curl examples cookbook

Table of Contents

Force curl not to show the progress bar

Use `-s` option to make it silent:

```
curl -o index.html -s https://yurisk.info
```

Download a web page via GET request setting Chrome version 74 as the User-Agent.

Use `-A` to set User-Agent.

```
curl -o Index.html  -A  "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36"
http://example.com
```

Resources: https://developers.whatismybrowser.com/useragents/explore/

Download a page via https ignoring ceritficate errors

Add `-k` to ignore any SSL certificate warnings/errors.

```
curl -k -o Index.html https://example.com
```

Download a page using SOCKS5 proxy listening on 127.0.0.1 port 10443

Useful when you have set up an SSH tunnel to remote server listening on local port, say 10443.

```
curl -x socks5://localhost:10443  https://yurisk.info
```

Download a page using SOCKS5 proxy listening on 127.0.0.1 port 10443 and use remote host to resolve the hostname

```
curl -x socks5h://localhost:10443  https://yurisk.info
```

The idea here is to tunnel DNS requests to the remote end of the tunnel as well, for example for privacy concerns to prevent https://en.wikipedia.org/wiki/DNS_leak.

Download a page and report time spent in every step starting with resolving:

Source: https://stackoverflow.com/questions/18215389/how-do-i-measure-request-and-response-times-at-once-using-curl.

- Step 1: Put the parameters to write into a file called say *curl-params* (just for the convenience instead of CLI):

```
    time_namelookup:  %{time_namelookup}\n
       time_connect:  %{time_connect}\n
    time_appconnect:  %{time_appconnect}\n
   time_pretransfer:  %{time_pretransfer}\n
      time_redirect:  %{time_redirect}\n
 time_starttransfer:  %{time_starttransfer}\n
                      ----------\n
         time_total:  %{time_total}\n
```

- Step 2: Run the curl supplying this file *curl-params*:

```
curl -w "@curl-params" -o /dev/null -s https://example.com
```

```
    time_namelookup:  0.062
       time_connect:  0.062
    time_appconnect:  0.239
   time_pretransfer:  0.239
      time_redirect:  0.000
 time_starttransfer:  0.240
                      ----------
         time_total:  0.241
```

Resolve IP address to the owner's Autonomous System Number

Do so by sending POST query with form fields to the Team Cymru whois server. When sending any POST data with form fields, the first task is to know the fields. The esiest way to do it is to browse the form page, fill the form, open the HTML code and write down fields and their values. I did it for the page at https://asn.cymru.com/ and noted 5 fields to fill with values, the field to place IP address to query for is `bulk_paste`. In curl you specify field values with `-F 'name=value'` option:

```
curl -s  -X POST -F 'action=do_whois' -F 'family=ipv4' -F 'method_whois=whois' \
 -F 'bulk_paste=35.1.33.192' -F 'submit_paste=Submit' https://asn.cymru.com/cgi-
```

```
bin/whois.cgi | grep "|"
```

Output:

```
AS      | IP               | AS Name
36375   | 35.1.33.192      | UMICH-AS-5, US
```

Make sure curl follows redirections (`Location:`) automatically, using the correct `Referer` on each redirection

Use `-L` option to tell curl to follow the *Location* header.

```
curl -L -e ';auto' -o index.html https://example.com
```

NOTE	All the downloaded pages will be appended to the same output file, here *index.html*.

Send GET request with digest authentication

```
curl --digest http://user:pass@example.com/login
```

Download a remote file only if it's newer than the local copy

```
curl -z index.html -o index.html https://example.com/index.html
```

NOTE	file to compare/download, here *index.html*, is compared for timestamp only, no content hashing or anything else.

Enable support for compressed encoding in response, as a real browser would do

```
curl -compressed  -o w3.css https://yurisk.info/theme/css/w3.css
```

Note: this option causes curl to sent `Accept-Encoding: gzip` in the request.

Verify CORS settings of a website

```
curl -H "Access-Control-Request-Method: GET" -H "Origin: http://localhost" \
--head https://yurisk.info/2020/03/05/fortiweb-cookbook-content-routing-based-on-url-in-request-configuration/pic1.png
```

Output:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET
```

Convert curl command into ready to be compiled C source file

```
curl -o index.html https://yurisk.info --libcurl index.c
```

The output file index.c will contain the source code to implement the same command using curl C library:

