

# CS305 Lab Tutorial

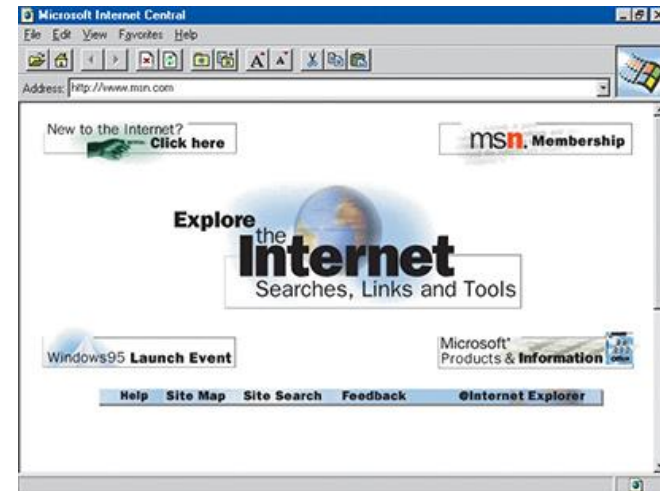
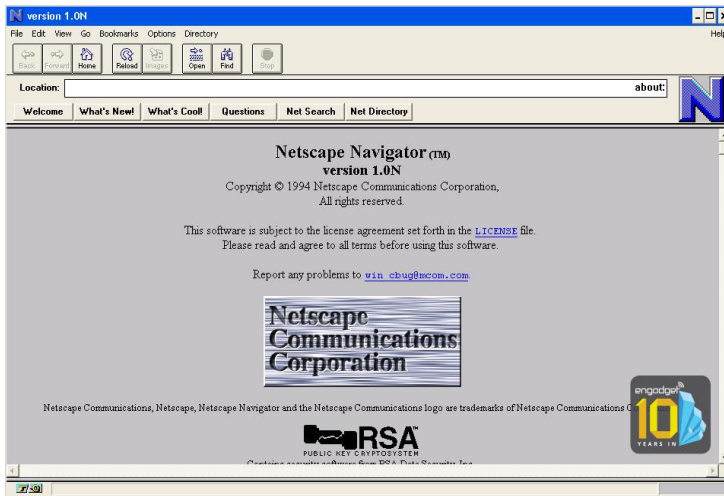
## Lab3 HTTP

Dept. Computer Science and Engineering  
Southern University of Science and Technology

# Web browsers

- A **web browser** is a program to retrieve and display resources on the Web

Example: Netscape 1.0N

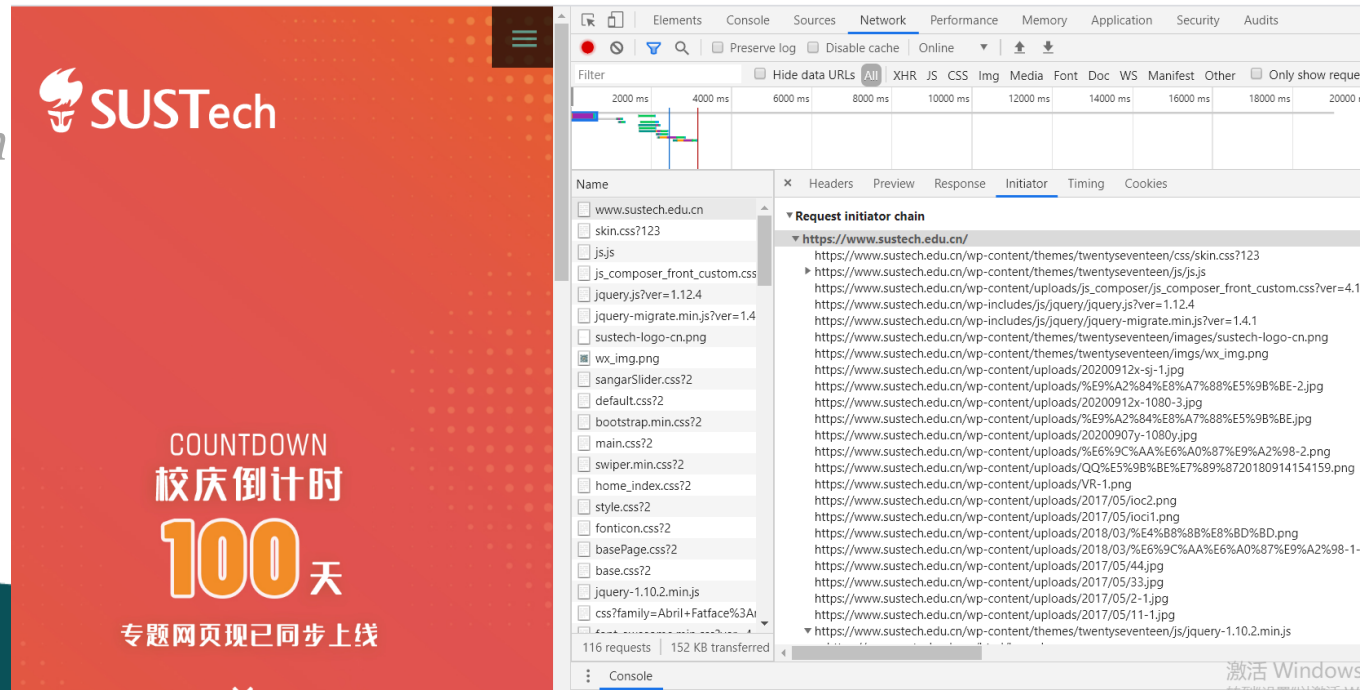


# Web browsers and servers

- Popular web browsers: Google Chrome, Apple Safari, Microsoft Internet Explorer, Mozilla Firefox
  - <https://www.w3counter.com/globalstats.php>
- A **web server** receives requests from a web browser and returns the requested resource.
  - Retrieve the resource from file system on server, or
  - Run a program to generate the resource
- Popular web servers: Microsoft IIS, Apache HTTP server, Nginx
  - <https://news.netcraft.com/archives/2020/08/26/august-2020-web-server-survey.html>

# Web resources in a web page

- A web page actually consists of several kinds of web resources
  - One or more HTML document
  - Some CSS style sheets
  - Some JavaScript programs
  - Images
  - Objects  
(e.g. flash)
  - Font
  - .. etc



# Web resources

- A **resource** is anything that is important enough to be referenced as a thing in itself.
- A resource has at least one **URL** as its address. A browser uses the URL to download the resource from a server.
- The web server uses **MIME type** to specify the data type of a representation of a resource.
  - Examples of resources:
    - The front page of SUSTech web site
    - The logo of SUSTech
    - A directory of resources pertaining to “web design” found by Google.
    - A json record returned by Facebook graph API:  
<https://graph.facebook.com/cocacola>

