

Laboratorio di Informatica di Base

Laurea in Informatica Multimediale

Docente: *Massimo Merro*
prof.sci.univr.it/~merro

Lucidi a cura di

Andrea Colombari, Carlo Drioli, Andrea Fusiello e Barbara
Oliboni

Lezione 5

Ambiente shell



Testo di riferimento:
M. Bertacca, e A. Guidi
“Introduzione a Linux”
McGrawHill

Modalità di funzionamento shell

- La shell ha tre modalità di funzionamento:
 - **Interattiva:**
La shell attende i comandi digitati dall'utente.
 - **Di configurazione:**
La shell viene utilizzata per definire variabili e parametri d'utente e di sistema.
 - **Di programmazione:**
La shell viene adoperata per realizzare **procedure**, dette **script**, contenenti costrutti di comandi/istruzioni di GNU/Linux.

Istruzioni della shell

- La **bash** accetta un certo numero di istruzioni (oltre ai comandi del s.o.)
- Ogni istruzione:
 - inizia con una parola chiave
 - può avere uno o più argomenti
 - viene chiusa da un ritorno a capo o da ;

Esempio:

```
$ echo esempio di echo
esempio di echo
$
```

```
$ echo uno; echo due;
uno
due
$
```

Variabili della shell

- Le istruzioni operano su variabili
- Una variabile della shell è un **contenitore** che ha un **nome**
- Il nome
 - non può contenere caratteri speciali (?, *, ecc.).
 - è case-sensitive, cioè maiuscole e minuscole sono diverse.
 - deve iniziare con una lettera o con underscore (_)
- Il contenuto è una *stringa*, ovvero una sequenza di caratteri

Assegnamento di una variabile

- Per inserire un valore in una variabile si usa l'istruzione di **assegnamento**, che corrisponde al simbolo '='. Non inserire spazi tra il nome della variabile, l'uguale e il valore da inserire.

```
$ VARIABILE1=valore1
```

```
$ VARIABILE1 = valore1
```

Errore!

- Se il valore da dare contiene uno spazio è indispensabile racchiudere il valore tra doppi apici (es: "valore con spazi")
- Una variabile viene creata al momento del suo primo assegnamento e rimane in memoria fino a che la shell rimane attiva.

Assegnamento di una variabile (2)

- Una variabile può assumere il valore speciale NULL, cioè il nulla, e per assegnare tale valore si può fare in due modi:

```
$ VARIABILE=
```

```
$ VARIABILE=""
```

- Esempi:

```
$ VARIABILE1=valore1  
$ VARIABILE2=""valore 2"
```

Note sull'uso di una variabile

- Per **accedere al contenuto** di una variabile si utilizza il '\$'. Questo permette di differenziare il semplice testo dal nome di variabili.
- Se si vuole accostare del testo a quello contenuto in una variabile è necessario **delimitare il nome della variabile** usando le graffe (es: \${var} testo).
- Per **vedere/stampare il contenuto** di una variabile si può usare il comando **echo**.

Esempio:

```
$ echo $VARIABLE2
valore 2
$ echo ${VARIABLE1}${VARIABLE2}000
valore1valore 2000
```


Variabili d'ambiente

- Le **variabili normali** sono visibili solo nella shell dove vengono dichiarate e il loro contenuto non è visibile dai processi lanciati dalla shell.
- **Variabili d'ambiente**
 - Possono essere associate ad un processo e sono visibili anche ai processi figli.
 - Possono essere usate per modificare il comportamento di certi comandi, senza dover impostare ripetutamente le stesse opzioni.
- Le **variabili normali** possono diventare **variabili d'ambiente** tramite l'istruzione **export**

Esempio:

```
$ export VARIABILE1  
$
```

Variabili d'ambiente (2)

- Quando ci si connette al sistema, alcune **variabili d'ambiente** vengono inizializzate con valori di default (modificabili solo dall'amministratore del sistema).
- Le principali **variabili d'ambiente** sono:
 - **HOME**: contiene il path assoluto della home dell'utente che ha fatto login.
 - **MAIL**: contiene il path assoluto di dove sono contenute le email dell'utente che sta usando la shell.
 - **PATH**: contiene la lista di directory, separate dai due punti, dove il sistema va a ricercare comandi e programmi.
 - **MANPATH**: lista di directory, separate dai due punti, per la ricerca delle pagine man da parte del comando man.
 - **PS1**: contiene la forma del prompt primario.
 - **PS2**: contiene la forma del prompt secondario.

Variabili d'ambiente (3)

- **SHELL**: contiene path assoluto e nome della shell in uso.
- **TERM**: contiene il nome che identifica il tipo di terminale in uso.
- **LOGNAME**: contiene il nome della login dell'utente che ha fatto login.
- **PWD**: contiene il path assoluto della directory corrente.
- L'utente può modificare a piacere il valore delle **proprie variabili d'ambiente**.

Variabili d'ambiente (4)

- Si può visualizzare la lista delle **variabili d'ambiente** con l'istruzione **env**

Esempio:

```
$ env  
HOME=/home/pippo  
LOGNAME=pippo  
MAIL=/var/spool/mail/pippo  
...  
$
```

Visualizzazione variabili

- Si può visualizzare la lista di **tutte le variabili** dichiarate nella shell con l'istruzione **set**

Esempio:

```
$ set
BASH=/bin/bash
BASH_VERSION=1.14.6(1)
...
HOME=/home/pippo
LOGNAME=pippo
MAIL=/var/spool/mail/pippo
...
SHELL=/bin/bash
TERM=linux
VARIABLE1=valore1
VARIABLE2=valore 2
$
```

Uso degli apici

- Una stringa racchiusa tra apici singoli non subisce espansione

```
$ echo '*$HOME*'
*$HOME*
```

- Una stringa racchiusa tra apici doppi subisce l'espansione delle sole variabili

```
$ echo "$HOME*"
*/home/pippo*
```

Uso degli apici (2)

- Un apice singolo o doppio può essere racchiuso tra apici sole se preceduto dal carattere di protezione \

```
$ echo 'Oggi e\' una bella giornata'  
Oggi e' una bella giornata  
$ echo "IL Linguaggio \"C\""  
IL Linguaggio "C"  
$
```

- Un apice può essere passato come argomento di un comando sole se preceduto dal carattere \'

```
$ echo \  
,  
$
```

Sostituzione comandi

- Trasforma in stringa il prodotto di un comando

- Formato:

```
`nome_comando`
```

```
$(nome_comando)
```

- Esempi:

```
$ DIR=`pwd`  
$ echo $DIR  
/home/pippo  
$
```

```
$ DIR=$(pwd)  
$ echo $DIR  
/home/pippo  
$
```

```
$ NOME=$(basename lenna.png)  
$ echo $NOME  
lenna  
$
```

```
$ NOME=$(basename lenna.png)  
$ echo $NOME  
lenna.png
```


Sostituzione processi

- Mette il prodotto del comando in un file e ne restituisce il nome
- Formato:

```
<(nome_comando)
```

- Esempio:

```
$ diff <(ls $DIR1) <(ls $DIR2)
```

```
$ echo <( ls .)
```

Codice di uscita di un comando

- Numero intero positivo compreso tra 0 e 255
 - Il codice di uscita è 0 se il comando svolge correttamente i propri compiti
 - Il codice di uscita è diverso da 0 altrimenti
- Il codice di uscita dell'ultimo comando lanciato dalla shell viene memorizzato nella variabile speciale **\$?**
- **Esempio:**

```
$ ls -l frase
-rw-r--r-- 1 pippo stud 332 Feb 23 17:40 frase
$ echo $?
0
$ ls -l canzone
ls: canzone: No such file o directory
$ echo $?
2
```

Lista di comandi

- Gruppo di comandi che la shell esegue in sequenza
- Connessione di comandi **incondizionata**
 - Tutti i comandi della lista vengono sempre eseguiti (a meno della terminazione della procedura)
 - Comandi su righe differenti o separati da **;**
- Connessione di comandi **condizionata**
 - Operatori **&&** e **||**

```
comando1 && comando2  
comando1 || comando2
```

Operatori && e ||

- **comando1 && comando2**
comando2 viene eseguito se e solo se **comando1** restituisce un codice di uscita **uguale a 0**
- **comando1 || comando2**
comando2 viene eseguito se e solo se **comando1** restituisce un codice di uscita **diverso da 0**

```
$ grep sole frase && echo " -->frase contiene 'sole'"
Il sole splende.
-->frase contiene 'sole'
$ grep luna frase || echo " -->frase non contiene 'luna'"
-->frase non contiene 'luna'
$
```

Operatori && e || (2)

- La shell scandisce sempre tutti i comandi, ma condiziona l'esecuzione verificando il codice di uscita
- **Esempio:**

```
$ grep luna frase &&  
> echo " -->frase contiene 'luna'" ||  
> echo " -->frase non contiene 'luna'"  
-->frase non contiene 'luna'  
$
```

Procedure (script)



Testo di riferimento:

M. Bertacca, e A. Guidi
“Introduzione a Linux”

McGrawHill

Procedure shell (shell script)

- Vengono usate nei programmi che interagiscono con il sistema operativo
 - Esempio: per semplificare le operazioni di installazione e /o configurazione di pacchetti software
- Il linguaggio shell comprende:
 - variabili locali e d'ambiente
 - operazioni di lettura/scrittura
 - strutture per il controllo del flusso di esecuzione: sequenziale, decisionale e iterativa
 - richiamo di funzioni con passaggio di parametri

Creare una procedura (script)

- Una **procedura** o **script**, non è altro che un file di testo contenente una serie di comandi da far eseguire alla shell.
- Passi sono per creare ed eseguire uno script:
 - Preparare lo script, che chiamiamo `mio_script`, mediante un elaboratore di testi (es: **emacs mio_script**)
 - Impostare i permessi per la sua esecuzione (es: **chmod +x mio_script**)
 - Far interpretare lo script alla shell:

```
$ ./mio_script  
$
```


Creare una procedura (script) (2)

- Il “./” davanti al nome serve per specificare il fatto che lo script si trova all’interno della cartella corrente.
- Se la directory corrente è nella variabile d’ambiente PATH, allora per eseguire lo script possiamo scrivere semplicemente:

```
$ mio_script  
$
```

Esempio di script

- Come primo esempio, vediamo uno script per scrivere sul terminale video la scritta “Ciao Mondo” avendo cura precedentemente di ripulire lo schermo. Creiamolo con **cat**:

```
$ cat > mio_script  
clear  
echo "Ciao Mondo"
```

- A questo punto premendo CTRL+D si specifica la fine del flusso di input (EOF), il comando **cat** termina e quanto inserito sullo standard input viene copiato sul file **mio_script**.

Esempio di script (2)

- Ora facendo un `ls -l` si può notare che i permessi di esecuzione mancano
- E' quindi necessario cambiare i permessi per poter eseguire lo script. Usiamo allora `chmod` per aggiungere il permesso di esecuzione, Controlliamo con `ls -l`
- A questo punto è possibile eseguire lo script
- Quello che accade è che viene ripulito il terminale, poi compare la scritta, seguita dal prompt.

Il comando **read**

- Il comando **read** legge una riga da standard input fino al ritorno a capo e assegna ogni parola della linea alla corrispondente variabile passata come argomento

```
$ read a b c
111 222 333 444 555
$ echo $a
111
$ echo $b
222
$ echo $c
333 444 555
$
```

- Il carattere separatore è definito dalla variabile **IFS** che per default contiene lo spazio

Uso di **read** in una procedura

- **Esempio:**
File **prova_read**
(N.B. deve essere eseguibile)

```
echo "dammi il valore di x"  
read x  
echo "dammi il valore di y"  
read y  
echo "x ha valore" $x  
echo "y ha valore" $y
```

```
$ ./prova_read  
dammi il valore di x  
15  
dammi il valore di y  
ottobre  
x ha valore 15  
y ha valore ottobre  
$
```

Parametri posizionali

- Valori passati alle procedure come argomenti sulla riga di comando
- Gli argomenti devono seguire il nome della procedura ed essere separati da almeno uno spazio

- **Esempio:** File **posizionali**

```
echo nome della procedura "$0"  
echo numero di parametri "$#" "  
echo parametri "$1 $2 $3 $4 $5 "$"
```

```
$ ./posizionali uno due tre  
nome della procedura [./posizionali]  
numero di parametri [3]  
parametri [ uno due tre ]  
$
```

Variabili `$*` e `$@`

- La variabile `$*` contiene una stringa con tutti i valori dei parametri posizionali
- La variabile `$@` contiene tante stringhe quanti sono i valori dei parametri posizionali

■ **Esempio:** File **argomenti**

```
./posizionali "$*"
./posizionali "$@"
```

```
$ ./argomenti uno due tre
nome della procedura [./posizionali]
numero di parametri [1]
parametri [ uno due tre ]
nome della procedura [./posizionali]
numero di parametri [3]
parametri [ uno due tre ]
$
```

Variabili \$* e S@ (2)

- Nel caso in cui gli argomenti contengano al loro interno degli spazi, l'uso di "\$@" è da preferirsi a "\$*" (o a "\$@" che è equivalente)

- Esempio: File **argomenti2**

```
./posizionali $*  
./posizionali "$@"
```

```
$ ./argomenti2 uno "uno e mezzo" due  
nome della procedura [./posizionali]  
numero di parametri [5]  
parametri [ uno uno e mezzo due ]  
nome della procedura [./posizionali]  
numero di parametri [3]  
parametri [ uno uno e mezzo due ]  
$
```


L'istruzione **set**

- **set** invocata senza argomenti permette di visualizzare tutte le variabili assegnate

- Forma generale di invocazione:

```
set [opzione|+opzione] [argomento ...]
```

- Gli **argumenti** passati diventano i parametri posizionali della procedura

- **Esempio:** File **prova_set**

```
set uno due tre  
echo $1 $2 $3
```

```
$/prova_set  
uno due tre
```

Alternative: il costrutto **if**

- Permette di controllare il flusso di esecuzione della procedura mediante un'alternativa

- Forma:

```
if lista_comandi
then
  lista_comandi
[else
  lista_comandi]
fi
```

- Un costrutto **if** ha un codice di uscita uguale a quello dell'ultimo comando eseguito
- I costrutti **if** possono essere annidati

Uso di **if** in una procedura

- Esempio: File **if1**

```
if ls $1
then
    echo "il file $1 esiste ..."
    if grep $2 $1
    then
        echo "... e contiene la parola $2!"
    else
        echo "... ma non contiene la parola $2!"
    fi
else
    echo "il file $1 non esiste!"
fi
$
```

Uso di **if** in una procedura (2)

- **Esempio:** Esecuzione del file **if1**

```
$ if1 frase sole
il file frase esiste ...
Il sole splende.
... e contiene la parola sole!
$ if1 frase luna
il file frase esiste ...
... ma non contiene la parola luna!
$
```

Il comando `test`

- `test` è un comando che permette di controllare il contenuto di una variabile
- Forma generale di invocazione:

```
test arg1 [arg2 ...]
```

- Gli argomenti del comando devono formare una espressione logica che viene controllata:
 - se risulta soddisfatta il comando torna 0
 - altrimenti torna un codice di uscita diverso da 0

Uso di **test** in una procedura

- Esempio: File **prova_test**

```
if test "$1" = si
then
    echo Risposta affermativa
elif test "$1" = no
then
    echo Risposta negativa
else
    echo Risposta indeterminata
fi
$
```

```
$ prova_test si
Risposta affermativa
```

Espressioni logiche su stringhe

- `stringa1 = stringa2`
vero se le stringhe sono uguali
- `stringa1 != stringa2`
vero se le stringhe sono diverse
- `-z stringa1`
vero se `stringa1` ha lunghezza 0
- `[-n] stringa1`
vero se `stringa1` ha lunghezza maggiore di 0

Il comando `test` (2)

- Il comando `test` ha un altro nome rappresentato da una parentesi quadra aperta (`[`)
- Quando si usa questa forma si deve aggiungere un argomento che deve essere una parentesi quadra chiusa (`]`)
- Esempio: File `prova_test1`

```
if [ "$1" = si ]
then
    echo Risposta affermativa
elif [ "$1" = no ]
then
    echo Risposta negativa
else
    echo Risposta indeterminata
fi
$
```


Composizione di espressioni logiche

- Operatori:
 - **-a** mette in AND due espressioni
 - **-o** mette in OR due espressioni
 - **!** nega l'espressione che segue

- Esempio: File **prova_test2**

```
if [ "$1" = si -o "$1" = SI ]
then
    echo Risposta affermativa
elif [ "$1" != no -a "$1" != NO ]
then
    echo Risposta negativa
else
    echo Risposta indeterminata
fi
$
```

Iterazioni: l'istruzione **for**

- Permette di eseguire un gruppo di comandi un determinato numero di volte, modificando ad ogni iterazione il contenuto di una variabile
- Forma:

```
for variabile_del_for [in lista_di_parole]
do
    lista_comandi
done
```

Uso di **for**

- Esempio:

```
$ for i in uno due tre 4 parola
> do
> echo Prova ciclo for: $i
Prova ciclo for: uno
Prova ciclo for: due
Prova ciclo for: tre
Prova ciclo for: 4
Prova ciclo for: parola
$
```

Uso di **for** (2)

- Esempio File **prova_for** :

```
for i
do
    Prova ciclo for: $i
done
```

```
$ prova_for uno due tre 4 parola
Prova ciclo for: uno
Prova ciclo for: due
Prova ciclo for: tre
Prova ciclo for: 4
Prova ciclo for: parola
$
```

Iterazioni: costruito **while**

- Permette di creare cicli condizionati
- Forma:

```
while lista_di_comandi1
do
    lista_di_comandi2
done
```

- I comandi di **lista_di_comandi1** vengono eseguiti, se l'ultimo ritorna 0, viene eseguita la parte di procedura tra **do** e **done**

Iterazioni: costruito **until**

- Permette di creare cicli condizionati
- Forma:

```
until lista_di_comandi1  
do  
  lista_di_comandi2  
done
```

- I comandi di **lista_di_comandi2** vengono eseguiti fino a quando l'esecuzione dell'ultimo comando in **lista_di_comandi1** restituisce 0

Calcoli

- La bash consente di valutare espressioni aritmetiche
- Le espressioni vengono considerate come se fossero racchiuse tra doppi apici, quindi le variabili vengono espresse prima dell'esecuzione dei calcoli
- Il risultato viene tornato come stringa
- Formati ammessi:

```
$(( espressione_aritmetica ))
```

```
#[ espressione_aritmetica ]
```

- Esempio:

```
$ b=7
$ echo $(( $b * 3 ))
21
$
```

Script per inizializzare l'ambiente

- L'utente può personalizzare le operazioni di inizializzazione dell'ambiente effettuate dal sistema ad ogni connessione.
- Ad ogni connessione la shell esegue una procedura.
 - La **bash** controlla nella home dell'utente la presenza del file **.bash_profile** e:
 - se esiste ne esegue tutte le righe
 - se non esiste la **bash** cerca il file **.bash_login**
 - se non esiste neanche questa esegue il file **.profile**
- Quindi, personalizzando tale script è possibile personalizzare il proprio ambiente di lavoro.

Script per inizializzare l'ambiente (2)

■ Un esempio di `.bash_profile`

```
# aggiunge al PATH la directory /etc e la directory
# bin contenuta nella propria home
PATH=$PATH:/etc:$HOME/bin

# crea la variabile MAIL, o se esiste la rimpiazza,
# inserendovi la directory mail presente nella propria home
MAIL=$HOME/mail

# imposta il prompt personalizzato con il nome utente
PS1=${LOGNAME}"> "

# directory contenente le mie lettere
export MIE_LETTERE=$HOME/lettere

# creazione di un alias del comando rm in modo che venga
# eseguito sempre con l'opzione -i
alias rm="rm -i"
```

Script per inizializzare l'ambiente (3)

- Si noti che il carattere **#** è utilizzato per inserire dei **commenti**, cioè testo che non viene interpretato dalla shell.
- Il **prompt** può essere personalizzato grazie all'impostazione della variabile **PS1**. Alcuni pattern utilizzabili:
 - **"\u"**: visualizza il nome utente.
 - **"\h"**: visualizza il nome della macchina (hostname).
 - **"\w"**: visualizza il percorso di dove vi trovate.
- Il comando **alias** permette di dare un "nome" ad una sequenza di comandi che, per esempio, usiamo spesso. Eseguito senza argomenti da la lista di tutti gli alias.
- Tipicamente vengono assegnate variabili usando la sostituzione di comandi, es:
`export OS=$(uname -s)`