

mastering the curl command line



Daniel Stenberg

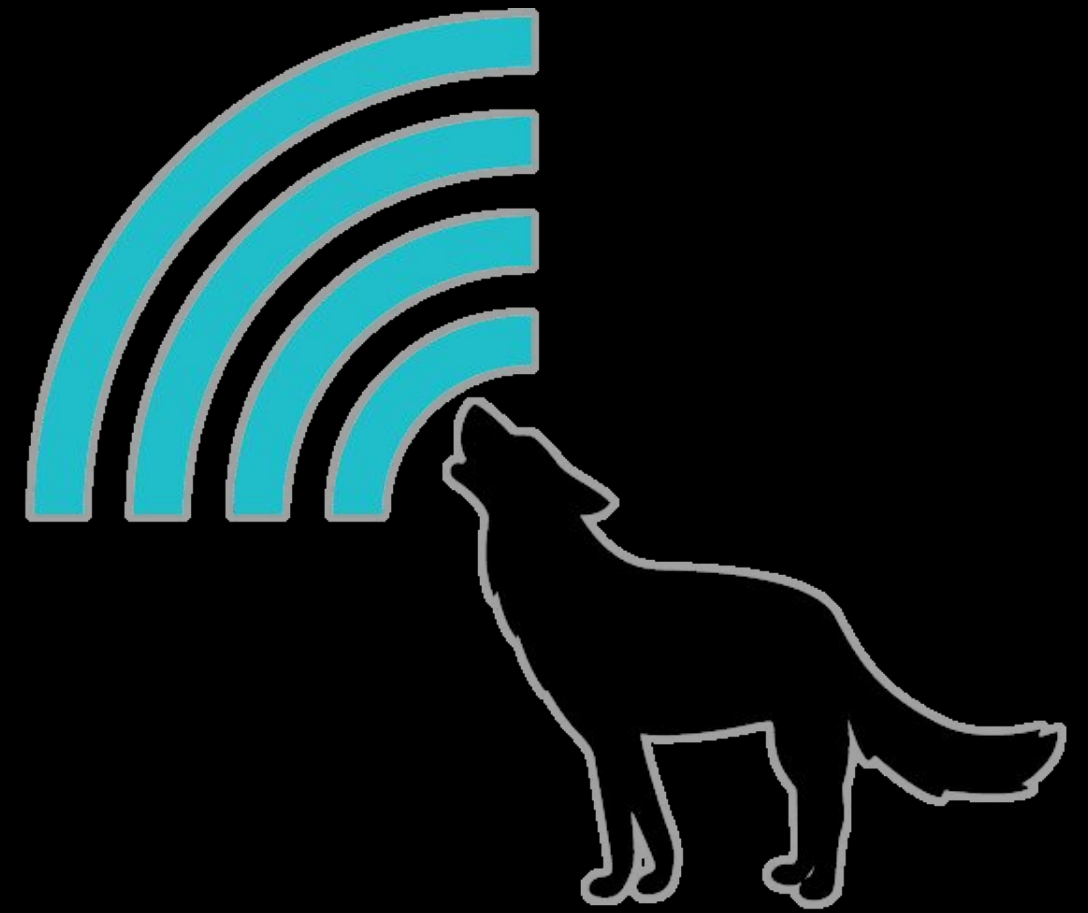
@bagder
@mastodon.social



wolfSSL



<https://daniel.haxx.se>



WolfSSL

curl.se/support.html

Setup - August 31 2023

Live-streamed

Expected to last multiple hours

Recorded

Pause when you want to - hydrate!

Lots of material never previously presented

Terminal overlays on/off



Just ask!

Live questions from non-visible chat

mastering the curl command line

- ❑ The project
- ❑ Internet transfers
- ❑ Command line
- ❑ curl basics
- ❑ TLS
- ❑ Proxies
- ❑ HTTP
- ❑ FTP
- ❑ Future



The project

November 1996: `httpget`

August 1997: `urlget`

March 1998: `curl`

August 2000: `libcurl`



Name

client for URLs

“see URL”

curl URL Request Library

Capable Ubiquitous Reliable Libre



Main products

curl - **command line tool** for client-side internet transfers with URLs

libcurl - **library** for client-side internet transfers with URLs

- ★ Always and only **client-side**
- ★ An **internet transfer**: upload or download or both
- ★ Endpoint described with a **URL**

Open Source

Everything in the curl project is open source

Every discussion and decision are held and done in the open

Open source means everyone can reshare and change

curl is (essentially) MIT licensed



COPYRIGHT AND PERMISSION NOTICE

Copyright © 1996 - 2023, Daniel Stenberg, <daniel@haxx.se>, and many contributors, see the THANKS file.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Development

curl is developed by “everyone”

only Daniel works on curl full-time

everyone can provide their proposed changes as “pull requests”

No paperwork required

Changes are reviewed and tested thoroughly

A small team of maintainers can accept and merge changes

Releases

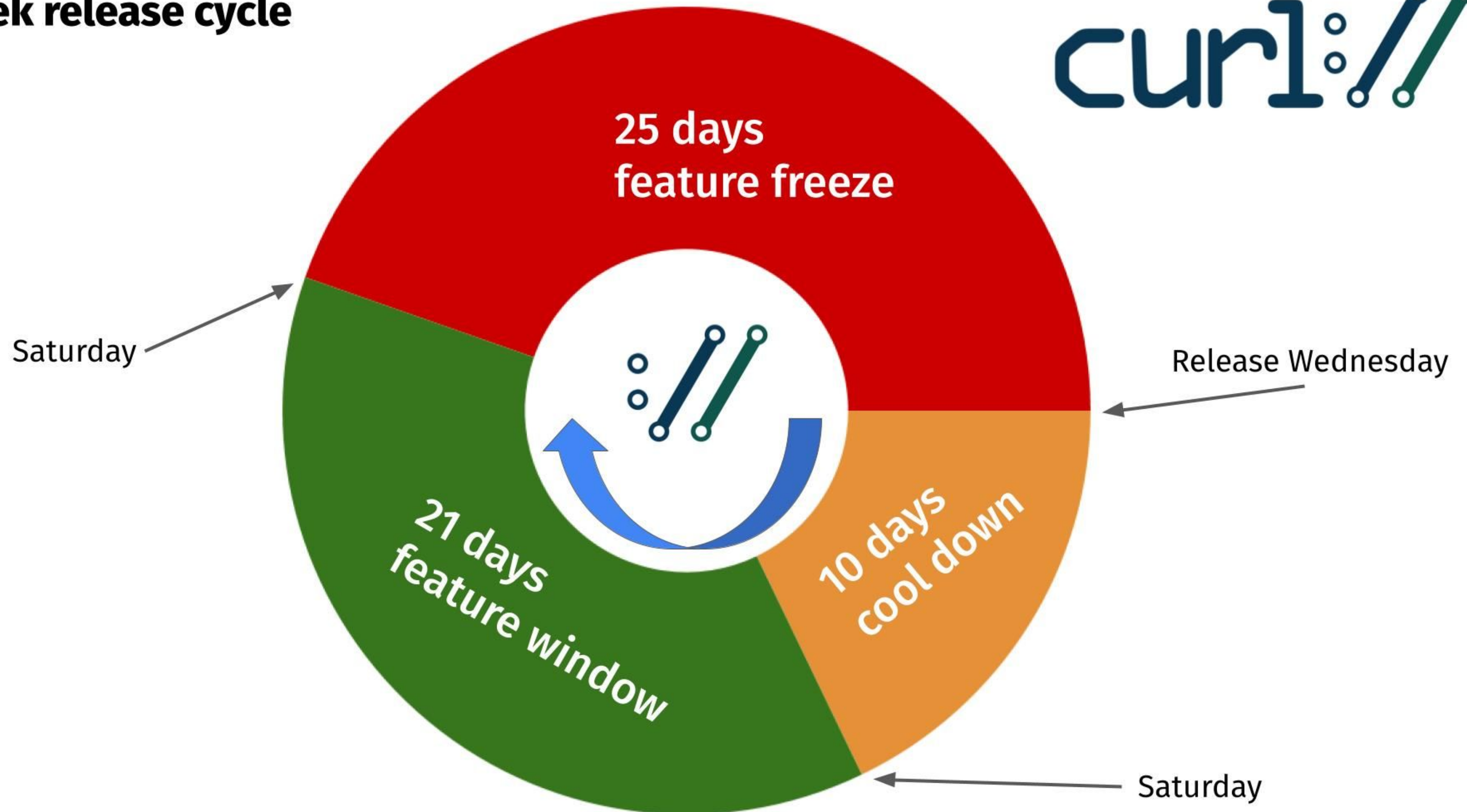
We do releases every 8 weeks (or sooner when necessary)

At 250 releases

We release what is in the master branch at the time

Releases

8 week release cycle



Round and round again until the end of time...

Releases



We **never** break existing functionality

Issues and pull requests

Problems are submitted as issues

Changes are proposed as pull requests

`https://github.com/curl/curl`



Learn more

man curl (curl --manual)

<https://curl.se/docs/manpage.html>

Everything curl

curl --help

@bagder

Everything



The complete guide to all there is
to know about the curl project

Daniel Stenberg

Asking for help

Mailing lists

command line tool questions: **curl-users**

<https://lists.haxx.se/mailman/listinfo/curl-users>

library/development/debugging questions: **curl-library**

<https://lists.haxx.se/mailman/listinfo/curl-library>

Web “forum”

<https://github.com/curl/curl/discussions>

Paying for curl help

support@wolfssl.com

wolfssl

DICT, FILE, FTP, FTPS, GOPHER, GOPHERS, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTMPS, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET, TFTP, WS and WSS

TLS certificates, HTTP POST, HTTP PUT, FTP upload, HTTP form based upload, proxies (SOCKS4, SOCKS5, HTTP and HTTPS), HTTP/2, HTTP/3, cookies, user+password authentication (Basic, Plain, Digest, CRAM-MD5, SCRAM-SHA, NTLM, Negotiate, Kerberos, Bearer tokens and AWS Sigv4), file transfer resume, proxy tunneling, HSTS, Alt-Svc, unix domain sockets, HTTP compression (gzip, brotli and zstd), etags, parallel transfers, DNS-over-HTTPS and much more

Runs everywhere

Linux

Windows (by default)

macOS (by default)

FreeBSD


OpenBSD

VMS

...

