Responses

HTTP responses

When an HTTP client talks HTTP to a server, the server *will* respond with an HTTP response message or curl will consider it an error and returns 52 with the error message "Empty reply from server".

Size of an HTTP response

An HTTP response has a certain size and curl needs to figure it out. There are several different ways to signal the end of an HTTP response but the most basic way is to use the `Content-Length:` header in the response and with that specify the exact number of bytes in the response body.

Some early HTTP server implementations had problems with file sizes greater than 2GB and wrongly managed to send Content-Length: headers with negative sizes or otherwise just plain wrong data. curl can be told to ignore the Content-Length: header completely with `--ignore-content-length` . Doing so may have some other negative side-effects but should at least let you get the data.

HTTP response codes

An HTTP transfer gets a 3 digit response code back in the first response line. The response code is the server's way of giving the client a hint about how the request was handled.

It is important to note that curl does not consider it an error even if the response code would indicate that the requested document could not be delivered (or similar). curl considers a successful sending and receiving of HTTP to be good.

The first digit of the HTTP response code is a kind of "error class":

- 1xx: transient response, more is coming
- 2xx: success
- 3xx: a redirect
- 4xx: the client asked for something the server could not or would not deliver
- 5xx: there's problem in the server

Remember that you can use curl's `--write-out` option to extract the response code. See the --write-out section.

CONNECT response codes

Since there can be a HTTP request and a separate CONNECT request in the same curl transfer, we often separate the CONNECT response (from the proxy) from the remote server's HTTP response.

The CONNECT is also an HTTP request so it gets response codes in the same numeric range and you can use `--write-out` to extract that code as well.

Chunked transfer encoding

An HTTP 1.1 server can decide to respond with a "chunked" encoded response, a feature that was not present in HTTP 1.0.

When receiving a chunked response, there's no Content-Length: for the response to indicate its size. Instead, there's a `Transfer-Encoding: chunked` header that tells curl there's chunked data coming and then in the response body, the data comes in a series of "chunks". Every individual chunk starts with the size of that particular chunk (in hexadecimal), then a newline and then the contents of the chunk. This is repeated over and over until the end of the response, which is signalled with a zero sized chunk. The point of this response encoding is for the client to be able to figure out when the response has ended even though the server did not know the full size before it started to send it. This is usually the case when the response is dynamic and generated at the point when the request comes.

Clients like curl will, of course, decode the chunks and not show the chunk sizes to users.

Gzipped transfers

Responses over HTTP can be sent in compressed format. This is most commonly done by the server when it includes a `Content-Encoding: gzip` in the response as a hint to the client. Compressed responses make a lot of sense when either static resources are sent (that were compressed previously) or even in run-time when there's more CPU power available than bandwidth. Sending a much smaller amount of data is often preferred.

You can ask curl to both ask for compressed content *and* automatically and transparently uncompress gzipped data when receiving content encoded gzip (or in fact any other compression algorithm that curl understands) by using `--compressed`:

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

Transfer encoding

A less common feature used with transfer encoding is compression.

Compression in itself is common. Over time the dominant and web compatible way to do compression for HTTP has become to use `Content-Encoding` as described in the section above. But HTTP was originally intended and specified to allow transparent compression as a transfer encoding, and curl supports this feature.

The client then simply asks the server to do compression transfer encoding and if acceptable, it will respond with a header indicating that it will and curl will then transparently uncompress that data on arrival. A user enables asking for compressed transfer encoding with `--tr-encoding`:

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

It should be noted that not many HTTP servers in the wild support this.

Pass on transfer encoding

In some situations you may want to use curl as a proxy or other in-between software. In those cases, curl's way to deal with transfer-encoding headers and then decoding the actual data transparently may not be desired, if the end receiver *also* expects to do the same.

You can then ask curl to pass on the received data, without decoding it. That means passing on the sizes in the chunked encoding format or the compressed format when compressed transfer encoding is used etc.

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