Tabelle Riassuntive della shell `bash` in UNIX/Linux

15 ottobre 2019

1 Principali comandi e parole chiave della shell

Argomento	**Comandi visti a lezione**
Shell e sessione	`bash` (altre shell: `sh`, `csh`, `tcsh`, `ksh`), `logout`, `exit`, `history`, `alias`, `unalias`, `man`, `who`, `date`
File e directory	`pwd`, `cd`, `ls`, `mkdir`, `rmdir`, `cp`, `mv`, `rm`, `ln`, `chmod`, `touch`, `mount`, `df`, `du`, `basename`, `cmp`, `diff`, `find`
Processi e job	`ps`, `top`, `kill`, `jobs`, `fg`, `bg`
Visualizzazione	`echo`, `cat`, `more`, `less`, `tail`, `head`
Filtro	`wc`, `uniq`, `grep`, `fgrep`, `egrep`, `sort`, `tr`, `cut`, `paste`, `sed`
Editor	`nano`, `vi`, `emacs`, `xemacs`, `mc`
Script	`set`, `shift`, `if then else fi`, `while do done`, `until do done`, `for in do done`, `case in esac`, `test`, `read`

La filosofia alla base della shell UNIX/Linux è che i comandi devono essere programmi efficienti e di dimensione contenuta in grado di svolgere compiti semplici. Per l'esecuzione di compiti complessi i comandi vanno combinati tramite:

1. pipeline (metacarattere `|`);
2. script (vedi Sezione 5).

Per usare proficuamente la shell ed evitare di digitare molto testo, esistono i seguenti meccanismi:

1. caratteri jolly (metacaratteri `?` e `*`);
2. command completion (tasto `<Tab>`);
3. command line editing della shell (`Ctrl-A`, `Ctrl-E`, tasti cursore, tasto Backspace ecc.);
4. history e ripetizione comandi (metacarattere `!`).

La prima fonte di informazione sulla shell e sui comandi della shell è il manuale online installato sui vari sistemi UNIX/Linux, accessibile tramite il comando `man`. In particolare la sezione 1 del manuale è interamente dedicata a questo scopo.

2 Metacaratteri della shell

Simbolo	Significato	Esempio d'uso
`>`	Ridirezione dell'output	`ls >temp`
`>>`	Ridirezione dell'output (append)	`ls >>temp`
`<`	Ridirezione dell'input	`wc -l <text`
`<<delim`	ridirezione dell'input da linea di comando (here document)	`wc -l <<delim`
`*`	Wildcard: stringa di 0 o più caratteri, ad eccezione del punto (`.`)	`ls *.c`
`?`	Wildcard: un singolo carattere, ad eccezione del punto (`.`)	`ls ?.c`
`[...]`	Wildcard: un singolo carattere tra quelli elencati	`ls [a-zA-Z].bak`
`{...}`	Wildcard: le stringhe specificate all'interno delle parentesi	`ls {prog,doc}*.txt`
`|`	Pipe	`ls | more`
`;`	Sequenza di comandi	`pwd;ls;cd`
`||`	Esecuzione condizionale. Esegue un comando se il precedente fallisce.	`cc prog.c || echo errore`
`&&`	Esecuzione condizionale. Esegue un comando se il precedente termina con successo.	`cc prog.c && a.out`
`(...)`	Raggruppamento di comandi	`(date;ls;pwd)>out.txt`
`#`	Introduce un commento	`ls # lista di file`
`\`	Fa in modo che la shell non interpreti in modo speciale il carattere che segue.	`ls file.\*`
`!`	Ripetizione di comandi memorizzati nell'history list	`!ls`

3 Metacaratteri delle espressioni regolari

metacarattere	tipo	significato
`^`	B	inizio della linea
`$`	B	fine della linea
`\<`	B	inzio di una parola
`\>`	B	fine di una parola
`.`	B	un singolo carattere (qualsiasi)
`[str]`	B	un qualunque carattere in `str`
`[^str]`	B	un qualunque carattere non in `str`
`[a-z]`	B	un qualunque carattere tra `a` e `z`
`\`	B	inibisce l'interpretazione del metacarattere che segue
`*`	B	zero o più ripetizioni dell'elemento precedente
`+`	E	una o più ripetizioni dell'elemento precedente
`?`	E	zero od una ripetizione dell'elemento precedente
`{j,k}`	E	un numero di ripetizioni compreso tra `j` e `k` dell'elemento precedente
`s|t`	E	l'elemento `s` oppure l'elemento `t`
`(exp)`	E	raggruppamento di `exp` come singolo elemento

Nella tabella precedente B (basic) indica che la sequenza di caratteri è utilizzabile sia in `grep` che in `egrep`, mentre E (extended) indica che la sequenza di caratteri è utilizzabile solo in `egrep` (o in `grep` usando l'opzione `-E`).

