`echo` 623 `expr` 638 `false` 636 `printf` 623 `sleep` 641 `tee` 641 `test` 636 `true` 636 `yes` 629 `[` 636

In questo capitolo si tratta in generale il problema dell'interpretazione di programmi in un sistema GNU/Linux e in particolare degli strumenti di cui ci si può avvalere utilmente per la realizzazione di script per una shell standard, con esempi e funzioni per risolvere problematiche comuni.

18.1 Eseguibili, interpretabili e automazione dell'interpretazione

Quando si utilizza un sistema operativo complesso come GNU/Linux, dove il kernel ha un ruolo così importante, è difficile stabilire una distinzione netta tra un programma eseguibile binario e un programma interpretato. A livello astratto si intende che il programma interpretato richiede un programma interprete che è di fatto il suo esecutore, ma anche l'interprete potrebbe a sua volta essere interpretato da un altro programma di livello inferiore. È un po' come quando per tradurre un testo dal cinese all'italiano, si preferisce partire dal lavoro di qualcun altro che l'ha già tradotto in inglese.

Evidentemente si pone il problema di stabilire il livello di astrazione a cui si vuole fare riferimento. Si potrebbe dire che un programma binario «normale» sia quello che viene eseguito direttamente dal kernel senza bisogno di altri sostegni da parte di programmi interpreti aggiuntivi. In questo senso, potrebbe accadere anche di avere un programma che nel sistema «A» è un binario normale, mentre nel sistema «B» potrebbe essere eseguito per opera di un interprete intermedio, diventando lì un programma interpretato.

18.1.1 Script

Il classico tipo di programma interpretato è lo script che normalmente viene individuato dalla stessa shell attraverso cui viene avviato. Per questo è stata stabilita la convenzione per cui questi programmi sono contenuti in file di testo, in cui la prima riga deve indicare il percorso dell'interprete necessario.

```
#/bin/sh
```

Tale convenzione impone che, in questo tipo di script, il simbolo '**#**' rappresenti l'inizio di un commento e che comunque si tratti di un file di testo normale. Inoltre, è stabilito implicitamente che il programma interprete indicato riceva il nome dello script da interpretare come primo argomento.

È il kernel che deve interpretare la prima riga di uno script, per mettere in esecuzione l'interprete indicato.

18.1.2 Programmi da interpretare che non sono script

Quando il file da interpretare non è così semplice come uno script, per esempio perché non si tratta di un file di testo, si pone il problema di stabilire un metodo per il suo riconoscimento, altrimenti si è costretti a usare sempre un comando che richiami esplicitamente il suo interprete. L'esempio più comune di questa situazione è il programma scritto per un'altra piattaforma che si vuole utilizzare attraverso un interprete (o un emulatore) adatto. Generalmente, questi programmi estranei sono riconoscibili in base a una stringa binaria tipica che si può trovare all'inizio del file che li contiene; in pratica, in base al magic number del file. In altre situazioni, si può essere costretti a definire un'estensione particolare per i nomi di questi file, come avviene nel Dos.

18.1.3 Gestione del kernel dei binari eterogenei

A partire dall'introduzione dell'interprete Java anche per GNU/Linux, si è sentito maggiormente il problema di organizzare in modo coerente la gestione dei programmi che per un motivo o per l'altro devono essere interpretati attraverso un programma esterno al kernel stesso. Il meccanismo attuale permette una configurazione molto semplice del sistema, attraverso la quale si può automatizzare l'interpretazione di ciò che si vuole (sezione 8.3.1).

Per verificare che il kernel sia in grado di gestire questa funzione, basta controllare che all'interno della directory '`/proc/sys/fs/binfmt_misc/`' appaiano i file '`register`' e '`status`'; il secondo in particolare, dovrebbe contenere la parola '**`enabled`**'. Se non ci sono, ma il kernel incorpora la gestione di binari da interpretare, è necessario innestare il file system '**`binfmt_misc`**':

```
# mount -t binfmt_misc none /proc/sys/fs/binfmt_misc [Invio]
```

Una volta che sono disponibili i file virtuali '`register`' e '`status`', per attivare la funzionalità occorre intervenire con il comando seguente:

```
# echo 1 > /proc/sys/fs/binfmt_misc/status [Invio]
```

Per disattivarla, basta utilizzare il valore zero.

```
# echo 0 > /proc/sys/fs/binfmt_misc/status [Invio]
```

Quando la gestione è disattivata, la lettura del file '`/proc/sys/fs/binfmt_misc/status`' restituisce la stringa '**`disabled`**'.

18.1.3.1 Configurazione

Trattandosi di un'attività che riguarda il kernel, non c'è un file di configurazione vero e proprio. Per informare il kernel della presenza di programmi da interpretare attraverso eseguibili esterni, occorre sovrascrivere un file virtuale del file system '`/proc/`'. In generale, questo si ottiene utilizzando un comando '**`echo`**', il cui standard output viene ridiretto nel file '`/proc/sys/fs/binfmt_misc/register`'. Per definire il supporto a un tipo di programma interpretato, si utilizza una riga secondo la sintassi seguente:

: ***nome*** **:** ***tipo*** **:** **[** ***scostamento*** **]** **:** ***riconoscimento*** **:** **[** ***maschera*** **]** ↩
↪ **:** ***programma_interprete*** **:**

Allo stato attuale, dal momento che i due punti verticali separano i vari campi di questo record, tale simbolo non può apparire all'interno di questi.

1. ***nome***
   Il primo campo serve a dare un nome a questo tipo di programma da interpretare. Ciò si traduce nella creazione di un file virtuale con lo stesso nome, '`/proc/sys/fs/binfmt_misc/`***nome***', che poi permette di controllarne le funzionalità.
2. ***tipo***
   Il secondo campo definisce il tipo di riconoscimento che si vuole utilizzare. La lettera '**`M`**' indica l'utilizzo di un magic number, ovvero una stringa nella parte iniziale del file, oppure la lettera '**`E`**' specifica che viene presa in considerazione l'estensione nel nome. Ciò serve a definire in che modo interpretare il quarto campo di questo record.
3. ***scostamento***
   Nel caso in cui si utilizzi un riconoscimento basato su una stringa iniziale, questa deve essere contenuta nei primi 128 byte, anche se non è detto che inizi dal primo. L'inizio della stringa di riconoscimento può essere indicato espressamente con un numero intero posto all'interno di questo campo: zero rappresenta il primo byte.
4. ***riconoscimento***

Il quarto campo consente di inserire la stringa di riconoscimento o l'estensione del file. La stringa, ovvero il magic number, può essere specificata utilizzando delle sequenze di escape che consentono l'indicazione di valori esadecimali. Per questo si usa il prefisso '**\x**', seguito da due cifre esadecimali che rappresentano un byte alla volta. A questo proposito, è bene ricordare che se il record viene definito in una riga di comando di una shell, è molto probabile che la barra obliqua inversa debba essere raddoppiata.

La stringa di riconoscimento può essere applicata a ciò che resta dopo il filtro con la maschera indicata nel campo successivo.

Nel caso si specifichi l'uso dell'estensione per riconoscere il tipo di file, questa non deve contenere il punto iniziale, che così è sottinteso.

5. ***maschera***

   Il quinto campo serve a indicare una maschera da utilizzare per filtrare i bit che compongono la parte di file che deve essere utilizzata per il riconoscimento attraverso il magic number. In pratica, di solito non si utilizza e si ottiene l'applicazione della maschera predefinita: '**\ff**'. La maschera viene applicata attraverso un AND con i byte corrispondenti del file; quello che ne deriva viene usato per il paragone con il modello specificato nel quarto campo.

   La maschera predefinita, evidentemente, non provoca alcuna modifica.

6. ***programma_interprete***

   L'ultimo campo serve a indicare il percorso assoluto dell'interprete da utilizzare per mettere in esecuzione il programma identificato attraverso questo record. Evidentemente, si presume che questo programma possa essere avviato indicando il file da interpretare come primo argomento. Se necessario, l'interprete può essere uno script predisposto opportunamente per avviare il vero interprete nel modo richiesto.

Attualmente, si pongono delle limitazioni a cui è già stato accennato in parte:

- il record che definisce un tipo di eseguibile da interpretare non può superare i 255 caratteri;
- la stringa binaria di riconoscimento, ovvero il magic number, deve trovarsi all'intero dei primi 128 byte del file, ovvero dal byte zero al byte 127, e lo scostamento non può modificare questo limite;
- il contenuto dell'ultimo campo, quello del percorso di avvio dell'interprete, non può superare i 127 caratteri.

Segue la descrizione di alcuni esempi.

- `# echo ':Java:M::\xca\xfe\xba\xbe::/usr/bin/java:' ↩`
  `↪ > /proc/sys/fs/binfmt_misc/register` [*Invio*]

  Definisce il binario Java, riconoscibile dalla sequenza esadecimale $CAFEBABE_{16}$, a partire dall'inizio del file. Per la sua interpretazione viene specificato il programma '`/usr/bin/java`', il quale potrebbe essere uno script che si occupa di avviare correttamente l'interprete giusto.

- `# echo ':Java:E::class::/usr/bin/java:' ↩`
  `↪ > /proc/sys/fs/binfmt_misc/register` [*Invio*]

  Come nell'esempio precedente, con la differenza che l'eseguibile Java viene identificato solo per la presenza dell'estensione '`.class`'.

- `# echo ':acqua:M::#acqua::/bin/acqua:' ↩`
  `↪ > /proc/sys/fs/binfmt_misc/register` [*Invio*]

  Definisce un eseguibile di tipo «acqua», riconoscibile dalla stringa iniziale '**#acqua**', a partire dall'inizio del file. Per la sua interpretazione viene specificato il programma '`/bin/acqua`'.

- `# echo ':acqua:E::acqua::/bin/acqua:' ↩`
  `↪ > /proc/sys/fs/binfmt_misc/register` [*Invio*]

  Definisce un eseguibile di tipo «acqua», riconoscibile dall'estensione '`.acqua`'. Per la sua interpretazione viene specificato il programma '`/bin/acqua`'.

Si osservi che i file eseguibili, anche se di fatto devono essere soltanto interpretati (quindi richiedono solo la lettura), devono avere i permessi di esecuzione.

18.1.3.2 Realizzazione pratica

Non si può pensare che ogni volta che si vuole utilizzare un binario estraneo da interpretare, si debba dare il comando apposito, come negli esempi mostrati nella sezione precedente. Evidentemente, si tratta di inserire queste dichiarazioni in uno script della procedura di inizializzazione del sistema (in mancanza d'altro si potrebbe usare il solito '`rc.local`', se predisposto, contenuto nella directory '`/etc/rc.d/`' o '`/etc/init.d/`', oppure in altra simile).

Una volta definito un tipo di eseguibile da interpretare, nella directory '`/proc/sys/fs/binfmt_misc/`' viene creato un file virtuale con il nome corrispondente a quanto indicato nel primo campo del record di definizione. Se questo file viene sovrascritto con il valore `-1`, si ottiene l'eliminazione del tipo corrispondente. Se si fa la stessa cosa con il file '`status`', si elimina la gestione di tutti i binari specificati precedentemente. Seguono due esempi.

- `# echo -1 > /proc/sys/fs/binfmt_misc/Java` [*Invio*]

  Elimina la gestione del tipo di binario '**Java**'.

- `# echo -1 > /proc/sys/fs/binfmt_misc/status` [*Invio*]

  Elimina la gestione di tutti i tipi di binari da interpretare.

18.2 Scansione di parole

Capita spesso la necessità di scandire un elenco di parole allo scopo di eseguire uno o più comandi nell'ambito di tale scansione, utilizzando le parole stesse come argomenti dei comandi. Per fare un esempio di cosa si intende con questo, si pensi a un elenco di nomi di file, a partire dal quale si vuole ottenere l'esecuzione di un comando, una volta per ognuno di questi nomi. Questo tipo di scansione si esegue facilmente realizzando dei piccoli programmi, in forma di script, oppure si utilizza il programma '**xargs**' quando ciò che si intende fare non è troppo complesso.

18.2.1 Il ciclo «for» di una shell standard

Il comando '**for**' di una shell standard scandisce degli elementi, in corrispondenza dei quali esegue una lista di comandi.

```
for variabile in valore…
do
    lista_di_comandi
done
```

L'elenco di parole che segue '**in**' viene espanso, generando una lista di elementi. La variabile indicata dopo '**for**' viene posta ogni volta al valore di ciascun elemento di questa lista e la lista di comandi che segue '**do**' viene eseguita ogni volta di conseguenza.

Per esempio, il comando seguente aggiunge a ogni file l'estensione '`.txt`', senza nemmeno bisogno di realizzare uno script:

```
$ for a in * ; do mv $a $a.txt ; done [Invio]
```

Volendo vedere meglio questo esempio in uno script, basta trasformarlo nel modo seguente:

```
#!/bin/sh
for a in *
do
    mv $a $a.txt
done
```

L'elenco di parole che segue '**in**', può anche essere contenuto in una variabile di ambiente. Per esempio, si osservi lo script seguente, che in pratica svolge la stessa operazione già descritta, ma raccogliendo prima l'elenco dei nomi attraverso il comando '**ls**':

```
#!/bin/sh
elenco='ls'

for a in $elenco
do
    mv $a $a.txt
done
```

L'esempio seguente è più complesso, perché consente di cambiare l'estensione dei file, non solo di aggiungerne una. In particolare, si vuole sostituire l'estensione '`.testo`' con '`.txt`':

```
#!/bin/sh
for a in *.testo
do
    mv $a 'echo $a | sed s/\.testo$/\.txt/'
done
```

In pratica, '**sed**' riceve dallo standard input il nome del file, nel quale trova la stringa finale '**.testo**' e la sostituisce con '**.txt**' (SED è descritto nella sezione 23.5).

18.2.2 Scansione con «xargs»

L'utilità del programma di servizio '**xargs**'[1] non si comprende immediatamente. Volendo sintetizzare, '**xargs**' estrae dallo standard input delle «parole», intese come sequenze di caratteri e simboli separati da spazi o da interruzioni di riga, per utilizzarle come argomenti di un comando che viene avviato più volte, in base alla quantità di queste parole disponibili.

La situazione più banale è quella rappresentata dal comando seguente:

```
$ xargs echo [Invio]
```

Volendo ottenere lo stesso risultato con uno script di shell, si potrebbe tradurre nel codice seguente:

```
#!/bin/sh
ELENCO='cat'
RISULTATO=""
for a in $ELENCO
do
    RISULTATO="$RISULTATO $a"
done
echo $RISULTATO
```

In pratica, viene letto tutto lo standard input (salvo limiti di memoria); quanto inserito viene scandito e si aggiunge ogni elemento a una lista separata solo da uno spazio singolo; alla fine viene emessa questa lista attraverso lo standard output.

Si osservi che utilizzando '**xargs**' senza alcun argomento, si ottiene comunque questo risultato in modo predefinito.

```
$ xargs echo [Invio]
uno due tre quattro [Invio]
cinque sei [Invio]
sette [Invio]
[Ctrl d]

 uno due tre quattro cinque sei sette
```

Quello che si vede sopra è un esempio di come può comportarsi '**xargs**' usato in questo modo. Benché sia questo il comportamento predefinito, di solito occorre specificare in che modo '**xargs**' debba leggere lo standard input:

```
xargs [opzioni] [comando [argomenti_del_comando]]
```


L'opzione che definisce in che modo debba comportarsi '**xargs**' nei confronti dello standard input va scelta tra quelle descritte nell'elenco che segue. È importante comprendere che non serve a nulla indicare più di una di queste opzioni, dal momento che solo l'ultima viene presa in considerazione.