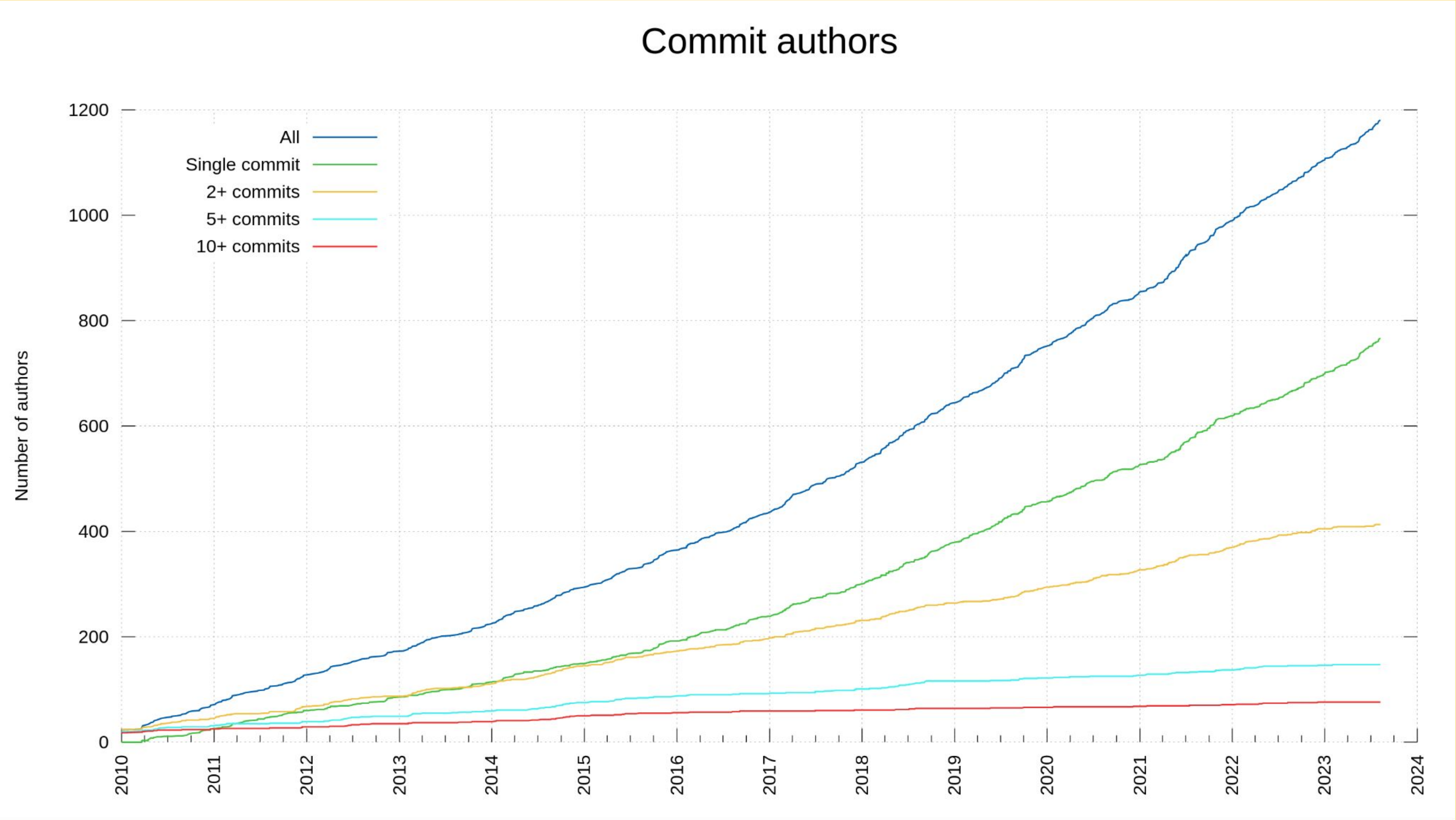


92 operating systems

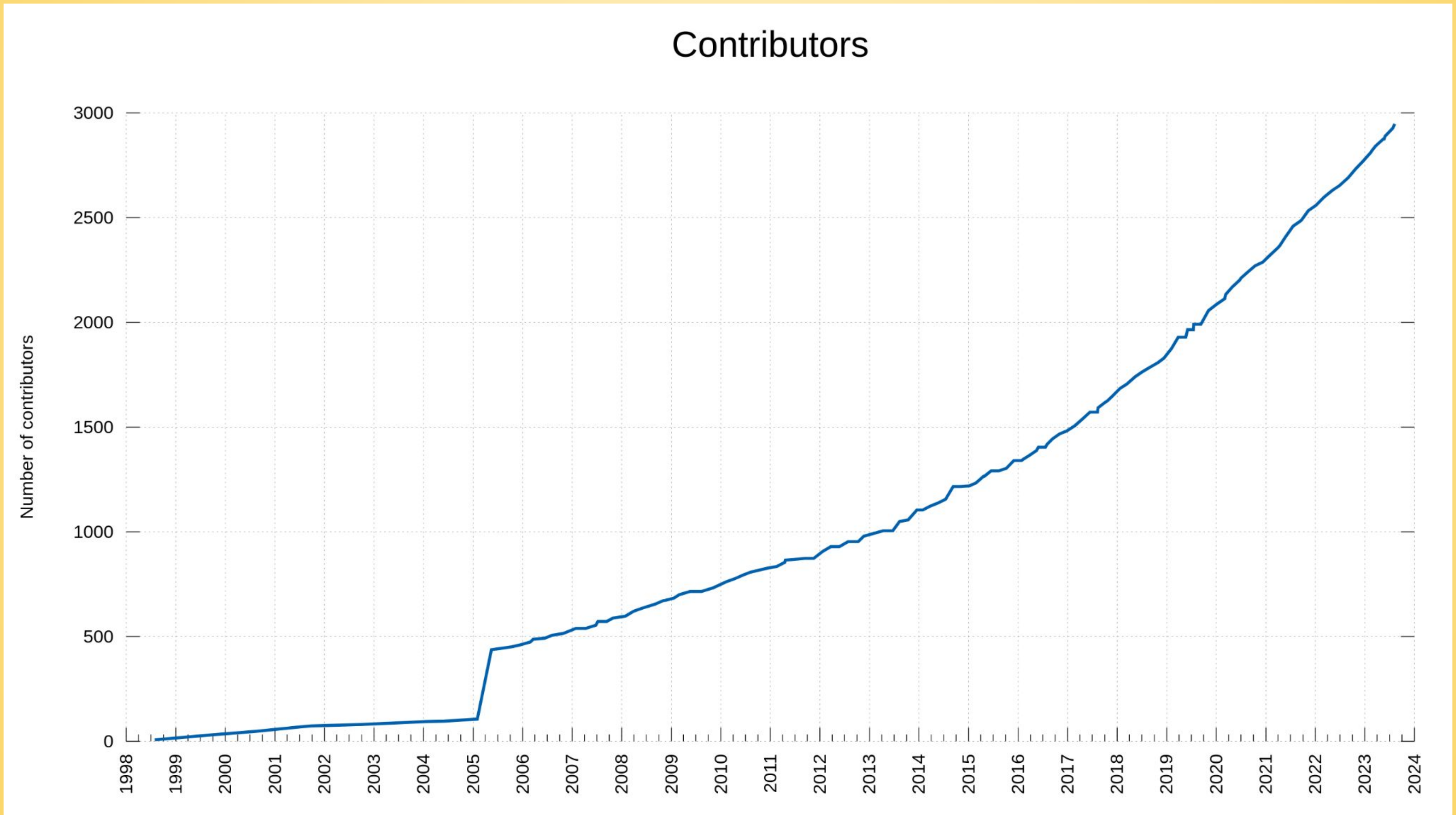
| | | | | | | | | |
|------------|----------------------|-----------|---------------------|----------------|-------------|---------------|---|-----------------|
| AIX | AmigaOS | Android | Aros | Atari FreeMiNT | BeOS | Blackberry 10 | Blackberry Tablet OS | Cell OS |
| Chrome OS | Cisco IOS | Cygwin | DG/UX | DragonFly BSD | DR DOS | eCOS | FreeDOS | FreeBSD |
| FreeRTOS | Fuchsia | Garmin OS | Genode | Haiku | HardenedBSD | HP-UX | Hurd | Illumos |
| Integrity | iOS | ipadOS | IRIX | Linux | Lua RTOS | Mac OS 9 | macOS | Mbed |
| Micrium | MINIX | MorphOS | MPE/iX | MS-DOS | NCR MP-RAS | NetBSD | NetWare | Nintendo Switch |
| NonStop OS | NuttX | OmniOS | OpenBSD | OpenStep | Orbis OS | OS/2 | OS/400 | OS21 |
| Plan 9 | PlayStation Portable | QNX | Qubes OS | ReactOS | Redox | RISC OS | RTEMS | Sailfish OS |
| SCO Unix | Serenity | SINIX-Z | Solaris | SunOS | Syllable OS | Symbian | Tizen | TPF |
| Tru64 | tvOS | ucLinux | Ultrix | UNICOS | UnixWare | VMS |  | |
| vxWorks | watchOS | WebOS | Wii System Software | Windows | Windows CE | Xbox System | | |
| Xenix | Zephyr | z/OS | z/TPF | z/VM | z/VSE | | | |

Operating systems known to have run curl

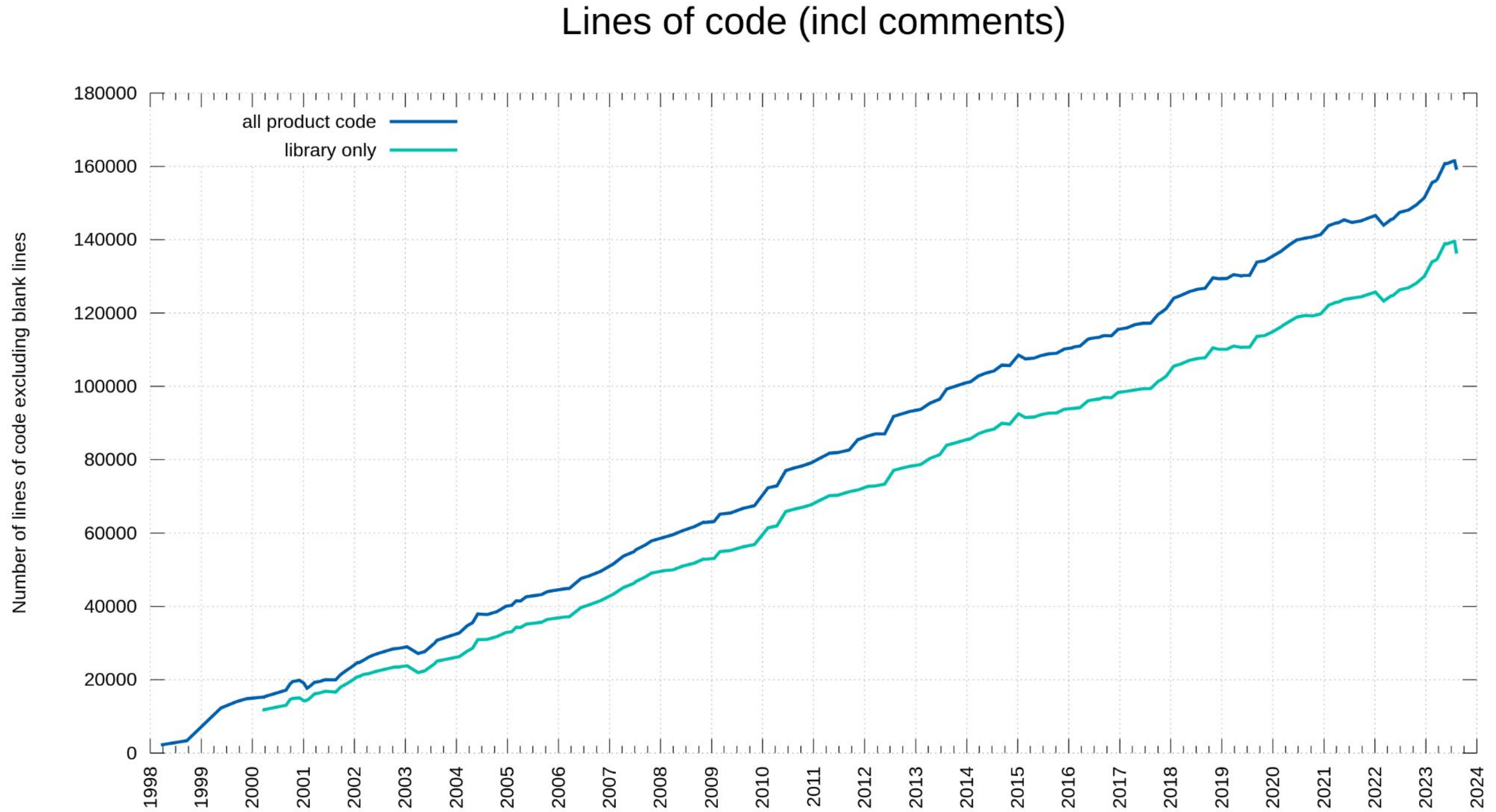
Commit authors



Contributors



Lines of code



**>20,000,000,000
installations**

“No big deal”



CertifiedToKnowItAll 2 days ago · *edited 2 days ago*

I used curl to test my handwritten HTTP 2.0 server. That includes sending chunked, compression support, SSL with the 3 basic auth methods, all the various verbs, etc. I could write curl in a 3 day weekend comfortably. Someone who doesn't know the

I think you could replace 99% of the uses of Curl (download one file via HTTPS) with like 100 lines of Python or Rust or Go. It's not critical infrastructure in the same way that OpenSSL or LLVM or WebKit are.



A library doesn't take a genius to maintain, this isn't a big deal.

Internet transfers

curl does internet transfers

A **server** is a remote machine running server software

curl acts as a **client** on the network

curl connects to a server in order to do **Internet transfers**

A stream of data **from** or **to** a server

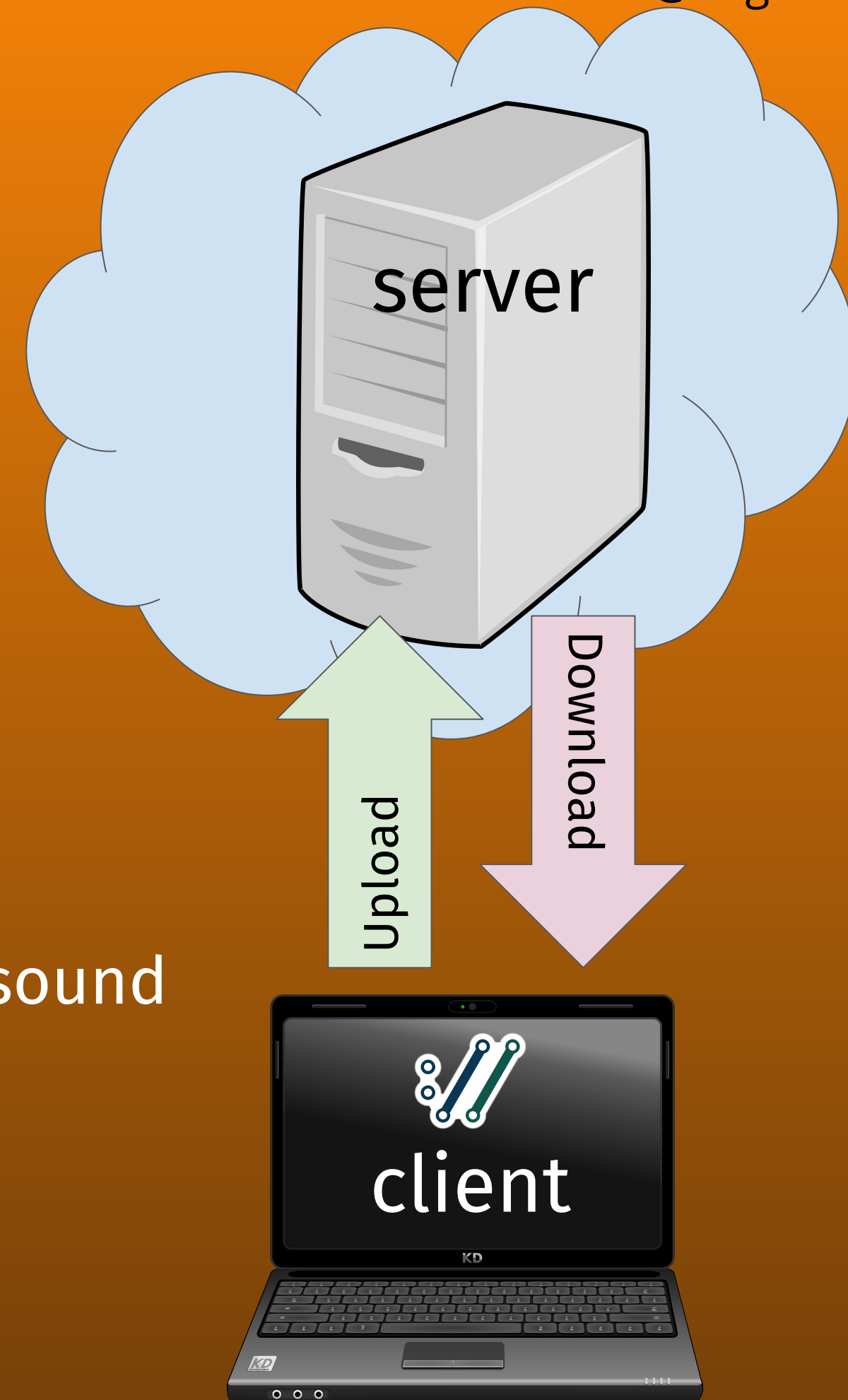
Data from a server to curl, is a **download**

Data from curl to a server, is an **upload**

The data can be anything: text, images, maps, code, film, sound

curl does not care

curl does not know





| Protocol | Download | Upload |
|-----------|----------|--------|
| HTTP(S) | ✓ | ✓ |
| GOPHER(S) | ✓ | ✗ |
| FTP(S) | ✓ | ✓ |
| TELNET | ! | ! |
| DICT | ✓ | ✗ |
| LDAP(S) | ✓ | ✗ |
| FILE | ✓ | ✓ |
| TFTP | ✓ | ✓ |
| SCP | ✓ | ✓ |
| SFTP | ✓ | ✓ |
| IMAP(S) | ✓ | ✓ |
| POP3(S) | ✓ | ✗ |
| SMTP(S) | ✗ | ✓ |
| RTSP | ✓ | ✗ |
| RTMP(S) | ✓ | ✗ |
| SMB(S) | ✓ | ✓ |
| MQTT | ✓ | ✓ |
| WS(S) | ✓ | ✓ |

authenticated vs unauthenticated protocols

Always use authenticated protocols

Authenticated means they use TLS or SSH

HTTPS, FTPS, LDAPS, IMAPS... end with S + SCP and SFTP

Never disable server verification (`--insecure`) in production

Unauthenticated transfers can be eavesdropped and tampered with

Without curl or the user knowing

Unauthenticated transfers are easily attacked - avoid!

Command line

command line options

short options: `-V`

long options: `--version`

boolean options: `--path-as-is`

options with arguments: `--output store.html`

arguments with spaces: `--write-out "received %{path-as-is}"`

negative boolean options: `--no-path-as-is`

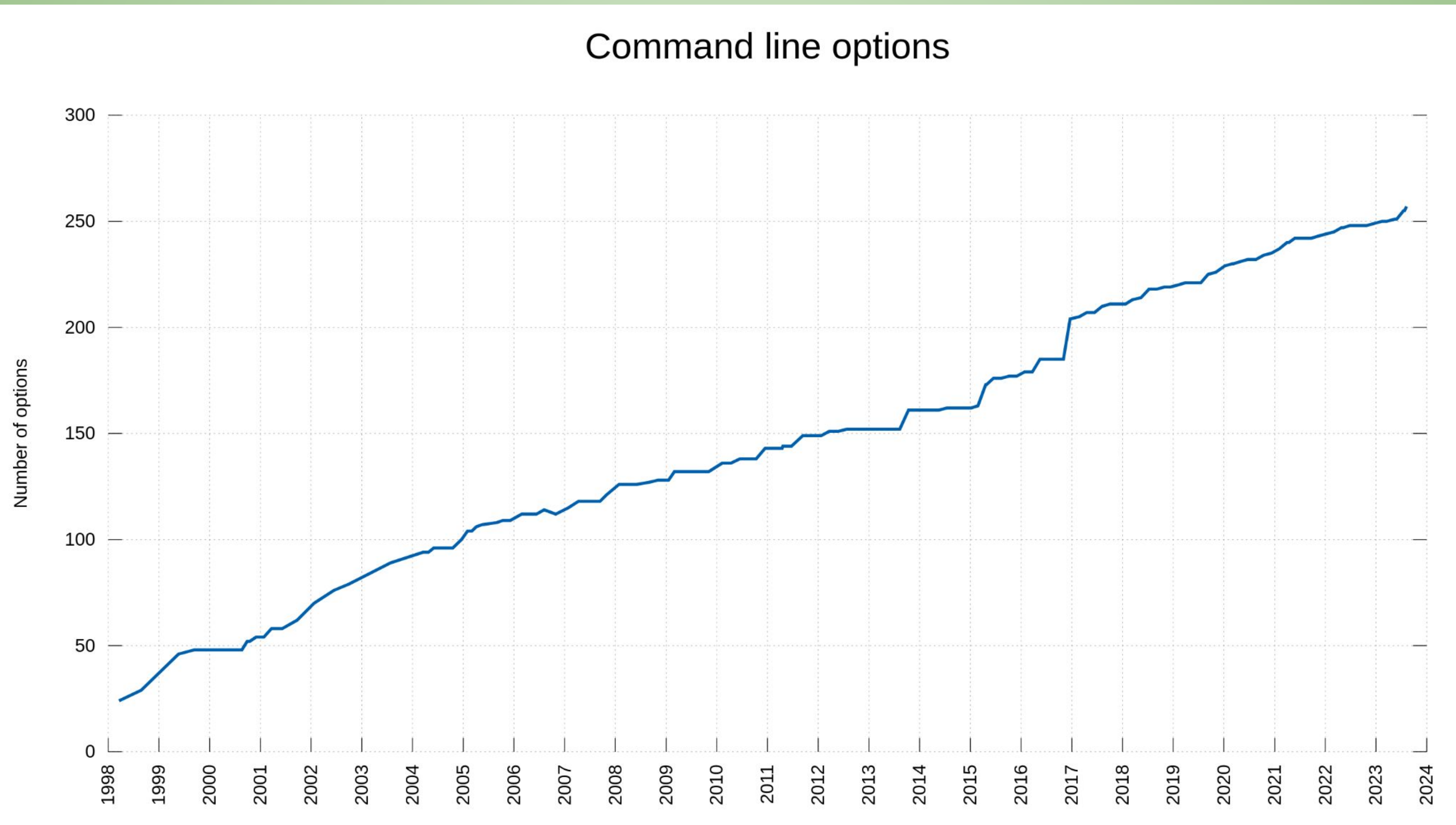
combinatorial explosion

availability depends on version

availability depends on build

availability depends on 3rd party libraries (and their versions)

command line options



URLs by RFC 3986

```
scheme://user:password@host:1234/path?query#fragment
```

RFC 3986+ really

no spaces, use %20

“schemeless” means guess

name and password - remember URL encode

hostname can be name, IDN name, IPv4 address or IPv6 address

`https://example.com/`

`http://日本語.tw`

`ftp://192.168.0.1/`

`imap://[2a04:4e42:800::347]/`

URLs

Anything not an option is a URL

URL port numbers

A port number is from 0 to 65535

Each URL scheme has a default port that curl uses

Unless another is set in the URL

```
curl https://example.com:8080/
```

```
curl tftp://[fdea::1]:8080/
```


