NOME

bash − GNU Bourne−Again SHell

SINTASSI

bash [opzioni] [file]

COPYRIGHT

DESCRIZIONE

Bash è un interprete di linguaggio di comandi **sh**−compatibile che esegue i comandi letti dallo standard input o da un file. **Bash** inoltre incorpora utili caratteristiche delle *Korn* e *C* shell (**ksh** e **csh**).

Bash è, in definitiva, progettata per essere una implementazione conforme alle IEEE Posix Shell and Tools specification (IEEE Working Group 1003.2).

OPZIONI

In aggiunta alle opzioni della shell a "singolo carattere" documentate nella descrizione del comando incorporato **set**, quando è lanciata, **bash** interpreta i seguenti flag:

−c *stringa* Se è presente il flag **−c**, allora i comandi sono letti da *stringa*. Se vi sono argomenti dopo la *stringa*, essi sono assegnati ai parametri posizionali, partendo da **$0**.

−i Se è presente il flag **−i**, la shell è *interattiva*.

−s Se è presente il flag **−s**, o se non rimane alcun argomento dopo che sono state esaminate le opzioni, allora i comandi sono letti dallo standard input. Questa opzione permette di predisporre i parametri posizionali quando si chiama una shell interattiva.

− Un singolo − segna la fine delle opzioni e disabilita l'ulteriore processo di opzioni. Qualsiasi argomento dopo il − è trattato come nome di file e argomento. Un argomento −− è equivalente a un argomento −.

Bash interpreta anche un certo numero di opzioni "multi carattere". Queste opzioni devono apparire nella linea di comando prima che siano rilevate delle opzioni a "singolo carattere".

−norc Non legge né esegue il file personale di inizializzazione *~/.bashrc* se la shell è interattiva. Questa opzione è attiva per default se la shell è chiamata come **sh**.

−noprofile Non legge né il file di inizializzazione generico di sistema */etc/profile* né alcuno dei file personali di inizializzazione *~/.bash_profile*, *~/.bash_login*, o *~/.profile*. Per default, **bash** normalmente legge questi file quando è chiamata come shell di login (si veda **ESECUZIONE** più avanti).

−rcfile *file* Esegue comandi letti da *file* invece che dal file personale di inizializzazione standard *~/.bashrc*, se la shell è interattiva (si veda **ESECUZIONE** più avanti).

−version Mostra il numero di versione della istanza di **bash** quando parte.

−quiet Non è "loquace" quando parte (non mostra la versione della shell né qualunque altra informazione). Questo è il default.

−login Fa agire **bash** come se fosse stata chiamata come shell di login.

−nobraceexpansion
Non esegue l'espansione delle parentesi graffe (si veda **Espansione delle parentesi graffe** più avanti).

−nolineediting
Non usa la libreria GNU *readline* per leggere le linee di comando, se interattiva.

−posix Cambia il comportamento di bash, dove le operazioni di default differiscono dallo standard Posix 1003.2, per combaciare con lo standard.

ARGOMENTI

Se rimangono argomenti dopo che sono state processate le opzioni, e né l'opzione **−c** né l'opzione **−s** sono state fornite, il primo argomento è assunto essere il nome del file che contiene i comandi di shell. Se **bash** è chiamata in questo modo, **$0** è posto uguale al nome del file, e i parametri posizionali sono posti uguali agli argomenti rimasti. **Bash** legge ed esegue comandi da questo file, poi esce. Lo stato di uscita di **Bash** è lo stato di uscita dell'ultimo comando eseguito nello script.

DEFINIZIONI

blank Uno spazio o carattere di tabulazione (tab).

parola Una sequenza di caratteri considerata come una singola unità dalla shell. Anche noto come **token**.

nome Una *parola* che consiste solo di caratteri alfanumerici e underscore (_), e comincia con un carattere alfabetico o un underscore. Anche indicata come **identificatore**.

metacarattere

Un carattere che, quando non quotato, separa le parole. Uno dei seguenti:
| & ; ( ) < > spazio tab

operatore di controllo

Un *token* che realizza una funzione di controllo. Esso è uno dei seguenti simboli:
|| & && ; ;; ( ) | <newline>

PAROLE RISERVATE

Le *parole riservate* sono parole che hanno un significato speciale per la shell. Le seguenti parole sono riconosciute come riservate quando non quotate e sono o la prima parola di un comando semplice (si veda **GRAMMATICA DELLA SHELL** più avanti) o la terza parola di un comando **case** o di un comando **for** :

! case do done elif else esac fi for function if in select then until while { }

GRAMMATICA DELLA SHELL

Comandi semplici

Un *comando semplice* è una sequenza opzionale di assegnamenti di variabile seguita da parole, separate da *blank*, e ridirezioni, e terminati da un *operatore di controllo*. La prima parola specifica il comando che deve essere eseguito. Le rimanenti parole sono passate come argomenti per il comando chiamato.

Il valore di ritorno di un *comando semplice* è il suo stato di uscita, o 128+*n* se il comando è terminato da un segnale *n*.

Pipeline

Una *pipeline* è una sequenza di uno o più comandi separati dal carattere |. Il formato per una pipeline è:

[!] *comando* [| *comando2* ...]

Lo standard output di *comando* è connesso allo standard input di *comando2*. Questa connessione è effettuata prima di qualsiasi ridirezione specificata dal comando (si veda **RIDIREZIONE** più avanti).

Se la parola riservata **!** precede una pipeline, lo stato di uscita di quella pipeline è il NOT logico dello stato di uscita dell'ultimo comando. Altrimenti, lo stato della pipeline è lo stato di uscita dell'ultimo comando. La shell aspetta che tutti i comandi nella pipeline terminino, prima di ritornare un valore.

Ogni comando in una pipeline è eseguito come un processo separato (cioè, in una subshell).

Liste

Una *lista* è una sequenza di una o più pipeline separate da uno degli operatori **;**, **&**, **&&**, o ||, e terminata da uno di **;**, **&**, o **<newline>**.

Di questi operatori di lista, **&&** e || hanno uguali precedenze, seguiti da **;** e **&,** che hanno uguali precedenze.

Se un comando è terminato dall'operatore di controllo **&**, la shell esegue il comando in *background* in una subshell. La shell non aspetta che il comando finisca, e lo stato di ritorno è 0. I comandi separati da un **;** sono eseguiti sequenzialmente; la shell aspetta che ogni comando, a turno, termini. Lo stato di ritorno è lo stato di uscita dell'ultimo comando eseguito.

L'operatore di controllo **&&** e || denotano liste AND e liste OR, rispettivamente. Una lista AND ha la forma

comando **&&** *comando2*

comando2 è eseguito se, e solo se, *comando* ritorna uno stato di uscita di zero.

Una lista OR ha la forma

comando || *comando2*

comando2 è eseguito se e solo se *comando* ritorna uno stato di uscita diverso da zero. Lo stato di ritorno di liste AND e OR è lo stato di uscita dell'ultimo comando eseguito nella lista.

Comandi composti

Un *comando composto* è uno dei seguenti:

(*lista*) *lista* è eseguita in una subshell. Assegnamenti di variabile e comandi incorporati che influenzano l'ambiente della shell non lasciano effetti dopo che il comando è completato. Lo stato di ritorno è lo stato di uscita di *lista*.

{ *lista*; }
lista è semplicemente eseguita nell'ambiente di shell corrente. Questo è conosciuto come *group command*. Lo stato di ritorno è lo stato di uscita di *lista*.

for *nome* [**in** *parola*;] **do** *lista* ; **done**
La lista di parole seguenti **in** è espansa, generando una lista di elementi. La variabile *nome* è posta, di volta in volta, a ciascun elemento di questa lista e *lista* è eseguita ogni volta. Se la **in** *parola* è omessa, il comando **for** esegue *lista* una volta per ogni parametro posizionale che è posto (si veda **PARAMETRI** più avanti).

select *nome* [**in** *parola*;] **do** *lista* ; **done**
La lista di parole seguenti **in** è espansa, generando una lista di elementi. L'insieme delle parole espanse è stampato sullo standard error, ognuna preceduta da un numero. Se la **in** *parola* è omessa, sono stampati i parametri posizionali (si veda **PARAMETRI** più avanti). È poi mostrato il prompt **PS3** ed è letta una linea dallo standard input. Se la linea consiste del numero corrispondente a una delle parole mostrate, allora il valore di *nome* è posto a quella parola. Se la linea è vuota, le parole e il prompt sono mostrati di nuovo. Se è letto EOF, il comando termina. Qualsiasi altro valore letto fa sì che *nome* sia posto al valore nullo. La linea letta è salvata nella variabile **REPLY**. La *lista* è eseguita dopo ciascuna selezione fino a che non sia eseguito un comando **break** o **return**. Lo stato di uscita di **select** è lo stato di uscita dell'ultimo comando eseguito in *lista*, o zero se nessun comando è stato eseguito.

case *parola* **in** [*pattern* [| *pattern*] ...) *lista* ;;] ... **esac**
Un comando **case** prima espande *parola*, e prova a confrontarla, a turno, con ognuno dei *pattern*, usando le stesse regole di confronto della espansione di percorso (si veda **Espansione di percorso** più avanti). Quando viene trovata una coincidenza, è eseguita la *lista* corrispondente. Dopo il primo confronto riuscito, non ne viene controllato nessuno dei successivi. Lo stato di uscita è zero se nessun pattern combacia. Altrimenti, esso è lo stato di uscita dell'ultimo comando eseguito in *lista*.

if *lista* **then** *lista* [**elif** *lista* **then** *lista*] ... [**else** *lista*] **fi**
La *lista* dopo **if** è eseguita. Se il suo stato di uscita è zero, è eseguita la *lista* dopo **then** . Altrimenti, è eseguita a turno ciascuna *lista* dopo **elif** , e se il suo stato di uscita è zero, è eseguita la corrisponddente *lista* dopo **then** e il comando termina. Altrimenti, se presente, è eseguita, la *lista* dopo **else** . Lo stato di uscita è lo stato di uscita dell'ultimo comando eseguito, o zero se nessuna delle conditioni provate era vera.

while *lista* **do** *lista* **done**
until *lista* **do** *lista* **done**
Il comando **while** esegue continuamente la *lista* dopo **do** fin tanto che l'ultimo comando in *lista* ritorna uno stato di uscita di zero. Il comando **until** è identico al comando **while**, con la sola differenza che il risultato del test è negato; la *lista* dopo **do** è eseguita fin tanto che l'ultimo comando in *lista* ritorna uno stato di uscita diverso da zero. Lo stato di uscita dei comandi **while** e **until** è lo stato di uscita dell'ultimo comando eseguito nella *lista* dopo **do** o zero se non ne è stato eseguito alcuno.

[**function**] *nome* () { *lista*; }
Questo definisce una funzione chiamata *nome*. Il *corpo* della funzione è la *lista* di comandi tra { e }. Questa lista è eseguita ogni volta che *nome* è specificato come nome di un comando semplice. Lo stato di uscita di una funzione è lo stato di uscita dell'ultimo comando eseguito nel corpo (si

veda **FUNZIONI** più avanti).

COMMENTI

In una shell non interattiva, o una shell interattiva in cui è abilitata l'opzione **-o interactive−comments** del comando incorporato **set**, una parola che inizia con **#** fa sì che la parola e tutti i rimanenti caratteri su quella linea siano ignorati. Una shell interattiva senza l'opzione **-o interactive−comments** abilitata non permette i commenti.

QUOTATURA

La *Quotatura* è usata per togliere il significato speciale, per la shell, di certi caratteri o parole. La quotatura può essere usata per disabilitare speciali trattamenti per i caratteri speciali, per prevenire che le parole riservate siano riconosciute come tali, e per prevenire l'espansione di parametro.

Ciascuno dei *metacaratteri* elencati prima sotto **DEFINIZIONI** ha un significato speciale per la shell e deve essere quotato se esso deve rappresentare se stesso. Vi sono tre meccanismi di quotatura: *caratteri di escape*, apostrofi, e virgolette.

Un backslash (\) non quotato è il *carattere di escape*. Esso conserva il valore letterale del successivo carattere, con l'eccezione di <newline>. Se vi è una coppia \<newline> , e il backslash non è quotato, il \<newline> è trattato come una continuazione di linea (cioè, viene realmente ignorato).

Racchiudere caratteri in apostrofi conserva il valore letterale di ogni carattere all'interno. Un apostrofo non può trovarsi tra apostrofi, nemmeno quando preceduto da un backslash.

Racchiudere caratteri in virgolette conserva il valore letterale di tutti i caratteri all'interno, con le eccezioni di **$**, **‘**, e \. I caratteri **$** e **‘** conservano il loro significato speciale anche tra virgolette. Il backslash mantiene il suo significato speciale solo quando seguito da uno dei seguenti caratteri: **$**, **‘**, **"** , \, o **<newline>**. Le virgolette possono essere racchiuse fra virgolette facendole precedere da un backslash.

I parametri speciali ***** e **@** hanno un significato speciale quando sono tra virgolette (si veda **PARAMETRI** più avanti).

PARAMETRI

Un *parametro* è una entità che immagazzina valori, qualcosa come una variabile in un linguaggio di programmazione convenzionale. Esso può essere un *nome*, un numero, o uno degli speciali caratteri elencati più avanti sotto **Parametri speciali**. Per gli scopi della shell, una *variabile* è un parametro indicato da un *nome*.

Un parametro è impostato se ad esso è stato assegnato un valore. La stringa nulla è un valore valido. Una volta che una variabile è impostata, essa può essere eliminata solo usando il comando incorporato **unset** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti).

Una *variabile* può essere assegnata da una istruzione della forma

nome=[*valore*]

Se *valore* non è dato, alla variabile è assegnata la stringa nulla. Tutti i *valori* sono sottoposti alla espansione della tilde, espansione di parametro e variabile, sostituzione di comando, espansione aritmetica, e rimozione dei caratteri di quotatura. Se la variabile ha il suo attributo **−i** posto (si veda **declare** più avanti in **COMANDI INCORPORATI DELLA SHELL**) allora *valore* è sottoposto alla espansione aritmetica perfino se la sintasi $[...] non appare. La suddivisione in parole non è effettuata, con l'eccezione di **"$@"** come spiegato più avanti sotto **Parametri speciali**. L'espansione di percorso non è effettuata.

Parametri posizionali

Un *parametro posizionale* è un parametro indicato da una o più cifre, diverse dalla singola cifra 0. I parametri posizionali sono assegnati dagli argomenti della shell quando questa è chiamata, e possono essere riassegnati usando il comando incorporato **set .** I parametri posizionali non possono essere assegnati con istruzioni di assegnamento. I parametri posizionali sono temporaneamente sostituiti quando è eseguita una funzione di shell (si veda **FUNZIONI** più avanti).

Quando si espande un parametro posizionale consistente di più di una sola cifra, esso deve essere racchiuso tra parentesi graffe (si veda **ESPANSIONE** più avanti).

Parametri speciali

La shell tratta molti parametri in modo speciale. Questi parametri possono solo essere referenziati; il loro assegnamento non è permesso.

***** Si espande nei parametri posizionali, a partire da uno. Quando l'espansione avviene tra virgolette, esso si espande in una singola parola con il valore di ogni parametro separato dal primo carattere della variabile speciale **IFS**. Cioè, "**$***" è equivalente a "**$1***c***$2***c***...**", dove *c* è il primo carattere del valore della variabile **IFS**. Se **IFS** è nulla o non posta, i parametri sono separati da spazi.

@ Si espande nei parametri posizionali, a partire da uno. Quando l'espansione avviene tra virgolette, ogni parametro si espande in una parola separata. Cioè, " **$@**" è equivalente a "**$1**" "**$2**" ... Quando non vi è alcun parametro posizionale, "**$@**" e **$@** si espandono in nulla (cioè, sono rimossi).

Si espande nel numero di parametri posizionali in decimale.

? Si espande nello stato della pipeline in foreground più recentemente eseguita.

− Si espande nei flag di opzione correnti come specificato in base alla chiamata, dal comando incorporato **set**, o quelli posti dalla shell stessa (come il flag **−i**).

$ Si espande nel ID di processo della shell. In una subshell (), esso si espande nel ID di processo della shell corrente, non la subshell.

! Si espande nel ID di processo del comando in background (asincrono) più recentemente eseguito.

0 Si espande nel nome della shell o script di shell. Questo è posto alla inizializzazione della shell. Se **bash** è chiamata con un file di comandi, **$0** è posto al nome di quel file. Se **bash** è avviata con l'opzione **−c**, allora **$0** è posto al primo argomento dopo la stringa che deve essere eseguita, se ve ne è una presente. Altrimenti, esso è posto al percorso usato per chiamare **bash**, come dato dall'argomento zero.

_ Si espande nell'ultimo argomento del precedente comando, dopo l'espansione. È anche posto al nome completo di ogni comando eseguito e messo nell'ambiente ed esportato per quel comando.

Variabili di shell

Le seguenti variabili sono poste dalla shell:

PPID L'ID di processo del genitore della shell.

PWD La directory di lavoro corrente come posta dal comando **cd**.

OLDPWD

La precedente directory di lavoro come posta dal comando **cd**.

REPLY

Posta alla linea di input letta dal comando incorporato **read** quando nessun argomento è dato.

UID Si espande all'ID di utente dell'utente corrente, inizializzata all'avvio della shell.

EUID Si espande all'effettivo ID di utente dell'utente corrente, inizializzata all'avvio della shell.

