Related:
Man Page
FAQ

curl tutorial

Simple Usage

Get the main page from a web-server:

```
curl https://www.example.com/
```

Get the README file the user's home directory at funet's ftp-server:

```
curl ftp://ftp.funet.fi/README
```

Get a web page from a server using port 8000:

```
curl http://www.weirdserver.com:8000/
```

Get a directory listing of an FTP site:

```
curl ftp://ftp.funet.fi
```

Get the definition of curl from a dictionary:

```
curl dict://dict.org/m:curl
```

Fetch two documents at once:

```
curl ftp://ftp.funet.fi/ http://www.weirdserver.com:8000/
```

Get a file off an FTPS server:

```
curl ftps://files.are.secure.com/secrets.txt
```

or use the more appropriate FTPS way to get the same file:

```
curl --ftp-ssl ftp://files.are.secure.com/secrets.txt
```

Get a file from an SSH server using SFTP:

```
curl -u username sftp://example.com/etc/issue
```

Get a file from an SSH server using SCP using a private key (not password-protected) to authenticate:

```
curl -u username: --key ~/.ssh/id_rsa scp://example.com/~/file.txt
```

Get a file from an SSH server using SCP using a private key (password-protected) to authenticate:

```
curl -u username: --key ~/.ssh/id_rsa --pass private_key_password
scp://example.com/~/file.txt
```

Get the main page from an IPv6 web server:

```
curl "http://[2001:1890:1112:1::20]/"
```

Get a file from an SMB server:

```
curl -u "domain\username:passwd" smb://server.example.com/share/file.txt
```

Download to a File

Get a web page and store in a local file with a specific name:

```
curl -o thatpage.html http://www.example.com/
```

Get a web page and store in a local file, make the local file get the name of the remote document (if no file name part is specified in the URL, this will fail):

```
curl -O http://www.example.com/index.html
```

Fetch two files and store them with their remote names:

```
curl -O www.haxx.se/index.html -O curl.se/download.html
```

Using Passwords

FTP

To ftp files using name+passwd, include them in the URL like:

```
curl ftp://name:passwd@machine.domain:port/full/path/to/file
```

or specify them with the -u flag like

```
curl -u name:passwd ftp://machine.domain:port/full/path/to/file
```

FTPS

It is just like for FTP, but you may also want to specify and use SSL-specific options for certificates etc.

Note that using `FTPS://` as prefix is the "implicit" way as described in the standards while the recommended "explicit" way is done by using FTP:// and the `--ftp-ssl` option.

SFTP / SCP

This is similar to FTP, but you can use the `--key` option to specify a private key to use instead of a password. Note that the private key may itself be protected by a password that is unrelated to the login password of the remote system; this password is specified using the `--pass` option. Typically, curl will automatically extract the public key from the private key file, but in cases where curl does not have the proper library support, a matching public key file must be specified using the `--pubkey` option.

HTTP

Curl also supports user and password in HTTP URLs, thus you can pick a file like:

```
curl http://name:passwd@machine.domain/full/path/to/file
```

or specify user and password separately like in

```
curl -u name:passwd http://machine.domain/full/path/to/file
```

HTTP offers many different methods of authentication and curl supports several: Basic, Digest, NTLM and Negotiate (SPNEGO). Without telling which method to use, curl defaults to Basic. You can also ask curl to pick the most secure ones out of the ones that the server accepts for the given URL, by using `--anyauth`.

Note! According to the URL specification, HTTP URLs can not contain a user and password, so that style will not work when using curl via a proxy, even though curl allows it at other times. When using a proxy, you *must* use the `-u` style for user and password.

HTTPS

Probably most commonly used with private certificates, as explained below.

Proxy

curl supports both HTTP and SOCKS proxy servers, with optional authentication. It does not have special support for FTP proxy servers since there are no standards for those, but it can still be made to work with many of them. You can also use both HTTP and SOCKS proxies to transfer files to and from FTP servers.

