-d, --data <data>

(HTTP MQTT) Sends the specified data in a POST request to the HTTP server, in the same way that a browser does when a user has filled in an HTML form and presses the submit button. This will cause curl to pass the data to the server using the content-type application/x-www-form-urlencoded. Compare to -F, --form.

--data-raw is almost the same but does not have a special interpretation of the @ character. To post data purely binary, you should instead use the --data-binary option. To URL-encode the value of a form field you may use --data-urlencode.

If any of these options is used more than once on the same command line, the data pieces specified will be merged together with a separating &-symbol. Thus, using '-d name=daniel -d skill=lousy' would generate a post chunk that looks like 'name=daniel&skill=lousy'.

If you start the data with the letter @, the rest should be a file name to read the data from, or - if you want curl to read the data from stdin. Posting data from a file named 'foobar' would thus be done with -d, --data @foobar. When -d, --data is told to read from a file like that, carriage returns and newlines will be stripped out. If you don't want the @ character to have a special interpretation use --data-raw instead.

See also --data-binary, --data-urlencode and --data-raw. This option overrides -F, --form and -I, --head and -T, --upload-file.

--disallow-username-in-url

(HTTP) This tells curl to exit if passed a url containing a username.

See also --proto. Added in 7.61.0.

-F, --form <name=content>

(HTTP SMTP IMAP) For HTTP protocol family, this lets curl emulate a filled-in form in which a user has pressed the submit button. This causes curl to POST data using the Content-Type multipart/form-data according to RFC 2388.

For SMTP and IMAP protocols, this is the means to compose a multipart mail message to transmit.

This enables uploading of binary files etc. To force the 'content' part to be a file, prefix the file name with an @ sign. To just get the content part from a file, prefix the file name with the symbol <. The difference between @ and < is then that @ makes a file get attached in the post as a file upload, while the < makes a text field and just get the contents for that text field from a file.

Tell curl to read content from stdin instead of a file by using - as filename. This goes for both @ and < constructs. When stdin is used, the contents is buffered in memory first by curl to determine its size and allow a possible resend. Defining a part's data from a named non-regular file (such as a named pipe or similar) is unfortunately not subject to buffering and will be effectively read at transmission time; since the full size is unknown before the transfer starts, such data is sent as chunks by HTTP and rejected by IMAP.

Example: send an image to an HTTP server, where 'profile' is the name of the form-field to which the file portrait.jpg will be the input:

```
 curl -F profile=@portrait.jpg https://example.com/upload.cgi
```

Example: send your name and shoe size in two text fields to the server:

```
 curl -F name=John -F shoesize=11 https://example.com/
```

Example: send your essay in a text field to the server. Send it as a plain text field, but get the contents for it from a local file:

```
 curl -F "story=<hugefile.txt" https://example.com/
```

You can also tell curl what Content-Type to use by using 'type=', in a manner similar to:

```
 curl -F "web=@index.html;type=text/html" example.com
```

or

```
 curl -F "name=daniel;type=text/foo" example.com
```

You can also explicitly change the name field of a file upload part by setting filename=, like this:

```
 curl -F "file=@localfile;filename=nameinpost" example.com
```

If filename/path contains ',' or ';', it must be quoted by double-quotes like:

```
 curl -F "file=@\"local,file\";filename=\"name;in;post\"" example.com
```

or

```
 curl -F 'file=@"local,file";filename="name;in;post"' example.com
```

Note that if a filename/path is quoted by double-quotes, any double-quote or backslash within the filename must be escaped by backslash.

Quoting must also be applied to non-file data if it contains semicolons, leading/trailing spaces or leading double quotes:

```
 curl -F 'colors="red; green; blue";type=text/x-myapp' example.com
```

You can add custom headers to the field by setting headers=, like

```
  curl -F "submit=OK;headers=\"X-submit-type: OK\"" example.com
```

or

```
  curl -F "submit=OK;headers=@headerfile" example.com
```

The headers= keyword may appear more that once and above notes about quoting apply. When headers are read from a file, Empty lines and lines starting with '#' are comments and ignored; each header can be folded by splitting between two words and starting the continuation line with a space; embedded carriage-returns and trailing spaces are stripped. Here is an example of a header file contents:

```
  # This file contain two headers.
  X-header-1: this is a header

  # The following header is folded.
  X-header-2: this is
  another header
```

To support sending multipart mail messages, the syntax is extended as follows:
- name can be omitted: the equal sign is the first character of the argument,
- if data starts with '(', this signals to start a new multipart: it can be followed by a content type specification.
- a multipart can be terminated with a '=)' argument.

Example: the following command sends an SMTP mime e-mail consisting in an inline part in two alternative formats: plain text and HTML. It attaches a text file:

```
 curl -F '=(;type=multipart/alternative' \
  -F '=plain text message' \
  -F '= <body>HTML message</body>;type=text/html' \
  -F '=)' -F '=@textfile.txt' ... smtp://example.com
```

Data can be encoded for transfer using encoder=. Available encodings are binary and 8bit that do nothing else than adding the corresponding Content-Transfer-Encoding header, 7bit that only rejects 8-bit characters with a transfer error, quoted-printable and base64 that encodes data according to the corresponding schemes, limiting lines length to 76 characters.

Example: send multipart mail with a quoted-printable text message and a base64 attached file:

```
 curl -F '=text message;encoder=quoted-printable' \
  -F '=@localfile;encoder=base64' ... smtp://example.com
```

See further examples and details in the MANUAL.

This option can be used multiple times.

This option overrides -d, --data and -I, --head and -T, --upload-file.

-G, --get

When used, this option will make all data specified with -d, --data, --data-binary or --data-urlencode to be used in an HTTP GET request instead of the POST request that otherwise would be used. The data will be appended to the URL with a '?' separator.

If used in combination with -I, --head, the POST data will instead be appended to the URL with a HEAD request.

If this option is used several times, only the first one is used. This is because undoing a GET doesn't make sense, but you should then instead enforce the alternative method you prefer.

--happy-eyeballs-timeout-ms <milliseconds>

Happy eyeballs is an algorithm that attempts to connect to both IPv4 and IPv6 addresses for dual-stack hosts, preferring IPv6 first for the number of milliseconds. If the IPv6 address cannot be connected to within that time then a connection attempt is made to the IPv4 address in parallel. The first connection to be established is the one that is used.

The range of suggested useful values is limited. Happy Eyeballs RFC 6555 says "It is RECOMMENDED that connection attempts be paced 150-250 ms apart to balance human factors against network load." libcurl currently defaults to 200 ms. Firefox and Chrome currently default to 300 ms.

If this option is used several times, the last one will be used.

Added in 7.59.0.

--ftp-create-dirs

(FTP SFTP) When an FTP or SFTP URL/operation uses a path that doesn't currently exist on the server, the standard behavior of curl is to fail. Using this option, curl will instead attempt to create missing directories.

See also --create-dirs.