Riferimenti per i comandi usati:

- nc: https://manpages.ubuntu.com/manpages/jammy/man1/nc_openbsd.1.html
- curl: https://manpages.ubuntu.com/manpages/jammy/man1/curl.1.html

Avvertenze

Leggere per intero ciascuno esercizio prima di svolgerlo. Nel farlo, soprattutto quando prevede di inviare più richieste su una connessione persistente, evitare pause eccessivamente lunghe, perché la connessione potrebbe essere chiusa.

Vi occorrerà anche il file *bruti.txt*

Esercizio 1

Eseguire il seguente comando

`nc -C art.uniroma2.it 80`

- `-C` fa in modo che i caratteri `\n` (non preceduti da `\r`) siano convertiti in `\r\n`. I caratteri ricevuti non sono mai trasformati.
- `art.uniroma2.it 80` indicano rispettivamente l'hostname e il numero di porta (80 è la porta *well-known* cui si mettono in ascolto i server web)

scrivere sulla console quanto segue (attenzione all'ultima riga vuota!):

```
GET / HTTP/1.1
Host: art.uniroma2.it
Connection: close

```

Si tratta di una richiesta `GET` per il percorso `/` (la radice della *Document Root*) all'host `art.uniroma2.it`

Avendo scelto di usare la versione 1.1 di HTTP, occorre usare la riga di intestazione `Connection: close` per indicare al server che vogliamo una connessione non persistente.

Si otterrà una risposta simile alla seguente (troncata per motivi di spazio)

```
HTTP/1.1 200
Server: nginx/1.25.3
Date: Tue, 19 Mar 2024 16:13:01 GMT
Content-Type: text/html;charset=UTF-8
Content-Length: 4673
Connection: close
```

```
Set-Cookie: JSESSIONID=89429FC9FAA5ED6A7BDEDF96C8DD203C; Path=/;
HttpOnly

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"><head id="j_idt2">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
[…]
```

La riga di stato ci dice che è una risposta positiva (200). Le righe di intestazione ci dicono rispettivamente che:

- la risposta è stata generata dal proxy Nginx
- la risposta è stata generata in una determinata data e ora
- i dati restituiti sono in formato HTML codificato usando UTF-8
- i dati restituiti sono lunghi 4673 byte
- il server chiude la connessione dopo aver risposto
- il server ha generato il cookie `JSESSIONID` con il valore `89429FC9FAA5ED6A7BDEDF96C8DD203C`. Questo cookie può essere incluso nelle richieste per qualunque path sullo stesso host (si veda l'attributo `Path`) e non è accessibile agli script in esecuzione nel browser (si veda l'attributo `HttpOnly`)

Il comando dovrebbe essere ancora in esecuzione dopo aver ricevuto la risposta. Si può premere `CTRL-D` per impostare la condizione di `EOF` (end-of-file) sullo standard input del processo (oppure, ma da evitare, `CTRL-C` per inviare il segnale SIGINT).

Nel caso si fosse usata una connessione persistente, `CTRL-D` non avrebbe fatto terminare il comando `nc`. In questo caso, si sarebbe potuto usare `CTRL-C` oppure aggiungere al comando `nc` l'opzione `-N`.

L'opzione `-N` fa in modo che `nc` chiuda la propria estremità della connessione quando lo standard input è nella condizione di `EOF` (pertanto non ci sono ulteriori dati da leggere dalla console e quindi inviare attraverso la connessione).

Nei prossimi esercizi verrà **sempre aggiunta** l'opzione `-N`.

Esercizio 2

Usiamo il comando `nslookup` per determinare l'indirizzo IP dell'host `art.uniroma2.it`

```
nslookup art.uniroma2.it
```

Si otterrà una risposta simile alla seguente:

```
Server:         10.255.255.254
Address:        10.255.255.254#53
```

```
Non-authoritative answer:
Name:   art.uniroma2.it
Address: 160.80.84.130
```

L'indirizzo IP cercato è dunque `160.80.84.130`

Eseguire il seguente comando

```
nc -CN 160.80.84.130 80
```

scrivere sulla console quanto segue (attenzione all'ultima riga vuota!). Rispetto all'esercizio precedente è stata tolta la riga di intestazione `Host`:

```
GET / HTTP/1.1
Connection: close

```

Si otterrà la seguente risposta di errore (`400 Bad Request`):

```
HTTP/1.1 400 Bad Request
Server: nginx/1.25.3
Date: Tue, 25 Mar 2025 15:41:38 GMT
Content-Type: text/html
Content-Length: 157
Connection: close

<html>
<head><title>400 Bad Request</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<hr><center>nginx/1.25.3</center>
</body>
</html>
```

Premere `CTRL-D` per chiudere la connessione.