URLs and browsers

browsers, URLs and their address bars



URLs and output options

curl accepts *any* amount of URLs

every downloaded URL needs a destination - stdout or a file

```
curl -o file1 -o file2 https://example.com/file1 https://curl.se/file2
```

```
curl -o file1 https://example.com/file1 https://curl.se/file2 -o file2
```

```
curl -O https://example.com/file1 -O https://curl.se/file2
```

```
curl https://example.com/file1 https://curl.se/file2 > everything
```

--remote-name-all automatically sets -O for all URLs

query

```
scheme://user:password@host:1234/path?query#fragment
```

Queries are often name=value pairs separated by amperands (&)

```
name=daniel & tool=curl & age=old
```

Add query parts to the URL with `--url-query [content]`

content is (for example) “name=value”

“value” gets URL encoded to keep the URL fine

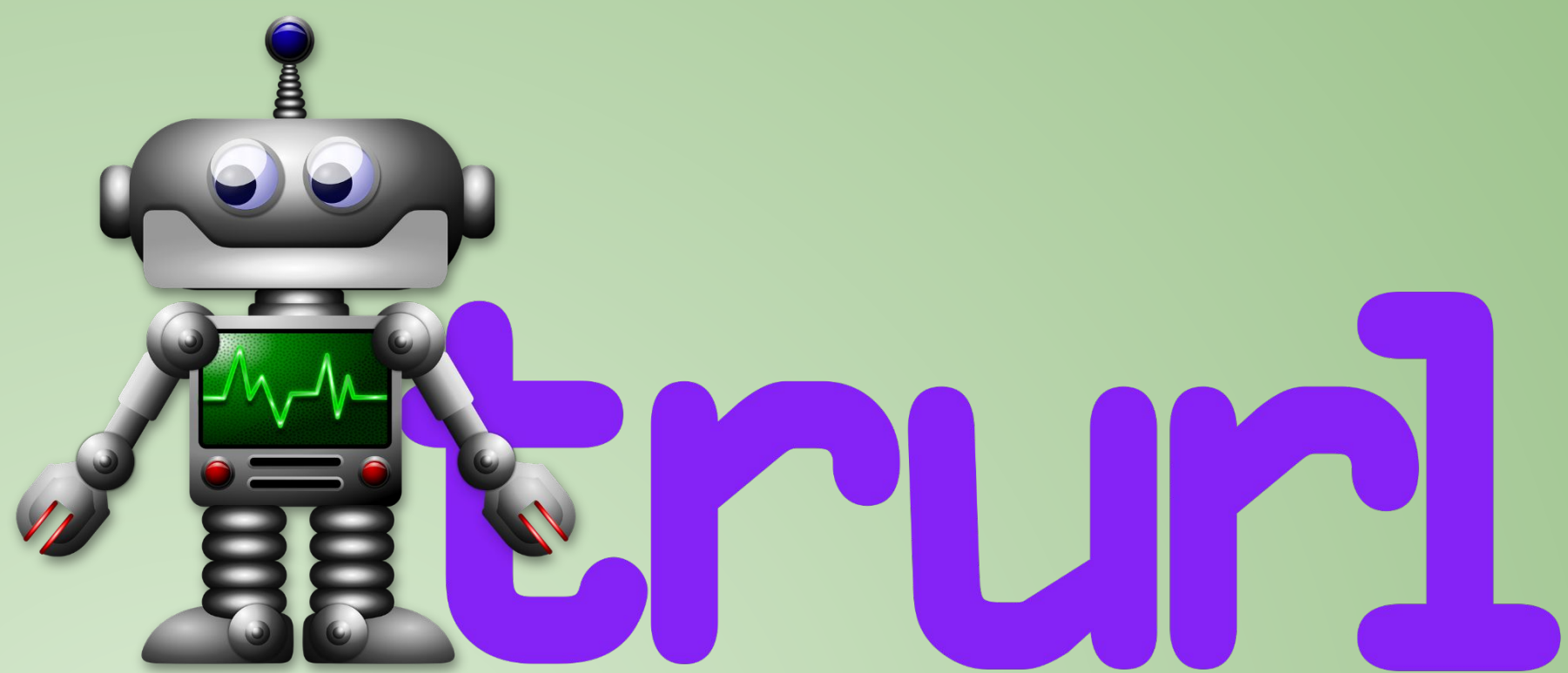
`name@file` reads the content from file before encoding it

... and more

```
curl https://example.com --url-query "name=Daniel Stenberg"
```


trurl

created in the spring of 2023
parses and manipulates URLs
companion tool to curl



```
$ trurl --url https://curl.se --set host=example.com  
$ trurl --url https://curl.se/we/are.html --redirect here.html  
$ trurl --url https://curl.se/we/ ../are.html --set port=8080  
$ trurl --url "https://curl.se?name=hello" --append query=search=string  
$ trurl "https://fake.host/search?q=answers&user=me#frag" --json  
$ trurl "https://example.com?a=home&here=now&thisthen" -g '{query:a}'
```

<https://curl.se/trurl/>

URL globbing

“globbing” = ranges and lists

[1-100]

[001-100]

[a-z]

[001-100:10]

[a-z:2]

{one,two,three}

{mon,tue,wed,thu}

```
$ curl https://{ftp,www,test}.example.com/img[1-22].jpg -o  
“hey_#2_#1.jpg”
```

Can do $9 * 10^{18}$ iterations - *per URL*

--globoff turns it off

Parallel transfers

by default, URLs are transferred serially, one by one

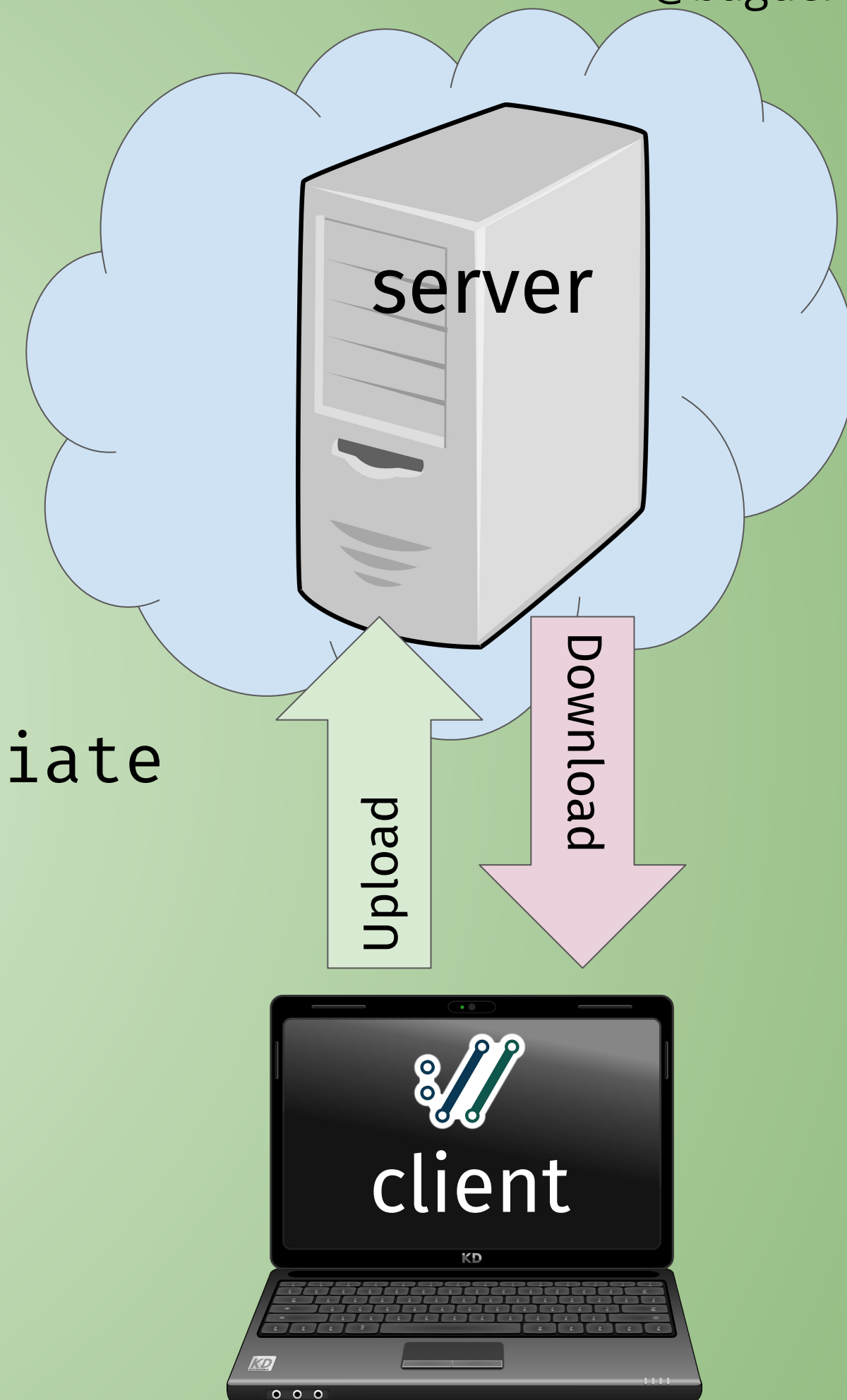
-Z (--parallel)

By default up to 50 simultaneous

Change with --parallel - max [num]

Prefer speed to multiplexing with --parallel - immediate

Works for downloads and uploads



list curl options

`--help`

`--help [category]`

`--help category`

`--help all`

`curl --manual`

config file

“command lines in a file”

one option (plus argument) per line

`$HOME/.curlrc` is used by default

`-K [file]` or `--config [file]`

can be read from stdin

can be generated (and huge)

10MB line length limit

p4ssw0rds

-u name:password

.netrc (more soon)

config files

local leakage

network leakage

debug log leakage

progress meters

Unless -s, --silent or --no-progress-meter

| % 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 |

-#, --progress-bar



Different again when doing parallel transfers

| DL% | UL% | Dled | Uled | Xfers | Live | Total | Current | Left | Speed |
|-----|-----|-------|------|-------|------|----------|---------|----------|-------|
| 12% | -- | 34.5G | 0 | 2 | 2 | --:--:-- | 0:00:09 | --:--:-- | 3903M |

--next

do everything to the left of it

then reset the state

continue on the other side

in perpetuity

```
$ curl -H "header: one" https://example.com/one  
-H "header: two" https://example.com/two
```

```
$ curl -H "header: one" https://example.com/one  
--next -H "header: two" https://example.com/two
```


curl basics

curl version

--version or -V

```
$ curl -V
curl 8.2.1 (x86_64-pc-linux-gnu) libcurl/8.2.1 OpenSSL/3.0.10 zlib/1.2.13
brotli/1.0.9 zstd/1.5.5 libidn2/2.3.4 libpsl/0.21.2 (+libidn2/2.3.3)
libssh2/1.11.0 nghttp2/1.55.1 librtmp/2.3 OpenLDAP/2.5.13
Release-Date: 2023-07-26
Protocols: dict file ftp ftps gopher gophers http https imap imaps ldap
ldaps mqtt pop3 pop3s rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: alt-svc AsynchDNS brotli GSS-API HSTS HTTP2 HTTPS-proxy IDN
IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL threadsafe
TLS-SRP UnixSockets zstd
```


verbose

--verbose or -v

-v and -vv is the same

--trace-ascii or --trace

--trace-time

--trace-ids

careful before you share debug logs with others

--write-out

outputs text, information and HTTP headers after a transfer is completed

```
curl -w "formatted string" http://example.com/
```

```
curl -w @filename http://example.com/
```

```
curl -w @- http://example.com/
```

Information from over 50 “variables”

```
"Type: %{content_type}\nCode: %{response_code}\n"
```

Show HTTP response header contents

```
"Server: %header{server}\nDate: %header{date}\n"
```


persistent connections

connections are kept “alive”

repeated transfers to the same host try to reuse the connection

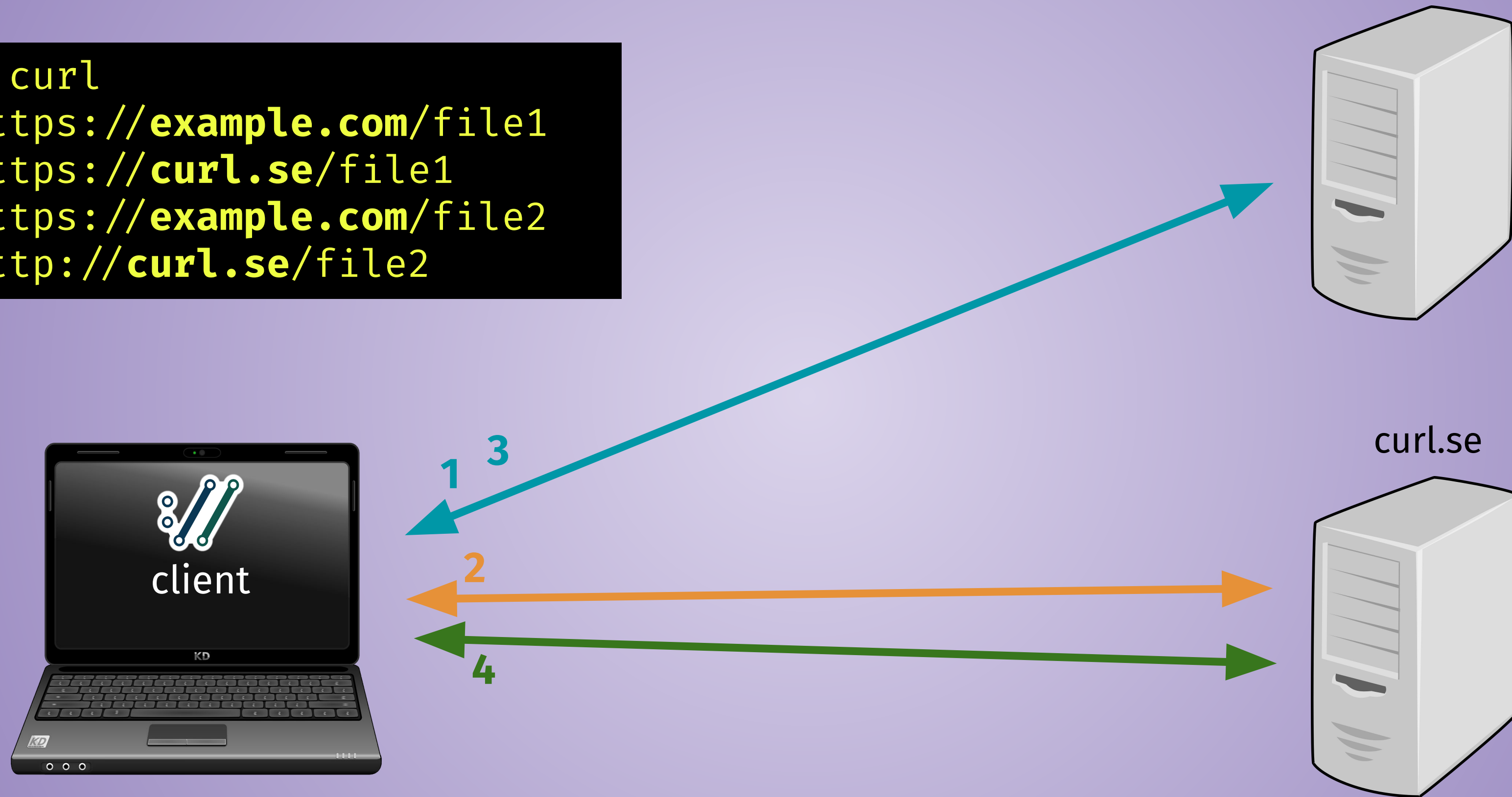
reused connections is a key to speedier transfers

reuse is done per scheme + host name + port - *not* IP address

all connections close when curl exits

persistent connections illustrated

```
$ curl  
https://example.com/file1  
https://curl.se/file1  
https://example.com/file2  
http://curl.se/file2
```



downloads

download to a file named by the URL: `-O` (`--remote-name`)

use the `Content-Disposition` name from the server:

`--remote-header-name`

danger!

shell redirect works: `curl https://curl.se > output.txt`

`curl https://curl.se https://example.com > output.txt`

maximum file size accepted: `--max-filesize <bytes>`

file size often not known ahead of time!

