



d1.unito.it

DIPARTIMENTO
DI INFORMATICA

DI INFORMATICA
AUTOMAZIONI
ED ELETTRONICA

di.ifiit.unito.it

laboratorio di sistemi operativi

bash scripting

Claudio Schifanella

argomenti del laboratorio UNIX

1. introduzione a UNIX;
2. integrazione C: operatori bitwise, precedenze, preprocessore, pacchettizzazione del codice, compilazione condizionale e utility make;
3. controllo dei processi;
4. pipe e fifo;
5. code di messaggi;
6. semafori;
7. memoria condivisa;
8. segnali;
9. introduzione alla programmazione bash.

bash

- Bash è la shell di default per Linux. È una shell Unix, cioè:
 - un interprete di comandi che costituisce l'interfaccia utente verso una ricca offerta di utility e linguaggi di programmazione
 - un linguaggio di programmazione tramite il quale combinare utility
- È possibile creare file di comandi (script di shell) che diventano a loro volta nuovi comandi.

```
$ ipcs -s
```

```
IPC status from <running system> as of Tue Dec 16 12:04:58 CET 2014
```

T	ID	KEY	MODE	OWNER	GROUP
---	----	-----	------	-------	-------

```
Semaphores:
```

s	196608	0x7402a210	--ra-----	root	wheel
---	--------	------------	-----------	------	-------

```
$
```

```
#!/bin/bash

for i in `ipcs -s | grep schi | awk '{print $2}'` ; do
    ipcrm -s $i;
done
```

caratteristiche principali

- è possibile **effettuare redirezioni** di input ed output :
 - `$ cmd > outfile`
- è possibile **sequenzializzare comandi** sullo stesso prompt:
 - `$ cmd1 < dati ; cmd2 ; cmd3 >> outfile`
- è possibile **combinare programmi** mediante pipe:
 - `$ cmd1 | cmd2 | cmd3`

shell scripting

- Un linguaggio di scripting è un **linguaggio** di programmazione **interpretato** destinato a compiti di automazione di sistema, piccole applicazioni.
 - generalmente si tratta di semplici programmi destinati a interagire con sistemi più complessi.
- I primi linguaggi di scripting nacquero dall'esigenza di **automatizzare operazioni ripetitive** come l'esecuzione di particolari **programmi**;
- attualmente sono molto diffusi anche nello sviluppo per il web.

script

- uno *script* è un programma interpretato; è un file di testo contenente dichiarazioni di variabili, comandi e strutture di controllo;
 - il file ha i diritti di esecuzione impostati
 - i contenuti di uno script vengono letti ed eseguiti direttamente dalla shell
- di norma lo script ha una prima linea speciale:

```
#!/bin/nomeshell
```

- dove *nomeshell* indica il tipo di shell in grado di interpretare lo script (*bash*, *tcsh*, *csh*, *ksh*, ...), es.

```
#!/bin/bash
```

```
#!/bin/bash  
# questo è un commento!!!  
echo Hello World
```

mio_script

```
-rwxr-xr-x 1 schi staff 18 Dec 4 03:54 mio_script
```

- La prima riga indica al sistema quale software usare per avviare lo script.
- La terza riga è l'azione eseguita dallo script che stamperà a video la scritta *Hello World*.
 - Fornendo diritti di esecuzione allo script esso potrà essere eseguito tramite **il comando**

./mio_script

8

esercizio

```
#!/bin/bash  
  
# questo è un commento!!!  
echo Hello World
```

- creare un file di testo;
- copiarvi il codice riportato nell'esempio, salvare e uscire;
- rendere eseguibile il file tramite `chmod`;
- eseguire il programma così prodotto.

variabili

- Una **variabile** viene dichiarata nel momento in cui le viene assegnato un valore (**stringa o numero**);
- **Dichiarazione**
nomevar=valore
- Si accede al valore di una variabile tramite **\$nomevar**,
echo \$nomevar

argomenti degli script

- Gli script possono avere parametri, identificati dalla loro posizione. per esempio

```
echo $0 $1
```

stampa i primi due parametri del programma

- Attenzione al valore di \$0 ... vi ricordate di argv?

esercizio

```
#!/bin/bash  
  
vrb1 = 5  
echo $vrb1
```

```
#!/bin/bash  
  
vrb1=5  
echo $vrb1
```

```
#!/bin/bash  
  
echo $0 $1
```

