Definizione di processo

Informalmente, il termine processo viene usato per indicare un **programma in esecuzione**

Esecuzione sequenziale del processo; istruzioni eseguite una dopo l'altra

Differenza tra processo e programma

Programma: **entità passiva** che descrive le azioni da compiere

Processo: **entità attiva** che rappresenta l'esecuzione di tali azioni

In un sistema multitasking **più** processi possono essere in esecuzione ***"concorrentemente"***

RIENTRANZA DELLO SHELL

Uno shell è un programma che esegue i comandi, forniti da terminale o da file

Si invocano gli shell come i normali comandi eseguibili con il loro nome

sh [<filecomandi>]
csh [<filecomandi>]

Le invocazioni attivano un processo che esegue lo shell

Gli shell sono RIENTRANTI:

più processi possono condividere il codice senza errori ed interferenze

```
sh
sh
csh
ps      # quanti processi si vedono?
```

METACARATTERI

Lo SHELL riconosce caratteri speciali (WILD CARD)

***** una qualunque stringa di zero o più caratteri in un nome di file

? un qualunque carattere in un nome di file

[*ccc*]
un qualunque carattere, in un nome di file, compreso tra quelli nell'insieme. Anche **range** di valori: **[*c*-*c*]**

Per esempio **ls [q-s]*** lista i file con nomi che iniziano con un carattere compreso tra q e s

commento fino alla fine della linea

**** escape (segnala di *non* interpretare il carattere successivo come speciale)

Il comando **echo** scrive la stringa successiva

echo * *stampa tutti i nomi di file del direttorio corrente*

echo * *stampa il carattere asterisco '*'*

ls [a-p,1-7]*[c,f,d]?
elenca i file i cui nomi hanno come iniziale un carattere compreso tra 'a e 'p' oppure tra 1 e 7, e il cui penultimo carattere sia 'c', 'f', o 'd'

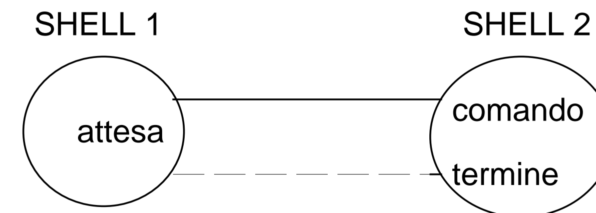
ESECUZIONE di COMANDI in SHELL

In UNIX ogni comando è eseguito da una nuova shell

La **shell attiva** mette in esecuzione una **seconda shell**

La **seconda shell**
- esegue le sostituzioni dei metacaratteri e dei parametri
- cerca il comando
- esegue il comando

Lo **shell padre** attende il completamento dell'esecuzione della sotto-shell (comportamento **sincrono**)



AMBIENTE DI SHELL (environment):
insieme di variabili di shell, per esempio:

- una variabile registra il **direttorio corrente**
- ogni utente specifica come fare la *ricerca dei comandi* nei vari direttori del file system: variabile **PATH** indica i direttori in cui cercare
- la variabile **HOME** indica il direttorio di accesso iniziale.

SCHEMA di un PROCESSORE COMANDI

procedure **shell** (ambiente, filecomandi);
< eredita ambiente (esportato) del padre, via copia>
begin
    repeat
        <leggi comando da filecomandi>
        if < è **comandoambiente**> *then*
            <modifica direttamente ambiente>;
        else if <è **comandoeseguibile**> *then*
            <esecuzione del comando via nuova shell>
        else if <è **nuovofilecomandi**> *then*
                shell (ambiente, nuovofilecomandi);
        else < errore>;
        endif
    until <fine file>
end shell;

Per abortire il comando corrente: CTRL-C

Non creare un file comandi ricorsivi senza una condizione di terminazione !!

