
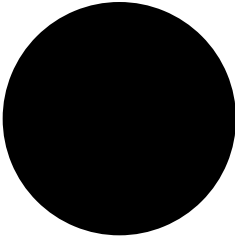
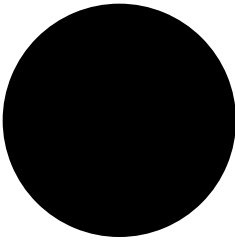
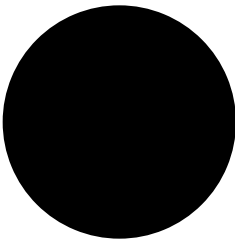


Navigation

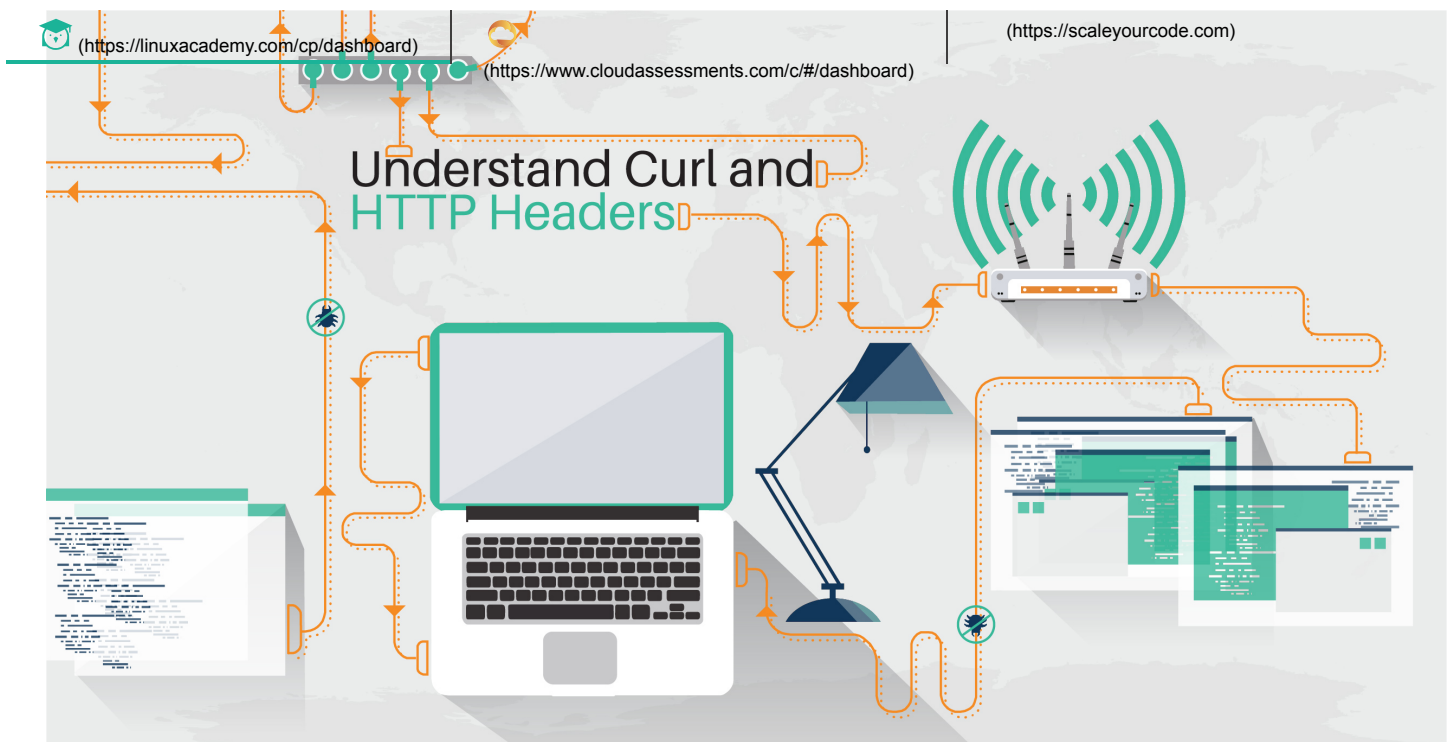
←
Back to community

 Fasih Khatib (/cp/socialize/profile/user/fasihxkhatib)



- Bookmark this post
- Sticky this post
- Follow this user
- Share!
- Spam!