4 Variabili di stato automatiche

La shell `bash` mette a disposizione numerose variabili di stato; le principali sono:

Variabile	Contenuto
`$?`	*exit status* dell'ultimo comando eseguito dalla shell
`$$`	`PID` della shell corrente
`$!`	il `PID` dell'ultimo comando eseguito in background
`$-`	le opzioni della shell corrente
`$#`	numero dei parametri forniti allo script sulla linea di comando
`$*`, `$@`	lista di tutti i parametri passati allo script sulla linea di comando

In particolare `$$` viene usata per generare nomi di file temporanei che siano unici fra utenti diversi e sessioni di shell diverse, e.g., `/tmp/tmp$$`.

5 Sintassi degli Script

5.1 Variabili ed assegnamento

Le variabili della shell sono stringhe di caratteri a cui è associato un certo spazio in memoria. Il valore di una variabile per la shell è sempre una stringa di caratteri (a meno di non forzare un'interpretazione diversa: si veda, e.g., la Sezione 5.1.4). Le variabili della shell possono essere utilizzate sia sulla linea di comando che negli script. Non c'è dichiarazione esplicita delle variabili (il primo utilizzo di una variabile la dichiara implicitamente).

5.1.1 Assegnamento di una variabile

La sintassi è la seguente: `variabile=valore` (importante: non lasciare spazi a sinistra ed a destra dell'operatore `=` per non confonderlo con l'operatore di confronto). Esempio:

```
> x=valore
```

5.1.2 Utilizzo di una variabile

Per accedere al valore di una variabile si utilizza il `$`:

```
> echo il valore di x e': $x
il valore di x: valore
```

5.1.3 Il comando `export`

Le variabili sono **locali** alla shell o allo script in cui sono definite. Per rendere globale una variabile (*variabile d'ambiente*) si usa il comando `export`:

```
> export x        # promuove x a variabile di ambiente
```

5.1.4 Aritmetica con le variabili della shell

La shell fornisce anche la possibilità di costruire espressioni numeriche complesse, da utilizzare con il comando di test, tramite la sintassi seguente:
`$[expression]`
Ad esempio:

```
> num1=2
> num1=$[$num1*3+1]
> echo $num1
7
```

Esiste anche un'altra sintassi per lo stesso scopo:

```
> num1=2
> num1=$(($num1*3+1))
> echo $num1
7
```

5.2 Controllo di flusso

Nel seguito verranno riportati brevemente i vari comandi per alterare il normale flusso di esecuzione sequenziale degli script. Essi sono analoghi ai vari costrutti sintattici disponibili nei comuni linguaggi imperativi (C, Java ecc.); tuttavia l'espressione che funge da *guardia* (denotata da `condition_command` nel seguito) può essere un qualunque comando UNIX/Linux che restituisca un exit status numerico (con la convenzione che `0` significhi *vero*, mentre un qualunque valore diverso da zero significhi *falso*).

5.2.1 Comando condizionale (`if`)

La sintassi è la seguente (il ramo `else` è opzionale)

```
if condition_command
then
    true_commands
else
    false_commands
fi
```

Se `condition_command` restituisce come exit status zero, allora vengono eseguiti i `true_commands`, altrimenti vengono eseguiti i `false_commands`.

5.2.2 Iterazione indeterminata (`while`)

La sintassi è la seguente:

```
while condition_command
do
    commands
done
```

L'effetto risultante è che vengono eseguiti i comandi `commands` finché `condition_command` restituisce zero come exit status.

5.2.3 Iterazione indeterminata (`until`)

La sintassi è la seguente:

```
until condition_command
do
    commands
done
```

L'effetto risultante è che vengono eseguiti i comandi `commands` finché `condition_command` restituisce un valore diverso da zero come exit status.

5.2.4 Iterazione determinata (`for`)

La sintassi è la seguente:

```
for var in wordlist
do
    commands
done
```

I comandi `commands` vengono ripetuti un numero di volte pari alla lunghezza di `wordlist`, istanziando la variabile `var` ad un valore di una delle componenti di `wordlist` ad ogni iterazione.

