
# When following a redirect, automatically set the previous URL as referer.
referer = ";auto"
# Wait 60 seconds before timing out.
connect-timeout = 60
```

After creating the above file, try making a request with `curl example.com`. You will find that these options have taken effect.

If you want to use a custom configuration file instead of the default one, then you can use `-K` option to point curl to your configuration file. As an example, if you have a configuration file called `config.txt`, then you can use it with:

```
curl -K config.txt example.com
```

Conclusion

In this article, we have covered the most common uses of the cURL command. Of course, this article only scratches the surface and cURL can do a whole lot of other things. You can type `man curl` in your terminal or just visit this page to see the man page which lists all the options.

If you liked this post, please share it ☺