Variabili nella shell

Ogni shell definisce:

- un insieme di variabili (trattate come stringhe) con **nome** e **valore**
- i riferimenti ai valori delle variabili si fanno con **$nomevariabile**
- si possono fare **assegnamenti**

nomevariabile=$nomevariabile
l-value *r-value*

Esempi:

```
x=123abc     # non si devono usare blank
echo $x      # visualizza 123abc
```

Sostituzioni della shell (parsing)

Prima della esecuzione, il comando viene scandito (*parsing*), alla ricerca di caratteri speciali (*, ?, $, >, <, |, etc.)

Come prima cosa, lo shell prepara i comandi come filtri:
ridirezione e piping di ingresso uscita
(su file o dispositivo)

Nelle successive scansioni, se la shell trova altri caratteri speciali, produce delle *sostituzioni*

1) **Sostituzione dei comandi**
I comandi contenuti tra **` `** (backquote) sono ***eseguiti*** e ne viene prodotto il risultato

```
echo `pwd`      # stampa il direttorio corrente
`pwd`           # tenta di eseguire /home/ptorroni
`pwd`/fileexe.exe    # e così ?
```

2) **Sostituzione delle variabili e dei parametri**
I nomi delle variabili (**$**nome) sono ***espansi*** nei valori corrispondenti

```
x=alfa          # non si devono usare blank
echo $x         # produce alfa
```

3) **Sostituzione dei nomi di file**
I metacaratteri ***, ?, [ ]** sono ***espansi nei nomi di file*** secondo un meccanismo di *pattern matching*

Sostituzioni della shell

Come si è visto, la shell opera con sostituzioni testuali sul comando e prepara l'ambiente di esecuzione per il comando stesso

Riassunto fasi

ridirezione e piping e le fasi di sostituzioni:
1) sostituzione comandi
2) sostituzione variabili e parametri
3) espansione dei nomi di file

Comandi relativi al controllo dell'espansione:

’ (quote)   nessuna espansione (né 1, né 2, né 3)

" (doublequote)   solo sostituzioni 1 e 2 (non la 3)

```
y=3
echo ' * e $y '     # produce * e $y
echo " * e $y "     # produce * e 3
echo "`pwd`"        # stampa nome dir corrente
```

sia **"** che **’** impediscono alla shell di interpretare i caratteri speciali per la ridirezione (< > >>) e per il piping (|)

Sostituzioni della shell: Esempi

Riassunto fasi
ridirezione e piping e le fasi di sostituzioni:
1) sostituzione comandi
2) sostituzione variabili e parametri
3) espansione dei nomi di file

Scansione della linea di comando con piu' passate successive (una per ciascuna fase).

Esempi:
$ **es='??'**
$ **$es**
tt: execute permission denied
$

shell esegue fasi 1, 2 (sostituzione di es con ??), 3 (sostituzione di ?? con il file del dir corrente tt) e prova quindi ad eseguire tt che non ha pero' i diritti di esecuzione

$ **rr='pwd'**
$ **echo $rr**
'pwd'

shell esegue fasi 1, 2 (sostituzione rr), 3, ed esegue quindi echo ''pwd''

PROGRAMMAZIONE NELLO SHELL

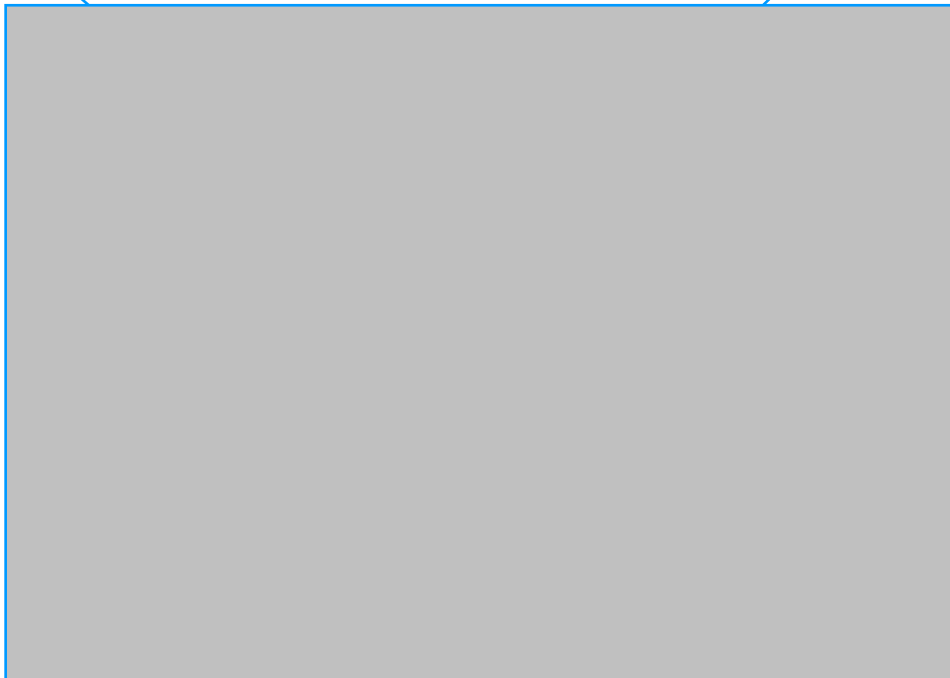
Il PROCESSORE COMANDI è in grado di elaborare comandi prendendoli da un file ➔ ***file comandi***

Linguaggio Comandi

Un ***file comandi*** può comprendere

- statement per il controllo di flusso
- variabili
- passaggio dei parametri

N.B. *quali* statement sono disponibili dipende da *quale shell* usiamo



VARIABILI

Sono disponibili **variabili** che contengono stringhe
NON è necessaria la definizione delle variabili

Il nome delle variabili è libero (alcune predefinite)
Il contenuto delle variabili è indicato dal metacarattere **$**

echo **$HOME**   # stampa il direttorio di default

echo PATH **$PATH**   **#** stampa PATH /:/bin:$HOME:.
(il carattere ':' è il separatore dei vari campi in PATH)

Assegnamento

<variabile>=<valore>   ***# niente spazi!***

i=12
echo i $i
j=$i+1
echo $j

La prima echo fornisce la stringa   i 12
la seconda echo fornisce la stringa   12+1

Le variabili sono trattate come stringhe di caratteri

v="ls -F" ; $v
v2="ls -lga a*"
$v2   # al momento dell'invocazione, viene espanso in
      # "tutti i file del dir. corrente che iniziano per a"

echo $v ; echo v ; echo `$v`

PASSAGGIO PARAMETRI

comando argomento1 argomento2 ... **argomentoN**

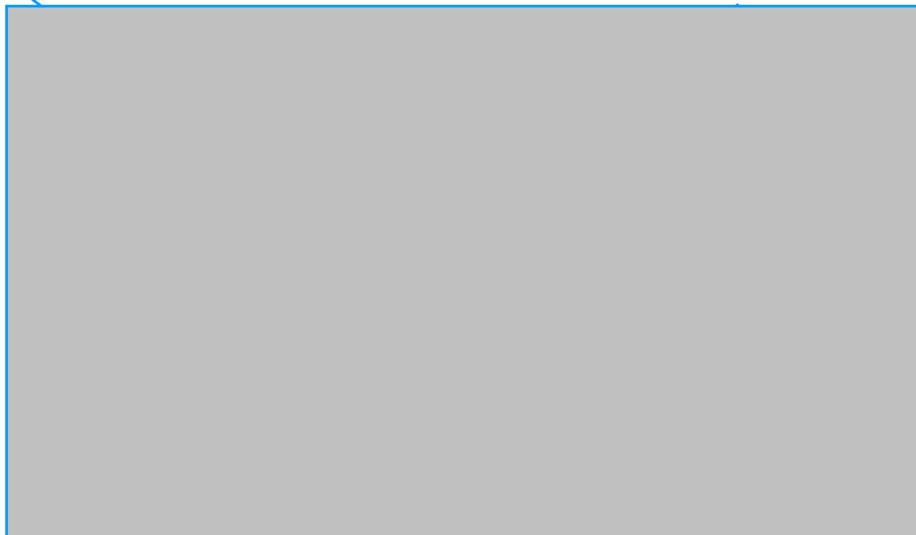
Gli argomenti sono **variabili posizionali** nella linea di invocazione

- **$0** rappresenta il comando stesso
- **$1** rappresenta il primo argomento
- ....

DIR /usr/utente1 (il file DIR contiene *ls $1*)
l'argomento $0 è DIR
l'argomento $1 è /usr/utente1

DIR1 /usr/utente1 "*" (DIR1 contiene *cd $1; ls $2)*
il direttorio è cambiato solo per la sotto-shell
produce la lista dei file del direttorio specificato

NB: * deve essere passato a ls senza essere sostituito
➔ virgolette! (cosa produrrebbe **DIR1 /usr/utente1 *** ?)



ALTRE VARIABILI

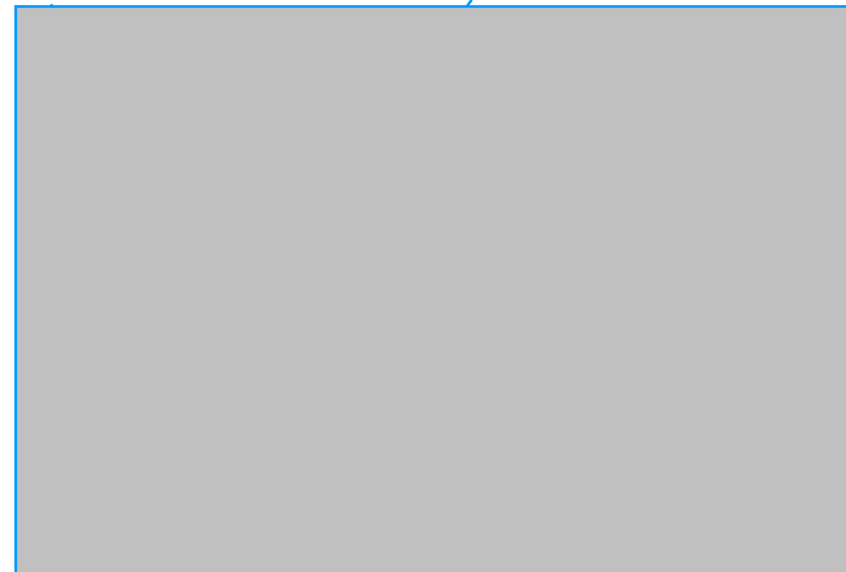
Oltre agli argomenti di invocazione del comando

$* l'insieme di tutte le variabili posizionali, che corrispondono agli argomenti del comando: $1, $2, ecc.
$# il numero di argomenti passati (***$0 escluso***)
$? il valore (int) restituito dall'ultimo comando eseguito
$$ l'identificatore numerico del processo in esecuzione (pid, proces identifier)

INPUT/OUTPUT

read var1 var2 var3 #input
echo var1 vale $var1 e var2 $var2 #output

read la stringa in ingresso viene attribuita alla/e variabile/i secondo corrispondenza posizionale



STRUTTURE DI CONTROLLO

Ogni statement in uscita restituisce un **valore di stato,** che indica **il completamento o meno del comando**
Tale valore di uscita è posto nella variabile ?

$? può essere riutilizzato in espressioni o per il controllo di flusso successivo

Stato **vale zero** ➔ comando OK
valore positivo ➔ errore

ESEMPIO 1

cp a.com b.com

se il comando non è riuscito (es: il file a.com *non* esiste) **allora** *errore* (valore > 0) **altrimenti** *successo* (valore 0)

$ **cp a.com b.com**

cp: cannot access a.com

$ **echo $?**

2

ESEMPIO 2

ls file
grep "stringa" file *# stato OK se trovato*
echo stato di ritorno $?

---

test

Comando per la **valutazione di una espressione**

test -<opzioni> <nomefile>

Restituisce uno stato uguale o diverso da zero

- valore **zero ➔ true**
- valore **non-zero ➔ false**

ATTENZIONE: convenzione opposta rispetto al C!

Motivo: i codici di errore possono essere più di uno e avere significati diversi

TIPI DI TEST POSSIBILI

test **-f** <nomefile> esistenza di file
-d <nomefile> esistenza di direttori
-r <nomefile> diritto di lettura sul file (**-w** e **-x**)

test <stringa1> **=** <stringa2> # uguaglianza stringhe
test <stringa1> !**=** <stringa2> # diversità stringhe

ATTENZIONE:

- gli **spazi intorno all' =** (o al !=) sono ***necessari***
- stringa1 e stringa2 possono contenere metacaratteri (attenzione alle espansioni, può essere necessario usare le virgolette ' oppure " a seconda dei casi)

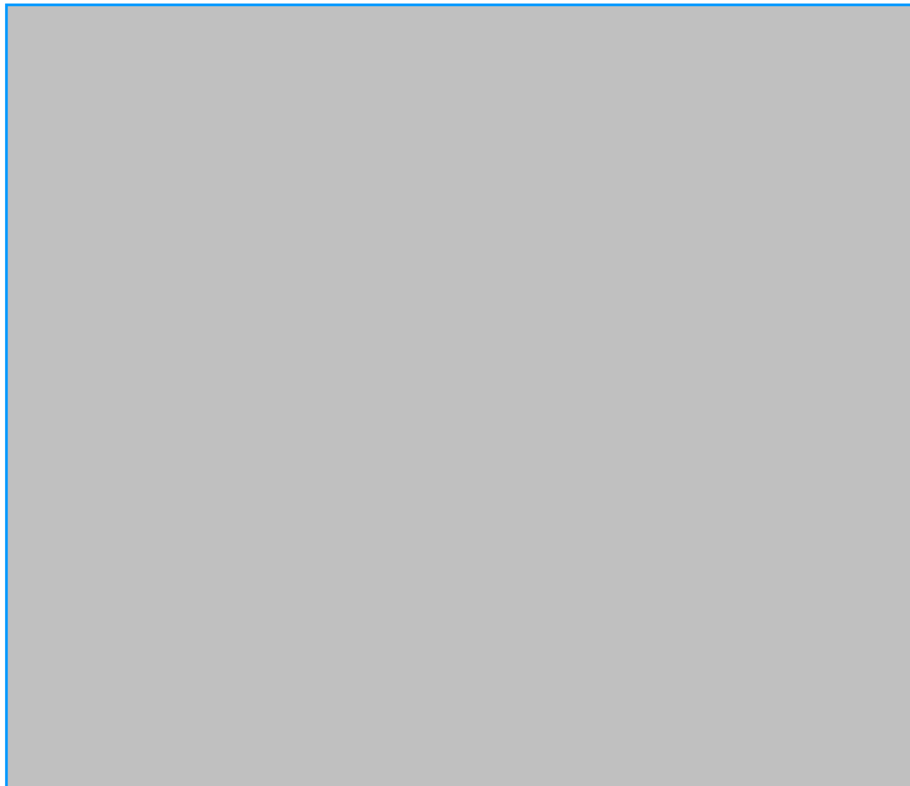
test **-z** <stringa> # vero se stringa è nulla
test <stringa> # vero se stringa *non* è nulla

TIPI DI TEST / segue

test <numero1> [**-eq -ne -gt -ge -lt -le**] <numero2>
confronta tra loro due stringhe numeriche, usando uno degli operatori relazionali indicati

Espressioni booleane

!	not
-a	and
-o	or



STRUTTURE DI CONTROLLO

ALTERNATIVA

if <lista-comandi>
  then
    <comandi>
  [**else** <comandi>]
fi

ATTENZIONE:

- le parole chiave (do, then, fi, etc.) devono essere o **a capo** o **dopo il separatore ;**
- if controlla il valore in uscita dall'ultimo comando di <lista-comandi>

