Everything curl

Downloads

"Download" means getting data from a server on a network, and the server is then clearly considered to be "above" you. This is loading data down from the server onto your machine where you are running curl.

Downloading is probably the most common use case for curl—retrieving the specific data pointed to by a URL onto your machine.

What exactly is downloading?

You specify the resource to download by giving curl a URL. curl defaults to downloading a URL unless told otherwise, and the URL identifies what to download. In this example the URL to download is "http://example.com":

```
curl http://example.com
```

The URL is broken down into its individual components (as explained elsewhere), the correct server is contacted and is then asked to deliver the specific resource—often a file. The server then delivers the data, or it refuses or perhaps the client asked for the wrong data and then that data is delivered.

A request for a resource is protocol-specific so a FTP:// URL works differently than an HTTP:// URL or an SFTP:// URL.

A URL without a path part, that is a URL that has a host name part only (like the "http://example.com" example above) will get a slash ('/') appended to it internally and then that is the resource curl will ask for from the server.

If you specify multiple URLs on the command line, curl will download each URL one by one. It will not start the second transfer until the first one is complete, etc.

Storing downloads

If you try the example download as in the previous section, you will notice that curl will output the downloaded data to stdout unless told to do something else. Outputting data to stdout is really useful when you want to pipe it into another program or similar, but it is not always the optimal way to deal with your downloads.

Give curl a specific file name to save the download in with `-o [filename]` (with `--output` as the long version of the option), where filename is either just a file name, a relative path to a file name or a full path to the file.

Also note that you can put the `-o` before or after the URL; it makes no difference:

```
curl -o output.html http://example.com/
curl -o /tmp/index.html http://example.com/
curl http://example.com -o ../../folder/savethis.html
```

This is, of course, not limited to http:// URLs but works the same way no matter which type of URL you download:

```
curl -o file.txt ftp://example.com/path/to/file-name.ext
```

If you ask curl to send the output to the terminal, it attempts to detect and prevent binary data from being sent there since that can seriously mess up your terminal (sometimes to the point where it stops working). You can override curl's binary-output-prevention and force the output to get sent to stdout by using `-o -`.

curl has several other ways to store and name the downloaded data. Details follow.

Download to a file named by the URL

Many URLs, however, already contain the file name part in the rightmost end. curl lets you use that as a shortcut so you do not have to repeat it with `-o`. So instead of:

```
curl -o file.html http://example.com/file.html
```

You can save the remove URL resource into the local file 'file.html' with this:

```
curl -O http://example.com/file.html
```

This is the `-O` (uppercase letter o) option, or `--remote-name` for the long name version. The -O option selects the local file name to use by picking the file name part of the URL that you provide. This is important. You specify the URL and curl picks the name from this data. If the site redirects curl further (and if you tell curl to follow redirects), it does not change the file name curl will use for storing this.

Get the target file name from the server

HTTP servers have the option to provide a header named `Content-Disposition:` in responses. That header may contain a suggested file name for the contents delivered, and curl can be told to use that hint to name its local file. The `-J / --remote-header-name` enables this. If you also use the `-O` option, it makes curl use the file name from the URL by default and only *if* there's actually a valid Content-Disposition header available, it switches to saving using that name.

-J has some problems and risks associated with it that users need to be aware of:

1. It will only use the rightmost part of the suggested file name, so any path or directories the server suggests will be stripped out.
2. Since the file name is entirely selected by the server, curl will, of course, overwrite any preexisting local file in your current directory if the server happens to provide such a file name.

3. File name encoding and character sets issues. curl does not decode the name in any way, so you may end up with a URL-encoded file name where a browser would otherwise decode it to something more readable using a sensible character set.

HTML and charsets

curl will download the exact binary data that the server sends. This might be of importance to you in case, for example, you download a HTML page or other text data that uses a certain character encoding that your browser then displays as expected. curl will then not translate the arriving data.

A common example where this causes some surprising results is when a user downloads a web page with something like:

```
curl https://example.com/ -o storage.html
```

…and when inspecting the `storage.html` file after the fact, the user realizes that one or more characters look funny or downright wrong. This might occur because the server sent the characters using charset X, while your editor and environment use charset Y. In an ideal world, we would all use UTF-8 everywhere but unfortunately, that is still not the case.

A common work-around for this issue that works decently is to use the common `iconv` utility to translate a text file to and from different charsets.

Compression

curl allows you to ask HTTP and HTTPS servers to provide compressed versions of the data and then perform automatic decompression of it on arrival. In situations where bandwidth is more limited than CPU this will help you receive more data in a shorter amount of time.

HTTP compression can be done using two different mechanisms, one which might be considered "The Right Way" and the other that is the way that everyone actually uses and is the widespread and popular way to do it. The common way to compress HTTP content is using the **Content-Encoding** header. You ask curl to use this with the `--compressed` option:

```
curl --compressed http://example.com/
```

With this option enabled (and if the server supports it) it delivers the data in a compressed way and curl will decompress it before saving it or sending it to stdout. This usually means that as a user you do not really see or experience the compression other than possibly noticing a faster transfer.