Get an ftp file using an HTTP proxy named my-proxy that uses port 888:

```
curl -x my-proxy:888 ftp://ftp.leachsite.com/README
```

Get a file from an HTTP server that requires user and password, using the same proxy as above:

```
curl -u user:passwd -x my-proxy:888 http://www.get.this/
```

Some proxies require special authentication. Specify by using -U as above:

```
curl -U user:passwd -x my-proxy:888 http://www.get.this/
```

A comma-separated list of hosts and domains which do not use the proxy can be specified as:

```
curl --noproxy localhost,get.this -x my-proxy:888 http://www.get.this/
```

If the proxy is specified with `--proxy1.0` instead of `--proxy` or `-x`, then curl will use HTTP/1.0 instead of HTTP/1.1 for any `CONNECT` attempts.

curl also supports SOCKS4 and SOCKS5 proxies with `--socks4` and `--socks5`.

See also the environment variables Curl supports that offer further proxy control.

Most FTP proxy servers are set up to appear as a normal FTP server from the client's perspective, with special commands to select the remote FTP server. curl supports the `-u`, `-Q` and `--ftp-account` options that can be used to set up transfers through many FTP proxies. For example, a file can be uploaded to a remote FTP server using a Blue Coat FTP proxy with the options:

```
curl -u "username@ftp.server Proxy-Username:Remote-Pass"
  --ftp-account Proxy-Password --upload-file local-file
  ftp://my-ftp.proxy.server:21/remote/upload/path/
```

See the manual for your FTP proxy to determine the form it expects to set up transfers, and curl's `-v` option to see exactly what curl is sending.

Ranges

HTTP 1.1 introduced byte-ranges. Using this, a client can request to get only one or more subparts of a specified document. Curl supports this with the `-r` flag.

Get the first 100 bytes of a document:

```
curl -r 0-99 http://www.get.this/
```

Get the last 500 bytes of a document:

```
curl -r -500 http://www.get.this/
```

Curl also supports simple ranges for FTP files as well. Then you can only specify start and stop position.

Get the first 100 bytes of a document using FTP:

```
curl -r 0-99 ftp://www.get.this/README
```

Uploading

FTP / FTPS / SFTP / SCP

Upload all data on stdin to a specified server:

```
curl -T - ftp://ftp.upload.com/myfile
```

Upload data from a specified file, login with user and password:

```
curl -T uploadfile -u user:passwd ftp://ftp.upload.com/myfile
```

Upload a local file to the remote site, and use the local file name at the remote site too:

```
curl -T uploadfile -u user:passwd ftp://ftp.upload.com/
```

Upload a local file to get appended to the remote file:

```
curl -T localfile -a ftp://ftp.upload.com/remotefile
```

Curl also supports ftp upload through a proxy, but only if the proxy is configured to allow that kind of tunneling. If it does, you can run curl in a fashion similar to:

```
curl --proxytunnel -x proxy:port -T localfile ftp.upload.com
```

SMB / SMBS

```
curl -T file.txt -u "domain\username:passwd"
  smb://server.example.com/share/
```

HTTP

Upload all data on stdin to a specified HTTP site:

```
curl -T - http://www.upload.com/myfile
```

Note that the HTTP server must have been configured to accept PUT before this can be done successfully.

For other ways to do HTTP data upload, see the POST section below.

Verbose / Debug

If curl fails where it isn't supposed to, if the servers don't let you in, if you can't understand the responses: use the `-v` flag to get verbose fetching. Curl will output lots of info and what it sends and receives in order to let the user see all client-server interaction (but it won't show you the actual data).

```
curl -v ftp://ftp.upload.com/
```

To get even more details and information on what curl does, try using the `--trace` or `--trace-ascii` options with a given file name to log to, like this:

```
curl --trace trace.txt www.haxx.se
```

Detailed Information

Different protocols provide different ways of getting detailed information about specific files/documents. To get curl to show detailed information about a single file, you should use `-I`/`--head` option. It displays all available info on a single file for HTTP and FTP. The HTTP information is a lot more extensive.