- `-I stringa_da_rimpiazzare`

  In questo modo, si stabilisce che deve essere presa in considerazione una riga alla volta di ciò che proviene dallo standard input; tuttavia, questa riga non viene fornita automaticamente come argomento finale del comando, al contrario va utilizzato un simbolo per collocare tali informazioni. Si osservi l'esempio seguente:

```
$ xargs -I ciao echo ciao ciao [Invio]
uno due tre quattro [Invio]

 uno due tre quattro uno due tre quattro
cinque sei [Invio]

 cinque sei cinque sei
sette [Invio]

 sette sette
[Ctrl d]
```

  Come si vede, viene stabilito che la stringa '**ciao**' serve a indicare in che punto deve essere collocato ciò che si estrae dallo standard input.

- `-L n_righe`

  Si stabilisce di utilizzare la quantità di righe non vuote indicata come argomento per ogni avvio del comando. Tuttavia, se una riga termina con uno spazio orizzontale (uno spazio vero e proprio o un carattere di tabulazione), si intende che questa non sia terminata e continui nella riga successiva. Si osservi l'esempio seguente che mostra proprio questo particolare:

```
$ xargs -L 2 echo [Invio]
uno due [Invio]
tre quattro cinque [Invio]

 uno due tre quattro cinque
sei sette [Invio]
otto [Invio]
nove dieci [Invio]

 sei sette otto nove dieci
undici [Invio]
[Ctrl d]

 undici
```

  Si può osservare che dopo la parola «sette» c'è uno spazio che crea una continuazione con la riga successiva.

- `--max-args=n_parole` | `-n n_parole`

  Si stabilisce di utilizzare la quantità di parole indicata come argomento per ogni avvio del comando. Si osservi l'esempio seguente:

```
$ xargs -n2 echo [Invio]
uno due tre [Invio]

 uno due
quattro cinque [Invio]

 tre quattro
[Ctrl d]

 cinque
```

Quelle che qui sono state chiamate «parole», ovvero le sequenze di caratteri che vengono prese in considerazione come elementi singoli, sono distinte in base all'uso di spazi, caratteri di tabulazione e interruzioni di riga. Queste parole possono anche essere delimitate tra apici singoli o doppi ('**'** e '**"**'), per esempio per includere degli spazi di qualche tipo; inoltre è possibile utilizzare il carattere '****' per confermare il valore letterale del carattere successivo.

Oltre all'opzione che specifica la modalità di lettura e scansione dello standard input se ne possono usare altre, tra le quali quelle descritte nel seguito.

- `-E` ***stringa_di_fine_file***

  In condizioni normali, la fine del file viene riconosciuta al termine dello stesso. Negli esempi mostrati in precedenza, veniva terminato l'inserimento dei dati attraverso la combinazione [*Ctrl d*], proprio in questo senso. Tuttavia, attraverso questa opzione, '**xargs**' riconosce una stringa particolare, individuata come parola isolata, con lo scopo di indicare la fine dei dati da prendere in considerazione. In mancanza dell'uso di questa opzione, potrebbe trattarsi in modo predefinito della stringa '_', ma ciò non è assicurato dallo standard. Pertanto, se si deve usare la stringa '_' senza il significato di terminazione che potrebbe avere, diventa necessario l'uso dell'opzione '**-E**' con qualcosa di diverso, eventualmente anche solo la stringa nulla: '**-E ""**'.

- `--interactive` | `-p`

  Con questa opzione, si fa in modo che '**xargs**' chieda conferma prima di eseguire qualsiasi comando (*prompt*). La conferma avviene inserendo la lettera '**y**' e premendo successivamente [*Invio*]. Qualunque altra cosa annulla il comando proposto.

- `--verbose` | `-t`

  Mostra i comandi prima di eseguirli.

- `--max-chars=`***n_max_caratteri*** | `-s` ***n_max_caratteri***

  Permette di stabilire un limite massimo per la lunghezza dei comandi generati da '**xargs**' (si include tutto il comando generato, non solo gli argomenti che gli vengono forniti).

- `--exit` | `-x`

  In presenza di un comando che eccede la quantità di caratteri massima predefinita, o fissata con l'opzione '**-s**', termina il funzionamento di '**xargs**'.

18.3 Scansione delle opzioni della riga di comando

Se si realizza uno script che deve essere richiamato fornendogli degli argomenti (dei parametri) sotto forma di opzioni, come si è abituati con i programmi di servizio comuni, può essere conveniente l'utilizzo di programmi o comandi appositi. Se si dispone di una shell POSIX, si può utilizzare il comando '**getopts**', come descritto nella sezione 17.3.4.1. Esiste anche un programma di servizio denominato '**getopt**' (senza la «s» finale), in grado di svolgere un compito equivalente (benché in modo differente), quando non ci si può avvalere del comando di shell. Va però osservato che il programma di servizio '**getopt**' non è previsto dallo standard POSIX. Quando nella documentazione di '**getopt**' si fa riferimento allo standard POSIX, ciò che può essere standard è il comportamento della funzione ***getopt()***, usata eventualmente da '**getopt**' stesso.

18.3.1 Programma «getopt»

Il programma di servizio '**getopt**' tradizionale ha la sintassi seguente:

```
getopt stringa_di_opzioni parametro...
```

La stringa di opzioni è un elenco di lettere che rappresentano le opzioni ammissibili; se ci sono opzioni che richiedono un argomento, le lettere corrispondenti di questa stringa devono essere seguite dal simbolo due punti ('**:**'). Gli argomenti successivi sono i valori dei parametri da analizzare. Lo scopo del programma è solo quello di controllare che tutto sia in ordine e di mettere a posto ciò che è possibile sistemare, emettendo l'elenco delle opzioni, nel modo «corretto». Per esempio:

```
$ getopt ab:c -a uno -b due -c tre quattro [Invio]
```

Potrebbe restituire il testo seguente:

```
 -a -b due -c -- uno tre quattro
```

Infatti, avendo utilizzato la definizione '**ab:c**', è stato stabilito che solo l'opzione '**-b**' ha un argomento, per cui, l'argomento '**uno**' è stato spostato alla fine delle opzioni, dopo il trattino doppio ('**--**').

Se il programma '**getopt**' di cui si dispone è aderente strettamente alle specifiche POSIX (in quanto si utilizza una funzione ***getopt()*** limitata alle specifiche dello standard POSIX), il risultato che si ottiene è diverso, dal momento che la scansione termina nel momento in cui si trova il primo argomento che non riguarda le opzioni:

```
 -a -- uno -b due -c tre quattro
```

L'esempio seguente dovrebbe chiarire in che modo si può utilizzare '**getopt**' per scandire gli argomenti della riga di comando:

```
#!/bin/sh
# scansione_1.sh

# Si raccoglie la stringa generata da getopt.
STRINGA_ARGOMENTI=‘getopt ab:c "$@"‘

# Si trasferisce nei parametri $1, $2,...
eval set -- "$STRINGA_ARGOMENTI"

while true ; do
    case "$1" in
        -a) echo "Opzione a"
            shift
            ;;
        -b) echo "Opzione b, argomento «$2»"
            shift 2
            ;;
        -c) echo "Opzione c"
            shift
            ;;
        --) shift
            break
            ;;
        *)  echo "Errore imprevisto!"
            exit 1
            ;;
        esac
done

echo "Argomenti rimanenti:"
for argomento in "$@"
do
    echo "$argomento"
done
```

In pratica, si comprende che lo scopo di '**getopt**' è solo quello di fare un po' di ordine tra le opzioni e di distinguere le opzioni dal resto. Supponendo che il nome dello script sia '**scansione_1.sh**', se si utilizza come nell'esempio già visto, si dovrebbe ottenere il risultato seguente:

```
$ ./scansione_1.sh -a uno -b due -c tre quattro [Invio]
```

```
Opzione a
Opzione b, argomento «due»
Opzione c
Argomenti rimanenti:
uno
tre
quattro
```

Se invece '**getopt**' si deve adeguare alla funzione ***getopt()*** dello standard POSIX, il risultato cambia come segue:

```
Opzione a
Argomenti rimanenti:
uno
-b
due
-c
tre
quattro
```

18.3.2 Il funzionamento di «getopt» nella versione dei programmi di servizio Linux

I programmi di servizio Linux si compongono anche di una versione di '**getopt**'[2] un po' più evoluta dello standard, ma ugualmente compatibile con le versioni «normali». È ammissibile l'uso della stessa sintassi vista nella sezione precedente e in particolare si può anche forzare l'aderenza alle specifiche POSIX definendo la variabile di ambiente ***POSIXLY_CORRECT***. Questa edizione di '**getopt**' è in grado di identificare anche le opzioni «lunghe». Oltre allo schema sintattico già visto, si può utilizzare in particolare quello seguente:

```
getopt [opzioni_di_getopt] -o|--options stringa_di_opzioni_corte ↩
↪      [opzioni_di_getopt] -- parametro_da_scandire...
```

In pratica, questa versione di '**getopt**' può avere delle opzioni proprie che ne regolano il funzionamento, tra le quali '**-o**' è obbligatoria, dal momento che il suo argomento è proprio la stringa che definisce quali opzioni possono essere presenti nei parametri. Eventualmente, per indicare opzioni lunghe, si utilizza l'opzione '**-l**'.

La stringa che definisce le opzioni corte, si comporta fondamentalmente come già spiegato nella sezione precedente. In particolare, se si usano due volte i due punti ('**::**'), si specifica che l'opzione ha un argomento facoltativo e non obbligatorio. La stringa che definisce le opzioni lunghe è simile a quella delle opzioni corte, con la differenza che, dovendo indicare dei nomi e non solo delle lettere singole, questi sono separati attraverso una virgola; per quanto riguarda l'uso dei due punti, la modalità è la stessa.

Questa versione di '**getopt**' ha anche la particolarità di essere in grado di proteggere gli argomenti che ne hanno bisogno, ma per arrivare a questo deve sapere con quale shell si sta operando. Infatti, dal momento che '**getopt**' restituisce una stringa che poi deve essere scandita nuovamente, se un argomento contiene caratteri particolari che richiedono una qualche forma di protezione (come gli spazi), è necessario che venga fatta una trasformazione opportuna, la quale non può essere unica per tutte le situazioni. In condizioni normali, il risultato che si ottiene è adatto per Bash, altrimenti occorre utilizzare l'opzione '**-s**'.

Tabella 18.21. Alcune opzioni.

Opzione	Descrizione
`-o` ***stringa_di_opzioni_corte*** `--options` ***stringa_di_opzioni_corte***	Definisce le opzioni normali che devono essere cercate tra i parametri, specificando anche se queste hanno un argomento, obbligatorio o facoltativo.
`-l` ***stringa_di_opzioni_lunghe*** `--longoptions` ↩ ↪***stringa_di_opzioni_lunghe***	Definisce le opzioni lunghe che devono essere cercate tra i parametri, specificando anche se queste hanno un argomento, obbligatorio o facoltativo.

Opzione	Descrizione
`-s` {`sh`|`bash`|`csh`|`tcsh`} `--shell` {`sh`|`bash`|`csh`|`tcsh`}	Definisce il tipo di shell che si sta utilizzando, permettendo di definire il modo migliore per proteggere i caratteri che richiedono questo tipo di accortezza.
`-n` ***nome_del_programma*** `--name` ***nome_del_programma***	Dal momento che '**getopt**' può segnalare gli errori, con questa opzione è possibile definire il nome del programma al quale attribuire l'errore generato.

Come esempio viene mostrata una variante dello script proposto nella sezione precedente, dove si scandiscono anche le opzioni lunghe e l'ultima ha un argomento facoltativo.

```
#!/bin/sh
# scansione_1.sh

# Si raccoglie la stringa generata da getopt.
STRINGA_ARGOMENTI='getopt -o ab:c:: -l a-lunga,b-lunga:,c-lunga:: -- "$@"'

# Si trasferisce nei parametri $1, $2,...
eval set -- "$STRINGA_ARGOMENTI"

while true ; do
    case "$1" in
        -a|--a-lunga)
            echo "Opzione a"
            shift
            ;;
        -b|--b-lunga)
            echo "Opzione b, argomento «$2»"
            shift 2
            ;;
        -c|--c-lunga)
            case "$2" in
                "") echo "Opzione c, senza argomenti"
                    shift 2
                    ;;
                *)  echo "Opzione c, argomento «$2»"
                    shift 2
                    ;;
            esac
            ;;
        --) shift
            break
            ;;
        *)  echo "Errore imprevisto!"
            exit 1
            ;;
        esac
done

echo "Argomenti rimanenti:"
for argomento in "$@"
do
    echo "$argomento"
done
```

Supponendo che il nome dello script sia '**scansione_2.sh**', se si utilizza come nell'esempio seguente,

```
$ ./scansione_2.sh -auno -bdue -ctre quattro [Invio]
```

oppure

```
$ ./scansione_2.sh --a-lunga=uno --b-lunga=due ↩
↪                  --c-lunga=tre quattro [Invio]
```

si dovrebbe ottenere il risultato seguente:

```
Opzione a
Opzione b, argomento «due»
Opzione c, argomento «tre»
Argomenti rimanenti:
uno
quattro
```

Tuttavia, se utilizzando le opzioni corte, gli argomenti di queste non vengono attaccati alle lettere rispettive, come nell'esempio seguente,

```
$ ./scansione_2.sh -a uno -b due -c tre quattro [Invio]
```

gli argomenti facoltativi non vengono presi in considerazione:

```
Opzione a
Opzione b, argomento «due»
Opzione c, senza argomenti
Argomenti rimanenti:
uno
tre
quattro
```

18.4 File temporanei

Quando si realizzano degli script, si ha spesso la necessità di realizzare dei file temporanei, magari solo per accumulare il risultato di un'elaborazione senza tentare di fare altri tipi di acrobazie. Il programma di servizio che si usa per queste cose è '**tempfile**':[3]

```
tempfile [opzioni]
```

Nella maggior parte dei casi, '**tempfile**' viene usato senza argomenti, ottenendo la creazione di un file vuoto nella directory temporanea ('/tmp/'), con permessi normali (lettura e scrittura per tutti, meno quanto filtrato dalla maschera dei permessi), ottenendo il percorso assoluto di questo file dallo standard output.

Tabella 18.25. Alcune opzioni.

Opzione	Descrizione
`-d` ***directory*** `--directory` ***directory***	Se non si vuole usare la directory temporanea standard, si può specificare la directory di destinazione del file temporaneo con questa opzione.
`-m` ***modalità_dei_permessi*** `--mode` ***modalità_dei_permessi***	Se si vuole evitare che il file temporaneo che viene creato abbia dei permessi di accesso troppo ampi, si può utilizzare questa opzione per stabilire qualcosa di diverso.

Segue la descrizione di alcuni esempi.