The `--compressed` option asks for Content-Encoding compression using one of the supported compression algorithms. There's also the rarer **Transfer-Encoding** method, which is the header that was created for this automated method but was never really widely adopted. You can tell curl to ask for Transfer-Encoded compression with `--tr-encoding`:

```
curl --tr-encoding http://example.com/
```

In theory, there's nothing that prevents you from using both in the same command line, although in practice, you may then experience that some servers get a little confused when ask to compress in two different ways. It's generally safer to just pick one.

Shell redirects

When you invoke curl from a shell or some other command-line prompt system, that environment generally provides you with a set of output redirection abilities. In most Linux and Unix shells and with Windows' command prompts, you direct stdout to a file with `> filename`. Using this, of course, makes the use of -o or -O superfluous.

```
curl http://example.com/ > example.html
```

Redirecting output to a file redirects all output from curl to that file, so even if you ask to transfer more than one URL to stdout, redirecting the output will get all the URLs' output stored in that single file.

```
curl http://example.com/1 http://example.com/2 > files
```

Unix shells usually allow you to redirect the *stderr* stream separately. The stderr stream is usually a stream that also gets shown in the terminal, but you can redirect it separately from the stdout stream. The stdout stream is for the data while stderr is metadata and errors, etc., that are not data. You can redirect stderr with `2>file` like this:

```
curl http://example.com > files.html 2>errors
```

Multiple downloads

As curl can be told to download many URLs in a single command line, there are, of course, times when you want to store these downloads in nicely named local files.

The key to understanding this is that each download URL needs its own "storage instruction". Without said "storage instruction", curl will default to sending the data to stdout. If you ask for two URLs and only tell curl where to save the first URL, the second one is sent to stdout. Like this:

```
curl -o one.html http://example.com/1 http://example.com/2
```

The "storage instructions" are read and handled in the same order as the download URLs so they do not have to be next to the URL in any way. You can round up all the output options first, last or interleaved with the URLs. You choose.

These examples all work the same way:

```
curl -o 1.txt -o 2.txt http://example.com/1 http://example.com/2
curl http://example.com/1 http://example.com/2 -o 1.txt -o 2.txt
curl -o 1.txt http://example.com/1 http://example.com/2 -o 2.txt
curl -o 1.txt http://example.com/1 -o 2.txt http://example.com/2
```

The `-O` is similarly just an instruction for a single download so if you download multiple URLs, use more of them:

```
curl -O -O http://example.com/1 http://example.com/2
```

Use the URL's file name part for all URLs

As a reaction to adding a hundred `-O` options when using a hundred URLs, we introduced an option called `--remote-name-all`. This makes `-O` the default operation for all given URLs. You can still provide individual "storage instructions" for URLs but if you leave one out for a URL that gets downloaded, the default action is then switched from stdout to -O style.

"My browser shows something else"

A common use case is using curl to get a URL that you can get in your browser when you paste the URL in the browser's address bar.

A browser getting a URL as input does so much more and in so many different ways than curl that what curl shows in your terminal output is probably not at all what you see in your browser window.

Client differences

Curl only gets exactly what you ask it to get and it never parses the actual content—the data—that the server delivers. A browser gets data and it activates different parsers depending on what kind of content it thinks it gets. For example, if the data is HTML it will parse it to display a web page and possibly download other sub resources such as images, JavaScript and CSS files. When curl downloads a HTML it will just get that single HTML resource, even if it, when parsed by a browser, would trigger a whole busload of more downloads. If you want curl to download any sub-resources as well, you need to pass those URLs to curl and ask it to get those, just like any other URLs.

Clients also differ in how they send their requests, and some aspects of a request for a resource include, for example, format preferences, asking for compressed data, or just telling the server from which previous page we are "coming from". curl's requests will differ a little or a lot from how your browser sends its requests.

Server differences

The server that receives the request and delivers data is often setup to act in certain ways depending on what kind of client it thinks communicates with it. Sometimes it is as innocent as trying to deliver the best content for the client, sometimes it is to hide some content for some clients or even to try to work around known problems in specific browsers. Then there's also, of course, various kind of login systems that might rely on HTTP authentication or cookies or the client being from the pre-validated IP address range.

Sometimes getting the same response from a server using curl as the response you get with a browser ends up really hard work. Users then typically record their browser sessions with the browser's networking tools and then compare that recording with recorded data from curl's `--trace-ascii` option and proceed to modify curl's requests (often with `-H / --header`) until the server starts to respond the same to both.

This type of work can be both time consuming and tedious. You should always do this with permission from the server owners or admins.

Intermediaries' fiddlings

Intermediaries are proxies, explicit or implicit ones. Some environments will force you to use one or you may choose to use one for various reasons, but there are also the transparent ones that will intercept your network traffic silently and proxy it for you no matter what you want.