# Address of web resource

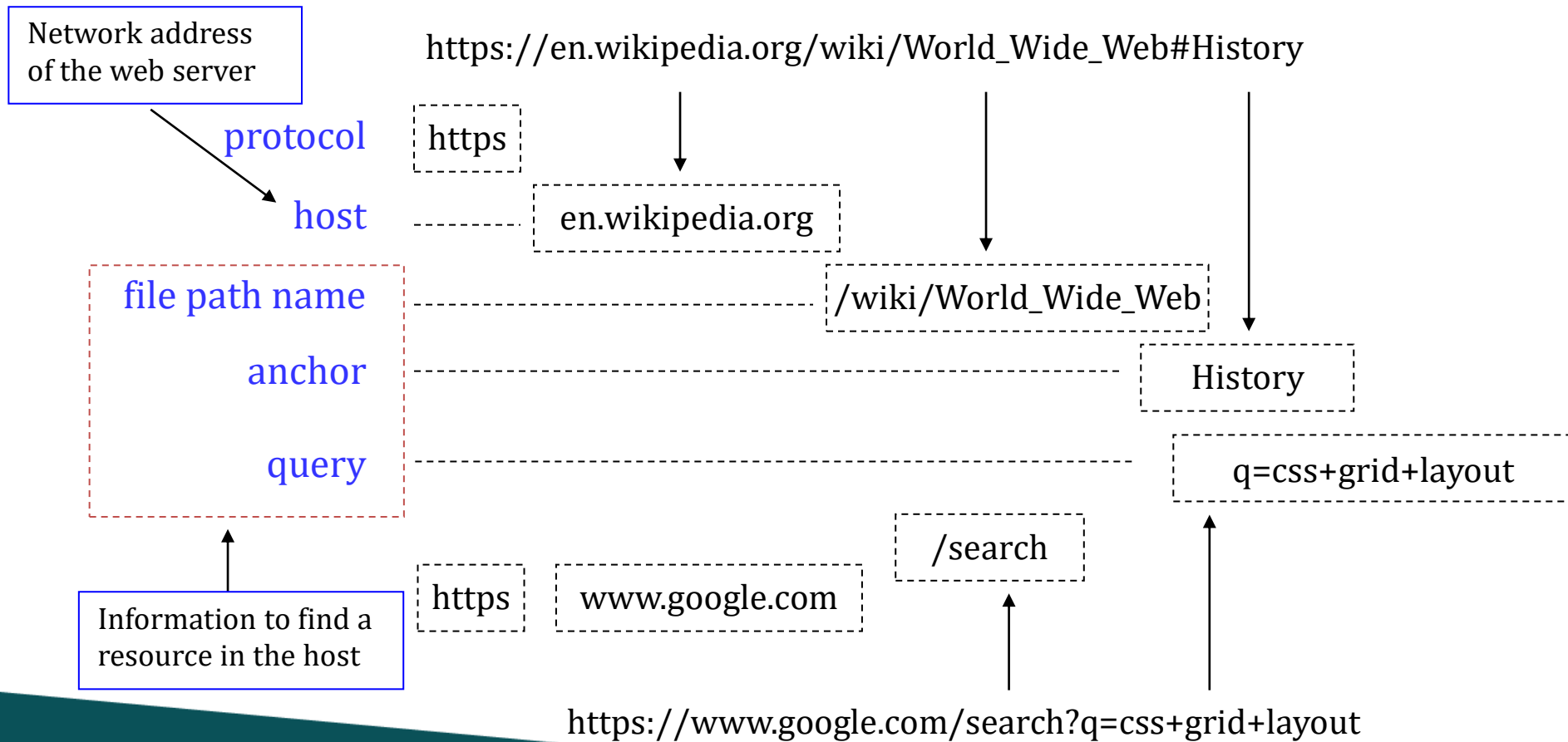
- Each web resource has a **URL** (Universal Resource Locator) as an address. It includes
  - address of a web server – **where to find the resource**
  - protocol – **how to communicate with the server**
  - additional info for the server to find the resource – **which resource in the server**
- Given an URL, a browser has enough info to construct a request to retrieve the resource

# Syntax of URL

`protocol://host/filepathname?query#anchor`

- **Protocol** is either 'http' or 'https'
- **Host** is usually a domain name of a web server
- **File path name** identifies a resource in the server
- **Query** is form data submitted to a server-side script. The format is `name1=value1&name2=value2`
- **Anchor** refers to an element with the id in an html or xml file.
- Notice that separator characters in **blue** must be encoded if they are used as ordinary text in URL.

# Inside an URL ...





# Encode characters in URL

- In general, characters other than letters and numbers should be encoded in URL.
  - Space may be written as '+' or '%20'
  - Other printable character in ASCII should be written in hexadecimal. e.g, 1/2 is encoded as '1%2F2'
  - Non-ascii characters, e.g, Chinese, should be written as utf-8 in hexadecimal. e.g. 中文 is encoded as '%E4%B8%AD%E6%96%87'

# Python urllib

```
C:\Users\Administrator>python
Python 3.8.6rc1 (tags/v3.8.6rc1:08bd63d, Sep 7 2020,
Type "help", "copyright", "credits" or "license" for m
>>> import urllib.parse
>>> urllib.parse.quote('foo=bar')
'foo%3Dbar'
>>> urllib.parse.unquote('foo%3Dbar')
'foo=bar'
>>> obj = { 'name': 'Tonny', 'age': 20 }
>>> urllib.parse.urlencode(obj)
'name=Tonny&age=20'
>>> urllib.parse.parse_qs('name=Tonny&age=20')
{'name': ['Tonny'], 'age': ['20']}
>>>
```

Reference:

<https://docs.python.org/3/library/urllib.parse.html>

# Data type of web resource

- There are various types of web resources
  - E.g. text, image, audio, video, data
- There are different formats to encode a certain type
  - E.g. an image can be in GIF, JPEG or PNG
  - Some formats are defined by W3C, e.g. HTML, CSS, XML (general data), PNG (bitmap image), SVG (vector graphics)
  - Others are de-facto standards defined by the industry, e.g. GIF, JPEG, SWF (flash movie), JavaScript
- These formats are identified by a standard called **MIME type**
- Most browsers support these formats, and can display them correctly.

# Common MIME types

category	MIME type	format
Text	text/html	HTML (.html)
	text/css	CSS (.css)
Images	image/gif	GIF (.gif)
	image/jpeg	JPEG (.jpg)
	image/png	PNG (.png) Portable Network Graphics
	image/svg+xml	SVG (.svg) Scalable Vector Graphics
Audio	audio/mp3, audio/m4a	Mp3 audio, MPEG4 audio
Video	video/mp4, video/m4v	MPEG4 video
Multipurpose	application/xml	General XML
	application/json	JSON data
	application/javascript	JavaScript source (.js)
	application/octet-stream	Arbitrary binary data
	application/x-www-form-urlencoded	HTML form submission

# HTTP operation

- Each HTTP transaction consists of one request and one response
  - A client sends a request for a resource to the server. Then the server returns a response containing a representation of the resource to the client.
- HTTP/1.0 requires a new TCP connection for each HTTP transaction
  - the server closes the connection after sending the response
- HTTP/1.1 can transfer several web resources in a TCP connection

# Basic retrieval

- to retrieve a static file in server
  - A browser sends a request to GET a resource at a URL
  - The server receives the request, maps the URL to a file in its file system, e.g.  
`http://example.com/home.html -> c:\inetpub\home.html`
  - The server infers the MIME-type from the file extension, e.g.  
`*.html -> text/html, *.gif -> image/gif`
  - The server constructs and returns the response



# Example: Basic retrieval

HTTP  
request

```
GET /home.html HTTP/1.1  
Host: example.com  
User-Agent: Mozilla/5.0 ... Chrome/57.0
```



example.com



Simple HTTP retrieval of the page  
`http://example.com/home.html`. Notice how the  
URL of the resource is specified in the request, and  
how the data type is stated in the response.

```
HTTP/1.1 200 OK  
Date: Fri, 18 Sep 2020 08:00:29 GMT  
Server: nginx/1.10.3  
Content-Type: text/html; charset=utf-8  
  
<html>... </html>
```

HTTP response

# Basic HTTP request and response

- A basic request contains
  - Method GET to retrieve a resource
  - URL (or part of URL) – address of the resource
  - Other headers, e.g. name of user-agent
- A basic response contains
  - Status code
  - A representation of the resource
  - its MIME-type and encoding (for text resource) in Content-type header
  - Other headers, e.g. name of server



# HTTP message structure

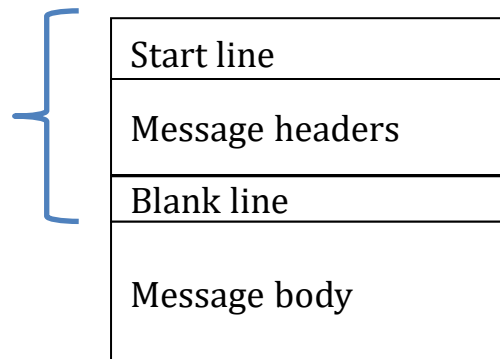
- HTTP requests and responses have similar structure
  - Start line – URL requested, any error
  - Headers – additional info
  - Blank line
  - Body – representation of the resource
- Start line and blank line are required; headers and body are optional.