Si esegua nuovamente il comando, usando la seguente richiesta (attenzione alla riga vuota):

```
GET / HTTP/1.1
Host: art.uniroma2.it
Connection: close

```

Premere `CTRL-D` per chiudere la connessione, dopo aver ricevuto la risposta.

Si esegua nuovamente il comando, usando la seguente richiesta (attenzione alla riga vuota):

```
GET / HTTP/1.1
Host: vocbench.uniroma2.it
Connection: close
```

Premere CTRL–D per chiudere la connessione, dopo aver ricevuto la risposta.

È facile rendersi conto che la seconda richiesta ha ottenuto una risposta diversa. Infatti, l'host 160.80.84.130, su cui è in esecuzione un server Web, ha due hostname (art.uniroma2.it e vocbench.uniroma2.it), ciascuno usato per un sito differente: si parla di *name-based virtual hosting*. In questo ed altri casi simili (es. web cache implementate come proxy server), il campo di intestazione Host permette di determinare l'hostname (quindi il sito) contenuto nella URL originale.

Esercizio 3

Usando nuovamente il comando nc, si sfrutterà una connessione persistente per richiedere due oggetti.

Eseguire il seguente comando

```
nc -CN art.uniroma2.it 80
```

scrivere sulla console quanto segue (attenzione all'ultima riga vuota!):

```
GET / HTTP/1.1
Host: art.uniroma2.it

```

Si otterrà una risposta simile alla seguente (troncata per motivi di spazio)

```
HTTP/1.1 200
Server: nginx/1.25.3
Date: Tue, 19 Mar 2024 16:13:01 GMT
Content-Type: text/html;charset=UTF-8
Content-Length: 4673
Connection: close
Set-Cookie: JSESSIONID=89429FC9FAA5ED6A7BDEDF96C8DD203C; Path=/;
HttpOnly

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"><head id="j_idt2">
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
[…]
```

Scrivere nuovamente sulla console, digitando quando segue (si noti la riga di intestazione con cui si chiede al server di chiudere la connessione):

```
GET /fiorelli/ HTTP/1.1
Host: art.uniroma2.it
Connection: close
```

Si otterrà una risposta simile alla seguente (troncata per motivi di spazio):

```
GET /fiorelli/ HTTP/1.1
Host: art.uniroma2.it

HTTP/1.1 200
Server: nginx/1.25.3
Date: Tue, 19 Mar 2024 17:12:55 GMT
Content-Type: text/html;charset=UTF-8
Content-Length: 5236
Connection: close
Set-Cookie: JSESSIONID=E1777A2DDF9B91DB65C9A3BE8BEA780B; Path=/;
HttpOnly

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"><head id="j_idt2">
```

Il comando `nc` dovrebbe essere ancora in esecuzione. Premere `CTRL-D` per farlo uscire.

Si noti che nella seconda richiesta il server ha inviato un cookie con lo stesso nome e valore diverso: il motivo è che la seconda richiesta non ha incluso il cookie generato in precedenza.

Esercizio 4

Eseguire il seguente comando

```
nc -CN art.uniroma2.it 80
```

scrivere sulla console quanto segue (attenzione all'ultima riga vuota!):

```
GET / HTTP/1.1
Host: art.uniroma2.it
Connection: close

```

Si otterrà una risposta simile alla seguente (troncata per motivi di spazio):

```
HTTP/1.1 200
Server: nginx/1.25.3
Date: Tue, 19 Mar 2024 17:17:40 GMT
Content-Type: text/html;charset=UTF-8
Content-Length: 4673
Connection: close
Set-Cookie: JSESSIONID=3F89C01960A2F88037FA17CEE05DC4DD; Path=/;
HttpOnly

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"><head id="j_idt2">
```

Premere `CTRL-D` per far terminare `nc`.