Esempio

```
# fileinutile
# risponde “sì” se invocato con “si”
if test $1 = si -a $#  = 1
    then echo si
    else echo no
fi
```

Esempio di invocazione:

fileinutile si ➔ *stampa* si

Esempio (controllo numero argomenti)

```
if  test  $1; then echo OK
            else echo Almeno un argomento
fi
```

Esempio

file leggiemostra (uso: leggiemostra filename)

```
    read var1
    if test $var1 = si
    then
      if test -f $1
      then ls -lga $1; cat $1
      fi
    else echo niente stampa $1
    fi
```


ALTERNATIVA MULTIPLA

```
case <var> in    # alternativa multipla dip. da var

    <pattern-1>) <comandi> ;;
                    ...
    <pattern-i> | <pattern-j> | <pattern-k>) <comandi>;;
                    ...
    <pattern-n>) <comandi> ;;
esac
```

ESEMPI:

```
read  risposta
case  $risposta in
    S* | s* | Y* | y* )  < OK >;;
    * )                   <problema>;;
esac
```

```
#Ancora controllo dei parametri
case $# in
  0) echo Usage is: $0 file etc
     exit 2;; # si esce
  *) ;;
esac
```

RIPETIZIONI ENUMERATIVE

for <var> [**in** <list>] # list = lista di stringhe
do
<comandi>
done

scansione della lista <list> e ripetizione del ciclo per ogni stringa presente nella lista

ESEMPIO

for i **in** *
esegue per tutti i file nel direttorio corrente

for i # cioè **in $***
esegue per tutti i parametri di invocazione

for i **in** `ls s*`
do <comandi>
done

for i **in** `cat file1`
do <comandi per ogni parola del file *file1*>
done

#file **crea**
for i # cioè in $*
do > $i; # ridirezione di input su $i con chiusura
done

NB: il comando >file crea il file “file” di 0 byte

RIPETIZIONI NON ENUMERATIVE

while <lista-comandi>
do
<comandi>
done

Si ripete per tutto il tempo che il valore di stato dell’ultimo comando della lista è zero (successo); si termina quando tale valore diventa diverso da zero.

file esiste (cicla fino a comparsa file di nome $1)
invocazione esiste nomefile
while test ! -f $1 # ls **non** ritorna 1 se non trova il file
do sleep 10; echo file assente
done

- exit [**status**]: funzione primitiva di UNIX

Comandi per lo sviluppo di un programma

CC o GCC: Compilatore C (+ linker)

cc -o prog -g nomefile1 ... nomefileN

- I file oggetto hanno estensione **.o**
- Opzione **-o** → specifica nome dell'eseguibile (qui, **prog**) [default: **a.out**]
- I vari *nomefile1* ... *nomefileN* possono essere anche dei file oggetto (.o) → vengono solo linkati (non compilati)
- Il qualificatore **-g** aggiunge le tabelle per il debugger

FILE COMANDI

la parte di controllo degli argomenti è fondamentale

E' necessario **verificare gli argomenti**

- devono essere **innanzitutto *nel numero giusto***
- **poi *del tipo richiesto***

```
# invocazione  di comando per ...
case $# in
0|1|2|3|4)    echo Errore. Almeno 4 argomenti ... &> 2
              exit 1;;
esac
#
# in caso di argomenti corretti:
# primo argomento: dev'essere un direttorio (o file)
    if test ! -d $1
     then echo argomento sbagliato: $1 direttorio; exit 2
    fi;;
#
# primo argomento: se è un nome di file assoluto
case $1 in
    /*)  if test ! -f $1
            then echo argomento sbagliato: $1 file &> 2; exit 2
         fi;;
    * ) echo argomento sbagliato: $1 assoluto; exit 3;;
 esac
#
# primo argomento: se è un nome di file relativo
case $1 in
   */*) echo argomento sbagliato: $1 nome relativo; exit 3;;
    * ) ;;
esac
```