- `$ tempfile` [*Invio*]

  Crea un file temporaneo e ne restituisce il nome attraverso lo standard output.

- ```
  #!/bin/sh
  TEMPORANEO='tempfile'
  ls -l / > $TEMPORANEO
  …
  rm -r $TEMPORANEO
  ```

  Quello che si vede è l'esempio tipico di uno script, incompleto, in cui si crea un file temporaneo accumulandone il nome in una variabile di ambiente; quindi si fa qualcosa con quel file (in questo caso si inserisce il risultato del comando '**ls -l**'), infine si elimina il file, sempre utilizzando l'espansione della variabile che ne contiene il nome.

18.5 Ambiente

In situazioni determinate, può essere importante avviare un programma, o un altro script con un insieme di variabili di ambiente diverso da quello che si erediterebbe normalmente. Per questo si può usare il programma di servizio '**env**':[4]

```
env [opzioni] [nome=valore]… [comando [argomenti_del_comando]]
```

Come si può intuire, le opzioni di '**env**' servono a eliminare o ad aggiungere delle variabili di ambiente, senza interferire con l'ambiente dello script.

Tabella 18.27. Alcune opzioni.

Opzione	Descrizione
`-u` ***variabile*** `--unset=`***variabile***	Permette di eliminare la variabile di ambiente nominata.

Opzione	Descrizione

`-`
`-i`
`--ignore-environment`
	Azzera completamente tutto l'ambiente.

A titolo di esempio, si supponga di avere due script: nel primo viene dichiarata la variabile di ambiente ***CIAO*** e viene chiamato il secondo eliminando questa variabile dall'ambiente; il secondo script si limita a mostrare il contenuto di questa variabile, se è disponibile.

```
#!/bin/sh
# ./primo.sh
CIAO="ciao a tutti"
export CIAO
env -u CIAO ./secondo.sh
echo $CIAO
```

```
#!/bin/sh
# ./secondo.sh
echo $CIAO
```

Il risultato è che funziona solo la visualizzazione della variabile che avviene con il comando '**echo**' del primo script, perché nel secondo non è disponibile. Sarebbe stato diverso se il primo e unico script fosse stato quello seguente:

```
#!/bin/sh
# ./primo.sh
CIAO="ciao a tutti"
export CIAO
env -u CIAO echo $CIAO
echo $CIAO
```

In questo caso, anche se il comando '**echo**' viene avviato senza la disponibilità della variabile ***CIAO***, si otterrebbe ugualmente la sua visualizzazione, dal momento che l'espansione della stessa avviene prima della chiamata del programma '**env**'.

18.6 Interazione con l'utente

Spesso, la realizzazione di uno script di shell interattivo, è molto difficile; o meglio, è difficile realizzare qualcosa di pratico da usare. La shell offre il comando interno '**read**', per leggere ciò che viene inserito attraverso la tastiera, ma questo permette di ottenere un'interazione molto banale, a livello di riga di comando. In alternativa si possono usare dei programmi realizzati appositamente per abbellire gli script, come nel caso di '**dialog**'.

18.6.1 Utilizzo di «echo» e di «printf»

Il programma di servizio '**echo**'[5] emette le stringhe indicate come argomento, separate da uno spazio e con l'aggiunta di un codice di interruzione di riga finale. Per usare '**echo**' nel modo più compatibile possibile, occorre limitarsi allo schema sintattico seguente, conforme allo standard, dove non è ammesso l'uso di alcuna opzione:

```
echo [stringa…]
```

Il programma potrebbe riconoscere alcune sequenze di escape, utili per comporre il testo da visualizzare. Tuttavia, la versione GNU del programma e il comando omonimo della shell Bash richiedono un'opzione per conformarsi a tali sequenze speciali. Pertanto, per poter usare il comando o il programma '**echo**' in modo uniforme tra i vari sistemi Unix, è necessario evitare le sequenze di escape e le opzioni.

Tabella 18.31. Elenco delle sequenze di escape riconoscibili da '**echo**' e da '**printf**'.

Codice	Descrizione
`\\`	Inserisce la barra obliqua inversa ('\').
```

Codice	Descrizione
`\a`	Inserisce il codice <*BEL*> (avvisatore acustico).
`\b`	Inserisce il codice <*BS*> (*backspace*).
`\c`	Alla fine di una stringa previene l'inserimento di una nuova riga.
`\f`	Inserisce il codice <*FF*> (*formfeed*).
`\n`	Inserisce il codice <*LF*> (*linefeed*).
`\r`	Inserisce il codice <*CR*> (*carriage return*).
`\t`	Inserisce una tabulazione normale (<*HT*>).
`\v`	Inserisce una tabulazione verticale (<*VT*>).
`\0`***n***	Inserisce il carattere corrispondente al codice ottale ***n***.

Il testo visualizzato dal comando o dal programma '**echo**' è concluso normalmente da un codice di interruzione di riga. Un'estensione diffusa del programma '**echo**' consiste nella disponibilità dell'opzione '**-n**' (non standard), con cui si sopprime l'aggiunta di tale codice finale, consentendo di mantenere il cursore alla fine del testo visualizzato. Per esempio così:

```
$ echo -n "ciao " ; echo "a tutti" [Invio]

ciao a tutti
```

Per ovviare alle carenze di '**echo**', occorre scegliere piuttosto il programma '**printf**',[6] il quale emette attraverso lo standard output la stringa di composizione fornita, utilizzando gli argomenti, con regole analoghe a quelle della funzione ***printf()*** del linguaggio C:

```
printf composizione [argomento...]
```

Nella stringa di composizione, vanno usate opportunamente le sequenze di escape della tabella 18.31. Per esempio, per mandare a capo il testo dopo la visualizzazione della stringa, occorre concludere con la sequenza '**\n**' (cosa che per '**echo**' è invece implicita. Inoltre, se dopo la stringa di composizione ci sono degli argomenti, si possono inserire degli ***specificatori di conversione***, caratterizzati dal fatto che iniziano con il simbolo di percentuale ('**%**'), dove ogni specificatore serve a rappresentare un argomento. Il modo in cui si esprime uno specificatore di conversione può essere complesso, pertanto viene mostrato un modello sintattico che descrive la sua struttura, limitatamente alle necessità del programma '**printf**':

```
%[simbolo][n_ampiezza][.n_precisione]tipo
```

La prima cosa da individuare in uno specificatore di conversione è il tipo di argomento che viene interpretato e, di conseguenza, il genere di rappresentazione che se ne vuole produrre. Il tipo viene espresso da una lettera alfabetica, alla fine dello specificatore di conversione. La tabella successiva riepiloga i tipi principali che dovrebbero essere accettabili in ogni realizzazione di '**printf**'.

Tabella 18.33. Tipi di conversione principali.

Simbolo	Tipo di argomento	Conversione applicata
`%...d` `%...i`	`int`	Numero intero con segno da rappresentare in base dieci.
`%...u`	`unsigned int`	Numero intero senza segno da rappresentare in base dieci.
`%...o`	`unsigned int`	Numero intero senza segno da rappresentare in ottale (senza lo zero iniziale che viene usato spesso per caratterizzare un tale tipo di rappresentazione).

Simbolo	Tipo di argomento	Conversione applicata
`%...x` `%...X`	`unsigned int`	Numero intero senza segno da rappresentare in esadecimale (senza il prefisso '**0x**' o '**0X**' che viene usato spesso per caratterizzare un tale tipo di rappresentazione).
`%...c`	`int`	Un carattere singolo, dopo la conversione in '**unsigned char**'.
`%...s`	`char *`	Una stringa.
`%...f`	`double`	Un numero a virgola mobile, da rappresentare in notazione decimale fissa: **[-]*iii.dddddd***
`%...e` `%...E`	`double`	Un numero a virgola mobile, da rappresentare in notazione esponenziale: **[-]*i.ddddde*±*xx*** **[-]*i.ddddd*E±*xx***
`%...g` `%...G`	`double`	Un numero a virgola mobile, rappresentato in notazione decimale fissa o in notazione esponenziale, a seconda di quale si presti meglio in base ai vincoli posti da altri componenti dello specificatore di conversione.
`%%`		Questo specificatore si limita a produrre un carattere di percentuale ('**%**') che altrimenti non sarebbe rappresentabile.

Nel modello sintattico che descrive lo specificatore di conversione, si vede che subito dopo il segno di percentuale può apparire un simbolo (*flag*). I simboli principali che possono essere utilizzati sono descritti nella tabella successiva.

Tabella 18.34. Alcuni simboli, o *flag*.

Simbolo	Corrispondenza

`%+...`
`%#+...`
`%+0`***ampiezza***`...`
`%#+0`***ampiezza***`...`
	Il segno «+» fa sì che i numeri con segno lo mostrino anche se è positivo. Può combinarsi con lo zero e il cancelletto.

`%0`***ampiezza***`...`
`%+0`***ampiezza***`...`
`%#0`***ampiezza***`...`
`%#+0`***ampiezza***`...`
	Lo zero fa sì che siano inseriti degli zeri a sinistra per allineare a destra il valore, nell'ambito dell'ampiezza specificata. Può combinarsi con il segno «+» e il cancelletto.

`%`***ampiezza***`...` `% `***ampiezza***`...`	In mancanza di uno zero iniziale, in presenza dell'indicazione dell'ampiezza, il valore viene allineato a destra usando degli spazi. È possibile esprimere esplicitamente l'intenzione di usare gli spazi mettendo proprio uno spazio, ma in generale non è richiesto. Se si mette lo spazio letteralmente, questo non è poi compatibile con lo zero, mentre le combinazioni con gli altri simboli sono ammissibili.

`%-`***ampiezza***`...`
`%-+`***ampiezza***`...`
`%#-`***ampiezza***`...`
`%#-+`***ampiezza***`...`
	Il segno meno, usato quando la conversione prevede l'uso di una quantità fissa di caratteri con un valore che appare di norma allineato a destra, fa sì che il risultato sia allineato a sinistra. Il segno meno si può combinare il segno «+» e il cancelletto.

Simbolo	Corrispondenza
%#...	Il cancelletto richiede una modalità di rappresentazione alternativa, ammesso che questa sia prevista per il tipo di conversione specificato. È compatibili con gli altri simboli, ammesso che il suo utilizzo serva effettivamente per ottenere una rappresentazione alternativa.

Tra il simbolo (*flag*) e il tipo può apparire un numero che rappresenta l'ampiezza da usare nella trasformazione ed eventualmente la precisione: '***ampiezza*[*.precisione*]**'. Il concetto parte dalla rappresentazione dei valori in virgola mobile, dove l'ampiezza indica la quantità complessiva di caratteri da usare e la precisione indica quanti di quei caratteri usare per il punto decimale e le cifre successive, ma si applica anche alle stringhe.

In generale, per quanto riguarda la rappresentazione di valori numerici, la parte intera viene sempre espressa in modo completo, anche se l'ampiezza indicata è inferiore; ai numeri interi la precisione non si applica; per i numeri in virgola mobile con rappresentazione esponenziale, la precisione riguarda le cifre decimali che precedono l'esponente; per le stringhe la precisione specifica la quantità di caratteri da considerare, troncando il resto.

Segue la descrizione di alcuni esempi.

- `$ printf "ciao " ; printf "a tutti\n"` [*Invio*]

  Questo esempio serve a dimostrare che '**`printf`**' non manda a capo il cursore, alla fine della visualizzazione, se non richiesto espressamente con la sequenza '**`\n`**'.

```
ciao a tutti
```

- `$ printf "%02x %02x %02x %02x\n" 192 168 1 71` [*Invio*]

  Converte in esadecimale, i valori forniti come argomento, facendo in modo che ogni numero occupi esattamente due cifre.

```
c0 a8 01 fe
```

- `$ printf "%03o %03o %03o %03o\n" 192 168 1 71` [*Invio*]

  Converte in ottale, i valori forniti come argomento, facendo in modo che ogni numero ottale occupi esattamente tre cifre.

```
300 250 001 376
```

Tabella 18.38. Esempi di utilizzo degli specificatori di conversione di '**`printf`**'.

Codice	Risultato emesso attraverso la funzione
`printf "[%i]" 123`	`[123]`
`printf "[%i]" -123`	`[-123]`
`printf "[%2d]" 123`	`[123]` L'indicatore '**`%2d`**' specifica che si devono usare almeno due cifre, ma se le cifre della parte intera sono in numero maggiore, queste vanno indicate tutte ugualmente.
`printf "[%6d]" 123`	`[ 123]`
`printf "[%6d]" -123`	`[ -123]`
`printf "[%+6d]" 123`	`[ +123]`
`printf "[%06d]" 123`	`[000123]`
`printf "[%06d]" -123`	`[-00123]`
`printf "[%+06d]" 123`	`[+00123]`
`printf "[%-6d]" 123`	`[123 ]`
`printf "[%u]" 123`	`[123]`

Codice	Risultato emesso attraverso la funzione
`printf "[%u]" -123`	`[18446744073709551493]` Evidentemente si ottiene la rappresentazione del valore binario, tale e quale, secondo la notazione usata per i valori negativi.
`printf "[%6x]" 123`	`[ 7b]`
`printf "[%06x]" 123`	`[00007b]`
`printf "[%x]" 123`	`[7b]`
`printf "[%x]" -123`	`[ffffffffffffff85]` Evidentemente si ottiene la rappresentazione del valore binario, tale e quale, secondo la notazione usata per i valori negativi.
`printf "[%-6x]" 123`	`[7b ]`
`printf "[%-06x]" 123`	`[7b ]`
`printf "[%o]" 123`	`[173]`
`printf "[%o]" -123`	`[1777777777777777777605]` Evidentemente si ottiene la rappresentazione del valore binario, tale e quale, secondo la notazione usata per i valori negativi.
`printf "[%6o]" 123`	`[ 173]`
`printf "[%06o]" 123`	`[000173]`
`printf "[%f]" 123.456`	`[123.456000]`
`printf "[%f]" -123.456`	`[-123.456000]`
`printf "[%12f]" 123.456`	`[ 123.456000]`
`printf "[%.4f]" 123.456`	`[123.4560]`
`printf "[%12.4f]" 123.456`	`[ 123.4560]`
`printf "[%12.4f]" -123.456`	`[ -123.4560]`
`printf "[%+12.4f]" 123.456`	`[ +123.4560]`
`printf "[%012.4f]" 123.456`	`[0000123.4560]`
`printf "[%012.4f]" -123.456`	`[-000123.4560]`
`printf "[%+012.4f]" 123.456`	`[+000123.4560]`
`printf "[%-12.4f]" 123.456`	`[123.4560 ]`
`printf "[%e]" 123.456`	`[1.234560e+02]`
`printf "[%e]" -123.456`	`[-1.234560e+02]`
`printf "[%15e]" 123.456`	`[ 1.234560e+02]`
`printf "[%.4e]" 123.456`	`[1.2346e+02]`
`printf "[%15.4e]" 123.456`	`[ 1.2346e+02]`
`printf "[%15.4e]" -123.456`	`[ -1.2346e+02]`
`printf "[%+15.4e]" 123.456`	`[ +1.2346e+02]`
`printf "[%015.4e]" 123.456`	`[000001.2346e+02]`
`printf "[%015.4e]" -123.456`	`[-00001.2346e+02]`
`printf "[%+015.4e]" 123.456`	`[+00001.2346e+02]`
`printf "[%-15.4e]" 123.456`	`[1.2346e+02 ]`
`printf "[%s]" "ciao amore"`	`[ciao amore]`
`printf "[%7s]" "ciao amore"`	`[ciao amore]` La stringa è più lunga di sette caratteri, ma viene visualizzata completamente.
`printf "[%.7s]" "ciao amore"`	`[ciao am]` La stringa viene troncata se è più lunga del valore della precisione.
`printf "[%.14s]" "ciao amore"`	`[ciao amore]`
`printf "[%14s]" "ciao amore"`	`[ ciao amore]`
`printf "[%14.7s]" "ciao amore"`	`[ ciao am]`
`printf "[%-14s]" "ciao amore"`	`[ciao amore ]`
`printf "[%-14.7s]" "ciao amore"`	`[ciao am ]`

18.6.2 Utilizzo di «read»

Di norma, '**read**' è un comando interno delle shell POSIX, anche se potrebbe essere disponibile un programma di servizio equivalente, da utilizzare con una shell differente. Il modello sintattico seguente rappresenta una semplificazione che dovrebbe essere compatibile in generale con le shell POSIX:

```
read [-p invito] [variabile...]
```

Il comando '**read**' potrebbe essere utilizzato da solo, senza argomenti; in questo caso servirebbe soltanto per attendere la pressione del tasto [*Invio*], permettendo all'utente di leggere un'informazione che appare sullo schermo, prima di proseguire con altre operazioni.

L'opzione '**-p**' dovrebbe essere abbastanza chiara: permette di definire una stringa di invito all'inserimento di qualcosa. Infine, i nomi che vengono collocati in coda alla riga di comando, rappresentano altrettante variabili di ambiente che vengono create appositamente, assegnando loro le parole inserite attraverso '**read**'; in particolare, l'ultima variabile dell'elenco raccoglie tutte le parole rimanenti.