- Scrivere ed eseguire gli script riportati sopra.

quoting

- Quotare significa interpretare come normali dei caratteri che hanno un significato speciale, come per esempio \$

```
#!/bin/bash
VAR=schi
VAR1=23
echo $VAR $VAR1
echo $0 $1

echo "il valore è $VAR" APICI DOPPI
echo `ls -l | grep schi` APICI SINGOLI INVERSI
echo `ls -l | grep $VAR` APICI SINGOLI
echo '$VAR'
```



quoting

- Quotare significa interpretare come normali dei caratteri che hanno un significato speciale, come per esempio \$

```
#!/bin/bash  
VAR=schi  
VAR1=23  
echo $VAR $VAR1  
echo $0 $1
```

stampa la stringa sostituendo il valore

```
echo "il valore è $VAR" APICI DOPPI
```

```
echo `ls -l | grep schi` APICI SINGOLI INVERSI
```

```
echo `ls -l | grep $VAR`
```

```
echo '$VAR'
```

APICI SINGOLI

quoting

- Quotare significa interpretare come normali dei caratteri che hanno un significato speciale, come per esempio \$

```
#!/bin/bash  
VAR=dicaro  
VAR1=23  
echo $VAR $VAR1  
echo $0 $1
```

effettua sostituzione,
interpreta come
comando e restituisce
il risultato
dell'esecuzione del
comando

```
echo "il valore è $VAR" APICI DOPPI  
echo `ls -l | grep dicaro` APICI SINGOLI INVERSI  
echo `ls -l | grep $VAR` APICI SINGOLI  
echo '$VAR'
```

quoting

- Quotare significa interpretare come normali dei caratteri che hanno un significato speciale, come per esempio \$

```
#!/bin/bash  
VAR=dicaro  
VAR1=23  
echo $VAR $VAR1  
echo $0 $1
```

non effettua
alcuna
sostituzione

```
echo "il valore è $VAR" APICI DOPPI
```

```
echo `ls -l | grep dicaro` APICI SINGOLI INVERSI  
echo `ls -l | grep $VAR`
```

```
echo '$VAR' APICI SINGOLI
```



quoting

- Quotare significa interpretare come normali dei caratteri che hanno un significato speciale, come per esempio \$

```
#!/bin/bash
VAR=dicaro
VAR1=23
echo $VAR $VAR1
echo $0 $1
```

”	permette l'interpolazione delle variabili
`	permette l'interpolazione delle variabili e dei comandi
'	disabilita qualsiasi interpolazione

```
echo "il valore è $VAR"
echo `ls -l | grep dicaro`
echo `ls -l | grep $VAR`
echo '$VAR'
```

assegnamento

```
#!/bin/bash

tmp=`ls -la | wc -l`
tmp1=`ls -l | wc -l`

echo "numero di elementi nella directory: "$tmp
echo "di questi, non hidden: " $tmp1
```

```
let "diff=$tmp-$tmp1"
echo "hidden: " $diff
```

