

Unix shell

Shell

Programma che viene eseguito all'atto del collegamento con il sistema e che resta attivo per tutta la durata della sessione di lavoro

La funzione principale della **shell** è di **interpretare** il linguaggio di **comando** attraverso il quale l'utente utilizza le risorse del sistema

La comunicazione tra l'utente e la **shell** avviene mediante i **comandi**

Shell

La comunicazione tra l'utente e la **shell** avviene mediante i **comandi**

La **prima parola** deve essere il nome di un **comando built-in** della shell

oppure

il nome di un **eseguibile**, cioè il nome di un file presente nel file system e dotato di permesso di esecuzione

I comandi

comando [argomento ...]

Gli argomenti possono essere:

- **opzioni o flag** (-)
- **parametri**

separati da almeno un separatore

Una volta interpretato il comando la **shell ricerca** nel **file system** un file con il nome uguale al primo nome sulla linea di **comando**

La **ricerca** avviene ordinatamente all'interno delle directory elencate nella variabile d'ambiente **PATH**

Esempio

```
$ echo $PATH
$ /usr/bin:/users/mariog:.
$ ps
sh: ps: No such file or directory
$ PATH=$PATH:/bin
$ export PATH
$ ps
PID TTY TIME CMD
2487 tty1 00:00:00 sh
2488 tty1 00:00:00 ps
$
```

Le shell di Unix

Unix offre la possibilità di lavorare con diverse **shell**:

/bin/sh

Unix System V Shell

(detta anche Bourne Shell)

/bin/csh

C Shell

/bin/ksh

Korn Shell

/bin/?sh

... Shell

Variabili di shell

La Shell offre la possibilità di definire delle variabili

Esempio

```
$ frutto=mela  
$ verbo=mangia  
$ nome=Luisa  
$ echo $nome $verbo una $frutto  
Luisa mangia una mela  
$
```

Esempio

```
$ frutto=mela
$ frutto=${frutto}banana
$ echo $frutto
melabanana
$ tipo="mela banana"
$ echo $tipo
mela banana
$
```

Variabili predefinite

Esistono delle **variabili** di shell **predefinite**

Alcune di queste sono modificabili

HOME Argomento di default per il comando `cd`,
inizializzato da login con il path della
home directory (**`/etc/passwd`**)

PATH Il **path** di ricerca degli eseguibili

PS1 stringa del **prompt**, di default " \$ "

Esempio

```
$ adesso=`date`  
$ echo $adesso  
Fri Jan 7 10:49:55 CEST 2000  
$ cat piatti  
risotto  
bistecca  
frutta  
$ contenuto=`cat piatti`  
$ echo $contenuto  
risotto bistecca frutta
```

Mario

*i caratteri di ritorno sono
sostituiti da spazi*

Script di shell

Programmazione della shell

Le operazioni di **configurazione** e **gestione**
di un sistema operativo sono
operazioni ripetitive



Automatizzare tali operazioni



Script di shell

Programmazione della shell

E' possibile scrivere programmi (script) nel linguaggio della shell

Gli script di shell sono semplici da modificare e verificare

Gli script sono interpretati: le istruzioni sono lette una alla volta ed eseguite dalla shell

```
$cat myscript
find . -name "*.bak" -exec rm -i {} \;
$./myscript
./lucidi.bak
./temp/script.bak
$
```

Argomenti degli script

E' possibile passare **argomenti** agli **script** di shell all'atto della chiamata

Tali **argomenti** sono **posizionali** e vengono indicati mediante un numero (**\$1**, **\$2**, **\$3...**)

\$0 contiene sempre il **nome** del file dello **script**

Argomenti degli script

Se si passano **meno argomenti**, quelli mancanti sono sostituiti con **stringhe vuote**

Il numero **massimo** di argomenti è **9**; per usarne di più:

- si utilizza il comando **shift** (elimina il primo e numera quelli restanti)
- si utilizza **\$*** ("tutti gli argomenti passati")

\$# è il **numero** di **parametri** passati dalla linea di comando (escluso **\$0**)

Strutture e comandi

All'interno degli script di shell è possibile utilizzare:

- strutture di controllo
- comandi built-in
- comandi esterni

Cicli for

```
for variabile in lista di valori  
    do esegui i comandi  
done
```

variabile assume in successione tutti i valori in *lista di valori*

Per ogni valore viene eseguita la sequenza di comandi compresa tra **do** e **done**

Esempio

```
for colore in giallo verde rosso
do
    echo colore $colore
done
```

Esempio

```
for file0 in `find . -name "lucidi04.pdf" -print 2> /dev/null`  
do  
    file $file0  
done
```

Istruzione **if**

if *condizione*

then *azione*

elif *condizione2*

then *azione2*

else *azione3*

fi

Viene eseguita **if** *condizione*: se vera, allora azione è eseguita.

Altrimenti se **elif** *condizione2* è vera viene eseguita *azione2* e il comando è completato

Altrimenti **else** *azione3* è eseguita, se presente

Esempio

```
if test $# -eq 0
  then echo "nessun argomento»&2
        exit 1
elif test ! -s $1
  then echo il file $1 non esiste»&2
        exit 1
else echo il file $1 esiste»&1
      exit 0
fi
```

Comando test

test *condizione*

l'utilità **test** valuta la *condizione* ritorna un valore 0 se vera, 1 altrimenti

test esegue tre tipo di controlli:

1. su **valori numerici**
2. su **tipi di file**
3. su **stringhe di caratteri**

test: valori numerici

test controlla la **relazione** che intercorre tra due **numeri**, rappresentabili anche mediante variabili di shell

$N <primitiva> M$

Le primitive sono:

-eq	i valori di M e N sono uguali
-ne	i valori di N ed M sono diversi
-gt	$N > M$
-lt	$N < M$
-ge	$N \geq M$
-le	$N \leq M$

Esempio

```
#!/bin/sh
utenti=`who | wc -l`
if test $utenti -lt 8
    then echo "Sono allocati meno di 8 utenti"
fi
```

test: tipi di file

E' possibile controllare le caratteristiche di un file

<primitiva> nomefile

Le primitive più comuni sono:

- s** esiste?
- f** è un file normale?
- d** è una directory?
- w** si hanno i permessi di scrittura?
- r** si hanno i permessi di lettura?