Ripetere la richiesta, aggiungendo però il cookie ottenuto in precedenza in una riga di intestazione.

```
GET / HTTP/1.1
Host: art.uniroma2.it
Cookie: JSESSIONID=3F89C01960A2F88037FA17CEE05DC4DD
Connection: close

```

Si otterrà una risposta simile alla seguente (troncata per motivi di spazio):

```
HTTP/1.1 200
Server: nginx/1.25.3
Date: Tue, 19 Mar 2024 17:20:29 GMT
Content-Type: text/html;charset=UTF-8
Content-Length: 4629
Connection: close

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"><head id="j_idt2">
```

Premere `CTRL-D` per far terminare `nc`. Si noti che nella risposta non è stato incluso alcun cookie, perché il server ha usato quello incluso nella richiesta anziché generarne uno nuovo.

Esercizio 5

Usando il commando `nc`, si invierà due richieste in pipelining. Per evitare di avere una risposta troppo lunga, si userà in questo caso il metodo HEAD.

Nel comando seguente, si noti lo *Here Document* (https://en.wikipedia.org/wiki/Here_document) con cui viene passato l'input al comando `nc` anziché digitarlo sulla console (o leggerlo da file).

Occorre digitare (o copiare & incollare) tutto il testo racchiuso nel rettangolo sottostante. Nel caso della digitazione manuale, la shell dovrebbe mostrare un carattere (di solito >) per indicare la continuazione del comando su più righe.

```
nc -C art.uniroma2.it 80 <<EOF
HEAD / HTTP/1.1
Host: art.uniroma2.it

HEAD /fiorelli/ HTTP/1.1
Host: art.uniroma2.it
Connection: close


EOF
```

Si otterrà una risposta simile alla seguente (troncata per motivi di spazio):

```
HTTP/1.1 200
Server: nginx/1.25.3
Date: Tue, 19 Mar 2024 17:31:48 GMT
Content-Type: text/html;charset=UTF-8
Connection: keep-alive
Set-Cookie: JSESSIONID=B17EFFB5902B84FC7E637B99F66388CB; Path=/;
HttpOnly

HTTP/1.1 200
Server: nginx/1.25.3
Date: Tue, 19 Mar 2024 17:31:48 GMT
Content-Type: text/html;charset=UTF-8
Connection: close
Set-Cookie: JSESSIONID=7ED8C3838AF6908EEC538A5BA8355394; Path=/;
HttpOnly
```

Esercizio 6

Come sottolineato a lezione i dati inclusi nella risposta HTTP non devono essere necessariamente testo ASCII. Non vi mostro qui un esempio con il comando `nc`, perché scrivere in stdout dati binari che sono mostrati sulla console può (temporaneamente) scombussolare la console stessa: https://askubuntu.com/questions/1105348/is-it-safe-to-use-standard-input-output-with-binary-data

Esercizio 7

Il seguente comando stampa l'entità contenuta nella risposta (una pagina HTML) sulla console.

```
curl https://art.uniroma2.it/
```

Esercizio 8

Il seguente comando esegue `curl` in modalità *verbose*: stampa l'entità contenuta nella risposta su `stdout`, mentre scrive su `stderr` i dati di intestazione inviati da `curl` (preceduti da >), i dati di intestazione ricevuti da `curl` (preceduti da <) e altre informazioni prodotte đa `curl` (precedute da `*`).

```
curl -v https://art.uniroma2.it/
```

Esercizio 9

Come prima, ma adesso redirezioniamo stdout e stderr su due file

```
curl -v https://art.uniroma2.it/images/ArtLogo.jpg > ArtLogo.jpg  2>
verbose.txt
```

Esercizio 10

Per salvare i dati ricevuti in un file si può usare l'opzione `-o` seguita dal nome del file