For HTTP, you can get the header information (the same as `-I` would show) shown before the data by using `-i`/`--include`. Curl understands the `-D`/`--dump-header` option when getting files from both FTP and HTTP, and it will then store the headers in the specified file.

Store the HTTP headers in a separate file (headers.txt in the example):

```
curl --dump-header headers.txt curl.se
```

Note that headers stored in a separate file can be very useful at a later time if you want curl to use cookies sent by the server. More about that in the cookies section.

POST (HTTP)

It's easy to post data using curl. This is done using the `-d <data>` option. The post data must be urlencoded.

Post a simple "name" and "phone" guestbook.

```
curl -d "name=Rafael%20Sagula&phone=3320780" http://www.where.com/guest.cgi
```

How to post a form with curl, lesson #1:

Dig out all the `<input>` tags in the form that you want to fill in.

If there's a "normal" post, you use `-d` to post. `-d` takes a full "post string", which is in the format

```
<variable1>=<data1>&<variable2>=<data2>&...
```

The 'variable' names are the names set with `"name="` in the `<input>` tags, and the data is the contents you want to fill in for the inputs. The data *must* be properly URL encoded. That means you replace space with + and that you replace weird letters with %XX where XX is the hexadecimal representation of the letter's ASCII code.

Example:

(page located at `http://www.formpost.com/getthis/`)

```
<form action="post.cgi" method="post">
<input name=user size=10>
<input name=pass type=password size=10>
<input name=id type=hidden value="blablabla">
<input name=ding value="submit">
</form>
```

We want to enter user 'foobar' with password '12345'.

To post to this, you enter a curl command line like:

```
curl -d "user=foobar&pass=12345&id=blablabla&ding=submit"
  http://www.formpost.com/getthis/post.cgi
```

While `-d` uses the application/x-www-form-urlencoded mime-type, generally understood by CGI's and similar, curl also supports the more capable multipart/form-data type. This latter type supports things like file upload.

`-F` accepts parameters like `-F "name=contents"`. If you want the contents to be read from a file, use `@filename` as contents. When specifying a file, you can also specify the file content type by appending `;type=<mime type>` to the file name. You can also post the contents of several files in one field. For example, the field name 'coolfiles' is used to send three files, with different content types using the following syntax:

```
curl -F "coolfiles=@fil1.gif;type=image/gif,fil2.txt,fil3.html"
  http://www.post.com/postit.cgi
```

If the content-type is not specified, curl will try to guess from the file extension (it only knows a few), or use the previously specified type (from an earlier file if several files are specified in a list) or else it will use the default type 'application/octet-stream'.

Emulate a fill-in form with `-F`. Let's say you fill in three fields in a form. One field is a file name which to post, one field is your name and one field is a file description. We want to post the file we have written named "cooltext.txt". To let curl do the posting of this data instead of your favourite browser, you have to read the HTML source of the form page and find the names of the input fields. In our example, the input field names are 'file', 'yourname' and 'filedescription'.

```
curl -F "file=@cooltext.txt" -F "yourname=Daniel"
  -F "filedescription=Cool text file with cool text inside"
  http://www.post.com/postit.cgi
```

To send two files in one post you can do it in two ways:

Send multiple files in a single "field" with a single field name:

```
curl -F "pictures=@dog.gif,cat.gif" $URL
```

Send two fields with two field names

```
curl -F "docpicture=@dog.gif" -F "catpicture=@cat.gif" $URL
```

To send a field value literally without interpreting a leading `@` or `<`, or an embedded `;type=`, use `--form-string` instead of `-F`. This is recommended when the value is obtained from a user or some other unpredictable source. Under these circumstances, using `-F` instead of `--form-string` could allow a user to trick curl into uploading a file.

