Esercitazione:
Aspetti avanzati della shell Linux

Corso di Laurea in Ingegneria Informatica

Università degli Studi di Napoli Federico II

Anno Accademico 2024/2025, Canale San Giovanni

Sommario della lezione

- Sommario della lezione:
  - Canali di I/O e redirezione
  - Pipe
  - Shell scripting
  - Variabili d'ambiente
  - Processi in background
  - Segnali UNIX
- Riferimenti
  - Ancillotti, Boari, Ciampolini, Lipari, "*Sistemi Operativi*", 2a ed., Appendice C: "***Linguaggio di comandi (shell) in UNIX***"
  - Dispensa didattica su Unix
  - Esempi su https://github.com/rnatella/so_esempi

Perché studiare la shell?

- Elencare gli utenti (indirizzo IP) che si collegano più spesso ad un server web
- Trovare le 10 cartelle che consumano più spazio su disco
- Convertire un gruppo di immagini in un altro formato
- Scambiare messaggi con un bot Telegram
- Riavviare un server web
- Automatizzare il backup
- Creare un "container" Docker
- ...

Varianti della shell

Questa lezione

KORNSHELL

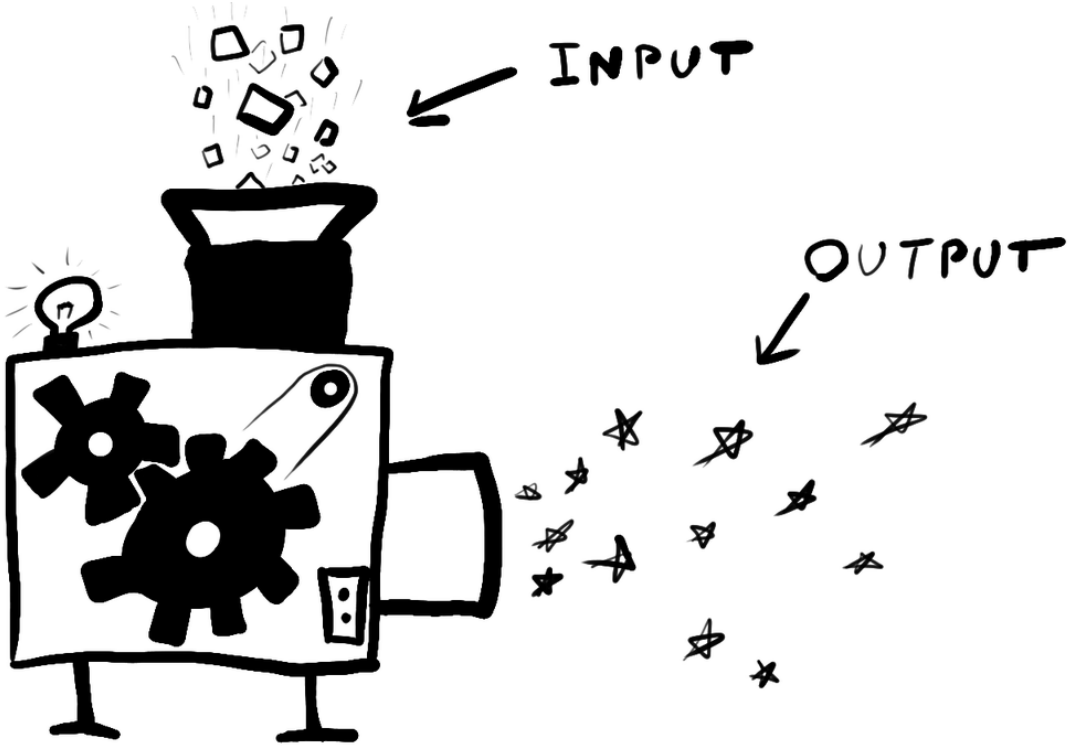
Microsoft PowerShell

csh & tcsh

La "filosofia" di UNIX

- Il principio fondante di UNIX è di collegare tanti piccoli programmi per creare workflow complessi
- Ogni comando ha in ingresso un flusso di caratteri...
- ...e produce in uscita un altro flusso di caratteri

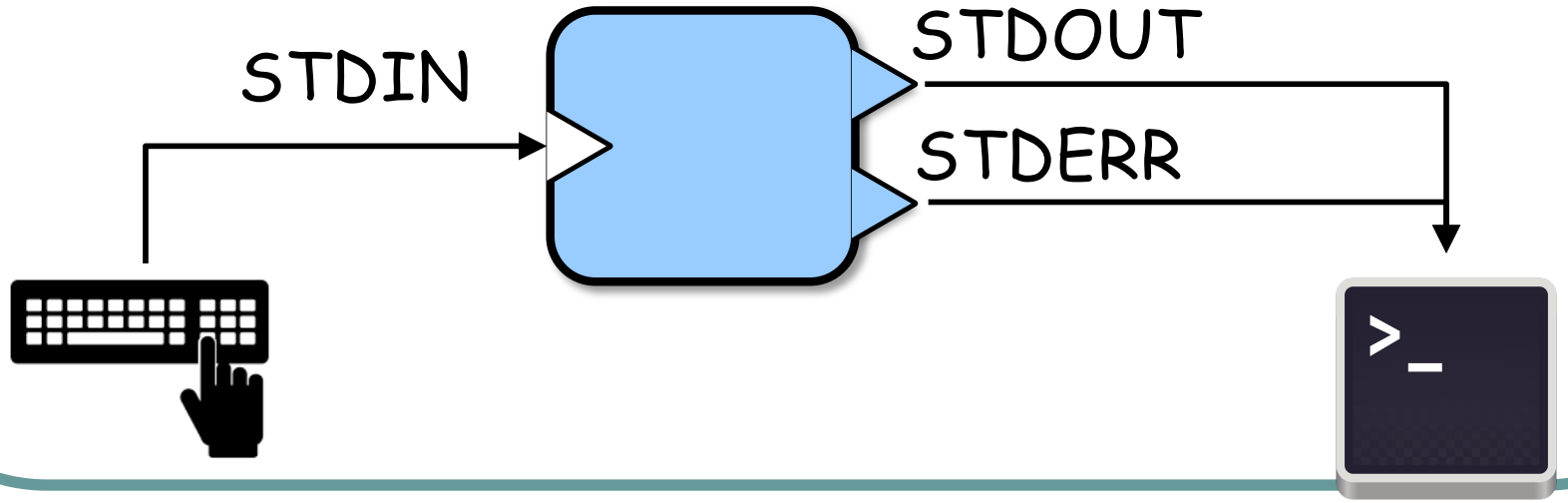
La "filosofia" di UNIX



https://learn.adafruit.com/basic-shell-magic

Canali di I/O dei processi

- Di default, ogni processo ha accesso ad almeno tre canali di I/O
  - **STDIN** (standard input, 0) – lettura da tastiera
  - **STDOUT** (standard output, 1) – scrittura a video
  - **STDERR** (standard error, 2) – scrittura a video



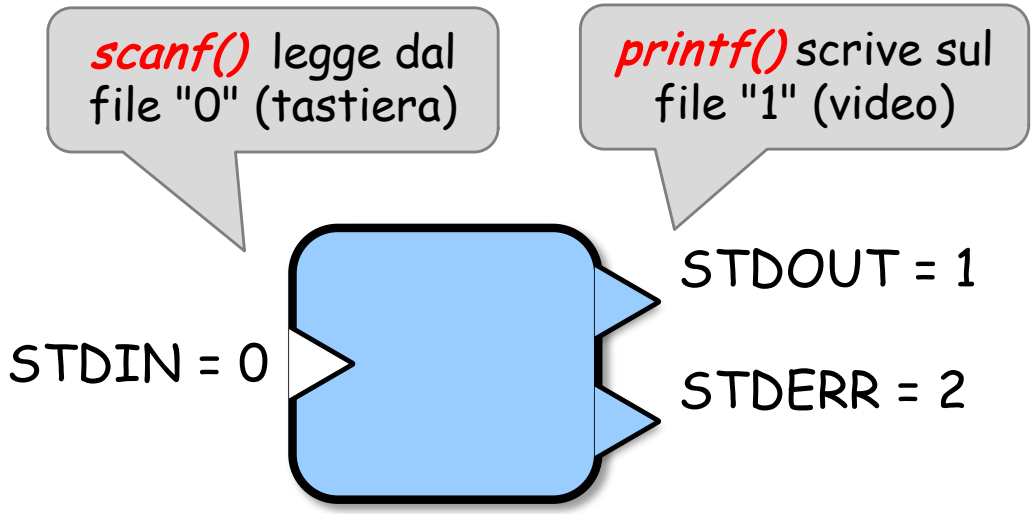
Canali di I/O dei processi

La **tabella dei file aperti** abbina un numero ("handle", "descrittore") ad ogni canale/file acceduto dal processo

File Handle Table del processo (nel kernel)

Handle	File
0	<tastiera>
1	<video terminale>
2	<video terminale>
3	/home/user/hello.txt
4	...

Canali di I/O dei processi



File Handle Table del processo (nel kernel)

Handle	File
0	<tastiera>
1	<video terminale>
2	<video terminale>
3	/home/user/hello.txt
4	...

