command & history

mobytrick

27 agosto 2014

Sommario

Uno sguardo in profondità su come impostare il salvataggio dei comandi per il loro successivo riutilizzo.

1 Come e cosa salvare

All'avvio del `PC`, il controllo viene assunto dal `BIOS`, un piccolo sistema operativo insito nell'hardware del `PC` stesso. Il `BIOS`, una volta esaurito il suo compito, cede il controllo al sistema operativo che alla fine delle operazioni di *boot* richiede l'immissione delle credenziali (operazione nota anche col termine di "*login*"). Ha inizio quella che viene chiamata "*sessione*" ovvero l'uso del `PC`. Per tutta la sua durata si è sempre sotto il controllo dell'interprete dei comandi, la *shell*, il cui comportamento è ampiamente personalizzabile.

La shell merita un piccolo approfondimento storico. I primi Unix, da cui discende Linux, avevano una shell (autore: Stephen Bourne) certamente un po' spartana ma non per questo non funzionale. Poi, come da tradizione del mondo *open source*, venne proposto un altro interprete dei comandi, chiamato C-shell, incompatibile con la shell originaria. Ora la dualità è superata. Per shell attualmente si intende la `bash` (**B**ourne **a**gain **sh**ell), un prodotto mal riuscito perché infarcito di funzionalità discutibili e mancanza di unitarietà.

Prima di essere operativa, la bash esegue nell'ordine:

1. il file *system wide* di inizializzazione (`/etc/profile`). È uguale per tutti gli utenti. La sua gestione è prerogativa di root
2. il file di inizializzazione dell'utente. Per default è `~/.profile`. Può essere modificato dall'utente, ma è pratica sconsigliabile

3. il file di personalizzazione dell'utente. Per default è `~/.bashrc`. È qui che ciascuno stabilisce le modalità operative della shell.

Gli ultimi 2 file possono mancare. Ma all'atto dell'istituzione di un utenza, entrambi vengono creati in maniera automatica partendo da un canovaccio predefinito. Ad ogni modo l'inizializzazione della shell non segue una scaletta standardizzata e tra le varie distro di Linux ci sono **notevoli** differenze.

Per completezza, quando si decide di terminare la sessione, sono possibili 3 opzioni:

1. il "*logout*" La sessione viene chiusa. Il `PC` rimane acceso e richiede l'immissione delle credenziali

2. il "*reboot*". La sessione viene chiusa ed il `PC` viene portato nella situazione in cui si troverebbe dopo essere stato appena acceso, ovvero non c'è spegnimento ma il controllo viene nuovamente preso dal `BIOS`

3. lo "*shutdown*". Le sessione viene chiusa e senza ulteriori interventi il `PC` viene spento

All'atto della chiusura regolare di una sessione, la bash esegue un file di chiusura, per default `~/.bash_logout`. È bene non far conto su tale possibilità in quanto la sessione potrebbe finire in maniera brusca (ad esempio, per blackout elettrico).

A credenziali immesse, si può usare Linux in maniera tradizionale, "*a riga di comando*". Si possono immettere sia comandi del repertorio Linux sia comandi provvisti dalla shell[1].

Senza tergiversare troppo, conservare i comandi usati può risultare comodo, in quanto è possibile richiamarli, evitando il tedio di dover riscriverli daccapo. Ma esiste anche l'altra faccia della medaglia. Il file che li conserva potrebbe contenere delle informazioni interessanti sia per un eventuale hacker sia per eventuali inquirenti (per la serie: non si sa mai).

Usando una versione recente di Linux in maniera tradizionale i comandi immessi vengono probabilmente salvati. In quest'evenienza la shell permette di lavorarci sopra in maniere diverse

- il `redo` (= la riesecuzione)

[1] più propriamente si parla di `built-in commands`

- l'`editing` (= la modifica seguita dall'esecuzione)
- il `listing` (= la visualizzazione)

Tutto il marchingegno, ancorché complesso, va capito, poiché è influenzato da molte variabili, tutte personalizzabili. Preventivamente si tratta di capire se il salvataggio dei comandi è attivo, e molto probabilmente lo è, oppure no. Ad ogni buon conto la seguente pipe

```
set -o | grep history
```

dà un responso che non ha bisogno di commenti.