 How To Guide



Understanding CURL and HTTP Headers

1/16/2017

🔒 Instructor Approved

Table of Contents

1. Introduction
2. Requirements
3. Using curl
 1. GET request
 2. HEAD request
 3. Verbose output
 4. Multiple requests
 5. POST request
 6. Setting headers
 7. Posting form data
 8. Uploading files
 9. PUT request
 10. DELETE request
 11. Basic authentication
 12. Setting cookies
4. HTTP headers
 1. Authorization header
 2. Cache-Control header
 3. Expires header
 4. Connection header
 5. Accept header
 6. Cookie header
 7. Content-Length header
 8. Content-Type header
5. Additional Resources

Introduction

curl is a command line tool which is used to transfer data over the internet. It began as a project by Daniel Stenberg to transfer data over HTTP but has now evolved into a very robust tool that transfers data not just over HTTP but also FTP, TELNET, IMAP, and many more. In this guide we will cover the basics of curl and look at how we can use it to retrieve and send data, and also take a deep dive and look at HTTP headers.

Requirements

To follow this guide, you'll need access to a Linux or Mac machine. You'll need some familiarity with using the terminal to execute commands. If you aren't familiar with using the terminal, feel free to take advantage of the resources available at Linux Academy to get up to speed. These resources are listed at the end of this guide.

Using curl

The best way to understand curl is to use it. This guide is an introduction-by-example. We'll use curl to send requests to httpbin.org which is an HTTP client testing service. Let's get started.

GET request



Fire up a terminal and execute the following command:
(<https://linuxacademy.com/cp/dashboard>)



(<https://www.cloudassessments.com/c/#/dashboard>)

(<https://scaleyourcode.com>)

```
me@home:~$ curl www.httpbin.org
```

This command will fetch the HTML for the HTTPBin homepage. Notice that we've not mentioned the protocol or the HTTP method. They've defaulted to HTTP and GET, respectively.

HEAD request

Sometimes you may want to see just the headers that the server would return when it's issued a GET request. In such a case, we'd use the HEAD method. The head request is issued with `-I` (capital i) or `--head`.

```
me@home:~$ curl -I www.httpbin.org
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 01 Dec 2016 18:50:56 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 12150
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Connection: keep-alive
```

Verbose output

Sometimes, you may want to debug a problem because the curl command hasn't returned what it was supposed to or just hasn't worked at all. In such cases, you can use the verbose mode using `-v` or `--verbose` to get more output from curl. Let's modify the previous HTTP request to become verbose.

```
me@home:~$ curl -Iv www.httpbin.org
```

Now there's a lot more output in the terminal. Let's see what it means.

Lines prefixed with an asterisk (*) show additional work curl has done. For example:

```
* Rebuilt URL to: www.httpbin.org/
* Trying 23.22.14.18...
* Connected to www.httpbin.org (23.22.14.18) port 80 (#0)
```

The first line shows that curl added a slash at the end of the URL.

The second line shows that curl has resolved the URL to an IP address.

Finally, the third line shows that curl has connected to the URL on port 80.

Lines prefixed with a greater-than (>) sign show the data curl has sent to the server. For example:

```
> HEAD / HTTP/1.1
> Host: www.httpbin.org
> User-Agent: curl/7.47.0
> Accept: */*
```

Lines prefixed with a less-than (<) sign show the data curl has received from the server. For example:

```
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Server: nginx
Server: nginx
```

Multiple requests

You can also send requests to multiple URLs by separating them by space. Here's how:

```
me@home:~$ curl httpbin.org/status/418 httpbin.org/status/418
```

The above example will send two GET requests, both to the same URL, one after the other. You should see two little teapots printed in your terminal.

POST request

You can send a POST request by using the `-d` (or `--data`) flag. It's up to you to properly URL encode the data or specify it as JSON. We'll see the examples for both of those. Here's how you'd send URL encoded data as a part of the POST body

```
me@home:~$ curl -d 'name=john&course=linux' httpbin.org/post
```

The response sent by HTTPBin will simply echo what you've sent.