Start line
Message headers
Blank line
Message body

# HTTP message structure

- Start line and headers must be in US-ASCII
  - I.e. Chinese characters must be encoded
- Message body can be text in any encoding or binary data
  - The header **Content-Type** defines the data type
  - The header **Content-Length** defines the size (in bytes)

*Each line ends with CR LF (ASCII 13 and 10), and can only contain US-ASCII characters.*



*The body can be in any encoding or binary data*

# Request Line

- The request line has three parts:
  - Method name: GET, POST, HEAD, etc
  - (partial) URL of the resource
  - Version of HTTP: HTTP/1.0 or HTTP/1.1

GET /test.html HTTP/1.0

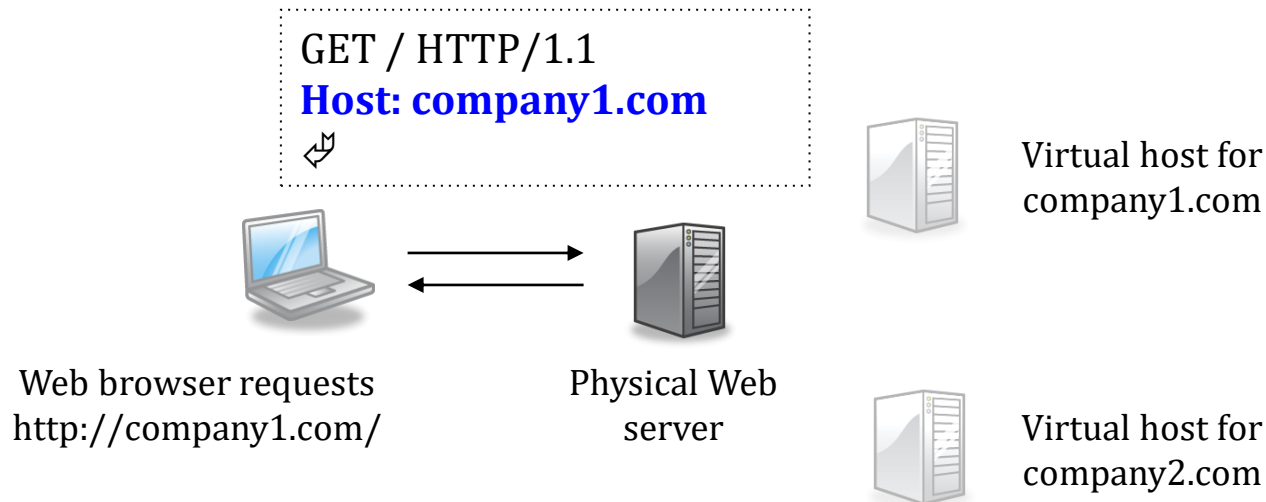
GET http://www.example.com/test.html HTTP/1.0

GET /search?hl=en&q=HTTP+headers&btnG=Google+Search HTTP/1.1

POST /accounts/ServiceLoginAuth?service=mail HTTP/1.1

# Host

- Compulsory header in HTTP/1.1
- Specifies the host of the requested URL
  - Useful for multi-home web servers. Several web sites (with different host names) may be using the same IP address.



# Example: HTTP requests

To retrieve the web page

<http://www.example.com/test.html>

The web browser constructs an HTTP request and sends it to the web server at [www.example.com](http://www.example.com).

```
GET /test.html HTTP/1.1 ↵  
Host: www.example.com ↵  
↵
```

```
GET /test.html HTTP/1.0 ↵  
↵
```

```
GET http://www.example.com/test.html HTTP/1.1 ↵  
Host: www.example.com ↵  
↵
```

# Status line

- The status line has three parts:
  - HTTP version
  - Response status code
    - 200 – success
    - 404 – file not found
    - 500 – server error
  - English reason phrase, which describes the status code
- HTTP defines five classes of status code
  - 1xx – informational message
  - 2xx - success of some kind
  - 3xx – redirects the client to another URL
  - 4xx – an error on the client's part
  - 5xx – an error on the server's part

HTTP/1.1 200 OK

HTTP/1.1 404 Not found

Here you go.  
Go away.  
you screwed up.  
I screwed up.

# Common codes for success and error

Status code	Meaning
200 OK	The server successfully carried out the action that the client requested. For GET request, the response body contains a representation of the requested resource.
204 No content	The server successfully carried out the action, but declined to return any representation. In Ajax, this usually means the browser should not refresh the user interface.
400 Bad request	Generic client-side error. Probably a request format error.
404 Not found	The server cannot find the resource at the requested URL.
500 Internal server error	Generic server-side error. Probably a server-side program run-time error.
503 Service unavailable	The web server is not available, probably because of overloading or maintenance.

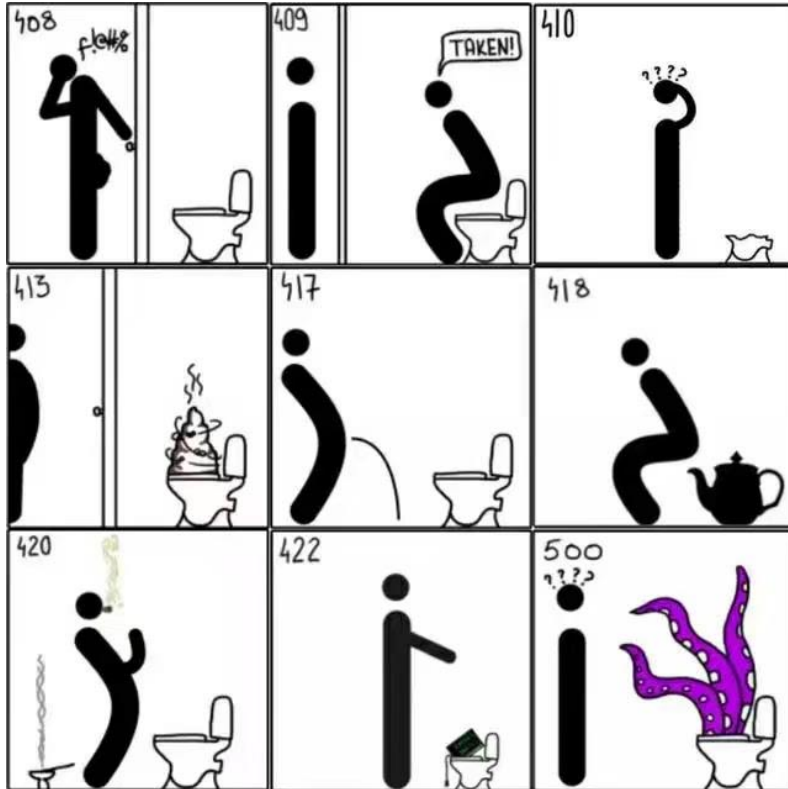
# Some Status Codes(1)