Un processo può leggere/scrivere su tastiera e video **come se scrivesse un normale file su disco (stesse system call)**

Esempio di STDIN ed STDOUT

```
$ cat stdio.c
#include <stdio.h>

int main() {

        int numero;

        printf("Inserisci un numero: ");
        scanf("%d", &numero);
        printf("Hai inserito il numero: %d\n", numero);

        return 0;
}
```

```
$ gcc stdio.c -o stdio
$ ./stdio
Inserisci un numero: 2
Hai inserito il numero: 2
```

Esempio di STDIN ed STDOUT

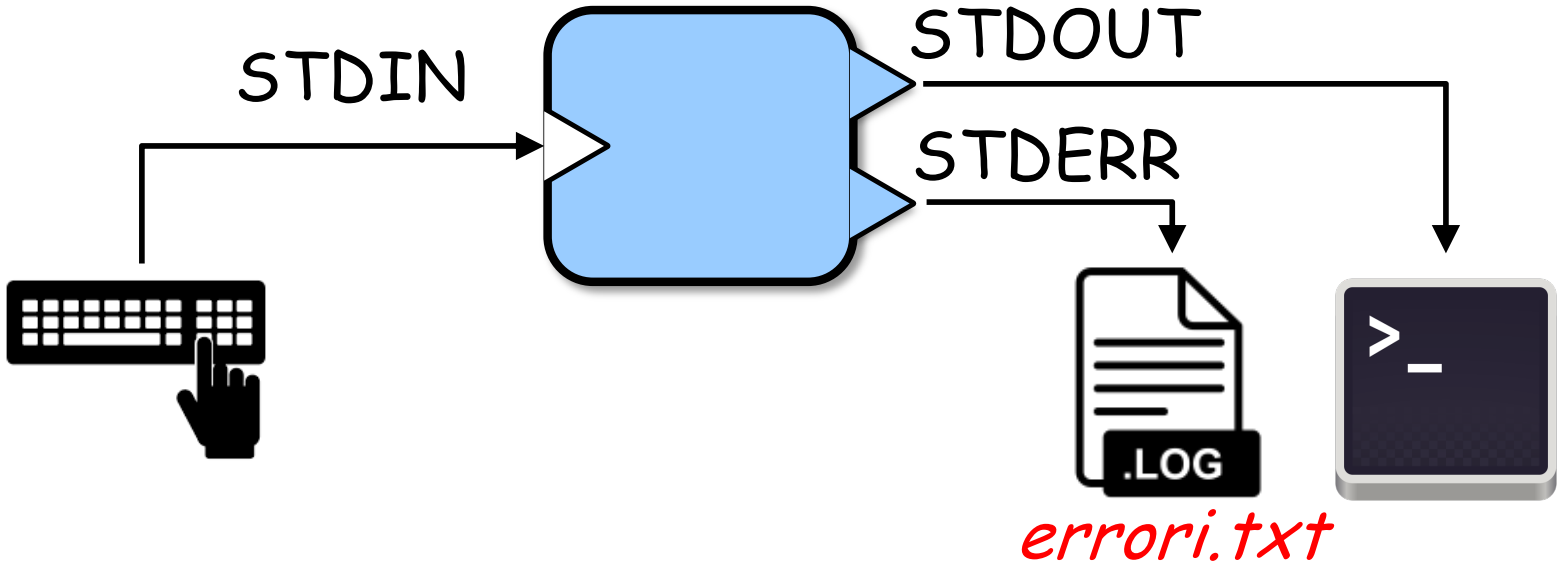
"*strace*" esegue "*stdio*" in un processo figlio.

Ne registra le chiamate di sistema, usando la system call *ptrace()*.

```
$ strace ./stdio
...
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
brk(NULL)                                = 0x55b95ce88000
brk(0x55b95cea9000)                      = 0x55b95cea9000
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "Inserisci un numero: ", 21)    = 21
read(0, "2\n", 1024)                     = 2
write(1, "Hai inserito il numero: 2\n", 26) = 26
lseek(0, -1, SEEK_CUR)                   = -1 ESPIPE (Illegal seek)
exit_group(0)                            = ?
+++ exited with 0 +++
```

STDOUT vs STDERR

- STDOUT: per i **normali messaggi** del programma
- STDERR: per gli eventuali **messaggi di errore**
- Si può **redirigere** i due tipi di messaggi su diverse destinazioni



http://www.treccani.it/vocabolario/redirigere_(Neologismi)

Esempio di STDERR

```
$ cat notfound.c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

int main() {

 int fd = open("/home/file_che_non_esiste.txt", O_RDONLY);

 if(fd < 0) {
    perror("Non riesco ad aprire il file");
    exit(1);
 }

 return 0;
}
```

NOTA: la convenzione in UNIX è che, in caso di errori, un comando esca con un codice numerico diverso da 0.

Equivale a "exit(0)".

```
$ strace -o systemcalls.txt -- ./notfound
Non riesco ad aprire il file!
```

Esempio di STDERR

Lo handle 3 è un "duplicato" del 2 (stderr)

```
$ cat systemcalls.txt
...
dup(2)                                  = 3
fcntl(3, F_GETFL)                       = 0x80002 (flags O_RDWR|O_CLOEXEC
getrandom("\xef\xc6\x0a\xf4\x87\x80\xbb\x2e", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x559185e8e000
brk(0x559185eaf000)                     = 0x559185eaf000
newfstatat(3, "", ...) = 0
write(3, "Non riesco ad aprire il file: No such file ...", 56) = 56
close(3)                                = 0
exit_group(1)                           = ?
+++ exited with 1 +++
```

Redirezione dell'I/O (STDOUT)

```
$ ls > cartelle.txt
$
$ cat cartelle.txt
cartelle.txt
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos

$ ls > /dev/null
$
```

Il simbolo ">" redirige i dati in uscita.

L'output è salvato su file (non appare a video).

UNIX ha un file fittizio "/dev/null".

La redirezione "silenzia" il comando.

Redirezione dell'I/O (STDIN)

Il comando "**echo**" stampa una stringa

```
$ echo "4" > input.txt
$
$ ./stdio < input.txt
Inserisci un numero: 4
Hai inserito il numero: 4
```

Il simbolo "**<**" redirige i dati in ingresso (da file invece che dalla tastiera).

Redirezione dell'I/O (STDERR)

```
$ ./notfound  2> errori.txt
$
$ cat errori.txt
Non riesco ad aprire il file!
```

Il numero (opzionale) prima del simbolo ">" indica il canale da redirigere.

"1" = STDOUT (il default)

"2" = STDERR

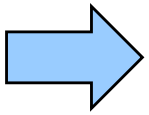
Redirezione dell'I/O nella shell

La shell effettua la redirezione dopo la fork(), e prima di exec()

```
int pid = fork();              // crea il figlio
if(pid == 0) {                 // il figlio continua qui
   close(STDOUT_FILENO);
   open("./file.out", ...);
   execl("program", arg0, arg1, arg2, …);
}
```

File Handle Table (padre)

Handle	File
0	stdin
1	stdout
2	stderr
...	...

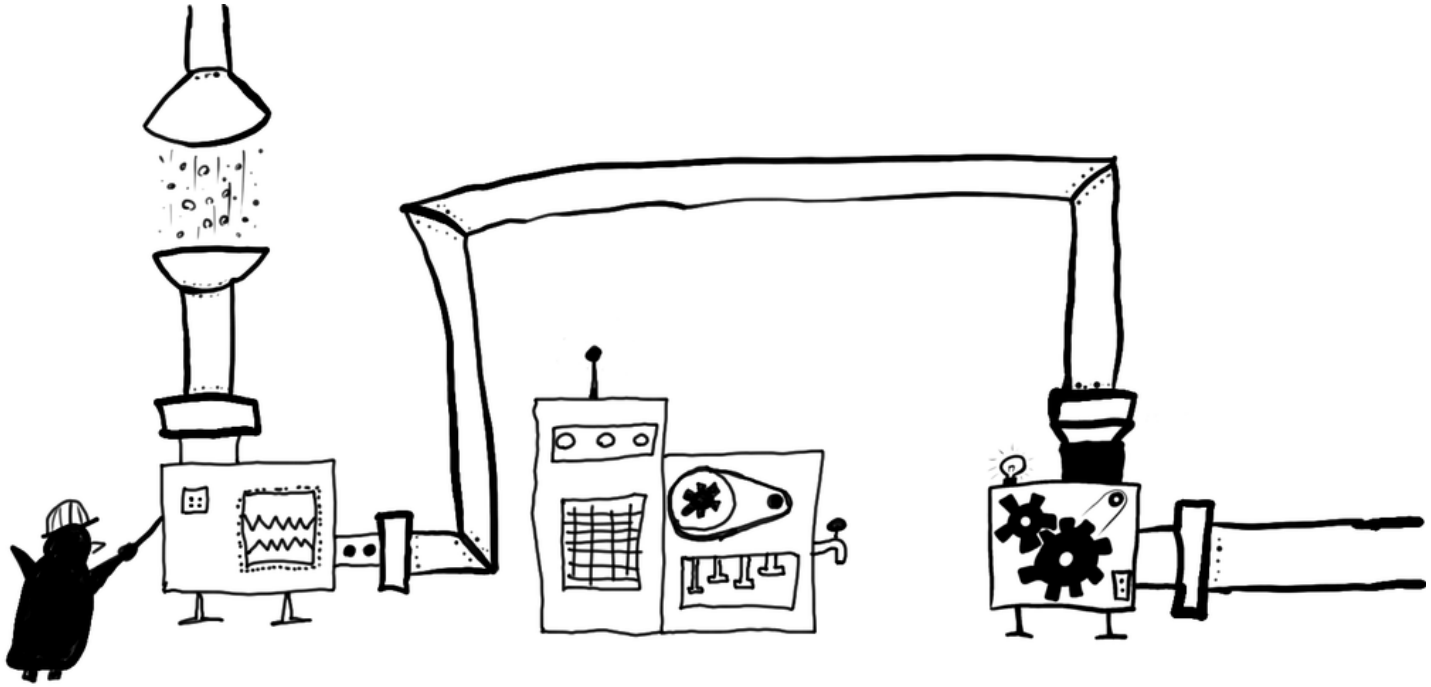


File Handle Table (figlio)

Handle	File
0	stdin
1	/home/user/file.out
2	stderr
...	...