`--output-dir` saves the `-O` data in another directory

`--create-dirs` is useful in a combination

retry

If a **transient** error is returned when curl tries to perform a transfer

Do a few retries: `--retry [num]`

Retry for this long: `--retry - max - time <seconds>`

Wait this long between retries: `--retry - delay <seconds>`

Consider “connection refused” to be transient: `--retry - connrefused`

Consider **all** errors transient: `--retry - all-errors`

uploads

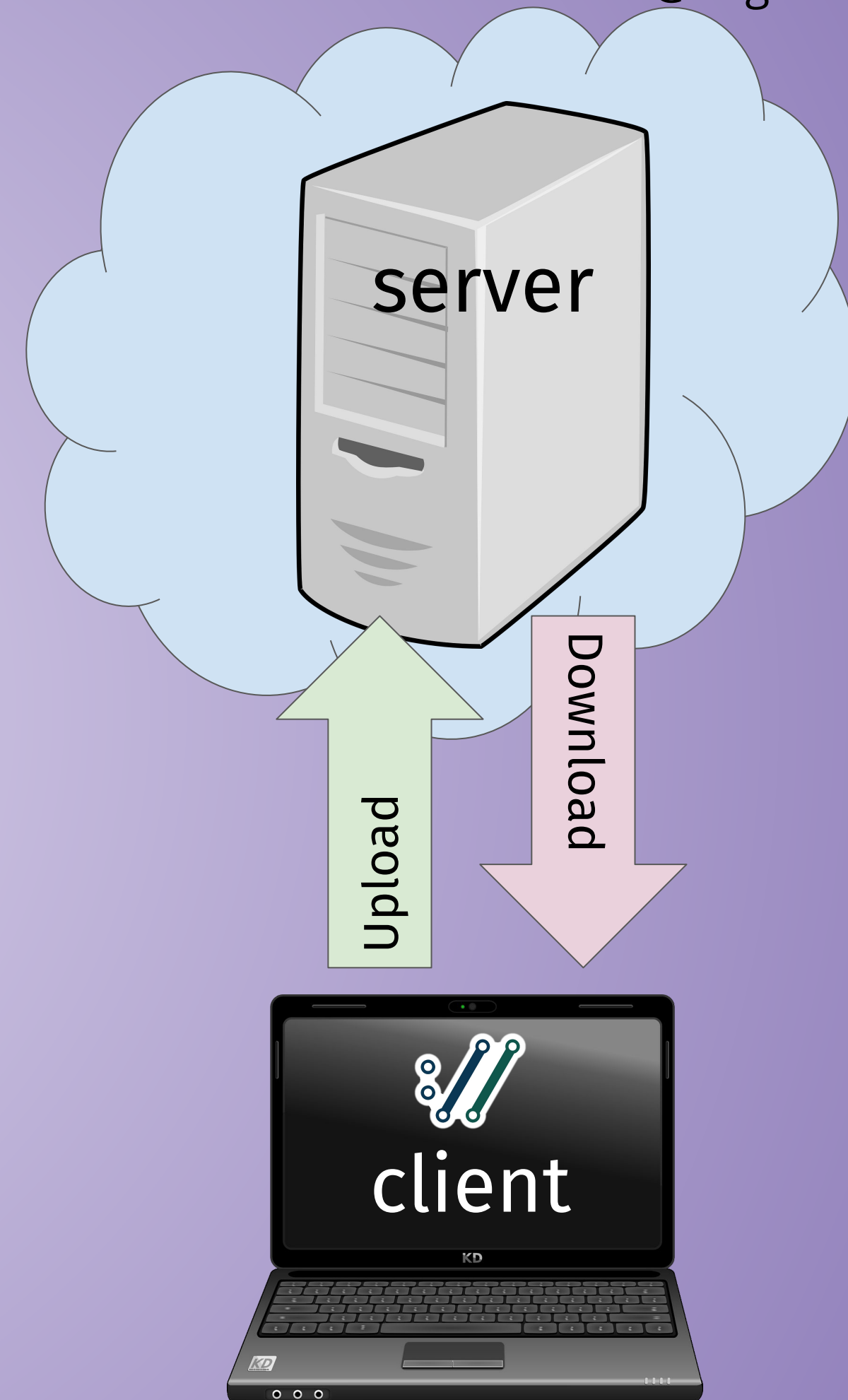
upload is data sent *to* the server

```
curl -T file $URL
```

if URL has no file name part, appends the -T name to the URL

```
curl -T file ftp://example.com/path/
```

HTTP offers several different uploads



transfer controls

stop slow transfers

```
--speed-limit <speed> --speed-time <seconds>
```

transfer rate limiting

```
curl --limit-rate 100K https://example.com
```

no more than this number of transfer starts per time unit

```
curl --rate 2/s https://example.com/[1-20].jpg
```

```
curl --rate 3/h https://example.com/[1-20].html
```

```
curl --rate 14/m https://example.com/day/[1-365]/fun.html
```



naming tricks

Provide a name + port => address mapping

```
curl --resolve example.com:443:127.0.0.1 https://example.com/
```

```
curl --resolve example.com:80:[2a04:4e42:200::347] http://example.com/
```

Provide a name + port => name + port mapping

```
curl --connect-to example.com:80:curl.se:8080 http://example.com/
```

Talking HTTP, it is also sometimes fun/useful to set Host: header:

```
curl -H "host: curl.fake" http://example.com/
```

When TLS is used, this might fail certificate checks

connection race

curl uses both IPv6 and IPv4 when possible - and **races** them against each other

“Happy eyeballs”

Restrict to a fixed IP version with `--ipv4` or `--ipv6`

```
curl.se has address 151.101.129.91
curl.se has address 151.101.193.91
curl.se has address 151.101.1.91
curl.se has address 151.101.65.91
curl.se has IPv6 address 2a04:4e42:800::347
curl.se has IPv6 address 2a04:4e42:a00::347
curl.se has IPv6 address 2a04:4e42:c00::347
curl.se has IPv6 address 2a04:4e42:e00::347
curl.se has IPv6 address 2a04:4e42::347
curl.se has IPv6 address 2a04:4e42:200::347
curl.se has IPv6 address 2a04:4e42:400::347
curl.se has IPv6 address 2a04:4e42:600::347
```

DNS



```
2a04:4e42:800::347
2a04:4e42:a00::347
2a04:4e42:c00::347
2a04:4e42:e00::347
2a04:4e42::347
2a04:4e42:200::347
2a04:4e42:400::347
2a04:4e42:600::347
```



curl.se



```
151.101.129.91
151.101.193.91
151.101.1.91
151.101.65.91
```


connections

Use a specific network interface

```
curl --interface enp3s0 https://example.com
```

local port number range

```
curl --local-port 1000-3000 https://example.com
```

TCP keep alive

```
curl --keepalive - time 23 https://example.com
```

DNS servers (when c-ares is used)

```
curl --dns-ipv4-addr 10.1.2.3 https://example.com
```


timeouts

Maximum total time allowed to spend

```
curl --max-time 12.34 https://curl.se/
```

Never spend more than this time to connect:

```
curl --connect-timeout 3.14 https://remote.example.com/
```

“Connect time” implies DNS and everything else before transfer starts



.netrc

a file for users to store their credentials for remote FTP servers

`$HOME/.netrc`

since 1978

`--netrc` makes curl use it

`--netrc - file [file]` to use another file

for all protocols

beware: weakly specified

```
$ cat $HOME/.netrc
machine example.com
login daniel
password qwerty
```


exit status

the numerical value curl returns back to the shell/prompt

zero for success

conveys the reason for errors

can be tested for in shell scripts

```
$ curl -o save https://example.com/
...
curl: (23) Failure writing output to destination
$ echo $?
23
```

```
#!/bin/sh
curl https://example.com/page.html -O
res=$?
if [ $res -ne 0 ]; then
    echo "curl command failed with $res"
fi
```


exit status

```
$ man curl
...
 0  Success. The operation completed successfully according to the instructions.

 1  Unsupported protocol. This build of curl has no support for this protocol.

 2  Failed to initialize.

 3  URL malformed. The syntax was not correct.

 4  A feature or option that was needed to perform the desired request was not
    enabled or was explicitly disabled at build-time. To make curl able to do
    this, you probably need another build of libcurl.

 5  Could not resolve proxy. The given proxy host could not be resolved.

 6  Could not resolve host. The given remote host could not be resolved.

 7  Failed to connect to host.

 8  Weird server reply. The server sent data curl could not parse.

 9  FTP access denied. The server denied login or denied access to the particu-
    lar resource or directory you wanted to reach. Most often you tried to
    change to a directory that does not exist on the server.

10  FTP accept failed. While waiting for the server to connect back when an ac-
    tive FTP session is used, an error code was sent over the control connection
    or similar.

11  FTP weird PASS reply. Curl could not parse the reply sent to the PASS re-
    quest.

12  During an active FTP session while waiting for the server to connect back to
    curl, the timeout expired.
...
```


SCP and SFTP

SSH-based instead of TLS

```
curl sftp://example.com/file.zip -u user
```

```
curl scp://example.com/file.zip -u user
```

```
curl sftp://example.com/ -u user
```

```
curl sftp://example.com/~/todo.txt -u daniel
```

```
~/.ssh/known_hosts
```

```
curl sftp://example.com -u user --insecure
```

A red starburst graphic with multiple points, containing the text 'Avoid --insecure' in white.

**Avoid
--insecure**

Reading email

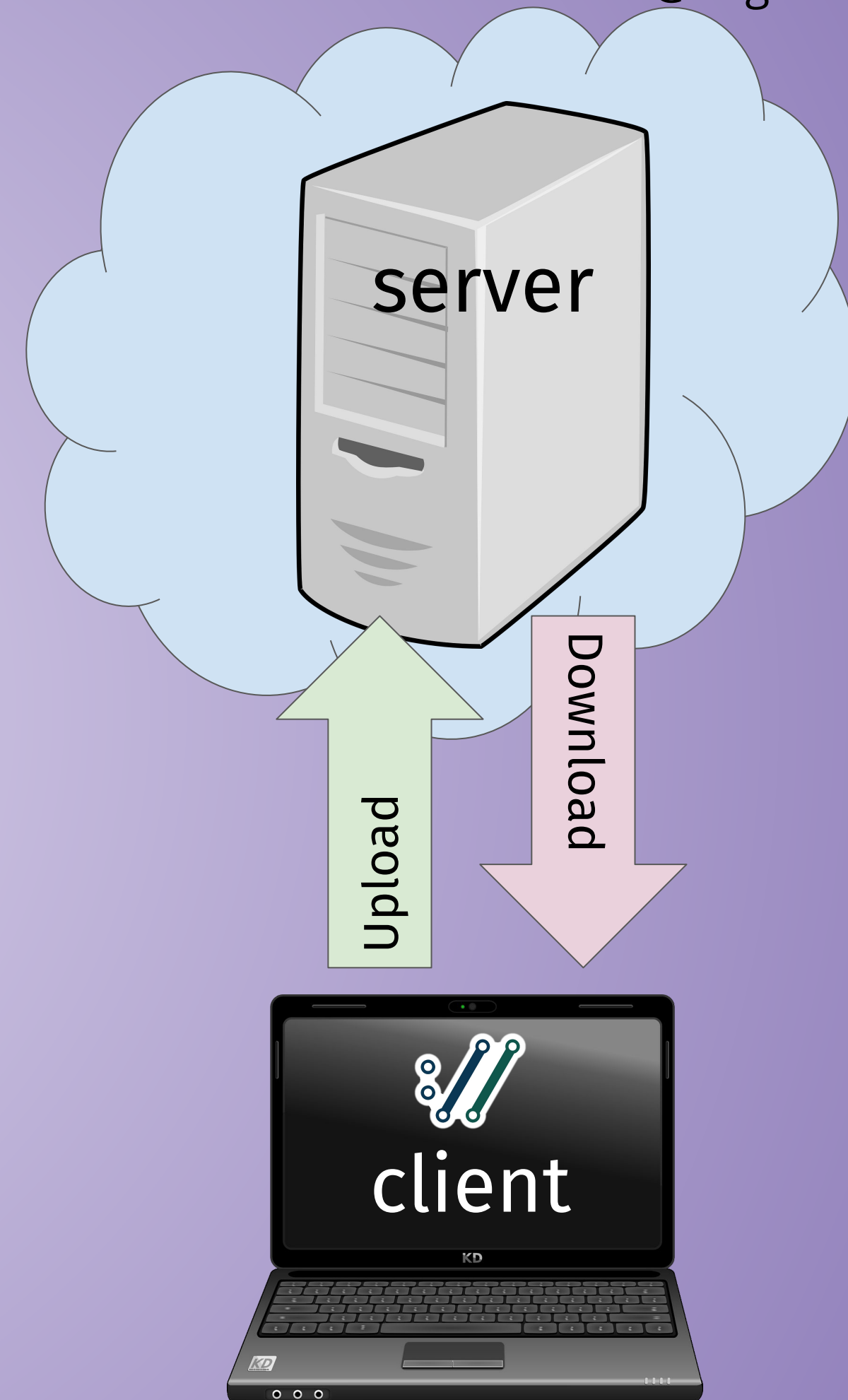
POP3 / IMAP

Reading is download

```
curl pop3://example.com/
```

```
curl imap://example.com/
```

usually with TLS (more on this a few slides later)



Sending email

SMTP

Sending is uploading

```
curl -T data smtp://example.com/ -u user:password
```

The file needs to have all the mail headers (To, From, Subject, ...)

The file needs to be “correctly” formatted

use this with TLS (more on this a few slides later)

MQTT

Subscribe to the bedroom temperature in the subject:

```
curl mqtt://example.com/home/bedroom/temp
```

Set the kitchen dimmer:

```
curl -d 75 mqtt://example.com/home/kitchen/dimmer
```


TFTP

Download a file from the TFTP server

```
curl -O tftp://localhost/file.boot
```

Upload a file to your TFTP server

```
curl -T file.boot tftp://localhost/
```


TELNET

An odd child in the curl family

Session, not really download/upload

```
curl telnet://example.com:80
```

Reads input on stdin

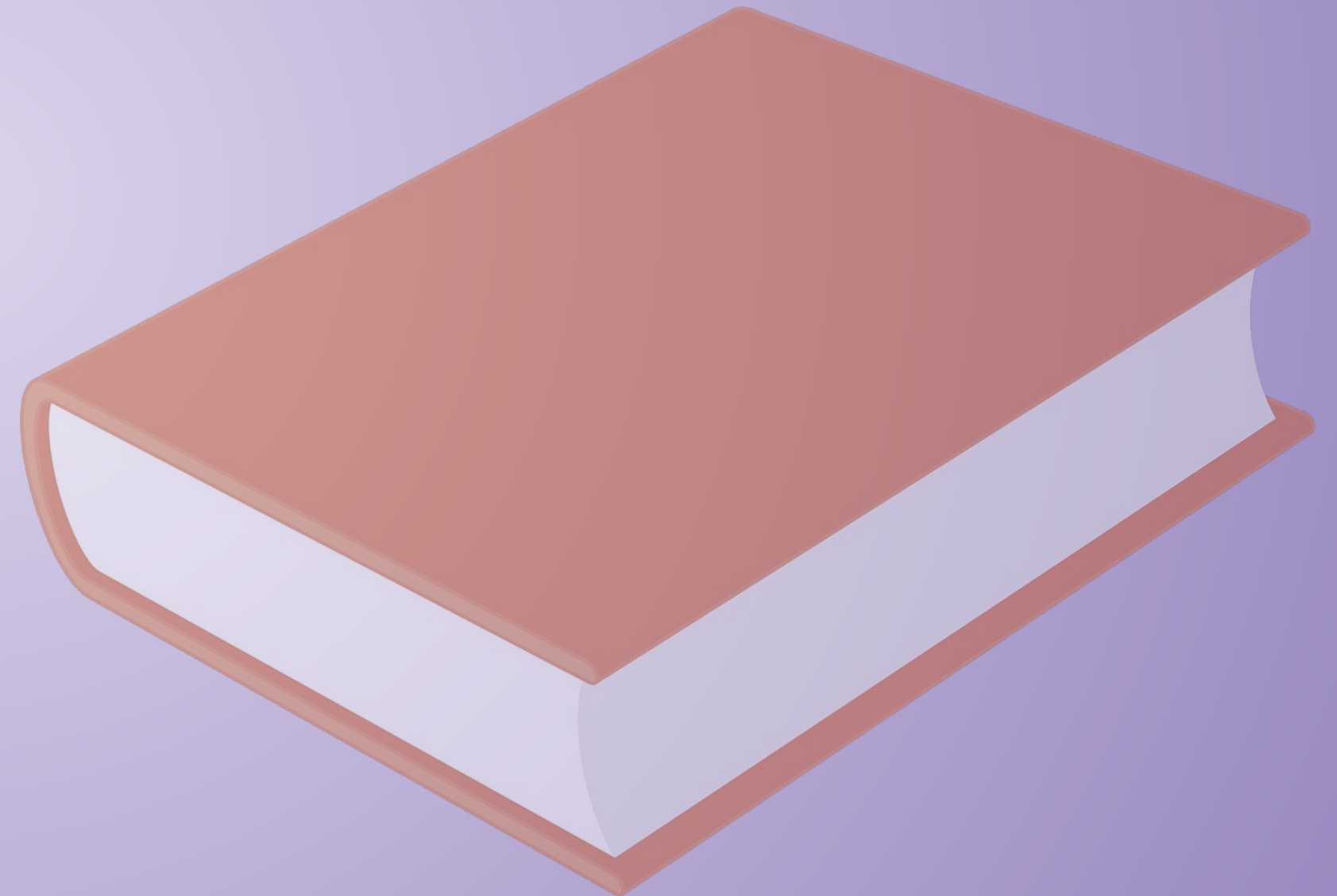
DICT

Dictionary lookups (downloads)