- 301 Moved Permanently
- 302 Found
- 305 Use Proxy
- 307 Temporary Redirect
- 400 Bad Request
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 406 Not Acceptable



# Some Status Codes(2)



- 408 Request Timeout
- 409 Conflict
- 410 Gone
- 413 Payload Too Large
- 417 Expectation Failed
- 418 I'm a teapot
- 420 Enhance Your Calm
- 422 Unprocessable Entity
- 500 Internal Server Error

# Example: HTTP responses

After receiving the request for

<http://www.example.com/test.html>

The web server will return a response containing the page, or an error message if it cannot find the page.

```
HTTP/1.1 200 OK ↵
```

```
Content-Type: text/html ↵
```

```
Content-Length: 2300 ↵
```

```
↵
```

```
<html>... content of test.html ...</html>
```

```
HTTP/1.1 404 Not found ↵
```

```
Content-Type: text/html ↵
```

```
Content-Length: 1024 ↵
```

```
↵
```

```
<html>... error message ...</html>
```

# Content-Type:

- Indicates the data type (MIME) and character encoding of the message body in requests and responses
  - Omitted if empty body
  - **Content-Type: application/octet-stream** if server cannot decide

HTTP request to login mail.com

**POST** /accounts/auth?service=mail HTTP/1.1

Host: www.mail.com

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

**Content-Length: 194**

...&email=username&passwd=password&...

Notice how character encoding is specified.

HTTP response

HTTP/1.1 200 OK

**Content-Type: text/html; charset=UTF-8**

**Content-Length: 12800**

<html>...</html>

# Content-Length:

- Indicates the size (in bytes) of the message body in requests and responses
  - Content-length header is sent before the message body
  - Difficult to determine message size if the response is generated dynamically by a program
  - A solution: chunked-transfer encoding (to be discussed)

HTTP request to login mail.com

**POST** /accounts/auth?service=mail HTTP/1.1

Host: www.mail.com

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

**Content-Length: 194**

...&email=username&passwd=password&...

HTTP response

HTTP/1.1 200 OK

**Content-Type: text/html; charset=UTF-8**

**Content-Length: 12800**

<html>...</html>

# Common headers

Header	Request	Response
Host:	Domain name of requested URL	
User-agent:	Which web browser?	
Server:		Which web server?
Referer:	From which web page does the browser obtain this URL?	
Content-type:	Type of content in the body	
Content-length:	Length of the body in bytes	

# User-Agent: & Server:

- Indicate which web browser generates the request and which web server generates the response
  - May be used for collecting statistics about web browser market share
  - A server may generate browser specific response

```
GET /test.html HTTP/1.1  
Host: httpbin.org  
User-Agent: Mozilla 5.0 (Windows)  
...
```

request

Browser and server names in actual use are much longer.

```
HTTP/1.1 200 Ok  
Server: nginx/1.10  
...
```

response

# Methods in HTTP/1.0

- HTTP methods are operations on resources
- HTTP/1.0 supports three methods
  - GET – to retrieve a representation of a resource
  - HEAD – to retrieve only the metadata of a resource
  - POST – to submit some data to a resource (program) in web server for processing

# GET method

- **GET** retrieves a representation of a resource
  - Usually empty body in request message
  - Browsers mainly use this method to retrieve web resources
  - Usually a read-only operation

```
GET /a.txt HTTP/1.1  
Host: example.com
```

request

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

Hello

response



# HEAD method

- **HEAD** is similar to **GET**, but the response only contains headers and an empty body
- Useful to check the characteristics of a resource without retrieving it, e.g.
  - What is its size?
  - Is the resource still available?

```
HEAD /a.txt HTTP/1.1  
Host: example.com
```

request

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

response

# POST method

- **POST** submits some data to a resource for processing
  - The resource is usually a server-side program
  - The data are usually encoded in the body of the POST request
  - Usually invokes some server-side action

```
POST /accounts/auth?service=mail HTTP/1.1
Host: www.mail.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 194

...&email=username&passwd=password&...
```

HTTP request to login mail.com

# Observation

- The **GET** request in form submission can be ...
  - saved in bookmark
  - saved in browser history (possible security problem!)
  - used in a hyperlink <a href=“..”>. When users click the link, the browser sends the same GET request as in submitting the HTML form.

```
GET /login.php?user=philip&passwd=12345 HTTP/1.1  
Host: example.com
```

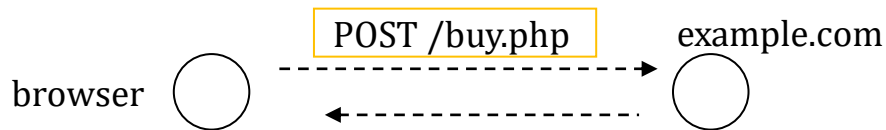
Request for  
<http://example.com/login.php?user=philip&passwd=12345>

# Side-effect

- **GET** and **HEAD** have no side-effect
  - read-only operations in web servers
  - A browser may repeat sending these requests without user confirmation
  - May repeat a **GET** request in case of network error
- **POST** may have side-effect
  - May cause some actions in the server (e.g. adding a record, sending an email, placing an order)
  - A browser must confirm with users before resending such requests

# Example: resubmit a POST request

- In this example, a purchase request is sent as a POST (because it has side-effect).
- The server returns the transaction result in the response
- If the user refreshes the result page, the browser would need to send the POST request twice.
  - What would happen?



HTTP/1.1 200 Ok  
Content-Type: text/html  
Content-Length: 5000

<html>...  
Thanks for your  
purchase! ...</html>

POST /buy.php HTTP/1.1  
Host: example.com  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 194  
  
...&productID=554433&quantity=1&...

# Confirmation when resubmitting request

- Because GET causes no action in the server, it should make no difference to repeat or skip a GET request
  - The browser may resubmit a GET request
- POST may have side-effect. If the browser resubmits a POST request, the server may perform some action twice.
  - The browser must confirm with users before resubmitting a POST request
  - *Users may be confused by the resend warning ... Solution: Post/Redirect/Get pattern*

# Comparison: GET and POST

	GET	POST
Typical usage	Readonly query like Google search, Google Charts	Request that triggers action on server side. E.g. place an order, login, save a file
Side-effect?	Read-only operations in servers. Browsers can resubmit requests without confirmation	May cause write operations in servers. Browsers must confirm with users.
Form data in URL?	Yes. The query URL can be saved in bookmark and hyperlinks.	No
Support file upload	No	Yes

# cURL: Utility for URL

- cURL is a computer software project providing a library and command-line tool for transferring data using various protocols.

