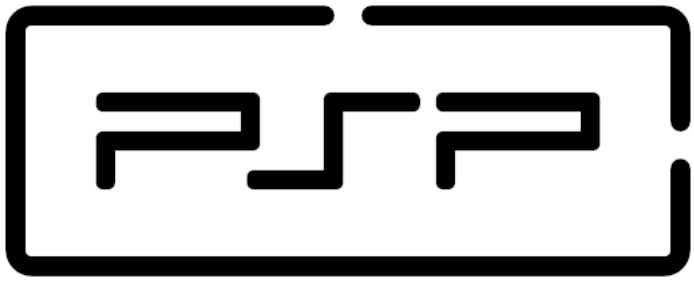

Process and Service Programming

2.4 Annex I - Curl

I.E.S.
Doctor Balmis

PSP class notes by Vicente Martínez is licensed under CC BY-NC-SA 4.0

2.4 Annex I - Curl

- 2.4.1 Get curl
- 2.4.2 Calling a GET method
- 2.4.2 Endpoints and routes
- 2.4.3 HTTP methods and headers
- 2.4.4 Authentication
- 2.4.5 References

Whether it's testing the output of a REST API on development or before deploying it to production, simply fetching a response from a website (for instance, to check it's not down), or getting response times from a site / API Curl is practically omnipresent.

Curl is a command-line tool that allows us to do HTTP requests from shell. This is its main use.

> **TIP**
>
> The tool was about uploading and downloading data specified with a URL. It was a client-side program (the 'c'), a URL client, and would show the data (by default). So 'c' for Client and URL: cURL.
>
> Most of us pronounce "curl" with an initial k sound, just like the English word curl. It rhymes with words like girl.
>
> But it can also be spelled as c-URL which means see-URL, that is also a good definition about what the tool does.

Curl supports protocols that allow "data transfers" in either or both directions. It supports protocols which have a "URI format" and are described in an RFC, as curl works primarily with URLs (URIs really) as the input key that specifies the transfer.

Curl actually supports these protocols:

> DICT, FILE, FTP, FTPS, GOPHER, GOPHERS, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, MQTT, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET, TFTP.

2.4.1 Get curl

curl is totally free, open and available. There are numerous ways to get it and install it for most operating systems and architecture. Some operating systems include curl by default.

You can always download the source from **CURL official site** or find binary packages to download from there.

- Linux (Ubuntu / Debian). curl is installed by default. Anyway, you can add with the APT package manager

  > apt install curl

- Windows 10 comes with the curl tool bundled with the operating system since version 1804

  > download the latest official curl release for Windows from **curl windows binaries** and install that.

- MacOS comes with the curl tool bundled with the operating system since many years. If you want to upgrade to the latest version shipped by the curl project, we recommend installing homebrew (a macOS software package manager)

  > brew install curl

2.4.2 Calling a GET method

In its most basic form, a curl command will look like this:

```sh
$> curl http://www.net.net
<head><title>Document Moved</title></head>
<body><h1>Object Moved</h1>This document may be found <a HREF="http://net.net">here</a></body>
```

The default behavior for curl is to invoke an HTTP GET method on the given URL. This way, the program's output for that command will be the whole HTTP response's body (in this case, HTML which will be written as given on stdout.

Many times we'll wish to direct the response's contents into a file. This is done with the `-o (--output)` argument, like this:

> curl -o output.html www.net.net
> // Equivalent to
> curl www.net.net > output.html

The URL must be in the last place, but optionally, you can specify the URL of the site you wish to call curl on with a `-s (--silent)` argument , allowing you to change the order of your arguments.

