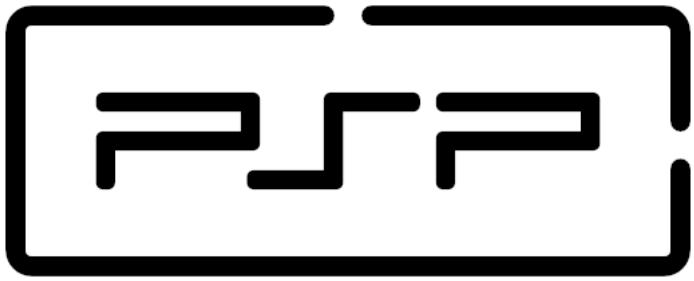

Programación de Servicios y Procesos

2.4 Anexo I - Curl

2.4 Anexo I - Curl

Curl está diseñado para funcionar como una forma de verificar la conectividad a las URL y como una gran herramienta para transferir datos. Tiene especial relevancia e interés cunado se trata de usar servicios basados en API REST, tanto para comprobar su funcionamiento durante la fase de pruebas o bien para cuando el sistema ya está en producción.

Curl es una herramienta de línea de comandos que nos permite hacer peticiones HTTP desde la consola. Su principal uso es obtener una respuesta de un sitio web (pro ejemplo, para saber que no está caído) o para comprobar tiempos de respuesta.

> **curl means ...**
>
> La herramienta fué pensada para subir y descargar información usando una URL. Es una aplicación cliente (de ahí la 'c'), y a su vez es un cliente de URLs. Por lo tanto, 'c' de cliente y URL: cURL.
>
> En inglés "curl" se pronuncia con un sonido inicial /k/, rimando con la pronunciación de la palabra girl.
>
> Por el contrario, si lo deletreamos como c-URL, entonces us significado también adquiere un sentido lógico, ver-URL (see-URL), lo cual también es una buena definición de la utilidad de esta herramienta.

Curl funciona con protocolos que permiten la transferencia de datos en una o dos direcciones. Soporta protocolos basados en una "URI" y que estén descritos en una RFC, por lo que curl funciona principalmente con URLs (URIs en realidad) como el origen y/o destino de las transferencias e intercambios de información que realiza.

Actualmente curl ofrece soporte para los siguiente protocolos:

> 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 Cómo obtener curl

curl es una utilidad libre, abierta y disponible en varios formatos. Podemos descargarla y configurarla en la mayoría de los sistemas operativos y arquitecturas hardware. Algunos SO ya la incluyen en sus distribuciones.

La fuente principal de información y descarga siempre debería ser el **sitio oficial de CURL** donde podemos descargar, entre otros, los instaladores para nuestros sistemas.

- Linux (Ubuntu / Debian). curl viene instalado por defecto. De todas formas, podemos actualizarlo e instalarlo usando el gesto de paquetes APT

  > apt install curl

- Windows 10 viene con la herramienta curl instalada en sus sistemas desde la versión 1804

  > podemos descargar e instalar la última versión oficial de curl para windows desde **curl windows binaries**

- MacOS también trae de serie la herramienta curl desde hace ya bastantes años. Si necesitamos actualizar la versión que tenemos en el sistema, se recomienda instalar homebrew (un gesto de paquetes para macOS)

  > brew install curl

2.4.2 Realizando una petición GET

La sintaxis básica de Curl se ve así:

curl [OPTIONS] [URL] El uso más simple de Curl es mostrar el contenido de una página. El siguiente comando mostrará la página de inicio de testdomain.com.

> curl testdomain.com

Esto generará el código fuente completo de la página de inicio del dominio. Si no se especifica ningún protocolo, curl lo interpretará a HTTP.

```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>
```

Si no se indica lo contrario, curl realiza una solicitud HTTP con el método GET a la URL indicada. La salida del programa para el comando enviado será el cuerpo de la respuesta HTTP, en el caos anterior, el código HTML de la URL solicitada.