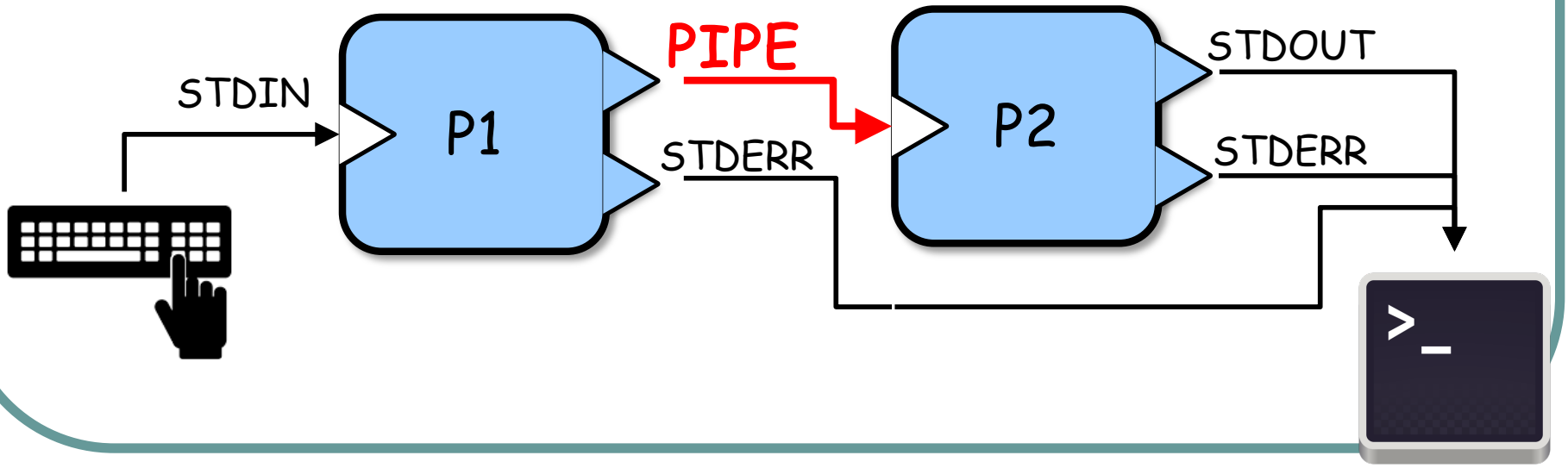
Pipe

- Il meccanismo di UNIX con cui poter collegare più programmi prende il nome di **pipe**

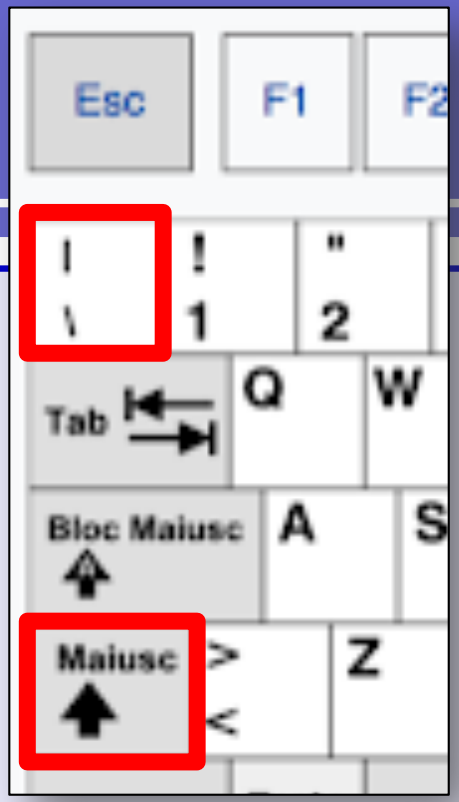


Pipe

- Il meccanismo delle **pipe** redirige i canali di I/O verso un altro processo
- Consente di combinare più comandi



Esempio di pipe




```
$ ls /bin
[
2to3
2to3-2.7
411toppm
a2x
aa-enabled
aa-exec
ab
...

$ ls /bin | grep "yes"
xeyes
yes
```

Il simbolo "|" indica che i due comandi vanno **concatenati in pipe**.

Il comando "**grep**" legge le righe dallo STDIN, e stampa sullo STDOUT le sole righe contenenti una certa **sottostringa**.

In questo esempio, "grep" legge dallo **output di "ls"**.

Esempio avanzato di pipe: top-3 client che accedono ad un sito web

https://github.com/acaudwell/Logstalgia/raw/master/data/example.log

```
...
192.169.133.249          - - [22/Apr/2009:18:53:04 +1200] "GET /adclick.php?..." ...
192.169.133.249          - - [22/Apr/2009:18:53:04 +1200] "GET /latest.php?..." ...
192.196.179.98           - - [22/Apr/2009:18:53:04 +1200] "GET /gallery.php?..." ...
proxy-367.bellsouth.net  - - [22/Apr/2009:18:53:04 +1200] "GET /images/photos/455.jpg ..." ...
bot-332.ihug.co.nz       - - [22/Apr/2009:18:53:04 +1200] "GET /adclick.php?..." ...
proxy-367.bellsouth.net  - - [22/Apr/2009:18:53:05 +1200] "GET /adclick.php?..." ...
192.169.133.249          - - [22/Apr/2009:18:53:05 +1200] "GET /news.html ..." ...
bot-332.ihug.co.nz       - - [22/Apr/2009:18:53:05 +1200] "GET /news.html ..." ...
192.169.133.249          - - [22/Apr/2009:18:53:05 +1200] "GET /adclick.php?..." ...
...
```

```
...
192.169.133.249
192.169.133.249
192.196.179.98
proxy-367.bellsouth.net
bot-332.ihug.co.nz
proxy-367.bellsouth.net
192.169.133.249
bot-332.ihug.co.nz
192.169.133.249
...
```

Estraiamo la **prima parola** da ogni riga

```
...
192.169.133.249
192.169.133.249
192.169.133.249
192.169.133.249
192.196.179.98
bot-332.ihug.co.nz
bot-332.ihug.co.nz
proxy-367.bellsouth.net
proxy-367.bellsouth.net
...
```

Ordiniamo le righe alfa-numericamente

```
...
 4 192.169.133.249
 1 192.196.179.98
 2 bot-332.ihug.co.nz
 2 proxy-367.bellsouth.net
...
```

Contiamo le **ripetizioni** di ogni riga

Esempio avanzato di pipe: top-3 client che accedono ad un sito web

```
192.115.109.123 - - [22/Apr/2009:18:52:51 +1200] "GET /stories.php?date=yesterday HTTP/1.1" ...
192.125.145.131 - - [22/Apr/2009:18:53:18 +1200] "GET /latest.php?category=national HTTP/1.1" ...
dyn-276.aol.com - - [22/Apr/2009:18:54:38 +1200] "GET /latest.php?category=national HTTP/1.1" ...
```

```
$ wget https://github.com/acaudwell/Logstalgia/raw/master/data/example.log

$ cat example.log | cut -d" " -f1
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
dyn-530.optus.com.au
dhcp-465.telstra.com.au
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
...
```

Il comando "cut":

- spezza il testo in colonne in base a un separatore (-d, in questo caso lo spazio)
- stampa solo alcune specifiche colonne (-f, in questo caso la prima parola di ogni riga)

Esempio avanzato di pipe: top-3 client che accedono ad un sito web

```
192.115.109.123 - - [22/Apr/2009:18:52:51 +1200] "GET /stories.php?date=yesterday HTTP/1.1" ...
192.125.145.131 - - [22/Apr/2009:18:53:18 +1200] "GET /latest.php?category=national HTTP/1.1" ...
dyn-276.aol.com - - [22/Apr/2009:18:54:38 +1200] "GET /latest.php?category=national HTTP/1.1" ...
```

```
$ wget https://github.com/acaudwell/Logstalgia/raw/master/data/example.log

$ cat example.log | awk '{print $1}'
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
dyn-530.optus.com.au
dhcp-465.telstra.com.au
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
proxy-435.dialup.xtra.co.nz
...
```

In alternativa, è possibile utilizzare **awk** per stampare una specifica colonna di testo.

AWK è un mini-linguaggio per filtrare le righe/colonne di un testo.

http://www.awklang.org/

Esempio avanzato di pipe: top-3 client che accedono ad un sito web

```
192.115.109.123 - - [22/Apr/2009:18:52:51 +1200] "GET /stories.php?date=yesterday HTTP/1.1" ...
192.125.145.131 - - [22/Apr/2009:18:53:18 +1200] "GET /latest.php?category=national HTTP/1.1" ...
dyn-276.aol.com - - [22/Apr/2009:18:54:38 +1200] "GET /latest.php?category=national HTTP/1.1" ...
```

```
$ wget https://github.com/acaudwell/Logstalgia/raw/master/data/example.log

$ cat example.log | cut -d" " -f1 | sort
192.115.109.123
192.115.109.123
192.115.109.124
192.115.109.124
192.118.184.192
192.125.145.131
192.125.145.131
192.125.145.131
192.125.145.131
192.125.145.131
...
```

"sort" ordina i risultati (alfabeticamente)

Esempio avanzato di pipe: top-3 client che accedono ad un sito web

```
192.115.109.123 - - [22/Apr/2009:18:52:51 +1200] "GET /stories.php?date=yesterday HTTP/1.1" ...
192.125.145.131 - - [22/Apr/2009:18:53:18 +1200] "GET /latest.php?category=national HTTP/1.1" ...
dyn-276.aol.com - - [22/Apr/2009:18:54:38 +1200] "GET /latest.php?category=national HTTP/1.1" ...
```

```
$ wget https://github.com/acaudwell/Logstalgia/raw/master/data/example.log

$ cat example.log | cut -d" " -f1 | sort | uniq -c
   2  192.115.109.123
   2  192.115.109.124
   1  192.118.184.192
  47  192.125.145.131
   2  192.126.158.42
  59  192.131.128.127
  55  192.142.80.43
   1  192.145.92.21
   4  192.147.202.14
  11  192.15.223.198
...
```

"uniq" rimuove le righe che si ripetono più volte (es. "192.115.109.123").

L'opzione "-c" stampa il numero di ripetizioni.

Esempio avanzato di pipe: top-3 client che accedono ad un sito web

```
192.115.109.123 - - [22/Apr/2009:18:52:51 +1200] "GET /stories.php?date=yesterday HTTP/1.1" ...
192.125.145.131 - - [22/Apr/2009:18:53:18 +1200] "GET /latest.php?category=national HTTP/1.1" ...
dyn-276.aol.com - - [22/Apr/2009:18:54:38 +1200] "GET /latest.php?category=national HTTP/1.1" ...
```

```
$ wget https://github.com/acaudwell/Logstalgia/raw/master/data/example.log

$ cat example.log | cut -d" " -f1 | sort | uniq -c | sort -rn | head -3
  84 dhcp-312.comcast.net
  83 dyn-547.dialup.xtra.co.nz
  76 dyn-276.aol.com
```

Ordiniamo i risultati di "uniq" in base al numero di ripetizioni, in ordine numerico ("-n") e decrescente ("-r")

Stampiamo solo i primi 3 risultati (gli IP con il numero maggiore di accessi)

Esercizio

```
...
192.169.133.249          - - [22/Apr/2009:18:53:04 +1200] "GET /adclick.php?..." ...
192.169.133.249          - - [22/Apr/2009:18:53:04 +1200] "GET /latest.php?..." ...
192.196.179.98           - - [22/Apr/2009:18:53:04 +1200] "GET /gallery.php?..." ...
proxy-367.bellsouth.net  - - [22/Apr/2009:18:53:04 +1200] "GET /images/photos/455.jpg ..." ...
bot-332.ihug.co.nz       - - [22/Apr/2009:18:53:04 +1200] "GET /adclick.php?..." ...
proxy-367.bellsouth.net  - - [22/Apr/2009:18:53:05 +1200] "GET /adclick.php?..." ...
192.169.133.249          - - [22/Apr/2009:18:53:05 +1200] "GET /news.html ..." ...
bot-332.ihug.co.nz       - - [22/Apr/2009:18:53:05 +1200] "GET /news.html ..." ...
192.169.133.249          - - [22/Apr/2009:18:53:05 +1200] "GET /adclick.php?..." ...
...
```

Come fare per stampare le pagine (7ª colonna) visitate dallo IP 192.39.35.118 ?