curl -s http://www.net.net -o output.html

In the previous example we are not getting the desired resource, because it has been moved or redirected to another URI. Using the `-L (--location)` mode, we can follow redirects and get the destination resource

```sh
$> curl http://www.dataden.tech
Redirecting
$> curl -L http://www.dataden.tech
<html><head><title>Loading...</title></head><body><script type='text/javascript'>window.location.replace('http://www.dataden.
$> curl -L http://www.net.net
<html>
        <head>
                <title>NET.NET [The first domain name on the Internet!]</title>
        </head>
        <body>
                <!-- Begin: Google Analytics -->
                <script>
                  (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
                  (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
                  m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
                  })(window,document,'script','//www.google-analytics.com/analytics.js','ga');
                  ga('create', 'UA-32196-28', 'auto');
                  ga('send', 'pageview');
                </script>
                <!-- End: Google Analytics -->
                <center>
                        <br /><br /><br /><br /><br /><br /><br /><br /><br />
                        <font face="impact, Arial, Helvetica, sans-serif" size="14">
                                NET.NET
                        </font>
                        <br /><br /><br /><br />
```

```
                        <font face="Arial, Helvetica, sans-serif" size="1">
                                <a href="http://who.godaddy.com/whoischeck.aspx?domain=NET.NET" target="_blank">NET.NET</a> i
                                <br />
                                Coming Soon...
                        </font>
                </center>
        </body>
</html>
```

So far we have only get the html page. If we want to see also the headers of our GET request and response headers, we have to user the `-v (--verbose)` option to get full information about the HTTP protocol.

```sh
$> curl -v  http://www.net.net
*   Trying 34.250.90.28:80...
* TCP_NODELAY set
* Connected to net.net (34.250.90.28) port 80 (#0)
> GET / HTTP/1.1
> Host: net.net
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Cache-Control: private
< Content-Type: text/html
< Server: Microsoft-IIS/10.0
< Set-Cookie: ASPSESSIONIDASRSRRAR=IMFFLMBBBIFJNLNDHLOACDAI; path=/
< X-Powered-By: ASP.NET
< Date: Mon, 04 Oct 2021 21:40:49 GMT
< Content-Length: 1080
<
<html>
        <head>
                <title>NET.NET [The first domain name on the Internet!]</title>
        </head>
        ...
```

In the previous output requests header are marked with `>` while response header are marked with `<`.

short and long command line options

Command line options pass on information to curl about how you want it to behave.

Single-letter options are convenient since they are quick to write and use, but as there are a limited number of letters and not all options are available like that. Long option names are therefore provided for those. Also, as a convenience and to allow scripts to become more readable, most short options have longer name aliases.

Short options are preceded by the minus symbol and a single letter immediately following it. They can be used with just that option name. You can then also combine several single-letter options after the minus.

> $> curl -v -L http://example.com $> curl -vL http://example.com

Long options are always written with two dashes and then the name, and you can only write one option name per double-dash.

> $> curl --verbose --location http://example.com

Finally, we can access partially the verbose mode information using the `-i (--include)` or `-I (--head)` to get the complete answer from the server (headers & data) or just the headers, respectively.

```sh
$>  curl -I https://jsonplaceholder.typicode.com/todos/1
HTTP/2 200
date: Mon, 04 Oct 2021 21:57:55 GMT
content-type: application/json; charset=utf-8
content-length: 83
x-powered-by: Express
x-ratelimit-limit: 1000
x-ratelimit-remaining: 999
x-ratelimit-reset: 1631546224
vary: Origin, Accept-Encoding
access-control-allow-credentials: true
cache-control: max-age=43200
pragma: no-cache
expires: -1
x-content-type-options: nosniff
etag: W/"53-hfEnumeNh6YirfjyjaujcOPPT+s"
via: 1.1 vegur
cf-cache-status: HIT
age: 10926
accept-ranges: bytes
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
report-to: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?s=LxJlkSosQdWmBFBOx1fB6zrbjSbU0iStl7jjtlVL27CtOEP
nel: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
server: cloudflare
cf-ray: 6991ab2c1a5037c7-MAD
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400, h3-28=":443"; ma=86400, h3-27=":443"; ma=86400
```

Finally, adding the `-w "%{time_total}\n"` will simply output the total time it took to fetch the response from the given domain.

2.4.2 Endpoints and routes

The term endpoint is focused on the URL that is used to make a request.