```
#!/bin/sh

printf "Inserisci una frase: "
read UNO DUE TRE
printf "La prima parola inserita è «$UNO»\n"
printf "La seconda parola inserita è «$DUE»\n"
printf "Il resto della frase è «$TRE»\n"
```

L'esempio dovrebbe permettere di capire il funzionamento di '**read**'. Si osservi in particolare il fatto che l'invito viene ottenuto attraverso il comando '**printf**', senza indicare alla fine il codice di interruzione di riga. Supponendo che si tratti dello script '**read.sh**':

```
$ ./read.sh [Invio]

Inserisci una frase: ciao come stai? io sto bene [Invio]

  La prima parola inserita è «ciao»
  La seconda parola inserita è «come»
  Il resto della frase è «stai? io sto bene»
```

18.6.3 Utilizzo di «select»

La shell Korn e la shell Bash offrono una struttura di controllo particolare, utile per la selezione interattiva di un elemento da un elenco. Si tratta di '**select**', la cui sintassi si riassume nello schema sintattico seguente:

```
select variabile [in valore...]
do
    lista_di_comandi
done
```

L'elenco di parole che segue '**in**' viene espanso, generando una lista di elementi. L'insieme delle parole espanse viene emesso attraverso lo standard error, ognuna preceduta da un numero. Se '**in**' (e i suoi argomenti) viene omesso, vengono utilizzati i parametri posizionali. In pratica è come se venisse usato '**in $@**'.

Dopo l'emissione dell'elenco, viene mostrato l'invito contenuto nella variabile ***PS3*** e viene letta una riga dallo standard input. Se la riga consiste del numero corrispondente a una delle parole mostrate, allora viene assegnato alla variabile indicata dopo '**select**' la parola corrispondente. Se la riga è vuota (probabilmente è stato premuto soltanto [*Invio*]), l'elenco e l'invito vengono emessi nuovamente. Se viene letto il codice corrispondente a EOF ([*Ctrl d*]), il comando termina. Qualsiasi altro valore letto fa sì che la variabile sia posta al valore della stringa nulla. La riga letta viene salvata nella variabile ***REPLY***. La lista di comandi che segue '**do**' viene eseguita dopo ciascuna selezione fino a che non viene incontrato un comando '**break**' o '**return**'.

Il valore restituito da '**select**' è quello dell'ultimo comando eseguito all'interno della lista '**do**', oppure zero se nessun comando è stato eseguito.

Viene mostrato nuovamente lo stesso esempio già presentato in occasione della descrizione di '**select**' fatta nell'ambito dei capitoli dedicati a Bash: fa apparire un menù composto dagli argomenti fornitigli; a ogni selezione mostra quello scelto.

```
#!/bin/sh
select i in $*
do
    echo "hai selezionato $i premendo $REPLY"
    echo ""
    echo "premi Ctrl+c per terminare"
done
```

L'esempio seguente proviene dagli script di nanoLinux II 1998 e rappresenta la selezione del nome di un'interfaccia di rete, il quale viene accumulato nella variabile di ambiente ***INTERFACCIA***:

```
echo "Selezionare l'interfaccia."
select i in eth0 eth1 eth2 plip0 plip1 plip2
do
    INTERFACCIA=$i
    break
done
```

18.6.4 Utilizzo di «yes»

Il programma di servizio '**yes**'[7] emette ripetitivamente senza fine, attraverso lo standard output, le stringhe indicate come argomento (separate da uno spazio l'una dall'altra), seguite dal codice di interruzione di riga.

```
yes [stringa...]
```

Se non viene indicata alcuna stringa come argomento, emette la lettera «y». Il programma continua la sua esecuzione fino a che non viene interrotto. Segue la descrizione di alcuni esempi.

- `$ yes` [*Invio*]

  ```
  y
  y
  y
  y
  y
  ...
  ```

  Senza argomenti, '**yes**' emette una serie indefinita di lettere «y» seguite dal codice di interruzione di riga.

- `$ yes n` [*Invio*]

  ```
  n
  n
  n
  n
  n
  ...
  ```

  Se vengono specificate delle stringhe come argomento, queste stringhe vengono emesse ripetitivamente.

- `$ yes | mio_prog` [*Invio*]

  Si invia una serie di lettere «y», seguite dal codice di interruzione di riga, al programma ipotetico '**mio_prog**' che probabilmente tende a fare delle domande alle quali si vuole rispondere sempre con una lettera «y».

18.6.5 Dialog e simili

Dialog e altri programmi più o meno compatibili, hanno lo scopo di gestire effetti più appariscenti in uno script di shell, interagendo con l'utilizzatore attraverso schermate colorate e finestre di dialogo, le quali, a seconda dei casi sono adatte allo schermo a caratteri, oppure richiedono la grafica.[8] [9] [10] [11]

```
dialog [opzioni_generali] [definizione_del_tipo_di_interazione]
```

```
whiptail [opzioni_generali] [definizione_del_tipo_di_interazione]
```

```
Xdialog [opzioni_generali] [definizione_del_tipo_di_interazione]
```

```
gdialog [opzioni_generali] [definizione_del_tipo_di_interazione]
```

Si può intuire che il programma «standard» sia Dialog, essendo fatto per i terminali a caratteri, utilizzando la libreria Ncurses.[12] Il programma Whiptail, è una rivisitazione, fatta sempre per i terminali senza grafica, ma usa la libreria Newt.[13] Gli altri programmi si usano con la grafica.

La riga di comando distingue due tipi di opzioni: quelle che hanno valore in senso generale influenzando il comportamento del programma e quelle che definiscono un tipo di interazione con l'utilizzatore. Nella documentazione originale, queste ultime sono definite *box-options*, perché si riferiscono ai riquadri che vengono mostrati sullo schermo. Evidentemente, si può utilizzare al massimo una sola opzione che definisca una finestra di dialogo.

Dovendo definire delle finestre su uno schermo a caratteri, le opzioni che permettono di descriverle, fanno riferimento a delle dimensioni in caratteri. Questi valori non possono essere omessi e in caso si voglia fare riferimento alle dimensioni ottimali, in base alla disponibilità dello schermo, basta indicare il valore zero, tenendo conto però che questa possibilità non funziona sempre.

La documentazione di Dialog in particolare è accompagnata da esempi di script più completi di quelli che si vedono qui. Vale la pena di studiarli per apprendere bene il funzionamento di questi programmi. In generale, dovrebbero trovarsi a partire dalla directory '`/usr/share/doc/dialog/`'.

Tabella 18.45. Alcune opzioni generali.

Opzione	Descrizione
`--clear`	Se si utilizza questa opzione generale, si fa in modo di ripulire lo schermo prima di mostrare il riquadro della finestra di dialogo. Nelle versioni grafiche del programma, viene ignorata.
`--title` ***titolo_finestra***	Permette di dare un titolo alla finestra di dialogo.
`--backtitle` ***sottotitolo_finestra***	Permette di dare un titolo allo sfondo, facendolo apparire nella parte superiore dello schermo, al di fuori della finestra di dialogo relativa. Alcune realizzazioni del programma, ignorano questa opzione.
`--separate-output`	Questa opzione altera il modo in cui viene emesso il risultato di un'interazione. Per la precisione serve quando si utilizza una finestra di dialogo contenente una lista di caselline da barrare. Si veda a questo proposito l'opzione '**`--checklist`**'.
`--fb`	Questa opzione riguarda esclusivamente Whiptail e consente di visualizzare pulsanti grafici più appariscenti del normale.

Tabella 18.46. Alcune opzioni per la definizione della finestra di dialogo.

Opzione	Descrizione
`--yesno` ***testo altezza larghezza***	Fa apparire una finestra di dialogo molto semplice, in cui viene mostrato il testo indicato, al quale si deve rispondere con un «sì», oppure con un «no», rappresentati da due pulsanti grafici: YES e NO. Se la risposta è «sì», viene restituito *Vero* (il valore zero), altrimenti si ottiene *Falso* (un valore diverso da zero).
`--msgbox` ***testo altezza larghezza***	La finestra di dialogo che si ottiene, serve a mostrare un messaggio, per il quale si attende la conferma da parte dell'utilizzatore. Alla base della finestra appare il pulsante grafico OK, selezionando il quale si conclude il funzionamento del programma.
`--infobox` ***testo altezza larghezza***	In questo caso, più che di una finestra di dialogo, si tratta di una finestra contenente un messaggio, per il quale non viene attesa alcuna azione da parte dell'utente. In pratica, il programma mostra il messaggio e termina immediatamente di funzionare. Può essere paragonato a un comando '**`echo`**', molto più appariscente.
`--inputbox` ***testo*** ↩ ↪***altezza larghezza*** ↩ ↪[***risposta_predefinita***]	Questa finestra di dialogo permette all'utilizzatore di inserire un testo libero, dove eventualmente è possibile mostrare inizialmente una risposta predefinita. Alla base della finestra appaiono i pulsanti grafici OK e CANCEL. Se si seleziona OK, si conferma il testo inserito, il quale viene emesso attraverso lo standard output; altrimenti, con CANCEL, non si ottiene alcun risultato.
`--passwordbox` ***testo*** ↩ ↪***altezza larghezza*** ↩ ↪[***risposta_predefinita***]	Questa finestra di dialogo è analoga a quella che si ottiene con '**`--inputbox`**', con la differenza che non si vede quanto digitato dall'utente. Funziona solo con Dialog e Whiptail.
`--password inputbox` ***testo*** ↩ ↪***altezza larghezza***	Questa finestra di dialogo è analoga a quella che si ottiene con '**`--inputbox`**', con la differenza che non si vede quanto digitato dall'utente. Funziona solo con Xdialog e Gdialog.

Opzione	Descrizione
`--textbox` ***file altezza larghezza***	Questa finestra di dialogo serve a permettere la visualizzazione di un file di testo. L'utilizzatore può usare intuitivamente i tasti [*pagina-su*], [*pagina-giù*] e i tasti freccia, anche per degli spostamenti orizzontali. Alla base della finestra si vede il pulsante grafico EXIT, che permette di concludere la visualizzazione. È il caso di annotare un problema relativo agli stop di tabulazione. Se il testo da mostrare ne contiene, è molto probabile che la visualizzazione di questo avvenga in modo disallineato rispetto alla realtà. Ciò dipende dal fatto che la visualizzazione avviene all'interno di un'area incorniciata, per cui gli stop di tabulazione si trovano spostati rispetto alla loro posizione originale. In questi casi, converrebbe rielaborare il file da visualizzare attraverso il programma '**expand**', prima di passare alla visualizzazione.
`--menu` ***testo*** ↩ ↪***altezza larghezza altezza_menù*** ↩ ↪[***elemento descrizione***]…	Questo tipo di finestra di dialogo comincia a essere un po' più complicato. Il suo scopo è quello di mostrare un menù, composto da coppie di valori, dove il primo è ciò che viene restituito attraverso lo standard output nel caso di selezione e il secondo è la sua descrizione. Il menù, ovvero l'elenco di queste voci, può avere un'altezza determinata, ma anche in questo caso si può stabilire una larghezza predefinita utilizzando semplicemente lo zero. Sulle voci del menù appare un cursore in forma di barra di scorrimento, la quale può essere spostata con i tasti freccia o i tasti pagina, mentre alla base della finestra appaiono i pulsanti grafici OK e CANCEL. Selezionando OK, il programma termina emettendo la stringa corrispondente all'elemento che si trova evidenziato dalla barra di scorrimento; selezionando CANCEL non si ottiene alcun risultato.

Opzione	Descrizione
`--checklist` ***testo*** ↩ ↪***altezza larghezza altezza_menù*** ↩ ↪[***elemento descrizione*** `on`|`off`]…	Questo tipo di finestra di dialogo è simile a quella che si ottiene con l'opzione '**`--menu`**'. La differenza fondamentale sta nel fatto che in questo caso è possibile selezionare più voci, attraverso delle caselle di selezione: anche qui c'è una barra di scorrimento e quando ci si trova sopra la voce desiderata, la [*barra-spaziatrice*] mette o toglie il segno di selezione. A differenza dell'opzione '**`--menu`**', le voci del menù possono essere già attivate o meno, pertanto si aggiunge la parola chiave '**`on`**' oppure '**`off`**'. La particolarità di questo tipo di selezione, richiede attenzione nel modo in cui deve essere interpretato il risultato emesso attraverso lo standard output. Infatti, in condizioni normali, vengono restituite le stringhe corrispondenti alle voci di menù selezionate, delimitate tra apici doppi, o in altro modo, a seconda del programma usato. Se questo sistema crea difficoltà, si può abbinare l'uso dell'opzione '**`--separate-output`**' perché queste stringhe siano separate dal codice di interruzione di riga, senza l'uso di delimitatori di altro tipo.
`--radiolist` ***testo*** ↩ ↪***altezza larghezza altezza_menù*** ↩ ↪[***elemento descrizione*** `on`|`off`]…	Questo tipo di finestra di dialogo si comporta in modo simile a quella ottenuta con l'opzione '**`--checklist`**'. La differenza sta nel fatto che si può selezionare solo una voce dall'elenco, per cui il risultato non comporta difficoltà nell'interpretazione. Evidentemente, si può preselezionare solo una delle voci del menù.

Gli esempi seguenti mostrano diverse situazioni, dove si mette anche a confronto il risultato che si ottiene tra i vari programmi.

```
#/bin/sh

if dialog --title "Domanda" --yesno "Ti piace Dialog?" 0 0
then
    echo "Ottimo!"
else
    echo "Peccato :-)"
fi
```

In questo script, viene mostrata la finestra di dialogo che si vede nella figura 18.48; in base alla scelta affermativa o negativa, si ottiene la visualizzazione di un messaggio differente.

Figura 18.48. Esempio del funzionamento della finestra di dialogo ottenuta con l'opzione '**`--yesno`**' con Dialog.