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

```
curl dict://dict.org/d:heisenbug:jargon
```

```
curl dict://dict.org/d:daniel:gcide
```



WebSocket

Still **experimental** in 8.2.1

Still not too fancy in the curl tool

```
curl wss://example.com/wss
```


curl vs browsers

“I downloaded HTML and it looks different”

Web Browsers supports different charsets and fonts

Web Browsers do (a lot of) JavaScript

Servers might provide different content depending on client

Servers might *act* differently: different cookies, different redirects

Some servers try very hard to identify clients (fingerprinting)



copy as curl

Generates a curl command line that mimics a previous transfer

Firefox, Chrome, Edge, Safari all feature this

other tools do as well

Never perfect, use as starting point

Live demo



figure out the browsers

Additional techniques to mimic browsers:

wireshark + SSLKEYLOGFILE (more on this soon)

`nc -l -p 8080` (next slide)



h2c - headers to curl command line

```
nc -l -p 8080
```

Make your browser go to [http://localhost:8080/...](http://localhost:8080/)

See what the request looks like

```
https://curl.se/h2c/
```

Demo

--libcurl

Generate libcurl source code from a command line

Gets a bootstrap skeleton for your program

Generates C code but often easy enough to convert

Demo

TLS

enable TLS

also still referred to as SSL

necessary for transport security (unless SSH is used)

TLS verifies the peer

prevents eavesdropping and tampering

typically indicated by the URL scheme (HTTPS://)

all schemes curl supports that end with S do TLS

`--ssl` - reqd for FTP IMAP POP3 SMTP LDAP

SCP and SFTP use SSH instead of TLS, similar but different



Use TLS!

TLS versions

curl uses latest version/suitable ciphers automatically

Offers: `--tlsv1.0` `--tlsv1.1` `--tlsv1.2` `--tlsv1.3` (lowest version accepted)

`--tls - max <VERSION>` (highest version accepted)

`--sslv2` `--sslv3` don't work anymore due to security problems



verifying server certificates

verifying certs is key to security

CAs sign certificates

curl trusts CAs

intermediate certificates

CA store as file or “native”

`--cacert`

`--insecure (-k)`

<https://curl.se/docs/caextract.html>



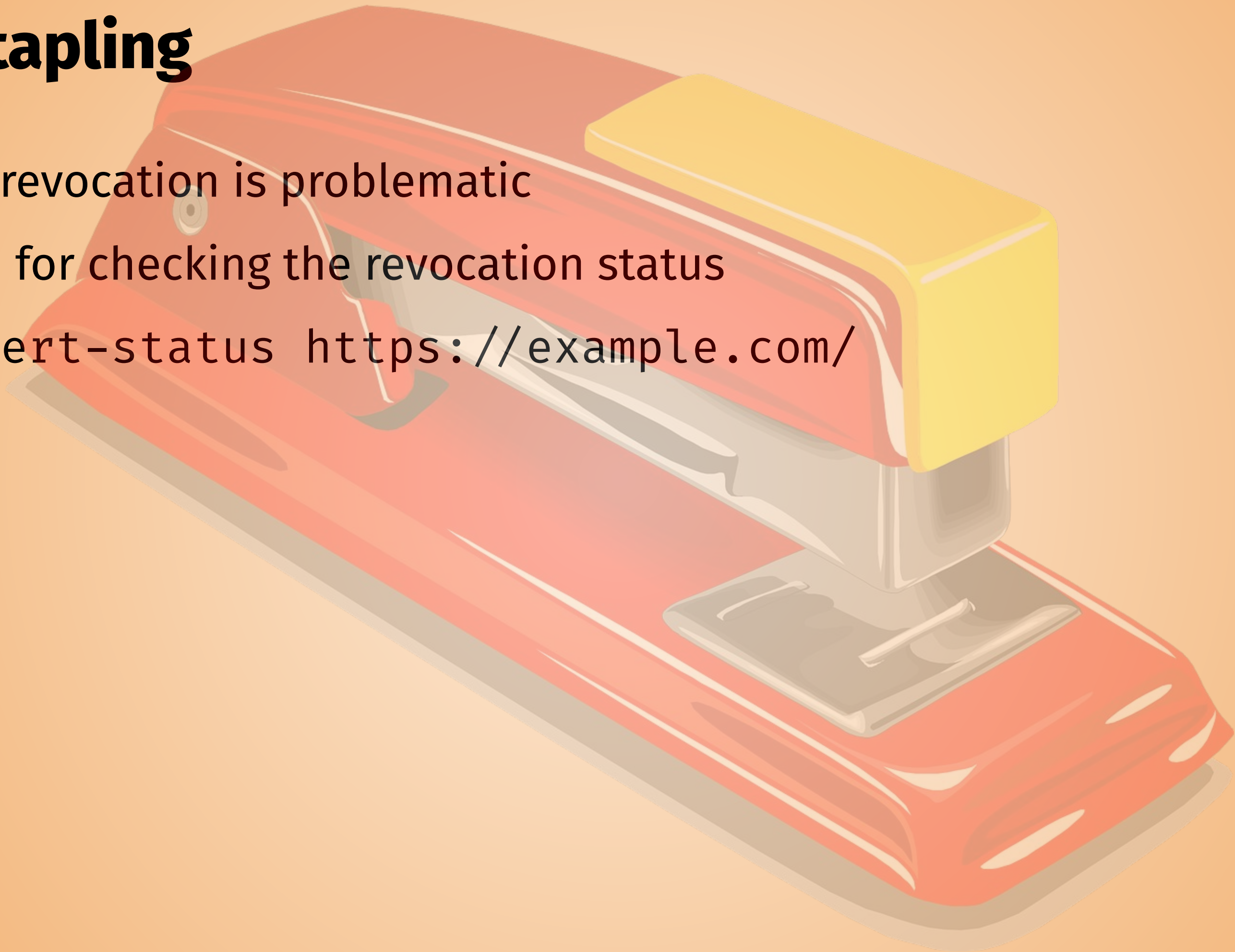
**Avoid
-k**

OCSP stapling

certificate revocation is problematic

a standard for checking the revocation status

```
curl --cert-status https://example.com/
```



client certificates

often called “mutual authentication”

prove to the server that the client is “legitimate”

```
curl --cert mycert:mypassword https://example.com
```

```
curl --cert mycert:mypassword --key mykey https://example.com
```


ciphers

TLS negotiates what ciphers to use

--ciphers

--tls13-ciphers

--proxy-ciphers

Post Quantum

<https://curl.se/docs/ssl-ciphers.html>

AES128-SHA256
AES256-SHA256
AES128-GCM-SHA256
DH-RSA-AES128-SHA256
DH-RSA-AES256-SHA256
DH-RSA-AES128-GCM-SHA256
DH-RSA-AES256-GCM-SHA384
DH-DSS-AES128-SHA256
DH-DSS-AES256-SHA256
DH-DSS-AES128-GCM-SHA256
DH-DSS-AES256-GCM-SHA384
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-SHA256
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES256-GCM-SHA384
DHE-DSS-AES128-SHA256
DHE-DSS-AES256-GCM-SHA256
DHE-DSS-AES128-GCM-SHA384
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-SHA256
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ADH-AES128-SHA256
ADH-AES256-SHA256
ADH-AES128-GCM-SHA256
ADH-AES256-GCM-SHA384
DHE-RSA-AES128-CCM
DHE-RSA-AES256-CCM
DHE-RSA-AES128-CCM8
DHE-RSA-AES256-CCM8
ECDHE-ECDSA-AES128-CCM
ECDHE-ECDSA-AES256-CCM
ECDHE-ECDSA-AES128-CCM8
ECDHE-ECDSA-AES256-CCM8
TLS_AES_128_GCM_SHA256
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_256_GCM_SHA256
TLS_AES_128_GCM_SHA256
TLS_AES_128_CCM_SHA256
TLS_AES_128_CCM8_SHA256

TLS backends

curl can get built with different TLS backends

different backends might..

- behave slightly different
- have different CA stores
- have different feature sets
- work on different platforms

OpenSSL is the most commonly used TLS library for curl

Secure Transport

OpenSSL

BearSSL

AWS-LC

BoringSSL

rustls

wolfSSL

AmiSSL

Schannel

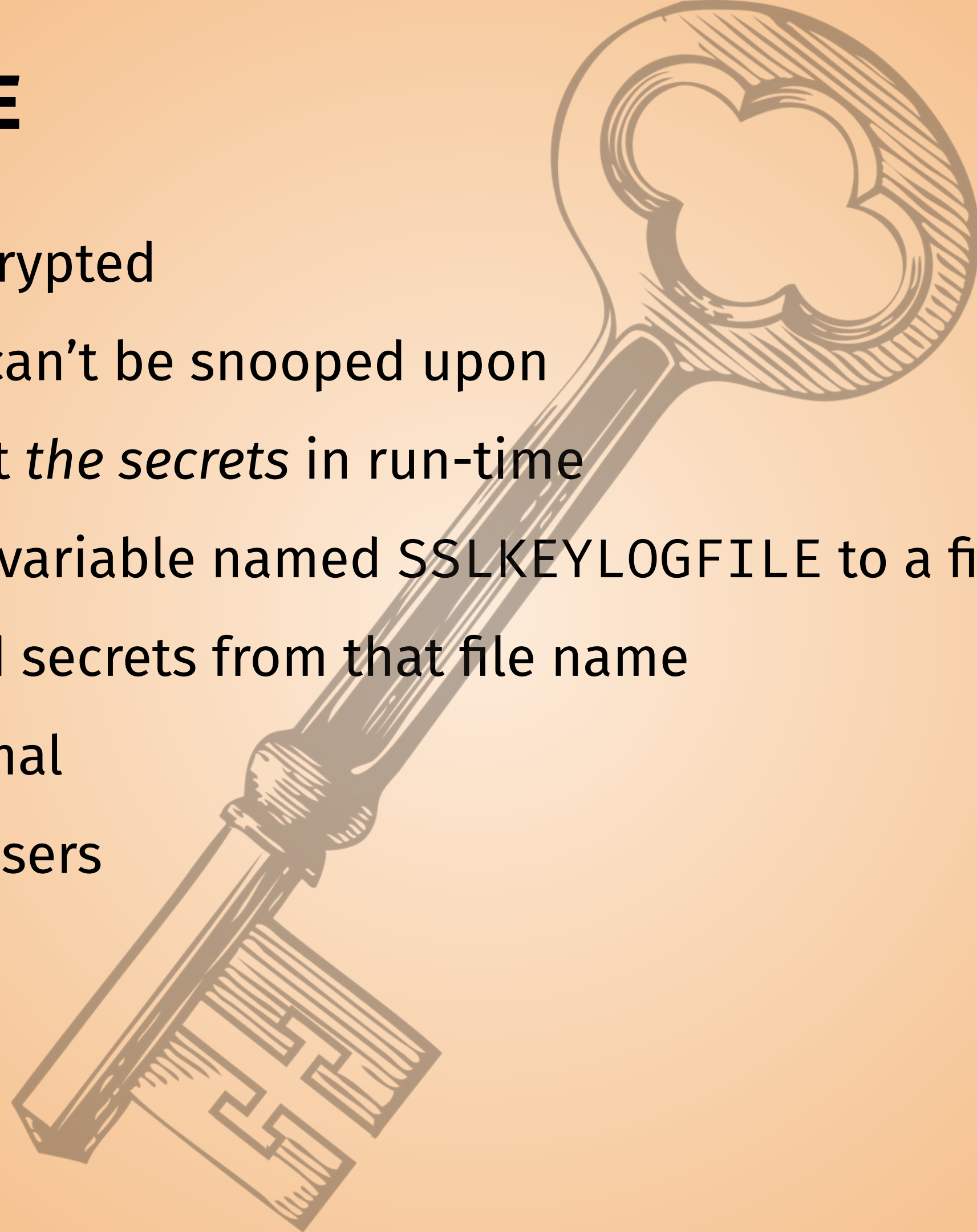
GnuTLS

mbedSSL

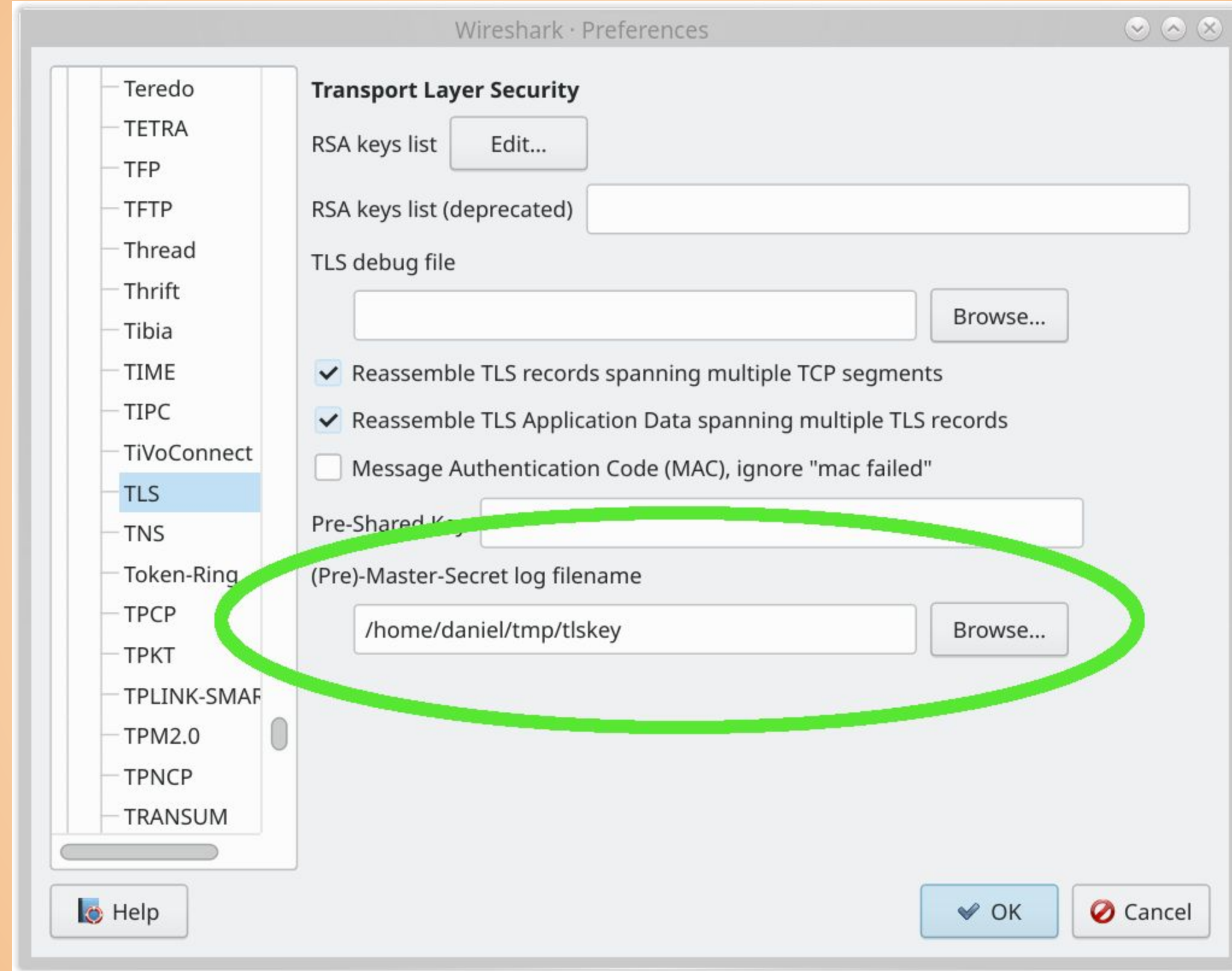
libressl

SSLKEYLOGFILE

TLS transfers are encrypted
encrypted transfers can't be snooped upon
unless we can extract *the secrets* in run-time
set the environment variable named SSLKEYLOGFILE to a file name
tell Wireshark to read secrets from that file name
then run curl as normal
also works with browsers



SSLKEYLOGFILE



SSLKEYLOG

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter:

Expression...