**Download: <https://curl.haxx.se/download.html>**

- Example:

```
$ curl http://httpbin.org/headers
```

```
{  
  "headers": {  
    "Accept": "*/*",  
    "Host": "httpbin.org",  
    "User-Agent": "curl/7.55.1"  
    "X-Amzn-Trace-Id": "Root=1-5f646cba-4f1892177757747d3883a5ed"  
  }  
}
```



# cURL: Inspect HTTP transaction

```
$ curl http://httpbin.org/headers --HEAD -v
```

```
> HEAD /headers HTTP/1.1
```

```
> Host: httpbin.org
```

```
> User-Agent: curl/7.55.1
```

```
> Accept: */*
```

```
...
```

```
< HTTP/1.1 200 OK
```

```
< Server: gunicorn/19.9.0
```

```
< Date: Fri, 18 Sep 2020 03:08:58 GMT
```

```
< Content-Type: application/json
```

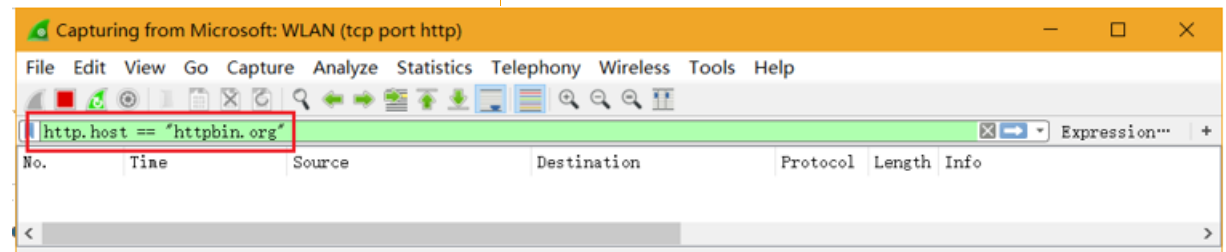
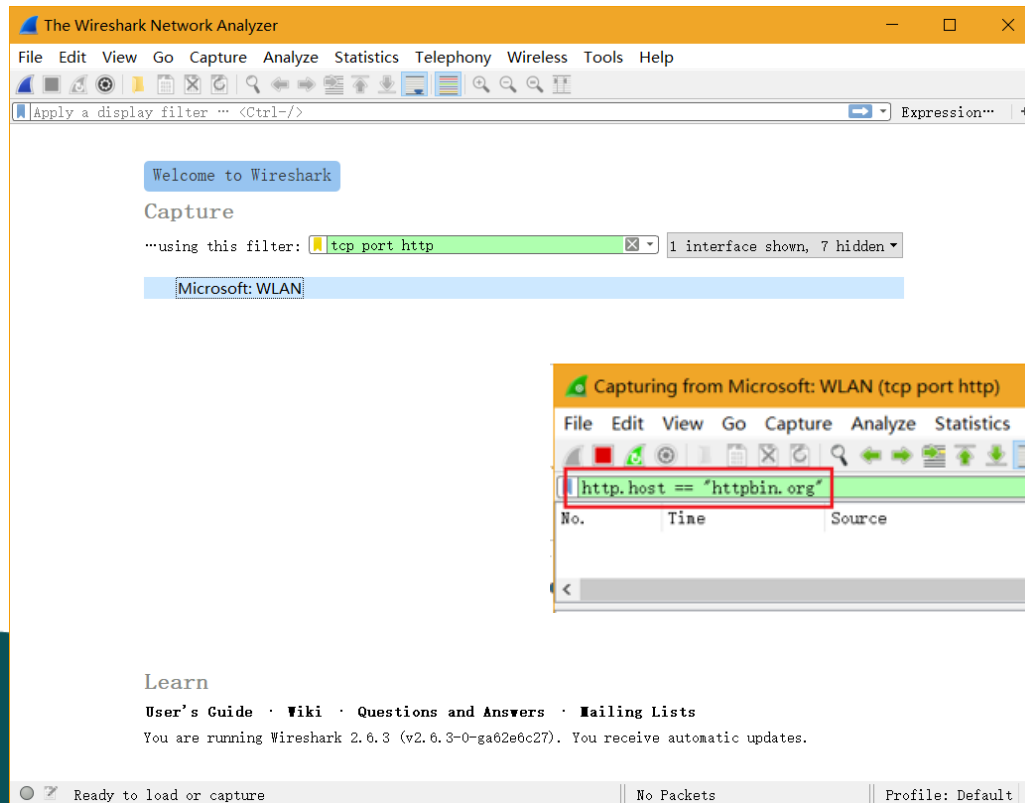
```
< Content-Length: 173
```

```
< Access-Control-Allow-Origin: *
```

```
< Access-Control-Allow-Credentials: true
```

# Wireshark: Capture HTTP Request

- Wireshark is a free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education.



# cURL: Inspect GET request

```
$ curl http://httpbin.org/get
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.55.1"
  },
  "origin": "116.6.234.271",
  "url": "http://httpbin.org/get"
}
```

```
$ curl http://httpbin.org/get?date=2020-09-18
{
  "args": {
    "date": "2020-09-18"
  },
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.55.1"
  },
  "origin": "116.6.234.271",
  "url": "http://httpbin.org/get?date=2020-09-18"
}
```

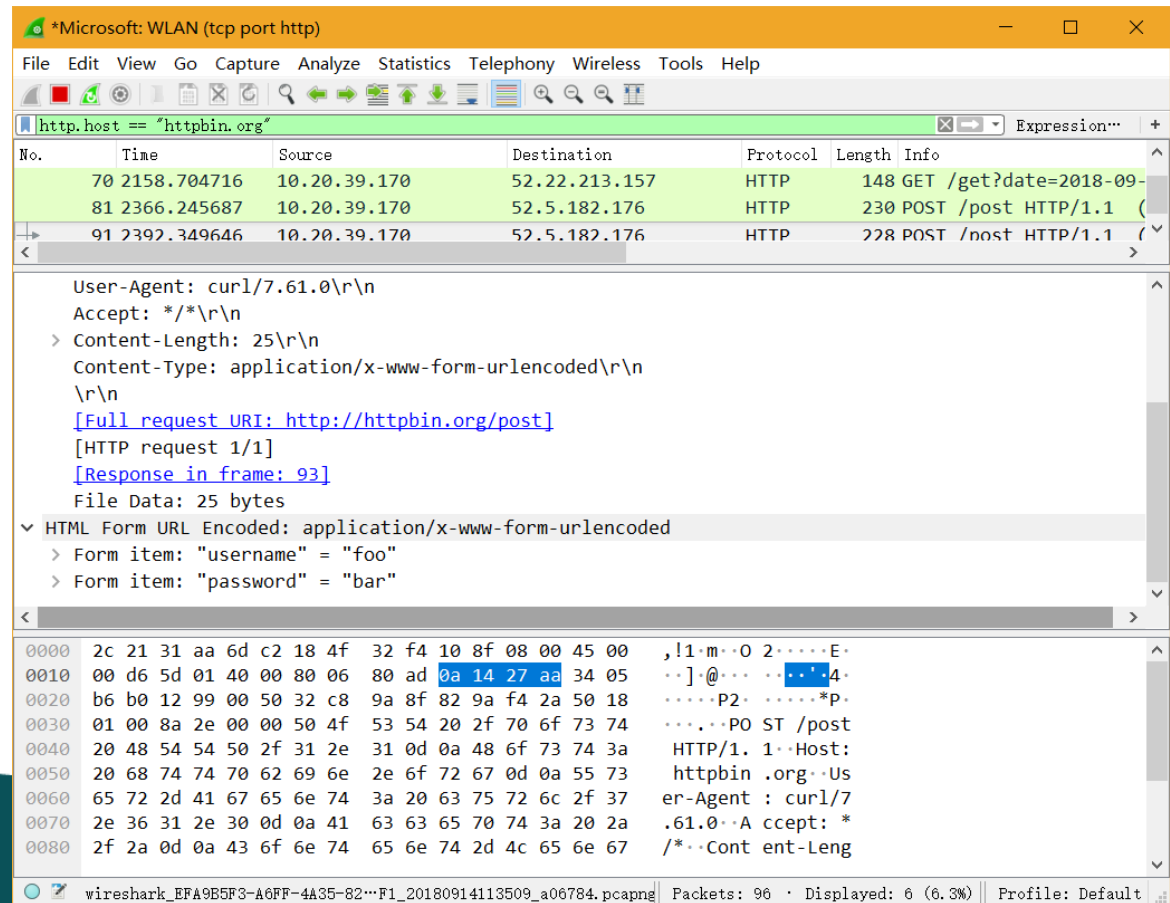
http.host=="httpbin.org"						
No.	Time	Source	Destination	Protocol	Length	Info
→ 1088	17.481147	10.0.0.1	52.6.34.179	HTTP	133	HEAD /get HTTP/1.1
> Frame 1088: 133 bytes on wire (1064 bits), 133 bytes captured (1064 bits) on interface \Device\NPF_{C89FA560-D...} > Ethernet II, Src: IntelCor_5c:69:58 (90:11:24:5c:69:58), Dst: JuniperN_aa:6d:c2 (2c:21:31:aa:6d:c2) > Internet Protocol Version 4, Src: 10.0.0.1, Dst: 52.6.34.179 > Transmission Control Protocol, Src Port: 57693, Dst Port: 80, Seq: 1, Ack: 1, Len: 79 > Hypertext Transfer Protocol > HEAD /get HTTP/1.1\r\n Host: httpbin.org\r\n User-Agent: curl/7.55.1\r\n Accept: */*\r\n \r\n						

