

Bash: history list (I)

L'**history list** è un tool fornito dalla shell bash che consente di evitare all'utente di digitare più volte gli stessi comandi:

- bash memorizza nell'history list gli ultimi **500 comandi** inseriti dall'utente;
- l'history list viene memorizzata nel file `.bash_history` nell'home directory dell'utente al momento del logout (e riletta al momento del login);
- il comando `history` consente di visualizzare la lista dei comandi:

```
$ history | tail -5
 511 pwd
 512 ls -al
 513 cd /etc
 514 more passwd
 515 history | tail -5
```

- ogni riga prodotta dal comando `history` è detta **evento** ed è preceduta dal **numero dell'evento**.

Bash: history list (II)

Conoscendo il numero dell'evento corrispondente al comando che vogliamo ripetere, possiamo eseguirlo, usando il metacarattere !:

```
$ !515
```

```
history | tail -5
```

```
512 ls -al
```

```
513 cd /etc
```

```
514 more passwd
```

```
515 history | tail -5
```

```
516 history | tail -5
```

Se l'evento che vogliamo ripetere è l'ultimo della lista è sufficiente usare !!:

```
$ !!
```

```
history | tail -5
```

```
513 cd /etc
```

```
514 more passwd
```

```
515 history | tail -5
```

```
516 history | tail -5
```

```
517 history | tail -5
```

Bash: history list (III)

Oltre a riferirsi agli eventi tramite i loro numeri, è possibile eseguire delle ricerche testuali per individuare quello a cui siamo interessati:

```
$ !ls
```

```
ls -al
```

```
total 491
```

```
drwxr-xr-x  16 root    root        0 Oct 15 21:35 .
```

```
drwxr-xr-x  16 root    root        0 Oct 15 21:35 ..
```

```
-rw-r--r--   1 root    root    87515 Jul 10 04:28 Mutttrc
```

```
drwxr-xr-x   2 root    root        0 Oct 15 21:27 WindowMaker
```

```
...
```

In questo modo la shell comincia a cercare a partire dall'ultimo evento, procedendo a ritroso, nell'history list un comando che inizi con `ls`.

Racchiudendo con due caratteri `?` la stringa da ricercare (e.g. `$!?ls?`), la shell controllerà che quest'ultima appaia in un punto qualsiasi del comando (non necessariamente all'inizio).

Bash: history list (IV)

Talvolta può capitare di voler ripetere un comando eseguito precedentemente dopo aver operato qualche modifica:

```
$ !ls:s/a1/i/
```

```
ls -i
```

```
1067566292 Mutttrc          123123 mib.txt
 204714 WindowMaker       123127 mime.conf
```

...

In questo modo la shell, dopo aver trovato l'evento cercato (`ls -a1`), sostituisce la stringa `a1` con `i` (`:s/a1/i/`), producendo il comando `ls -i`.

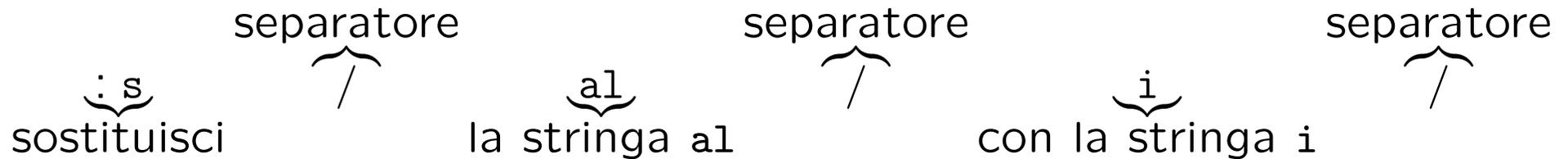


Diagram illustrating the command `!ls:s/a1/i/` and its components:

- `:s` is labeled "sostituisci" (replace).
- `a1` is labeled "la stringa a1" (the string a1).
- `i` is labeled "con la stringa i" (with the string i).
- The slashes (`/`) are labeled "separatore" (separator).