Il salvataggio dei comandi funziona come qui di seguito descritto. Il file che contiene i comandi delle precedenti sessioni viene copiato in un file temporaneo. A tale file vengono aggiunti via via i comandi immessi nella sessione in corso. In tal modo è possibile usare con profitto gli eventuali comandi già immessi sia recentemente che in tempi più lontani. Solo al termine della sessione il file temporaneo viene salvato. La modalità è regolata dall'opzione `histappend`, attivata o meno tramite il comando shell `shopt`[2] secondo quando indicato dalla tabella 1 a pag. 3.

settaggio	significato
`shopt -s histappend`	accodamento
`shopt -u histappend`	sovrascrittura

Tabella 1: Salvataggio `history` comandi

Se la sessione termina inopinatamente i comandi usati in essa vanno persi. È possibile premunirsi, pagando un prezzo in fatto di efficienza. Con

```
PROMPT_COMMAND="history -a"
```

si forza la scrittura immediata dei comandi via via che vengono digitati. È possibile aggiungere informazioni di tipo cronologico settando la variabile `HISTTIMEFORMAT` usando la sintassi del comando Unix `date`:

```
HISTTIMEFORMAT="%F %T "
```

[2] il comando shell `shopt` prevede 2 opzioni: `-s` (= set, ovvero attivazione) e `-u` (= unset, ovvero disattivazione)

il comando viene memorizzato facendolo precedere dalla data (`%F`), da uno spazio, dall'ora (`%T`) e da uno spazio. Gli spazi, sopratutto il secondo, sono inseriti per migliorare la leggibilità. La data e l'ora sono espresse rispettivamente nei formati `yyyy-mm-dd` e `hh:mm:ss`.

La dimensione del file temporaneo (`HISTSIZE`) viene misurata in termini di numero di comandi riutilizzabili. Per onorare questo limite (per default è posto a 500), i comandi più vecchi vengono eliminati. Il file che contiene la storia pregressa -`HISTFILE`- è, per default, `~/.bash_history`. Pure per questo file è possibile determinare, oltre al nome, il dimensionamento, tramite `HISTFILESIZE` che, per default, è posto pari a 500. Ed anche in questo caso per restare entro il limite i comandi più datati vengono eliminati.

Il contenuto è influenzato pure da `HISTCONTROL`, dagli effetti generali, e da `HISTIGNORE`, più specifico. Al primo dei due si possono assegnare dei valori simbolici separati dal carattere `:` (doppio punto). I valori sono:

- `ignoredups` evita la presenza consecutiva dello stesso comando
- `ignorespace` i comandi che iniziano con uno o più spazi non vengono memorizzati
- `erasedups` prima di memorizzare il comando, le occorrenze precedenti vengono eliminate

`HISTIGNORE` permette una regolazione abbastanza fine basata sull'esclusione dalla memorizzazione di certi comandi. Un esempio per chiarire il funzionamento. Col seguente settaggio:

```
HISTIGNORE="pwd:date:ls:history*:[fb]g"
```

i comandi `pwd`, `date`, `ls`, `fg`, `bg` -di scarso o nullo valore- e quelli che iniziano con la stringa `history` non vengono salvati. Come si evince dall'esempio, è possibile avvarlersi della sintassi propria delle *Regular Expressions*.

In alcuni casi i comandi immessi sono talmente lunghi da dover essere spezzati in più righe. Sono i cosiddetti "*multi-line command*". Per far sì che vengano memorizzati come singola linea si interviene col comando shell `shopt`:

```
shopt -s cmdhist
```

Accanto a `cmdhist` esiste `lithist`. Tra i due esiste una correlazione. `cmdhist`, se attivato, memorizza i comandi multi line in un'unica riga. `lithist` gestisce l'interruzione delle linee secondo la tabella 2 a pag. 5.