Soluzione

```
...
192.169.133.249          - - [22/Apr/2009:18:53:04 +1200] "GET /adclick.php?..." ...
192.169.133.249          - - [22/Apr/2009:18:53:04 +1200] "GET /latest.php?..." ...
192.196.179.98           - - [22/Apr/2009:18:53:04 +1200] "GET /gallery.php?..." ...
proxy-367.bellsouth.net  - - [22/Apr/2009:18:53:04 +1200] "GET /images/photos/455.jpg ..." ...
bot-332.ihug.co.nz       - - [22/Apr/2009:18:53:04 +1200] "GET /adclick.php?..." ...
proxy-367.bellsouth.net  - - [22/Apr/2009:18:53:05 +1200] "GET /adclick.php?..." ...
192.169.133.249          - - [22/Apr/2009:18:53:05 +1200] "GET /news.html ..." ...
bot-332.ihug.co.nz       - - [22/Apr/2009:18:53:05 +1200] "GET /news.html ..." ...
192.169.133.249          - - [22/Apr/2009:18:53:05 +1200] "GET /adclick.php?..." ...
...
```

```
$ cat example.log | grep "192.39.35.118" | cut -d" " -f7
/news.html
/stories/business/070201/7.htm
/stories/world/080201/47.htm
```

Log Parsing Cheat Sheet

GREP	GREP allows you to search patterns in files. ZGREP for GZIP files. **$grep <pattern> file.log**	-n: Number of lines that matches -i: Case insensitive -v: Invert matches -E: Extended regex -c: Count number of matches -l: Find filenames that matches the pattern
NGREP	NGREP is used for analyzing network packets. **$ngrep -I file.pcap**	-d: Specify network interface -i: Case insensitive -x: Print in alternate hexdump -t: Print timestamp -I: Read pcap file
CUT	The CUT command is used to parse fields from delimited logs. **$cut -d ":" -f 2 file.log**	-d: Use the field delimiter -f: The field numbers -c: Specifies characters position
SED	SED (Stream Editor) is used to replace strings in a file. **$sed s/regex/replace/g**	s: Search -e: Execute command g: Replace -n: Suppress output d: Delete w: Append to file

SORT	SORT is used to sort a file. **$sort foo.txt**	-o: Output to file -c: Check if ordered -r: Reverse order -u: Sort and remove -n: Numerical sort -f: Ignore case -k: Sort by column -h: Human sort
UNIQ	UNIQ is used to extract uniq occurrences. **$uniq foo.txt**	-c: Count the number of duplicates -d: Print duplicates -i: Case insensitive
DIFF	DIFF is used to display differences in files by comparing line by line. **$diff foo.log bar.log**	How to read output? a: Add #: Line numbers c: Change <: File 1 d: Delete >: File 2
AWK	AWK is a programming language use to manipulate data. **$awk {print $2} foo.log**	Print first column with separator ":" **$awk -F: '{print $1}' /etc/passwd** Extract uniq value from two files: awk 'FNR==NR {a[$0]++; next} !($0 in a)' f1.txt f2.txt

@FROGGER_
THOMAS ROCCIA

Esempio avanzato di pipe: ricerca dei "disk hogs"

```
$ find . -type d
.
./Orari
./Slides
./Slides/UML
./Slides/legacy
./Programma
./Esercitazioni
.....

$ find . -type d -exec du -s -m {} \;
1094    .
1       ./Orari
128     ./Slides
6       ./Slides/UML
16      ./Slides/legacy
1       ./Programma
196     ./Esercitazioni
.....
```

Il comando "**find .**" cerca tutti i file e cartelle all'interno del percorso "**.**"

Il parametro "**-type d**" seleziona le sole cartelle.

Il comando "**find .**" ha un parametro speciale "**-exec**", per eseguire un certo comando su tutti i file/cartelle trovati.

"**du -s -m**" calcola la dimensione in MB di una cartella.

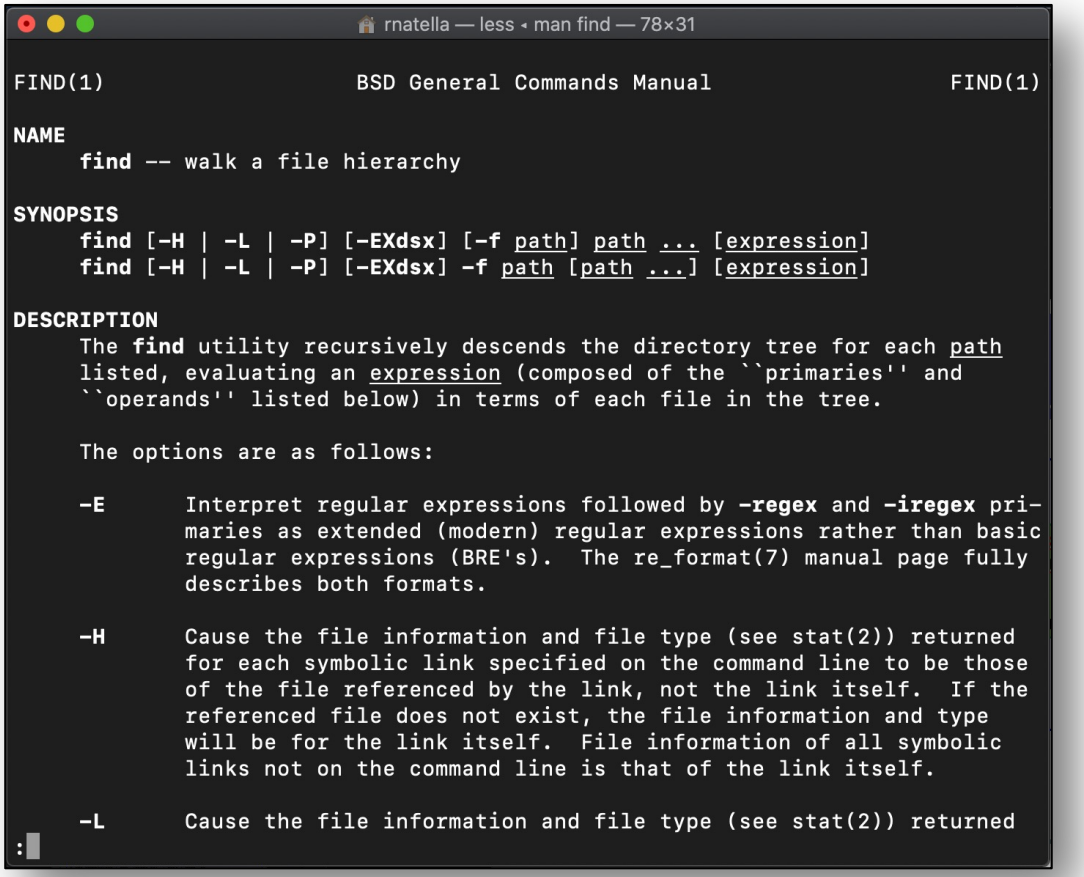
Esempio avanzato di pipe: ricerca dei "disk hogs"

```
$ find . -type d -exec du -s -m {} \; > find.txt

$ cat find.txt | sort -n -r | head -10

1094    .
443     ./Libri
320     ./Esami
196     ./Esercitazioni
128     ./Slides
83      ./Libri/Sommerville
80      ./Esami/Prova 2019 Dicembre
77      ./Esercitazioni/ACTS
75      ./Esami/Prova 2019 Dicembre/repo
74      ./Libri/Pressman
```

Il comando "sort" legge righe da STDIN, le intepreta come numeri ("-n") e le ordina in senso decrescente ("-r").

"head -10" filtra i soli primi 10 risultati.