BASH Si espande al nome completo usato per chiamare questa istanza di **bash**.

BASH_VERSION

Si espande al numero di versione della istanza di **bash**.

SHLVL

Incrementato di uno ogni volta che una istanza di **bash** è avviata.

RANDOM

Ogni volta che questo parametro è referenziato, viene generato un numero intero casuale. La sequenza di numeri casuali può essere inizializzata assegnando un valore a **RANDOM**. Se **RANDOM** viene eliminato, perde le sue speciali proprietà, perfino se esso viene successivamente rimesso.

SECONDS

Ogni volta che questo parametro è referenziato, viene ritornato il numero di secondi dalla chiamata della shell. Se un valore è assegnato a **SECONDS**, il valore ritornato in base ai riferimenti successivi è il numero di secondi trascorsi dall'assegnamento più il valore assegnato. Se **SECONDS** viene eliminato, perde le sue speciali proprietà, perfino se esso viene successivamente rimesso.

LINENO

Ogni volta che questo parametro è referenziato, la shell sostituisce un numero decimale che rappresenta il numero di sequenza della linea corrente (partendo da 1) dentro uno script o funzione.

Quando non in uno script o funzione, non è garantito che il valore sostituito sia significativo. Quando in una funzione, il valore non è il numero della linea sorgente su cui appare il comando (quella informazione è andata persa dal momento che la funzione viene eseguita), ma è una approssimazione del numero di *comandi semplici* eseguiti nella funzione corrente. Se **LINENO** viene eliminato, perde le sue speciali proprietà, perfino se esso viene successivamente rimesso.

HISTCMD

Il numero di storia, o indice nella lista della storia, del comando corrente. Se **HISTCMD** viene eliminato, perde le sue speciali proprietà, perfino se esso viene successivamente rimesso.

OPTARG

Il valore dell'ultimo argomento opzione processato dal comando incorporato **getopts** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti).

OPTIND

L'indice del prossimo argomento che deve essere processato dal comando incorporato **getopts** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti).

HOSTTYPE

Automaticamente posto a una stringa che univocamente descrive il tipo di macchina su cui **bash** sta girando. Il default è dipendente dal sistema.

OSTYPE

Automaticamente posto a una stringa che descrive il sistema operativo su cui **bash** sta girando. Il default è dipendente dal sistema.

Le seguenti variabili sono usate dalla shell. In alcuni casi, **bash** assegna un valore di default a una variabile; questi casi sono elencati più avanti.

IFS L' *Internal Field Separator* (separatore di campo interno) che è usato per la suddivisione in parole dopo l'espansione e per dividere le linee in parole con il comando incorporato **read**. Il valore di default è ''<space><tab><newline>''.

PATH Il percorso di ricerca dei comandi. Esso è un elenco di directory, separate da ":", nelle quali la shell cerca i comandi (si veda **ESECUZIONE DEI COMANDI** più avanti). Il percorso di default è dipendente dal sistema ed è posto dall'amministratore che installa **bash**. Un valore comune è ''/usr/gnu/bin:/usr/local/bin:/usr/ucb:/bin:/usr/bin:.''.

HOME

La home directory dell'utente corrente; l'argomento di default per il comando incorporato **cd** .

CDPATH

Il percorso di ricerca per il comando **cd**. Questo è un elenco di directory, separate da ":", nelle quali la shell cerca la directory di destinazione specificata dal comando **cd**. Un valore di esempio è ''.:~:/usr''.

ENV Se questo parametro è posto quando **bash** sta eseguendo uno script di shell, il suo valore è interpretato come un nome di file che contiene comandi per inizializzare la shell, come in *.bashrc*. Il valore di **ENV** è sottoposto ad espansione di parametro, sostituzione di comando, ed espansione aritmetica prima di essere interpretato come un percorso. **PATH** non è usato per cercare il nome risultante.

MAIL Se questo parametro è posto a un nome di file e la variabile **MAILPATH** non è impostata, **bash** informa l'utente dell'arrivo di posta nel file specificato.

MAILCHECK

Specifica quanto spesso (in secondi) **bash** controlla la posta. Il default è 60 secondi. Quando è il momento di controllare la posta, la shell lo fa prima di dare il prompt. Se questa variabile non è impostata, la shell disabilita il controllo della posta.

MAILPATH

Una lista di percorsi di nomi, separati da ":" da usare per il controllo della posta. Il messaggio che deve essere stampato può essere specificato separando il percorso dal messaggio con un '?'. $_ sta per il nome del file di posta corrente. Per esempio:

MAILPATH='/usr/spool/mail/bfox?"You have mail":~/shell-mail?"$_ has mail!"'

Bash fornisce un valore di default per questa variabile, ma la locazione dei file di posta degli utenti che è usata è dipendente dal sistema (per esempio, /usr/spool/mail/**$USER**).

MAIL_WARNING
Se è impostata, e viene fatto accesso al file che **bash** controlla per la posta, dopo l'ultima volta che è stato controllato, viene stampato il messaggio ''The mail in *mailfile* has been read''.

PS1 Il valore di questo parametro è espanso (si veda **PROMPTING** più avanti) e usato come stringa del prompt primario. Il valore di default è ''**bash\$** ''.

PS2 Il valore di questo parametro è espanso e usato come stringa del prompt secondario. Il default è ''> ''.

PS3 Il valore di questo parametro è usato come prompt per il comando *select* (si veda **GRAMMATICA DELLA SHELL** sopra).

PS4 Il valore di questo parametro è espanso ed il valore è stampato prima di ogni comando che **bash** mostra durante un trace di esecuzione. Il primo carattere di **PS4** è replicato tante volte, quanto necessario, per indicare livelli multipli di indirezione. Il default è ''+ ''.

HISTSIZE
Il numero di comandi da ricordare nella storia dei comandi (si veda **STORIA** più avanti). Il valore di default è 500.

HISTFILE
Il nome del file nel quale è salvata la storia dei comandi (si veda **STORIA** più avanti). Il valore di default è *~/.bash_history*. Se non posto, la storia dei comandi non è salvata quando una shell interattiva termina.

HISTFILESIZE
Il numero massimo di linee contenute nel file di storia. Quando a questa variabile è assegnato un valore, il file di storia è accorciato, se necessario, per contenere non più di quel numero di linee. Il valore di default è 500.

OPTERR
Se posto al valore 1, **bash** mostra i messaggi di errore generati dal comando incorporato **getopts** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti). **OPTERR** è inizializzato ad 1 ogni volta che è chiamata la shell o è eseguito uno script di shell.

PROMPT_COMMAND
Se posto, il valore è eseguito come un comando prima di emettere ogni prompt primario.

IGNOREEOF
Controlla l'azione della shell al ricevimento di un carattere **EOF** come unico input. Se posto, il valore è il numero di caratteri **EOF** consecutivi digitati come primi caratteri su una linea di input prima che **bash** esca. Se la variabile esiste ma non ha un valore numerico, o non ha alcun valore, il valore di default è 10. Se essa non esiste, **EOF** significa la fine dell'input per la shell. Questo è valido solo per le shell interattive.

TMOUT
Se posto a un valore più grande di zero, il valore è interpretato come il numero di secondi da aspettare per l'input dopo l'emissione del prompt primario. Se non arriva l'input **Bash** termina dopo aver aspettato per quel numero di secondi.

FCEDIT
L'editor di default per il comando incorporato **fc**.

FIGNORE
Una lista di suffissi, separati da ":", da ignorare quando si effettua il completamento del nome di file (si veda **READLINE** più avanti). Un nome di file il cui suffisso combacia con uno di quelli presenti in **FIGNORE** è escluso dalla lista dei nomi di file che combaciano. Un valore di esempio è ''.o:~''.

INPUTRC
Il nome di file, per il file di avvio di readline che scavalca il default *~/.inputrc* (si veda **READLINE** più avanti).

notify Se posto, **bash** informa immediatamente dei job in background terminati, piuttosto che aspettare fino a prima di stampare il successivo prompt primario (si veda anche l'opzione **−b** del comando incorporato **set**).

history_control
HISTCONTROL
Se posto al valore *ignorespace*, le linee che iniziano con un carattere **space** non sono inserite nella lista della storia. Se posto al valore *ignoredups*, le linee che coincidono con l'ultima linea della storia non sono inserite. Il valore *ignoreboth* combina le due opzioni. Se non posto, o se posto a qualsiasi altro valore che quelli sopra, tutte le linee lette dall'analizzatore sono salvate nella lista della storia.

command_oriented_history
Se posta, **bash** tenta di salvare tutte le linee di un comando su più linee nella stessa posizione della storia. Questo permette un facile re−editing dei comandi su più linee.

glob_dot_filenames
Se posto, **bash** include i nomi di file che iniziano con un '.' nel risultato della espansione di percorso.

allow_null_glob_expansion
Se posto, **bash** permette ai pattern di percorso che non combaciano con alcun file (si veda **Espansione dei percorsi** più avanti) di espandersi in una stringa nulla, piuttosto che in loro stessi.

histchars
I due o tre caratteri che controllano l'espansione della storia e la tokenizzazione (si veda **ESPANSIONE DELLA STORIA** più avanti). Il primo carattere è il *carattere di espansione storia*, cioè, il carattere che segnala l'inizio di una espansione della storia, normalmente '**!**'. Il secondo carattere è il carattere di *sostituzione rapida*, che è usato come scorciatoia per rieseguire il comando precedentemente inserito, sostituendo una stringa con un'altra nel comando. Il default è '**^**'. Il terzo carattere opzionale è il carattere che indica che il resto della linea è un commento, quando trovato come primo carattere di una parola, normalmente '**#**'. Il carattere di commento della storia fa sì che la sostituzione della storia venga saltata per le rimanenti parole sulla linea. Esso non provoca necessariamente che l'analizzatore della shell tratti il resto della linea come un commento.

nolinks
Se posto, la shell non segue i link simbolici quando esegue dei comandi che cambiano la directory di lavoro corrente. Essa usa invece la struttura fisica della directory. Per default, **bash** segue la catena logica delle directory quando effettua comandi che cambiano la directory corrente, come **cd**. Si veda anche la descrizione della opzione **−P** del comando incorporato **set** (**COMANDI INCORPORATI DELLA SHELL** più avanti).

hostname_completion_file
HOSTFILE
Contiene il nome di un file nello stesso formato di */etc/hosts* che dovrà essere letto quando la shell ha bisogno di completare un nome di host. Il file può essere cambiato interattivamente; la prossima volta che è tentato il completamento del nome di host, **bash** aggiunge il contenuto del nuovo file al database già esistente.

noclobber
Se posto, **bash** non sovrascrive un file esistente con gli operatori di ridirezione **>**, **>&**, e **<>**. Questa variabile può essere ignorata quando si creano file di output usando l'operatore di ridirezione **>|** invece di **>** (si veda anche l'opzione **−C** del comando incorporato **set**).

auto_resume
Questa variabile controlla il modo con cui la shell interagisce con l'utente ed il job control. Se questa variabile è posta, i comandi semplici di una singola parola senza ridirezioni sono trattati come candidati per la ripresa di un esistente job fermato. Non è permessa alcuna ambiguità; se vi è più di un job che comincia con la stringa digitata, è selezionato il job acceduto più recentemente. Il *nome* di un job fermo, in questo contesto, è la linea di comando usata per avviarlo. Se posto al valore *exact*, la stringa fornita deve combaciare esattamente con il nome di un job fermo; se posto a *substring*, la stringa fornita deve combaciare con una sottostringa del nome di un job fermo. il valore *substring* fornisce funzionalità analoghe all'id di job **%?** (si veda **JOB CONTROL** più avanti). Se posto a qulsiasi altro valore, la stringa fornita deve essere un prefiso del nome di un job

fermo; questo da funzionalità analoghe all'id di job **%**.

no_exit_on_failed_exec
Se questa variabile esiste, una shell non interattiva non terminerà se non può eseguire il file specificato nel comando incorporato **exec**. Una shell interattiva non termina se **exec** fallisce.

cdable_vars
Se questa è posta, un argomento per il comando incorporato **cd** che non è una directory è assunto essere il nome di una variabile il cui valore è la directory su cui andare.

ESPANSIONE

L'espansione è effettuata sulla linea di comando dopo che essa è stata divisa in parole. Vi sono sette tipi di espansione effettuati: *espansione delle parentesi graffe*, *espansione della tilde*, *espansione di parametro e variabile*, *sostituzione di comando*, *espansione aritmetica*, *suddivisione in parole*, ed *espansione di percorsi*.

L'ordine di espansione è: espansione delle parentesi graffe, espansione della tilde, sostituzione di parametro, variabile, comando e aritmetica (fatta da sinistra a destra), suddivisione in parole ed espansione di percorso.

Sui sistemi che possono supportarla, è disponibile una espansione aggiuntiva: la *sostituzione di processo*.

Solo l'espansione delle parentesi graffe, la suddivisione in parole, e l'espansione di percorso possono cambiare il numero di parole della espansione; le altre espansioni espandono una singola parola in una singola parola. La sola eccezione a questo è l'espansione di "**$@**" come spiegato sopra (si veda **PARAMETRI**).

Espansione delle parentesi graffe

L' *espansione delle parentesi graffe* è un meccanismo con il quale possono essere generate stringhe arbitrarie. Questo meccanismo è simile alla *espansione di percorso*, ma non è necessario che i nomi di file generati esistano. I pattern che devono subire l'espansione delle parentesi graffe hanno la forma di un *preambolo* opzionale, seguito da una serie di stringhe, separate da virgola, tra una coppia di parentesi graffe, seguite da un *postambolo* opzionale. Il preambolo è preposto a ogni stringa contenuta dentro le parentesi graffe, e il postambolo è poi appeso ad ogni stringa risultante, espandendo da sinistra a destra.

Le espansioni delle parentesi graffe possono essere nidificate. Il risultato di ogni stringa espansa non viene ordinato; è conservato l'ordine da sinistra a destra. Per esempio, a**{**d,c,b**}**e si espande in 'ade ace abè.

L'espansione delle parentesi graffe è effettuata prima di qualsiasi altra espansione, e qualunque carattere speciale per le altre espansioni viene conservato nel risultato. Essa è strettamente testuale. **Bash** non applica alcuna interpretazione sintattica al contesto della espansione o al testo tra le parentesi graffe.

Una espansione delle parentesi graffe correttamente formata deve contenere parentesi graffe aperte e chiuse, non quotate, e almeno una virgola non quotata. Qualunque espansione delle parentesi graffe erroneamente formata è lasciata inalterata.

Questo costrutto è tipicamente usato come abbreviazione quando il prefisso comune delle stringhe da generare è più lungo che negli esempi sopra:

```
mkdir /usr/local/src/bash/{old,new,dist,bugs}
```

o

```
chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}}
```

L'espansione delle parentesi graffe introduce una leggera incompatibilità con le versioni tradizionali della Bourne shell **sh**. **sh** non tratta le parentesi graffe aperte e chiuse, specialmente quando esse appaiono come parte di una parola, e le conserva in uscita. **Bash** rimuove le parentesi graffe dalle parole come conseguenza della espansione. Per esempio, una parola data a **sh** come *file{1,2}* appare identica nell'output. La stessa parola è data in output come *file1 file2* dopo l'espansione operata da **bash**. Se si desidera, una stretta compatibilità con **sh** si avvia **bash** con il flag **−nobraceexpansion** (si veda **OPZIONI** sopra) o si disabilita l'espansione delle parentesi graffe con l'opzione **+o braceexpand** del comando **set** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti).

Espansione della tilde

Se una parola comincia con un carattere tilde (‘**~**’), tutti i caratteri che precedono il primo slash (o tutti caratteri, se non vi è alcuno slash) sono trattati come un possible *nome di login*. Se questo *nome di login* è la stringa nulla, la tilde è sostituita con il valore del parametro **HOME**. Se **HOME** non è posto, è invece sostituita la home directory dell’utente che sta eseguendo la shell.

Se un ‘+’ segue la tilde, il valore di **PWD** sostituisce la tilde e ‘+’. Se segue un ‘−’, è sostituito il valore di **OLDPWD**. Se il valore che segue la tilde è un *nome di login* valido, la tilde e il *nome di login* sono rimpiazzati con la home directory associata con quel nome. Se il nome non è valido, o la espansione della tilde fallisce, la parola non viene cambiata.

Ogni assegnamento di variabile è controllato per istanze non quotate di tilde seguenti un **:** o **=**. Anche in questi casi viene effettuata la sostituzione della tilde. Di conseguenza, si possono usare dei percorsi con delle tilde negli assegnamenti a **PATH**, **MAILPATH**, e **CDPATH**, e la shell assegna il valore espanso.

Espansione di parametro

Il carattere ‘**$**’ introduce l’espansione di parametro, la sostituzione di comando, o l’espansione aritmetica. Il nome o simbolo del parametro che deve essere espanso può essere racchiuso tra parentesi graffe, che sono opzionali ma servono a proteggere la variabile che deve essere espansa, dai caratteri immediatamente seguenti, che potrebbero essere interpretati come parte del nome.

${*parametro*}

È sostituito con il valore di *parametro*. Le parentesi graffe sono richieste quando *parametro* è un parametro posizionale con più di una cifra, o quando *parametro* è seguito da un carattere che non deve essere interpretato come parte del suo nome.