To send JSON:
(<https://linuxacademy.com/cp/dashboard>)



(<https://www.cloudassessments.com/c/#/dashboard>)

(<https://scaleyourcode.com>)

```
me@home:~$ curl -d '{"name":"John Doe","course":"Linux"}' httpbin.org/post
```

Notice that in both the examples, the data is enclosed in single quotes. Using double quotes is okay but it may cause problems if the body also contains double quotes like JSON. Using single quotes is the safer approach.

Setting headers

You can set headers by using the `-H` flag. In our previous example, we've sent JSON without setting the Content-Type header to `application/json`. Here's the previous example modified to set the header:

```
me@home:~$ curl -H 'Content-Type: application/json' -d '{"name":"John Doe","course":"Linux"}' httpbin.org/post
```

You set the header as a key-value pair consisting of the header name followed by a colon and the value.

Posting form data

You can send form data by using the `-F` flag. Use this flag for every form field that you want to send. Here's how you'd send form data:

```
me@home:~$ curl -F "name=John Doe" -F "course=linux" httpbin.org/post
```

Notice that we've written the name as John Doe without a plus sign in between. The response received will have the Content-Type header set to `multipart/form-data`.

Uploading files

Using `curl`, you can upload files both as a part of a form field or to a ReST API endpoint. We'll first create a file and then upload it both as a part of a form and to a ReST endpoint.

```
me@home:~$ echo "hello, world" >> file.txt
me@home:~$ curl -F "file=@file.txt" httpbin.org/post
```

Since we're using the `-F` flag, the file is uploaded as a part of the form. You specify the file to upload by an `@` followed by the path to the file. Since our file is in the current directory, we just specify the file name.

Uploading to a ReST endpoint is similar. We just use the `-d` flag instead.

```
me@home:~$ curl -d @file.txt httpbin.org/post
```

Notice now that there are no quotes around `file.txt`. This is because we want the contents of the file to be the body of the POST request and not string `"@file.txt"` itself.

PUT request

ReST APIs use the PUT request to update an existing resource that the server manages. Making a PUT request with `curl` is similar to making a POST request. Here's how you'd make a PUT request:

```
me@home:~$ curl -X PUT -d '{"name":"Jane Doe"}' httpbin.org/put
```

DELETE request

Similarly, ReST APIs use DELETE request to delete an existing resource. Here's how to make a delete request:

```
me@home:~$ curl -X DELETE httpbin.org/delete
```

Basic authentication

`Curl` includes support for basic authentication. With basic authentication, you send your username and password as a part of the URL. There's a two ways in which you can do this in `curl`. Here's how:

```
me@home:~$ curl -u john:abc123 httpbin.org/basic-auth/john/abc123
me@home:~$ curl john:abc123@httpbin.org/basic-auth/john/abc123
```

In the first one you specify the username and password using the `-u` (short for `--user`) flag and `curl` appends this to the URL you provide. In the second case, you do it manually. Basic authentication does not encrypt your credentials in any way and is thus insecure unless used with HTTPS.

Setting cookies

You can set cookies using the `-b` (short for `--cookie`) flag. You specify the key and the value for the cookie with the flag. Here's an example showing how to set two cookies:

```
me@home:~$ curl --cookie "name=john;course=linux beginner" httpbin.org/cookies
```

Notice that cookies are separated by a semicolon and that you do not need to encode spaces as a plus sign. This brings us to the end of the guide on using `curl`. The next part of the guide will introduce you to HTTP headers.

HTTP headers



The HTTP protocol allows a client to send a request to a server. With every HTTP request and response, there are associated headers that provide some more information. Let's make an HTTP request and see what we get: (<https://scaleyourcode.com>)

```
me@home:~$ curl -Iv httpbin.org/get
* Trying 23.22.14.18...
* Connected to httpbin.org (23.22.14.18) port 80 (#0)
> HEAD /get HTTP/1.1
> Host: httpbin.org> User-Agent: curl/7.49.1
> Accept: */*
>
< HTTP/1.1 200 OKHTTP/1.1 200 OK
< Server: nginxServer: nginx
< Date: Wed, 14 Dec 2016 16:23:11 GMTDate: Wed, 14 Dec 2016 16:23:11 GMT
< Content-Type: application/json
Content-Type: application/json
< Content-Length: 186
Content-Length: 186
< Connection: keep-alive
Connection: keep-alive
< Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
Access-Control-Allow-Credentials: true
```

In the output that's printed, the following are headers: Host, User-Agent, Accept, Server, Content-Type, Content-Length, Connection, Access-Control-Allow-Origin, and Access-Control-Allow-Credentials.