Manuale



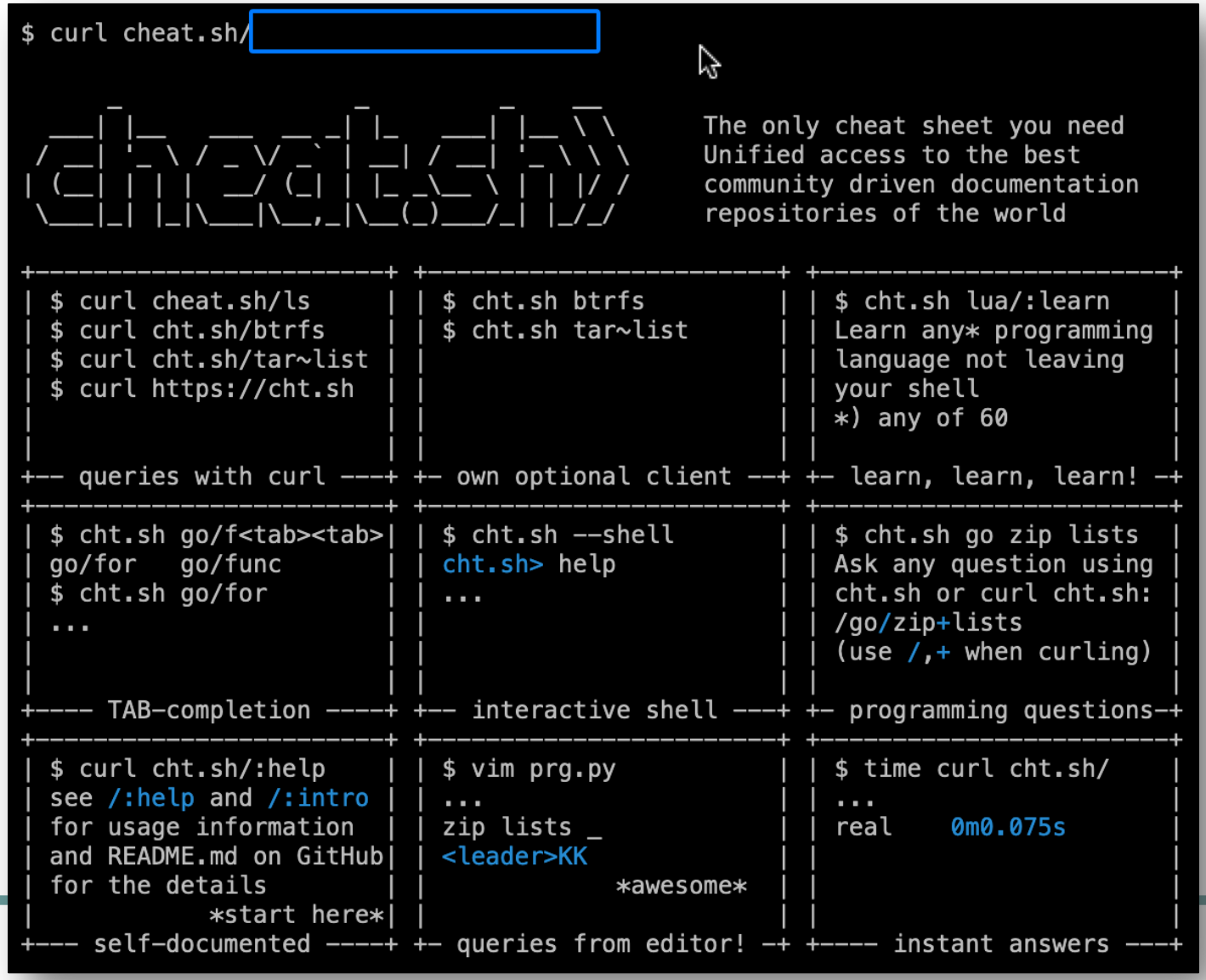
Tutti i comandi UNIX hanno una pagina di manuale, con la lista di tutti i parametri di ingresso (es., **man find**)

Help in linea

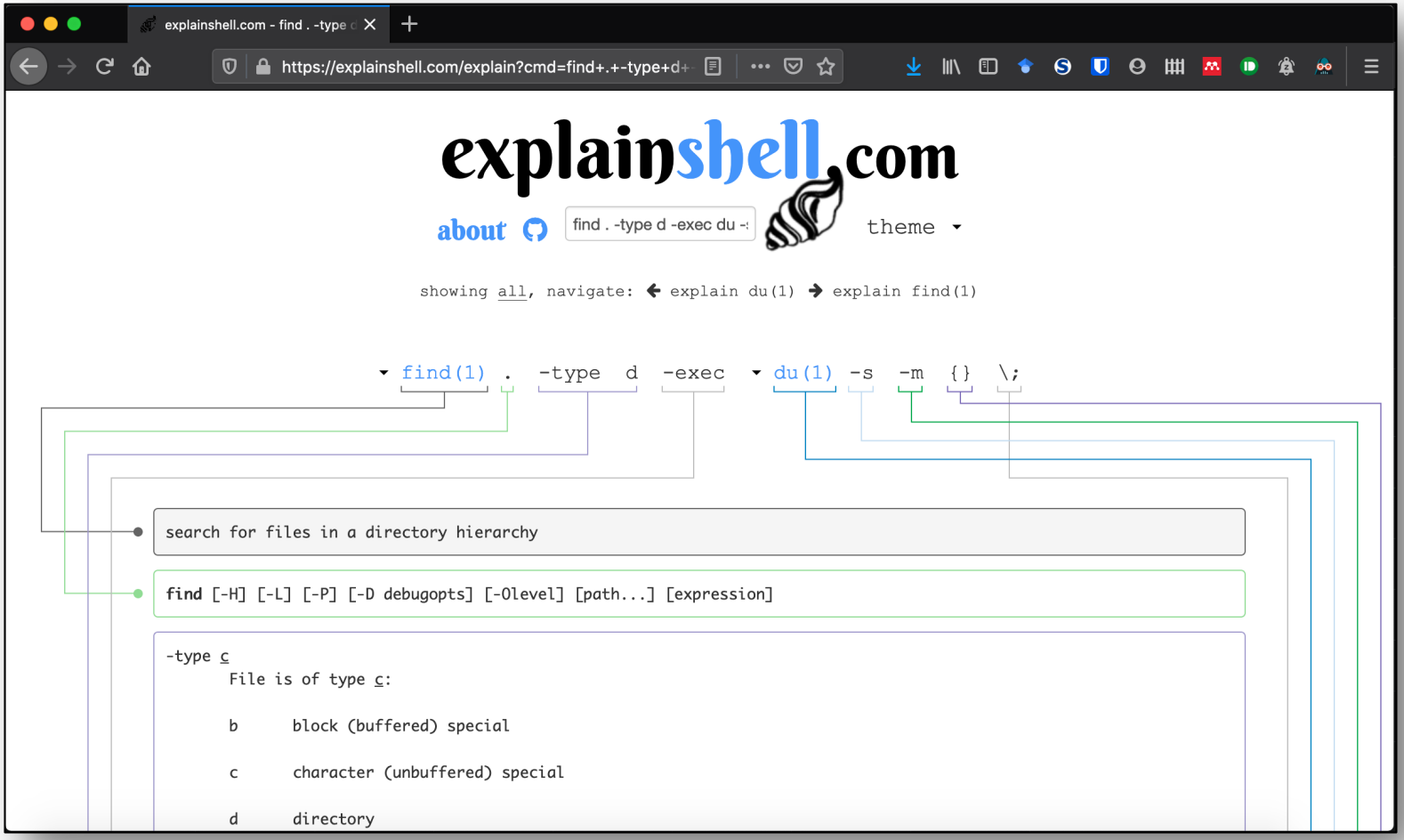
La maggior parte dei comandi, se chiamati con la sola opzione "-h" oppure "--help", mostra una sintesi dei parametri

```
$ ipcs -h

Usage:
 ipcs [resource-option...] [output-option]
 ipcs -m|-q|-s -i <id>

Show information on IPC facilities.

Options:
 -i, --id <id>  print details on resource identified by <id>
 -h, --help     display this help
 -V, --version  display version

Resource options:
 -m, --shmems      shared memory segments
 -q, --queues      message queues
 -s, --semaphores  semaphores
 -a, --all         all (default)

...

For more details see ipcs(1).
```

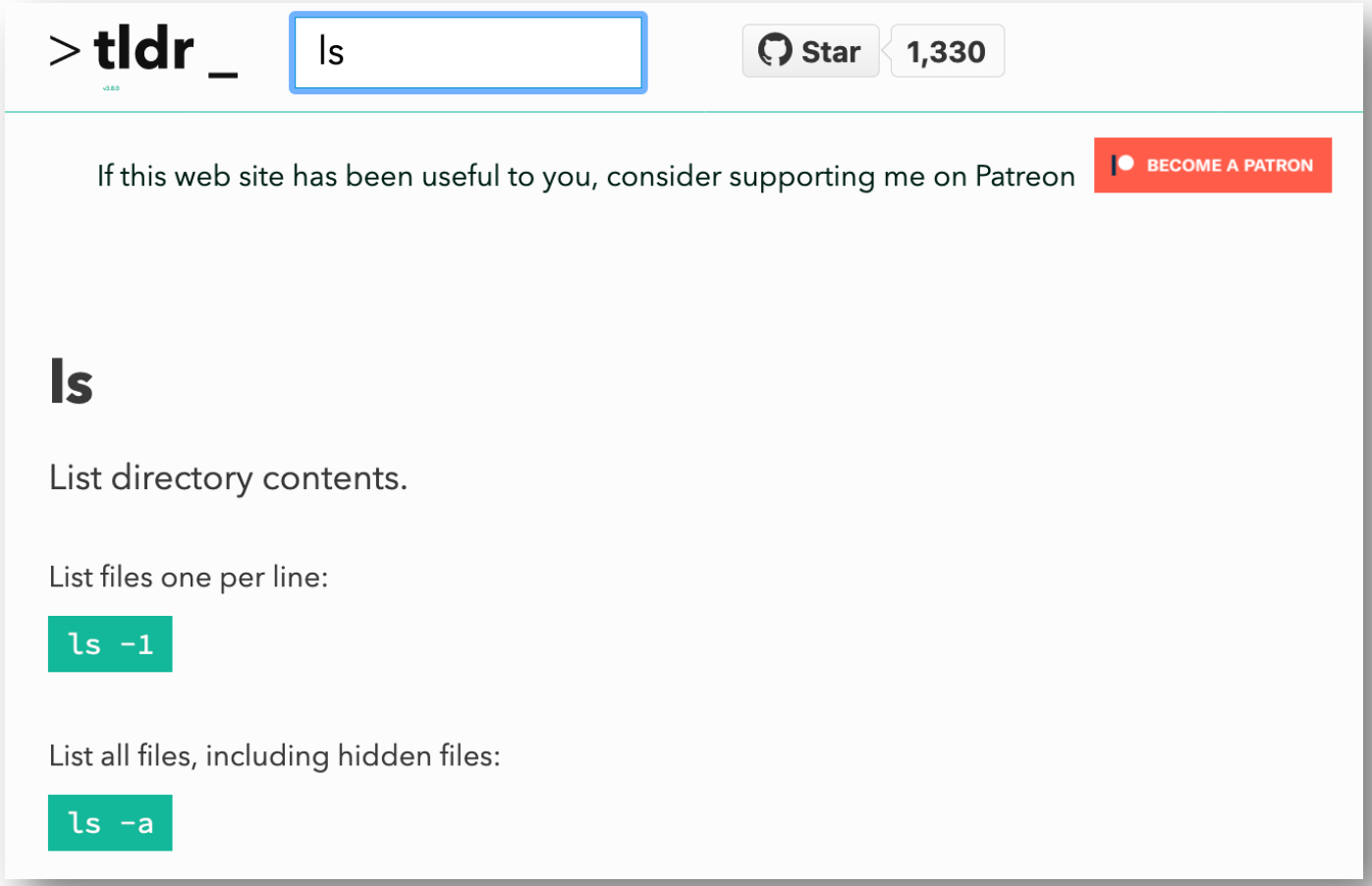
cheat.sh


$ curl cheat.sh/
The only cheat sheet you need
Unified access to the best
community driven documentation
repositories of the world
$ curl cheat.sh/ls
$ curl cht.sh/btrfs
$ curl cht.sh/tar~list
$ curl https://cht.sh
queries with curl
$ cht.sh btrfs
$ cht.sh tar~list
own optional client
$ cht.sh lua/:learn
Learn any* programming
language not leaving
your shell
*) any of 60
learn, learn, learn!
$ cht.sh go/f<tab><tab>
go/for go/func
$ cht.sh go/for
...
TAB-completion
$ cht.sh --shell
cht.sh> help
...
interactive shell
$ cht.sh go zip lists
Ask any question using
cht.sh or curl cht.sh:
/go/zip+lists
(use /,+ when curling)
programming questions
$ curl cht.sh/:help
see /:help and /:intro
for usage information
and README.md on GitHub
for the details
start here
self-documented
$ vim prg.py
...
zip lists _
<leader>KK
awesome
queries from editor!
$ time curl cht.sh/
...
real 0m0.075s
instant answers