In ognuno dei casi più avanti, *parola* è soggetta alla espansione della tilde, espansione di parametro, sostituzione di comando ed espansione aritmetica. **Bash** controlla se un parametro non è posto o è nullo; l’omissione dei "due punti" provoca il solo controllo di parametro non posto.

${*parametro***:−***parola*}

Usa i valori di default. Se *parametro* non è posto o è nullo, è sostituito con l’espansione di *parola*. Altrimenti, è sostituito con il valore di *parametro*.

${*parametro***:=***parola*}

Assegna i valori di default. Se *parametro* non è posto o è nullo, l’espansione di *parola* è assegnata a *parametro*. Il valore di *parametro* è quindi sostituito. I parametri posizionali e parametri speciali non possono essere assegnati in questo modo.

${*parametro***:?***parola*}

Mostra un errore se nullo o non posto. Se *parametro* è nullo o non posto, l’espansione di *parola* (o un messaggio dello stesso effetto, se *parola* non è presente) viene scritta sullo standard error e la shell, se non è interattiva, termina. Altrimenti, è sostituito il valore di *parametro*.

${*parametro***:+***parola*}

Usa un valore alternativo. Se *parametro* è nullo o non posto, non è sostituito niente, altrimenti è sostituita l’espansione di *parola*.

${**#***parametro*}

È sostituita la lunghezza in caratteri del valore di *parametro*. Se *parametro* è ***** o **@**, la lunghezza sostituita è la lunghezza di ***** espanso tra virgolette.

${*parametro***#***parola*}
${*parametro***##***parola*}

La *parola* è espansa per produrre un pattern proprio come nella espansione di percorso. Se il pattern combacia con l’inizio del valore di *parametro*, allora l’espansione è il valore di *parametro* con il più corto dei pattern combacianti cancellato (nel caso di ‘‘**#**’’) o il più lungo dei pattern combacianti cancellato (nel caso di ‘‘**##**’’).

${*parametro***%***parola*}
${*parametro***%%***parola*}

La *parola* è espansa per produrre un pattern proprio come nella espansione di percorso. Se il pattern combacia con una parte finale del valore di *parametro*, allora l’espansione è il valore di *parametro* con il più corto dei pattern combacianti cancellato (nel caso di ‘‘**%**’’) o il più lungo dei

pattern combacianti cancellato (nel caso di ‘‘**%%**’’).

Sostituzione di comando

La *sostituzione comando* permette che l’output di un comando rimpiazzi il nome del comando. Vi sono due forme:

$(*comando* **)**

o

‘*comando***‘**

Bash effettua l’espansione eseguendo il *comando* e rimpiazzando la sostituzione di comando con lo standard output del comando, con ogni newline finale cancellato.

Quando è usata la forma di sostituzione in vecchio stile, con le backquote, il backslash conserva il suo significato letterale tranne quando seguito da **$**, **‘**, o \. Quando si usa la forma $(*comando*), tutti i caratteri tra le parentesi formano il comando; nessuno è trattato in modo speciale.

La sostituzione di comando può essere nidificata. Per nidificare quando si usa la vecchia forma, bisogna far precedere le backquote più interne con un carattere backslash di escape.

Se la sostituzione appare tra virgolette, la suddivisione in parole e l’espansione di percorso non sono effettuate sui risultati.

Espansione aritmetica

L’espansione artimetica permette la valutazione di una espressione aritmetica e la sostituzione del risultato. Vi sono due formati per l’ espansione aritmetica:

$[*espressione***]**

$((*espressione***))**

L’ *espressione* è trattata come se fosse tra virgolette, ma le virgolette dentro le parentesi graffe o tonde non sono trattate in modo speciale. Tutti i token nella espressione subiscono l’espansione di parametro, la sostituzione comando e la rimozione dei caratteri di quotatura. Le sostituzioni aritmetiche possono essere nidificate.

Il calcolo è effettuato in accordo con le regole elencate più avanti sotto **CALCOLO ARITMETICO**. Se *espressione* non è valida, **bash** stampa un messaggio che indica il fallimento e non avviene alcuna sostituzione.

Sostituzione di processo

La *sostituzione di processo* è supportata sui sistemi che supportano le pipe con nome (*FIFO*) o il metodo **/dev/fd** per nominare i file aperti. Essa prende la forma di **<(***lista***)** o **>(***lista***)**. La *lista* di processi è eseguita con il suo input o output connesso ad un *FIFO* o un file in **/dev/fd**. Il nome di questo file è passato come un argomento al comando corrente come risultato della espansione. Se è usata la forma **>(***lista***)**, la scrittura sul file fornirà input per la *lista*. Se è usata la forma **<(***lista***)**, il file passato come argomento dovrà essere letto per ottenere l’output di *lista*.

Su sistemi che la supportano, la *sostituzione di processo* è effettuata simultaneamente alla *espansione di parametro e variabile* , *sostituzione di comando*, ed *espansione aritmetica*.

Suddivisione in parole

La shell scandisce il risultato dell’espansione di parametro, sostituzione di comando ed espansione aritmetica, che non si trovano tra virgolette, per effettuare la *suddivisione in parole*.

La shell tratta ogni carattere di **IFS** come un delimitatore, e spezza in parole, in coincidenza di questi caratteri, i risultati delle altre espansioni. Se il valore di **IFS** è esattamente **<space><tab><newline>**, il default, allora qualsiasi sequenza di caratteri **IFS** serve per delimitare le parole. Se **IFS** ha un valure diverso dal default, allora sequenze di caratteri di spaziatura **space** e **tab** sono ignorati all’inizio e fine di parola, fin tanto che il carattere di spaziatura è nel valore di **IFS** (un carattere di spaziatura **IFS**). Qualunque carattere in **IFS** che non è un carattere di spaziatura **IFS**, insieme con qualsiasi carattere di spaziatura **IFS** adiacente, delimita un campo. Una sequenza di caratteri di spaziatura **IFS** è anche trattata come un delimitatore. Se il valore di **IFS** è nullo, non avviene alcuna suddivisione in parole. **IFS** non può essere non posto.

Argomenti esplicitamente nulli (**""** o **''**) sono conservati. Argomenti implicitamente nulli, risultanti dalla espansione di *parametri* che non hanno alcun valore, sono rimossi.

È da notare che se non avviene alcuna espansione, nessuna suddivisione è effettuata.

Espansione di percorso

Dopo la suddivisione in parole, a meno che sia stata posta l'opzione **−f**, **bash** scandisce ogni *parola* alla ricerca dei caratteri *****, **?**, e **[**. Se uno di questi caratteri compare, allora la parola è considerata come un *pattern*, e sostituita con una lista, ordinata alfabeticamente, di percorsi che combaciano con il pattern. Se nessun percorso che combacia viene trovato, e la variabile di shell **allow_null_glob_expansion** non è posta, la parola è lasciata inalterata. Se la variabile è posta, e nessuna coincidenza è trovata, la parola è rimossa. Quando un pattern è usato per la generazione di percorsi, il carattere **"."** all'inizio di un nome o immediatamente seguente uno slash deve combaciare esplicitamente, a meno che la variabile di shell **glob_dot_filenames** sia posta. Il carattere slash deve sempre combaciare esplicitamente. Negli altri casi, il carattere **"."** non è trattato in modo speciale.

I caratteri speciali dei pattern hanno i seguenti significati:

***** Combacia con qualsiasi stringa, inclusa la stringa nulla.

? Combacia con qualsiasi carattere singolo.

[...] Combacia con uno qualsiasi dei caratteri racchiusi. Una coppia di caratteri separati da un segno meno denota un *range* (intervallo); è combaciante qualunque carattere lessicalmente tra questi due caratteri, inclusi. Se il primo carattere che segue il **[** è un **!** o un **^** allora qualunque carattere non racchiuso è combaciante. Un **−** o **]** può essere confrontato includendolo come primo o ultimo carattere dell'insieme.

Rimozione dei caratteri di quotatura

Dopo le precedenti espansioni, tutte le occorrenze non quotate dei caratteri \, **'**, e **"** sono rimosse.

RIDIREZIONE

Prima che un comando sia eseguito, i suoi input e output possono essere *ridiretti* usando una speciale notazione interpretata dalla shell. La ridirezione può anche essere usata per aprire e chiudere file per l'ambiente di esecuzione della shell corrente. I seguenti operatori di ridirezione possono precedere o apparire ovunque dentro un *comando semplice* o possono seguire un *comando*. Le ridirezioni sono processate nell'ordine in cui compaiono, da sinistra a destra.

Nelle seguenti descrizioni, se il numero di descrittore di file è omesso, e il primo carattere dell'operatore di ridirezione è <, la ridirezione si riferisce allo standard input (descrittore di file 0). Se il primo carattere dell'operatore di ridirezione è >, la ridirezione si riferisce allo standard output (descrittore di file 1).

La parola che segue l'operatore di ridirezione nelle seguenti descrizioni è sottoposta alla espansione delle parentesi graffe, espansione della tilde, espansione di parametro, sostituzione di comando, espansione aritmetica, rimozione dei caratteri di quotatura ed espansione dei percorsi. Se si espande a più di una parola, **bash** riporta un errore.

È da notare che l'ordine delle ridirezioni è significativo. Per esempio, il comando

ls > dirlist 2>**&**1

dirige sia lo standard output che lo standard error sul file *dirlist*, mentre il comando

ls 2>**&**1 > dirlist

dirige solo lo standard output sul file *dirlist*, poiché lo standard error è stato duplicato come standard output prima che lo standard output fosse ridiretto su *dirlist*.

Ridirezione dell'input

La ridirezione dell'input fa sì che il file il cui nome risulta dalla espansione di *parola* venga aperto in lettura sul descrittore di file *n*, o lo standard input (descrittore di file 0) se *n* non è specificato.

Il formato generico per ridirigere l'input è:

[*n*]<*parola*

Ridirezione dell'output

La ridirezione dell'output fa sì che il file il cui nome risulta dalla espansione di *parola* venga aperto in scrittura sul descrittore di file *n*, o lo standard output (descrittore di file 1) se *n* non è specificato. Se il file non esiste è creato; se esiste è troncato alla grandezza zero.

Il formato generico per ridirigere l'output è:

[*n*]>*parola*

Se l'operatore di ridirezione è >|, allora il valore dell'opzione **-C** del comando incorporato **set** non è controllata, ed è tentata la creazione del file (vedi anche la descrizione di **noclobber** sotto **Variabili di shell** sopra).

Aggiungere in coda l'output ridiretto

La ridirezione dell'output in questo modo fa sì che il file il cui nome risulta dalla espansione di *parola* venga aperto per aggiungere in coda (append) sul descrittore di file *n*, o lo standard output (descrittore di file 1) se *n* non è specificato. Se il file non esiste viene creato.

Il formato generico per aggiungere in coda l'output è:

[*n*]>>*parola*

Ridirezione di standard output e standard error

Bash permette, con questo costrutto, che sia lo standard output (descrittore di file 1) che lo standard error (descrittore di file 2) siano ridiretti sul file il cui nome risulta dall'espansione di *parola*.

Vi sono due formati per ridirigere lo standard output e lo standard error:

&>*parola*

e

>**&***parola*

Delle due forme, la prima è preferita. Questo è semanticamente equivalente a

>*parola* 2>**&**1

Testo in linea

Questo tipo di ridirezione istruisce la shell a leggere l'input dal file sorgente corrente, finché non venga vista una linea contenente solo *parola* (con nessun carattere bianco di seguito). Tutte le linee lette fino a quel punto sono quindi usate come standard input per un comando.

Il formato del testo in linea è come segue:

<<[−]*parola*
testo-in-linea
delimitatore

Nessuna espansione di parametro, sostituzione di comando, espansione di percorso o espansione aritmetica è effettuata su *parola*. Se qualche carattere in *parola* è quotato, il *delimitatore* è il risultato della rimozione dei caratteri di quotatura su *parola*, e le linee nel testo-in-linea non sono espanse. Altrimenti, tutte le linee del testo-in-linea sono soggette a espansione di parametro, sostituzione di comando ed espansione aritmetica. Nell'ultimo caso, la coppia \<**newline**> è ignorata, e \ deve essere usata per quotare i caratteri \, **$**, e **‘**.

Se l'operatore di ridirezione è <<−, allora tutti i caratteri tab iniziali sono eliminati dalle linee in input e dalla linea che contiene *delimitatore*. Questo permette che un testo-in-linea dentro uno script di shell possa essere indentato in un modo naturale.

Duplicazione dei descrittori di file

L'operatore di ridirezione

[*n*]<**&***parola*

è usata per duplicare descrittori di file di input. Se *parola* si espande in una o più cifre, il descrittore di file indicato da *n* è fatto diventare una copia di quel descrittore di file. Se *parola* vale −, il descrittore di file *n* viene chiuso. Se *n* non è specificato, è usato lo standard input (descrittore di file 0).

L'operatore

[*n*]>**&***parola*

è usato similarmente per duplicare i descrittori di file di output. Se *n* non è specificato, è usato lo standard output (descrittore di file 1). Come caso speciale, se *n* è omesso, e *parola* non si espande a una o più cifre, lo standard output e lo standard error sono ridiretti come descritto in precedenza.

Apertura di descrittori di file per lettura e scrittura

L'operatore di ridirezione

[*n*]<>*parola*

fa sì che il file il cui nome è l'espansione di *parola* venga aperto sia per lettura che scrittura sul descrittore di file *n*, o come standard input e standard output se *n* non è specificato. Se il file non esiste, è creato.

FUNZIONI

Una funzione di shell, definita come descritto prima sotto **GRAMMATICA DELLA SHELL**, immagazzina una serie di comandi per una futura esecuzione. Le funzioni sono eseguite nel contesto della shell corrente; nessun nuovo processo è creato per interpretarle (in contrasto, questo, con l'esecuzione di uno script di shell). Quando una funzione è eseguita, gli argomenti della funzione diventano i parametri posizionali durante la sua esecuzione. Il parametro speciale # viene aggiornato per riflettere il cambiamento. Il parametro posizionale 0 rimane inalterato.

Variabili locali alla funzione possono essere dichiarate con il comando incorporato **local**. Usualmente, le variabili e i loro valori sono condivisi tra la funzione e il suo chiamante.

Se il comando incorporato **return** è eseguito in una funzione, la funzione termina e l'esecuzione riprende con il successivo comando dopo la chiamata di funzione. Quando una funzione termina, i valori dei parametri posizionali e il parametro speciale # sono ripristinati ai valori che essi avevano prima della esecuzione della funzione.

I nomi delle funzioni possono essere elencati con l'opzione **−f** dei comandi incorporati **declare** o **typeset**. Le funzioni possono essere esportate, in modo che le subshells automaticamente le abbiano definite, con l'opzione **−f** del comado incorporato **export**.

Le funzioni possono essere ricorsive. Nessun limite è posto sul numero di chiamate ricorsive.

ALIAS

La shell mantiene una lista di *alias* che possono essere posti e rimossi con i comandi incorporati **alias** e **unalias** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti). La prima parola di ogni comando, se non quotata, viene controllata per vedere se essa ha un alias. Se così, quella parola è sostituita dal testo dell'alias. Il nome dell'alias e il testo da sostituire possono contenere qualunque dato valido per la shell, inclusi i *metacaratteri* elencati sopra, con l'eccezione che il nome dell'alias non può contenere =. La prima parola del testo da sostituire è controllata per vedere se contiene alias, ma una parola che è identica ad un alias che si sta espandendo non viene espansa una seconda volta. Questo significa che si può soprannominare **ls** con **ls −F**, per esempio, e **bash** non prova ad espandere ricorsivamente il testo da sostituire. Se l'ultimo carattere del valore di un alias è un *blank*, allora la successiva parola di comando che segue l'alias è anche controllata per l'espansione di alias.

Gli alias sono creati ed elencati con il comando **alias** e rimossi con il comando **unalias**.

Non vi è alcun meccanismo per usare argomenti nel testo da sostituire, come in **csh**. Se sono necessari degli argomenti, si dovrà usare una funzione di shell.

Gli alias non sono espansi quando la shell non è interattiva.

Le regole che riguardano la definizione e l'uso degli alias lasciano abbastanza perplessi. La **bash** legge sempre almeno una linea completa di input prima di eseguire uno dei comandi su quella linea. Gli alias sono espansi quando un comando è letto, non quando è eseguito. Perciò, una definizione di alias che appare sulla stessa linea di un altro comando non ha effetto fino a che non è letta la successiva linea di input. Questo questo significa che i comandi che seguono la definizione di un alias sulla stessa linea non sono influenzati dal nuovo alias. Questo comportamento è un problema anche quando sono eseguite

funzioni. Gli alias sono espansi quando viene letta la definizione di funzione, non quando la funzione è eseguita, poiché una definizione di funzione è essa stessa un comando composto. Come conseguenza, gli alias definiti in una funzione sono disponibili solo dopo che la funzione è eseguita. Per essere sicuri, conviene sempre porre le definizioni di alias su una linea separata, e non usare **alias** in comandi composti.

È da notare che per quasi ogni scopo, gli alias sono soppiantati dalle funzioni di shell.

JOB CONTROL

indica la capacità di fermare (*sospendere*) selettivamente l'esecuzione di processi e continuare (*riprendere*) la loro esecuzione più tardi. Tipicamente, un utente impiega questo servizio attraverso una interfaccia interattiva fornita unitamente dal driver del terminale del sistema e dalla **bash**.