Proxies are "middle men" that terminate the traffic and then act on your behalf to the remote server. This can introduce all sorts of explicit filtering and "saving" you from certain content or even "protecting" the remote server from what data you try to send to it, but even more so it introduces another software's view on how the protocol works and what the right things to do are.

Interfering intermediaries are often the cause of lots of head aches and mysteries down to downright malicious modifications of content.

We strongly encourage you to use HTTPS or other means to verify that the contents you are downloading or uploading are really the data that the remote server has sent to you and that your precious bytes end up verbatim at the intended destination.

Rate limiting

When curl transfers data, it will attempt to do that as fast as possible. It goes for both uploads and downloads. Exactly how fast that will be depends on several factors, including your computer's ability, your own network connection's bandwidth, the load on the remote server you are transferring to/from and the latency to that server. And your curl transfers are also likely to compete with other transfers on the networks the data travels over, from other users or just other apps by the same user.

In many setups, however, you will find that you can more or less saturate your own network connection with a single curl command line. If you have a 10 megabit per second

connection to the Internet, chances are curl can use all of those 10 megabits to transfer data.

For most use cases, using as much bandwidth as possible is a good thing. It makes the transfer faster, it makes the curl command complete sooner and it will make the transfer use resources from the server for a shorter period of time.

Sometimes you will, however, find that having curl starve out other network functions on your local network connection is inconvenient. In these situations you may want to tell curl to slow down so that other network users get a better chance to get their data through as well. With `--limit-rate [speed]` you can tell curl to not go faster than the given number of bytes per second. The rate limit value can be given with a letter suffix using one of K, M and G for kilobytes, megabytes and gigabytes.

To make curl not download data any faster than 200 kilobytes per second:

```
curl https://example.com/ --limit-rate 200K
```

The given limit is the maximum *average speed* allowed, counted during the entire transfer. It means that curl might use higher transfer speeds in short bursts, but over time it uses no more than the given rate.

Also note that curl never knows what the maximum possible speed is—it will simply go as fast as it can and is allowed. You may know your connection's maximum speed, but curl does not.

Maximum filesize

When you want to make sure your curl command line will not try to download a too-large file, you can instruct curl to stop before doing that, if it knows the size before the transfer starts! Maybe that would use too much bandwidth, take too long time or you do not have enough space on your hard drive:

```
curl --max-filesize 100000 https://example.com/
```

Give curl the largest download you will accept in number of bytes and if curl can figure out the size before the transfer starts it will abort before trying to download something larger.

There are many situations in which curl cannot figure out the size at the time the transfer starts and this option will not affect those transfers, even if they may end up larger than the specified amount.

Storing metadata in file system

When saving a download to a file with curl, the `--xattr` option tells curl to also store certain file metadata in "extended file attributes". These extended attributes are standardized name/value pairs stored in the file system, assuming one of the supported file systems and operating systems are used.

Currently, the URL is stored in the `xdg.origin.url` attribute and, for HTTP, the content type is stored in the `mime_type` attribute. If the file system does not support extended attributes when this option is set, a warning is issued.

Raw

When `--raw` is used, it disables all internal HTTP decoding of content or transfer encodings and instead makes curl passed on unaltered, raw, data.

This is typically used if you are writing a middle software and you want to pass on the content to another HTTP client and allow that to do the decoding instead.

Retrying failed attempts

Normally curl will only make a single attempt to perform a transfer and return an error if not successful. Using the `--retry` option you can tell curl to retry certain failed transfers.

If a transient error is returned when curl tries to perform a transfer, it will retry this number of times before giving up. Setting the number to 0 makes curl do no retries (which is the default). Transient error means either: a timeout, an FTP 4xx response code or an HTTP 5xx response code.

When curl is about to retry a transfer, it will first wait one second and then for all forthcoming retries it will double the waiting time until it reaches 10 minutes which then will be the delay between the rest of the retries. Using `--retry-delay` you can disable this exponential backoff algorithm and set your own delay between the attempts. With `--retry-max-time` you cap the total time allowed for retries. The `--max-time` option will still specify the longest time a single of these transfers is allowed to spend.

Resuming and ranges

Resuming a download means first checking the size of what is already present locally and then asking the server to send the rest of it so it can be appended. curl also allows resuming the transfer at a custom point without actually having anything already locally present.

curl supports resumed downloads on several protocols. Tell it where to start the transfer with the `-C, --continue-at` option that takes either a plain numerical byte counter offset where to start or the string `-` that asks curl to figure it out itself based on what it knows. When using `-`, curl will use the destination file name to figure out how much data that is already present locally and ask use that as an offset when asking for more data from the server.

To start downloading an FTP file from byte offset 100:

```
curl --continue-at 100 ftp://example.com/bigfile
```

Continue downloading a previously interrupted download:

```
curl --continue-at - http://example.com/bigfile -O
```

If you instead just want a specific byte range from the remote resource transferred, you can ask for only that. For example, when you only want 1000 bytes from offset 100 to avoid having to download the entire huge remote file:

```
curl --range 100-1099 http://example.com/bigfile
```