Bash: command line editing

La shell bash mette a disposizione dell'utente dei semplici **comandi di editing** per facilitare la ripetizione degli eventi:

- utilizzando i **tasti cursore**:
 - con la **freccia verso l'alto** si scorre l'history list a ritroso (un passo alla volta) facendo apparire al prompt il comando corrispondente all'evento;
 - analogamente con la **freccia verso il basso** si scorre l'history list nella direzione degli eventi più recenti.
 - le frecce sinistra e destra consentono di spostare il cursore sulla linea di comando verso il punto che si vuole editare;
- le combinazioni di tasti **Ctrl-A** e **Ctrl-E** spostano il cursore, rispettivamente all'inizio ed alla fine della linea di comando;
- il tasto **Backspace** consente di cancellare il carattere alla sinistra del cursore;
- il tasto invio (**enter**) esegue il comando.

Bash: command completion (I)

Una caratteristica molto utile della shell bash è la sua abilità di **tentare di completare** ciò che stiamo digitando al prompt dei comandi (nel seguito <Tab> indica la pressione del tasto Tab).

```
$ pass<Tab>
```

La pressione del tasto <Tab> fa in modo che la shell, sapendo che vogliamo impartire un comando, cerchi quelli che iniziano con la stringa `pass`. Siccome l'unica scelta possibile è data da `passwd`, questo sarà il comando che ritroveremo automaticamente nel prompt.

Se il numero di caratteri digitati è insufficiente per la shell al fine di determinare univocamente il comando, avviene quanto segue:

- viene prodotto un suono di avvertimento al momento della pressione del tasto Tab;
- alla seconda pressione del tasto Tab la shell visualizza una lista delle possibili alternative.
- digitando ulteriori caratteri, alla successiva pressione del tasto Tab, la lunghezza della lista diminuirà fino ad individuare un unico comando.

Bash: command completion (II)

Oltre a poter completare i comandi, la shell bash può anche completare i nomi dei file usati come argomento:

```
$ tail -2 /etc/p<Tab><Tab>
```

```
passwd printcap profile
```

```
$tail -2 /etc/pa<Tab><Tab>
```

```
bianchi:fjKppCZxEvouc:500:500:~/home/bianchi:/bin/bash
```

```
rossi:Yt1a4ffkGr02:501:500:~/home/rossi:/bin/bash
```

In questo caso alla prima doppia pressione del tasto Tab, la shell presenta tre possibili alternative; digitando una a e premendo il tasto Tab, la shell ha una quantità di informazione sufficiente per determinare in modo univoco il completamento del nome di file.

Alias

Alias già visti:

1. . (directory corrente)
2. .. (directory madre)

Esiste anche l'alias `~user` che sta per la directory home dell'utente `user`:

```
user> cd ~user          # equivale a cd /home/user
user> cd ~user/doc     # equivale a cd /home/user/doc
```

Gli alias sono trattati dalla shell come i metacaratteri, nel senso che la shell scandisce la linea di comando impartita dall'utente processando i caratteri alias prima di eseguire i comandi.

Il comando alias

Il comando `alias` serve per creare nuovi alias:

```
user> alias dir='ls -a'
```

```
user> dir
```

```
.  ..  .bash_history
```

```
user> alias ls='ls -l'
```

```
user> ls *.java
```

```
-rw-r--r--    1 user      users          0 Oct 16 17:24 Figura.java
-rw-r--r--    1 user      users          0 Oct 16 17:24 Quadrato.java
-rw-r--r--    1 user      users          0 Oct 16 17:24 Triangolo.java
```

Per rimuovere uno o più alias:

```
user> unalias dir ls
```

All'uscita dalla shell gli alias creati con il comando `alias` sono automaticamente rimossi.