La shell associa un *job* a ogni pipeline. Essa mantiene una tabella dei job correntemente in esecuzione, che può essere mostrata con il comando **jobs**. Quando **bash** avvia un job in modo asincrono (in *background*), essa stampa una linea di questo tipo:

[1] 25647

che indica che questo job è il job numero 1 e che l'ID di processo dell'ultimo processo nella pipeline associata a questo job è 25647. Tutti i processi in una singola pipeline sono membri dello stesso job. **Bash** usa l'astrazione *job* come la base per il job control.

Per facilitare l'implementazione dell'interfaccia utente per il job control, il sistema mantiene la nozione di *ID del gruppo di processi del terminale corrente*. I membri di questo gruppo di processi (processi il cui ID di gruppo di processi è uguale all'ID di gruppo di processi del terminale corrente) ricevono i segnali generati da tastiera, come **SIGINT**. Si dice che questi processi sono in *foreground* (primo piano). I processi in *background* (sullo sfondo) sono quelli il cui ID di gruppo di processi differisce da quello del terminale; tali processi sono immuni ai segnali generati da tastiera. Solo ai processi in foreground è permesso di leggere o scrivere sul terminale. Ai processi in background che tentano di leggere (scrivere) sul terminale è inviato un segnale **SIGTTIN (SIGTTOU)** dal driver del terminale, che, se non intercettato, sospende il processo.

Bash permette di usare il job control, se questo è supportato dal sistema operativo su cui **bash** sta girando. Digitare il carattere di *sospensione* (tipicamente **^Z**, Control-Z) mentre un processo sta girando, provoca lo stop di quel processo e il ritorno del controllo di **bash** all'utente. Digitare il carattere di *sospensione ritardata* (tipicamente **^Y**, Control-Y) provoca lo stop del processo quando questo tenta di leggere input dal terminale, e il ritorno del controllo di **bash**. Si può poi maneggiare lo stato di questo job, usando il comando **bg** per continuarlo in background, il comando **fg** per continuarlo in foreground, o il comando **kill** per ucciderlo. Un **^Z** ha effetto immediatamente, ed ha l'effetto collaterale di causare la perdita dell'output in sospeso.

Vi sono numerosi modi per riferirsi a un job nella shell. Il carattere **%** introduce un nome di job. Un job con numero *n* può essere indicato come **%n**. Un job può anche essere indicato usando un prefisso del nome usato per avviarlo, o usando una sottostringa che appare nella sua linea di comando. Per esempio, **%ce** si riferisce a un job **ce** fermato. Se un prefisso combacia con più di un job, **bash** riporta un errore. Usare **%?ce**, d'altra parte, indica qualsiasi job che contiene la stringa **ce** nella sua linea di comando. Se la sottostringa combacia con più di un job, **bash** rirorta un errore. I simboli **%%** e **%+** si riferiscono alla nozione della shell del *job corrente*, che è l'ultimo job fermato mentre era in foreground. Il *job precedente* può essere indicato usando **%−**. Nell'output che riguarda i job (per esempio, l'output del comando **jobs**), il job corrente è sempre segnalato con un **+**, ed il job precedente con un **−**.

Il semplice nominare un job può essere usato per riportarlo in foreground: **%1** è un sinonimo per **"fg %1"**, che porta il job 1 da background in foreground. Nello stesso modo, **"%1 &"** riprende il job 1 in background, equivalente a **"bg %1"**.

La shell apprende immediatamente ogni volta che un job cambia stato. Normalmente, **bash** aspetta finché non sta per stampare un prompt prima di informare dei cambiamenti nello stato di un job in modo tale da non interrompere qualunque altro output. Se l'opzione **-b** del comando incorporato **set** è posta, **bash** riporta tali cambiamenti immediatamente (si veda anche la descrizione della variabile **notify** sotto **Variabili di shell** sopra).

Se si tenta di uscire da **bash** mentre vi sono dei job fermi, la shell stampa un messaggio di avvertimento. Si può quindi usare il comando **jobs** per ispezionare il loro stato. Se si fa così, o si prova ad uscire di nuovo immediatamente, non si è più avvisati e i job fermi vengono terminati.

SEGNALI

Quando **bash** è interattiva, ignora **SIGTERM** (così che **kill 0** non uccide una shell interattiva), e **SIGINT** viene intercettato e gestito (così che il comando incorporato **wait** è interrompibile). In tutti i casi, **bash** ignora **SIGQUIT**. Se è in uso il job control, **bash** ignora **SIGTTIN**, **SIGTTOU**, e **SIGTSTP**.

I job sincroni avviati da **bash** hanno i segnali impostati sui valori che la shell ha ereditato dal suo genitore. Quando il job control non è in uso, i job in background (i job avviati con **&**) ignorano **SIGINT** e **SIGQUIT**. I comandi eseguiti come risultato di una sostituzione di comando ignorano i segnali di controllo dei job generati da tastiera **SIGTTIN**, **SIGTTOU**, e **SIGTSTP**.

ESECUZIONE DI UN COMANDO

Dopo che un comando è stato suddiviso in parole, se esso ha la forma di un comando semplice e con una lista opzionale di argomenti, sono eseguite le seguenti azioni.

Se il nome del comando non contiene alcuno slash (barra), la shell tenta di localizzarlo. Se esiste una funzione di shell con quel nome, viene chiamata quella funzione, come descritto prima in **FUNZIONI**. Se il nome non combacia con una funzione, la shell lo cerca nella lista dei comandi incorporati della shell. Se ne viene trovato uno combaciante, quel comando incorporato viene eseguito.

Se il nome non è ne una funzione di shell ne un comando incorporato, e non contiene alcuno slash, **bash** cerca tra gli elementi della variabile **PATH** una directory che contenga un file eseguibile con quel nome. Se la ricerca non ha successo, la shell stampa un messaggio di errore e ritorna uno stato di uscita diverso da zero.

Se la ricerca ha successo, o se il nome del comando contiene uno o più slash, la shell esegue il programma indicato. L'argomento 0 è posto al nome dato, e i rimanenti argomenti del comando sono posti agli argomenti dati, se ve ne sono.

Se questa operazione fallisce perché il file non è in formato eseguibile e il file non è una directory, esso è assunto essere uno *script di shell*, cioè, un file che contiene comandi di shell. Una subshell viene attivata per eseguirlo. Questa subshell reinizializza se stessa, così che l'effetto è lo stesso come se fosse stata lanciata una nuova shell per gestire lo script, con la differenza che le locazioni dei comandi ricordati dal genitore (si veda **hash** più avanti sotto **COMANDI INCORPORATI DELLA SHELL**) sono conservate dal figlio.

Se il programma è un file che inizia con **#!**, il resto della prima linea specifica un interprete per il programma. La shell esegue l'interprete specificato sui sistemi operativi che non gestiscono questo formato eseguibile da loro stessi. Gli argomenti per l'interprete consistono di un singolo argomento opzionale che segue il nome dell'interprete sulla prima linea del programma, seguito dal nome del programma, seguito dagli argomenti del comando, se ve ne sono.

AMBIENTE

Quando viene lanciato un programma gli viene dato un vettore di stringhe chiamato *ambiente*. Questa è una lista di coppie *nome−valore*, della forma *nome=valore*.

La shell permette di manipolare l'ambiente in molti modi. Alla chiamata, la shell esamina il suo ambiente e crea un parametro per ogni nome trovato, marcandolo automaticamente per essere *esportato* ai processi figli. I comandi eseguiti ereditano l'ambiente. I comandi **export** e **declare −x** permettono di aggiungere o togliere dall'ambiente parametri e funzioni. Se il valore di un parametro nell'ambiente viene modificato, il nuovo valore diventa parte dell'ambiente, sostituendo il valore vecchio. L'ambiente ereditato da qualsiasi comando eseguito consiste dell'ambiente iniziale della shell, i cui valori possono essere modificati nella shell, meno ogni coppia rimossa dal comando **unset**, più ognuna aggiunta attraverso i comandi **export** e **declare −x .**

L'ambiente per qualsiasi *comando semplice* o funzione può essere aumentato temporaneamente prefissandolo con degli assegnamenti di parametro, come descritto prima in **PARAMETRI**. Queste istruzioni di assegnamento influenzano solo l'ambiente visto da quel comando.

Se è posto il flag **−k** (si veda il comando incorporato **set** più avanti), allora *tutti* gli assegnamenti di

parametro sono posti nell'ambiente per un comando, non solo quelli che precedono il nome del comando.

Quando **bash** chiama un comando esterno, la variabile **_** viene posta al nome del comando, completo di percorso, e passato nell'ambiente di quel comando.

STATO DI USCITA

Per gli scopi della shell, un comando che esce con uno stato di uscita zero ha avuto successo. Uno stato di uscita pari a zero indica successo. Uno stato di uscita diverso da zero indica fallimento. Quando un comando termina su un segnale fatale, **bash** usa il valore di 128+**segnale** come stato di uscita.

Se un comando non è trovato, il processo figlio creato per eseguirlo ritorna uno stato pari a 127. Se un comando è trovato ma non è eseguibile lo stato di ritorno è 126.

Bash stessa ritorna lo stato di uscita dell'ultimo comando eseguito, a meno che non avvenga un errore di sintasi, nel qual caso essa esce con un valore diverso da zero. Si veda anche il comando incorporato **exit** più avanti.

MESSAGGI DI PROMPT

Quando eseguita interattivamente, **bash** mostra il prompt primario **PS1** quando è pronta per leggere un comando, e il prompt secondario **PS2** quando necessita di altro input per completare un comando. **Bash** permette di personalizzare queste stringhe di prompt inserendo un certo numero di caratteri speciali preceduti da backslash che sono decodificati come segue:

\t	l'ora corrente nel formato HH:MM:SS
\d	la data nel formato "Nome-giorno Mese Numero-giorno" (per esempio, "Tue May 26")
\n	nuova linea
\s	il nome della shell, il "basename" di **$0** (la parte che segue l'ultimo slash)
\w	la directory di lavoro corrente
\W	il basename della directory di lavoro corrente
\u	lo username dell'utente corrente
\h	il nome dell host
\#	il numero di comando del comando
\!	il numero di storia del comando
\$	se l'UID effettivo è 0, un **#**, altrimenti un **$**
\nnn	il carattere corrispondente al numero ottale **nnn**
\\	un backslash
\[	inizia una sequenza di caratteri non stampabili, che può essere usata per inglobare una sequenza di controllo di terminale nel prompt
\]	termina una sequenza di caratteri non stampabili

Il numero del comando e numero della storia sono generalmente differenti: il numero della storia di un comando è la sua posizione nella lista della storia, che può includere comandi recuperati dal file di storia (si veda **STORIA** più avanti), mentre il numero del comando è la posizione nella sequenza di comandi eseguiti durante la corrente sessione di shell. Dopo che la stringa è decodificata, essa è espansa attraverso l'espansione di parametro, la sostituzione di comando, l'espansione aritmetica e la suddivisione in parole.

READLINE

Questa è la libreria che gestisce la lettura dell'input quando si usa una shell interattiva, a meno che non sia data l'opzione **−nolineediting**. Per default, i comandi per l'editor di linea sono simili a quelli di emacs. È anche disponibile interfaccia per editor di linea in stile vi.

In questa sezione, è usata la notazione in stile emacs per indicare i tasti digitati. I tasti di controllo sono indicati da C−*tasto*, per esempio, C−n significa Control−N. In modo simile, i *meta* tasti sono indicati da M−*tasto*, cosi M−x significa Meta−X. Sulle tastiere senza un tasto *meta*, M−*x* significa ESC *x*, cioè, si preme il tasto Escape e poi il tasto *x*. Questo rende ESC il *meta prefisso*. La combinazione M−C−*x* significa ESC−Control−*x*, o si preme il tasto Escape poi si tiene il tasto Control mentre si preme il tasto *x*.

Le associazioni di default dei tasti possono essere cambiate con un file *~/.inputrc* Il valore della variabile di shell **INPUTRC**, se posta, è usato invece di *~/.inputrc*. Altri programmi che usano questa libreria possono aggiungere il loro propri comandi e associazioni.

Per esempio, porre

M−Control−u: universal−argument

o

C−Meta−u: universal−argument

nel file *~/.inputrc* farebbe eseguire a M−C−u il comando *universal−argument* della readline.

Sono riconosciuti i seguenti nomi simbolici di carattere: *RUBOUT*, *DEL*, *ESC*, *LFD*, *NEWLINE*, *RET*, *RETURN*, *SPC*, *SPACE*, e *TAB*. In aggiunta al nome del comando, readline permette che i tasti siano combinati in una stringa che è inserita quando il tasto è premuto (una *macro*).

Readline viene personalizzata ponendo i comandi in un file di inizializzazione. Il nome di questo file è preso dal valore della variabile **INPUTRC**. Se questa variabile non è posta, il default è *~/.inputrc*. Quando si avvia un programma che usa la libreria readline, viene letto il file di inizializzazione, e sono impostate le associazioni dei tasti e le variabili. Vi sono solo pochi costrutti base permessi nel file di inizializzazione di readline. Le linee bianche sono ignorate. Le linee che iniziano con un **#** sono commenti. Le linee che iniziano con un **$** indicano costrutti condizionali. Le altre linee denotano le associazioni dei tasti e l'assegnamento di variabili.

La sintassi per il controllo delle associazioni dei tasti nel file *~/.inputrc* è semplice. Tutto quello che è richiesto è il nome del comando o il testo di una macro e una sequenza di tasti alla quale dovrà essere collegata. Il nome può essere specificato in uno dei modi: come nome simbolico di un tasto, eventualmente con i prefissi *Meta-* o *Control-*, o come una sequenza di tasti. Quando si usa la forma **nome-tasto**:*nome-funzione* o *macro*, *nome-tasto* è il nome di un tasto nella lingua inglese. Per esempio:

Control-u: universal−argument
Meta-Rubout: backward-kill-word
Control-o: ">&output"

Negli esempi sopra, *C-u* viene collegato alla funzione **universal−argument**, *M-DEL* viene collegato alla funzione **backward−kill−word**, e *C-o* viene collegato all'esecuzione della macro indicata sul lato destro (cioè, inserire il testo *>&output* nella linea).

Nella seconda forma, **"sequenza-tasti"**:*nome-funzione* o *macro*, **sequenza-tasti** differisce da **nome-tasto** sopra, per il fatto che la stringa che denota una intera sequenza di tasti può essere specificata ponendo la sequenza all'interno delle virgolette. Alcuni tasti di escape nello stile di GNU Emacs possono essere usati, come nei seguenti esempi.

"\C-u": universal−argument
"\C-x\C-r": re−read−init−file
"\e[11~": "Function Key 1"

In questo esempio, *C-u* viene ancora collegato alla funzione **universal−argument**. *C-x C-r* viene collegato alla funzione **re−read−init−file**, e *ESC [1 1 ~* viene collegato all'inserimento del testo **Function Key 1**. L'insieme completo delle sequenze di escape è

\C− prefisso control

\M- prefisso meta

\e un carattere di escape

\\ backslash

\" il carattere "

\' il carattere '

Quando si inserisce il testo di una macro, apostrofi o virgolette dovrebbero essere usati per indicare una definizione di macro. Il testo non quotato si assume che sia un nome di funzione. Il backslash quota qualsiasi carattere nel testo della macro, inclusi " e '.

Bash permette di mostrare o modificare le associazioni correnti dei tasti di readline con il comando

incorporato **bind**. Il modo di editing può essere cambiato durante l'uso interattivo usando l'opzione **−o** del comando incorporato **set** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti).

Readline ha delle variabili che possono essere usate per personalizzare ulteriormente il suo comportamento. Una variabile può essere impostata nel file *inputrc* con un'istruzione della forma

set *nome−di−variabile valore*

Tranne dove indicato diversamente, le variabili di readline possono avere solo i valori **On** o **Off**. Le variabili e i loro valori di default sono:

horizontal−scroll−mode (Off)
Quando posto a **On**, indica a readline di usare una sola linea per la visualizzazione, effettuando lo scroll dell'input in orizzontale su una sola linea dello schermo quando essa diventa più lunga della larghezza dello schermo, piuttosto che tornare a capo su una nuova linea.

editing−mode (emacs)
Controlla se readline parte con un insieme di associazioni di tasti simile a *emacs* o *vi* . **editing−mode** può essere posto a **emacs** o **vi**.

mark−modified−lines (Off)
Se posto a **On**, le linee della storia che sono state modificate sono mostrate precedute da un asterisco (*****).

bell−style (audible)
Controlla cosa accade quando readline vuole suonare il campanello del terminale. Se posto a **none**, readline non suona mai il campanello. Se posto a **visible**, readline usa un campanello visible se ve ne è uno disponibile. Se posto a **audible**, readline tenta di suonare il campanello del terminale.

comment−begin ("#")
La stringa che è inserita in modo **vi** quando è eseguito il comando **vi−comment**.

meta−flag (Off)
Se posto a **On**, readline abilita l'input a otto-bit (cioè, non eliminerà il bit alto dai caratteri che legge), indipendentemente da ciò che il terminale dichiara di poter supportare.

convert−meta (On)
Se posto a **On**, readline converte i caratteri con l'ottavo bit a 1, a una sequenza di caratteri ASCII eliminando l'ottavo bit e preponendo un carattere di escape (in effetti, usando l'escape come *prefisso meta*).

output−meta (Off)
Se posto a **On**, readline mostra direttamente i caratteri con l'ottavo bit a 1, piuttosto che come sequence di escape con prefisso meta.

completion−query−items (100)
Questo determina quando l'utente è interpellato per la visione del totale dei possibili completamenti generati dal comando **possible−completions**. Esso può essere posto a qualsiasi valore intero maggiore o uguale a zero. Se il numero di possibili completamenti è maggiore o uguale al valore di questa variabile, all'utente viene chiesto se desidera o no vederli; altrimenti essi sono semplicemente mostrati sul terminale.

keymap (emacs)
Imposta la mappa corrente dei tasti di readline. Il set dei nomi legali per le mappe dei tasti è *emacs, emacs-standard, emacs-meta, emacs-ctlx, vi, vi-move, vi-command,* e *vi-insert*. *vi* è equivalente a *vi-command*; *emacs* è equivalente a *emacs-standard*. Il valore di default è *emacs*; il valore di **editing−mode** influenza anche la mappa dei tasti di default.

show−all−if−ambiguous (Off)
Questo altera il comportamento di default delle funzioni di completamento. Se posto a **on**, le parole che hanno più di un possibile completamento provocano l'elencazione immediata delle coincidenze invece del suono del campanello.

expand−tilde (Off)
Se posto a **on**, l'espansione della tilde è effettuata quando readline tenta il completamento della parola.