```
/********* Sample code generated by the curl command line tool **********
 * All curl_easy_setopt() options are documented at:
 * https://curl.haxx.se/libcurl/c/curl_easy_setopt.html
 ************************************************************************/
#include <curl/curl.h>

int main(int argc, char *argv[])
{
  CURLcode ret;
  CURL *hnd;

  hnd = curl_easy_init();
  curl_easy_setopt(hnd, CURLOPT_BUFFERSIZE, 102400L);
  curl_easy_setopt(hnd, CURLOPT_URL, "https://yurisk.info");
  curl_easy_setopt(hnd, CURLOPT_USERAGENT, "curl/7.66.0");
  curl_easy_setopt(hnd, CURLOPT_MAXREDIRS, 50L);
  curl_easy_setopt(hnd, CURLOPT_HTTP_VERSION, (long)CURL_HTTP_VERSION_2TLS);
  curl_easy_setopt(hnd, CURLOPT_SSH_KNOWNHOSTS, "/home/yuri/.ssh/known_hosts");
  curl_easy_setopt(hnd, CURLOPT_TCP_KEEPALIVE, 1L);

  /* Here is a list of options the curl code used that cannot get generated
     as source easily. You may select to either not use them or implement
     them yourself.

  CURLOPT_WRITEDATA set to a objectpointer
```

```
  CURLOPT_INTERLEAVEDATA set to a objectpointer
  CURLOPT_WRITEFUNCTION set to a functionpointer
  CURLOPT_READDATA set to a objectpointer
  CURLOPT_READFUNCTION set to a functionpointer
  CURLOPT_SEEKDATA set to a objectpointer
  CURLOPT_SEEKFUNCTION set to a functionpointer
  CURLOPT_ERRORBUFFER set to a objectpointer
  CURLOPT_STDERR set to a objectpointer
  CURLOPT_HEADERFUNCTION set to a functionpointer
  CURLOPT_HEADERDATA set to a objectpointer

  */

  ret = curl_easy_perform(hnd);

  curl_easy_cleanup(hnd);
  hnd = NULL;

  return (int)ret;
}
/**** End of sample code ****/
```

You can now compile it to executable, provided you have `libcurl` library and its headers installed:
`gcc index.c -lcurl -o index`

Display just the HTTP response code

```
curl -w  '%{http_code}' --silent -o /dev/null https://yurisk.info
```

Output:

```
200
```

Get a page using specific version of HTTP protocol

```
 curl --http2  -s  -O  https://yurisk.info
```

Download file with SCP protocol

```
 curl scp://99.23.5.18:/root/pdf.pdf -o pdf.pdf -u root
```

Note: curl checks `~/.ssh/known_hosts` file to verify authenticity of the remote server. If the remote server is not already in the `known_hosts`, curl will refuse to connect. To prevent it - forst connect to the remote server via SSH, this will add it to the known hosts. Also, curl should be compiled with support for `libssh2` library.

Get external IP address of the machine where the curl is installed

```
curl -s http://whatismyip.akamai.com/
```

Output:

```
87.123.255.103
```

Send e-mail via SMTP

First, put the message body and From/To/Subject fields in a file:

```
# cat message.txt
From: Joe Dow <joedow@example.com>
To: Yuri <yuri@yurisk.info>
Subject: Testing curl SMTP sending

Hi, curl can now send e-mails as well!
```

Now, send the e-mail using the created file and setting e-mail envelope on the CLI:

```
curl -v  smtp://aspmx.l.google.com/smtp.example.com  --mail-from Joedow@example.com  \
 --mail-rcpt yuri@yurisk.info  --upload-file message.txt
```

Here:

- `aspmx.l.google.com` - the mail server for the recipient domain (`curl` does NOT look for the MX record itself).
- `smtp.example.com` (Optional) - domain the `curl` will use in greeting the mail server (HELO/EHLO).
- `--mail-from` - sender address set in the envelope.
- `--mail-rcpt` - recipient for the mail set in the envelope.

NOTE	the mail sending is subject to all the anti-spam checks by the receiving mail server, so I recommend to run this with the `-v` option set to see what is going on in real-time.

Make curl resolve a hostname to the custom IP address you specify without modifying hosts file or using DNS server hacks

Useful to test local copy of a website. Problem: You want curl to reach a website "example.com" at IP address 127.0.0.1 without changing local `hosts` file or setting up fake DNS server.

Solution: Use `--resolve` to specify IP address for a hostname, so curl uses it without querying real DNS servers.