Metacaratteri comuni a tutte le shell (I)

Simbolo	Significato	Esempio d'uso
>	Ridirezione dell'output	<code>ls >temp</code>
>>	Ridirezione dell'output (append)	<code>ls >>temp</code>
<	Ridirezione dell'input	<code>wc -l <text</code>
<<delim	ridirezione dell'input da linea di comando (here document)	<code>wc -l <<delim</code>
*	Wildcard: stringa di 0 o più caratteri, ad eccezione del punto (.)	<code>ls *.c</code>
?	Wildcard: un singolo carattere, ad eccezione del punto (.)	<code>ls ?.c</code>
[...]	Wildcard: un singolo carattere tra quelli elencati	<code>ls [a-zA-Z].bak</code>
{...}	Wildcard: le stringhe specificate all'interno delle parentesi	<code>ls {prog,doc}*.txt</code>

Metacaratteri comuni a tutte le shell (II)

Simbolo	Significato	Esempio d'uso
	Pipe	<code>ls more</code>
;	Sequenza di comandi	<code>pwd;ls;cd</code>
	Esecuzione condizionale. Esegue un comando se il precedente fallisce.	<code>cc prog.c echo errore</code>
&&	Esecuzione condizionale. Esegue un comando se il precedente termina con successo.	<code>cc prog.c && a.out</code>
(...)	Raggruppamento di comandi	<code>(date;ls;pwd)>out.txt</code>
#	Introduce un commento	<code>ls # lista di file</code>
\	Fa in modo che la shell non interpreti in modo speciale il carattere che segue.	<code>ls file.*</code>
!	Ripetizione di comandi memorizzati nell'history list	<code>!ls</code>

Controllo di processi

Ogni processo del sistema ha un **PID** (**Process Identity Number**).

Ogni processo può generare nuovi processi (figli).

La radice della gerarchia di processi è il processo **init** con PID=1.

init è il primo processo che parte al boot di sistema.

Il comando `ps` fornisce i processi presenti nel sistema:

```
user> ps    # fornisce i processi dell'utente associati al terminale corrente
```

PID	TTY	TIME	CMD
23228	pts/15	0:00	xdvi.bin
9796	pts/15	0:01	bash
23216	pts/15	0:04	xemacs-2
9547	pts/15	0:00	cs

Legenda: PID = PID; TTY = terminale (virtuale); TIME = tempo di CPU utilizzato; CMD = comando che ha generato il processo.

Per ottenere il nome del terminale corrente:

```
user> tty
/dev/pts/15
```

Il comando `ps` e sue varianti (I)

Per ottenere tutti i processi nel sistema associati ad un terminale (`-a`), full listing (`-f`):

```
user> ps -af
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
lenisa	10922	9560	0	Oct 17	pts/17	0:00	bash
pietro	23410	23409	0	11:07:08	pts/26	0:01	xdvi.bin -name xdvi main.dvi
root	24188	9807	0	12:34:10	pts/13	0:00	ps -af
.....							

Legenda: UID = User Identifier; PPID = Parent PID; c = informazione obsoleta sullo scheduling; STIME = data di inizio del processo.

Il comando `ps` e sue varianti (II)

Per ottenere tutti i processi nel sistema, anche non associati ad un terminale (`-e`), long listing (`-l`):

```
user> ps -el
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
19	T	0	0	0	0	0	SY	?	0		?	0:12	sched
8	S	0	1	0	0	41	20	?	100	?	?	0:03	init
8	S	140	12999	12997	0	56	20	?	278	?	pts/12	0:00	tcsh
8	R	159	9563	9446	0	50	20	?	2110		?	0:30	acroread

....

Legenda: F = flag obsoleti; S = stato del processo (T=stopped); PRI = priorità; NI = nice value (usato per modificare la priorità); ADDR = indirizzo in memoria; SZ = memoria virtuale usata; WCHAN = evento su cui il processo è sleeping.

Terminazione di un processo

Per arrestare un processo in esecuzione si può utilizzare

- la sequenza `Ctrl-c` dal terminale stesso su cui il processo è in esecuzione;
- il comando `kill` seguito dal PID del processo (da qualsiasi terminale):

```
user> ps
```

```
  PID      TTY          TIME CMD
  .....
 28015 pts/14      0:01 xemacs
  .....
```

```
user> kill 28015
```

- il comando `kill` con il segnale `SIGKILL`

```
user> kill -9 28015
```

```
user> kill -s kill 28015
```

Processi in background

Un comando (pipeline, sequenza) seguito da & dà luogo ad uno o più **processi in background**. I processi in background sono eseguiti in una **sottoshell, in parallelo** al processo padre (la shell) e **non** sono controllati da tastiera.