Clear

Apply

Save

| No. | Time | Source | src port | Destination | dest port | Protocol | Length | Info |
|-----|-------------|--------------|----------|----------------|-----------|----------|--------|---|
| 10 | 0.005909000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | TLSv1.2 | 1529 | Server Hello, Certificate, Server Key Exchange, |
| 11 | 0.000011000 | 192.168.0.19 | 37965 | 106.186.112.11 | 443 | TCP | 66 | 37965 → 443 [ACK] Seq=211 Ack=1464 Win=251 Len= |
| 12 | 0.005044000 | 192.168.0.19 | 37965 | 106.186.112.11 | 443 | TLSv1.2 | 296 | Client Key Exchange, Change Cipher Spec, Encryp |
| 13 | 0.000332000 | 192.168.0.19 | 37965 | 106.186.112.11 | 443 | HTTP2 | 144 | Magic, SETTINGS, WINDOW_UPDATE |
| 14 | 0.000011000 | 192.168.0.19 | 37965 | 106.186.112.11 | 443 | HTTP2 | 270 | HEADERS, WINDOW_UPDATE |
| 15 | 0.236724000 | 106.186.112. | 443 | 192.168.0.192 | 37966 | TCP | 66 | 443 → 37966 [ACK] Seq=1 Ack=211 Win=235 Len=0 T |
| 16 | 0.006685000 | 106.186.112. | 443 | 192.168.0.192 | 37966 | TLSv1.2 | 1529 | Server Hello, Certificate, Server Key Exchange, |
| 17 | 0.000019000 | 192.168.0.19 | 37966 | 106.186.112.11 | 443 | TCP | 66 | 37966 → 443 [ACK] Seq=211 Ack=1464 Win=251 Len= |
| 18 | 0.005084000 | 192.168.0.19 | 37966 | 106.186.112.11 | 443 | TLSv1.2 | 296 | Client Key Exchange, Change Cipher Spec, Encryp |
| 19 | 0.000195000 | 192.168.0.19 | 37966 | 106.186.112.11 | 443 | HTTP2 | 144 | Magic, SETTINGS, WINDOW_UPDATE |
| 20 | 0.048348000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | TCP | 66 | 443 → 37965 [ACK] Seq=1464 Ack=723 Win=252 Len= |
| 21 | 0.000017000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | HTTP2 | 164 | SETTINGS |
| 22 | 0.000222000 | 192.168.0.19 | 37965 | 106.186.112.11 | 443 | HTTP2 | 103 | SETTINGS |
| 23 | 0.000139000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | HTTP2 | 103 | SETTINGS |
| 24 | 0.001705000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | TCP | 2962 | [TCP segment of a reassembled PDU] |
| 25 | 0.000017000 | 192.168.0.19 | 37965 | 106.186.112.11 | 443 | TCP | 66 | 37965 → 443 [ACK] Seq=760 Ack=4495 Win=297 Len= |
| 26 | 0.000169000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | HTTP2 | 1152 | HEADERS, DATA |
| 27 | 0.000016000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 28 | 0.000009000 | 192.168.0.19 | 37965 | 106.186.112.11 | 443 | TCP | 66 | 37965 → 443 [ACK] Seq=760 Ack=7029 Win=342 Len= |
| 29 | 0.000221000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 30 | 0.000005000 | 106.186.112. | 443 | 192.168.0.192 | 37965 | HTTP2 | 670 | DATA, DATA |

▼ Stream: SETTINGS, Stream ID: 0, Length 10

...0 0000 0000 1010 = Length: 10

000. = Reserved: 0

Type: SETTINGS (4)

▶ Flags: 0x00

0... = Reserved: 0x00000000

.000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0

▶ Settings - Max concurrent streams : 100

▶ Settings - Initial windows size : 65535

0000 00 0a 04 00 00 00 00 00 03 00 00 00 64 04 00 00d...

0010 ff ff ..

Frame (164 bytes)

Decrypted SSL record (16 bytes)

Decrypted SSL data (18 bytes)

Settings (http2.settings), 5 bytes

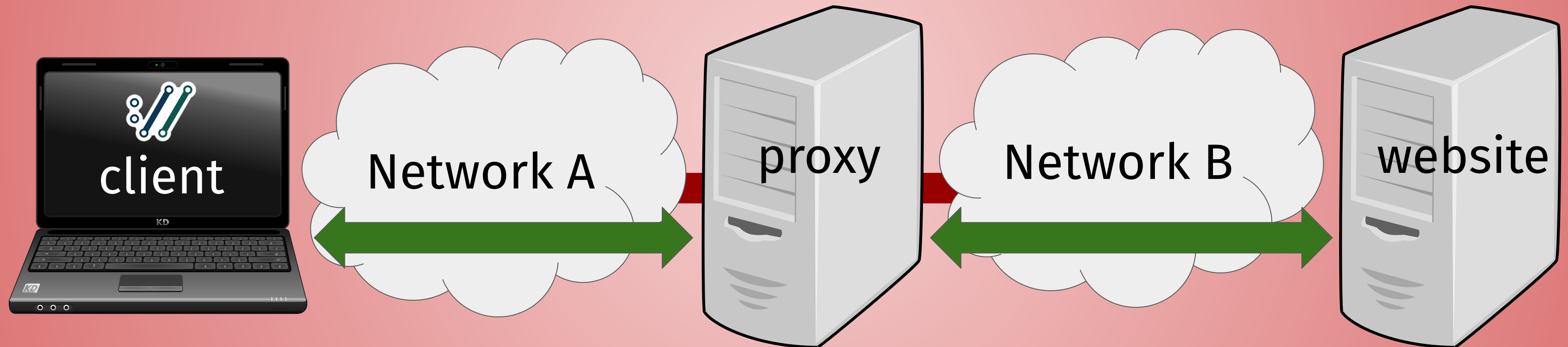
Packets: 179 · Displayed: 179 (100.0%) · ...

Profile: Default

Proxies

a proxy is an intermediary

*a server application that acts as **an intermediary** between a client requesting a resource and the server providing that resource*



discover your proxy

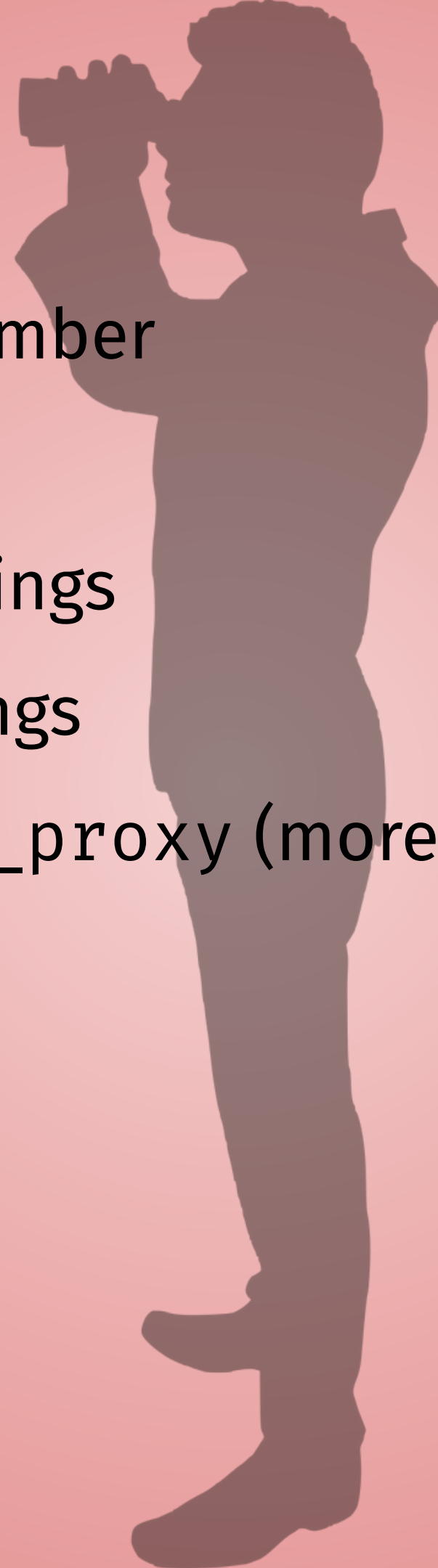
a proxy is a type + host + port number

curl does not auto-detect it

check your browser network settings

check your system network settings

environment variables like `http_proxy` (more on this soon)



PAC - Proxy Auto-Configuration

JavaScript that determines which proxy to use for a given URL

curl does not support PAC

real-life PACs are all from simple to super complicated

when too complicated for manual inspection: check behavior

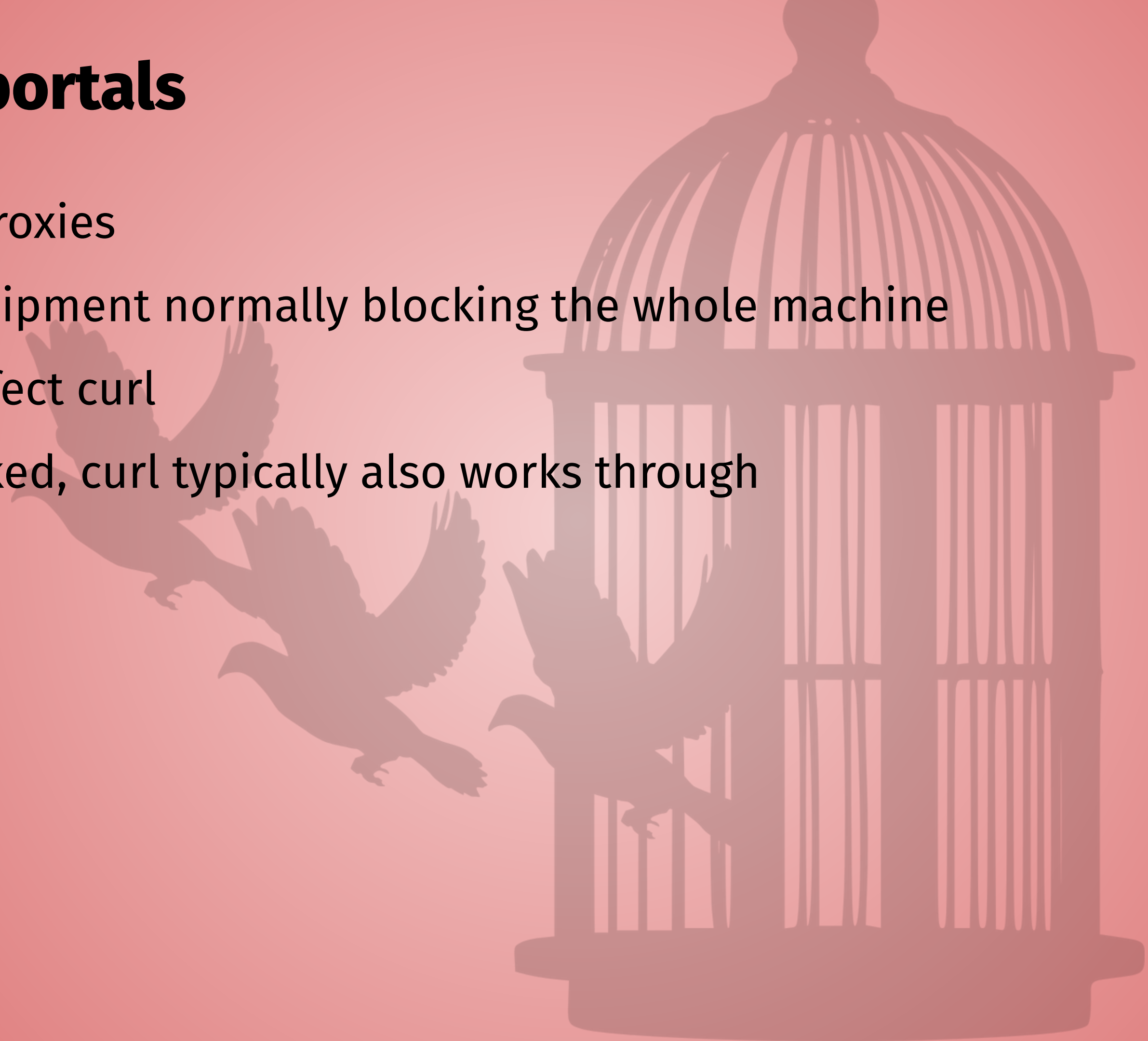
Captive portals

Not actual proxies

Network equipment normally blocking the whole machine

They also affect curl

When unlocked, curl typically also works through



Proxy types

There are different proxy *types*

HTTP vs SOCKS

HTTP / HTTPS / HTTPS-H2 / SOCKS4 / SOCKS4a / SOCKS5 / SOCKS5h

use type as URL scheme when set with --proxy

--preproxy allows both SOCKS **and** HTTP(S) proxy

TOR is SOCKS

forward proxy and reverse proxy

HTTPS proxy

HTTPS to the proxy, *anything* over it

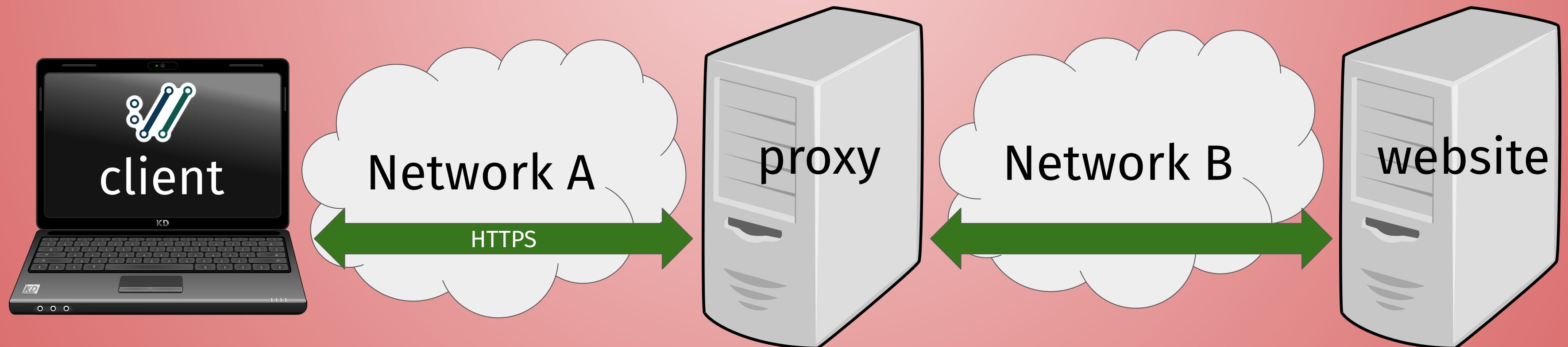
HTTPS authenticates the proxy: ☐

curl supports HTTP/1 and HTTP/2 over HTTPS proxy

Prevents users on network A from eavesdropping

Features its own set of TLS options

--proxy-*, like --proxy-insecure and --proxy-tlsv1.3



MITM proxy

Sometimes used for debugging

Sometimes used for surveillance

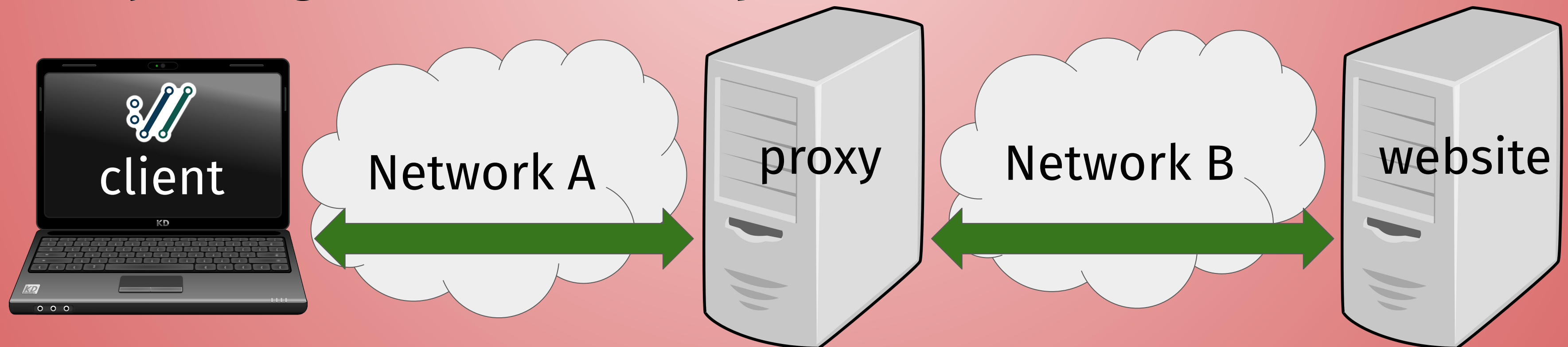
The proxy terminates TLS and can snoop on data