```
.------- Domanda --------.
|   Ti piace Dialog?    |
|-----------------------|
| < Yes >      <  No  > |
'-----------------------'
```

Figura 18.49. Esempio del funzionamento della finestra di dialogo ottenuta con l'opzione '**--yesno**' con Xdialog e Gdialog, rispettivamente.

Lo script successivo mostrata la finestra di dialogo che si vede nella figura 18.51 e, in base alla scelta del colore, si ottiene il numero corrispondente.

```
#!/bin/sh

RISULTATO=‘tempfile‘

dialog --title "Menu’" \
--menu "Scegli il colore che preferisci" \
0 0 0 \
0 nero \
1 marrone \
2 rosso \
3 arancio \
4 giallo \
5 verde \
6 blu \
7 viola \
8 grigio \
9 bianco 2> $RISULTATO

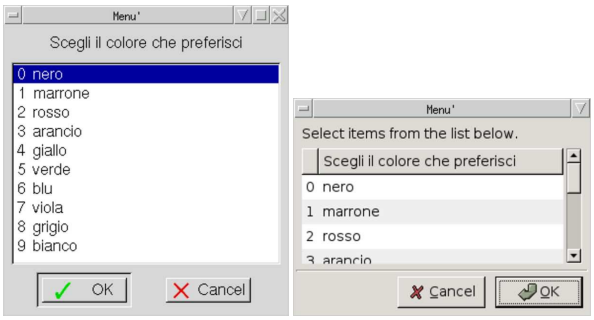
echo ‘cat $RISULTATO‘
```

Si osservi che è stato evitato l'uso di una lettera accentata, perché alcune delle varie interpretazioni del programma, non sono in grado di gestire altro che la codifica ASCII pura e semplice.

Figura 18.51. Esempio del funzionamento della finestra di dialogo ottenuta con l'opzione '**--menu**'.

```
.------------- Menu’ -------------.
| Scegli il colore che preferisci |
| .------------------------------. |
| |         0  nero              | |
| |         1  marrone           | |
| |         2  rosso             | |
| |         3  arancio           | |
| |         4  giallo            | |
| |         5  verde             | |
| |         6  blu               | |
| |         7  viola             | |
| |         8  grigio            | |
| |         9  bianco            | |
| ‘------------------------------’ |
|----------------------------------|
|     <  OK  >      <Cancel>       |
‘----------------------------------’
```


Figura 18.52. Esempio con Xdialog e con Gdialog.



Lo script successivo è una variante di quello precedente in cui si possono selezionare più colori assieme. Nella figura 18.54 si vede la finestra di dialogo che si ottiene.

```
#!/bin/sh

RISULTATO=‘tempfile‘

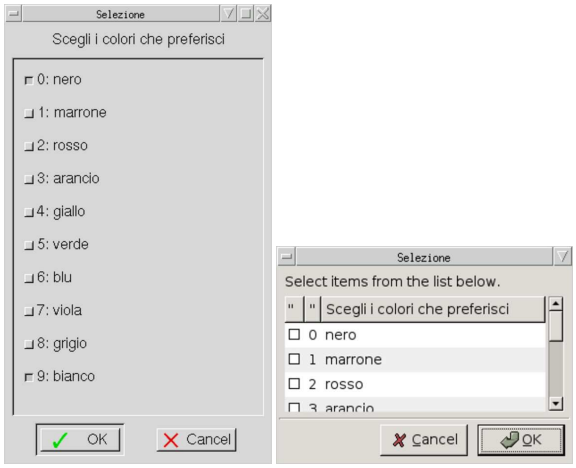
dialog --separate-output --title "Selezione" \
--checklist "Scegli i colori che preferisci" \
0 0 10 \
0 nero on \
1 marrone off \
2 rosso off \
3 arancio off \
4 giallo off \
5 verde off \
6 blu off \
7 viola off \
8 grigio off \
9 bianco on \
    2> $RISULTATO

echo ‘cat $RISULTATO‘
```

Figura 18.54. Esempio del funzionamento della finestra di dialogo ottenuta con l'opzione '**--checklist**'.

```
.------------- Menù -------------.
| Scegli i colori che preferisci |
| .------------------------------. |
| |      [X] 0  nero             | |
| |      [ ] 1  marrone          | |
| |      [ ] 2  rosso            | |
| |      [ ] 3  arancio          | |
| |      [ ] 4  giallo           | |
| |      [ ] 5  verde            | |
| |      [ ] 6  blu              | |
| |      [ ] 7  viola            | |
| |      [ ] 8  grigio           | |
| |      [X] 9  bianco           | |
| ‘------------------------------’ |
|----------------------------------|
|     <  OK  >      <Cancel>       |
‘----------------------------------’
```

Figura 18.55. Esempio con Xdialog e con Gdialog. Nel caso di Gdialog si può osservare che i valori predefiniti non vengono presi in considerazione.



Altri programmi affini.

kaptain(1)[14]	*http://kaptain.sourceforge.net*

18.7 Espressioni

Alcuni programmi sono particolarmente indicati per la costruzione di espressioni e, per questo motivo, il risultato della loro elaborazione si traduce essenzialmente nella restituzione di un valore (*exit status*).

18.7.1 Utilizzo di «false» e «true»

Il programma di servizio '**`false`**'[15] si limita a restituire il valore uno, corrispondente in pratica a *Falso* nell'ambito dei comandi di shell:

```
false
```

Il programma di servizio '**`true`**'[16] si limita a restituire il valore zero, corrispondente in pratica a *Vero* nell'ambito dei comandi di shell:

```
true
```

18.7.2 Utilizzo di «test» o di «[»

Il programma di servizio '**`test`**',[17] ovvero il comando interno di una shell standard, avente lo stesso nome, risolve (valuta) l'espressione indicata. Il valore restituito può essere *Vero* (corrispondente a zero) o *Falso* (corrispondente a uno) ed è pari al risultato della valutazione dell'espressione.

```
test espressione_condizionale
```

```
[ espressione_condizionale ]
```

Come si può osservare dai modelli mostrati, si può usare questo programma anche con il nome '**`[`**', ovvero una parentesi quadra aperta, ma in tal caso, alla fine dell'espressione deve apparire un'altra parentesi quadra chiusa. Questo strattagemma consente di scrivere delle espressioni con una notazione simile a quella di un linguaggio di programmazione comune.

Bisogna ricordare che, sia '**`test`**', sia '**`[`**', rappresentano il nome di un programma, o al limite il nome di un comando interno di alcune shell, pertanto le parentesi quadre non possono essere attaccate all'espressione da valutare, così come il nome '**`test`**' non potrebbe esserlo.

Le espressioni possono essere unarie o binarie. Le espressioni unarie sono usate spesso 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.

Per fare riferimento a un descrittore di I/O (per esempio uno dei flussi di dati standard), si può indicare un file nella forma '`/dev/fd/`***n***', dove il numero finale rappresenta l'***n***-esimo descrittore. In alternativa, si può fare riferimento direttamente ai file '`/proc/self/fd/`***n***', secondo lo standard del kernel Linux.

Nella tabella 18.57 e in quelle successive, vengono elencate le espressioni elementari che possono essere utilizzate in questo modo.

Tabella 18.57. Espressioni per la verifica del tipo di file.

Espressione	Descrizione
`-e` ***file***	*Vero* se il file esiste ed è di qualunque tipo.
`-b` ***file***	*Vero* se il file esiste ed è un dispositivo a blocchi.
`-c` ***file***	*Vero* se il file esiste ed è un dispositivo a caratteri.
`-d` ***file***	*Vero* se il file esiste ed è una directory.
`-f` ***file***	*Vero* se il file esiste ed è un file normale.
`-L` ***file***	*Vero* se il file esiste ed è un collegamento simbolico.
`-p` ***file***	*Vero* se il file esiste ed è un file FIFO (*pipe* con nome).
`-S` ***file***	*Vero* se il file esiste ed è un socket.
`-t` ***descrittore_file***	*Vero* se il descrittore indicato è aperto su un terminale.

Tabella 18.58. Espressioni per la verifica dei permessi e delle modalità dei file.

Espressione	Descrizione
`-g` ***file***	*Vero* se il file esiste ed è impostato il suo bit SGID.
`-u` ***file***	*Vero* se il file esiste ed è impostato il suo bit SUID.
`-k` ***file***	*Vero* se il file ha il bit Sticky attivo.
`-r` ***file***	*Vero* se il file esiste ed è leggibile.
`-w` ***file***	*Vero* se il file esiste ed è scrivibile.
`-x` ***file***	*Vero* se il file esiste ed è eseguibile.
`-O` ***file***	*Vero* se il file esiste e appartiene al numero UID efficace dell'utente attuale.
`-G` ***file***	*Vero* se il file esiste e appartiene al GID efficace dell'utente attuale.

Tabella 18.59. Espressioni per la verifica di altre caratteristiche dei file.

Espressione	Descrizione
`-s` ***file***	*Vero* se il file esiste e ha una dimensione maggiore di zero.
file_1 `-nt` ***file_2***	*Vero* se il primo file ha la data di modifica più recente.
file_1 `-ot` ***file_2***	*Vero* se il primo file ha la data di modifica più vecchia.
file_1 `-et` ***file_2***	*Vero* se i due nomi corrispondono allo stesso inode.

Tabella 18.60. Espressioni per la verifica e la comparazione delle stringhe.

Espressione	Descrizione
`-z` ***stringa***	*Vero* se la lunghezza della stringa è zero.
`-n` ***stringa***	*Vero* se la lunghezza della stringa è diversa da zero.
stringa_1 `=` ***stringa_2***	*Vero* se le stringhe sono uguali.
stringa_1 `!=` ***stringa_2***	*Vero* se le stringhe sono diverse.

Espressione	Descrizione
stringa_1 < *stringa_2*	*Vero* se la prima stringa è lessicograficamente precedente.
stringa_1 > *stringa_2*	*Vero* se la prima stringa è lessicograficamente successiva.

Tabella 18.61. Espressioni per il confronto numerico. Come operandi possono essere utilizzati numeri interi, positivi o negativi, oppure l'espressione speciale '`-l` *stringa*' che restituisce la lunghezza della stringa indicata.

Espressione	Descrizione
op_1 `-eq` *op_2*	*Vero* se gli operandi sono uguali.
op_1 `-ne` *op_2*	*Vero* se gli operandi sono differenti.
op_1 `-lt` *op_2*	*Vero* se il primo operando è inferiore al secondo.
op_1 `-le` *op_2*	*Vero* se il primo operando è inferiore o uguale al secondo.
op_1 `-gt` *op_2*	*Vero* se il primo operando è maggiore del secondo.
op_1 `-ge` *op_2*	*Vero* se il primo operando è maggiore o uguale al secondo.

Tabella 18.62. Operatori logici.

Espressione	Descrizione
`!` *espressione*	Inverte il risultato logico dell'espressione.
`(` *espressione* `)`	*Vero* se l'espressione contenuta tra parentesi risulta vera. Le parentesi possono essere usate per cambiare l'ordine normale di risoluzione delle espressioni.
espressione `-a` *espressione*	*Vero* se entrambe le espressioni danno un risultato *Vero*.
espressione `-o` *espressione*	*Vero* se almeno un'espressione dà un risultato *Vero*.

Seguono due esempi senza descrizione, mostrando il risultato ottenuto attraverso lo standard output:

```
$ test 1 -lt 2 && echo "ok" [Invio]

  ok
$ [ "prova" = "pro" ] ↩
↪|| echo "le stringhe non combaciano" [Invio]

  le stringhe non combaciano
$ test -d /bin && echo "/bin è una directory" [Invio]

  /bin è una directory
$ [ -e /bin/sh && ] echo "/bin/sh è un file eseguibile" [Invio]

  /bin/sh è un file eseguibile
```

18.7.3 Utilizzo di «expr»

Il programma di servizio '**`expr`**'[18] valuta un'espressione e ne emette il risultato attraverso lo standard output. Ogni elemento dell'espressione deve essere un argomento separato.

```
expr espressione
```

Gli operandi possono essere numeri o stringhe a seconda del tipo di operazione che si intende applicare. Se vengono usate le parentesi, è molto probabile che la shell utilizzata costringa a proteggerle attraverso le tecniche che la stessa mette a disposizione. Il valore restituito da '**`expr`**' dipende essenzialmente dal risultato dell'espressione nel modo seguente:

Valore di uscita	Quando si ottiene
0	se il risultato è diverso sia dal valore nullo che da zero;
1	se il risultato è nullo oppure zero;
2	se l'espressione non è valida.

Non si deve confondere il valore restituito dal programma con il risultato delle espressioni: '**`expr`**' valuta le espressioni come farebbe un linguaggio di programmazione comune, attribuendo al valore uno il significato di *Vero* e a zero il valore *Falso*.

Le espressioni possono essere concatenate attraverso degli operatori logici, come descritto nella tabella successiva.

Tabella 18.68. Operatori logici.

Operatore	Descrizione
`|`	È simile all'operatore OR: se la prima delle due espressioni genera un risultato diverso da zero e dal valore nullo, il risultato globale è questo; altrimenti il risultato è quello della seconda.
`&`	È simile all'operatore AND: se entrambe le espressioni generano un risultato diverso da zero e dal valore nullo, il risultato globale è quello della prima espressione; altrimenti il risultato è zero.

Gli operatori di comparazione sono i soliti che si usano in matematica, come descritto nella tabella successiva.

Tabella 18.69. Operatori di comparazione.

Operatore	Descrizione
`<`	Minore.
`<=`	Minore o uguale.
`=`	Uguale.
==	Identicamente uguale (di fatto equivalente alla notazione '`=`').
`!=`	Diverso.
`>`	Maggiore.
`>=`	Maggiore o uguale.

Se la comparazione è corretta (*Vero*), genera il valore uno, altrimenti si ottiene zero.

'**`expr`**' tenta inizialmente di considerare gli operatori da confrontare come numerici; se in questo modo fallisce l'operazione, tenta quindi di eseguire una comparazione lessicografica.

Gli operatori numerici sono i soliti che si usano in matematica, come descritto nella tabella successiva.

Tabella 18.70. Espressioni numeriche.

Operatore	Descrizione
`+`	Addizione.
`-`	Sottrazione.
`*`	Moltiplicazione.
`/`	Divisione.
`%`	Resto o modulo.

Tabella 18.71. Espressioni su stringhe.

Operatore	Descrizione
stringa : *espressione_regolare*	L'operatore '**:**' viene usato per comparare una stringa con un'espressione regolare. Questa espressione regolare può essere solo di tipo elementare (BRE) e si considera che contenga implicitamente un accento circonflesso ('**^**') iniziale (pertanto, l'espressione regolare deve corrispondere a partire dell'inizio della stringa). L'espressione regolare può contenere una coppia di parentesi tonde protette nel modo seguente: '**\(**…**\)**'. Se vengono usate, il risultato di questa comparazione è la stringa che corrisponde alla parte di espressione regolare racchiusa tra queste parentesi. Se non vengono usate, il risultato è il numero di caratteri che corrispondono. Se la comparazione fallisce e sono state usate le parentesi '**\(**' e '**\)**', il risultato è una stringa nulla. Se la comparazione fallisce e non sono state usate queste parentesi, il risultato è zero.
`match` *stringa espressione_regolare*	Questa sintassi è un modo alternativo per eseguire una comparazione di stringhe. Si comporta in modo identico all'uso dell'operatore '**:**'.
`substring` *stringa* ↩ ↪*posizione lunghezza*	Restituisce una sottostringa a partire dalla posizione indicata per una lunghezza massima stabilita. Se uno dei valori usati per indicare la posizione o la lunghezza è negativo, il risultato è la stringa nulla.
`index` *stringa insieme_di_caratteri*	Restituisce la prima posizione all'interno della stringa, a cui corrisponde uno dei caratteri che compone l'insieme di caratteri.
`length` *stringa*	Restituisce la lunghezza della stringa.

Segue la descrizione di alcuni esempi.

- `$ export miavar=2` [*Invio*]

  `$ expr $miavar + 1` [*Invio*]

  ```
  3
  ```

  Viene creata la variabile '**miavar**' assegnandole il valore '**2**', quindi calcola la somma tra il suo contenuto e '**1**'.

- `$ expr abc : 'a\(.\)c'` [*Invio*]

  ```
  b
  ```

  Estrae dalla stringa la lettera centrale attraverso l'espressione regolare.

- `$ expr index ambaraba br` [*Invio*]

  ```
  3
  ```

  Cerca la prima posizione all'interno della stringa '**ambaraba**' che corrisponda alla lettera '**b**', oppure alla lettera '**r**'.

18.8 Ridirezione

La ridirezione dei flussi di input e di output dei programmi viene svolta dalle shell. Il programma '**tee**' è molto importante in queste situazioni perché permette di copiare in un file il flusso di dati che lo attraversa.

18.8.1 Utilizzo di «tee»

Il programma di servizio '**tee**'[19] emette attraverso lo standard output quanto ricevuto dallo standard input, facendone una copia anche nei file indicati come argomento. Si tratta quindi di un filtro che permette di copiare i dati in transito in un file.

```
tee [opzioni] [file…]
```

Tabella 18.75. Alcune opzioni.

Opzione	Descrizione
`-a` `--append`	Aggiunge i dati ai file di destinazione, accodandoli, invece di sovrascriverli.
`-i` `--ignore-interrupts`	Ignora i segnali di interruzione.

18.9 Pause

Nella scrittura di script ci sono situazioni in cui è necessario fare delle pause, per permettere il completamento di qualcosa che non può essere controllato in modo sequenziale.

18.9.1 Utilizzo di «sleep»

Il programma di servizio '**sleep**'[20] attende per il tempo indicato come argomento, quindi termina la sua esecuzione. La durata si esprime attraverso un numero intero che rappresenta una quantità di secondi.

```
sleep durata
```

A seconda della realizzazione del programma, potrebbe essere consentito l'uso di numeri non interi, oppure l'aggiunta di un moltiplicatore, ma in generale, per ottenere la compatibilità massima, è meglio limitarsi all'uso di numeri interi che rappresentano secondi.

L'esempio seguente richiede una pausa di 10 s e quindi termina la sua esecuzione:

`$ sleep 10` [*Invio*]

18.10 Raccolta di funzioni per una shell POSIX

In questa sezione viene proposta una raccolta di funzioni per una shell POSIX comune, allo scopo di facilitare la gestione di un sistema GNU/Linux. Queste funzioni derivano dall'esperienza di NLNX (una distribuzione GNU/Linux per architettura x86, derivata da Debian.

Queste funzioni si avvalgono evidentemente di programmi di servizio comuni nei sistemi Unix; in particolare SED (sezione 23.5). Di conseguenza, la comprensione del funzionamento di queste funzioni richiede una buona conoscenza nell'uso di tali programmi.

18.10.1 Estrapola da «/etc/passwd» le righe di un certo intervallo di numeri UID

```
password_records_by_uid_range uid_min uid_max ↩
↪                             < /etc/passwd > output_file
```

```
passwd_records_by_uid_range () {
    local UID_MIN="$1"
    local UID_MAX="$2"
    local RECORD=""
    local USER_ID=""
    #
    while read -r RECORD
    do
        USER_ID=‘echo $RECORD | sed "s/^[^:]*:[^:]*:\([0-9]*\):.*/\1/"‘
        #
        if [ 0$USER_ID -ge $UID_MIN ] && [ 0$USER_ID -le $UID_MAX ]
        then
            echo $RECORD
        fi
    done
}
```

L’esempio seguente utilizza la funzione per estrapolare le righe di ‘`/etc/passwd`’ associate a numeri UID tra 1000 e 29999. Il risultato viene emesso semplicemente attraverso lo standard output.

```
cat /etc/passwd | passwd_records_by_uid_range 1000 29999
```

18.10.2 Estrapola da «/etc/passwd» le righe di un certo intervallo di numeri GID

«

```
password_records_by_gid_range gid_min gid_max ↩
↪                             < /etc/passwd > output_file
```

```
passwd_records_by_gid_range () {
    local GID_MIN="$1"
    local GID_MAX="$2"
    local RECORD=""
    local GROUP_ID=""
    #
    while read -r RECORD
    do
        GROUP_ID=‘echo $RECORD | sed "s/^[^:]*:[^:]*:[^:]*:\([0-9]*\):.*/\1/"‘
        #
        if [ 0$GROUP_ID -ge $GID_MIN ] && [ 0$GROUP_ID -le $GID_MAX ]
        then
            echo $RECORD
        fi
    done
}
```

L’esempio seguente utilizza la funzione per estrapolare le righe di ‘`/etc/passwd`’ associate al numero GID zero.

```
cat /etc/passwd | passwd_records_by_gid_range 0 0
```

18.10.3 Estrapola da «/etc/group» le righe di un certo intervallo di numeri GID

«

```
group_records_by_gid_range gid_min gid_max ↩
↪                          < /etc/group > output_file
```

```
group_records_by_gid_range () {
    local GID_MIN="$1"
    local GID_MAX="$2"
    local RECORD=""
    local GROUP_ID=""
    #
    while read -r RECORD
    do
        GROUP_ID=‘echo $RECORD | sed "s/^[^:]*:[^:]*:\([0-9]*\):.*/\1/"‘
        #
        if [ 0$GROUP_ID -ge $GID_MIN ] && [ 0$GROUP_ID -le $GID_MAX ]
        then
            echo $RECORD
        fi
    done
}
```

L’esempio seguente utilizza la funzione per estrapolare le righe di ‘`/etc/group`’ associate ai numeri GID da 1000 a 29999.

```
cat /etc/group | group_records_by_gid_range 1000 29999
```

18.10.4 Seleziona un utente interattivamente

«

Per selezionare interattivamente l’utente si usa Dialog e si preferisce depositare il nome scelto in un file:

```
select_user uid_min uid_max output_file < /etc/passwd
```

```
select_user_menu () {
    #
    local UID_MIN="$1"
    local UID_MAX="$2"
    local RECORD=""
    local USER_ID=""
    #
    local USER_AND_HOME_LIST=""
    #
    while read -r RECORD
    do
        USER_ID=‘echo $RECORD | sed "s/^[^:]*:[^:]*:\([0-9]*\):.*/\1/"‘
        #
        if [ 0$USER_ID -ge $UID_MIN ] && [ 0$USER_ID -le $UID_MAX ]
        then
            echo $RECORD | sed "s/^\(.*\):.*:.*:.*:.*:\(.*\):.*/\1 \2/g"
        fi
    done
}
```

```
select_user () {
    #
    local UID_MIN="$1"
    local UID_MAX="$2"
    local INPUT_FILE="$3"
    local OUTPUT_FILE="$4"
    #
    local USER_AND_HOME_LIST=""
    local SELECTED_USER=""
    #
    local TEMPORARY=‘tempfile‘
    printf "" > $TEMPORARY
    #
    USER_AND_HOME_LIST=‘cat $INPUT_FILE \
                        | select_user_menu $UID_MIN $UID_MAX \
                        | sort -u‘
    #
    if dialog                           \
        --clear                         \
        --title "Users"                 \
        --menu  "Select a user name."   \
        0 0 0                           \
        $USER_AND_HOME_LIST             \
        "!EXIT!" "."                    \
        2> $TEMPORARY
    then
        SELECTED_USER=‘cat $TEMPORARY‘
        echo "" > $TEMPORARY
        #
        if [ "$SELECTED_USER" = "!EXIT!" ]
        then
            SELECTED_USER=""
        fi
    else
        SELECTED_USER=""
    fi
    #
    echo "$SELECTED_USER" > "$OUTPUT_FILE"
    rm $TEMPORARY
}
```

L’esempio seguente utilizza la funzione per estrapolare un utente dal file ‘`/etc/passwd`’, tra i numeri UID 1000 e 29999. Il risultato viene visualizzato attraverso lo standard output.

```
cat /etc/passwd | select_user 1000 29999 /etc/passwd /tmp/ciao
cat /tmp/ciao
```

18.10.5 Seleziona un campo di una certa riga da un file come «/etc/passwd», «/etc/group» e simili

«

```
table_get_column_field file indice n_colonna
```

```
table_get_column_field () {
    #
    local INPUT_FILE="$1"
    local INDEX="$2"
    local COLUMN="$3"
    local RECORD=""
    local FIELD=""
    #
    if [ ! -r "$INPUT_FILE" ]
    then
        echo 1>&2 "[$0] cannot read \"$INPUT_FILE\"."
        return
    fi
    #
    if   [ "$COLUMN" = "1" ]
    then
```

```
        echo 1>&2 "[$0] index should be more than one."
        echo "$INDEX"
        return
    fi
    #
    RECORD=‘grep -m 1 "^${INDEX}:" "$INPUT_FILE"‘
    #
    if [ "$RECORD" = "" ]
    then
        echo 1>&2 "[$0] index \"$INDEX\" not found inside \"$INPUT_FILE\"."
        return
    fi
    #
    while [ "$COLUMN" -gt "1" ]
    do
        RECORD=‘echo $RECORD | sed "s/^[^:]*://"‘
        COLUMN=$(($COLUMN - 1))
    done
    #
    FIELD=‘echo $RECORD | sed "s/^\([^:]*\).*$/\1/"‘
    #
    echo $FIELD
    return
}
```

L’esempio seguente utilizza la funzione per estrapolare la parola d’ordine cifrata dell’utente «tizio» dal file ‘`/etc/shadow`’, visualizzando il risultato attraverso lo standard output.

```
table_get_column_field /etc/shadow tizio 2
```

18.10.6 Aggiunge un utente Unix e Samba, simultaneamente

«

```
user_add_unix_samba user passwd home full_name room work_ph ↩
↪ home_ph
```

```
user_add_unix_samba () {
    local NEW_USER="$1"
    local NEW_PASSWD="$2"
    local NEW_HOME="$3"
    local NEW_FULL_NAME="$4"
    local NEW_ROOM="$5"
    local NEW_WORK_PHONE="$6"
    local NEW_HOME_PHONE="$7"
    local TEMP_USER=""

    # Elimina le virgole e altri caratteri inopportuni
    # dalle informazioni dell’utente.
    NEW_FULL_NAME=‘echo $NEW_FULL_NAME   | sed "/,:=/ /g"‘
    NEW_ROOM=‘echo $NEW_ROOM             | sed "/,:=/ /g"‘
    NEW_WORK_PHONE=‘echo $NEW_WORK_PHONE | sed "/,:=/ /g"‘
    NEW_HOME_PHONE=‘echo $NEW_HOME_PHONE | sed "/,:=/ /g"‘

    # Verifica che i dati siano validi.
    if   [ "$NEW_USER" = "" ]
    then
        echo 1>&1 "[$0] cannot create user with no name."
        false
        return
    elif echo "$NEW_USER" | grep "[^a-z0-9]" > "/dev/null"
    then
        echo 1>&1 "[$0] user name must contain only lower case letters and numbers."
        false
        return
    fi

    # Il nominativo utente non deve superare i 16 caratteri
    TEMP_USER=‘echo $NEW_USER | sed "s/^\(................\).*$/\1/"‘
    if [ "$TEMP_USER" = "$NEW_USER" ]
    then
        true
    else
        echo 1>&2 "[$0] user name cannot be longer than 16 characters."
        false
        return
    fi

    # Controlla la directory personale
    if [ "$NEW_HOME" = "" ]
    then
        NEW_HOME="/home/$NEW_USER"
    fi
    #
    if [ -e "$NEW_HOME" ]
    then
        echo 1>&2 "[$0] home directory \"$NEW_HOME\" cannot be created."
        false
        return
    fi

    # Controlla che ci sia la parola d’ordine.
    if [ "$NEW_PASSWD" = "" ]
    then
        echo 1>&2 "[$0] the new user must have a password."
        false
        return
    fi
```



```
    # Crea l’utente, usando inizialmente lo script "adduser",
    # secondo le convenzioni Debian.
    if adduser \
        --disabled-password \
        --no-create-home \
        --home "$NEW_HOME" \
        --gecos "$NEW_FULLNAME,$NEW_ROOM,$NEW_WORK_PHONE,$NEW_HOME_PHONE"\
        $NEW_USER
    then

        # Crea manualmente la directory personale.
        mkdir -p "$NEW_HOME"
        rmdir    "$NEW_HOME"
        cp -dpR  /etc/skel  "$NEW_HOME"
        chown -R $NEW_USER: "$NEW_HOME"
        chmod 0755          "$NEW_HOME"

        # Aggiunge un collegamento simbolico all’interno di
        # "/var/mail/", perché alcuni programmi ne hanno bisogno.
        # Si presume che il file "~/mail/mbox" venga usato per
        # accumulare i messaggi di posta elettronica dell’utente.
        rm "/var/mail/$NEW_USER" 2> "/dev/null"
        ln -s "../../$NEW_HOME/mail/mbox" "/var/mail/$NEW_USER"

        # Attribuisce la parola d’ordine (nel farlo, crea anche
        # l’utenza per Samba).
        if user_passwd_unix_samba "$NEW_USER" "$NEW_PASSWD"
        then
            true
        else
            # Elimina l’utenza.
            user_del_unix_samba "$NEW_USER"
            false
            return
    else
        echo 1>&2 "[$0] cannot create user \"$NEW_USER\"."
        false
        return
    fi
    #
    rm $TEMPORARY
}
```

18.10.7 Cambia la parola d’ordine a un utente Unix e Samba, simultaneamente

«

```
user_passwd_unix_samba user passwd
```

```
user_passwd_unix_samba () {
    local OLD_USER="$1"
    local NEW_PASSWD="$2"
    #
    local TEMPORARY=‘tempfile‘
    printf "" > $TEMPORARY

    # Tenta di eliminare un’utenza vecchia, con lo stesso nome
    # dalla gestione di Samba.
    smbpasswd -x $OLD_USER 2> "/dev/null" 1> "/dev/null"

    # Imposta la parola d’ordine Unix.
    if ( sleep 1 ; echo $NEW_PASSWD ; sleep 1 ; echo $NEW_PASSWD ) \
       | /usr/bin/passwd $NEW_USER 2> $TEMPORARY
    then
        # Crea o ricrea l’utenza Samba.
        if [ -x /usr/bin/smbpasswd ]
        then
            if ( sleep 1 ; echo $NEW_USER ; sleep 1 ; echo $NEW_USER ) \
               | /usr/bin/smbpasswd -s -a $NEW_USER 2> $TEMPORARY
            then
                true
            else
                echo 1>&2 "[$0] problem changing Samba password to user \
"$NEW_USER": ‘cat $TEMPORARY‘."
                rm "$TEMPORARY"
                false
                return
            fi
        fi
    else
        echo 1>&2 "[$0] problem changing Unix password to user \
"$NEW_USER": ‘cat $TEMPORARY‘."
        rm "$TEMPORARY"
        false
        return
    fi
    rm "$TEMPORARY"
}
```

18.10.8 Elimina un utente Unix e Samba, simultaneamente

«

```
user_del_unix_samba user
```

```
user_del_unix_samba () {
    local OLD_USER="$1"
    local OLD_HOME=""
```

```
    # Trova la directory personale.
    OLD_HOME=‘cat /etc/passwd \
             | grep "^$OLD_USER:" \
             | sed "s/^[^:]*:[^:]*:[^:]*:[^:]*:[^:]*:\([^:]*\):[^:]*$/\1/"‘

    # Elimina l’utenza dalla gestione di Samba.
    if [ -x /usr/bin/smbpasswd ]
    then
        if smbpasswd -x $OLD_USER
        then
            true
        else
            echo 1>&2 "[$0] cannot remove Samba user \"$OLD_USER\"."
        fi
    fi

    # Elimina l’utenza Unix.
    if deluser --remove-home "$OLD_USER"
    then
        # Elimina il collegamento simbolico in "/var/mail".
        rm "/var/mail/$OLD_USER"
    else
        echo 1>&2 "[$0] cannot remove Unix user \"$OLD_USER\"."
        false
        return
    fi
}
```

18.10.9 Seleziona interattivamente salvando la selezione

La funzione che viene proposta richiede due elenchi, contenuti in altrettanti file di testo, contenenti rispettivamente un insieme di voci e un sottoinsieme di voci da selezionare. Lo scopo è quello di cambiare il sottoinsieme selezionato e di aggiornare il contenuto del secondo file.