explainshell.com



tldr pages (simplified, community-driven man pages)



https://tldr.ostera.io/

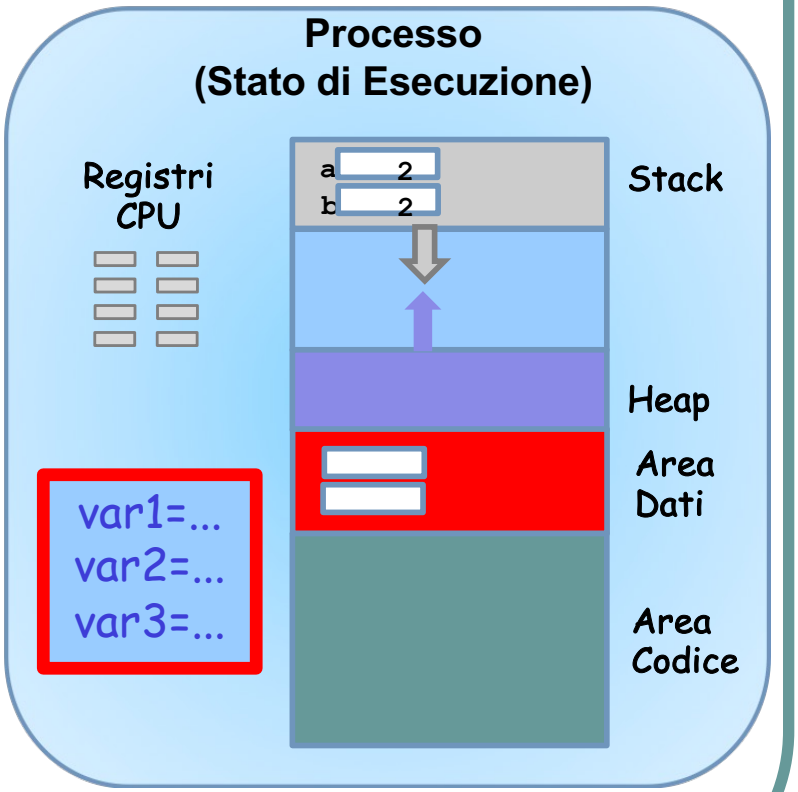
The UNIX Game

https://www.unixgame.io/

Variabili d'ambiente

- UNIX consente di memorizzare in un processo delle variabili d'ambiente
- es. variabili di "configurazione"
- Formato (stringhe):

NOMEVAR= *VALORE*



Variabili d'ambiente

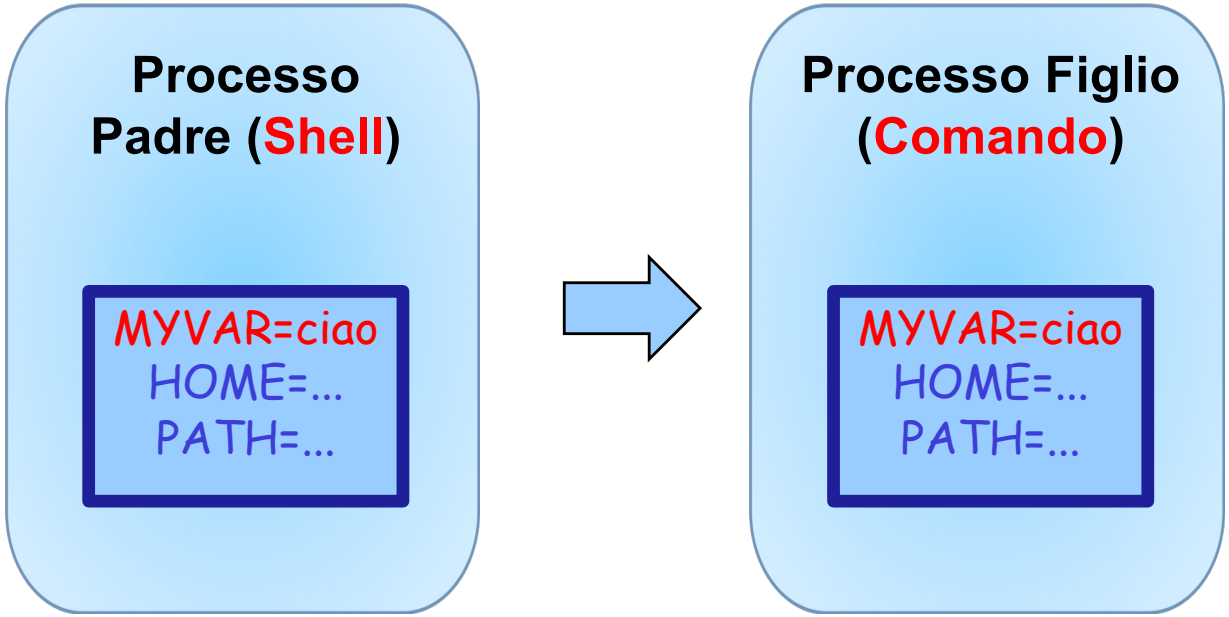
```
$ env

LANG=it_IT.UTF-8
HOME=/home/so
PWD=/home/so
USERNAME=so
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:...
...
```

La variabile PATH indica alla shell le cartelle di sistema in cui cercare i programmi eseguibili (elenco separato con ":")

Variabili d'ambiente

- Tipicamente, l'utente imposta le variabili d'ambiente nella shell
- Le variabili sono copiate nei processi avviati dalla shell

```
$ export MYVAR=ciao

$ ./comando
```



int execve (const char *filename, char *const argv[], char *const envp[]);

Shell scripting

- È possibile usare la shell per creare piccoli programmi (**script**)
- File di testo con una lista di comandi
- La shell ne "**interpreta**" le righe una ad una, come se fossero digitate a mano

Shell scripting

- Uno script può essere eseguito tramite il comando "bash"

```
$ bash mioscript.sh
```

Shell scripting

- Lo script può essere anche avviato come se fosse un classico programma

```
$ ./mioscript.sh
```

- Il file deve avere i permessi di esecuzione, ed iniziare con la seguente linea:

```
#!/bin/bash

... resto dello script ...
```

https://en.wikipedia.org/wiki/Shebang_(Unix)

Esempio di shell scripting

```
$ cat dante.sh

#!/bin/bash

URL=http://www.mirrorservice.org/sites/ftp.ibiblio.org/
                        pub/docs/books/gutenberg/1/0/1/1012/1012-0.txt

wget -q $URL -O la-divina-commedia.txt

echo "Il numero di linee della Divina Commedia è circa:"
cat la-divina-commedia.txt | wc -l

echo "Il numero di caratteri è circa:"
cat la-divina-commedia.txt | wc -c
```

Per convenzione, il nome del file ha estensione ".sh"

Esempio di shell scripting

Il SO rileva "#!/bin/bash" all'inizio del file.

Equivale a dare il comando:

/bin/bash dante.sh

```
$ chmod +x dante.sh

$ ./dante.sh
Il numero di linee della Divina Commedia è circa:
   20023
Il numero di caratteri è circa:
  626459
```

Esempio di shell scripting

```
$ cat dante.sh

#!/bin/bash

URL=http://www.mirrorservice.org/sites/ftp.ibiblio.org/
                        pub/docs/books/gutenberg/1/0/1/1012/1012-0.txt

wget -q $URL -O la-divina-commedia.txt

echo "Il numero di linee della Divina Commedia è circa:"
cat la-divina-commedia.txt | wc -l

echo "Il numero di caratteri è circa:"
cat la-divina-commedia.txt | wc -c
```

Lo script esegue in totale 5 processi, che eseguono i programmi "wget", "cat", "wc"

Esempio di shell scripting

```
$ cat dante.sh

#!/bin/bash

URL=http://www.mirrorservice.org/sites/ftp.ibiblio.org/
                        pub/docs/books/gutenberg/1/0/1/1012/1012-0.txt

wget -q $URL -O la-divina-commedia.txt

echo "Il numero di linee della Divina Commedia è circa:"
cat la-divina-commedia.txt | wc -l

echo "Il numero di caratteri è circa:"
cat la-divina-commedia.txt | wc -c
```

Il comando "**echo**" non lancia un eseguibile esterno,
è un **comando "built-in"** della shell
(come "cd" e alcuni altri)

Shell scripting

- La shell fornisce un linguaggio completo, con **variabili**, **condizioni**, e **cicli**

```
$ VAR=abc

$ echo "Il valore è "; echo $VAR
Il valore è
abc
```

Il simbolo "$" è usato per leggere le variabili.

Nota: con la parola "export", la variabile viene inserita tra le variabili d'ambiente del processo e trasmessa ai figli

Shell scripting

- La shell fornisce il meccanismo della **interpolazione**, per combinare più stringhe e variabili

```
$ VAR=abc

$ echo "Il valore è: $VAR"
Il valore è: abc
```

Il contenuto fra i **doppi-apici** contiene una variabile, ne viene inserito il valore

Shell scripting - variabili

```
$ cat esempio-if.sh
#!/bin/bash

VAR="ciao mondo"

if [ "$VAR" == "" ]
then
    echo "La variabile è vuota"
else
    echo "La variabile è: $VAR"
fi
```

IMPORTANTE: **non lasciare spazi** prima e dopo il simbolo "**=**"

IMPORTANTE: occorre lasciare degli spazi **prima** e **dopo** le parentesi quadre (" **[...]** ")

IMPORTANTE: è opportuno fare **sempre quoting** (**"$VAR"**) delle variabili e delle stringhe se c'è la possibilità che siano **vuote** o contengano **spazi**, per evitare errori sintattici

```
$ ./esempio-if.sh
La variabile è: ciao mondo
```

È bene utilizzare il linguaggio della shell solo per programmi molto semplici.
Nei casi più complessi, è opportuno usare altri linguaggi di scripting, come Python.