Referrer

An HTTP request has the option to include information about which address referred it to the actual page. Curl allows you to specify the referrer to be used on the command line. It is especially useful to fool or trick stupid servers or CGI scripts that rely on that information being available or contain certain data.

```
curl -e www.coolsite.com http://www.showme.com/
```

User Agent

An HTTP request has the option to include information about the browser that generated the request. Curl allows it to be specified on the command line. It is especially useful to fool or trick stupid servers or CGI scripts that only accept certain browsers.

Example:

```
curl -A 'Mozilla/3.0 (Win95; I)' http://www.nationsbank.com/
```

Other common strings:

- `Mozilla/3.0 (Win95; I)` - Netscape Version 3 for Windows 95
- `Mozilla/3.04 (Win95; U)` - Netscape Version 3 for Windows 95
- `Mozilla/2.02 (OS/2; U)` - Netscape Version 2 for OS/2
- `Mozilla/4.04 [en] (X11; U; AIX 4.2; Nav)` - Netscape for AIX
- `Mozilla/4.05 [en] (X11; U; Linux 2.0.32 i586)` - Netscape for Linux

Note that Internet Explorer tries hard to be compatible in every way:

- `Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)` - MSIE for W95

Mozilla is not the only possible User-Agent name:

- `Konqueror/1.0` - KDE File Manager desktop client
- `Lynx/2.7.1 libwww-FM/2.14` - Lynx command line browser

Cookies

Cookies are generally used by web servers to keep state information at the client's side. The server sets cookies by sending a response line in the headers that looks like `Set-Cookie: <data>` where the data part then typically contains a set of `NAME=VALUE` pairs (separated by semicolons `;` like `NAME1=VALUE1; NAME2=VALUE2;`). The server can also specify for what path the "cookie" should be used for (by specifying `path=value`), when the cookie should expire (`expire=DATE`), for what domain to use it (`domain=NAME`) and if it should be used on secure connections only (`secure`).

If you've received a page from a server that contains a header like:

```
Set-Cookie: sessionid=boo123; path="/foo";
```

it means the server wants that first pair passed on when we get anything in a path beginning with "/foo".

Example, get a page that wants my name passed in a cookie:

```
curl -b "name=Daniel" www.sillypage.com
```

Curl also has the ability to use previously received cookies in following sessions. If you get cookies from a server and store them in a file in a manner similar to:

```
curl --dump-header headers www.example.com
```

... you can then in a second connect to that (or another) site, use the cookies from the 'headers' file like:

```
curl -b headers www.example.com
```

While saving headers to a file is a working way to store cookies, it is however error-prone and not the preferred way to do this. Instead, make curl save the incoming cookies using the well-known netscape cookie format like this:

```
curl -c cookies.txt www.example.com
```

Note that by specifying `-b` you enable the "cookie awareness" and with `-L` you can make curl follow a location: (which often is used in combination with cookies). So that if a site sends cookies and a location, you can use a non-existing file to trigger the cookie awareness like:

```
curl -L -b empty.txt www.example.com
```

The file to read cookies from must be formatted using plain HTTP headers OR as netscape's cookie file. Curl will determine what kind it is based on the file contents. In the above command, curl will parse the header and store the cookies received from www.example.com. curl will send to the server the stored cookies which match the request as it follows the location. The file "empty.txt" may be a nonexistent file.

To read and write cookies from a netscape cookie file, you can set both `-b` and `-c` to use the same file:

```
curl -b cookies.txt -c cookies.txt www.example.com
```

Progress Meter

The progress meter exists to show a user that something actually is happening. The different fields in the output have the following meaning:

```
  % Total    % Received % Xferd  Average Speed          Time             Curr.
                                 Dload  Upload Total    Current  Left    Speed
  0  151M    0 38608    0     0   9406      0  4:41:43  0:00:04  4:41:39  9287
```

From left-to-right:

- % - percentage completed of the whole transfer
- Total - total size of the whole expected transfer
- % - percentage completed of the download
- Received - currently downloaded amount of bytes
- % - percentage completed of the upload
- Xferd - currently uploaded amount of bytes
- Average Speed Dload - the average transfer speed of the download
- Average Speed Upload - the average transfer speed of the upload
- Time Total - expected time to complete the operation
- Time Current - time passed since the invoke
- Time Left - expected time left to completion
- Curr.Speed - the average transfer speed the last 5 seconds (the first 5 seconds of a transfer is based on less time of course.)