Si en vez de mostrar la salida recibida por pantalla nos interesa guardar la información en un archivo, podemos usar el parámetro `-o (--output)` :

> curl -o output.html www.net.net
> // Es equivalente a hacer esto en el SO
> curl www.net.net > output.html

Como hemos visto en la sintaxis, por norma general la URL debe ponerse al final del comando. Opcionalmente podemos especificar la URL en cualquier lugar del comando si usamos el modificador `-s (--silent)`, pudiendo así alterar el orden de los argumentos de curl.

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

En los ejemplos anteriores, el resultado que estamos obteniendo no es el deseado ya que el recurso se ha cambiado de ubicación o bien el sitio nos está redirigiendo a otra URI. Si usamos el parámetro `-L (--location)` podemos hacer que curl siga las redirecciones y obtenga el contenido final que buscamos.

```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 /><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>
```

De momento sólo hemos obtenido el HTML del recurso solicitado. Si queremos ver las cabeceras HTTP de nuestra solicitud y de la respuesta recibida podemos usar el parámetro `-v (--verbose)` que mostrará toda la información que intercambia el protocolo HTTP..

```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>
        ...
```

En el ejemplo anterior podemos ver las cabeceras de nuestra solicitud (REQUEST) precedidas del caracter `>` mientras que las cabeceras de la respuesta (RESPONSE) se muestran precedidas dek caracter `<`.

formato corto y largo de los parámetros

> Como ya hemos visto, los parámetros sirven para modificar el comportamiento por defecto de curl en función de las necesidades.
>
> Los parámetros de una sola letra son rápidos de usar y recordar, pero tenemos un número limitado de parámetros de este tipo por lo que no podemos tener uno para cada opción. Los parámetros que usan palabras están para solucionar la falta de opciones con formato corto. Por otro lado, su uso hace que los comandos sean más legibles, por este motivo la mayoría de parámetros cortos tienen su equivalente largo.
>
> En el formato corto, los parámetros están formados por un guión seguido de la letra correspondiente a la opción. Se puede usar un guión para cada opción, o incluir varias opciones detrás de un único guión.
>
> > $> curl -v -L http://example.com $> curl -vL http://example.com
>
> En el formato largo, los parámetros se preceden de dos guiones y la palabra o palabras que forman la opción. Tras cada doble guión sólo se puede indicar una opción.
>
> > $> curl --verbose --location http://example.com

Por último, aunque ya hemos visto como obtener la información de las cabeceras, podemos visualizar la información de la respuesta de forma completa usando la opción `-i (--include)` or obtener sólo las cabeceras de la respuesta con el uso de la opción `-I (--head)`. Esto sólo afecta al información de la respuesta (HTTP RESPONSE) y en el primer caso veremos la respuesta completa (headers & data) o sólo los headers con la segunda opción.

```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
```

Para acabar, y como curiosidad, añadiendo la opción `-w "%{time_total}\n"` podremos ver el tiempo total que se ha tardado en recibir la respuesta del servidor.

2.4.2 Puntos finales y rutas

El término técnico `endpoint` hace referencia la la URL que se utiliza URL para hacer una petición. Para un mismo servicio, esta URL suele ser siempre la misma y es una de las características de las API REST.

Para un API, los *puntos finales* son URLs y se describen en la documentación de la APU de forma que los programadores sepan cómo usarla y acceder a los recursos y servicios que proporciona.

En el siguiente ejemplo tenemos una API con el siguiente endpoint.

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

Esto devolvería una lista completa de los libros que hay en la biblioteca.

Una ruta ("route") no es más que la parte de la URL que completa al endpoint y que se utiliza para seleccionar y/o acceder a diferentes componentes del servicio o API.

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

El ejemplo anterior devolvería el libro con identificador 341 de los que hay en la biblioteca.

Como ejemplo en SWAPI (Star Wars API) el endpoint es `https://swapi.dev/api/`. Este es el punto de entrada para todas las peticiones.

Además, hay muchas rutas dependiendo de la información que queramos recuperar/añadir/modificar/borrar..