Shell scripting - variabili

```
$ cat esempio-if-4.sh
#!/bin/bash

wget -q http://www.gutenberg.org/non-esiste.txt

if [ $? -ne 0 ]
then
    echo "Download fallito :-("
else
    echo "Download riuscito!"
fi
```

La shell aggiorna automaticamente una **variabile speciale $?** ogni volta che viene eseguito un comando.

Se $? è diverso da 0, allora si è verificato un qualche errore nel comando.

```
$ ./esempio-if-4.sh
Download fallito :-(
```

NOTA: Il confronto tra interi si effettua con gli operatori "-ne", "-eq", "-gt", etc.

Shell scripting – comandi esterni

La sintassi **$(...)** permette di lanciare un **comando esterno**

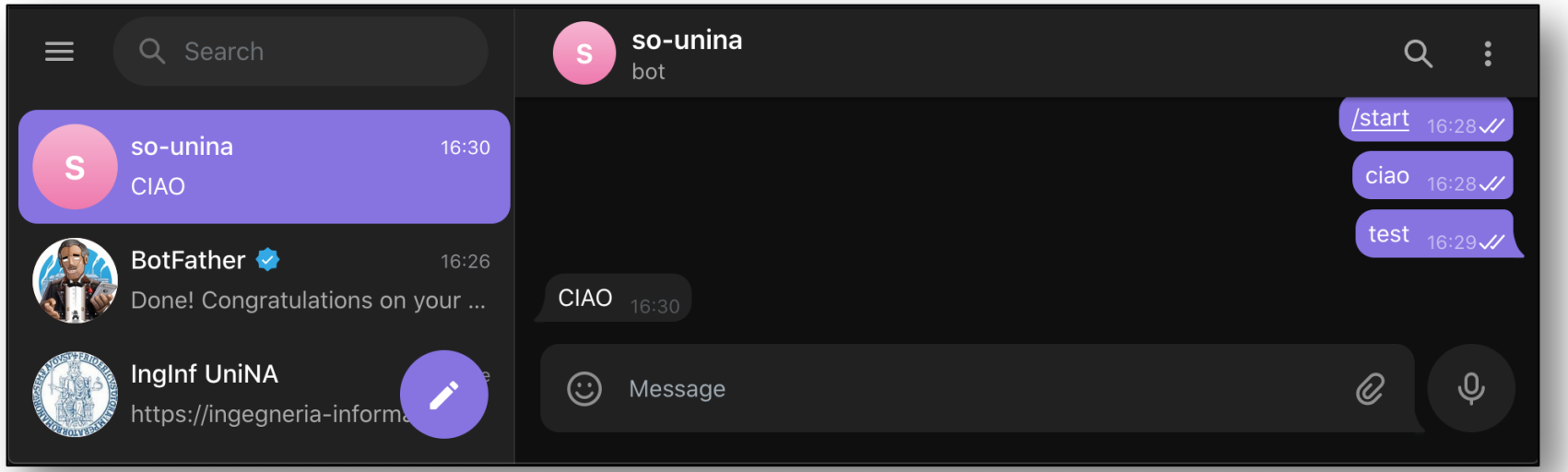
L'output è disponibile come variabile

```
$ cat esempio-command-sub.sh
#!/bin/bash

NAME=$(whoami)

echo "Il mio username è $NAME"
```

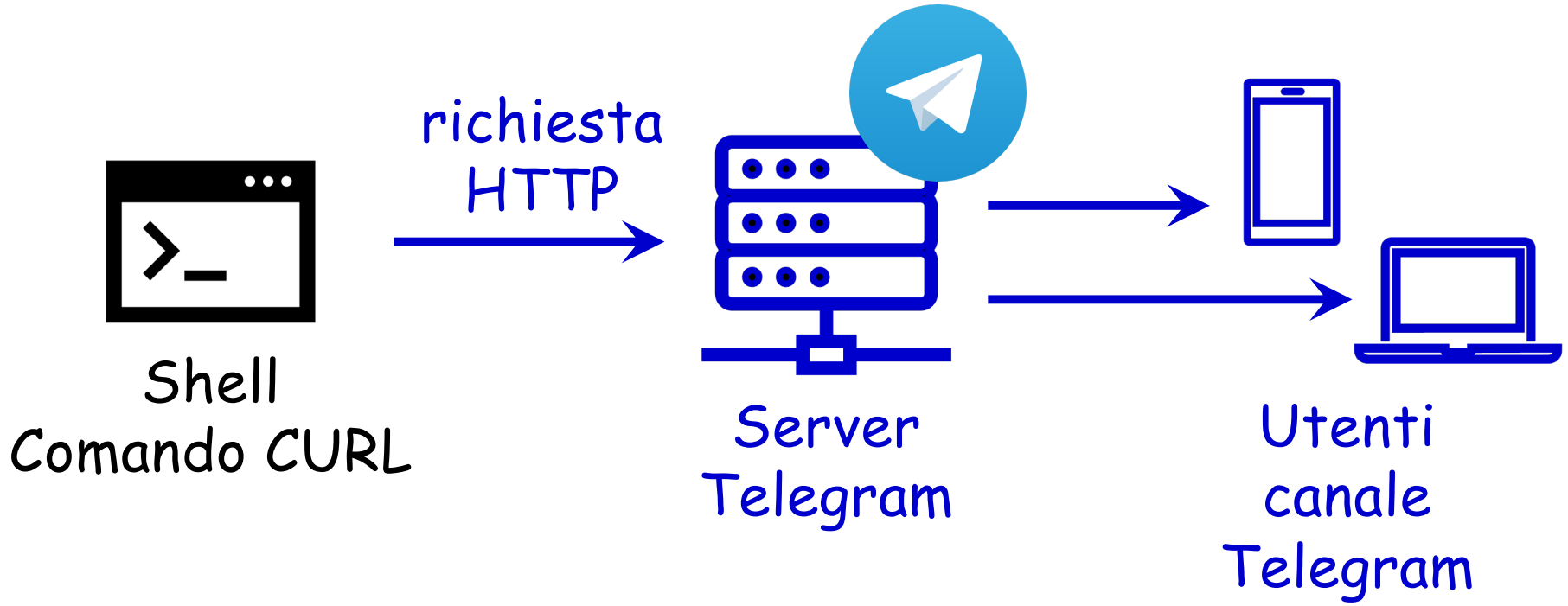
```
$ ./esempio-command-sub.sh

Il mio username è so
```

Shell scripting – esempio HTTP

- La shell permette di inviare **richieste HTTP** verso servizi remoti (cloud)
- Esempio: Invio di messaggio con un **bot Telegram**



Shell scripting – esempio HTTP



Shell scripting – esempio HTTP

```
POST /bot123456:xxxxxxxxxxxxxxxxxxxxxxxxxx/sendMessage HTTP/2
Host: api.telegram.org
User-Agent: curl/8.1.2
Accept: */*
Content-Type: multipart/form-data; ...
Content-Length: ...

...
name="chat_id"
-123456789
...
name="text"
il messaggio
```

Shell scripting – esempio HTTP

Tipo di richiesta

Percorso da accedere (sendMessage)

HEADER

BODY

```
POST /bot123456:xxxxxxxxxxxxxxxxxxxxxxxxx/sendMessage HTTP/2
Host: api.telegram.org
User-Agent: curl/8.1.2
Accept: */*
Content-Type: multipart/form-data; ...
Content-Length: ...
```

Server da accedere

...linea vuota...

```
...
name="chat_id"
-123456789
...
name="text"
il messaggio
```

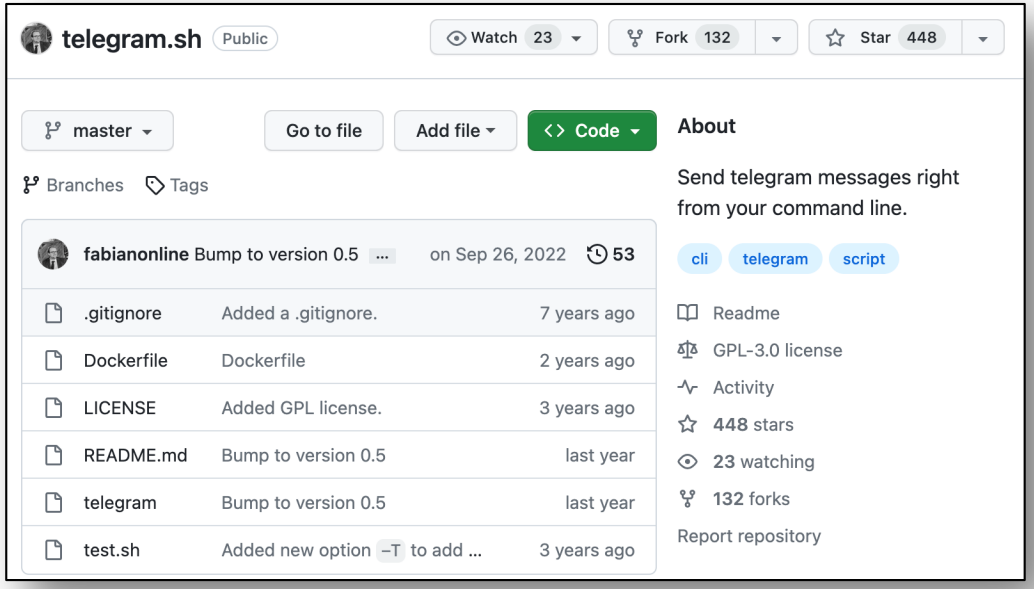
Shell scripting – esempio HTTP

```
# token segreto del bot
$ TOKEN="123456:xxxxxxxxxxxxxxxxxxxxxxxxxxx"

# ID del gruppo Telegram
$ CHAT_ID="-100123456789"

$ curl -X POST
       -H "Content-Type: multipart/form-data"
       -F chat_id=$CHAT_ID
       -F text="il messaggio"
       "https://api.telegram.org/bot$TOKEN/sendMessage"
```