Esempio:

```
for i in 1 2 3 4 5 6 7 8 9 10
do
    echo $i
done
```

Sintassi alternativa (simile a C, C++, Java ecc.):

```
for ((i=1; i<=10; i++))
do
    echo $i
done
```

5.2.5 Selezione (`case`)

La sintassi è la seguente:

```
case string in
expression_1)
    commands_1
    ;;
expression_2)
    commands_2
    ;;
...
*)
    default_commands
    ;;
esac
```

L'effetto risultante è che vengono eseguiti i comandi `commands_1`, `commands_2`,... a seconda del fatto che `string` sia uguale a `expression_1`, `expression_2`,...
I comandi `default_commands` vengono eseguiti soltanto se il valore di `string` non coincide con nessuno fra `expression_1`, `expression_2`,...
I valori `expression_1`, `expression_2`,... possono essere specificati usando le solite regole per l'espansione del percorso (caratteri jolly).

5.3 Comando `test`

Se la condizione che si vuole specificare non è esprimibile tramite l'exit status di un "normale" comando, si può utilizzare l'apposito comando `test`:
`test expression`
che restituisce un exit status pari a `0` se `expression` è vera, pari a `1` altrimenti.
Si possono costruire vari tipi di espressioni:

- espressioni che controllano se un file possiede certi attributi:
  `-e` *f* restituisce vero se *f* esiste;
  `-f` *f* restituisce vero se *f* è un file ordinario;
  `-d` *f* restituisce vero se *f* è una directory;
  `-r` *f* restituisce vero se *f* è leggibile dall'utente;
  `-w` *f* restituisce vero se *f* è scrivibile dall'utente;
  `-x` *f* restituisce vero se *f* è eseguibile dall'utente;
- espressioni su stringhe:
  `-z` *str* restituisce vero se *str* è di lunghezza zero;
  `-n` *str* restituisce vero se *str* non è di lunghezza zero;
  str1 `=` *str2* restituisce vero se *str1* è uguale a *str2*;
  str1 `!=` *str2* restituisce vero se *str1* è diversa da *str2*;
- espressioni su valori numerici:
  num1 `-eq` *num2* restituisce vero se *num1* è uguale a *num2*;
  num1 `-ne` *num2* restituisce vero se *num1* non è uguale a *num2*;
  num1 `-lt` *num2* restituisce vero se *num1* è minore di *num2*;
  num1 `-gt` *num2* restituisce vero se *num1* è maggiore di *num2*;
  num1 `-le` *num2* restituisce vero se *num1* è minore o uguale a *num2*;
  num1 `-ge` *num2* restituisce vero se *num1* è maggiore o uguale a *num2*
- espressioni composte:
  exp1 `-a` *exp2* restituisce vero se sono vere sia *exp1* che *exp2*
  exp1 `-o` *exp2* restituisce vero se è vera *exp1* o *exp2*
  `!` *exp* restituisce vero se non è vera *exp*

Invece di `test expression` si può utilizzare la sintassi alternativa `[ expression ]`, facendo attenzione a lasciare uno spazio dopo `[` e prima di `]`.

6 Editor

6.1 Modalità e comandi principali dell'editor `vi`

6.1.1 Edit mode

La modalità di edit è usata principalmente per muovere il cursore nel punto di interesse all'interno del file di testo che si sta editando (è anche la modalità di

default quando si avvia `vi` e ci si può sempre ritornare, nel caso ci si trovi in un'altra modalità, premendo il tasto `Esc`).