http.host=="httpbin.org"						
No.	Time	Source	Destination	Protocol	Length	Info
→ 2689	40.490466	10.0.0.1	3.221.81.55	HTTP	492	GET /get?date=2020-09-18 HTTP/1.1
> Frame 2689: 492 bytes on wire (3936 bits), 492 bytes captured (3936 bits) on interface \Device\NPF_{C89FA560-D...} > Ethernet II, Src: IntelCor_5c:69:58 (90:11:24:5c:69:58), Dst: JuniperN_aa:6d:c2 (2c:21:31:aa:6d:c2) > Internet Protocol Version 4, Src: 10.0.0.1, Dst: 3.221.81.55 > Transmission Control Protocol, Src Port: 57471, Dst Port: 80, Seq: 1, Ack: 1, Len: 438 > Hypertext Transfer Protocol > GET /get?date=2020-09-18 HTTP/1.1\r\n Host: httpbin.org\r\n Connection: keep-alive\r\n Upgrade-Insecure-Requests: 1\r\n User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.398... Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/si... Accept-Encoding: gzip, deflate\r\n Accept-Language: zh-CN,zh;q=0.9\r\n \r\n						

# cURL: Inspect POST request

```
$ curl -d "username=foo&password=bar" -X POST http://httpbin.org/post
```

```
{  
  "args": {},  
  "data": "",  
  "files": {},  
  "form": {  
    "password": "bar",  
    "username": "foo"  
  },  
  "headers": {  
    ...  
  }  
}
```



\*Microsoft: WLAN (tcp port http)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http.host == "httpbin.org" Expression...

No.	Time	Source	Destination	Protocol	Length	Info
70	2158.704716	10.20.39.170	52.22.213.157	HTTP	148	GET /get?date=2018-09-
81	2366.245687	10.20.39.170	52.5.182.176	HTTP	230	POST /post HTTP/1.1 (
91	2392.349646	10.20.39.170	52.5.182.176	HTTP	228	POST /post HTTP/1.1 (

User-Agent: curl/7.61.0\r\n  
 Accept: \*/\*\r\n  
 > Content-Length: 25\r\n  
 Content-Type: application/x-www-form-urlencoded\r\n  
 \r\n  
[\[Full request URI: http://httpbin.org/post\]](http://httpbin.org/post)  
 [HTTP request 1/1]  
[\[Response in frame: 93\]](#)  
 File Data: 25 bytes

HTML Form URL Encoded: application/x-www-form-urlencoded

- > Form item: "username" = "foo"
- > Form item: "password" = "bar"

0000	2c 21 31 aa 6d c2 18 4f 32 f4 10 8f 08 00 45 00	,!1.m..0 2.....E.
0010	00 d6 5d 01 40 00 80 06 80 ad 0a 14 27 aa 34 05	..].@... ..'.4.
0020	b6 b0 12 99 00 50 32 c8 9a 8f 82 9a f4 2a 50 18	.....P2. ....*p.
0030	01 00 8a 2e 00 00 50 4f 53 54 20 2f 70 6f 73 74	....PO ST /post
0040	20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a	HTTP/1. 1..Host:
0050	20 68 74 74 70 62 69 6e 2e 6f 72 67 0d 0a 55 73	httpbin .org..Us
0060	65 72 2d 41 67 65 6e 74 3a 20 63 75 72 6c 2f 37	er-Agent : curl/7
0070	2e 36 31 2e 30 0d 0a 41 63 63 65 70 74 3a 20 2a	.61.0..A ccept: *
0080	2f 2a 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67	/*..Cont ent-Leng

wireshark\_EFA9B5F3-A6FF-4A35-82...F1\_20180914113509\_a06784.pcapng Packets: 96 · Displayed: 6 (6.3%) Profile: Default

# Socket Example: Echo Server

```
import socket

def echo():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('127.0.0.1', 5555))
    sock.listen(10)
    while True:
        conn, address = sock.accept()
        while True:
            data = conn.recv(2048)
            if data and data != b'exit\r\n':
                conn.send(data)
                print(data)
            else:
                conn.close()
                break

if __name__ == "__main__":
    try:
        echo()
    except KeyboardInterrupt:
        pass
```

```
light@DESKTOP-K4SPJVV MINGW64 /c/Users/light/PycharmProjects/CS305-2
$ python echo.py
b'test\r\n'
b'CS305 is Awsome.\r\n'
```

```
light@DESKTOP-K4SPJVV MINGW64 /
$ telnet 127.0.0.1 5555
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
test
test
CS305 is Awsome.
CS305 is Awsome.
exit
Connection closed by foreign host.

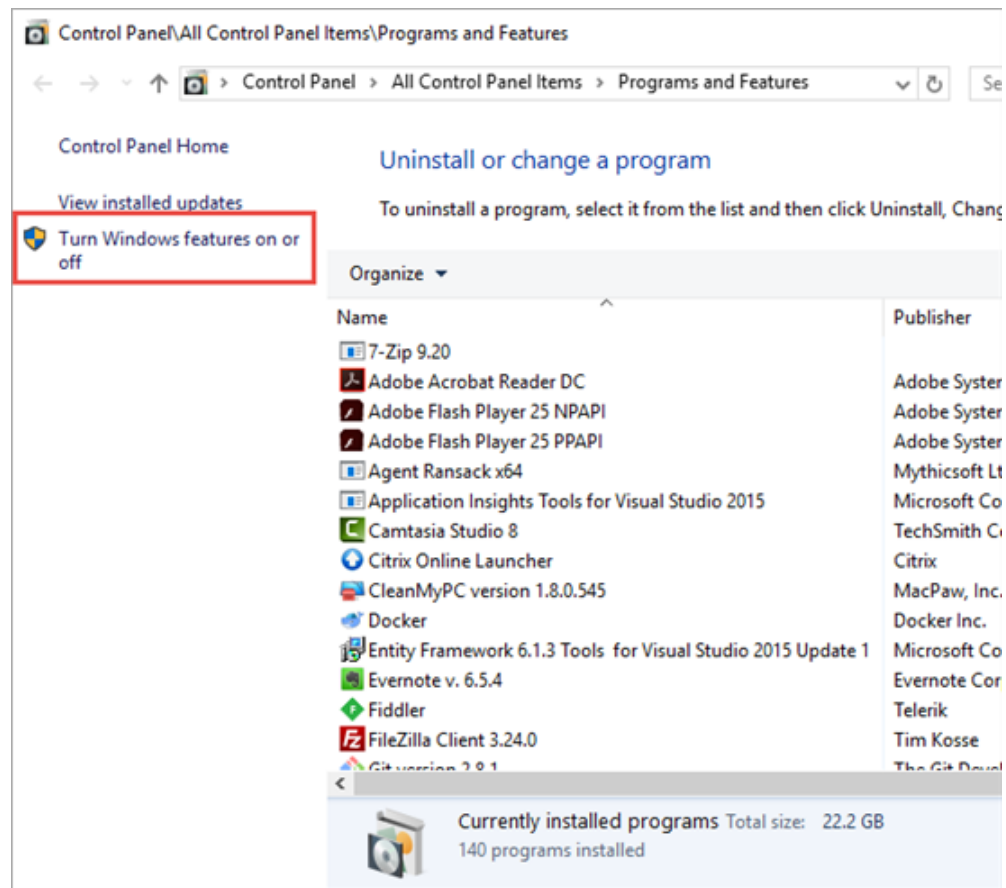
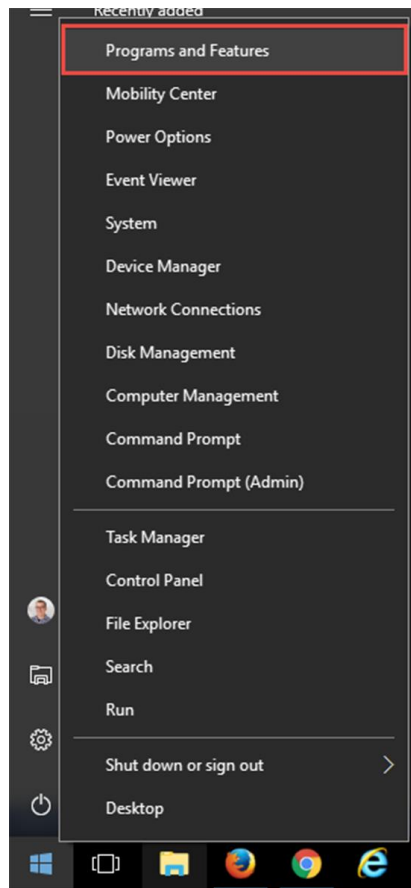
light@DESKTOP-K4SPJVV MINGW64 /
$
```