MITM proxies present a certificate from a CA normally not trusted

curl must be told to trust that CA to allow this stunt

blindly trusting a middleman is a recipe for disasters

Avoid



Proxy authentication

Proxies might require authentication

curl supports numerous proxy auth methods

-U, --proxy-user <user:password>

--socks5-basic

--socks5-gssapi

--proxy-basic

--proxy-digest

--proxy-negotiate

--proxy-ntlm

Proxy environment variables

[scheme]_proxy

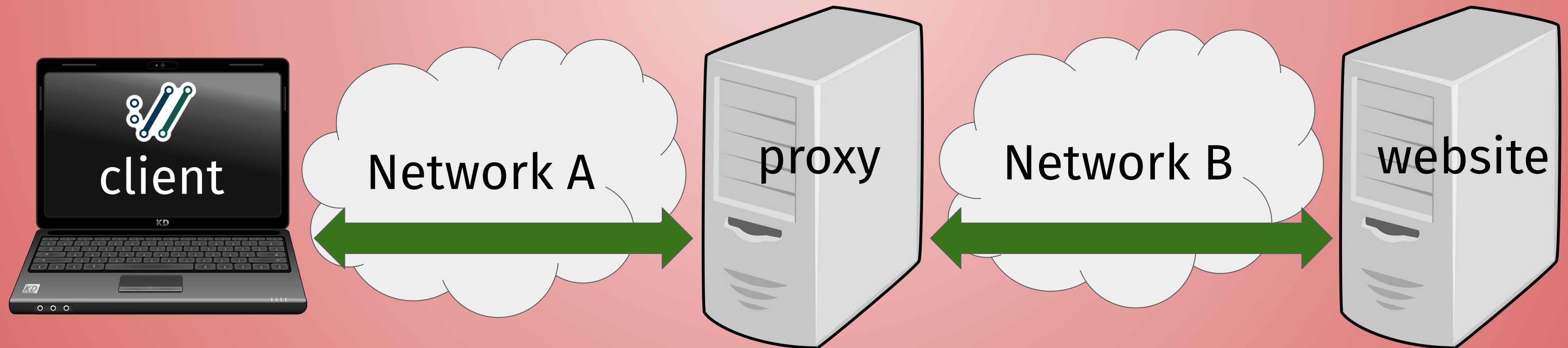
| | |
|---------------------------|-------------|
| curl https://example.com/ | HTTPS_PROXY |
| curl ftp://example.com/ | FTP_PROXY |
| curl http://example.com/ | http_proxy |

ALL_PROXY

NO_PROXY

Proxy headers

--proxy-header vs --header



HTTP

HTTP protocol basics

HTTP/1 - HTTP/2 - HTTP/3

HTTPS is HTTP + TLS

Name resolve + TCP+TLS/QUIC

Request - Response

Method - verbs

```
HTTP/1.1 200 OK
Server: example-server/1.1
Content-Length: 5
Content-Type: plain/text
```

```
hello
```

```
GET /index.html HTTP/1.1
User-agent: curl/2000
Host: example.com
```

```
HEAD /index.html HTTP/1.1
User-agent: curl/2000
Host: example.com
```

```
POST /index HTTP/1.1
Host: example.com
User-agent: curl/2000
Content-Length: 5
```

```
hello
```

```
PUT /index HTTP/1.1
Host: example.com
User-agent: curl/2000
Content-Length: 5
```

```
hello
```

HTTP Method

Sometimes called “verb”

GET

HEAD (-I)

POST (-d or -F)

PUT (-T)

anything

methods are sometimes abused

-X [whatever]



Avoid -X

HTTP headers in terminal

bold header names

“linkified” Location: URLs

```
$ curl -I curl.se
HTTP/1.1 301 Moved Permanently
Connection: close
Content-Length: 0
Server: Varnish
Retry-After: 0
Location: https://curl.se/
Accept-Ranges: bytes
Date: Fri, 25 Aug 2023 18:33:12 GMT
Via: 1.1 varnish
X-Served-By: cache-qpg1235-QPG
X-Cache: HIT
X-Cache-Hits: 0
X-Timer: S1692988392.312085,VS0,VE0
alt-svc: h3=":443";ma=86400,h3-29=":443";ma=86400,h3-27=":443";ma=86400
```

HTTP response code

HTTP “success”

200 OK

404 File not found

curl does not care

-f makes curl care

-w ‘%{response_code}’

```
HTTP/1.1 200 OK
Server: example-server/1.1
Content-Length: 5
Content-Type: plain/text

hello
```


HTTP response headers

-v or -l to see them

or -i

describes the body or the transfer

save them with -D [filename]

looks like HTTP/1 even when other HTTP versions are used

```
HTTP/1.1 200 OK
Server: example-server/1.1
Content-Length: 5
Content-Type: plain/text

hello
```

HTTP response bodies

The “payload”

Content-Length

HTTP/1.1 chunked-encoding

HTTP/2 and HTTP/3 have distinct end of data markers

--compressed

store to file with `-o [file]` or `-O`

```
HTTP/1.1 200 OK
Server: example-server/1.1
Content-Length: 5
Content-Type: plain/text

hello
```


HTTP authentication

web logins are often done with cookies, we get to those later

if HTTP based, returns 401 for server auth needed

returns 407 for proxy auth

WWW-Authenticate:

Basic, Digest, NTLM, Negotiate, etc

`-u [user]:[password]`

`--anyauth`

`--basic, --digest, --ntlm, --negotiate`

HTTP ranges

Ask for *a piece* of a remote resource

```
curl --range 500-999 https://example.com/file.txt
```

server may ignore the ask

curl's -C [where] *resumes* the transfer at that index



HTTP versions

curl supports HTTP/0.9, HTTP/1.0, HTTP/1.1, HTTP/2 and HTTP/3

Generally: you don't need to care

Different over the wire, made to look similar for users

HTTP/0.9 must be enabled with `--http0.9`

HTTP/1.0 with `--http1.0`

HTTP/1.1 is a general default or `--http1.1`

HTTP/2 is default over HTTPS, or asked for with `--http2`

HTTP/3 is experimental, asked for with `--http3`

HTTP versions

What version does the server support?

```
curl -sI https://curl.se -o/dev/null -w '%{http_version}\n'
```

```
curl -sI --http3 https://curl.se -o/dev/null -w '%{http_version}\n'
```


HTTP time based conditions

do the transfer only if...

The remote resource is **newer**:

```
curl --time-cond "Wed 01 Sep 2021 12:18:00" https://example.com/file
```

The remote resource is **older**:

```
curl --time-cond "-Wed 01 Sep 2021 12:18:00" https://example.com/file
```

Newer than the local file:

```
curl --time-cond file https://example.com/file
```

Set the remote date on the local file:

```
curl -R -O https://example.com/file
```

HTTP etags

do the transfer only if...

The remote resource is “different”

```
curl --etag-save remember -0 https://example.com/file
```

```
curl --etag-compare remember -0 https://example.com/file
```

Both can be used at once for convenient updates:

```
curl --etag-save remember --etag-compare remember -0  
https://example.com/file
```


HTTPS

HTTPS is HTTP with added TLS for security

HTTPS: // instead of HTTP: // **should** be the only difference

curl negotiates the latest TLS version + suitable cipher

HTTPS:// tries HTTP/2 by default, HTTP:// does HTTP/1.1

HTTP/3 will only be done for HTTPS:// (more on this soon)

HTTP POST: simple

pass *any* data to a HTTP(S) server

```
curl -d 'name=admin&shoesize=12' https://example.com/
```

```
curl -d name=admin -d shoesize=12 https://example.com/
```

```
curl -d @filename http://example.com
```

```
curl --data-raw '@string' https://example.com
```

```
curl --data-binary @filename https://example.com
```


HTTP POST: content-type

-d / -data defaults to

Content-Type: application/x-www-form-urlencoded

```
curl -d dust -H 'Content-Type: stuff/dream' https://example.com
```

HTTP POST: JSON

```
curl --json '{"name": "daniel"}' https://example.com
```

```
curl --json @object.json https://example.com
```

Sets **Content-Type: application/json** and **Accept: application/json**

Create JSON easily

```
jo -p name=jo n=17 parser=false | curl --json @- https://example.com/
```

Receive/parse JSON easily

```
curl --json '{"tool": "curl"}' https://example.com/ | jq
```

curl + jo + jq

```
jo -p name=jo n=17 | curl --json @- https://example.com/ | jq
```


HTTP POST: URL encoding

`--data-urlencode` helps URL encode data to send

```
curl --data-urlencode "name=John Doe (Junior)" http://example.com  
sends name=John%20Doe%20%28Junior%29
```

`-data-urlencode [content]` where content is...

| | |
|------------------|--|
| anything | URL encode the content |
| =anything | URL encode the content (leave out the '=') |
| any=thing | Send as "any=[URL encoded thing]" |
| @anything | Read content from file, URL encode and use |
| any@thing | Send as "any=[URL encoded file contents]" |

HTTP POST: convert to GET (query)

`curl -d name=admin -d shoesize=12 https://example.com/`
but do it as a GET instead

`curl -d name=admin -d shoesize=12 https://example.com/ --get`

```
GET /?name=admin&shoesize=12 HTTP/1.1
User-agent: curl/8.2.1
Host: example.com
```

we recommend using `--url-query` instead!

HTTP POST: Expect 100-continue

For HTTP/1.1 only

A request header used by curl when POST or PUT > 1 megabyte

Meant to avoid sending lots of data if server does not want it

Server responding 100 means: 👍 - go ahead

If it bothers you, disable with `curl -H Expect: https://example.com/`

Commonly ignored by servers, leading to wasted waiting time

HTTP POST: chunked

Sends data without specifying the size up front

For HTTP/1.1 only

```
curl -H "Transfer-Encoding: chunked" -d @file http://example.com
```


HTTP POST: <form>

The HTML <form> tag is “filled in” with a POST

type=hidden fields as well

```
-d name1=var1 -d name2=var2 -d name3=var3 ...
```

The `action=[here]` identifies where to send the POST

“Copy as curl” is your friend

HTTP multipart formpost

This is a POST sending data in a special multipart format

Content-Type multipart/form-data

The data is sent as a series of “parts”, one or more

Each part has a name, separate headers, file name and more

Each part is separated by a “mime boundary”

```
-----d74496d66958873e
Content-Disposition: form-data; name="person"

anonymous
-----d74496d66958873e
Content-Disposition: form-data; name="secret"; filename="file.txt"
Content-Type: text/plain

contents of the file
-----d74496d66958873e--
```


HTTP multipart formpost

`curl -F [content]` adds one part per instance

Use as many -F as you like

Insert plain text content `-F "name=Daniel Stenberg"`

Insert content from a file `-F name=<file.txt`

Insert a file as an "upload" `-F name=@file.jpg`

Insert a file, different file name `-F name=@file.jpg; filename=fake.jpg`

Set a custom content-type: `-F name=@file.jpg; type=image/myown`

`<form action="submit" method="post" enctype="multipart/form-data">`

HTTP -d or -F

Both sends HTTP POST

Both works over every HTTP version

Both can “fill in” HTML <form>

What data do you need to send?

Very rarely can the client decide. Do what the server expects!

HTTP redirects

The response you want is ... *over there!*

```
HTTP/1.1 301
Server: example-server/1.1
Location: https://example.com/over-here.html

hello
```



Avoid -X

A 30X response code + Location: header

tell curl to “follow” with --location (-L)

```
curl -L curl.se
```

```
curl -L curl.se --max-redirs 7
```

The numeric code defines method in redirected-to request

```
--location-trusted
```

HTTP modify the request

Sensible and basic by default

You as a user add the bells and whistles

Modify the method with **--request**

Add/change/remove/blank headers with **--header**

```
curl -H "curl-master: very-soon" http://example.com/
```

```
curl -H "Host: test.example" http://example.com/
```

```
curl -H "User-agent:" http://example.com/
```

```
curl -H "User-agent;" http://example.com/
```


HTTP modify the request

The request “target” is made from the URL path + query

```
GET /user/profile?shoesize=12 HTTP/1.1
User-agent: curl/8.2.1
Host: example.com
```

```
curl -X OPTIONS --request-target "*" http://example.com/
```

Convenient shortcuts:

--user-agent [string]

--referer [URL] (yes spelled wrong)

Remember “copy as curl”

HTTP PUT

The “upload file” of HTTP (and others)

```
curl -T localfile https://example.com/destination/replacement
```

```
curl -T - https://example.com/destination/replacement
```

```
curl -T file https://example.com
```

```
curl -T "img[1-1000].png" http://example.com/images/
```

```
curl --upload-file "{file1,file2}" https://example.com
```

```
curl -d "data to PUT" -X PUT http://example.com/new/resource/file
```


HTTP cookies: an explainer

key/value pairs that a client stores on the behalf of a server

sent back in subsequent requests

only when the cookie properties match

each cookie has an **expiration date** - or end of “session”

A **session** typically ends when user closes browser

When running curl command lines, when do you close the browser?



HTTP cookies: send some

you can tell curl to send specific cookies name + values

```
curl -b "name=daniel;talks=alot" https://example.com
```

rarely what you want

Often what “copy as curl” will give you



HTTP cookies: start the engine

curl ignores cookies by default
needs the *cookie engine* enabled first

specify a file to read from or use a blank string
`curl -b "" https://example.com`

More practically combined with redirect follows
`curl -L -b "" https://example.com`

The cookie engine keeps the cookies in memory
it forgets cookies that expire
only sends cookies according to the rules



HTTP cookies: cookie jar

Maybe also combined with saving a *cookie jar*

Writes the in-memory cookies to the given file at exit

```
curl -L -b "" -c cookies.txt https://example.com
```

Read from and write to the *cookie jar* (can be different files)

```
curl -L -b cookies.txt -c cookies.txt https://example.com
```

The *cookie jar* is a readable text file

The *cookie jar* uses the “netscape cookie format”

Also includes “session cookies” because ... sessions



HTTP cookies: session

curl does not know when a “session” ends

you need to say when a new cookie session starts

```
--junk - session - cookies
```

```
curl -J -b cookies.txt https://example.com
```



The logo for HTTP/2, featuring the text "HTTP/2" in white and yellow on a blue hexagonal background.