Esempio

```
if test -d $1  
  then echo "$1 rappresenta una directory"  
fi
```

test: stringhe

E' possibile confrontare due stringhe

$S <primitiva> R$

Le primitive disponibili sono:

$=$ le due stringhe sono uguali?

$!=$ le due stringhe sono diverse?

test: stringhe

E' possibile controllare l'esistenza di una stringa

<primitiva> S

Le primitive disponibili sono:

- z** la stringa S ha lunghezza nulla?
- n** la stringa S ha lunghezza non nulla?

Esempio

```
$ numero=1
$ number=01
$ nombre=" 1"
$ if test $numero -eq $number ; then echo ok; fi
ok
$ if test $numero = $number ; then echo ok; fi
$ if test $numero = $nombre ; then echo ok; fi
ok
$ if test "$numero" = "$nombre" ; then echo ok; fi
$ echo "$numero"
1
$ echo "$nombre"
1
$
```

Combinazione di più controlli

Per combinare **più espressioni** di controllo si possono utilizzare i due operatori **-a** e **-o**

L'operatore **-a** indica la funzione logica **AND** tra due espressioni: il risultato di tale funzione è **vero** solo se **entrambe** le espressioni sono **vere**

L'operatore **-o** indica la funzione logica **OR** tra due espressioni: il risultato di tale funzione è **vero** se **almeno una** delle due espressioni è **vera**

Esempio

Aggiungi il contenuto di **unfile** ad **altrofile**

```
#!/bin/sh
if test -w $2 -a -r $1
then cat $1 >> $2
    else "echo impossibile accodare"
fi
```

Cicli while

while *questo comando
ha avuto successo*
do *esegui questi
comandi*
done

```
if test $# -eq 0
then echo Uso: $0 file1...»&2
exit
fi
while test $# -gt 0
do if test ! -s $1
then echo $1 non esiste»&2
else cat $1
fi
shift
done
```

Istruzione until

until *questa condizione
diventa vera*
do *esegui questi comandi*
done

```
if test $# -eq 0
    then echo Uso: $0 file1...»&2
fi
until test $# -eq 0
do if test ! -s $1
    then echo $1 non esiste»&2
    else cat $1
    fi
    shift
done
```

Comandi true e false

Questi comandi si limitano a restituire un valore **VERO** (zero) e **FALSO** (diverso da zero) rispettivamente

Sono utili, ad esempio, per **realizzare cicli infiniti**

```
while true
```

```
do
```

```
    sleep 300
```

```
    lpstat
```

```
done
```

```
until false
```

```
do
```

```
    sleep 300
```

```
    lpstat
```

```
done
```

Esecuzione selettiva con case

case *stringa* **in**

stringa1)

se stringa = stringa1 esegui i comandi fino a ;; e ignora tutti i casi restanti

;;

stringa2)

se stringa = stringa2 esegui i comandi fino a ;; e ignora tutti i casi restanti

;;

...

esac

Comando `expr`

`expr` considera i suoi argomenti come un'espressione aritmetica da valutare

il **risultato** viene visualizzato sullo **standard output**

Gli operatori aritmetici che si possono usare sono:

+	addizione
-	sottrazione
*	moltiplicazione
/	divisione (parte intera del quoziente)
%	resto di una divisione

L'operatore `*` va differenziato dal relativo metacarattere

Esempio

```
$ expr 2 * 3
```

```
6
```

Si possono utilizzare altri operatori per effettuare operazioni su stringhe

Esempio

Riconoscere se un argomento di uno script è un'opzione

```
if expr "$1" : "-." >/dev/null
then echo L'argomento costituisce una opzione
else echo "L'argomento non costituisce una opzione"
fi
```

Passaggio di dati

E' possibile eseguire un programma interattivo da uno **script** di shell e **passare argomenti** al programma in maniera **trasparente**

Esempio

```
$ cat calcolatrice
```

```
bc -l <&0
```

```
$
```

Comando read

E' possibile utilizzare gli **script** in maniera **interattiva**

Esempio

```
echo "Dimmi qualcosa: \c"  
read cosa  
echo "Ti faccio l'eco: $cosa"
```

```
if conferma
then echo "Si"
else echo "No"
fi
conferma() {
    echo -n "Sei sicuro? (S)i/(N)o? [S] \c"
    read answer
    case $answer in
    s|S|")
        return 0
        ;;
    n|N)
        return 1
        ;;
    *)
        conferma
        ;;
    esac
}
```

Verifica degli script

`sh` accetta delle opzioni per la verifica delle procedure:

- v** visualizza i comandi prima di eseguirlo
- x** visualizza i comandi ed il valore delle variabili
- n** stampa i comandi senza eseguirli

Sommario

Per **automatizzare** l'esecuzione di **operazioni** ripetitive si possono usare gli **script** di shell

In tali **script** si possono utilizzare **funzioni built-in** e **comandi** di sistema

La **sintassi** degli script **varia** a seconda della shell