The `-#` option will display a totally different progress bar that doesn't need much explanation!

Speed Limit

Curl allows the user to set the transfer speed conditions that must be met to let the transfer keep going. By using the switch `-y` and `-Y` you can make curl abort transfers if the transfer speed is below the specified lowest limit for a specified time.

To have curl abort the download if the speed is slower than 3000 bytes per second for 1 minute, run:

```
curl -Y 3000 -y 60 www.far-away-site.com
```

This can very well be used in combination with the overall time limit, so that the above operation must be completed in whole within 30 minutes:

```
curl -m 1800 -Y 3000 -y 60 www.far-away-site.com
```

Forcing curl not to transfer data faster than a given rate is also possible, which might be useful if you're using a limited bandwidth connection and you don't want your transfer to use all of it (sometimes referred to as "bandwidth throttle").

Make curl transfer data no faster than 10 kilobytes per second:

```
curl --limit-rate 10K www.far-away-site.com
```

or

```
curl --limit-rate 10240 www.far-away-site.com
```

Or prevent curl from uploading data faster than 1 megabyte per second:

```
curl -T upload --limit-rate 1M ftp://uploadshereplease.com
```

When using the `--limit-rate` option, the transfer rate is regulated on a per-second basis, which will cause the total transfer speed to become lower than the given number. Sometimes of course substantially lower, if your transfer stalls during periods.

Config File

Curl automatically tries to read the `.curlrc` file (or `_curlrc` file on Microsoft Windows systems) from the user's home dir on startup.

The config file could be made up with normal command line switches, but you can also specify the long options without the dashes to make it more readable. You can separate the options and the parameter with spaces, or with = or `:`. Comments can be used within the file. If the first letter on a line is a #-symbol the rest of the line is treated as a comment.

If you want the parameter to contain spaces, you must enclose the entire parameter within double quotes (`"`). Within those quotes, you specify a quote as `\"`.

NOTE: You must specify options and their arguments on the same line.

Example, set default time out and proxy in a config file:

```
# We want a 30 minute timeout:
-m 1800
#. .. and we use a proxy for all accesses:
proxy = proxy.our.domain.com:8080
```

Whitespaces ARE significant at the end of lines, but all whitespace leading up to the first characters of each line are ignored.

Prevent curl from reading the default file by using -q as the first command line parameter, like:

```
curl -q www.thatsite.com
```

Force curl to get and display a local help page in case it is invoked without URL by making a config file similar to:

```
# default url to get
url = "http://help.with.curl.com/curlhelp.html"
```

You can specify another config file to be read by using the `-K`/`--config` flag. If you set config file name to `-` it'll read the config from stdin, which can be handy if you want to hide options from being visible in process tables etc:

```
echo "user = user:passwd" | curl -K - http://that.secret.site.com
```

Extra Headers

When using curl in your own very special programs, you may end up needing to pass on your own custom headers when getting a web page. You can do this by using the `-H` flag.

Example, send the header `X-you-and-me: yes` to the server when getting a page:

```
curl -H "X-you-and-me: yes" www.love.com
```

This can also be useful in case you want curl to send a different text in a header than it normally does. The `-H` header you specify then replaces the header curl would normally send. If you replace an internal header with an empty one, you prevent that header from being sent. To prevent the `Host:` header from being used:

```
curl -H "Host:" www.server.com
```

FTP and Path Names

Do note that when getting files with a `ftp://` URL, the given path is relative the directory you enter. To get the file `README` from your home directory at your ftp site, do:

```
curl ftp://user:passwd@my.site.com/README
```

But if you want the README file from the root directory of that very same site, you need to specify the absolute file name:

```
curl ftp://user:passwd@my.site.com//README
```

(I.e with an extra slash in front of the file name.)