# practice

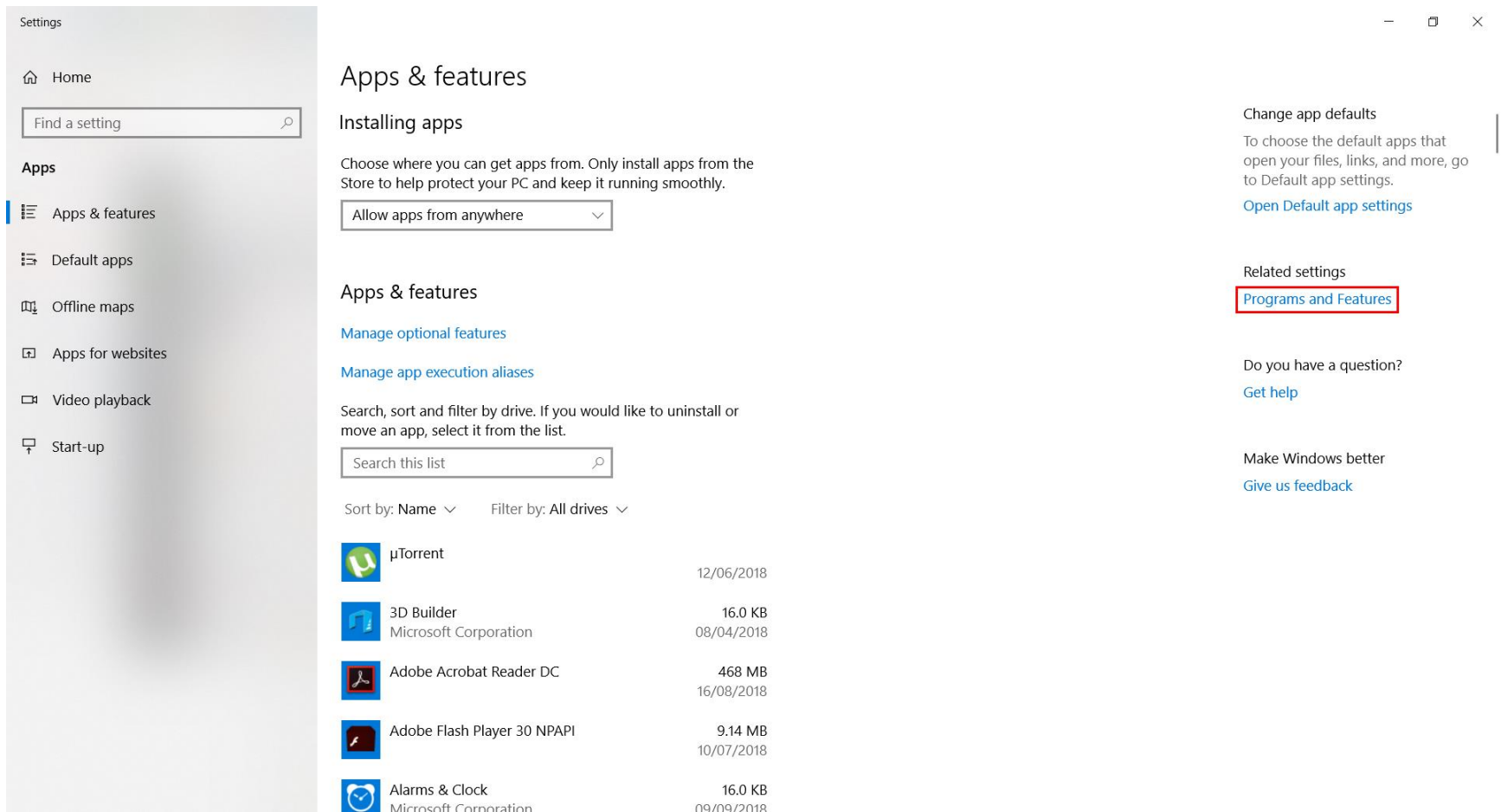
- 1. practice the “echo server.py”
  - modify the code to Match your own OS if needed, to make the server close the connection with the client who send the specified character or the specified string.
  - There are two nested loops in this demo, while server close the connection with client, does the sever exit? is it normal for a server, why?
- 2. envoke a http session with “www.example.com” by curl and web browser seperatly, captrue the packets to find out following information:
  - ip address and ports of both client and server
  - http.host, http.user\_agent, http.server
  - is ther any relationship between ports and http.host, http.user\_agent, http.server ?
- 3. ( optional ) build a web server on your PC, start the “directory browse” service which could visit the specified directory by http service
  - create a subdirectory of the directory, move some files to this subdirectory, including pictures, vedio, audio, text and some files without suffix.
  - visit the web sit, capture the packets to find: what's the MIME of different sources ? what happend while open the file which has no suffix ? is there any http session while change the directory? is there any http request for favicon.ico, what's the corresponding response ?

# TIPS: Enable Telnet on Windows



Reference: <https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx>

# Enable Telnet on Windows



Settings

Home

Find a setting

**Apps**

- Apps & features
- Default apps
- Offline maps
- Apps for websites
- Video playback
- Start-up

## Apps & features

### Installing apps

Choose where you can get apps from. Only install apps from the Store to help protect your PC and keep it running smoothly.

Allow apps from anywhere

### Apps & features






[Manage optional features](#)

[Manage app execution aliases](#)

Search, sort and filter by drive. If you would like to uninstall or move an app, select it from the list.

Search this list

Sort by: Name Filter by: All drives

	uTorrent	12/06/2018
	3D Builder	16.0 KB
	Microsoft Corporation	08/04/2018
	Adobe Acrobat Reader DC	468 MB
		16/08/2018
	Adobe Flash Player 30 NPAPI	9.14 MB
		10/07/2018
	Alarms & Clock	16.0 KB
	Microsoft Corporation	09/09/2018

**Change app defaults**

To choose the default apps that open your files, links, and more, go to Default app settings.