Readline implementa un servizio simile, nello spirito, alla caretteristica di compilazione condizionale del preprocessore C che permette di effettuare associazioni di tasti e impostazioni di variabili in base al risultato di test. Vi sono tre direttive di controllo usate.

$if Il costrutto **$if** permette che le associazioni siano fatte in base al modo di editing, al terminale che viene usato, o della applicazione che fa uso di readline. Il testo di condizione si estende fino alla fine della linea; nessun carattere è richiesto per separarlo.

mode La forma **mode=** della direttiva **$if** è usata per verificare se readline è in modo emacs o vi. Questo può essere usato in congiunzione con il comando **set keymap**, per esempio, per impostare le associazioni delle mappe dei tasti di *emacs-standard* e *emacs-ctlx* solo se readline è avviata nel modo emacs.

term La forma **term=** può essere usata per includere associazioni di tasti specifiche per un terminale, probabilmente per associare le sequenze di tasti emesse dai tasti funzione dei terminali. La parola a destra dell' = viene confrontata con il nome completo del terminale e la parte del nome del terminale che precede il primo −. Questo permette a *sun* di combaciare sia con *sun* che con *sun−cmd*, per esempio.

application
Il costrutto **application** è usato per includere le impostazioni specifiche della applicazione. Ogni programma che usa la libreria readline imposta il suo nome con *application nome*, e un file di inizializzazione può poi controllare un particolare valore. Questo può essere usato per associare sequenze di tasti a funzioni utili per uno specifico programma. Per esempio, il comando seguente aggiunge una sequenza di tasti che quota la parola corrente o la parola precedente in Bash:
$if Bash
Quota la parola corrente o precedente
"\C-xq": "\eb\"\ef\""
$endif

$endif Questo comando, come si è visto nell'esempio precedente, termina un comando **$if** .

$else I comandi in questa parte della direttiva **$if** sono eseguiti se il test fallisce.

Ai comandi di readline possono essere dati *argomenti* numerici, che normalmente agiscono come contatori di ripetizione. Qualche volta, tuttavia, è il segno dell'argomento che è significativo. Passare un argomento negativo a un comando che agisce nella direzione avanti (per esempio, **kill−line**) fa sì che il comando agisca nella direzione indietro. I comandi con argomenti che hanno un comportamento che differisce da questo saranno indicati.

Quando un comando è descritto come *eliminatore* di testo, il testo cancellato è salvato per eventuali recuperi futuri (*yanking*). Il testo eliminato è salvato in un *kill−ring*. Eliminazioni consecutive provocano l'accumulo del testo in una unità, che può essere recuperata tutta in una volta. I comandi che non eliminano testo separano i pezzi di testo sul kill−ring.

La seguente è una lista dei nomi dei comandi e le sequenze di tasti di default a cui essi sono collegati.

Comandi per spostarsi

beginning−of−line (C−a)
Si sposta all'inizio della linea corrente.

end−of−line (C−e)
Si sposta alla fine della linea.

forward−char (C−f)
Si sposta avanti di un carattere.

backward−char (C−b)
Si sposta indietro di un carattere.

forward−word (M−f)
Si sposta in avanti alla fine della parola successiva. Le parole sono composte di caratteri alfanumerici (lettere e cifre).

backward−word (M−b)
Si sposta indietro all'inizio di questa, o della precedente, parola. Le parole sono composte di caratteri alfanumerici (lettere e cifre).

clear−screen (C−l)
Pulisce lo schermo lasciando la linea corrente in cima allo schermo. Con un argomento, rinfresca la linea corrente senza cancellare lo schermo.

redraw−current−line
Rinfresca la linea corrente. Per default, questo non è associato.

Comandi per manipolare la storia

accept−line (Newline, Return)
Accetta la linea senza curarsi di dove sia il cursore. Se questa linea non è vuota, la aggiunge alla lista della storia in accordo con lo stato della variabile **HISTCONTROL**. Se la linea è una linea della storia modificata, allora ripristina la linea della storia al suo stato originale.

previous−history (C−p)
Prende il precedente comando dalla lista della storia, spostandosi indietro nella lista.

next−history (C−n)
Prende il successivo comando dalla lista della storia, spostandosi avanti nella lista.

beginning−of−history (M−<)
Si sposta alla prima linea nella storia.

end−of−history (M−>)
Si sposta alla fine della storia dell'input, cioè, la linea che deve correntemente essere inserita.

reverse−search−history (C−r)
Cerca all'indietro partendo dalla linea corrente spostandosi attraverso la storia come necessario. Questa è una ricerca incrementale.

forward−search−history (C−s)
Cerca in avanti partendo dalla linea corrente e spostandosi attraverso la storia come necessario. Questa è una ricerca incrementale.

non−incremental−reverse−search−history (M−p)
Cerca all'indietro attraverso la storia partendo dalla linea corrente, una stringa fornita dall'utente, usando una ricerca non incrementale.

non−incremental−forward−search−history (M−n)
Cerca in avanti attraverso la storia una stringa fornita dall'utente, usando una ricerca non incrementale.

history−search−forward
Cerca in avanti attraverso la storia una stringa di caratteri tra l'inizio della linea corrente e il punto corrente. Questa è una ricerca non incrementale. Per default, questo comando non è associato.

history−search−backward
Cerca all'indietro attraverso la storia una stringa di caratteri tra l'inizio della linea corrente e il punto corrente. Questa è una ricerca non incrementale. Per default, questo comando non è associato.

yank−nth−arg (M−C−y)
Inserisce il primo argomento del comando precedente (generalmente la seconda parola sulla linea precedente) nel punto (la posizione corrente del cursore). Con un argomento *n*, inserisce la *n*−sima parola del comando precedente (le parole nel comando precedente iniziano con la parola 0). Un argomento negativo inserisce la *n*−sima parola dalla fine del comando precedente.

yank−last−arg (M−., M−_)
Inserisce l'ultimo argomento del comando precedente (l'ultima parola sulla linea precedente). Con un argomento, si comporta esattamente come @code{yank-nth-arg}.

shell−expand−line (M−C−e)
Espande la linea nello stesso modo in cui fa la shell quando la legge. Questo effettua espansione degli alias e della storia così come tutte le espansioni di parola della shell. Si veda **ESPANSIONE DELLA STORIA** più avanti per una descrizione della espansione della storia.

history−expand−line (M−^)
Effettua l'espansione della storia sulla linea corrente. Si veda **ESPANSIONE DELLA STORIA** più avanti per una descrizione della espansione della storia.

insert−last−argument (M−., M−_)
Un sinonimo per **yank−last−arg**.

operate-and-get-next (C−o)
Accetta la linea corrente per l'esecuzione e prende dalla storia la linea successiva relativa alla linea corrente, per l'editing. Qualsiasi argomento è ignorato.

Comandi per cambiare il testo

delete−char (C−d)
Cancella il carattere sotto il cursore. Se il punto è all'inizio della linea, non vi è alcun carattere nella linea, e l'ultimo carattere battuto non era **C−d**, allora ritorna **EOF**.

backward−delete−char (Rubout)
Cancella il carattere dietro il cursore. Quando è dato un argomento numerico, salva il testo cancellato sul kill−ring.

quoted−insert (C−q, C−v)
Aggiunge "alla lettera" il successivo carattere che si digita sulla linea. Questo è il modo per inserire caratteri come **C−q**, per esempio.

tab−insert (C-v TAB)
Inserisce un carattere tab.

self−insert (a, b, A, 1, !, ...)
Inserisce il carattere digitato.

transpose−chars (C−t)
Trascina il carattere prima del cursore in avanti sopra il carattere sotto il cursore. Anche il cursore si sposta in avanti. Se il cursore è alla fine della linea, allora transpone i due caratteri prima del cursore. Argomenti negativi non funzionano.

transpose−words (M−t)
Trascina la parola prima del cursore dopo la parola oltre il cursore spostando inoltre il cursore dopo quella parola.

upcase−word (M−u)
Rende maiuscola la corrente (o seguente) parola. Con un argomento negativo, opera sulla parola precedente, ma non sposta il cursore.

downcase−word (M−l)
Rende minuscola la corrente (o seguente) parola. Con un argomento negativo, opera sulla parola precedente, ma non sposta il cursore.

capitalize−word (M−c)
Rende maiuscola la prima lettera della corrente (o seguente) parola. Con un argomento negativo, opera sulla parola precedente, ma non sposta il cursore.

Eliminazione e recupero

kill−line (C−k)
Elimina il testo dalla posizione corrente del cursore fino alla fine della linea.

backward−kill−line (C−x C−Rubout)
Elimina all'indietro fino all'inizio della linea.

unix−line−discard (C−u)
Elimina all'indietro dal cursore fino all'inizio della linea.

kill−whole−line
Elimina tutti i caratteri sulla linea corrente, non importa dove sia il cursore. Per default, questo non è associato.

kill−word (M−d)
Elimina dal cursore fino alla fine della parola corrente, o se tra parole, fino alla fine della successiva parola. I confini delle parole sono gli stessi che sono usati da **forward−word**.

backward−kill−word (M−Rubout)
Elimina la parola dietro il cursore. I confini delle parole sono gli stessi che sono usati da **forward−word**.

unix−word−rubout (C−w)
Elimina la parola dietro il cursore, usando gli spazi bianchi come un confine di parola. I confini di parola sono diversi rispetto a backward−kill−word.

delete−horizontal−space
Cancella tutti gli spazi e tab attorno al cursore. Per default, questo non è associato.

yank (C−y)
Recupera il contenuto in cima al kill−ring e lo pone prima del cursore.

yank−pop (M−y)
Ruota il kill−ring, e recupera la nuova cima. Funziona solo dopo **yank** o **yank−pop**.

Argomenti numerici

digit−argument (M−0, M−1, ..., M−−)
Aggiunge questa cifra (digit) all'argomento che sta già accumulando, o inizia un nuovo argomento. M−− avvia un argomento negativo.

universal−argument
Ogni volta che è eseguito questo, l'argument count è moltiplicato per 4. L'argument count è inizialmente uno, così eseguendo questa funzione la prima volta mette argument count a quattro. Per default, questo non è associato a un tasto.

Completamento

complete (TAB)
Tenta di effettuare il completamento del testo che precede il cursore. **Bash** tenta il completamento trattando il testo, rispettivamente, come una variabile (se il testo inizia con **$**), nome di utente (se il testo comincia con **~**), nome di host (se il testo comincia con **@**), o comando (includsi alias e funzioni). Se nessuna di queste produce una coincidenza, viene tentato il completamento di nome di file.

possible−completions (M-?)
Elenca i possibili completamenti del testo che precede il cursore.

insert−completions
Inserisce tutti i completamenti del testo che precede il cursore che sarebbero stati generati da **possible−completions**. Per default, questo non è associato a un tasto.

complete−filename (M−/)
Tenta il completamento del nome del file sul testo che precede il cursore.

possible−filename−completions (C−x /)
Elenca i possibili completamenti del testo che precede il cursore, trattandolo come un nome di file.

complete−username (M−~)
Tenta il completamento del testo che precede il cursore, trattandolo come un nome di utente.

possible−username−completions (C−x ~)
Elenca i possibili completamenti del testo che precede il cursore, trattandolo come un nome di utente.

complete−variable (M−$)
Tenta il completamento del testo che precede il cursore, trattandolo come una variabile di shell.

possible−variable−completions (C−x $)
Elenca i possibili completamenti del testo che precede il cursore, trattandolo come una variabile di shell.

complete−hostname (M−@)
Tenta il completamento del testo che precede il cursore, trattandolo come un nome di host.

possible−hostname−completions (C−x @)
Elenca i possibili completamenti del testo che precede il cursore, trattandolo come un nome di host.

complete−command (M−!)
Tenta il completamento del testo che precede il cursore, trattandolo come nome di comando. Il completamento di comando tenta di far combaciare il testo confrontandolo con alias, parole riservate, funzioni di shell, comandi incorporati e, in fine, nomi di file eseguibili, in questo ordine.

possible−command−completions (C−x !)
Elenca i possibili completamenti del testo che precede il cursore, trattandolo come un nome di comando.

dynamic−complete−history (M-TAB)
Tenta il completamento del testo che precede il cursore, confrontando il testo con le linee della lista della storia cercando i possibili completamenti combacianti.

complete−into−braces (M−{)
Effettua il completamento del nome di file e ritorna la lista dei possibili completamenti racchiusi tra parentesi graffe, così la lista è disponibile per la shell (si veda **Espansione delle parentesi graffe** sopra).

Macro di tastiera

start−kbd−macro (C-x (
Inizia a salvare i caratteri digitati nella corrente macro di tastiera.

end−kbd−macro (C-x))
Smette di salvare i caratteri digitati nella corrente macro di tastiera e salva la definizione.

call−last−kbd−macro (C-x e)
Riesegue l'ultima macro di tastiera definita, facendo si che i caratteri nella macro appaiano come se fossero digitati alla tastiera.

Varie

re−read−init−file (C−x C−r)
Legge il contenuto del file personale di inizializzazione, e incorpora ogni associazione o assegnamento di variabile trovata.

abort (C−g)
Annulla il corrente comando di editing e suona il campanello del terminale (dipendente dall'impostazione di **bell−style**).

do−uppercase−version (M−a, M−b, ...)
Esegue il comando che è collegato al corrispondente carattere maiuscolo.

prefix−meta (ESC)
Mette il tasto Meta per il successivo carattere digitato. **ESC f** è equivalente a **Meta−f**.

undo (C−_, C−x C−u)
Undo incrementale, ricordato separatamente per ogni linea.

revert−line (M−r)
Toglie tutti i cambiamenti fatti su questa linea. Questo è come digitare il comando **undo** un numero sufficiente di volte da riportare la linea al suo stato iniziale.

tilde−expand (M−~)
Effettua l'espansione della tilde sulla parola corrente.

dump−functions
Stampa tutte le funzioni e le loro associazioni di tasti sul flusso di uscita di readline. Se è fornito un argomento numerico, l'uscita è formattata in modo tale che può essere fatta parte di un file *inputrc*.

display−shell−version (C−x C−v)
Mostra l'informazione sulla versione della corrente instanza di **bash**.

emacs−editing−mode (C−e)
Quando in modo di editing **vi**, questo causa la commatazione al modo di editing **emacs**.

STORIA

Quando interattiva, la shell fornisce l'accesso alla *storia dei comandi*, la lista dei comandi precedentemente digitati. Il testo degli ultimi **HISTSIZE** comandi (default 500) è salvato in una lista della storia. La shell immagazzina ogni comando nella lista della storia prima della espansione di parametro e variabile (si veda **ESPANSIONE** sopra) ma dopo che è effettuata l'espansione della storia, ed è dipendente dai valori delle variabili di shell **command_oriented_history** e **HISTCONTROL**. All'avviamento, la storia è inizializzata dal file indicata dalla variabile **HISTFILE** (default *~/.bash_history*). **HISTFILE** viene accorciato, se necessario, per contenere non più di **HISTFILESIZE** linee. Il comando incorporato **fc** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti) può essere usato per elencare o editare e rieseguire una parte

della lista della storia. Il comando incorporato **history** può essere usato per mostrare la lista della storia e manipolare il file di storia. Quando si usa l'editing della linea di comando, sono disponibili comandi di ricerca in ciascun modo di editing che fornisce accesso alla lista della storia. Quando una shell interattiva esce, le ultime **HISTSIZE** linee sono copiate dalla lista della storia su **HISTFILE**. Se **HISTFILE** non è posto, o se il file di storia non è scrivibile, la storia non viene salvata.

ESPANSIONE DELLA STORIA

La shell supporta una caratteristica di espansione della storia che è simile alla espansione della storia in **csh.** Questa sezione descrive quali caratteristiche di sintassi sono disponibili. Questa caratteristica è abilitata per default per le shell interattive, e può essere disabilitata usando l'opzione **+H** del comando incorporato **set** (si veda **COMANDI INCORPORATI DELLA SHELL** più avanti). Le shell non interattive non effettuano l'espansione della storia.