SFTP and SCP and Path Names

With sftp: and scp: URLs, the path name given is the absolute name on the server. To access a file relative to the remote user's home directory, prefix the file with `/~/` , such as:

```
curl -u $USER sftp://home.example.com/~/.bashrc
```

FTP and Firewalls

The FTP protocol requires one of the involved parties to open a second connection as soon as data is about to get transferred. There are two ways to do this.

The default way for curl is to issue the PASV command which causes the server to open another port and await another connection performed by the client. This is good if the client is behind a firewall that doesn't allow incoming connections.

```
curl ftp.download.com
```

If the server, for example, is behind a firewall that doesn't allow connections on ports other than 21 (or if it just doesn't support the `PASV` command), the other way to do it is to use the `PORT` command and instruct the server to connect to the client on the given IP number and port (as parameters to the PORT command).

The `-P` flag to curl supports a few different options. Your machine may have several IP-addresses and/or network interfaces and curl allows you to select which of them to use. Default address can also be used:

```
curl -P - ftp.download.com
```

Download with `PORT` but use the IP address of our `le0` interface (this does not work on windows):

```
curl -P le0 ftp.download.com
```

Download with `PORT` but use 192.168.0.10 as our IP address to use:

```
curl -P 192.168.0.10 ftp.download.com
```

Network Interface

Get a web page from a server using a specified port for the interface:

```
curl --interface eth0:1 http://www.example.com/
```

or

```
curl --interface 192.168.1.10 http://www.example.com/
```

HTTPS

Secure HTTP requires a TLS library to be installed and used when curl is built. If that is done, curl is capable of retrieving and posting documents using the HTTPS protocol.

Example:

```
curl https://www.secure-site.com
```

curl is also capable of using client certificates to get/post files from sites that require valid certificates. The only drawback is that the certificate needs to be in PEM-format. PEM is a standard and open format to store certificates with, but it is not used by the most commonly used browsers. If you want curl to use the certificates you use with your (favourite) browser, you may need to download/compile a converter that can convert your browser's formatted certificates to PEM formatted ones.

Example on how to automatically retrieve a document using a certificate with a personal password:

```
curl -E /path/to/cert.pem:password https://secure.site.com/
```

If you neglect to specify the password on the command line, you will be prompted for the correct password before any data can be received.

Many older HTTPS servers have problems with specific SSL or TLS versions, which newer versions of OpenSSL etc use, therefore it is sometimes useful to specify what SSL-version curl should use. Use -3, -2 or -1 to specify that exact SSL version to use (for SSLv3, SSLv2 or TLSv1 respectively):

```
curl -2 https://secure.site.com/
```

Otherwise, curl will attempt to use a sensible TLS default version.

Resuming File Transfers

To continue a file transfer where it was previously aborted, curl supports resume on HTTP(S) downloads as well as FTP uploads and downloads.

Continue downloading a document:

```
curl -C - -o file ftp://ftp.server.com/path/file
```

Continue uploading a document:

```
curl -C - -T file ftp://ftp.server.com/path/file
```

Continue downloading a document from a web server

```
curl -C - -o file http://www.server.com/
```

Time Conditions

HTTP allows a client to specify a time condition for the document it requests. It is `If-Modified-Since` or `If-Unmodified-Since`. curl allows you to specify them with the `-z`/`--time-cond` flag.

For example, you can easily make a download that only gets performed if the remote file is newer than a local copy. It would be made like:

```
curl -z local.html http://remote.server.com/remote.html
```

Or you can download a file only if the local file is newer than the remote one. Do this by prepending the date string with a `-`, as in:

```
curl -z -local.html http://remote.server.com/remote.html
```

You can specify a "free text" date as condition. Tell curl to only download the file if it was updated since January 12, 2012:

```
curl -z "Jan 12 2012" http://remote.server.com/remote.html
```

Curl will then accept a wide range of date formats. You always make the date check the other way around by prepending it with a dash (`-`).