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

2.4.3 Métodos HTTP y cabeceras

Todas las peticiones HTTP tienen un método asociado, también llamado verbo. Por defecto ya hemos visto que se usa GET, pero también tenemos POST, HEAD y PUT que se utilizan bastante.

POST es el método HTTP que se ideó para enviar y recibir información de una aplicación web y es el que utilizan, entre otros, los formularios web.

Cuando los datos de un formulario se envían desde un navegador se mandan `URL encoded`, como una serie de pares nombre=valor separados con símbolos ampersand (&).

Para enviar datos con curl lo indicamos con la opción `-d (--data)` con el siguiente formato:

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

Curl es capaz de seleccionar el método HTTP que debe utilizar en función de las opciones que recibe. Si utilizamos `-d` curl hará un POST, con `-I` hará un HEAD, etc. Con la opción `-X (--request)` podemos indicar qué método queremos que use curl.

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

Si usamos la opción POSTing with curl's `-d` ya hemos visto que por defecto se hace POST. Además de eso, se incluye una cabecera para indicar el formato de los datos enviados `Content-Type: application/x-www-form-urlencoded`. Esto es lo que hace un navegador cuando enviamos un formulario.

Puede ocurrir que los datos que estamos enviando no tengan ese formato, bien porque estemos subiendo algún archivo, enviando información binaria, usando JSON, XML, ... en estos casos podemos, a través de la misma cabecera Content-Type, indicar el formato de la información que estamos enviando.

Por ejemplo si queremos enviar información en formato jSON:

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

2.4.4 Authentication

Cada solicitud HTTP puede ser autenticada. Un servidor o proxy puede necesitar que el usuario confirme su identidad o que tiene los permisos necesarios para acceder a una URL para realizar una acción. En ese caso se le enviará al cliente una respuesta en la que se le indique que debe proporcionar información de autenticación en la cabecera HTTP para que se le permita completar esa solicitud.

En curl, la forma de realizar una petición HTTP autenticadaes con la opción `-u (--user)` y proporcionando un usuario y contraseña (separados por `:`).

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

Esto hará que curl utilice el método de autenticación HTTP por defecto, denominado "Basic"

Otras aplicaciones hacen uso de una clave secreta `secret key` o testigos de autorización `Authorization token`. Esta información nos la proporciona del servicio cuando lo creamos y configuramos.

Trello API Introduction

Azure Translator API Reference

Por ejemplo, si queremos usar el servicio `translate` de Azure tendremos que obtener una *secret key* y utilizarla en la cabecera de cada solicitud que realicemos

```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
```

Otras veces, para no enviar siempre la secret key, solicitamos una autorización temporal, obteniendo un `Authorization token` que podemos usar para acceder al servicio durante un breve período de tiempo. Una vez que el token expire, debemos solicitar otro. Para el envío de tokens se utiliza la cabecera `Authorization: Bearer <token>`.

```java
$> curl -X POST "https://api.cognitive.microsoft.com/sts/v1.0/issueToken"
        -H "Ocp-Apim-Subscription-Key: <here goes the secret key>" -d {}
eyJhbGciOiJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNobWFjLXNoYTI1NiIsInR5cCI6IkpXVCJ9.eyJyZWdpb24iOiJnbG9iYWwiLCJz
$> curl -X POST
"https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&to=en&to=it"
-H "Authorization: Bearer eyJhbGciOiJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNobWFjLXNoYTI1NiIsInR5cCI6IkpXVCJ9.ey
-H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'Hola, com esteu?'}]"
{"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: Bearer eyJhbGciOiJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGRzaWctbW9yZSNobWFjLXNoYTI1NiIsInR5cCI6IkpXVCJ9.ey
-H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'Hola, com esteu?'}]"
[{"detectedLanguage":{"language":"ca","score":1.0},"translations":[{"text":"Hello, how are you?","to":"en"},{"text":"Ciao com
```

2.4.5 References

Everything curl es un libro gratuito que cubre con detalle prácticamente todo lo que hay que saber sobre curl.

freecodecamp.org

Sitio oficial de curl