let permette di assegnare a variabili il valore di espressioni

sintassi

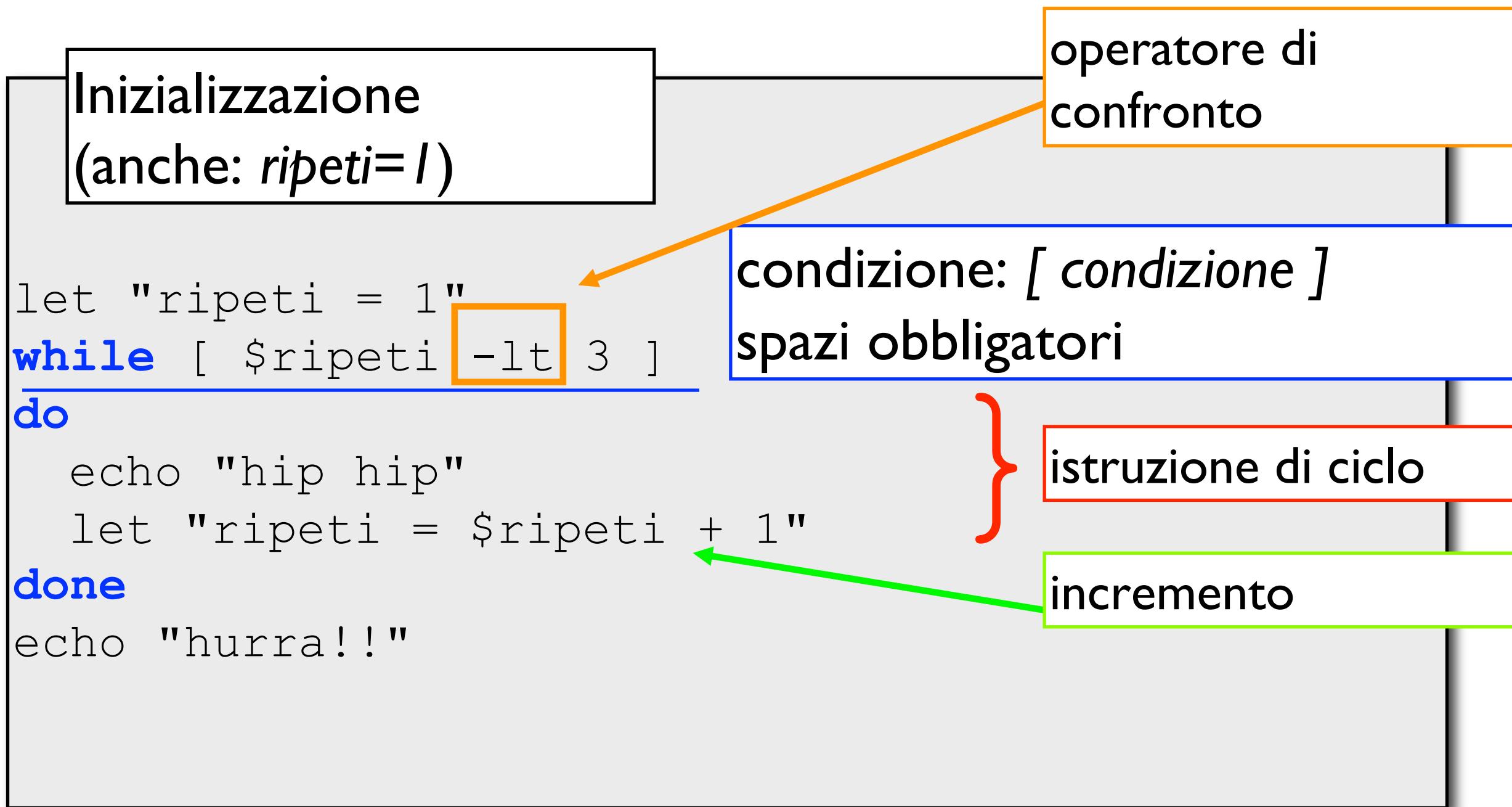
```
let "var = espressione"
```

operatori

costrutti di controllo

```
let "ripeti = 1"
while [ $ripeti -lt 3 ]
do
    echo "hip hip"
    let "ripeti = $ripeti + 1"
done
echo "hurra!!"
```

costrutti di controllo



= uguale	-gt maggiore	-ge maggiore uguale	-eq uguale
!= diverso	-lt minore	-le minore uguale	-ne diverso

lettura da *stdin*

```
finito=go
while [ $finito != quit ]
do
    echo "un altro giro? [go/quit] "
    read finito
done
```

legge un valore da standard
input e lo assegna alla
variabile indicata

if then else

```
DEFDIR=path_to_dir
if [ $1 ]
then
if [ $1 = "-d" ]
then
mia_var=`wc -l $2`
echo $mia_var
else
echo "opzione sconosciuta"
fi
else
mia_var=`wc -l $DEFDIR`
echo $mia_var
fi
```

if then else

```
DEFDIR=path_to_dir
if [ $1 ] se il parametro 1 è definito
then
  if [ $1 = "-d" ] se è uguale a “-d”
    then esegui queste istruzioni
      mia_var=`wc -l $2`
      echo $mia_var
    else altrimenti
      echo "opzione sconosciuta"
    fi
  else se param. 1 non è definito
    mia_var=`wc -l $DEFDIR`
    echo $mia_var
  fi
```

if then else

```
DEFDIR=path_to_dir
if [ $1 ]
then
  if [ $1 = "-d" ]
  then
    mia_var=`wc -l $2`
    echo $mia_var
  else
    echo "opzione sconosciuta"
  fi
else
  mia_var=`wc -l $DEFDIR`
  echo $mia_var
fi
```

condizioni composte $[\text{condizione1}] \&& [\text{condizione2}]$

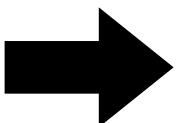
test su file

- Bash definisce una **serie di operatori** per effettuare test su file

-f file esiste
-s file non è vuoto
-r file leggibile
-w file scrivibile
-x file eseguibile
-d è una directory
-h è un link simbolico