DICT

For fun try

```
curl dict://dict.org/m:curl
curl dict://dict.org/d:heisenbug:jargon
curl dict://dict.org/d:daniel:gcide
```

Aliases for 'm' are 'match' and 'find', and aliases for 'd' are 'define' and 'lookup'. For example,

```
curl dict://dict.org/find:curl
```

Commands that break the URL description of the RFC (but not the DICT protocol) are

```
curl dict://dict.org/show:db
curl dict://dict.org/show:strat
```

Authentication support is still missing

LDAP

If you have installed the OpenLDAP library, curl can take advantage of it and offer `ldap://` support. On Windows, curl will use WinLDAP from Platform SDK by default.

Default protocol version used by curl is LDAPv3. LDAPv2 will be used as fallback mechanism in case if LDAPv3 will fail to connect.

LDAP is a complex thing and writing an LDAP query is not an easy task. I do advise you to dig up the syntax description for that elsewhere. One such place might be: RFC 2255, The LDAP URL Format

To show you an example, this is how I can get all people from my local LDAP server that has a certain sub-domain in their email address:

```
curl -B "ldap://ldap.frontec.se/o=frontec??sub?mail=*sth.frontec.se"
```

If I want the same info in HTML format, I can get it by not using the `-B` (enforce ASCII) flag.

You also can use authentication when accessing LDAP catalog:

```
curl -u user:passwd "ldap://ldap.frontec.se/o=frontec??sub?mail=*"
curl "ldap://user:passwd@ldap.frontec.se/o=frontec??sub?mail=*"
```

By default, if user and password provided, OpenLDAP/WinLDAP will use basic authentication. On Windows you can control this behavior by providing one of `--basic`, `--ntlm` or `--digest` option in curl command line

```
curl --ntlm "ldap://user:passwd@ldap.frontec.se/o=frontec??sub?mail=*"
```

On Windows, if no user/password specified, auto-negotiation mechanism will be used with current logon credentials (SSPI/SPNEGO).

Environment Variables

Curl reads and understands the following environment variables:

```
http_proxy, HTTPS_PROXY, FTP_PROXY
```

They should be set for protocol-specific proxies. General proxy should be set with

```
ALL_PROXY
```

A comma-separated list of host names that shouldn't go through any proxy is set in (only an asterisk, * matches all hosts)

```
NO_PROXY
```

If the host name matches one of these strings, or the host is within the domain of one of these strings, transactions with that node will not be proxied. When a domain is used, it needs to start with a period. A user can specify that both www.example.com and foo.example.com should not use a proxy by setting `NO_PROXY` to `.example.com`. By including the full name you can exclude specific host names, so to make `www.example.com` not use a proxy but still have `foo.example.com` do it, set `NO_PROXY` to `www.example.com`.

The usage of the `-x/--proxy` flag overrides the environment variables.

Netrc

Unix introduced the `.netrc` concept a long time ago. It is a way for a user to specify name and password for commonly visited FTP sites in a file so that you don't have to type them in each time you visit those sites. You realize this is a big security risk if someone else gets hold of your passwords, so therefore most unix programs won't read this file unless it is only readable by yourself (curl doesn't care though).

Curl supports `.netrc` files if told to (using the `-n/--netrc` and `--netrc-optional` options). This is not restricted to just FTP, so curl can use it for all protocols where authentication is used.

A very simple `.netrc` file could look something like:

```
machine curl.se login iamdaniel password mysecret
```

Custom Output

To better allow script programmers to get to know about the progress of curl, the `-w/--write-out` option was introduced. Using this, you can specify what information from the previous transfer you

want to extract.

To display the amount of bytes downloaded together with some text and an ending newline:

```
curl -w 'We downloaded %{size_download} bytes\n' www.download.com
```

Kerberos FTP Transfer

Curl supports kerberos4 and kerberos5/GSSAPI for FTP transfers. You need the kerberos package installed and used at curl build time for it to be available.