`curl -o Logo-ART.jpg https://art.uniroma2.it/images/ArtLogo.jpg`

Esercizio 11

Usando l'opzione `-O` i dati ricevuti saranno scritti in un file il cui nome viene derivato dalla URL (cosa non possibile, se l'URL termina con /): nell'esempio l'oggetto verrà scritto nel file `ArtLogo.jpg`

`curl -O https://art.uniroma2.it/images/ArtLogo.jpg`

Esercizio 12

Alla richiesta di un file non presente sul sito, curl potrebbe salvare un eventuale messaggio di errore o produrre un file vuoto (a seconda della risposta del server). Usando l'opzione `-f` o `--fail`, curl termina con un exit code diverso da zero e senza scrivere l'output in caso lo stato della risposta del server indica un errore (4xx o 5xx).

Si noti la differenza:

curl https://example.org/non-existing-bla

produce

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
        <head>
                <title>404 - Not Found</title>
        </head>
        <body>
                <h1>404 - Not Found</h1>
                <script type="text/javascript"
src="//obj.ac.bcon.ecdns.net/ec_tpm_bcon.js"></script>
        </body>
</html>
```

Ed eseguendo `echo $?` si ottiene zero

Il seguente comando:

`curl --fail https://example.org/non-existing-bla`

produce

```
curl: (22) The requested URL returned error: 500
```

Ed eseguendo `echo $?` si ottiene 22

Se avessimo usato l'opzione `-o` oppure `-O` non avremmo scritto alcun file.

Esercizio 13

Per impostazione predefinita `curl` non segue i redirect.

`curl -v https://art.uniroma2.it/fiorelli`

stampa quanto segue, senza seguire il redirect.

```
   Trying 160.80.84.130:443...
* Connected to art.uniroma2.it (160.80.84.130) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*  CAfile: /etc/ssl/certs/ca-certificates.crt
*  CApath: /etc/ssl/certs
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: CN=art.uniroma2.it
*  start date: Feb 26 06:04:05 2024 GMT
*  expire date: May 26 06:04:04 2024 GMT
*  subjectAltName: host "art.uniroma2.it" matched cert's
"art.uniroma2.it"
*  issuer: C=US; O=Let's Encrypt; CN=R3
*  SSL certificate verify ok.
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
> GET /fiorelli HTTP/1.1
> Host: art.uniroma2.it
> User-Agent: curl/7.81.0
> Accept: */*
>
```

```
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* Mark bundle as not supporting multiuse
< HTTP/1.1 302
< Server: nginx/1.25.3
< Date: Tue, 19 Mar 2024 18:06:25 GMT
< Content-Length: 0
< Connection: keep-alive
< Location: http://art.uniroma2.it/fiorelli/
<
* Connection #0 to host art.uniroma2.it left intact
```

Aggiungendo l'opzione `-L`, `curl` seguirà i redirect. Per

`curl -v -L https://art.uniroma2.it/fiorelli`

Esercizio 14

Usando l'opzione `-v` si può notare che in ogni risposta il server aggiunge sempre un nuovo cookie.

L'opzione `-b` permette di specificare manualmente un cookie come coppia `chiave=valore` oppure di fornire un file contenente cookie formatto nello stile dell'header `Set-Cookie` oppure nel formato dei file dei cookie di Netscape/Mozilla.

L'opzione `-c` permette a curl di scrivere in un file (nel formato dei cookie di Netscape/Mozilla).

Le due opzioni possono essere combinate. Eseguendo più volte il seguente comando, si vedrà che le richieste successive contengono il cookie generato inizialmente dal server.

`curl -v -b cookie.txt -c cookie.txt https://art.uniroma2.it`

Esercizio 15

L'opzione `-X` permette di specificare il metodo della richiesta.

Il seguente comando invia una POST ad un endpoint del servizio di testing httpbin.org

`curl -v -X POST http://httpbin.org/post`

La risposta include svariate informazioni circa la richiesta effettuata.

Per usare il metodo `HEAD` non si può usare l'opzione `-X` ma l'opzione `-I` (oppure `--head`)

`curl -v -I http://httpbin.org/get`

Se viene fatta una richiesta HTTP con un metodo non supportato, il server risponde con un `405 METHOD NOT ALLOWED`:

```
curl -v -X POST http://httpbin.org/get
*   Trying 100.28.166.39:80...
* Connected to httpbin.org (100.28.166.39) port 80 (#0)
> POST /get HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 405 METHOD NOT ALLOWED
< Date: Fri, 28 Mar 2025 19:20:12 GMT
< Content-Type: text/html
< Content-Length: 178
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Allow: HEAD, GET, OPTIONS
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
* Connection #0 to host httpbin.org left intact
```

Esercizio 16

Il seguente comando invia degli argomenti nella query string della richiesta GET. Si noti l'uso delle virgolette per evitare la shell interpreti il carattere `&`. Le parti evidenziate in giallo mostrano il *percent encoding*: i caratteri non ammessi dalla sintassi delle URL sono prima convertiti in UTF-8 e ciascun byte espresso come `%xx` dove `xx` è il codice esadecimale.

```
curl "http://httpbin.org/get?title=C%26C&gerne=real-time%20strategy"
```

La risposta ci dice che abbiamo fornito due argomenti.

```
{
  "args": {
    "gerne": "real-time strategy",
    "title": "C&C"
  },
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
```

```
    "User-Agent": "curl/7.81.0",
    "X-Amzn-Trace-Id": "Root=1-67e2dabc-0d60a58532ec098c2de6a912"
  },
  "origin": "x.x.x.x",
  "url": "https://httpbin.org/get?title=C%26C&gerne=real-time
strategy"
}
```

Si veda la funzione *encodeURIComponent* in Javascript: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent

Il media type `application/x-www-form-urlencoded` (usato, per esempio, per inviare un form tramite POST) impiega una variante di questa codifica: per esempio, lo spazio viene codificato come `+` anziché `%20`.

Esercizio 17

Usando l'opzione `--data-urlencode` è possibile inviare dati (in formato `application/x-www-form-urlencoded`) con una richiesta `POST`, applicando la codifica in modo automatico. Si noti l'uso delle virgolette per evitare la shell interpreti il carattere `&`.

```
curl --data-urlencode "title=C&C" \
     --data-urlencode "genre=real-time strategy" \
     "http://httpbin.org/post"
```

La risposta ci indica che il `Content-Type` della richiesta era `application/x-www-form-urlencoded` e ci restituisce i valori passati come fossero stati campi di un form.

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "genre": "real-time strategy",
    "title": "C&C"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "36",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.81.0",
    "X-Amzn-Trace-Id": "Root=1-67e31645-38555fda121234bf609224fa"
  },
  "json": null,
  "origin": "x.x.x.x",
  "url": "http://httpbin.org/post"
}
```

Esercizio 18

L'opzione `--data-binary` permette di inviare con una POST dati senza alcuna interpretazione. Sebbene il content type predefinito sia ancora `application/x-www-form-urlencoded`, usando l'opzione `-H` è possibile impostare la riga di intestazione `Content-Type` a un valore diverso.

Il seguente commando invia la stringa abc al server nel corpo di una richiesta POST.

```
curl --data-binary "abc" \
     -H "Content-Type: text/plain" \
     http://httpbin.org/post
```

Il seguente comando invia il contenuto del file *bruti.txt* (si noti la @ all'inizio dell'argomento del parametro che lo qualifica come nome di file) al server nel corpo di una richiesta POST.

```
curl --data-binary "@bruti.txt" \
     -H "Content-Type: text/plain"
     http://httpbin.org/post
```

Esercizio 19

Usando l'opzione `--form` (oppure `-F`), è possibile inviare dati con una POST in formato `multipart/form-data`

L'opzione va ripetuta per ogni campo del form.

```
curl --form "titolo=dante" --form "file1=@bruti.txt" --form
"file2=@bruti.txt;type=text/plain" --form "text=<bruti.txt"
http://httpbin.org/post
```

In questo esempio, abbiamo i seguenti campi:

- *titolo*
- *file1*: file allegato di tipo *text/plain*
- *file2*: file allegato *text/plain* (dichiarato esplicitamente)
- *text*: contenuto preso da file senza produrre un file allegato

Questa mediatype codifica i vari campi come un flusso di byte separato da un separatore specificato nell'header della richiesta (scelto in modo che non compaia nei dati incapsulati!).