Headers can be classified as request and response headers that contain information about the request and response, respectively, or as general headers that contain information about both request and response. There's also another class of headers known as entity headers that contain more information about an entity, such as a file (i.e. it's length and MIME-type). Here's some of the most common headers and what they mean.

Authorization header

The Authorization header is used to provide authentication information such as bearer tokens. The server then uses this information to find out if the request should be processed further or not, depending on the validity of the authentication information provided. Example:

```
Authorization: Bearer 3beca038a248ff027d0445342fe285
```

Cache-Control header

The Cache-Control header decides how long applications, such as the browser, can keep a local copy of the data. This helps improve the efficiency since it helps avoid a round-trip to the server. Cache-Control has various directives that control various aspects of caching, such as the length of time data may be cached or if it shouldn't be cached at all. Example:

```
Cache-Control: max-age=100
```

Expires header

The Expires header is also used for caching and specifies the date and time after which a particular cached resource is considered stale. When both Expires and Cache-Control header are set, the Cache-Control header takes higher priority. Example:

```
Expires: Wed, 21 Oct 2015 07:28:00 GMT
```

Connection header

The Connection header specifies whether a connection should last for the duration of the request or if it should stay open, allowing it to be reused for subsequent requests. The default value is "close" which creates a connection that only lasts for the duration of the request. Setting the header to "keep-alive" allows it to be persistent. Example:

```
Connection: keep-alive
```

Accept header

The Accept header tells the server what kind of content the client is expecting. The value of this header is the MIME type of the content. Example:

```
Accept: application/json
```

Cookie header

The cookie header contains all the cookies and their values. Cookies are used to identify the users, maintain sessions, and so on. Example:



Cookie: name=John;course=Linux
(https://linuxacademy.com/cp/dashboard)



(https://www.cloudassessments.com/c/#/dashboard)

(https://scaleyourcode.com)

Content-Length header

The Content-Length header specifies the length of the content in bytes. Example:

Content-Length: 111020

Content-Type header

The Content-Type header tells the client the MIME-type of the content that it has received. Example:

Content-Type: application/json

There are a lot more headers than that can be covered in a simple guide, which brings us to the end of the guide on curl and HTTP headers. If you are eager to learn more about Linux command line, please check out the following section.

Additional Resources

If you are new to the Linux operating system, take a look at the Linux Essentials course available at Linux Academy. The course will give you a basic understanding of Linux and give you a gentle introduction to the command line.

<https://linuxacademy.com/cp/modules/view/id/38> (<https://linuxacademy.com/cp/modules/view/id/38>)

If you'd like to master the terminal, have a look at Mastering Linux Command Line:

<https://linuxacademy.com/cp/modules/view/id/10> (<https://linuxacademy.com/cp/modules/view/id/10>)

To take your skills to the expert level, take a look at Linux by Example from Novice to Pros:

<https://linuxacademy.com/cp/modules/view/id/19> (<https://linuxacademy.com/cp/modules/view/id/19>)

Linux Academy provides a large library of in-depth online Linux training, and they also give access to many other online courses in topics like AWS (<https://linuxacademy.com/amazon-web-services/courses>), DevOps (<https://linuxacademy.com/devops/courses>), Azure (<https://linuxacademy.com/azure/courses>), OpenStack (<https://linuxacademy.com/openstack/courses>), and Big Data (<https://linuxacademy.com/bigdata/courses>).

18 30 + -

... Show Previous Comments



Mbiru Gatora (/cp/socialize/profile/user/tmel173) 7/26/2017

This is awesome !

0 0 + - !



bobson richard (/cp/socialize/profile/user/bric0937) 8/4/2017

great stuff

0 0 + - !

Reply