Un breve spaccato esemplificativo di `~/.bashrc` per la gestione del file di comandi con relativo commento:

settaggio	gestione new line
`shopt -u lithist`	mantenuti
`shopt -s lithist`	sostituiti da ; (punto-e-virgola)

Tabella 2: Gestione dei new line

```
HISTCONTROL=ignoredups
HISTFILE=${HOME}/.history
HISTSIZE=1024
export HISTCONTROL HISTFILE HISTSIZE
if [ ! -e ${HISTFILE} ]
  then
    touch ${HISTFILE}
    chmod 600 ${HISTFILE}
fi
```

- `HISTCONTROL=ignoredups` per evitare i doppioni
- `HISTFILE=${HOME}/.history` il file dei comandi pregressi viene chiamato come indicato
- `HISTSIZE=1024` numero di comandi su cui poter operare. Ma verranno salvati solo 500 (il limite per `HISTFILESIZE`). Attenzione a non usare numeri troppo grandi perché impattano negativamente sulle performance
- `export ...` le variabili d'ambiente vengono rese disponibili urbi&orbi
- `if ... fi` se il file dei comandi non c'è viene creato (comando `touch`) e gli viene imposta una maschera (comando `chmod`)

2 Al lavoro

I comandi `fc` (= **f**ix **c**ommand) ed `history`[3] fanno parte dell'interprete dei comandi. Servono per la gestione dei comandi immessi. `history` è dedicato maggiormente alla gestione del file che contiene i comandi mentre `fc` è dedicato esclusivamente ai comandi. A meno che non si abbia dimestichezza e si sappia come operare, la prima cosa da fare è listare i comandi pregressi. Ci sono vari modi:

[3] giova ricordare che `history` è anche il nome di una libreria di funzioni che servono per agire nel file dei comandi tramite programmi

- un comando alla volta
- elenco dei comandi tramite
    1. `fc`
    2. `history`

Per procedere un comando alla volta si usano i tasti freccia «giù» e freccia «su». Si deve forzatamente iniziare con questo. Premendolo, appare l'ultimo comando immesso. Se si batte nuovamente la stessa freccia, appare il penultimo: ovvero, si procede a ritroso nel tempo (con la freccia giù ci si avvicina all'ultimo comando immesso). Una volta ritrovato il comando che interessa, col tasto `"Invio"` lo si manda nuovamente in esecuzione. Sono tuttavia possibili delle modifiche. Il tasto `"Backspace"` cancella il carattere alla sinistra del cursore. Nessun problema per l'immissione: basta battere i caratteri che servono, nel posto giusto. I tasti freccia a sinistra rispettivamente a destra servono per spostarsi nel verso indicato. Ma lentamente perché ci si sposta di 1 carattere alla volta. Alcune *scorciatoie* velocizzano particolari operazioni. Risentono però della configurazione della shell. Infatti possono essere usati (combinazioni di) tasti usuali in ambiente `vi` oppure `emacs`, le due uniche alternative previste. Per impostare l'ambiente simile all'editor `vi` si usa:

```
set -o vi
```

La tabella 3 a pag. 6 riporta le scorciatoie *á la vi* usate più comunemente (per *parola* si intende una stringa delimitata da spazi oppure tabulatori).

spostamento a	inizio parola	`b`
	fine parola	`w`
spostamento a	inizio riga	`0` (zero)
	fine riga	`$`
cancellazione sino a	inizio riga	`d^`
	fine riga	`D`
cancellazione parola	precedente	`CTRL/w`
	successiva	`dw`
annulla	`u`	

Tabella 3: Spostamenti rapidi

Tale modalità può andar bene solo se si opera sugli ultimi comandi, quindi con un uso moderato dei tasti freccia.

I comandi `fc` ed `history` permettono di ottenere in maniera semplice una lista dei comandi pregressi ed il modo per poterli utilizzare. Preventivamente, giova tener presente che l'editor predefinito quando si manipola la history dei comandi è `vi`. Per cambiarlo, conviene posizionare due variabili:

```
FCEDIT=xxx
EDITOR=xxx
```

ove xxx rappresenta l'editor preferito (emacs, nedit, ...)