Comando	Effetto
`k`, `j`, `h`, `l` (od i tasti cursore)	muove il cursore su, giù, a sinistra ed a destra
`Ctrl-f`, `Ctrl-b`	muove il cursore avanti/indietro di una pagina
`H`, `M`, `L`	muove il cursore alla prima riga, all'ultima od a quella nel mezzo dello schermo
`w`	muove il cursore all'inizio della parola successiva
`e`	muove il cursore alla fine della parola successiva
`b`	muove il cursore all'inizio della parola precedente
`0`	muove il cursore all'inizio della linea corrente
`^`	muove il cursore nella posizione del primo carattere della linea che non sia un `whitespace`
`$`	muove il cursore alla fine della linea corrente
`/`*string*	cerca nel file la stringa *string*
`?`*string*	cerca "all'indietro" nel file la stringa *string*
`n`	cerca l'occorrenza della stringa successiva (in avanti o all'indietro)
n`r`*c*	rimpiazza *n* caratteri con *c* a partire dalla posizione del cursore
n`x`	cancella *n* caratteri dalla posizione del cursore
n`dw`	cancella *n* parole dalla posizione del cursore
n`db`	cancella *n* parole prima del cursore
n`dd`	cancella *n* linee a partire da quella del cursore
`d$`	cancella tutti i caratteri dalla posizione del cursore fino alla fine della linea
`d0`	cancella tutti i caratteri dalla posizione del cursore fino all'inizio della linea
`J`	unisce la linea corrente alla successiva
`p`	incolla il testo copiato/cancellato a destra del cursore
`P`	incolla il testo copiato/cancellato a sinistra del cursore
`yy`	copia la riga corrente in memoria
n`yy`	copia *n* righe in memoria a partire dalla posizione del cursore
`u`	annulla l'ultimo comando
`.`	ripete l'ultimo comando
`ZZ`	termina l'esecuzione di `vi`, salvando le modifiche

6.1.2 Insert mode

Siccome l'edit mode utilizza un gran numero di tasti alfanumerici, per inserire del testo in un file si rende necessaria un'altra modalità: l'**insert** mode. Per uscire dalla modalità di inserimento basta premere il tasto `Esc` o `Ctrl-[` nei terminali senza tasto `Esc`.

Comando	Effetto
`i`	inserisce del testo alla sinistra del cursore
`a`	inserisce del testo alla destra del cursore
`I`	inserisce del testo all'inizio della linea corrente
`A`	inserisce del testo alla fine della linea corrente
`o`	inserisce una nuova linea sotto la posizione del cursore
`O`	inserisce una nuova linea sopra la posizione del cursore

6.1.3 Command mode

Tutti i comandi del **command** mode iniziano con i due punti (`:`); dopo aver inserito tale carattere il cursore si sposta nell'ultima riga dello schermo dove compaiono i caratteri del comando successivamente digitati. La pressione del tasto invio provoca l'esecuzione del comando.

Comando	Effetto
`:q`	termina `vi` se non vi sono delle modifiche non salvate
`:q!`	termina `vi` perdendo le eventuali modifiche non salvate
`:w`	salva il file originale
`:wq`	salva il file originale e termina `vi` (stesso effetto di `ZZ`)
`:w` *file*	salva il contenuto nel file *file*
`:r` *file*	legge il contenuto del file *file* inserendolo dopo la posizione del cursore
`:e` *file*	edita il file *file*, sostituendo il contenuto corrente
`:f` *file*	cambia il nome del contenuto corrente in *file*
`:f`	stampa il nome e lo stato del testo corrente
`:`*n*	sposta il cursore alla linea *n*
`:/`*str*`/`	sposta il cursore alla prossima linea contenente *str*
`:s/`*str1*`/`*str2*`/`	sostituisce la prima occorrenza di *str1* sulla linea con *str2*
`:set` *option*	definisce un'opzione, e.g. `:set` *number* aggiunge i numeri di linea

6.2 Emacs: il minibuffer ed alcuni comandi utili

Sotto la mode line si trova il **minibuffer**, che è una parte dell'interfaccia di Emacs che si occupa di visualizzare e di permettere all'utente di editare/completare i comandi.

Comando	Effetto
`M-x text-mode`	cambia il major mode in text (modalità testo inglese)
`C-h m`	informazioni sul modo corrente
`C-x C-f` *file*	apre il file *file*
`C-x C-s`	salva il buffer corrente
`C-<Spacebar>`	imposta il mark (inizio di una regione di testo su cui eseguire un comando in seguito)
`M-w`	copia la regione di testo compresa fra il mark e la posizione del cursore
`C-w`	taglia la regione di testo compresa fra il mark e la posizione del cursore
`C-y`	incolla la regione di testo copiata/tagliata in precedenza
`C-x u`	undo
`C-x C-c`	termina Emacs

6.2.1 Comandi essenziali di `mc` (Midnight Commander)

Midnight Commander, che si esegue con `mc`, è un browser per l'esplorazione del filesystem da console, che permette anche di visualizzare e modificare i file.

Pur non utilizzando X, fornisce una interfaccia più user-friendly rispetto ad altri editor e permette di:

- navigare tra file e cartelle utilizzando le frecce e il tasto `Invio`;
- visualizzare un file con `F3`;
- modificare un file con `F4`;
- salvare un file con `F2` (in edit mode);
- uscire dall'editor e da `mc` con `F10`.