I processi in background sono quindi utili per eseguire task in parallelo che non richiedono controllo da tastiera.

```
user> xemacs &
```

```
[1] 24760
```

[1] è il numero del job, 24760 il PID del processo

```
user> xemacs &
```

```
user> ls -R / >tmp 2>err &
```

Il comando jobs mostra la lista dei job in esecuzione:

```
user> jobs
```

```
[1]    Running                xemacs &
```

```
[2]-  Running                xemacs &
```

```
[3]+  Running                ls -R / >tmp 2>err
```

Controllo di job

Un job si può **sospendere** e poi **rimandare in esecuzione**

```
user> cat >temp      # job in foreground
```

```
Ctrl-z  # sospende il job
```

```
[1]+ Stopped
```

```
user> jobs
```

```
[1]+ Stopped      cat >temp
```

```
user> fg      # fa il resume del job in foreground
```

```
Ctrl-z  # sospende il job
```

```
user> bg      # fa il resume del job in background
```

```
user> kill %1  # termina il job 1
```

```
[1]+ Terminated
```

Monitoraggio della memoria

Il comando `top` fornisce informazioni sulla memoria utilizzata dai processi, che vengono aggiornate ad intervalli di qualche secondo. I processi sono elencati secondo la quantità di tempo di CPU utilizzata.

```
user> top
```

```
load averages:  0.68,  0.39,  0.27                               14:34:55
```

```
245 processes: 235 sleeping, 9 zombie, 1 on cpu
```

```
CPU states: 91.9% idle,  5.8% user,  2.4% kernel,  0.0% iowait,  0.0% swap
```

```
Memory: 768M real, 17M free, 937M swap in use, 759M swap free
```

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	TIME	CPU	COMMAND
12887	root	1	59	0	65M	56M	sleep	105:00	3.71%	Xsun
4210	lenisa	1	48	0	2856K	2312K	cpu	0:00	1.50%	top
9241	root	1	59	0	35M	26M	sleep	15:58	1.47%	Xsun
24389	pietro	4	47	0	28M	25M	sleep	16:30	0.74%	opera
.....										

Legenda: la prima riga indica il carico del sistema nell'ultimo minuto, negli ultimi 5 minuti, negli ultimi 15 minuti, rispettivamente; il carico è espresso come numero di processori necessari per far girare tutti i processi a velocità massima; alla fine della prima riga c'è l'ora; la seconda contiene numero e stato dei processi nel sistema; la terza l'utilizzo della CPU; la quarta informazioni sulla memoria; le restanti righe contengono informazioni sui processi (THR=thread, RES=resident)

Esercizi (I)

- Ridefinire il comando `rm` in modo tale che non sia chiesta conferma prima della cancellazione dei file.
- Definire il comando `rmi` (`rm` interattivo) che chiede conferma prima di rimuovere un file.
- Sapendo che il comando `ps` serve ad elencare i processi del sistema, scrivere una pipeline che fornisca in output il numero di tutti i processi in esecuzione.
- Salvare in un file di testo l'output dell'ultimo evento contenente il comando `ls`.
- Scrivere un comando che fornisce il numero dei comandi contenuti nella history list.

Esercizi (II)

- Scrivere un comando che fornisce i primi 15 comandi della history list.
- Quali sono i comandi Unix disponibili nel sistema che iniziano con `ls`?
- Fornire almeno due modi diversi per ottenere la lista dei file della vostra home directory il cui nome inizia con `al`.
- Qual è l'effetto dei seguenti comandi?
 - `ls -R || (echo file non accessibili > tmp)`
 - `(who | grep rossi) && cd ~rossi`
 - `(cd / ; pwd ; ls | wc -l)`

Esercizi (III)

- Qual è la differenza tra **programma** e **processo**?
- Qual è la differenza tra **processo** e **job**?
- Scrivere una pipeline che fornisca in output il numero di processi appartenenti all'utente `root`.

- Il comando

> `emacs &`

provoca l'avvio di un processo in **background**. Invece il comando

> `emacs`

provoca l'avvio di un processo in **foreground**. Come si può mandare tale processo in esecuzione in background in modo da rendere il terminale nuovamente disponibile per l'invio di ulteriori comandi?