HTTP/2

HTTP version 2 changed how data is sent over the wire

curl hides those differences from users

curl tries to negotiate HTTP/2 for all HTTPS transfers

with `--http2` you can ask for HTTP/2 for HTTP:// transfers

With HTTP/2, curl can do multiplexed transfers with `-Z`

HTTP/3

HTTP version 3 changed how data is sent over the wire - again

HTTP/3 is done over QUIC, a new transport protocol

QUIC replaces TCP + TLS, and runs over UDP

curl hides protocol differences from users

HTTP/3 is **experimental** in curl

HTTP/3 is **only for HTTPS**, there is no clear text version

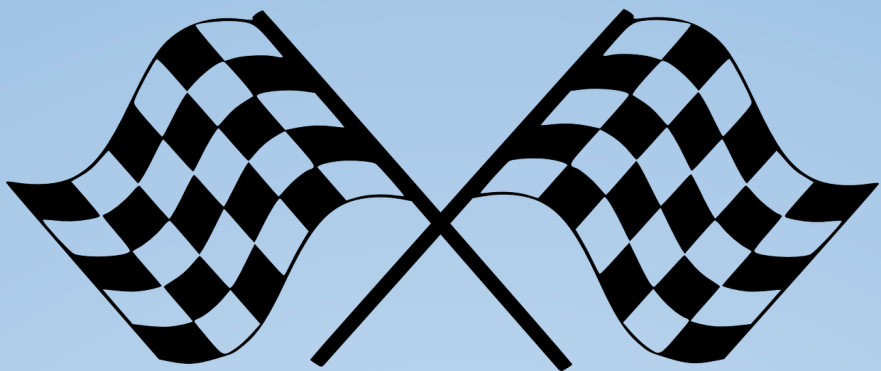
with `--http3` you can ask curl to attempt HTTP/3

`--http3` **races** HTTP/3 against HTTP/1+2 and picks the winner

With HTTP/3, curl can do multiplexed transfers with `-Z`

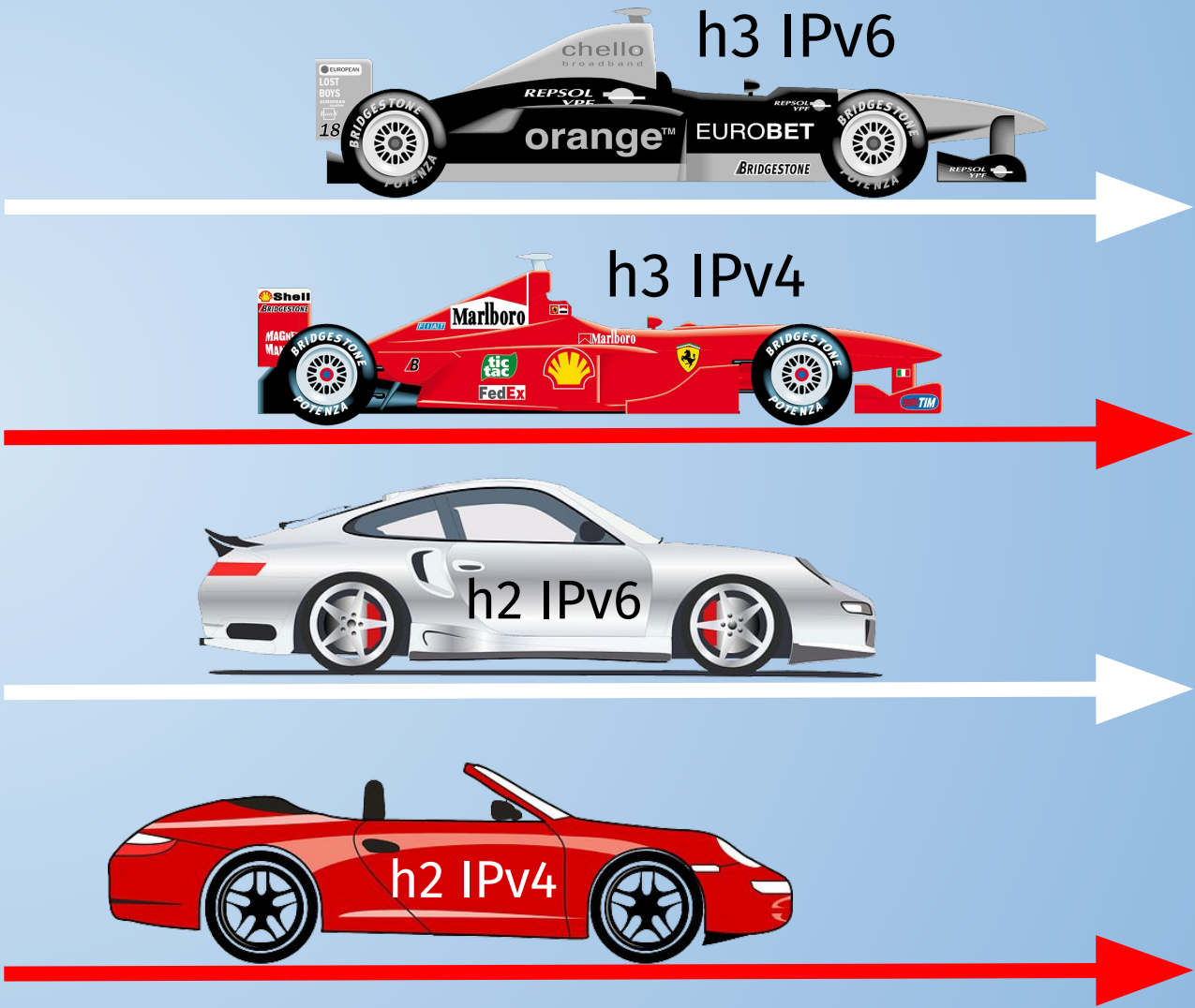
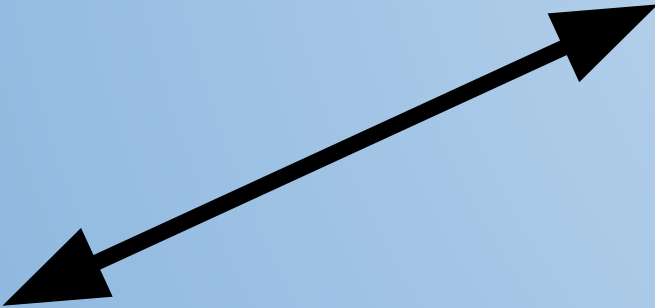
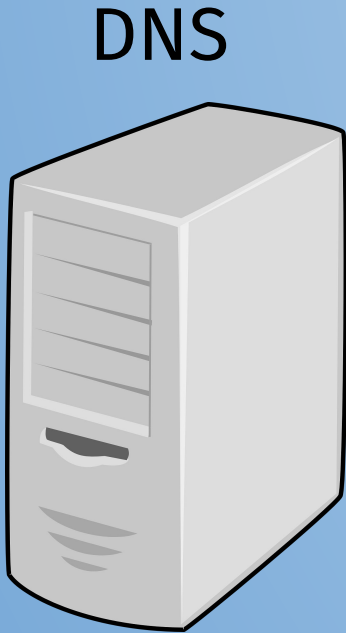


racing



2a04:4e42:800::347
2a04:4e42:a00::347
2a04:4e42:c00::347
2a04:4e42:e00::347
2a04:4e42::347
2a04:4e42:200::347
2a04:4e42:400::347
2a04:4e42:600::347

curl.se has address 151.101.129.91
curl.se has address 151.101.193.91
curl.se has address 151.101.1.91
curl.se has address 151.101.65.91
curl.se has IPv6 address 2a04:4e42:800::347
curl.se has IPv6 address 2a04:4e42:a00::347
curl.se has IPv6 address 2a04:4e42:c00::347
curl.se has IPv6 address 2a04:4e42:e00::347
curl.se has IPv6 address 2a04:4e42::347
curl.se has IPv6 address 2a04:4e42:200::347
curl.se has IPv6 address 2a04:4e42:400::347
curl.se has IPv6 address 2a04:4e42:600::347



151.101.129.91
151.101.193.91
151.101.1.91
151.101.65.91

HTTP alt-svc

server tells client: there is one or more *alternatives* at "another place"

(possibly using a different HTTP version)

The Alt-Svc: response header

Each entry has an expiry time

Only recognized over HTTPS

curl can save alt-svc alternatives

curl can use previously saved alt-svc alternatives

```
curl --alt-svc altcache.txt https://example.com/
```

The alt-svc cache is a text based readable file

HTTP HSTS

HSTS - HTTP Strict Transport Security

Lets an HTTPS server declare that clients should automatically interact with this host name **using only HTTPS** going forward

The `Strict-Transport-Security`: response header

Only recognized over HTTPS

Each entry has an expiry time

curl can save HSTS data

curl can use previously saved HSTS data

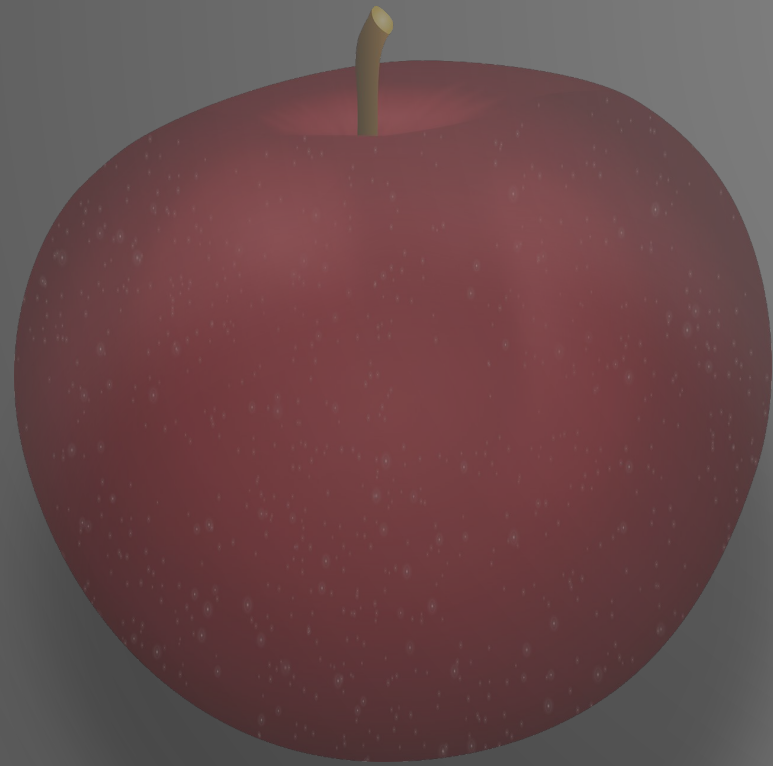
```
curl --hsts hsts.txt http://example.com/
```

The HSTS cache is a text based readable file

FTP

FTP(S) is not SFTP

They are two completely different protocols, both supported by curl



FTP uses two connections

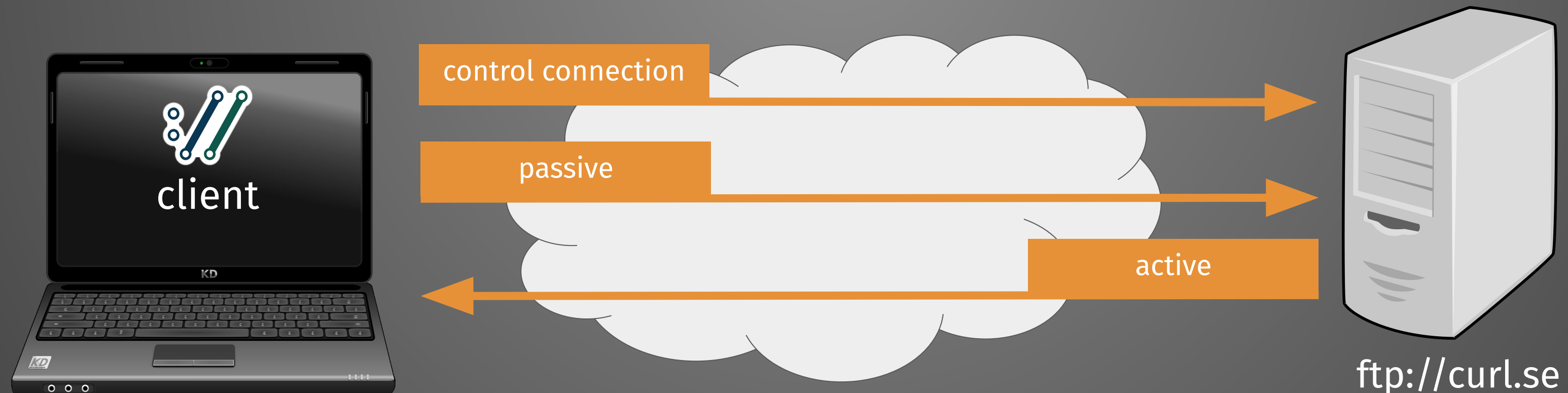
With FTP, a *second* connection is setup for the data transfer

This second connection adds complications for firewalls and more

The 2nd connection is **server-to-client** (active) or **client-to-server** (passive)

Passive is default (`--ftp - pasv`)

Enable active mode with `--ftp-port (-P)`



FTP authentication

by default: user: **anonymous** password: **ftp@example.com**

-u user:password

FTP directory listing

FTP can list the contents of a remote directory

curl does not know what is a directory or not

tell curl with a trailing slash

```
curl ftp://example.com/pub/linux/
```

The list format is not standardized 😞

show only file names with `--list-only (-l)`

```
curl --list-only ftp://example.com/pub/linux/
```

FTP upload

curl -T is for upload

normally requires -u to be allowed

```
curl -T localfile ftp://example.com/pub/linux/newfilename
```

```
curl -T localfile ftp://example.com/pub/linux/
```

append to remote file

```
curl --append -T localfile ftp://example.com/pub/linux/
```

Create dir on the server if missing:

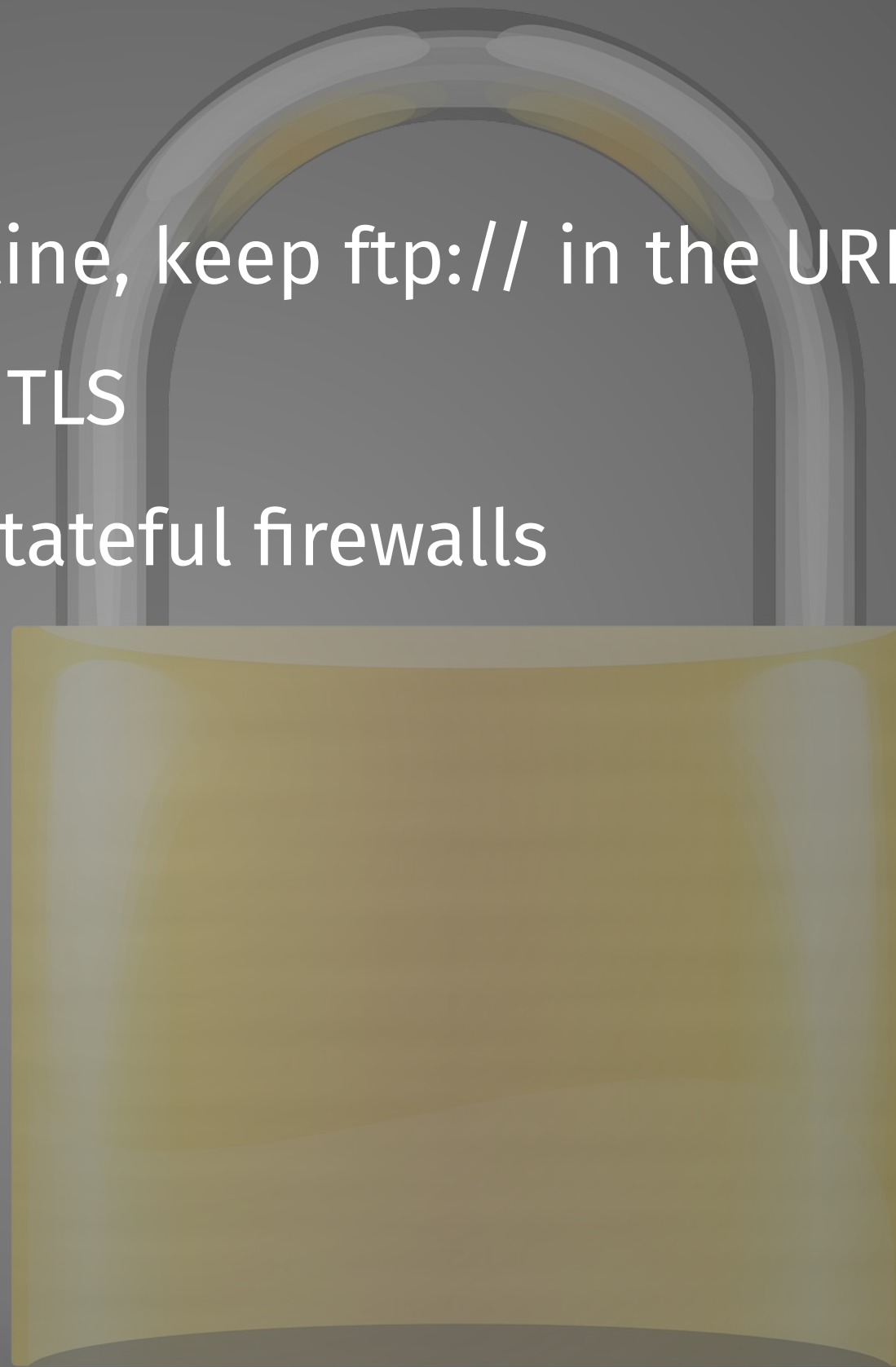
```
curl --ftp-create-dirs -T localfile ftp://example.com/pub/linux/
```


FTPS is FTP with TLS

add `--ssl-reqd` to the command line, keep `ftp://` in the URL

use `FTPS://` if using (rare) implicit TLS

FTPS is even more problematic for stateful firewalls



Future

How to dig deeper

curl is the accumulated results and experiences from 25 years of improvements

man curl

Everything curl

source code

ask the community!



@bagder

Everything

curl!!!

<https://everything.curl.dev/>

The complete guide to all there is
to know about the curl project

Daniel Stenberg

Going next?

curl is 25 years old

curl has been growing and developed its entire lifetime

curl development speed is increasing

the Internet does not stop or slow down

protocols and new ways of doing Internet transfers keep popping up

new versions, new systems, new concepts and new ideas keep coming

there is now slowdown in sight

reasonably, curl will keep develop

curl will keep expanding, get new features, get taught new things

we, the community, make it do what we think it should do

you can affect what's next for curl



You can help!



Thank you!

Questions?

Daniel Stenberg
@bagder@mastodon.social
<https://daniel.haxx.se/>