First, get the krb-ticket the normal way, like with the kinit/kauth tool. Then use curl in way similar to:

```
curl --krb private ftp://krb4site.com -u username:fakepwd
```

There's no use for a password on the `-u` switch, but a blank one will make curl ask for one and you already entered the real password to kinit/kauth.

TELNET

The curl telnet support is basic and very easy to use. Curl passes all data passed to it on stdin to the remote server. Connect to a remote telnet server using a command line similar to:

```
curl telnet://remote.server.com
```

And enter the data to pass to the server on stdin. The result will be sent to stdout or to the file you specify with `-o`.

You might want the `-N`/`--no-buffer` option to switch off the buffered output for slow connections or similar.

Pass options to the telnet protocol negotiation, by using the `-t` option. To tell the server we use a vt100 terminal, try something like:

```
curl -tTTYPE=vt100 telnet://remote.server.com
```

Other interesting options for it `-t` include:

- `XDISPLOC=<X display>` Sets the X display location.
- `NEW_ENV=<var,val>` Sets an environment variable.

NOTE: The telnet protocol does not specify any way to login with a specified user and password so curl can't do that automatically. To do that, you need to track when the login prompt is received and send the username and password accordingly.

Persistent Connections

Specifying multiple files on a single command line will make curl transfer all of them, one after the other in the specified order.

libcurl will attempt to use persistent connections for the transfers so that the second transfer to the same host can use the same connection that was already initiated and was left open in the previous transfer. This greatly decreases connection time for all but the first transfer and it makes a far better use of the network.

Note that curl cannot use persistent connections for transfers that are used in subsequence curl invokes. Try to stuff as many URLs as possible on the same command line if they are using the same host, as that'll make the transfers faster. If you use an HTTP proxy for file transfers, practically all transfers will be persistent.

Multiple Transfers With A Single Command Line

As is mentioned above, you can download multiple files with one command line by simply adding more URLs. If you want those to get saved to a local file instead of just printed to stdout, you need to add one save option for each URL you specify. Note that this also goes for the `-O` option (but not `--remote-name-all`).

For example: get two files and use `-O` for the first and a custom file name for the second:

```
curl -O http://url.com/file.txt ftp://ftp.com/moo.exe -o moo.jpg
```

You can also upload multiple files in a similar fashion:

```
curl -T local1 ftp://ftp.com/moo.exe -T local2 ftp://ftp.com/moo2.txt
```

IPv6

curl will connect to a server with IPv6 when a host lookup returns an IPv6 address and fall back to IPv4 if the connection fails. The `--ipv4` and `--ipv6` options can specify which address to use when both are available. IPv6 addresses can also be specified directly in URLs using the syntax:

```
http://[2001:1890:1112:1::20]/overview.html
```

When this style is used, the `-g` option must be given to stop curl from interpreting the square brackets as special globbing characters. Link local and site local addresses including a scope identifier, such as `fe80::1234%1`, may also be used, but the scope portion must be numeric or match an existing network interface on Linux and the percent character must be URL escaped. The previous example in an SFTP URL might look like:

```
sftp://[fe80::1234%251]/
```

IPv6 addresses provided other than in URLs (e.g. to the `--proxy`, `--interface` or `--ftp-port` options) should not be URL encoded.

Mailing Lists

For your convenience, we have several open mailing lists to discuss curl, its development and things relevant to this. Get all info at https://curl.se/mail/.

Please direct curl questions, feature requests and trouble reports to one of these mailing lists instead of mailing any individual.

Available lists include:

curl-users

Users of the command line tool. How to use it, what doesn't work, new features, related tools, questions, news, installations, compilations, running, porting etc.

curl-library

Developers using or developing libcurl. Bugs, extensions, improvements.

curl-announce

Low-traffic. Only receives announcements of new public versions. At worst, that makes something like one or two mails per month, but usually only one mail every second month.

curl-and-php

Using the curl functions in PHP. Everything curl with a PHP angle. Or PHP with a curl angle.

curl-and-python

Python hackers using curl with or without the python binding pycurl.