L'espansione della storia è effettuata immediatamente dopo che una linea completa è letta, prima che la shell la spezzi in parole. Essa ha luogo in due parti. La prima è determinare quale linea dalla storia precedente usare durante la sostituzione. La seconda è per selezionare parti di quella linea da includere in quella corrente. La linea selezionata dalla storia precedente è l'*evento*, e la parte di quella linea su cui si agisce sono le *parole*. La linea è spezzata in parole nello stesso modo di quando è letta in input, così che più parole separate da *metacaratteri* circondate da virgolette o apostrofi, sono considerate come una parola. Solo il backslash (\) e gli apostrofi possono quotare il carattere di escape della storia, che è **!** per default.

La shell permette il controllo dei vari caratteri usati dal meccanismo di espansione della storia (si veda la precedente descrizione di **histchars** sotto **Variabili di shell**).

Designatore di evento

Un designatore di evento è un riferimento a un elemento di linea di comando nella lista della storia.

! Inizia una sostituzione di storia, tranne quando seguita da un **blank**, newline, = o (.

!! Si riferisce al comando precedente. Questo è un sinonimo per '!−1'.

!*n* Si riferisce alla linea di comando *n*.

!−*n* Si riferisce alla linea di comando corrente meno *n*.

!*stringa*
Si riferisce al più recente comando che inizia con *stringa*.

!?*stringa***[?]**
Si riferisce al più recente comando che contiene *stringa*.

^*stringa1***^***stringa2***^**
Sostituzione rapida. Ripete l'ultimo comando, rimpiazzando *stringa1* con *stringa2*. Equivalente a "!!:s/*stringa1*/*stringa2*/" (si veda **Modificatori** più avanti).

!# L'intera linea di comando digitata fino a questo punto.

Designatori di parola

Un **:** separa la specificazione di evento dal designatore di parola. Esso può essere omesso se il designatore di parola inizia con un **^**, **$**, *****, o **%**. Le parole sono numerate dall'inizio della linea, con la prima parola indicata da uno 0 (zero).

0 (zero)
La zeresima parola. Per la shell, questa è la parola di comando.

n La *n*−esima parola.

^ Il primo argomento. Cioè, la parola 1.

$ L'ultimo argomento.

% La parola che combacia con '?*stringa*?' nella più recente ricerca.

x−*y* Un intervallo di parole; '−*y*' abbrevia '0−*y*'.

***** Tutte le parole tranne la zeresima. Questo è un sinonimo per '*1−$*'. Non è un errore usare ***** se vi è solo una parola nell'evento; in quel caso è ritornata la stringa vuota.

x* Abbrevia *x*−*$*.

x− Abbrevia *x*−*$* come **x***, ma omette l'ultima parola.

Modificatori

Dopo l'opzionale designatore di parola, si può aggiungere una sequenza di uno o più dei seguenti modificatori, ognuno preceduto da un ':'.

h Rimuove un componente di coda in un percorso, lasciando solo la testa.
r Rimuove un suffiso di coda della forma *.xxx*, lasciando il basename (nome base).
e Rimuove tutto tranne il suffisso di coda.
t Rimuove tutti i componenti di testa in un percorso, lasciando la coda.
p Stampa il nuovo comando ma non lo esegue.
q Quota le parole sostituite, prevenendo ulteriori sostituzioni.
x Quota le parole sostituite, come con **q**, ma spezza in parole sui **blank** e newline.
s/*vecchio*/*nuovo*/
Sostituisce *nuovo* per la prima occorrenza di *vecchio* nella linea di evento. Qualsiasi delimitatore può essere usato in luogo di /. Il delimitatore finale è opzionale se esso è l'ultimo carattere della linea di evento. Il delimitatore può essere quotato in *vecchio* e *nuovo* con un singolo backslash. Se compare & in *nuovo*, esso è rimpiazzato da *vecchio*. Un singolo backslash quoterà il carattere &.
& Ripete la precedente sostituzione.
g Fa sì che i cambiamenti siano applicati su l'intera linea di evento. Questo è usato in congiunzione con '**:s**' (per esempio, '**:gs**/*vecchio*/*nuovo*/') o '**:&**'. Se usato con '**:s**', qualsiasi delimitatore può essere usato in luogo di /, e il delimitatore finale è opzionale se esso è l'ultimo carattere della linea di evento.

CALCOLO ARITMETICO

La shell permette di calcolare espressioni aritmetiche, sotto certe circostanze (si veda il comando incorporato **let** e l'**espansione artimetica**). La valutazione viene fatta in interi long senza controllo per l'overflow, benché la divisione per 0 sia intercettata e segnalata come errore. La seguente lista di operatori è raggruppata per operatori di uguale livello di precedenza. I livelli sono elencati in ordine decrescente di precedenza.

− + meno e più unari
! ~ negazione logica e "bit a bit"
*** / %** moltiplicazione, divisione, modulo
+ − addizione, sottrazione
<< >> shift "bit a bit" a sinistra e a destra
<= >= < >
confronti
== != uguaglianza e disuaglianza
& AND "bit a bit"
^ OR esclusivo "bit a bit"
| OR "bit a bit"
&& AND logico
|| OR logico
= *= /= %= += −= <<= >>= &= ^= |=
assegnamento

Le variabili di shell sono permesse come operandi; l'espansione di parametro è effettuata prima che l'espressione sia valutata. Il valore di un parametro dentro una espressione è forzato a un intero long. Una variabile di shell non ha bisogno di avere il suo attributo intero posto a on per essere usata in una espressione.

Le constanti con uno 0 iniziale sono interpretate come numeri ottali. Uno *0x* o *0X* iniziale indica numeri esadecimali. Altrimenti, i numeri prendono la forma [*base#*]n, dove *base* è un numero decimale tra 2 e 36 che rappresenta la base aritmetica, e *n* è un numero in quella base. Se *base* è omessa, allora è usata la base 10.

Gli operatori sono valutati in ordine di precedenza. Le sottoespressioni tra parentesi sono valutate prima e possono superare le regole di precedenza di cui sopra.

COMANDI INCORPORATI DELLA SHELL

: [*argomenti*]

Nessun effetto; il comando non fa nient'altro oltre l'espansione degli *argomenti* e l'effettuazione di ogni ridirezione specificata. È ritornato un codice di uscita zero.

. *nomefile* [*argomenti*]
source *nomefile* [*argomenti*]

Legge ed esegue comandi dal *nomefile* nell'ambiente di shell corrente e ritorna lo stato di uscita dell'ultimo comando eseguito da *nomefile*. Se *nomefile* non contiene uno slash, i percorsi in **PATH** sono usati per trovare la directory contenente *nomefile*. Il file cercato in **PATH** non ha bisogno di essere eseguibile. La ricerca è fatta nella directory corrente se nessun file viene trovato in **PATH**. Se sono forniti degli *argomenti*, essi diventano i parametri posizionali quando *nomefile* è eseguito. Altrimenti i parametri posizionali sono inalterati. Lo stato di ritorno è lo stato dell'ultimo comando terminato dentro lo script (0 se nessun comando è eseguito), e falso se *namefile* non è trovato.

alias [*nome*[*=valore*] ...]

Alias senza argomenti stampa la lista degli alias nella forma *nome=valore* sullo standard output. Quando sono forniti gli argomenti, è definito un alias per ogni *nome* per cui è dato il *valore*. Uno spazio finale in *valore* provoca il controllo della successiva parola per la sostituzione di alias quando l'alias è espanso. Per ogni *nome* nella lista di argomenti per cui nessun *valore* è fornito, è stampato il nome e valore dell'alias. **Alias** ritorna vero a meno che sia dato un *nome* per il quale nessun alias è stato definito.

bg [*jobspec*]

Pone *jobspec* in background, come se esso fosse stato avviato con **&**. Se *jobspec* non è presente, è usata la nozione della shell di *job corrente*. **bg** *jobspec* ritorna 0 a meno che sia eseguito quando il job control è disabilitato o, con il job control abilitato, se *jobspec* non è stato trovato o è stato avviato senza job control.

bind [**−m** *keymap*] [**−lvd**] [**-q** *nome*]
bind [**−m** *keymap*] **-f** *nomefile*
bind [**−m** *keymap*] *keyseq*:*nome-funzione*

Mostra le correnti associazioni di tasti e funzioni di **readline**, o associa una sequenza di tasti a una funzione o macro di **readline**. La sintassi di associazione accettata è identica a quella di *.inputrc*, ma ciascuna associazione deve essere passata come argomento separato; per esempio, '"\C-x\C-r": re−read−init−file´. Le opzioni, se fornite, hanno i seguenti significati:

−m *keymap*

Use *keymap* come la mappa di tasti che deve essere influenzata dalle successive associazioni. I nomi accettabili per *keymap* sono *emacs*, *emacs-standard*, *emacs-meta*, *emacs-ctlx*, *vi*, *vi-move*, *vi-command*, e *vi-insert*. *vi* è equivalente a *vi-command*; *emacs* è equivalente a *emacs-standard*.

−l Elenca i nomi di tutte le funzioni di **readline**

−v Elenca i nomi delle funzioni correnti e le loro associazioni

−d Emette i nomi delle funzioni e loro associazioni in modo tale che possano essere riletti

−f *nomefile*

Legge le associazioni dei tasti dal *nomefile*

−q *funzione*

Chiede quali tasti chimano la *funzione* indicata

Il valore di ritorno è 0 a meno che non sia data una opzione non riconosciuta o sia avvenuto un errore.

break [*n*]

Esce dall'interno di un ciclo **for**, **while**, o **until**. Se *n* è specificato, interrompe *n* livelli. *n* deve essere ≥ 1. Se *n* è più grande del numero di cicli racchiusi, tutti i cicli racchiusi sono terminati. Il valore di ritorno è 0 a meno che la shell non stia eseguendo un ciclo quando **break** è eseguito.

builtin *comando−incorporato* [*argomenti*]
Esegue il comando incorporato specificato, passandogli gli *argomenti*, e ritorna il suo stato di uscita. Questo è utile quando si desidera definire una funzione il cui nome è lo stesso di un comando incorporato della shell, ma le funzionalità del comando incorporato sono necessarie dentro la funzione stessa. Il comando incorporato **cd** è comunemente ridefinito in questo modo. Lo stato di ritorno è falso se *comando−incorporato* non è un comando incorporato della shell.

cd [*dir*] Cambia la directory corrente con *dir*. La variabile **HOME** è il *dir* di default. La variabile **CDPATH** definisce i percorsi di ricerca per le directory che contengano *dir*. Nomi di directory alternative sono separate da un "due punti" (:). Un nome di directory nullo in **CDPATH** è lo stesso della directory corrente, cioè, ‘‘**.**’’. Se *dir* inizia con uno slash (/), allora **CDPATH** non è usato. Un argomento **−** è equivalente a **$OLDPWD**. Il valore di ritorno è vero se la directory è stata cambiata con successo; falso altrimenti.

command [**-pVv**] *comando* [*arg* ...]
Esegue *comando* con gli *arg* sopprimendo la normale ricerca tra le funzione di shell. Sono eseguiti solo i comandi incorporati o comandi trovati in **PATH**. Se è data l’opzione **−p**, la ricerca di *comando* viene effettuata usando un valore di default per **PATH**, per il quale è garantito che siano trovate tutte le utility standard. Se è fornita l’opzione **−V** o **−v**, viene stampata una descrizione di *comando*. L’opzione **−v** provoca la stampa di una singola parola che indica il comando o il percorso completo usato per chiamare *comando*; l’opzione **−V** produce una descrizione più completa. Un argomento **−−** disabilita il controllo delle opzioni per il resto degli argomenti. Se è fornita l’opzione **−V** o **−v**, lo stato di uscita è 0 se *comando* è stato trovato, e 1 se no. Se nessuna delle due opzioni è fornita e avviene un errore o *comando* non può essere trovato, lo stato di uscita è 127. Altrimenti, lo stato di uscita del comando incorporato **command** è lo stato di uscita di *comando*.

continue [*n*]
Riprende la successiva iterazione che contiene il ciclo **for**, **while**, o **until**. Se *n* è specificato, riprende all’*n*−esima nidificazione del ciclo. *n* deve essere ≥ 1. Se *n* è più grande del numero di cicli nidificati, riprende dall’ultimo ciclo che racchiude (il ciclo a più alto livello). Il valore di ritorno è 0 a meno che la shell non stia eseguendo un ciclo quando **continue** è eseguito.

declare [**−frxi**] [*nome*[*=valore*]]
typeset [**−frxi**] [*nome*[*=valore*]]
Dichiara variabili e/o da loro degli attributi. Se non è dato alcun *nome*, allora mostra i valori delle variabili. Le opzioni possono essere usate per restringere l’uscita alle sole variabili con gli attributi specificati.
−f Usa solo nomi di funzioni
−r Rende *nomi* accessibili in sola lettura. A questi nomi non possono quindi essere assegnati valori da successive istruzioni di assegnamento.
−x Marca *nomi* per l’esportazione ai successivi comandi attraverso l’ambiente.
−i La variabile è trattata come un intero; la valutazione aritmetica (si veda **CALCOLO ARITMETICO)** è effettuata quando alla variabile è assegnato un valore.

Usare ‘+’ invece di ‘−’ mette a off gli attributi. Quando usato in una funzione, rende *nomi* locali, come con il comando **local**. Il valore di ritorno è 0 a meno che non sia incontrata una opzione illegale, sia fatto un tentativo di definire una funzione usando "-f foo=bar", uno dei *nomi* non è un nome legale di variabile di shell, e fatto un tentativo di mettere a off lo stato di sola lettura per una variabile a sola lettura, o è fatto un tentativo di mostrare una funzione non esistente con -f.

dirs [-l] [+/−n]
Mostra la lista delle directory correntemente ricordate. Le directory sono aggiunte alla lista con il comando **pushd**; il comando **popd** risale attraverso la lista.
+n evidenzia l’*n*−esima directory contando dalla sinistra della lista mostrata da **dirs** quando è chiamato senza opzioni, partendo da zero.
−n evidenzia l’*n*−esima directory contando dalla destra della lista mostrata da **dirs** quando è chiamato senza opzioni, partendo da zero.

−l produce un elenco più lungo; il formato di default dell'elenco usa un carattere tilde per indicare la home directory.

Il valore di ritorno è 0 a meno che sia fornita una opzione illegale o *n* indirizza oltre la fine dello stack delle directory.

echo [**−neE**] [*arg* ...]

Emette gli *arg*s, separati da spazii. Lo stato di ritorno è sempre 0. Se è specificato **−n**, il newline in coda è soppresso. Se è data l'opzione **−e**, viene abilitata l'interpretazione dei successivi caratteri preceduti dal carattere di escape backslash. L'opzione **−E** disabilita l'interpretazione di questi caratteri di escape, perfino su sistemi dove essi sono interpretati per default.

\a alert (campanello)
\b spazio indietro
\c sopprime il newline in coda
\f avanzamento pagina
\n nuova linea
\r ritorno carrello
\t tab orizzontale
\v tab verticale
\\ backslash
\nnn il carattere il cui codice ASCII è *nnn* (ottale)

enable [**−n**] [**−all**] [*nome* ...]

Abilita e disabilita i comandi incorporati della shell. Questo permette l'esecuzione di un comando su disco che ha lo stesso nome di un comado incorporato della shell senza dover specificare un percorso completo. Se è usato **−n**, ogni *nome* è disabilitato; altrimenti, *nomi* sono abilitati. Per esempio, per usare il programma **test** trovato attraverso il **PATH**, invece della versione incorporata nella shell, si digita ''enable -n test''. Se nessun argomento è dato, viene stampata una lista di tutti i comandi incorporati della shell abilitati. Se è fornita solo **−n**, viene stampata una lista di tutti i comandi incorporati disabilitati. Se è fornita solo **−all**, la lista stampata include tutti i comandi incorporati, con l'indicazione per ciascuno se è abilitato o no. **enable** accetta **−a** come sinonimo per **−all**. Il valore di ritorno è 0 a meno che *nome* non è un comando incorporato della shell.

eval [*arg* ...]

Gli *arg* sono letti e concatenati insieme in un singolo comando. Questo comando è quindi letto ed eseguito dalla shell, e il suo stato di uscita è ritornato come valore del comando **eval**. Se non vi è alcun *arg*, o vi sono solo argomenti nulli, **eval** ritorna vero.

exec [[**−**] *comando* [*argomenti*]]

Se *comando* è specificato, esso sostituisce la shell. Nessun nuovo processo è creato. Gli *argomenti* diventano gli argomenti per *comando*. Se il primo argomento è **−**, la shell pone un segno meno nello zeresimo arg passato al *comando*. Questo è ciò che fa login. Se il file non può essere eseguito, per alcune ragioni, una shell non interattiva termina, a meno che esista la variabile di shell **no_exit_on_failed_exec**, nel qual caso essa ritorna fallimento. Una shell interattiva ritorna fallimento se il file non può essere eseguito. Se *comando* non è specificato, qualsiasi ridirezione ha effetto nella corrente shell, e lo stato di ritorno è 0.

exit [*n*] Causa l'uscita della shell con uno stato di uscita *n*. Se *n* è omesso, lo stato di uscita è quello dell'ultimo comando eseguito. Una trappola su **EXIT** è eseguita prima che la shell termini.

export [**−nf**] [*nome*[=*parola*]] ...
export −p

I *nomi* forniti sono marcati per esportazione automatica nell'ambiente dei comandi successivamente eseguiti. Se è data l' opzione **−f**, i *nomi* si riferiscono a funzioni. Se non è dato alcun *nome* o se se è fornita l'opzione **−p**, viene stampata una lista di tutti i nomi che sono esportati in questa shell. L'opzione **−n** provoca la rimozione della proprietà di esportazione dalle variabili indicate. Un argomento **−−** disabilita il controllo delle opzioni per il resto degli argomenti. **export** ritorna una stato di uscita di 0 a meno che sia incontrata una opzione illegale, uno dei *nomi* non è un nome legale di variabile di shell, o è fornita l'opzione **−f** con un *nome* che non è una

funzione.

fc [**−e** *nome−editor*] [**−nlr**] [*primo*] [*ultimo*]
fc −s [*pat=rep*] [*cmd*]

Corregge un comando. Nella prima forma, un intervallo di comandi da *primo* a *ultimo* è selezionata dalla lista della storia. *primo* e *ultimo* possono essere specificati come una stringa (per localizzare l'ultimo comando che inizia con quella stringa) o come numero (un indice nella lista della storia, dove un numero negativo è usato come offset dal numero del comando corrente). Se *ultimo* non è specificato esso è posto al comando corrente, se per l'elencazione (così che **fc −l −10** stampa gli ultimi 10 comandi), e a *primo* altrimenti. Se *primo* non è specificato esso è posto al precedente comando, se per l'editing e a −16 per l'elencazione.