Il comando per listare i comandi utilizzabili è:

```
fc -l -num
history 16
```

`fc` stampa `num` comandi. Se il numero viene omesso, si usa il valore di default (16). Col comando proposto, `history` viene costretto a stampare 16 comandi. Senza il numero, stamperebbe tutti i comandi. In entrambi i casi ciascun comando è preceduto da un numero.

In prima battuta verranno usati gli "*event designators*". Sono così chiamate le notazioni che iniziano col carattere **!** (punto esclamativo).

Per rimandare in esecuzione quello voluto si usa la seguente notazione:

```
!num
```

ove `num` (notazione assoluta) è tratto dalla lista di cui sopra.

È possibile usare una notazione relativa:

```
!-delta
```

ove `delta` è la distanza dal comando attuale. Quindi `-5` fa riferimento al quintultimo comando immesso e `-1` fa riferimento all'ultimo. Ma in questo caso particolare si preferisce l'abbreviazione `!!` (due punti esclamativi).

Talvolta c'è la necessità di ripetere l'esecuzione di un comando di cui si conosce parzialmente il contenuto. Allora

```
!?stringa?
```

ricerca a ritroso nella storia dei comandi sino ad incontrare quello che contiene con la stringa indicata e lo riesegue. Tale meccanismo potrebbe essere sfruttato per la risottomissione di un comando, il cui nome è a tutti gli effetti una stringa. Si preferisce usare una sintassi più leggera:

```
!comand
```

La riesecuzione incondizionata di un comando non è scevra da trabocchetti. Ad esempio è imprudente la ripetizione del comando `rm` (remove). In Linux, come è noto, la cancellazione di un file è unidirezionale, ovvero non c'è modo di ripristinare un file cancellato per sbaglio.

Quando si immette un comando qualsiasi, la shell opera una specie di dissezione, permettendo di far riferimento a qualsiasi componente ("*word designator*"). La tabella 4 a pag. 8 riporta i riferimenti principali.

`0` (zero)	nome del programma
`^`	primo argomento
`$`	ultimo argomento
`n`	`n`-esimo argomento
`*`	tutti gli argomenti (eccetto il nome)

Tabella 4: Riferimenti alle componenti di un comando

Nel seguente spezzone di comandi

```
touch abracadabra accesso accumulatore zuzzurellone
ls !!:1
```

il primo comando `touch` aggiorna la data di ultima modifica oppure crea quattro file. Il comando successivo `ls` fa riferimento al comando precedente (`!!`) di cui prende il primo (`1` argomento,) cioè `abracadabra` con carattere `:` in funzione di separatore. In definitiva verrà eseguito il comando `ls abracadabra` e sarà questo comando quello memorizzato nel file temporaneo dei comandi.

Oltre ai "*word designator*" ci sono i "*modifier*". Sia i primi che i secondi sono opzionali, però talvolta possono essere utili. Di modificatori ce ne sono molti, ma uno in particolare è utile. Permette la sostituzione di una stringa con un'altra. Vediamo un esempio:

```
rm -f alfa gamma
touch !!:*
ls alba
ls !!:1:s/b/f/
```

il comando `rm` rimuove incondizionatamente (`-f`) due file. Il comando successivo crea gli stessi due file. Qui il word designator `*` sta ad indicare tutti i

parametri. Quali? Quelli del comando precedente (`!!`). Poi si cerca di ottenere le informazioni (comando `ls`) del primo dei due file. Ma viene commesso un errore di battitura.
Quindi: `ls` nuovamente il comando `ls`. Poi, dal comando precedente (`!!`) si prende il primo (`1`) parametro, ovvero la stringa `alba`, ove si sostituisce (`s`) la lettera `b` con la `f`. In definitiva viene costruito il comando `ls alfa`. Sarà sia memorizzato nel file temporaneo che eseguito. Il modificatore `s` sostituisce solo la prima occorrenza della vecchia stringa con la nuova. Se la sostituzione deve essere ripetuta più volte, allora deve essere **preceduto** da un altro modificatore, la lettera `g`, che sta per `global`.

