Lezione 5
BASH

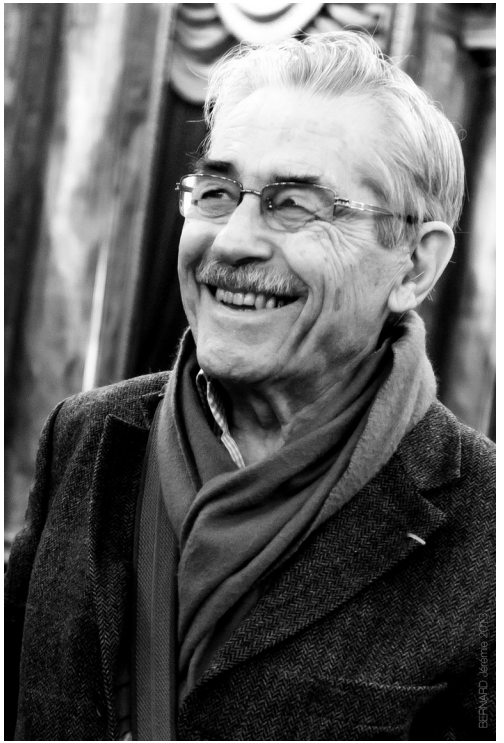
Sistemi Operativi (9 CFU), CdL Informatica, A. A. 2022/2023
Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Università di Modena e Reggio Emilia
http://weblab.ing.unimo.it/people/andreolini/didattica/sistemi-operativi

Quote of the day

(Meditate, gente, meditate…)

"… I felt that commands should be usable as building blocks for writing more commands, just like subroutine libraries. Hence, I wrote "RUNCOM", a sort of shell driving the execution of command scripts, with argument substitution.

Louis Pouzin (1931-)



Creatore di RUNCOM (prima shell in MULTICS)
Inventore del termine "shell"
Progettista di CYCLADES (prima rete ad adottare i datagrammi)

SOLUZIONI DEGLI ESERCIZI

Esercizio 1 (2 min.)

Aprite una nuova finestra di terminale.
Dichiarate una variabile di nome **`a`** con attributo intero e valore pari a **`8/3`**.
Stampate il valore della variabile. Notate qualcosa di strano?

Definizione e stampa della variabile

Si dichiara una variabile di nome **a** con attributo intero e valore pari a **8/3**:

```
declare -i a=8/3
```

Si stampa il valore di **a**:

```
echo $a
```

Si ottiene l'output seguente:

```
2
```

Osservazione

La divisione effettuata è intera. Se ne deduce che in BASH l'operatore aritmetico **/** supporta solo divisioni intere.

Esercizio 2 (2 min.)

Aprite una nuova finestra di terminale.
Dichiarate una variabile **`a`** con attributo intero.
Leggete il valore **`8/3`** tramite terminale nella variabile.
Stampate il valore della variabile.

Definizione e lettura variabile

Si dichiara una variabile di nome **a** con attributo intero:

```
declare -i a
```

Si memorizza in **a** una stringa letta da terminale:

```
read a
```

Si immette il valore **8/3** e si preme **<INVIO>**.

Stampa valore variabile

Si stampa il valore di **a**:

```
echo $a
```

Si ottiene l'output seguente:

```
2
```

Esercizio 3 (2 min.)

Eseguite i due comandi seguenti e spiegate la differenza di comportamento:

```
echo -e a\n
echo -e a\\n
```

Esecuzione comandi

Si eseguono i due comandi.

```
echo -e a\n
an
echo -e a\\n
a
<riga vuota>
```

I due output sono diversi. In particolare, il primo comando non sembra interpretare la sequenza di escape **\n**, mentre il secondo sì.

Spiegazione del primo comando

L'opzione **`-e`** del comando **`echo`** interpreta le sequenze di escape negli argomenti.

Purtroppo, prima di eseguire il comando BASH prova a sua volta ad interpretare i caratteri speciali.

> Il carattere **`\`** è speciale, ed introduce l'operazione di escaping del carattere successivo.

Il carattere successivo viene considerato per quello che è (una misera **`n`**).