Il flag **−n** sopprime i numeri dei comandi quando li elenca. Il flag **−r** inverte l'ordine dei comandi. Se è dato il flag **−l**, i comandi sono elencati sullo standard output. Altrimenti, è chiamato l'editor indicato da *nome−editor* su un file che contiene questi comandi. Se *nome−editor* non è dato, è usato il valore della variabile **FCEDIT**, o il valore di **EDITOR** se **FCEDIT** non è posta. Se ne l'una ne l'altra variabile è posta, è usato *vi* Quando l'editing è completato, i comandi editati sono ribaditi ed eseguiti.

Nella seconda forma, *comando* è rieseguito dopo che ogni instanza di *pat* è sostituita da *rep*. Un utile alias da usare con questo è ''r=fc −s'', così che digitare ''r cc'' esegue l'ultimo comando che inizia con ''cc'' e digitando ''r'' riesegue l'ultimo comando.

Se è usata la prima forma, il valore di ritorno è 0 a meno che sia incontrata una opzione illegale o *primo* o *ultimo* specificano linee della storia fuori dall'intervallo. Se è fornita l'opzione **−e**, il valore di ritorno è il valore dell'ultimo comando eseguito, o fallimento se avviene un errore con il file di comandi temporanei. Se è usata la seconda forma, lo stato di ritorno è quello del comando rieseguito, a meno che *cmd* specifichi una linea di storia non valida, nel qual caso **fc** ritorna fallimento.

fg [*jobspec*]

Pone *jobspec* in foreground, e lo rende il job corrente. Se *jobspec* non è presente, è usata la nozione della shell di *job corrente*. Il valore di ritorno è quello del comando posto in foreground, o fallimento se eseguito quando il job control è disabilitato o, quando eseguito con job control abilitato, se *jobspec* non specifica un job valido o *jobspec* specifica un job che era stato avviato senza job control.

getopts *optstring nome* [*args*]

getopts è usato dalle procedure di shell per analizzare i parametri posizionali. *optstring* contiene le lettere delle opzioni che devono essere riconosciute; se una lettera è seguita da un ":", l'opzione si aspetta di avere un argomento, che dovrà essere separato da essa da spazi bianchi. Ogni volta che è chiamata, **getopts** pone la successiva opzione nella variabile di shell *nome*, inizializzando *nome* se non esiste, e l'indice del prossimo argomento che deve essere processato nella variabile **OPTIND**. **OPTIND** è initializzato a 1 ogni volta che la shell o uno shell script viene chiamato. Quando una opzione richiede un argomento, **getopts** pone quell'argomento nella variabile **OPTARG**. La shell non azzera **OPTIND** automaticamente; essa deve essere manualmente azzerata tra più chiamate a **getopts** dentro la stessa esecuzione della shell, se deve essere usato un nuovo insieme di parametri.

getopts puo informare degli errori in due modi. Se il primo carattere di *optstring* è un "due punti", è usata una informazione di errore *silenzioza*. Nelle normali operationi i messaggi diagnostici sono stampati quando sono incontrate opzioni illegali o mancano gli argomenti delle opzioni. Se la variabile **OPTERR** è posta a 0, nessun messaggio di errore sarà mostrato, perfino se il primo carattere di *optstring* non è un "due punti".

Se è incontrata una opzione illegale, **getopts** pone ? nel *nome* e, se non silenziosa, stampa an

messaggio di errore ed elimina **OPTARG**. Se **getopts** è silenziosa, il carattere di opzione trovato è posto in **OPTARG** e nessun messaggio diagnostico è stampato.

Se un argomento richiesto non è trovato, e **getopts** non è silenziosa, un punto interrogativo (**?**) è posto in *nome*, **OPTARG** non è posto, ed è stampato un messaggio diagnostico. Se **getopts** è silenzioza, allora un "due punti" (**:**) è posto in *nome* e **OPTARG** è posto al carattere di opzione trovato.

getopts normamente analizza i parametri posizionali, ma se sono dati più argomenti in *args*, **getopts**, invece, analizza questi. **getopts** ritorna vero se è trovata una opzione, specificata o non specificata. Ritorna falso se è incontrata la fine delle opzioni o avviene un errore.

hash [**−r**] [*nome*]
Per ciascun *nome*, è determinato e ricordato il percorso completo del comando. L'opzione **−r** fa sì che la shell dimentichi tutte le locazioni ricordate. Se nessun argomento è dato, è stampata l'informazione circa i comandi ricordati. Un argomento **−−** disabilita il controllo delle opzioni per il resto degli argomenti. Lo stato di ritorno è vero a meno che un *nome* sia irreperibile o sia fornita una opzione illegale.

help [*pattern*]
Mostra utili informazioni sui i comandi incorporati. Se *pattern* è specificato, **help** da un aiuto dettagliato su tutti i comandi che combaciano con *pattern*; altrimenti è stampata una lista dei comandi incorporati. Lo stato di ritorno è 0 a meno che nessun comando combaci con *pattern*.

history [*n*]
history −rwan [*nomefile*]
Senza opzioni, mostra la lista della storia dei comandi con i numeri di linea. Le linee elencate con un ***** sono state modificate. Con argomento *n* elenca solo le ultime *n* linee. Se è fornito un argomento non-opzione, esso è usato come nome del file di storia; se no, è usato il valore di **HISTFILE**. Le opzioni, se fornite, hanno i seguenti significati:

- **−a** Aggiunge le ''nuove'' linee della storia (linee della storia inserite dall'inizio della corrente sessione di **bash**) al file di storia
- **−n** Inserisce le linee della storia non ancora lette dal file di storia, nella corrente lista della storia. Queste sono le linee aggiunte al file di storia dall'inizio della corrente sessione di **bash**.
- **−r** Legge il contenuto del file di storia e lo usa come storia corrente.
- **−w** scrive la storia corrente sul file di storia, sovrascrivendo il contenuto del file di storia.

Il valore di ritorno è 0 a meno che sia incontrata una opzione illegale o avvenga un errore mentre si legge o si scrive il file di storia.

jobs [**−lnp**] [*jobspec* ...]
jobs −x *comando* [*args* ...]
La prima forma elenca i job attivi. L'opzione **−l** elenca gli ID dei processi in aggiunta alle normali informazioni; l'opzione **−p** elenca solo l'ID del primo processo del gruppo di processi del job. L'opzione **−n** mostra solo i job che hanno cambiato stato dall'ultima notifica. Se *jobspec* è dato, l'uscita è ristretta alle informazioni su quel job. Lo stato di ritorno è 0 a meno che sia incontrata una opzione illegale o sia fornito un *jobspec* illegale.

Se è fornita l'opzione **−x**, **jobs** sostituisce qualsiasi *jobspec* trovato in *comando* o *args* con il corrispondente ID del gruppo di processi, ed esegue *comando* passandogli *args*, e ritornando il suo stato di uscita.

kill [**-s sigspec** | **−sigspec**] [*pid* | *jobspec*] ...
kill −l [*signum*]
Manda il segnale indicato da *sigspec* al processo indicato da *pid* o *jobspec*. *sigspec* è o un nome di segnale, come **SIGKILL**, o un numero di segnale. Se *sigspec* è un nome di segnale, il nome è insensibile alle minuscole e maiuscole, e può essere dato con o senza il prefisso **SIG**. Se *sigspec*

non è presente, si assume **SIGTERM**. Un argomento **−l** elenca i nomi dei segnali. Se sono forniti degli argomenti quando è data l'opzione **−l**, sono elencati i nomi dei segnali specificati, e lo stato di ritorno è 0. Un argomento **−−** disabilita il controllo delle opzioni per il resto degli argomenti. **kill** ritorna vero se almeno un segnale è stato mandato con successo, o falso se avviene un errore o è incontrata una opzione illegale.

let *arg* [*arg* ...]
Ogni *arg* è una espressione aritmetica che deve essere valuatata (si veda **CALCOLO ARITMETICO**). Se l'ultimo *arg* viene valutato 0, **let** ritorna 1; altrimenti è ritornato 0.

local [*nome*[=*valore*] ...]
Per ogni argomento, crea una variabile locale chiamata *nome*, e gli assegna *valore*. Quando **local** è usato dentro una funzione, fa sì che la variabile *nome* abbia una visibilità ristretta a quella funzione e ai suoi figli. Senza alcun operando, **local** scrive una lista di variabili locali sullo standard output. È un errore usare **local** quando non si è dentro una funzione. Lo stato di ritorno è 0 a meno che **local** sia usata fuori da una funzione, o è fornito un *nome* illegale.

logout Termina una shell di login.

popd [**+/−n**]
Rimuove degli elementi dallo stack delle directory. Senza argomenti, rimuove la directory in cima allo stack, ed effettua un **cd** verso la nuova directory in cima allo stack.
+n rimuove l'*n*−esimo elemento contando dalla sinistra della lista mostrata da **dirs**, partendo da zero. Per esempio: ''popd +0'' rimuove la prima directory, ''popd +1'' la seconda.
−n rimuove l'*n*−esimo elemento contando dalla destra della lista mostrata da **dirs**, partendo da zero. Per esempio: ''popd -0'' rimuove l'ultima directory, ''popd -1'' la penultima.

Se il comando **popd** ha successo, viene anche effettuato un **dirs**, e lo stato di ritorno è 0. **popd** ritorna falso se è incontrata una opzione illegale, lo stack delle directory è vuoto, è specificato un elemento non esistente nello stack delle directory, o il cambio di directory fallisce.

pushd [*dir*]
pushd +/−n
Aggiunge una directory in cima allo stack delle directory, o ruota lo stack, rendendo la corrente directory di lavoro la nuova cima dello stack. Senza argomenti, scambia le due directory in cima e ritorna 0, a meno che lo stack delle directory sia vuoto.
+n Ruota lo stack così che la *n*−esima directory (contando dalla sinistra della lista mostrata da **dirs**) vada in cima.
−n Ruota lo stack così che la *n*−esima directory (contando da destra) vada in cima.
dir aggiunge *dir* in cima allo stack delle directory, rendendola la nuova directory di lavoro corrente.

Se il comando **pushd** ha successo, viene anche effettuato un **dirs**. Se è usata la prima forma, **pushd** ritorna 0 a meno che il cd verso *dir* fallisca. Con la seconda forma, **pushd** ritorna 0 a meno che lo stack delle directory sia vuoto, sia specificato un elemento non esistente nello stack delle directory, o il cambiamento di directory verso la nuova directory corrente specificata fallisca.

pwd Stampa il percorso assoluto della corrente directory di lavoro. Il percorso stampato non contiene alcun link simbolico, se è posta l'opzione **−P** al comando incorporato **set** (si veda anche la descrizione di **nolinks** sotto **Variabili di shell** sopra). Lo stato di ritorno è 0 a meno che avvenga un errore mentre si legge il percorso della directory corrente.

read [**−r**] [*nome* ...]
Una linea è letta dallo standard input, e la prima parola è assegnata al primo *nome*, la seconda parola al secondo *nome*, e così via, con le parole rimaste assegnate all'ultimo *nome*. Solo i caratteri in **IFS** sono riconosciuti come delimitatori di parola. Se non è fornito alcun *nome*, la linea letta è assegnata alla variabile **REPLY**. Il codice di ritorno è zero, a meno che sia incontrata la fine del file. Se è data l'opzione **−r**, una coppia backslash-newline non è ignorata, e il backslash è considerato essere parte della linea.

readonly [**−f**] [*nome* ...]
readonly -p

I *nomi* dati sono marcati per sola lettura e i valori di questi *nomi* non possono essere cambiati dagli assegnamenti successivi. Se è fornita l'opzione **−f**, sono così marcate le funzioni corrispondenti ai *nomi*. Se nessun argomento è dato, o se è fornita l'opzione **−p**, viene stampata una lista di tutti i nomi a sola lettura. Un argomento **−−** disabilita il controllo delle opzioni per il resto degli argomenti. Lo stato di ritorno è 0 a meno che sia incontrata una opzione illegale, uno dei *nomi* non è un nome legale di variabile di shell, o sia fornita l'opzione **−f** con un *nome* che non è una funzione.

return [*n*]

Fa sì che una funzione esca con il valore di ritorno specificato da *n*. Se *n* è omesso, lo stato di ritorno è quello dell'ultimo comando eseguito nel corpo della funzione. Se usato fuori da una funzione, ma durante l'esecuzione di un script tramite il comando **.** (**sorgente**), fa sì che la shell smetta di eseguire quello script e ritorni o *n* o lo stato di uscita dell'ultimo comando eseguito dentro lo script, come stato di uscita dello script. Se usato fuori da una funzione e non durante l'esecuzione di uno script per mezzo di **.**, lo stato di ritorno è falso.

set [**−−abefhkmnptuvxldCHP**] [**-o** *opzione*] [*arg* ...]

−a Automaticamente marca le variabili, che sono modificate o create, per l'esportazione verso l'ambiente dei comandi successivi.

−b Fa sì che lo stato di un job in background terminato venga riportato immediatamente, piuttosto che prima del successivo prompt primario (si veda anche **notify** sotto **Variabili di shell** precedentemente).

−e Esce immediatamente se un *comando semplice* (si veda **GRAMMATICA DELLA SHELL** sopra) termina con uno stato diverso da zero. La shell non esce se il comando che fallisce è parte di un ciclo *until* o *while*, parte di una istruzione *if*, parte di una lista **&&** o **||**, o se il valore di ritorno del comando è stato invertito per mezzo di **!**.

−f Disabilita l'espansione di percorso.

−h Localizza e ricorda i comandi funzione appena le funzioni sono definite. Normalmente i comandi funzione sono cercati quando la funzione è eseguita.

−k Tutti gli argomenti delle parole chiave sono posti nell'ambiente per un comando, non solo quelli che precedono il nome del comando.

−m Modo monitor. Il job control è abilitato. Questo flag è on per default per le shell interattive, su sistemi che lo supportano (si veda **JOB CONTROL** sopra). I processi in background girano in un gruppo di processi separato e una linea che contiene il loro stato di uscita viene stampata al loro completamento.

−n Legge i comandi ma non li esegue. Questo può essere usato per controllare uno shell script per errori di sintassi. Questo è ignorato per le shell interattive.

−o *nome-opzione*

Il *nome-opzione* può essere uno dei seguenti:

allexport
Lo stesso di **−a**.

braceexpand
La shell effettua l'espansione delle parentesi graffe (si veda **Espansione delle parentesi graffe** sopra). Questo è on per default.

emacs Usa una interfaccia di editing della linea di comando in stile emacs. Questo è abilitato per default quando la shell è interattiva, a meno che la shell sia avviata con l'opzione **−nolineediting**.

errexit Lo stesso di **−e**.

histexpand
Lo stesso di **−H**.

ignoreeof
L'effetto è lo stesso che se fosse stato eseguito il comando di shell 'IGNOREEOF=10' (si veda **Variabili di shell** sopra).

interactive−comments
Permette che una parola che inizia con **#** causi che la parola e tutti i rimanenti caratteri su quella linea siano ignorati in una shell interattiva (si veda **COMMENTI** sopra).

monitor Lo stesso di **−m**.

noclobber
Lo stesso di **−C**.

noexec Lo stesso di **−n**.

noglob Lo stesso di **−f**.

nohash Lo stesso di **−d**.

notify Lo stesso di **−b**.

nounset Lo stesso di **−u**.

physical Lo stesso di **−P**.

posix Cambia il comportamento di bash dove le operazioni di default differiscono dallo standard Posix 1003.2 per farlo combaciare con lo standard.

privileged
Lo stesso di **−p**.

verbose Lo stesso di **−v**.

vi Usa una interfaccia di editing della linea di comando in stile vi.

xtrace Lo stesso di **−x**.

Se non è fornito alcun *nome-opzione*, sono stampati i valori delle opzioni correnti.

−p Attiva il modo *privilegiato*. In questo modo, il file **$ENV** non è processato, e le funzioni di shell non sono ereditate dall'ambiente. Questo è abilitato automaticamente all'avvio se l'id effettivo dell'utente (gruppo) non è uguale all'id reale dell'utente (gruppo). Disattivare questa opzione fa sì che l'id effettivo dell'utente e del gruppo sia posto all'id reale dell'utente e del gruppo.

−t Esce dopo aver letto ed eseguito un comando.