Shell scripting – esempio CLI

- Ci sono spesso script open-source per command-line client (CLI) verso servizi cloud

```
$ ./telegram -t <TOKEN> -c <CHAT ID> "CIAO"
```



https://github.com/fabianonline/telegram.sh

Shell scripting – esempio

- L'esempio "**ostep-book.sh**" scarica i singoli PDF dei capitoli, e li unisce in un unico PDF
- Utilizza variabili, cicli, e comandi esterni

```
1  #!/bin/bash
2
3  # Occorre avere installato il pacchetto "pdftk-java":
4  #   sudo apt install pdftk-java
5
6  awk -F, '{print $3}' list.txt | while read PDF
7  do
8          I=$(($I+1))
9          wget -O "OSTEP-$I-$PDF"  "http://pages.cs.wisc.edu/~remzi/OSTEP/$PDF"
10 done
11
12 pdftk OSTEP-*.pdf cat output output.pdf
13
```

Nota, installare:

```
sudo apt install openjdk-11-jdk pdftk-java poppler-utils
```

Processi in foreground/background

- La shell attende la terminazione dei comandi usando *wait()*
- Il cursore dei comandi non è disponibile
- Esecuzione "**in foreground**"

```
$ ./script.sh
<...attende il termine del comando...>

$
```

Processi in foreground/background

- Con &, si evita l'attesa dell'esecuzione
- Il cursore è subito disponibile
- Esecuzione "in background"

```
$ ./script.sh &
$
<...lo script continua a eseguire...>
```

Esempio di processi paralleli in background

```
for F in *.jpg
do
  convert "$F" "$F.png" &
done

jobs
wait

echo "All images have been converted."
```

Lancia più volte *convert* su in parallelo

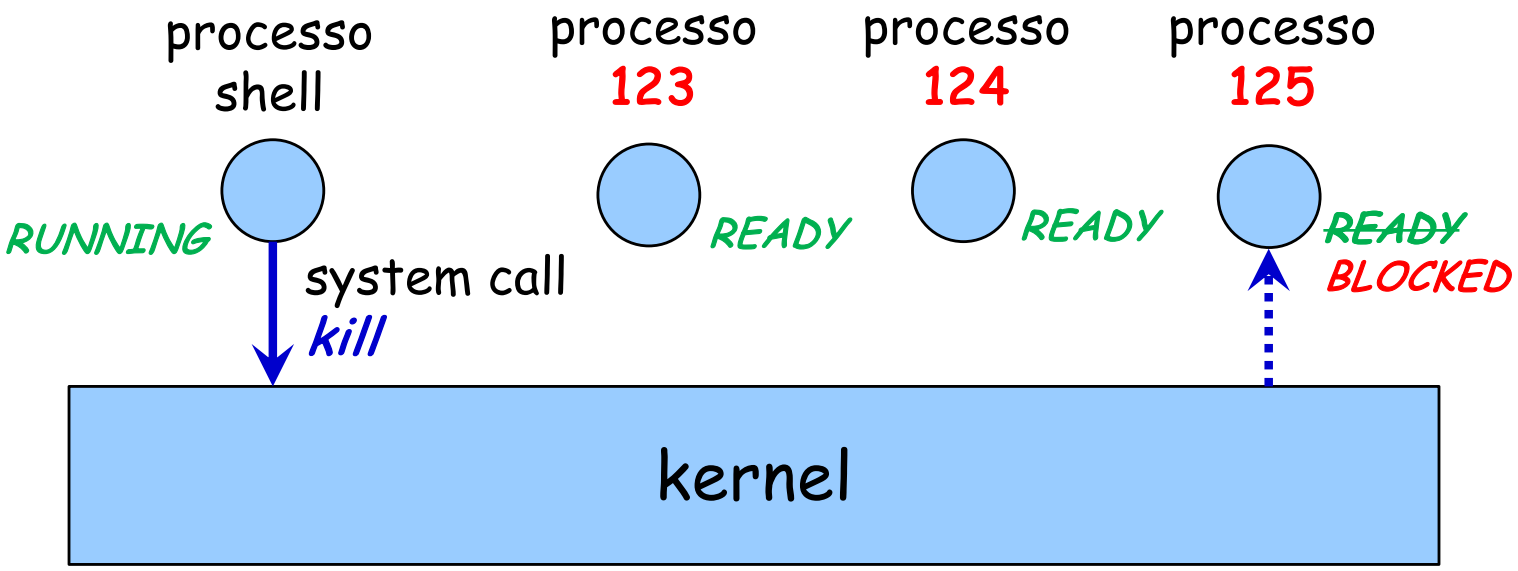
Il comando "jobs" stampa la lista dei processi in background

```
$ ./esempio-background.sh
[1]   Done                    convert "$F" "$F.png"
[2]   Running                 convert "$F" "$F.png" &
[3]   Running                 convert "$F" "$F.png" &
...
All images have been converted.
```

Richiede di installare: `sudo apt install imagemagick`

Segnali UNIX

- La shell usa il meccanismo dei **segnali** per controllare lo **stato di un processo**



Segnali UNIX tramite la shell

- Comando "**kill**":
  invia un segnale a un processo

- **Control+C**:
  il processo in foreground nella shell viene terminato

- **Control+Z**:
  il processo in foreground nella shell viene sospeso (può essere riattivato)

Il comando "kill"

- Il comando "**kill**" permette di inviare segnali ai **processi di sistema** (non collegati alla propria shell)
- Parametri:
  - il tipo di segnale da inviare (acronimo o codice numerico)
  - il PID del processo da segnalare

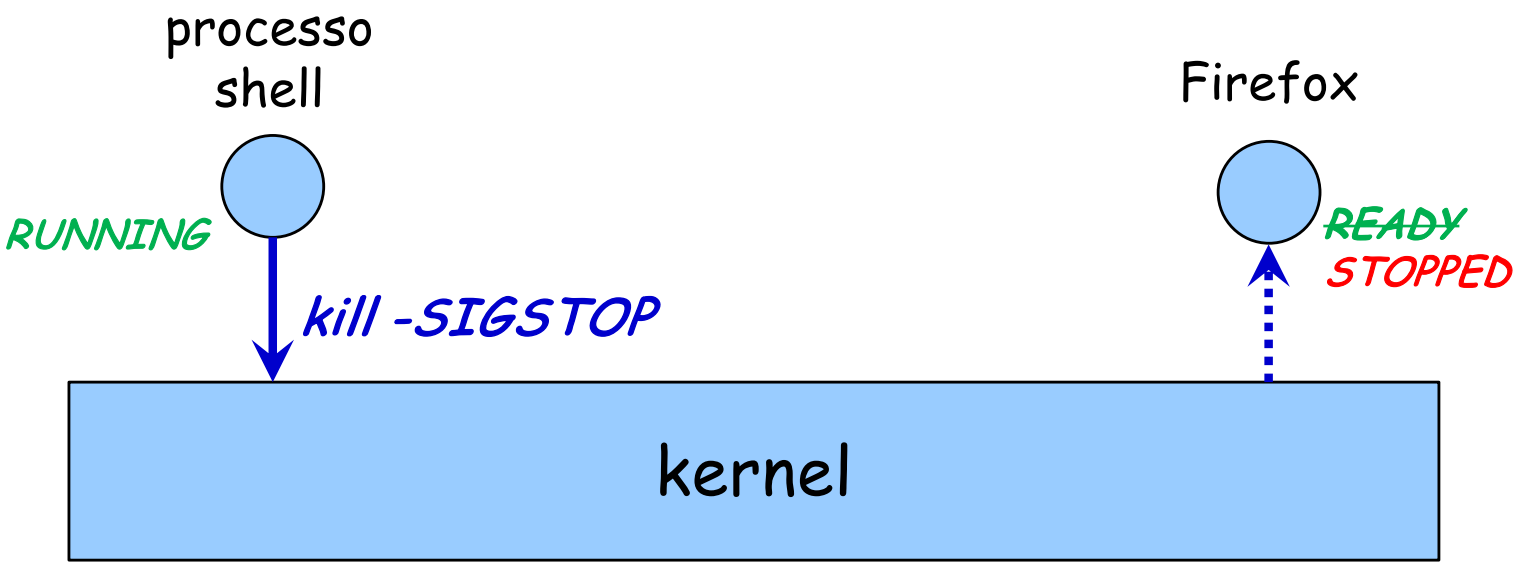
```
$ kill -INT 1234
```

Il segnale **INT** ("interrupt") equivale a **Control-C**

Segnali UNIX

- Esempio: sospensione di un processo

```
ps aux | grep firefox | head -1
kill -SIGSTOP $PID
```



Segnali UNIX

- Numerosi tipi di segnali, innescano varie azioni (configurate nel programma) da parte del processo destinatario o dal SO

SIGINT

Default action: **Terminate**
Possible action: Terminate, Ignore, Function Call

Inviato dalla shell al processo foreground con ^C. Il processo può ignorarlo.

SIGCONT

Default action: **Wake up**
Possible action: Wake up, Wake up + Function call

Inviato dal comando "fg"

SIGKILL

Default action: **Terminate**
Possible action: Terminate

Forza la terminazione

The TTY demystified: http://linusakesson.net/programming/tty/index.php

Segnali UNIX

Signal name	Signal num.	Description
SIGHUP	1	**Hang-up** detected on controlling terminal, or death of controlling process
SIGINT	2	Issued if the user sends an **interrupt** signal (Ctrl+C)
SIGQUIT	3	Issues if the user sends a **quit** signal (Ctrl+D), **core dump**
SIGKILL	9	If a process gets this signal, it must **quit immediately** and will not perform any clean-up operations
SIGSEGV	11	Issued when a **memory access violation** occurs
SIGALRM	14	Alarm clock signal (used for **timers**)
SIGTERM	15	Software termination signal (sent by **kill** by default)

Utilizzare i signal handler solo per **operazioni semplici** (es., clean-up risorse, riconfigurazione, riportare lo stato, abilitare debugging, etc.).

Possono soffrire di race condition e non sono indicati per la sincronizzazione.

Il comando "kill"

Doom as an Interface for Process Management:
https://www.cs.unm.edu/~dlchao/flake/doom/chi/chi.html
https://psdoom.sourceforge.net/


I WOULDN'T LEAVE IF I WERE YOU.
DOS IS MUCH WORSE.
(PRESS Y TO QUIT TO DOS.)