Dopo l'escape, BASH rimuove **`\`** ed esegue ciò che rimane:

```
echo -e an
```

Spiegazione del secondo comando

BASH interpreta i caratteri speciali prima di eseguire il comando.
È presente il carattere speciale ****, che annulla il carattere successivo (un altro ****) e lo considera per quello che è (un misero carattere).
Dopo l'escape, BASH rimuove il primo **** ed esegue ciò che rimane, permettendo ad **`echo`** di vedere correttamente la stringa **`a\n`**:

```
echo -e a\n
```

Esercizio 4 (2 min.)

Sia **`f`** una variabile contenente il valore **`image.jpg`**. Si chiede di produrre una serie di trasformazioni che cambino il valore di **`f`** in **`image.png`**.

Dichiarazione e sostituzione variabile

Si imposta inizialmente **f** al valore richiesto:

```
declare f=image.jpg
```

È necessario sostituire l'estensione **jpg** con l'estensione **png**. A tal scopo si può procedere come segue:

usare la trasformazione **${f%jpg}** per rimuovere l'estensione **jpg**;
concatenare l'estensione **png** al risultato della trasformazione.

```
f=${f%jpg}png
```

Esercizio 5 (2 min.)

Definite una variabile **`a`** con attributo intero e valore a vostra scelta. Scrivete uno statement che permetta di stabilire se **`$a`** sia un numero pari o dispari.

Criterio di (dis)parità

Un numero N è pari se il resto della sua divisione con il numero 2 è nullo:

N % 2 = 0

Un numero N è dispari se il resto della sua divisione con il numero 2 è pari a 1:

N % 2 = 1

Stampa valore espressione **a%2**

Si dichiara una variabile a con attributo intero e valore pari:

```
declare -i a=12
```

Si stampa il valore dell'espressione aritmetica **a%2**:

```
echo $((a%2))
0
```

a pari → **$((a%2))** = 0

Stampa valore espressione **a%2**

Si dichiara una variabile a con attributo intero e valore dispari:

```
declare -i a=13
```

Si stampa il valore dell'espressione aritmetica **a%2**:

```
echo $((a%2))
1
```

a dispari → **$((a%2))** = **1**

Esercizio 6 (2 min.)

Eseguite i due comandi seguenti:

```
ls file_non_esistente
LANG=C ls file_non_esistente
```

Notate qualcosa di diverso nell'output dei due programmi?

Esecuzione primo comando

Si esegue il primo comando:

```
ls file_non_esistente
```

Viene stampato il messaggio di errore seguente:

```
ls: impossibile accedere a 'file_non_esistente': File o directory non esistente
```

Esecuzione secondo comando

Si esegue il primo comando:

```
LANG=C ls file_non_esistente
```

Viene stampato il messaggio di errore seguente:

```
ls: cannot access 'file_non_esistente':
No such file or directory
```

Osservazione

La variabile di ambiente **`LANG`** sembra controllare la lingua dei messaggi stampati dalle applicazioni.
In particolare, sembra che **`LANG=C`** implichi l'uso della lingua inglese.

Esercizio 7 (3 min.)

Effettuate il test della condizione seguente nei due modi previsti da BASH:

```
stringa1 > stringa2
```

Stampate il risultato del test in entrambi i casi.

Confronto tramite `test`

Un modo per testare la condizione consiste nell'uso del comando test:

```
test stringa1 \> stringa2
```

Si stampa lo stato di uscita di tale comando, che contiene il valore di verità:

```
echo $?
1
```

Confronto tramite [ ]

Un altro modo per testare la condizione consiste nell'uso del comando `[`:

```
[ stringa1 \> stringa2 ]
```

Si stampa lo stato di uscita di tale comando, che contiene il valore di verità:

```
echo $?
1
```

Osservazioni in ordine sparso

Per quanto possa sembrare folle, il carattere **`[`** è in realtà un comando esterno a BASH!

Provare per credere: **`ls -l /usr/bin/[`**

In BASH il carattere **>** è speciale (introduce la redirezione dell'output su un file). Pertanto, se lo si vuole passare al comando **[** senza subire l'interpretazione di BASH, occorre effettuarne l'escape: **\>**.

Esercizio 8 (4 min.)

Definite una variabile **`a`** con attributo intero e valore pari a **`55`**.
Definite una variabile **`b`** con attributo intero. Leggete un valore a caso di **`b`** da terminale.
Scrivete un costrutto if con i rami seguenti:

se a > b, stampate "a è maggiore di b".
se a < b, stampate "a è minore di b".
se a=b, stampate "a è uguale a b".

Definizione e inizializzazione variabili

Si definisce una variabile **a** con attributo intero e valore pari a **55**:

```
declare -i a=55
```

Si definisce una variabile **b** con attributo intero:

```
declare -i b
```

Si legge un valore arbitrario di **b** da terminale:

```
read b
```

Costrutto if richiesto

Il costrutto if richiesto è il seguente:

```
if [ $a -gt $b ]; then
   echo "a è maggiore di b";
elif [ $a -lt $b ]; then
   echo "a è minore di b";
else
   echo "a uguale a b";
fi
```

Esercizio 9 (4 min.)

Definite una variabile **`a`** con un valore qualunque tra **`start`**, **`stop`** e **`restart`**.
Scrivete un costrutto case con i rami seguenti:

- se **`a = start`**, stampate "system started".
- se **`a = stop`**, stampate "system stopped".
- se **`a = "restart"`**, stampate "system stopped" e "system started".
- in tutti gli altri casi, stampate "wrong command".

Definizione e inizializzazione variabili

Si definisce una variabile **a** con un valore qualunque tra **start**, **stop** e **restart**:

```
a="restart"
```

Costrutto case richiesto

Il costrutto case richiesto è il seguente:

```
case $a in
start)
     echo "system started";;
stop)
     echo "system stopped";;
restart)
     echo "system stopped";;
     echo "system started";;
*)
     echo "wrong command";;
esac
```

Esercizio 10 (4 min.)

Definite una variabile **`a`** con attributo intero e valore pari a **`1`**.
Scrivete un ciclo while che stampa tutti i numeri dispari da 1 a 10.

Definizione e inizializzazione variabile

Si definisce una variabile **a** con attributo intero e valore pari a **1**:

```
declare -i a=1
```

Il costrutto while richiesto

```
while [ $a -lt 10 ]; do
   if [ $((a%2)) -eq 1 ]; then
      echo $a
   fi
   let a=++a
done
```

Esercizio 11 (2 min.)

Definite una variabile **`a`** con attributo intero e valore pari a **`1`**.
Scrivete un ciclo for che stampa i quadrati dei numeri da 1 a 10.

Definizione e inizializzazione variabile

Si definisce una variabile **a** con attributo intero e valore pari a **1**:

```
declare -i a=1
```

Il costrutto for richiesto

```
for a in 1 2 3 4 5 6 7 8 9 10; do
   echo $((a**2))
done
```

Esercizio 12 (1 min.)

Stampate il messaggio “Shell installate” se e solo se i due binari eseguibili **`/bin/bash`** e **`/bin/sh`** sono presenti.

Costruzione logica del comando

Il comando richiesto deve avere per forza la forma seguente:

```
INTERPRETE_BASH_ESISTE && INTERPRETE_SH_ESISTE
&& echo "Shell installate"
```

dove:

INTERPRETE_BASH_ESISTE è un comando che ritorlo stato di uscita 0 se BASH è installato;

INTERPRETE_SH_ESISTE è un comando che ritorlo stato di uscita 0 se SH è installato.

Controllo esistenza delle shell

Un controllo molto semplice per l'esistenza di una shell consiste nell'elencare il file associato tramite **`ls`**.
Se il file esiste, **`ls`** ritorna lo stato di uscita **`0`**.

Controllo esistenza BASH:

```
ls /bin/bash
```

Controllo esistenza SH:

```
ls /bin/sh
```

Il comando richiesto

```
ls /bin/bash && ls /bin/sh && echo "Shell installate"
```

Esercizio 13 (4 min.)

Scrivete uno script shell di nome **`ciclo.sh`** che svolge le operazioni seguenti.

Riceve in ingresso un parametro e lo memorizza nella variabile con attributo intero **`i`**. Se il parametro non è definito, associa il valore di default **`10`**.

Esegue un ciclo da **`1`** a **`i`**.

Ad ogni iterazione del ciclo, stampa il valore del numero.

Iterazione 1 → Stampa "1"
Iterazione 2 → Stampa "2"
…

Esce con lo stato **`0`**.

Lo script `ciclo.sh`

```
declare -i i=${1:-10}
declare -i c=1

while [ $c -le $i ]; do
   echo $c
   let c=++c
done

exit 0
```

Esercizio 14 (2 min.)

Girovagando sul Web avete trovato il comando seguente, che vi piace moltissimo e decidete di usare:

```
strings /dev/urandom | grep -o '[[:alnum:]]' | head -n 30 | tr -d '\n'; echo
```

Create un alias di nome **gp** per questo comando.

Eseguite l'alias **gp**. Che cosa ottenete?

Creazione ed esecuzione dell'alias

Si crea l'alias **gp** al comando richiesto:

```
alias gp="strings /dev/urandom | grep -o '[[:alnum:]]' | head -n 30 | tr -d '\n'; echo"
```

Si esegue l'alias **gp** qualche volta:

```
gp
QAkWbMhlrGZ1O4TyUVKDzve51urOTD
gp
BMep6A8Lb8iJoyBCuNO035rqCb4mWI
gp
9hauxJzyleZUoQ79lII4z9PIFmvzFZ
...
```

Osservazione

L'alias `gp` stampa stringhe alfanumeriche casuali di lunghezza pari a 30 caratteri.

Esercizio 15 (2 min.)

Scrivete una funzione **`square()`** che accetta un parametro, lo interpreta come un intero e ne ritorna il quadrato.
Usate **`square()`** per stampare il quadrato di **`4`**.

Calcolo del quadrato di un numero

Per calcolare il quadrato di una variabile **`var`** si può usare l'operatore aritmetico **`**`**:

```
$((var**2))
```

Per calcolare il quadrato del parametro passato alla funzione si può usare l'espressione seguente:

```
$(($1**2))
```

Ritorno del quadrato

Per ritornare il quadrato del parametro passato alla funzione si può usare lo statement **`return`** con l'espressione seguente:

```
return $(($1**2));
```

La funzione richiesta

```
function square() {
   return $(($1**2));
}
```

Invocazione di `square(4)`

```
function square() {
   return $(($1**2));
}

square 4
```

Stampa di `square()`

```
function square() {
   return $(($1**2));
}

Square 4
echo $?
```

Esercizio 16 (3 min.)

Provate ad indovinare l'output dello script seguente di nome **`scope.sh`**, senza eseguirlo.

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

La variabile **x** è inizializzata con il valore **1**.

Lo scope di **x** è globale: **x=1** è visibile ovunque, purché

- lo statement sia già stato eseguito;
- **x** non sia stata ridefinita localmente.

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

Viene invocata la funzione **f**.

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

La variabile **x** è inizializzata con il valore **3**.

Lo scope di **x** è locale:

- **x=3** è visibile solo durante l'attivazione di **f**;
- **x=3** "oscura" la definizione globale vista in precedenza.

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

Viene invocata la funzione **g**.

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

Viene stampato il valore della variabile **x**. Per individuare la definizione corrispondente, si scrive la cascata di funzioni che ha portato alla esecuzione di g: **SCRIPT** → **f** → **g**.
Poi si cerca a ritroso una definizione di **x** in **f** e in **SCRIPT**.

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

Non c'è una definizione locale di **x** in **g** prima dell'**echo**. Si passa oltre. C'è una definizione locale di **x** in **f**. Pertanto, la variabile **x** modificata è quella definita in **f**. Viene stampato il valore **3**.

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

La variabile **x** viene posta al valore **2**.
Poiché lo scope di BASH è dinamico, **x** si riferisce alla definizione della funzione invocante più vicina, ovvero **f**.
Pertanto, la variabile **x** modificata è quella definita in **f**.

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

Una volta terminata **g**, termina anche **f** (e con essa viene distrutta la variabile locale **x**).

Esecuzione passo passo su carta

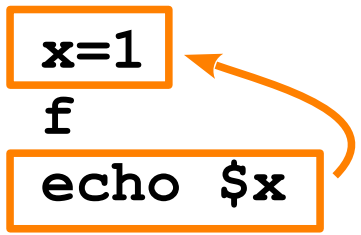
```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

Viene stampato il valore della variabile **x**. A quale definizione punta **x**?

Esecuzione passo passo su carta

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}
```



La cascata di funzioni che porta a **`echo $x`** è: **`SCRIPT`**.
Pertanto, si cerca a ritroso una definizione di **`x`** in **`SCRIPT`**.
La definizione esiste, ed è quella evidenziata nel codice.
Pertanto, viene stampato 1.

Risoluzione dei riferimenti

```
function g() {
   echo $x; x=2;
}
function f() {
   local x=3; g;
}

x=1
f
echo $x
```

In definitiva, l'output dello script è il seguente:

```
3
1
```

Esercizio 17 (1 min.)

Individuate il comando più recente presente nella vostra storia dei comandi.

Stampa ultimo comando eseguito

L'ultimo comando immesso è recuperabile con il comando **`history 1`**:

```
history 1
```

L'output è:

```
N history 1
```

dove **`N`** è un indice intero. Come potrebbe essere altrimenti, dal momento che l'ultimo comando eseguito è proprio **`history 1`**?

Stampa ultimo comando utile

L'ultimo comando immesso utile (quello prima di **`history 1`**) è recuperabile con il comando **`history 2`**:

```
history 2
```

L'output è:

```
N    COMANDO_IMMESSO_PRIMA_DI_HISTORY_1
N+1  history 1
```

Il comando richiesto è quello di indice **`N`**.

Esercizio 18 (1 min.)

Elencate tutti i comandi a partire dall'ultimo **`ls`**.

Stampa intervallo comandi con **`fc`**

Per elencare i comandi si usa l'opzione **`-l`** di **`fc`**:

```
fc -l
```

Passando ad **`fc -l`** un argomento, è possibile individuare il comando a partire dal quale iniziare l'elenco. L'identificazione può avvenire tramite intero o tramite stringa. Nell'esercizio in questione, conviene identificare tramite la stringa **`ls`**:

```
fc -l ls
```

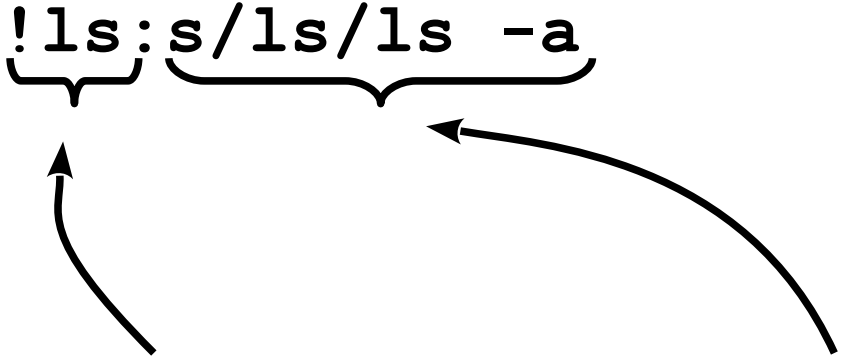
Esercizio 19 (1 min.)

Eseguite l'ultima istanza di **`ls`** nella storia dei comandi, aggiungendo l'opzione **`-a`**.

Soluzione

Si può operare una esecuzione di comando precedente con sostituzione:

```
!ls:s/ls/ls -a
```



Identificazione del comando da eseguire nuovamente.

Sostituzione del comando originale con quello corretto.

Esercizio 20 (1 min.)

Individuate la tipologia dei comandi seguenti:

- `cd`
- `stat`

Individuazione tipologie con **`type`**

Per individuare la tipologia di più comandi, è possibile passare i loro nomi come argomenti del comando **`type`**:

```
type cd stat
```

Si ottiene l'output seguente:

```
cd è un comando interno di shell
stat è /usr/bin/stat
```

Il comando **`cd`** è interno. Il comando **`stat`** è esterno.

Esercizio 21 (1 min.)

Individuate le diverse tipologie del comando seguente:

```
kill
```

Individuazione omonimie con **`type -a`**

Per individuare tutte le tipologie del comando **`kill`**, è possibile eseguire **`type`** con l'opzione **`-a`**:

```
type -a kill
```

Si ottiene l'output seguente:

```
kill è un comando interno di shell
kill è /usr/bin/kill
kill è /bin/kill
```

Il comando **`kill`** è presente sia in forma di builtin di BASH, sia come comando esterno (nei percorsi **`/bin`** e **`/usr/bin`**).