For a typical web API, endpoints are URLs, and they are described in the API's documentation so programmers know how to use/consume them. For example, a particular web API may have this endpoint:

> GET https://my-api.com/Library/Books

This would return a list of all books in the library.

A "route" is typically a part of URL endpoint that routes the pages to different components.

> GET https://my-api.com/Library/Books/341

This would access book with id 341 using the Library/Books endpoint

For instance, for **SWAPI (Star Wars API)** the endpoint is `https://swapi.dev/api/`. That's the entry point for all requests.

Thus there are many routes depending on the information we want to access/add/modify/delete.

```java
$> curl https://swapi.dev/api/people/1
$> curl https://swapi.dev/api/planet/3
$> curl https://swapi.dev/api/vehicles
```

2.4.3 HTTP methods and headers

In every HTTP request, there's a method. Sometimes called a verb. The most commonly used ones are GET, POST, HEAD and PUT.

POST is the HTTP method that was invented to send data to a receiving web application, and it is how most common HTML forms on the web works.

When the data is sent by a browser it will send it URL encoded, as a serialized name=value pairs separated with ampersand symbols (&).

You send such data with curl's `-d (--data)` option like this:

> curl -d 'name=admin&shoesize=12' http://example.com/

Curl selects which methods to use on its own depending on what action to ask for. -d will do POST, -I will do HEAD and so on. If you use the `-X (--request)` option you can change the method keyword curl selects.

> curl -X POST -d 'imageSize=big&imageType=jpg' http://example.org/

POSTing with curl's -d option will make it include a default header that looks like `Content-Type: application/x-www-form-urlencoded`. That's what your typical browser will use for a plain POST.

If that header is not good enough for you, you should, of course, replace that and instead provide the correct one. Such as if you POST JSON to a server and want to more accurately tell the server about what the content is:

> curl -X "POST" -d '{"imageSize":"big","imageType":"jpg","scale":"false"}' -H 'Content-Type: application/json' https://example.com

2.4.4 Authentication

Each HTTP request can be made authenticated. If a server or a proxy wants the user to provide proof that they have the correct credentials to access a URL or perform an action, it can send back a HTTP response code that informs the client that it needs to provide a correct HTTP authentication header in the request to be allowed.

To tell curl to do an authenticated HTTP request, you use the `-u (--user)` option to provide user name and password (separated with a colon). Like this:

> curl --user daniel:secret http://example.com/

This will make curl use the default "Basic" HTTP authentication method.

Many applications and services make use of a secret key or an Authorization token provided by the service provider when you create the service.

Trello API Introduction

Azure Translator API Reference

If we want to use the Azure service for translate text, first we need to obtain the secret key and send it with each call to identify the user and get the permission to use the service.

```java
$> curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&to=en&to=it" -H "Ocp-Apim-Subscripti
[{"detectedLanguage":{"language":"ca","score":1.0},"translations":[{"text":"Hello, how are you?","to":"en"},{"text":"Ciao com
```

Sometimes we can get a temporal authorization by getting an Authorization token, that later must be provided to access the service during a short period of time. Once the time expires, another token must be requested. The `Authorization: Bearer <token>` header is used.

```java
$> curl -X POST "https://api.cognitive.microsoft.com/sts/v1.0/issueToken" -H "Ocp-Apim-Subscription-Key: <here goes the secre
eyJhbGciOiJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNobWFjLXNoYTI1NiIsInR5cCI6IkpXVCJ9.eyJyZWdpb24iOiJnbG9iYWwiLCJz
$> curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&to=en&to=it" -H "Authorization: Bear
{"error":{"code":401000,"message":"The request is not authorized because credentials are missing or invalid."}}
$> curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&to=en&to=it" -H "Authorization: Bear
[{"detectedLanguage":{"language":"ca","score":1.0},"translations":[{"text":"Hello, how are you?","to":"en"},{"text":"Ciao com
```

2.4.5 References

Everything curl is a detailed and totally free book available that explains basically everything there is to know about curl.

freecodecamp.org

curl official site