−u Considera errore le variabili non impostate quando effettua l'espansione di parametro. Se l'espansione è tentata su una variabile non impostata, la shell stampa un messaggio di errore e, se non interattiva, esce con uno stato diverso da zero.

−v Stampa le linee in input alla shell appena sono lette.

−x Dopo l'espansione di ogni *comando semplice* , **bash** mostra il valore espanso di **PS4**, seguito dal comando e dai suoi argomenti espansi.

−l Salva e ripristina l'associazione di *nome* in un comando **for** *nome* [**in** parola] (si veda **GRAMMATICA DELLA SHELL** sopra).

−d Disabilita l'hashing dei comandi che sono cercati per essere eseguiti. Normalmente, i comandi sono ricordati in una tavola hash, e una volta trovati, non devono essere più cercati.

−C L'effetto è lo stesso che se fosse stato eseguito il comando di shell 'noclobber=' (si veda **Variabili di shell** sopra).

−H Abilita la sostituzione della storia in stile **"!"**. Questo flag è on per default quando la shell è interattiva.

−P Se posto, non segue i link simbolici quando esegue comandi che, come **cd**, cambiano la directory corrente. È, invece, usata la directory fisica.

−− Se nessun argomento segue questo flag, allora i parametri posizionali non sono impostati. Altrimenti, i parametri posizionali sono posti agli *arg*, persino se alcuni di loro iniziano con un **−**.

− Segnala la fine delle opzioni, e fa sì che tutti i rimanenti *arg*s siano assegnati ai parametri posizionali. Le opzioni **−x** e **−v** sono messe a off. Se non vi è alcun *arg*, i parametri posizionali rimangono inalterati.

I flag sono messi a off per default salvo specificato diversamente. Usare + piuttosto che − causa che questi flag siano posti a off. I flag possono anche essere specificati come opzioni in una chiamata della shell. Il corrente insieme di flag può essere trovato in **$−**. Dopo che gli argomenti opzione sono processati, i rimanenti *n arg* sono trattati come valori per i parametri posizionali e

sono assegnati, in ordine, a **$1**, **$2**, **...** **$***n*. Se non è fornita nessuna opzione o *arg*, sono stampate tutte le variabili di shell. Lo stato di ritorno è sempre vero a meno che sia incontrata una opzione illegale.

shift [*n*]

I parametri posizionali da *n*+1 ... sono rinominati **$1 ....** I parametri rappresentati dai numeri **$#** fino a **$#**−*n*+1 non sono impostati. Se *n* è 0, nessun parametro viene cambiato. Se *n* non è dato, è assunto essere 1. *n* deve essere un numero non negativo minore o uguale a **$#**. Se *n* è più grande di **$#**, i parametri posizionali non sono cambiati. Lo stato di ritorno è più grande di 0 se *n* è più grande di **$#** o minore di 0; altrimenti 0.

suspend [**−f**]

Sospende l'esecuzione di questa shell fino a che non riceve un segnale **SIGCONT**. L'opzione **−f** dice di non lamentarsi se questa è una shell di login; ma la sospende comunque. Lo stato di ritorno è 0 a meno che la shell sia una shell di login e non è fornito **−f**, o se il job control non è abilitato.

test *expr*

[*expr* **]** Ritorna uno stato di 0 (vero) o 1 (falso) dipendente dalla valutazione della espressione condizionale *expr*. Le espressioni possono essere unarie o binarie. Le espressioni unarie sono spesso usate per esaminare lo stato di un file. Vi sono operatori su stringa e anche operatori di comparazione numerica. Ogni operatore e operando deve essere un argomento separato. Se *file* è della forma /dev/fd/*n*, allora è controllato il descrittore di file *n*.

−b *file* Vero se *file* esiste ed è speciale a blocchi.
−c *file* Vero se *file* esiste ed è speciale a caratteri.
−d *file* Vero se *file* esiste ed è una directory.
−e *file* Vero se *file* esiste.
−f *file* Vero se *file* esiste ed è un file normale.
−g *file* Vero se *file* esiste ed è impostato il suo bit set-group-id.
−k *file* Vero se *file* ha il suo ''sticky'' bit impostato.
−L *file* Vero se *file* esiste ed è un link simbolico.
−p *file* Vero se *file* esiste ed è una pipe con nome.
−r *file* Vero se *file* esiste ed è leggibile.
−s *file* Vero se *file* esiste ed ha una grandezza maggiore di zero.
−S *file* Vero se *file* esiste ed è un socket.
−t *fd* Vero se *fd* è aperto su un terminale.
−u *file* Vero se *file* esiste ed è impostato il suo bit set-user-id.
−w *file* Vero se *file* esiste ed è scrivibile.
−x *file* Vero se *file* esiste ed è eseguibile.
−O *file* Vero se *file* esiste ed è posseduto dall'utente con l'id effettivo.
−G *file* Vero se *file* esiste ed è posseduto dal gruppo con l'id effettivo.
file1 **−nt** *file2*
Vero se *file1* è più recente (rispetto alla data di modifica) di *file2*.
file1 **−ot** *file2*
Vero se *file1* è più vecchio di file2.
file1 **−ef** *file*
Vero se *file1* e *file2* sono sullo stesso dispositivo e hanno lo stesso numero di inode.
−z *stringa*
Vero se la lunghezza di *stringa* è zero.
−n *stringa*
stringa Vero se la lunghezza di *stringa* è diversa da zero.
stringa1 **=** *stringa2*
Vero se le stringhe sono uguali.
stringa1 **!=** *stringa2*
Vero se le stringhe non sono uguali.

! *expr* Vero se *expr* è falso.

expr1 **−a** *expr2*
Vero se entrambe *expr1* E *expr2* sono vere.

expr1 **−o** *expr2*
Vero se o *expr1* O *expr2* è vera.

arg1 **OP** *arg2*
OP è uno tra **−eq**, **−ne**, **−lt**, **−le**, **−gt**, o **−ge**. Questi operatori aritmetici binari ritornano vero se *arg1* è, rispettivamente, uguale, non uguale, minore, minore o uguale, maggiore, o maggiore o uguale ad *arg2*. *Arg1* e *arg2* possono essere interi positivi, interi negativi, o la speciale espressione **−l** *stringa*, che calcola la lunghezza di *stringa*.

times Stampa i tempi accumulati in user e system per la shell e per i processi eseguiti dalla shell. Lo stato di ritorno è 0.

trap [**−l**] [*arg*] [*sigspec*]
Il comando *arg* dovrà essere letto e eseguito quando la shell riceve il segnale(i) *sigspec*. Se *arg* è assente o −, tutti i segnali specificati sono riportati al loro valore originale (i valori che essi avevano al momento dell'ingresso nella shell). Se *arg* è la stringa nulla questo segnale è ignorato dalla shell e dai comandi che essa lancia. *sigspec* è o un nome di segnale definito in <*signal.h*>, o un numero di segnale. Se *sigspec* è **EXIT** (0) il comando *arg* è eseguito all'uscita della shell. Senza argomenti, **trap** stampa la lista di comandi associati con ciascun numero di segnali. L'opzione **−l** fa sì che la shell stampi una lista di nomi di segnali e i loro numeri corrispondenti. Un argomento −− disabilita il controllo delle opzioni per il resto degli argomenti. I segnali ignorati al momento dell'ingresso della shell non possono essere intercettati o resettati. I segnali intercettati sono riportati al loro valore originale in un processo figlio quando esso è creato. Lo stato di ritorno è falso se o il nome o il numero della trappola non è valido; altrimenti **trap** ritorna vero.

type [**−all**] [**−type** | **−path**] *nome* [*nome* ...]
Senza opzioni, indica come dovrà essere interpretato ciascun *nome* se usato come un nome di comando. Se è usato il flag **−type**, **type** stampa una frase che è una fra *alias*, *keyword*, *function*, *builtin*, o *file* se *nome* è, rispettivamente un alias, una parola riservata della shell, una funzione, un comando incorporato, o un file su disco. Se il nome non è trovato, allora non è stampato nulla, ed è ritornato uno stato di uscita di falso. Se è usato il flag **−path**, **type** ritorna o il nome del file su disco che dovrebbe essere eseguito se *nome* fosse specificato come un nome di comando, o nulla se **−type** non avrebbe ritornato *file*. Se un comando è nella tavola hash, **−path** stampa il valore hash, non necessariamente il file che appare per primo in **PATH**. Se è usato il flag **−all**, **type** stampa tutti i posti che contengono un eseguibile chiamato *nome*. Questo include alias e funzioni, se e solo se non è usato anche il flag **−path**. La tavola hash dei comandi non è consultata quando si usa **−all**. **type** accetta **−a**, **−t**, e **−p** in luogo di **−all**, **−type**, e **−path**, rispettivamente. Un argomento −− disabilita il controllo delle opzioni per il resto degli argomenti. **type** ritorna vero se uno qualsiasi degli argomenti viene trovato, falso se non ne è trovato nessuno.

ulimit [**−SHacdfmstpnuv** [*limite*]]
Ulimit fornisce il controlo sulle risorse disponibili per la shell e per i processi avviati da essa, sui sistemi che permettono un tale controllo. Il valore di *limite* può essere un numero nell'unità specificata per la risorsa, o il valore **unlimited**. Le opzioni **H** e **S** specificano che viene impostato il limite hard o limite soft per la data risorsa. Un limite hard non può essere aumentato una volta che è impostato; un limite soft può essere aumentato fino al valore del limite hard. Se non è specificato ne **H** ne **S**, il comando applica il limite soft. Se *limite* è omesso, viene stampato il valore corrente del limite soft della risorsa, a meno che sia data l'opzione **H**. Quando è specificata più di una risorsa, il nome del limite e l'unità vengono stampati prima del valore. Le altre opzioni sono interpretate come segue:

−a sono riportati tutti i limiti correnti
−c la grandezza massima dei file core creati
−d la grandezza massima del segmento dati di un processo
−f la grandezza massima dei file creati dalla shell

−m la grandezza massima della memoria occupata.
−s la grandezza massima dello stack
−t il massimo quantitativo di tempo di cpu in secondi
−p la grandezza della pipe in blocchi da 512 byte (questo non può essere cambiato)
−n il numero massimo di descrittori di file aperti (la maggior parte dei sistemi non permette che questo valore sia impostato, ma solo mostrato)
−u il numero massimo di processi disponibili per un singolo utente
−v Il massimo ammontare di memoria virtuale disponibile per la shell

Un argomento **−−** disabilita il controllo delle opzioni per il resto degli argomenti. Se *limite* è dato, esso è il nuovo valore della risorsa specificata (l'opzione **−a** è solo per visualizzazione). Se nessuna opzione è data, allora è assunta **−f**. I valori sono in multipli di 1024 byte, tranne che per **−t**, che è in secondi, **−p**, che è in unità di blocchi da 512 byte, e **−n** e **−u**, che sono numeri senza unità. Lo stato di ritorno è 0 a meno che sia incontrata una opzione illegale, è fornito come *limite* un argomento non numerico diverso da **unlimited**, o avvenga un errore mentre si imposta un nuovo limite.

umask [**−S**] [*modo*]
La maschera di creazione dei file dell'utente è posta a *modo*. Se *modo* inizia con una cifra, esso è interpretato come numero ottale; altrimenti è interpretato come un modo di maschera simbolico simile a quello accettato da *chmod*(1). Se *modo* è omesso, o se è fornita l'opzione **−S**, viene stampato il valore corrente della maschera. L'opzione **−S** fa sì che la maschera venga stampata in formato simbolico; l'uscita di default è un numero ottale. Un argomento **−−** disabilita il controllo delle opzioni per il resto degli argomenti. Lo stato di ritorno è 0 se il modo è stato cambiato con successo o se nessun argomento *modo* era stato fornito, e falso altrimenti.

unalias [**−a**] [*nome* ...]
Rimuove *nome* dalla lista degli alias definiti. Se è fornita **−a**, sono rimosse tutte le definizioni di alias. Il valore di ritorno è vero a meno che un *nome* fornito non è un alias già definito.

unset [**−fv**] [*nome* ...]
Per ciascun *nome*, rimuove la corrispondente variabile o, se è data l'opzione **−f**, funzione. Un argomento **−−** disabilita il controllo delle opzioni per il resto degli argomenti. È da notare che **PATH**, **IFS**, **PPID**, **PS1**, **PS2**, **UID**, e **EUID** non possono essere rimosse. Se una qualsiasi fra **RANDOM**, **SECONDS**, **LINENO**, o **HISTCMD** è rimossa, perde la sua speciale proprietà, persino se essa viene successivamente ripristinata. Lo stato di uscita è vero a meno che un *nome* è inesistente o è "non impostabile".

wait [*n*]
Aspetta la terminazione del processo specificato e ritorna il suo stato di uscita. *n* può essere un ID di processo o una specificazione di job; se è data una specificazione di job, si aspetta la terminazione di tutti i processi nella pipeline di quel job. Se *n* non è dato, si aspetta la terminazione di tutti i processi figli correntemente attivi, e lo stato di ritorno è zero. Se *n* specifica un processo o job non esistente, lo stato di ritorno è 127. Altrimenti, lo stato di ritorno è lo stato di uscita dell'ultimo processo o job per cui si era in attesa del termine.

INVOCAZIONE

Una *shell di login* è una il cui primo carattere dell'argomento zero è un **−**, o una avviata con il flag **−login**.

Una shell *interattiva* è una i cui standard input e output sono entrambi connessi ai terminali (come determinato da *isatty*(3)), o una avviata con l'opzione **−i**. **PS1** è impostato e **$−** include **i** se **bash** è interattiva, permettendo a uno script di shell o un file di inizializzazione di controllare questo stato.

Shell di login:
Alla login (dipendentemente dalla opzione **−noprofile**):
se */etc/profile* esiste, lo legge.

se *~/.bash_profile* esiste, lo legge,
altrimenti se *~/.bash_login* esiste, lo legge,

altrimenti se *~/.profile* esiste, lo legge.

All'uscita:
se *~/.bash_logout* esiste, lo legge.

Shell interattiva non di login:
Alla partenza (dipendentemente dalle opzioni **−norc** e **−rcfile**):
se *~/.bashrc* esiste, lo legge.

Shell non interattiva:
Alla partenza:
se la variabile di ambiente **ENV** non è nulla, la espande
e legge il file che indica, come se fosse stato eseguito,
il comando
if ["$ENV"]; then . $ENV; fi
ma non usa **PATH** per cercare il percorso del nome.
Quando non è avviata in modo Posix, bash
guarda **BASH_ENV** prima di **ENV**.

Se Bash è chiamata come **sh**, prova a imitare il comportamento di **sh** nel modo più vicino possibile. Per una shell di login, essa tenta di leggere solo */etc/profile* e *~/.profile*, in questo ordine. L'opzione **−noprofile** può anche essere usata per disabilitare questo comportamento. Una shell invocata come **sh** non tenta di leggere nessun altro file di inizializzazione.

Quando **bash** è avviata in modo *posix*, come con l'opzione **−posix** sulla linea di comando, essa segue lo standard Posix per i file di inizializzazione. In questo modo, la variabile **ENV** è espansa e quel file letto; nessun altro file di inizializzazione viene letto.

VEDERE ANCHE

Bash Features, Brian Fox e Chet Ramey
The Gnu Readline Library, Brian Fox e Chet Ramey
The Gnu History Library, Brian Fox e Chet Ramey
A System V Compatible Implementation of 4.2BSD Job Control, David Lennert
Portable Operating System Interface (POSIX) Part 2: Shell e Utilities, IEEE
sh(1), *ksh*(1), *csh*(1)
emacs(1), *vi*(1)
readline(3)

FILE USATI

/bin/bash
The **bash** eseguibile
/etc/profile
Il file di inizializzazione generale per il sistema, eseguito per le shell di login
~/.bash_profile
Il file di inizializzazione personale, eseguito per le shell di login
~/.bashrc
Il file di inizializzazione individuale per shell interattiva
~/.inputrc
Il file di inizializzazione individuale per *readline*

AUTORI

Brian Fox, Free Software Foundation (primo autore)
bfox@ai.MIT.Edu

Chet Ramey, Case Western Reserve University
chet@ins.CWRU.Edu

NOTIFICA DEI BACHI

Se trovate un baco in **bash,** dovreste renderlo noto, ma prima, dovreste assicurarvi che esso è veramente un baco, e che appare nella versione più recente di **bash** che voi avete.

Una volta che avete determinato che vi è realmente un baco, inviate per posta un rapporto sul baco a *bash−maintainers@prep.ai.MIT.Edu*. Se avete una correzione, siete invitati a inviare anche quella! Suggerimenti e rapporti su bachi "filosofici" possono essere inviati a *bug-bash@prep.ai.MIT.Edu* o inviati al newsgroup su Usenet **gnu.bash.bug**.

TUTTI i rapporti su bachi dovranno includere:

Il numero di versione di **bash**
L'hardware e il sistema operativo
Il compilatore usato per compilare
Una descrizione del comportamento del baco
Uno corto script o "ricetta" che manifesti il baco

Commenti e rapporti su bachi riguardo a questa pagina di manuale dovranno essere indirizzati a *chet@ins.CWRU.Edu* (per la traduzione italiana, a *augusto@comune.modena.it*).

BACHI

È troppo grande e troppo lenta.

Vi sono alcune sottili differenze tra **bash** e le versioni tradizionali di **sh**, soprattutto per via delle specifiche **POSIX**

Gli alias lasciano perplessi in alcuni usi.