La semplice riesecuzione di 1 solo comando sarebbe ben poca cosa. `fc`, ma non `history`, permette ben di più. Ovvero il recupero non di un solo comando, ma di una serie, con eventuale modifiche e quindi la riesecuzione.

```
fc first last
```

recupera un intervallo di comandi individuati all'inizio da `first` ed alla fine da `last`. Gli estremi possono essere espressi sia in maniera testuale che numerica, assoluta oppure relativa. I comandi vengono visualizzati usando l'editor `vi` oppure quello definito tramite `FCEDIT`. I comandi possono essere modificati a piacimento ed alla fine vengono mandati in esecuzione.

C'è un'ultima modalità d'uso di `fc`:

```
fc -s [old=new] cmd
```

tutte le occorrenze di `old` vengono rimpiazzate da `new` e quindi il comando viene rieseguito. Ma poiché Linux permette di creare degli *alias*[4], è abbastanza comune crearne uno in questa maniera:

```
alias r="fc -s"
```

Ad esempio, con `r make` viene lanciato in esecuzione l'ultimo comando `make` mentre `r` da sola manda in esecuzione l'ultimo comando. Valgono le stesse considerazioni fatte a proposito della ripetizione dei comandi, basandosi sul nome del comando stesso.

Il comando `history` può essere corredato dall'opzione `-c`. Il file temporaneo dei comandi viene cancellato senza possibilità di ripensamenti.

[4] Talvolta si usano comandi molto complessi oppure molto lunghi. Anziché digitarli per intero, viene loro assegnato un nome corto noto come *alias*. Si usa quindi questo, che diventa a tutti gli effetti un comando

3 Asterischi

Se ci si accorge di aver commesso un piccolo errore nel comando precedente, esiste un metodo abbastanza carino nonché veloce per correggerlo:

```
^errore^correzione^
```

il comando viene corretto e rieseguito.

⌣⌢⌣⌢⌣⌢

Se si sta digitando un comando e ci si accorge immediatamente di aver scambiato di posto due caratteri, si rimedia seduta stante tramite `CTRL/T`. In pratica si corregge l'errore battendo 1 carattere. Senza conoscere la notazione ci sarebbero voluti 4 caratteri (2 volte per retrocedere ed i due caratteri ribattuti nella giusta sequenza).

⌣⌢⌣⌢⌣⌢

La notazione `$_` indica l'ultimo parametro del comando precedente. Può risultare utile in molti casi.

⌣⌢⌣⌢⌣⌢

Da moltissimo tempo nelle varie shell è presente un meccanismo noto come "*file name completion*". Non è necessario scrivere per intero il nome di un file, ma basta scrivere solo alcuni caratteri iniziali. Poi si preme il tasto "*Tab(ulator)*". Se c'è 1 solo file che inizia con i caratteri già scritti, il Tab completa automaticamente il nome aggiungendo quanto manca. Se invece ci sono più file che hanno in comune i caratteri iniziali, premendo due volte il Tab si ottengono le alternative. Si batte il carattere di quella desiderata e poi si procede con il Tab. Un esempio per chiarire. Si abbiano i seguenti 4 file:

```
abracadabra
accesso
accumulatore
zuzzurellone
```

Se si battere `z` e poi Tab, c'è il completamento (`zuzzurellone`), poiché non ci sono ambiguità. Se invece si batte `a` ci sono 3 alternative che si ottengono con 2 colpi di Tab. Scegliendo `b` basta un ulteriore Tab per ottenere il nome completo. Se invece si batte `c` e poi Tab viene aggiunta una `c` perché non c'è ambiguità. Poi con 2 Tab vengono mostrati `accesso` ed `accumulatore`. Basta scegliere (1 carattere tra `e` ed `m`) e poi si completa automaticamente con Tab.