[Open Default app settings](#)

**Related settings**

[Programs and Features](#)

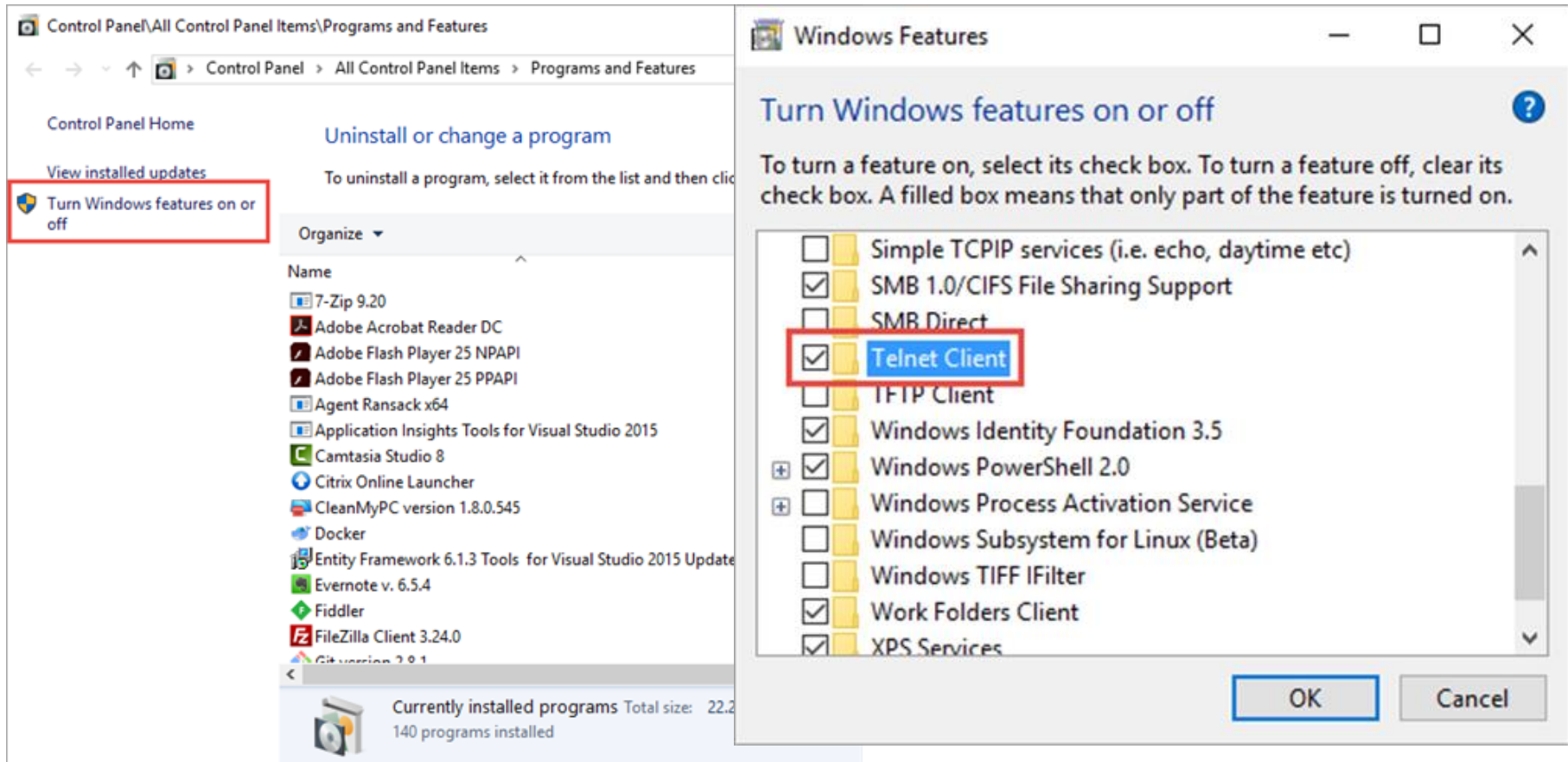
**Do you have a question?**

[Get help](#)

**Make Windows better**

[Give us feedback](#)

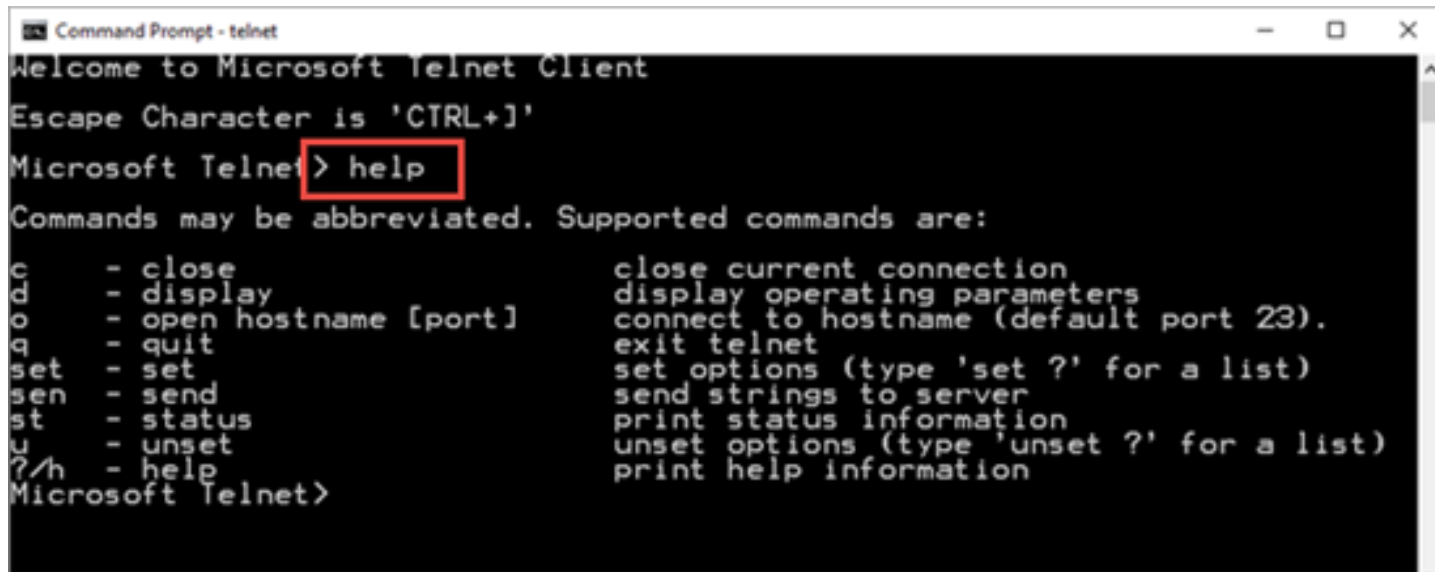
# Enable Telnet on Windows



Reference: <https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx>

# Verify

- Win+R, run “cmd”
- Type “telnet”, enter



```
Command Prompt - telnet
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+\'
Microsoft Telnet> help
Commands may be abbreviated. Supported commands are:
c      - close                close current connection
d      - display             display operating parameters
o      - open hostname [port] connect to hostname (default port 23).
q      - quit                exit telnet
set    - set                 set options (type 'set ?' for a list)
sen    - send                send strings to server
st     - status              print status information
u      - unset               unset options (type 'unset ?' for a list)
?/h    - help                print help information
Microsoft Telnet>
```

Reference: <https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx>