Lo shell C di UNIX

Lo shell C, sviluppato dopo il Bourne, è stato progettato con l'idea di fornire un linguaggio con costrutti "vicini" a quelli del linguaggio di programmazione C.

Lo shell C presenta le seguenti caratteristiche:

- vari meccanismi per accedere ed usare le variabili
- uso di alias
- linguaggio di programmazione che supporta costrutti condizionali, ciclici, e di gestione delle interruzioni
- accesso ai comandi precedenti mediante un meccanismo di "storia"

Quando uno shell C è invocato la sequenza di "startup" dipende dal modo in cui è invocato (cioè login o non-login).

In particolare:

1. entrambi: esegue i comandi in `$HOME/.cshrc`, se esiste
2. login: esegue i comandi in `/etc/login`, se esiste
3. login: esegue i comandi in `$HOME/.login`, se esiste

Il file "`.login`" contiene tipicamente comandi che inizializzano variabili di ambiente come `$TERM` o `$PATH`.

Il file "`.cshrc`" contiene invece comandi che creano alias di uso comune.

Variabili semplici (1)

Lo shell C supporta sia le variabili locali che quelle di ambiente.

Una variabile locale può contenere uno o più valori. Nel primo caso è chiamate variabile *semplice*, nel secondo *lista*.

Per creare o assegnare una variabile semplice si usa il comando `set`, secondo la seguente sintassi:

```
set { name [ =word] }*
```

Se non si specifica nessun argomento, lo shell visualizza tutte le variabili locali.
Se non si specifica `word`, lo shell assegna valore nullo alla varaibile.
Se la variabile name non esiste, viene implicitamente creata.

Esempi:

```
% set flag                ...flag è una stringa nulla
% echo $flag
% set color = red
echo $color
red
% set name = Marco Rossi
% echo $name
Marco
% set name = "Marco Rossi"
% echo $name
Marco Rossi
% set
color    red
flag
name     Marco Rossi
...
...
```

Variabili semplici (2)

Oltre al meccanismo di accesso già introdotto (`$name`) lo shell C supporta i seguenti meccanismi alternativi:

- `${name}` sostituito dal valore della variabile `name`
- `${?name}` vale 1 se la variabile esiste, 0 altrimenti

Esempi:

```
% set verb = sing
% echo I like $verbing
verbing: Undefined variable!
% echo I like ${verb}ing
I like singing
% _
```

```
% more flag.csh
#
set flag
if (${?flag}) then
    echo flag is set
endif

% flag.csh
flag is set
% _
```

Liste (1)

Per assegnare una lista di valori ad una variabile si usa il comando `set` con la seguente sintassi:

```
set { name = ( { word}* )}*
```

Esempio:

```
% set colors = ( red yellow green )
% echo $colors
red yellow green
```

Per accedere ad una lista ci sono due modi, ciascuno realizzabile in due forme diverse:

- `$name[selector]`
- `${name[selector]}`

Entrambe le forme sono sostituite dal valore della lista il cui indice è specificato da "`selector`", il quale può essere:

  - un singolo numero
  - un intervallo, del tipo `start-end` (se si omette `start`, viene assunto pari a 1; se si omette `end` si assume pari all'indice dell'ultimo valore)
  - uno "`*`", che si riferisce a tutti gli elementi della lista

- `$#name`
- `${#name}`

Entrambe le forme sono sostituite dal numero di elementi di `name`.

Esempi:

```
% set colors = ( red yellow green )
% echo $colors[1]
red
% echo $colors[2-3]
yellow green
% echo $colors[4]
Subscript out of range.
% echo $#colors
3
```

Liste (2)

Per aggiungere un elemento alla fine della lista, è necessario eseguire una set in cui si specificano la variabile originale e l'elemento da aggiungere.

Sei si tenta di assegnare il valore direttamente, verrà generato un messaggio di errore.

Esempi:

```
% set colors = ( red yellow green )
% set colors[4] = pink
Subscript out of range
% set colors = ( $colors blue )
% echo $colors
red yellow green blue
% set colors[4] = pink
% echo $colors
red yellow green pink
% set girls = ( sally georgia)
% set boys = ( harry blair )
% set both = ( $girls $boys )
% echo $both
sally georgia harry blair
```

Lo shell C supporta un certo numero di variabili predefinite:

`$<`	la linea successiva dello standard input
`$cwd`	la directory corrente
`$home`	la home directory dello shell
`$status`	il codice di uscita del comando precedente

Esempio:

```
% more prova.csh
#
echo -n "Inserire il nome: "
set name = $<           # legge una linea da stdin
echo Ciao $name, la tua directory corrente è $cwd
% prova.csh
Inserire il nome: Pluto
Ciao Pluto, la tua directory corrente è /usr/disney
```

Espressioni su stringhe

Lo shell C supporta espressioni di tre tipi:

- su stringhe
- aritmetiche
- orientate al file

Lo shell C supporta i seguenti operatori su stringhe:

`==`	restituisce TRUE se le due stringhe sono esattamente uguali
`!=`	restituisce TRUE se le due stringhe sono differenti
`=~`	come ==, con la differenza che l'operando di destra può contenere "wildcard" (cioè "`*`", "`?`", etc.)
`!~`	come `!=`, con la differenza che l'operando di destra può contenere "wildcard" (cioè "`*`", "`?`", etc.)

Esempi:

```
% more expr1.csh
#
echo -n "Do you like the C shell? "
set reply = $<
if ( $reply == "yes" ) then
    echo You entered yes
else if ( $reply =~ y* ) then
    echo I assume you mean yes
endif

% expr1.csh
do you like the C shell? yeah!!!!!
I assume you mean yes
% _
```

Espressioni aritmetiche (1)

Lo shell C supporta i seguenti operatori aritmetici:

`!` negazione logica

`* / %` moltiplicazione, divisione, resto

`+ -` addizione, sottrazione

`<= >= < >` operatori relazionali

`== !=` eguaglianza, diseguaglianza

`& ^ |` and, xor, e or a bit

`|| &&` or logico, and logico

Questi operatori operano esattamente come quelli del linguaggio C, con la differenza che questi lavorano soltanto su interi.

Tutte le espressioni che usano gli operatori `&`, `&&`, `||`, `|`, `<`, `>` devono essere racchiuse fra parentesi tonde, per evitare che lo shell li interpreti erroneamente.

Esempio:

```
% more expr2.csh
#
set a = 3
set b = 5
if ( $a > 2 && $b > 4 ) then
    echo e' tutto OK!!!
endif

% expr2.csh
e' tutto OK!!!
% _
```

Espressioni aritmetiche (2)

Per assegnare il risultato di una espressione ad una variabile si usa il comando `@`, il quale può assumere le seguenti forme:

`@`	visualizza tutte le variabili shell
`@ var op expr`	calcola `expr` e pone il risultato in `var` tenendo conto di `op`
`@ var[index] op expr`	pone il risultato di `expr` in `var[index]` tenendo conto di `op`

dove `op` può essere: `=`, `+=`, `-=`, `*=`, `/=`.

Esempi:

```
% set a = 2 * 2
set: Syntax error.

% @ a = 2 * 2
% echo $a
4

% @ a = $a + $a
% echo $a
8

% set flag = 1
% @ b = ( $a && $flag )     ... parentesi necessarie
% echo $b
1
```

E' anche possibile incrementare/decrementare una variabile usando gli operatori `++`/`--`.

Esempio:

```
% set value = 1
% @ value ++
% echo $value
2
```

Espressioni orientate al file

Lo shell C supporta espressioni orientate al file, secondo la seguente sintassi:

```
-option filename
```

Lo shell ritorna un valore TRUE (1) se l'opzione selezionata è vera, e zero altrimenti.

Le opzioni supportate sono le seguenti:

`r`	lo shell ha il permesso di accedere su `filename` in lettura
`w`	lo shell ha il permesso di accedere `filename` in scrittura
`x`	lo shell ha il permesso di accedere `filename` in esecuzione
`e`	`filename` esiste
`o`	`filename` è di properietà dello stesso utente dello shell
`z`	`filename` esiste ma la sua dimensione (in byte) è 0
`f`	`filename` è un file regolare
`d`	`filename` è una directory

Esempio:

```
% more expr3.csh
#
echo -n "Nome del file da cancellare: "
set name = $<
if ( ! (-w "$name") )
     echo Non hai il permesso per cancellare $name
else
     rm $name
     echo File cancellato
endif

% expr3.csh
Nome del file da cancellare: /bin
Non hail permesso per cancellare /bin
% _
```

Alias

Lo shell C consente di creare e personalizzare comandi usando il comando `alias`, la cui sintassi è la seguente:

```
alias [word [string] ]
```

Il comando consente di usare la parola `word`, al posto di `string` in tutti i comandi successivi.

Se si omette `string`, lo shell visualizza la stringa corrispondente all'alias `word` (se esiste).
Se si omette `word`, lo shell visualizza una lista di tutti gli alias correnti.
Se string incomincia con `word`, non è rielaborata per evitare problemi di loop infiniti. Se invece `string` contiene `word` altrove, viene segnalato un errore.

Esempio:

```
% alias dir 'ls -aF'
% dir          ... equivalente a scrivere "ls -aF"
./       ../      prova1.c     prova1.o     dati/

% dir *.c
prova1.c

% alias dir
ls -aF

% alias ls "ls -aF"
% ls *.c
./       ../      prova1.c     prova1.o     dati/

% alias who "date; who"
Alias loop.                  ... segnala errore
% alias "date; /bin/who"     ... risolto!
Mon Apr 21 15:18:11 MET DST 1997
root         ttyp1       Apr 21 14:06
rossi        ttyp2       Apr 21 09:43
verdi        ttyp3       Apr 19 11:15
```

Per rimuovere un alias si usa il comando **`unalias`**.

Storia

Lo shell C mantiene traccia degli ultimi comandi mediante un meccanismo di "storia".

Per visualizzare la lista degli ultimi comandi si usa il comando **`history`**.

Il numero di comandi da memorizzare è contenuto nella variabile `history`.

Ci sono anche meccanismi per rieseguire comandi già lanciati.

!!	sostituito con il testo dell'ultimo comando
!num	sostituito con il testo del comando num
!prefix	sostituito con il testo dell'ultimo comando che che incomincia con prefix
!?substring?	sostituito con il testo l'ultimo comando che contiene "substring"

Strutture di controllo: foreach

Lo shell supporta una serie di comandi che consentono di sviluppare programmi ad alto livello.

I più usati sono:

- foreach
- goto
- if-then-else
- switch
- while

foreach-end

La sintassi del comando foreach è la seguente:

```
foreach name ( wordList )
    commandList
end
```

Il comando itera il valore `name` in `wordList`, ed esegue i comandi in `commandList` ad ogni step.

Un comando `break` all'interno del loop provoca un'uscita dal ciclo.

Un comando `continue` all'interno del loop provoca un salto allo step successivo.

Esempio:

```
% more foreach.csh
#
foreach color = ( red yellow green blue )
    echo one color is $color
end
% foreach.csh
one color is red
one color is yellow
one color is green
one color is blue
% _
```

Strutture di controllo: goto

Il comando goto consente di saltare ad un particolare punto del programma etichettato con una label.

La sua sintassi è:

`goto` `name`

e la sintassi dell'etichetta è:

name:

Esempio:

```
% more goto.csh
#
echo salto!
goto fine_script
echo Questo non verra' visualizzato...
fine_script:
echo Questo verra' visualizzato!

% goto.csh
salto!
Questo verra' visualizzato!
% _
```

Strutture di controllo: if-then-else

Ci sono due forme del comando `if`:

```
if ( expr ) command
```

In questo caso non è previsto il ramo else. Se `expr` è vera lo shell eseguirà il comando `command`.

Esempio:

```
% if ( 5 > 3 ) echo cinque e' maggiore di tre
cinque e' maggiore di tre
% _
```

La seconda forma è:

```
if ( expr1 ) then
     list1
else if ( expr2 ) then
     list2
else
     list3
endif
```

I rami `else` ed `else-if` sono opzionali.

Esempio:

```
% more if.csh
echo -n "Scrivi un numero: "
set number = $<
if( $number < 0 ) then
     echo negativo
else if ($number == 0 ) then
     echo zero
else
     echo positivo
endif

% if.csh
Scrivi un numero: 457
positivo
% _
```

Strutture di controllo: while

Consente di ripetere una sequenza di comandi finchè non si verifichi una condizione di uscita.

La sua sintassi è

```
while ( expr )
    commandList
end
```

Il sigificato è il seguente: `commandList` viene eseguita finche' `expr` diventa falsa.

Un comando `break` all'interno del loop provoca un'uscita dal ciclo.

Un comando `continue` all'interno del loop provoca un salto allo step successivo.

Esempio: (generazione di una tavola pitagorica)

```
% more multi.csh
#
set x = 1
while ( $x <= $1 )
     set y = 1
     while ( $y <= $1 )
          @ v = $x * $y
          echo -n $v "     "
          @ y++
     end
     echo ""
     @ x ++
end

% multi.csh 4
1     2     3     4
2     4     6     8
3     6     9     12
4     8     12    16
% _
```

Strutture di controllo: switch

Il costrutto switch consente di realizzare diramazione multiple nel flusso di programma.

La sua forma è:

```
switch ( expr )
    case pattern1:
        list
        breaksw
    case pattern2:
    case pattern3:
        list
        breaksw
    default:
        defaultlist
endsw
```

Il comando confronta `expr` (che deve essere di tipo stringa) con i pattern, finchè non trova una corrispondenza. In quel caso esegue i comandi del ramo associato.

Switch: un esempio

```
% more menu.csh
#
echo menu test program
set stop = 0
while( $stop == 0 )
    cat << ENDOFMENU

  1:   print the date
  2,3: print the current working directory
  4:   quit
ENDOFMENU
  echo ""
  echo -n "Your choice? "
  set reply = $<
  switch ( $reply )
    case "1":
        date
        breaksw
    case "2":
    case "3":
        pwd
        breaksw
    case "4":
        set stop = 1
        breaksw
    default
        echo Illegal choice!
        breaksw
  endsw
end

% menu.csh
menu test program

  1:   print the date
  2,3: print the current working directory
  4:   quit

Your choice? 1
Mon Apr 21 17:34:44 MET DST 1997

  1:   print the date
  2,3: print the current working directory
  4:   quit

Your choice? 4
%
```

Progetto

Realizzare uno script che sostituisca il comando `rm`.

La sua sintassi sarà

junk -lp {filename}*

Invece di rimuovere il file, il comando lo muove in una directory temporanea chiamata "`.junk`" nella propria home directory. Se non esiste la crea automaticamente.

L'opzione `l` consente di visualizzare il contenuto della directory "`.junk`".

L'opzione `p` consente di cancellare il contenuto di "`.junk`".