...

```
[ -r documento.txt ]  
[ -x $1 ]  
[ -d $1 ] && [ -w $1 ]
```



il file indicato è leggibile?
il parametro è un eseguibile?
\$1 è una directory scrivibile?

for ... in ...

cursore:

```
for myf in `ls *.c`  
do  
    echo $myf  
done
```

for variabile in lista-valori
do
istruzioni
Done

```
#!/bin/bash  
  
for i in `ipcs -s`; do  
    // fai qualcosa  
done
```

esportazione di variabili

- **Scope di una variabile:** lo script in cui è dichiarata
- ...e se da uno script ne richiamiamo un altro?

mio_script

```
dato=`< mioinput`  
saluta
```

saluta

```
echo "ciao " $dato
```

- se da *mio_script* richiamo *saluta*, dal secondo ho accesso alle variabili dichiarate nel primo ?

esportazione di variabili

- **Scope di una variabile:** lo script in cui è dichiarata
- ...e se da uno script ne richiamiamo un altro?

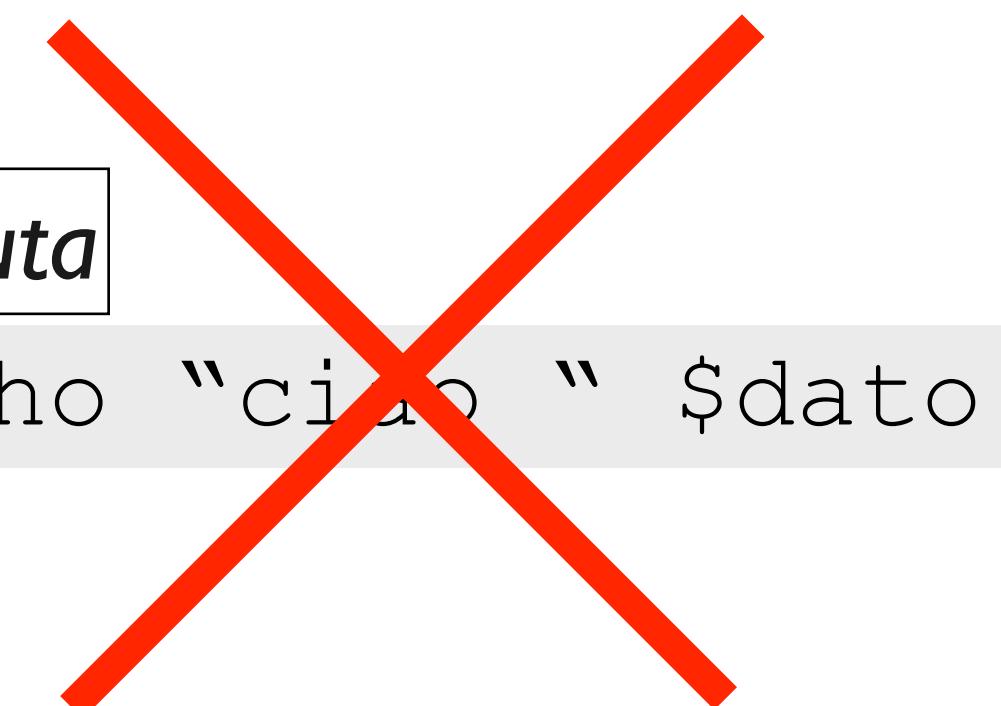
`mio_script`

```
dato=`< mioinput`  
saluta
```

`saluta`

```
echo "ciao" $dato
```

- "saluta" viene eseguito da una **bash figlia** di quella che esegue *mio_script* dalla quale non si ha accesso alle variabili di *mio_script*



NO! ²⁸

esportazione di variabili

- per rendere una variabile visibile dalle sotto-shell
occorre **esportarla**

mio_script

```
export dato=< mioinput
saluta
```

saluta

```
echo "ciao " $dato
```

File di configurazione: *.bashrc*

```
export PATH=.:mybin:$PATH
```

concatena alla lista attuale di directory in cui cercare gli eseguibili le directory '.' (directory di lavoro, qualunque essa sia) e 'mybin'