```
select_subset_save file_list_full file_list_selected ↩
↪                  selection_description unselection_description ↩
↪                  selection_word ↩
↪                  list_header list_description ↩
↪                  file_return_value_selected ↩
↪                  file_return_value_unselected
```

La funzione si avvale di un’altra che deve produrre l’opposto del sottoinsieme selezionato:

```
selection_invert ()
{
    local LIST_FULL="$1"
    local LIST_SELECTED="$2"
    local LIST_UNSELECTED=""
    local l=""
    local s=""
    local ITEM_FOUND=""
    #
    if [ "$LIST_SELECTED" = "" ]
    then
        LIST_UNSELECTED="$LIST_FULL"
    else
        for l in $LIST_FULL
        do
            ITEM_FOUND="0"
            for s in $LIST_SELECTED
            do
                if [ "$l" = "$s" ]
                then
                    ITEM_FOUND="1"
                    break
                fi
            done
            if [ "$ITEM_FOUND" = "1" ]
            then
                true
            else
                LIST_UNSELECTED="$LIST_UNSELECTED $l"
            fi
        done
    fi
    #
    echo "$LIST_UNSELECTED"
}
```

```
select_subset_save ()
{
```



```
    local FILE_LIST_FULL="$1"
    local FILE_LIST_SELECTED="$2"
    local SELECTION_DESCRIPTION="$3"
    local UNSELECTION_DESCRIPTION="$4"
    local SELECTION_WORD="$5"
    local LIST_HEADER="$6"
    local LIST_DESCRIPTION="$7"
    local FILE_RETURN_VALUE_SELECTED="$8"
    local FILE_RETURN_VALUE_UNSELECTED="$9"
    #
    local LIST_FULL=""
    local LIST_SELECTED=""
    local LIST_SELECTED_VALID=""
    local LIST_UNSELECTED=""
    local ITEM_FOUND=""
    local ITEM_SELECTION=""
    local SELECTION=""
    local SELECTED=""
    #
    local TEMPORARY=‘tempfile‘
    touch $TEMPORARY
    #
    LIST_FULL=‘cat $FILE_LIST_FULL | sed "s/#.*$//" 2> /dev/null‘
    LIST_SELECTED=‘cat $FILE_LIST_SELECTED | sed "s/#.*$//" 2> /dev/null‘
    #
    if [ "$LIST_FULL" = "" ]
    then
        dialog --msgbox "The file \"$FILE_LIST_FULL\" is empty!" 0 0
        false
        return
    fi

    # Unselected and select again.
    LIST_UNSELECTED=‘selection_invert "$LIST_FULL" "$LIST_SELECTED"‘
    LIST_SELECTED=‘selection_invert "$LIST_FULL" "$LIST_UNSELECTED"‘

    # Item selection list.
    for i in $LIST_SELECTED
    do
        ITEM_SELECTION="$ITEM_SELECTION $i ${SELECTION_WORD}_$i on"
    done
    for i in $LIST_UNSELECTED
    do
        ITEM_SELECTION="$ITEM_SELECTION $i ${SELECTION_WORD}_$i off"
    done

    # Dialog.
    while true
    do
        if dialog  \
            --clear \
            --title "$LIST_HEADER" \
            --checklist "$LIST_DESCRIPTION" \
            0 0 0 \
            "!ALL!"  "$SELECTION_DESCRIPTION" off     \
            "!NONE!"  "$UNSELECTION_DESCRIPTION" off   \
            $ITEM_SELECTION   \
            2> $TEMPORARY
        then
            SELECTION=‘cat $TEMPORARY‘
            echo "" > $TEMPORARY

            # Do something.
            if   echo "$SELECTION" | grep "!ALL!" > "/dev/null"
            then
                ITEM_SELECTION=""
                for i in $LIST_FULL
                do
                    ITEM_SELECTION="$ITEM_SELECTION $i ${SELECTION_WORD}_$i on"
                done
                continue
            elif   echo "$SELECTION" | grep "!NONE!" > "/dev/null"
            then
                ITEM_SELECTION=""
                for i in $LIST_FULL
                do
                    ITEM_SELECTION="$ITEM_SELECTION $i ${SELECTION_WORD}_$i off"
                done
                continue
            else
                LIST_SELECTED=""
                for s in $SELECTION
                do

                    # Remove the double quotes.
                    SELECTED=‘echo $s | sed "s/\"//g"‘
                    LIST_SELECTED="$LIST_SELECTED $SELECTED"
                done
                LIST_UNSELECTED=‘selection_invert "$LIST_FULL" "$LIST_SELECTED"‘

                # Save data.
                printf "" > "$FILE_LIST_SELECTED"
                for s in $LIST_SELECTED
                do
                    echo "$s" >> "$FILE_LIST_SELECTED"
                done
                # Stop the loop.
                break
            fi
        else
            false
            return
        fi
```

```
    done
    #
    rm -f $TEMPORARY
    #
    printf "" > "$FILE_RETURN_VALUE_SELECTED"
    for s in $LIST_SELECTED
    do
        echo "$s" >> "$FILE_RETURN_VALUE_SELECTED"
    done
    #
    printf "" > "$FILE_RETURN_VALUE_UNSELECTED"
    for s in $LIST_UNSELECTED
    do
        echo "$s" >> "$FILE_RETURN_VALUE_UNSELECTED"
    done
    #
    true
}
```

Si osservi l'esempio seguente:

```
select_subset_save  /etc/xxx/ELENCO_COMPLETO \
                    /etc/xxx/ELENCO_SELEZIONATO \
                    "selezione di tutte le voci" \
                    "rimozione di tutte le selezioni" \
                    "attiva" \
                    "attivazione voci" \
                    "Selezionare le voci da attivare:" \
                    /tmp/xxx.selezionati \
                    /tmp/xxx.non_selezionati
```

In questo c'è il file '`/etc/xxx/ELENCO_COMPLETO`', contenente un elenco di voci (ogni voce deve costituire una parola sola), quindi c'è il file '`/etc/xxx/ELENCO_SELEZIONATO`' con un sottoinsieme delle voci del primo file. Attraverso la selezione che si esegue con la funzione, il file '`/etc/xxx/ELENCO_SELEZIONATO`' viene aggiornato con il nuovo sottoinsieme, ma in ogni caso si genera il file '`/tmp/xxx.selezionati`' con le voci selezionate e il file '`/tmp/xxx.non_selezionati`' con le altre voci.

18.10.10 Verifica che l'utente proprietario di un file possa accedervi

Si tratta di tre funzioni, le quali, rispettivamente, verificano se l'utente proprietario di un file o di una directory ha il permesso di accedervi in lettura, scrittura, esecuzione o attraversamento. Si osservi che si tratta di un controllo basato sul proprietario del file e non sui privilegi che l'interprete dello script si trova ad avere in un certo momento.

```
file_owner_access_r file
```

```
file_owner_access_w file
```

```
file_owner_access_x file
```

Le tre funzioni restituiscono *Vero* nel caso il permesso richiesto sia disponibile; altrimenti *Falso* in caso contrario.

```
file_owner_access_r ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^.\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "r" ]
        then
            true
            return
        fi
    fi
    false
}
```



```
file_owner_access_w ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^..\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "w" ]
        then
            true
            return
        fi
    fi
    false
}
```

```
file_owner_access_x ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^...\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "x" ] || [ "$ACCESS" = "s" ]
        then
            true
            return
        fi
    fi
    false
}
```

18.10.11 Verifica che il gruppo proprietario di un file possa accedervi

Si tratta di tre funzioni analoghe a quelle della sezione precedente, le quali verificano se il gruppo proprietario di un file o di una directory ha il permesso di accedervi in lettura, scrittura, esecuzione o attraversamento.

```
file_group_access_r file
```

```
file_group_access_w file
```

```
file_group_access_x file
```

Le tre funzioni restituiscono *Vero* nel caso il permesso richiesto sia disponibile; altrimenti *Falso* in caso contrario.

```
file_group_access_r ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^....\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "r" ]
        then
            true
            return
        fi
    fi
    false
}
```

```
file_group_access_w ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^.....\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "w" ]
        then
            true
            return
        fi
    fi
    false
}
```

```
file_group_access_x ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^.....\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "x" ] || [ "$ACCESS" = "s" ]
        then
            true
            return
        fi
    fi
    false
}
```

18.10.12 Verifica che gli utenti diversi possano accedere a un file

Si tratta di tre funzioni analoghe a quelle della sezione precedente, le quali verificano se gli utenti diversi dal proprietario e dal gruppo del file o di una directory, hanno il permesso di accedervi in lettura, scrittura, esecuzione o attraversamento.

```
file_other_access_r file
```

```
file_other_access_w file
```

```
file_other_access_x file
```

Le tre funzioni restituiscono *Vero* nel caso il permesso richiesto sia disponibile; altrimenti *Falso* in caso contrario.

```
file_other_access_r ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^.......\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "r" ]
        then
            true
            return
        fi
    fi
    false
}
```



```
file_other_access_w ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^........\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "w" ]
        then
            true
            return
        fi
    fi
    false
}
```

```
file_other_access_x ()
{
    local FILE_NAME="$1"
    local ACCESS=""
    #
    if [ -e "$FILE_NAME" ]
    then
        ACCESS=‘ls -n -d "$FILE_NAME" \
                  | sed "s/^........\(.\).*$/\1/"‘
        #
        if [ "$ACCESS" = "x" ] || [ "$ACCESS" = "s" ]
        then
            true
            return
        fi
    fi
    false
}
```

18.10.13 Estrapola il nome dell’utente proprietario di un file

```
file_owner_name file
```

```
file_owner_name ()
{
    local FILE_NAME="$1"
    local OWNER_UID=""
    local OWNER_NAME=""
    #
    if [ -e "$FILE_NAME" ]
    then
        OWNER_UID=‘ls -n -d "$FILE_NAME" \
             | sed "s/^[^ ]*  *[^ ]*  *\([^ ]*\)  *.*$/\1/"‘
        #
        OWNER_NAME=‘cat /etc/passwd \
                   | grep "^[^:]*:[^:]*:$OWNER_UID:" \
                   | head -n 1 \
                   | sed "s/^\(.*\):.*:.*:.*:.*:.*:.*$/\1/"‘
        #
        echo "$OWNER_NAME"
    else
        false
    fi
}
```

18.10.14 Estrapola la directory personale di un utente

```
user_home_directory utente
```

```
user_home_directory ()
{
    local USER_NAME="$1"
    local HOME_DIRECTORY=""
    #
    HOME_DIRECTORY=‘cat /etc/passwd \
                | grep "^$USER_NAME:" \
                | head -n 1 \
                | sed "s/^.*:.*:.*:.*:.*:\(.*\):.*$/\1/"‘
    #
    echo "$HOME_DIRECTORY"
    #
}
```

18.10.15 Estrapola il numero UID di un utente

```
user_uid utente
```

```
user_uid ()
{
    local USER_NAME="$1"
    local USER_UID=""
    #
    USER_UID=‘cat /etc/passwd \
              | grep "^$USER_NAME:" \
              | head -n 1 \
              | sed "s/^.*:.*:\(.*\):.*:.*:.*:.*$/\1/"‘
    #
    echo "$USER_UID"
    #
}
```

18.11 Approfondimento: un esercizio con Dialog

In questa sezione viene proposto uno script basato sull’uso di Dialog, per realizzare un piccolo programma frontale con uno scopo preciso: facilitare la scrittura di testi come lo si potrebbe fare con una macchina da scrivere. Questo tipo di lavoro, così come è impostato, è organizzato in modo da definire prima l’azione, poi gli oggetti coinvolti, come succedeva spesso nei programmi gestionali dei primi anni 1980. Lo script è disponibile qui: *allegati/shell-testi.sh*.

Lo script è pensato per l’uso da parte di chi non vuole sapere come si usa un sistema operativo, a costo di accontentarsi di poche cose. Per questa ragione, lo script è pensato per sostituire il file ‘`~/.profile`’, in modo che dopo l’identificazione dell’utente, ciò che appare sia un menù di funzioni.

Questo script deve organizzare la gestione di file di testo, cercando di evitare che l’utente incorra in errori gravi, dovuti all’assoluta ignoranza di questioni che generalmente sono ritenute banali. Per questo, all’avvio verifica l’esistenza di una directory che ha lo scopo di raccogliere i documenti testuali e di sottodirectory per la conservazione di diverse versioni precedenti. Se queste directory mancano, provvede a crearle.

Figura 18.105. All’avvio, lo script mostra il menù delle funzioni disponibili.

