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 di richiede di inviare più richieste su una connessione persistente, evitare pause eccessivamente lunghe, perché la connessione potrebbe essere chiusa.

Vi occorreranno anche i file *form5.txt* e *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 `\n\r`. 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 web 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 inviare un carattere `EOF` (oppure, ma da evitare, `CTRL-C` per inviare il segnale SIGINT, che per impostazione predefinita termina il processo che lo riceve).

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

Esercizio 2

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

Eseguire il seguente comando

```
nc -C 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 3

Eseguire il seguente comando

`nc -C 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 4

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 5

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 6

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

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

Esercizio 7

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 da `curl` (precedute da *).

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

Esercizio 8

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 9

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 10

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 /)

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

Esercizio 11

Alla richiesta di un file non presente sul sito, curl potrebbe salvare un eventuale messaggio di errore o produrre un file vuota (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.

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 12

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 13

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 14

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 -X POST https://httpbin.org/post
```

La risposta include svariate informazioni circa la richiesta effettuata.

Esercizio 15

Il seguente commando invia degli argomenti nella query string della richiesta GET. Si noti l'uso delle virgolette per evitare la shell interpreti il carattere &.

```
curl "https://httpbin.org/get?nome=Duffy&cognome=Duck"
```

La risposta ci dice che abbiamo fornito due argomenti.


```
{
  "args": {
    "cognome": "Duck",
    "nome": "Duffy"
  },
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.81.0",
    "X-Amzn-Trace-Id": "Root=1-65f9d6e2-4cc852692b5dd619501a69a0"
  },
  "origin": "79.36.223.201",
  "url": "https://httpbin.org/get?nome=Duffy&cognome=Duck"
}
```


Esercizio 16

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

```
curl -d"nome=Duffy&cognome=Duck" https://httpbin.org/post
```

La risposta ci fornisce 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": {
    "cognome": "Duck",
    "nome": "Duffy"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "23",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.81.0",
    "X-Amzn-Trace-Id": "Root=1-65f9d78e-1c8345824b07757504b81c37"
  },
  "json": null,
  "origin": "79.36.223.201",
  "url": "https://httpbin.org/post"
}
```


Esercizio 17

L'opzione -d può leggere i dati da un file (preceduto da @).

```
curl -d @form5.txt https://httpbin.org/post
```

Esercizio 18

L'opzione --data-binary permette di inviare con una POST dati senza alcuna interpretazione. Sebbene in 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.

```
curl --data-binary @bruti.txt -H "Content-Type: plain/text"
https://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=plain/text" --form "text=<bruti.txt"
https://httpbin.org/post
```

In questo esempio, abbiamo i seguenti campi:

- *nome*, di tipo stringa

- *file1*, con associato un file di tipo plain/text
- *file2*, con associato un file di tipo plain/text (dichiarato esplicitamente)
- *text*, di tipo stringa ma con contenuto preso da file

Questa mediatype codifica i vari campi come un flusso di byte separato da un separato specificato nell'header della richiesta. Ciò permette di caricare dati binari senza codifiche complicate.