```
curl -v  --resolve "example.com:80:127.0.0.1" http://example.com
```

```
* Added example.com:80:127.0.0.1 to DNS cache
* Hostname example.com was found in DNS cache
*   Trying 127.0.0.1:80...
* Connected to example.com (127.0.0.1) port 80 (#0)
> GET / HTTP/1.1
> Host: example.com
> User-Agent: curl/7.67.0
> Accept: */*
```

Show how many redirects were followed fetching the URL

Use `num_redirects` variable for that:

```
 curl -w '%{num_redirects}' -L  -o /dev/null https://cnn.com -s
2
```

Use your browser to prepare the complete curl command via "copy as curl" feature

We can use a regular browser to prepare the complete curl command by just browsing to the target site. For that: . Open Developer Tools - **F12** (works in Chrome and Firefox) . Browse to the target site/page. . In the "Network" tab of the Developer Tools find the item you want to GET with curl, right click on it, find menu "Copy as cURL", click on it - this copies to the clipboard ready-to-run curl command to that asset.

Test if a website supports the given cipher suite, e.g. obsolete sslv3 & DES

Helps to monitor servers for obsolete or not yet widely supported cipher suites. Check if site supports sslv3 (old and dangerously broken):

```
curl -k  https://yurisk.info:443 -v  --sslv3
```

Output:

```
curl: (35) error:1408F10B:SSL routines:ssl3_get_record:wrong version number
```

Check if the newest (experimental as of 2020) TLS v1.3 is enabled:

```
curl -k  https://yurisk.info:443 -v  --tlsv1.3
```

Output:

```
* OpenSSL SSL_connect: SSL_ERROR_ZERO_RETURN in connection to yurisk.info:443
* Marked for [closure]: Failed HTTPS connection
```

Check if your version of curl supports easily breakable DES algorithm:

```
curl -k -o /dev/null  https://yurisk.info:443  --ciphers DES
```

Output:

```
curl: (59) failed setting cipher list: DES
```

Fetch multiple pages with predictable pattern in their URLs

If a website has a repeating pattern in naming its resources, we can use **URL globbing**. curl understands ranges `[start-end]` and lists `{item1,item2,⋯}`. Ranges can be alphanumeric and are inclusive, i.e. [0-100] starts at 0 and includes up to 100. Ranges optionally accept step/increment value: `[10-100:2]`, here 2 is added on each step. We can use both, ranges and lists, in the same URL.

Output files: curl remembers the matched glob patterns and we can use them with `-o` to specify custom output filenames.

- Fetch all pages in https://yurisk.info/category/checkpoint-ngngxNNN.html where *NNN* goes from 2 to 9. Pay attention to the single quotes - when using on the Bash command line, the range [] and list {} operators would be otherwise interpreted by the Bash itself instead of curl.

```
curl -s -O 'https://yurisk.info/category/checkpoint-ngngx[2-9].html'
```

Output directory:

```
checkpoint-ngngx2.html
checkpoint-ngngx3.html
checkpoint-ngngx4.html
checkpoint-ngngx5.html
checkpoint-ngngx6.html
checkpoint-ngngx7.html
checkpoint-ngngx8.html
checkpoint-ngngx9.html
```

- Fetch all pages *cisco.html,fortinet.html,linux.html,checkpoint-ngngx.html* inside the *category* folder:

```
 curl   -O 'https://yurisk.info/category/{cisco,fortinet,linux,checkpoint-ngngx}.html'
```

Output:

```
checkpoint-ngngx.html
cisco.html
fortinet.html
linux.html
```

- Download pages with alphabetical ranges.

```
curl-O -s https://yurisk.info/test[a-z]
```

How to prevent errors on URLs that contain brackets

If the curl uses brackets (square and curly) for ranges (see above), how do we work with URLs containing such symbols? By using the -g option to curl which turns off globbing. It also means we can't use ranges with URLs that contain brackets.

```
curl -g https://example.com/{ids}?site=example.gov
```

Github: list names of all public repositories for a given user

To query the user's repositories, the URL should have the form of `https://api.github.com/users/<username>/repos`. For example, let's get all the repositories for `curl` project:

```
curl -s  https://api.github.com/users/curl/repos | awk '/\wname/'
```

Output:

```
    "full_name": "curl/build-images",
    "full_name": "curl/curl",
    "full_name": "curl/curl-cheat-sheet",
    "full_name": "curl/curl-docker",
    "full_name": "curl/curl-for-win",
    "full_name": "curl/curl-fuzzer",
    "full_name": "curl/curl-up",
    "full_name": "curl/curl-www",
    "full_name": "curl/doh",
    "full_name": "curl/fcurl",
    "full_name": "curl/h2c",
    "full_name": "curl/stats",
```

Note: Github imposes rate limits on the unauthorized requests, currently 60 requests/hour is the maximum. You can check how many queries are left with the *X-Ratelimit-Remaining* header:

```
 curl -s -i  https://api.github.com/users/curl/repos | grep X-Ratelimit-Remaining
X-Ratelimit-Remaining: 54`
```

Display weather report for a given city

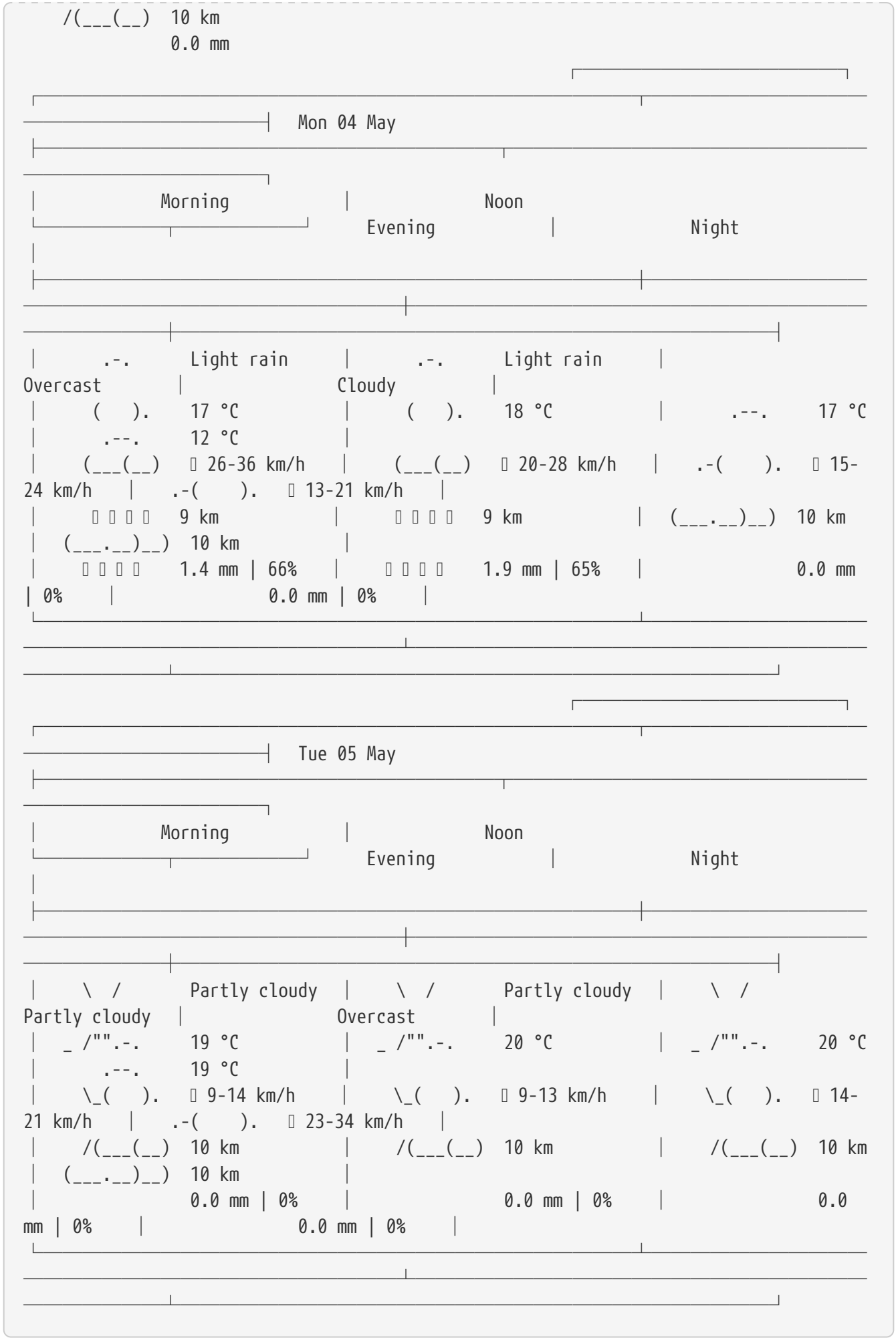
There are many websites to query for weather information on the CLI, most popular seems to be wttr.in, so let's use it to get the weather in Milan:

```
 curl wttr.in/Milan
```

Output:

```
Weather report: Milan

    \  /       Partly cloudy
  _ /"".-.     17 °C
    \_(   ).   ↙ 6 km/h
```


/(___(__) 10 km
0.0 mm
Mon 04 May
Morning
Noon
Evening
Night
Light rain
17 °C
12 °C
26-36 km/h
9 km
1.4 mm | 66%
Light rain
18 °C
20-28 km/h
9 km
1.9 mm | 65%
17 °C
15-24 km/h
13-21 km/h
10 km
0.0 mm | 0%
0.0 mm | 0%
Tue 05 May
Morning
Noon
Evening
Night
Partly cloudy
19 °C
19 °C
9-14 km/h
10 km
0.0 mm | 0%
Partly cloudy
20 °C
9-13 km/h
10 km
0.0 mm | 0%
Partly cloudy
20 °C
14-21 km/h
23-34 km/h
10 km
0.0 mm | 0%
0.0 mm | 0%