```
.--------------------Menù principale-----------------------.
| Funzioni disponibili:                                     |
| .-------------------------------------------------------. |
| |   new     crea un documento nuovo                     | |
| |   view    visualizza un documento già esistente       | |
| |   edit    modifica un documento già esistente         | |
| |   copy    copia un documento                          | |
| |   rename  cambia nome a un documento                  | |
| |   delete  elimina un documento che non serve più      | |
| |   list    elenca il contenuto di un disco rimovibile  | |
| |   view2   visualizza un documento contenuto in un dis| |
| |   erase   cancella un disco rimovibile                | |
| |   format  inizializza un disco rimovibile             | |
| |   export  esporta dei documenti in un disco rimovibil| |
| |   import  importa dei documenti da un disco rimovibil| |
| |   print   stampa un documento                         | |
| |   startx  avvia la grafica                            | |
| |   quit    fine lavoro                                 | |
| ‘-------------------------------------------------------’ |
|-----------------------------------------------------------|
|               <  OK  >           <Cancel>                |
‘-----------------------------------------------------------’
```

Come si può vedere, le lettere accentate sono state realizzate con l’uso di un apostrofo inverso, per evitare qualunque problema di compatibilità con la configurazione della console.

Nella parte iniziale dello script vengono inseriti i comandi che dovrebbero trovarsi invece nel file ‘`~/.profile`’, quindi vengono dichiarate delle variabili di ambiente il cui valore iniziale può essere modificato:

```
umask 022
alias cp="cp -f"
alias rm="rm -f"
alias mv="mv -f"
alias ln="ln -f"
#
#
DATA_DIRECTORY="$HOME/Documenti"
DATA_BACKUP_LEVELS="77"
DATA_NAME_DEFAULT="Nuovo"
DATA_EDITOR_COMMAND="luit -encoding ISO-8859-1 mcedit"
PRINTER_FILTER="\
| enscript -1 -M a4 -f CourierBold@9.1/9.5 \
    --margin=72:72:72:72 --header=||$% -o -
PRINTER_COMMAND="lpr"
IMPORT_EXPORT_MOUNT_POINTS_LIST="\
/mnt/a dischetto \
/mnt/d penna_USB"
IMPORT_EXPORT_DEVICES_LIST="/dev/fd0 dischetto"
```

La tabella successiva riepiloga il significato di queste variabili di ambiente:

Variabile	Descrizione
`DATA_DIRECTORY`	La directory da usare per la gestione dei file di testo.
`DATA_BACKUP_LEVELS`	Il numero di livelli per la conservazione delle versioni precedenti.
`DATA_NAME_DEFAULT`	La radice del nome predefinito da usare per i documenti nuovi.
`DATA_EDITOR_COMMAND`	Il comando da usare per accedere alla modifica di un file. Il nome del file viene fornito alla fine.
`PRINTER_FILTER`	Il comando per trasformare un file di testo in un file PostScript con un’impaginazione corretta, comprendente dei margini e possibilmente la numerazione delle pagine.
`PRINTER_COMMAND`	Il comando per stampare, in grado di ricevere il file dallo standard input.

Variabile	Descrizione
`IMPORT_EXPORT_MOUNT_POINTS_LIST`	L'elenco dei punti di innesto per inserire unità di memorizzazione esterne. Queste directory devono essere state previste nel file '`/etc/fstab`', in modo da consentire all'utente in questione di poter eseguire il comando '**`mount`**'. L'elenco è composto da coppie di «parole», dove la prima è la directory e la seconda è la descrizione da visualizzare nel menù.
`IMPORT_EXPORT_DEVICES_LIST`	L'elenco dei file di dispositivo delle unità di memorizzazione esterne da usare per il trasferimento dei dati. Attraverso questi nomi è possibile inizializzare tali unità, ammesso che l'utente disponga dei permessi necessari. L'elenco è composto da coppie di «parole», dove la prima è il file di dispositivo e la seconda è la descrizione da visualizzare nel menù.

18.11.1 Gestione dei file di testo

«

Per la gestione dei file di testo, ci si avvale di un programma esterno, definito dalla variabile di ambiente ***DATA_EDITOR_COMMAND***, usato solo per modificare un file, mentre il resto può essere gestito esternamente, attraverso le voci previste nel menù.

La creazione di un nuovo documento, si ottiene con la selezione della voce *new*, a cui segue la richiesta di specificare il nome che si preferisce:

```
.--crea un documento nuovo---.
| Inserisci il nome del      |
| documento da creare:       |
| .------------------------. |
| |Nuovo-1                 | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

Inizialmente viene proposto un nome predefinito, in base all'indicazione della variabile di ambiente ***DATA_NAME_DEFAULT***. Il nome può essere modificato e lo si può scrivere come si vuole: se il nome contiene simboli diversi dalle lettere alfabetiche latine, da numeri e da trattini ('**-**' o '_'), questo viene aggiustato in qualche modo; se il nome esiste già, viene modificato in modo da evitare il conflitto. Il file viene creato vuoto e viene avviato il programma di modifica per consentire la scrittura di ciò che si desidera.

Per visualizzare il contenuto di un documento, ovvero di un file di testo contenuto nella directory prevista, si può selezionare la voce *view*, ottenendo così la richiesta di selezionare il nome di questo:

```
.-visualizza un documento gia' esistente-.
| Seleziona il documento da              |
| visualizzare:                          |
| .------------------------------------. |
| |       appunti-1            .       | |
| |       appunti-2            .       | |
| |       da-fare              .       | |
| |       cose-vecchie         .       | |
| '------------------------------------' |
|----------------------------------------|
|         <  OK  >     <Cancel>          |
'----------------------------------------'
```

Si osservi che dall'elenco sono esclusi volutamente i nomi che contengono una tilde ('**~**'), che normalmente rappresenta una copia di sicurezza generata dal programma usato per modificare i file.

La modifica si ottiene selezionando la voce *edit* e il comportamento è simile a quanto già descritto a proposito della visualizzazione, con la differenza che viene avviato il programma per modificare i file di testo. Si presume che l'utente si limiti a salvare, senza modificare il nome, altrimenti questo script potrebbe essere inutile.

```
.-modifica un documento gia' esistente-.
| Seleziona il documento da              |
| modificare:                            |
| .------------------------------------. |
| |       appunti-1            .       | |
| |       appunti-2            .       | |
| |       da-fare              .       | |
| |       cose-vecchie         .       | |
| '------------------------------------' |
|----------------------------------------|
|         <  OK  >     <Cancel>          |
'----------------------------------------'
```

Selezionando dal menù la voce '**copy**' è possibile fare una copia di un file esistente, all'interno della stessa directory. In pratica, questa copia potrebbe servire per realizzare un documento a partire dai contenuti di un altro già scritto in precedenza. Viene richiesto di selezionare il file da copiare, quindi viene chiesto il nome da assegnare: se il nome nuovo va in conflitto con un altro file già esistente, la copia viene annullata.

```
.-------copia un documento-------.
| Seleziona il documento da      |
| copiare:                       |
| .----------------------------. |
| |     appunti-1            . | |
| |     appunti-2            . | |
| |     da-fare              . | |
| |     cose-vecchie         . | |
| '----------------------------' |
|--------------------------------|
|     <  OK  >     <Cancel>      |
'--------------------------------'
```

In questo esempio si suppone di avere selezionato il nome *appunti-1*, perciò è lo stesso nome '**appunti-1**' che viene proposto inizialmente per la copia, ma ovviamente deve essere modificato:

```
.-----copia un documento-----.
| Inserisci il nome per la   |
| copia del documento:       |
| .------------------------. |
| |appunti-1               | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

Dal menù è possibile selezionare la voce *rename* per essere guidati alla modifica del nome di uno dei file che compongono i documenti. Come già per la copia, viene richiesto di selezionare un nome esistente, quindi viene offerta una mascherina per inserire il nome nuovo. Il nome che si attribuisce non può essere uguale a uno già presente; in caso contrario, l'operazione non viene eseguita.

```
.---cambia nome a un documento---.
| Seleziona il documento da      |
| rinominare:                    |
| .----------------------------. |
| |     appunti-1            . | |
| |     appunti-2            . | |
| |     da-fare              . | |
| |     cose-vecchie         . | |
| '----------------------------' |
|--------------------------------|
|     <  OK  >     <Cancel>      |
'--------------------------------'
```

```
.-cambia nome a un documento-.
| Inserisci il nome nuovo    |
| per il documento:          |
| .------------------------. |
| |appunti-1               | |
| '------------------------' |
|----------------------------|
|   <  OK  >  <Cancel>       |
'----------------------------'
```

Per cancellare un file è possibile selezionare la voce *delete*, a cui segue la richiesta di selezionare il nome da eliminare. Prima di procedere alla cancellazione, una copia del file viene conservata nelle directory usate per le versioni precedenti.

```
.-elimina un documento che non serve piu'-.
| Seleziona il documento da cancellare    |
| .-------------------------------------. |
| |     appunti-1            .          | |
| |     appunti-2            .          | |
| |     da-fare              .          | |
| |     cose-vecchie         .          | |
| '-------------------------------------' |
|-----------------------------------------|
|          <  OK  >      <Cancel>         |
'-----------------------------------------'

.--elimina un documento che non serve piu'--.
| Vuoi cancellare il documento "appunti-1"? |
|                                           |
|-------------------------------------------|
|          <  Yes  >      <  No  >         |
'-------------------------------------------'
```

18.11.2 Copie delle versioni precedenti

Lo script prevede che per alcune operazioni delicate venga fatta la copia dei documenti che devono essere modificati, cancellati o sostituiti. Questa copia viene eseguita all'interno di sottodirectory con il nome '`~backup.`***n***`/`', dove la tilde ('`~`') fa parte del nome e il numero finale rappresenta il livello della copia. In pratica, la sottodirectory '`~backup.1/`' contiene le copie più recenti.

Il numero di livelli di copia è definito dalla variabile di ambiente ***DATA_BACKUP_LEVELS*** e può essere modificato liberamente: lo script, a ogni avvio, verifica la presenza di queste sottodirectory e in caso siano assenti le crea senza fare domande all'utente.

Lo script non prevede un sistema di recupero dei documenti nelle loro versioni precedenti. Eventualmente, per questo, l'utente deve chiedere aiuto a persona più preparata.

18.11.3 Esportazione e importazione dati

Pur trattandosi di un lavoro molto semplice, si presume che la persona che lo utilizza abbia la necessità di trasferire i propri documenti da e verso un disco rimovibile. Per poter fare questo, è necessario che sia stato previsto nel file '`/etc/fstab`' il modo di accedere a dei dischi esterni anche a un utente comune. In base a questa configurazione è necessario modificare la variabile di ambiente ***IMPORT_EXPORT_MOUNT_POINTS_LIST***.

La variabile di ambiente ***IMPORT_EXPORT_MOUNT_POINTS_LIST*** deve contenere un elenco di coppie di «parole», secondo la forma:

directory_innesto descrizione

Per esempio, ammesso che la directory '`/mnt/fd0/`' serva per innestare un dischetto e che la directory '`/mnt/sda1/`' serva per innestare la prima partizione di un'unità USB, si potrebbe inizializzare la variabile nel modo seguente:

```
IMPORT_EXPORT_MOUNT_POINTS_LIST="/mnt/a dischetto /mnt/d penna_USB"
```

Si osservi che non si possono inserire spazi nella descrizione.

Sulla base di questo esempio, il file '`/etc/fstab`' potrebbe contenere le righe seguenti, dove si dà per scontato che il file system sia di tipo Dos-FAT:

```
/dev/fd0  /mnt/a  vfat  defaults,user,noauto,exec,sync,umask=0000  0  0
/dev/sda1 /mnt/d  vfat  defaults,user,noauto,exec,sync,umask=0000  0  0
```

Oltre a questo è necessario che l'utente possa inizializzare i dischetti, pertanto deve avere i privilegi per poterlo fare. Nella variabile di ambiente ***IMPORT_EXPORT_DEVICES_LIST*** si mette l'elenco dei file di dispositivo che l'utente può inizializzare. Anche in questo caso i nomi sono associati a una descrizione. Supponendo che sia concesso di inizializzare solo il dischetto, la variabile può essere inizializzata così:

```
IMPORT_EXPORT_DEVICES_LIST="/dev/fd0 dischetto"
```

Anche per questa situazione, non si possono inserire spazi nella descrizione.

Attraverso le voci del menù è possibile: ottenere l'elenco del contenuto di un'unità di memorizzazione esterna; vedere il contenuto di un file di tale unità; cancellare il contenuto dell'unità esterna o inizializzarla. Inoltre, è possibile copiare file da o verso l'unità esterna.

Per tutte queste funzioni è richiesto di specificare l'unità esterna. In tutti i casi, escluso quello dell'inizializzazione, viene mostrato l'elenco contenuto nella variabile di ambiente ***IMPORT_EXPORT_MOUNT_POINTS_LIST***, mentre per l'inizializzazione vale l'altra variabile (***IMPORT_EXPORT_DEVICES_LIST***).

Viene mostrato il caso dell'esportazione, che si ottiene con la voce *export* del menù. Si comincia dalla selezione dell'unità che deve accogliere la copia:

```
.esporta dei documenti in un disco rimovibile--.
| Seleziona l'unita' da usare per              |
| l'esportazione dei documenti:                |
| .-------------------------------------------. |
| |            /mnt/a  dischetto              | |
| |            /mnt/d  penna_USB              | |
| '-------------------------------------------' |
|-----------------------------------------------|
|          <  OK  >      <Cancel>               |
'-----------------------------------------------'
```

Si selezionano i documenti da esportare, con l'aiuto della barra spaziatrice:

```
.esporta dei documenti in un disco rimovibile--.
| Seleziona o deseleziona i documenti da       |
| esportare, con l'aiuto della barra           |
| spaziatrice:                                 |
| .-------------------------------------------. |
| |      [X] appunti-2            .           | |
| |      [X] da-fare              .           | |
| |      [ ] cose-vecchie         .           | |
| '----------------v(+)----------------------' |
|-----------------------------------------------|
|          <  OK  >      <Cancel>               |
'-----------------------------------------------'
```

Alla conferma inizia la copia; se ci sono file con lo stesso nome, lo script verifica se il contenuto è lo stesso, altrimenti chiede conferma per procedere alla sovrascrittura.

L'importazione avviene nello stesso modo, con la differenza che se viene richiesta la sovrascrittura, i documenti precedenti vengono salvati attraverso il sistema di rotazione delle versioni.

18.11.4 Stampa

La stampa si ottiene selezionando la voce *print* dal menù principale. A questo segue la richiesta di selezionare un solo documento dall'elenco di quelli esistenti. La stampa avviene con i comandi indicati nelle variabili di ambiente ***PRINTER_FILTER*** e ***PRINTER_COMMAND***.

18.12 Riferimenti

- Richard Günther, *Kernel Support for miscellaneous (your favorite) Binary Formats*, '***sorgenti_linux***`/Documentation/binfmt_misc.txt`'
- Brian A. Lantz, Richard Günther, *Java(tm) Binary Kernel Support for Linux*, '***sorgenti_linux***`/Documentation/java.txt`'

[1] **GNU findutils** GNU GPL

[2] **util-linux: getopt** UCB BSD

[3] **Debianutils: tempfile** UCB BSD
[4] **GNU core utilities** GNU GPL
[5] **GNU core utilities** GNU GPL
[6] **GNU core utilities** GNU GPL
[7] **GNU core utilities** GNU GPL
[8] **Dialog** GNU GPL
[9] **Whiptail** GNU LGPL
[10] **Xdialog** GNU GPL
[11] **Gdialog (Zenity)** GNU GPL
[12] **Ncurses** software libero con licenza speciale FSF
[13] **Newt** GNU LGPL
[14] **Kaptain** GNU GPL
[15] **GNU core utilities** GNU GPL
[16] **GNU core utilities** GNU GPL
[17] **GNU core utilities** GNU GPL
[18] **GNU core utilities** GNU GPL
[19] **GNU core utilities** GNU GPL
[20] **GNU core utilities** GNU GPL