

# Lezione 6 in laboratorio - parte 2

## processi, stringhe

hic sunt  
canes stercore



NOTA BENE:

A questo punto abbiamo già visto ed usato anche i comandi:

ps nohup disown bg fg kill wait

Usare il comando man nomecomando per ottenere informazioni sull'uso di uno specifico comando di nome nomecomando.

# Familiarizzarsi con processi PID e signal (1)

## 1. Scrivere tre script bash **lanciaPipeSignal.sh** **uno.sh** e **due.sh**.

- Lo script **lanciaPipeSignal.sh** mette in esecuzione in una pipe i due script uno.sh e due.sh facendo sì che l'input di uno.sh sia preso dal contenuto del file /usr/include/stdio.h e che l'output di uno.sh venga usato come input di due.sh.
- Lo script **uno.sh** inizia scrivendo in un file uno.pid.txt il proprio process identifier. Poi aspetta 2 secondi. Poi legge dal file due.pid.txt il process identifier del processo bash che sta eseguendo due.sh. Poi uno.sh legge la prima riga dal suo standard input, la mette in output aggiungendovi all'inizio la stringa "UNO ", poi manda il segnale SIGUSR2 al processo due.sh di cui possiede il pid. Poi si mette in attesa in un loop infinito contenente sleep 1.

Successivamente, ogni volta che lo script uno.sh riceve il segnale SIGUSR2 da due.sh, uno.sh deve leggere una riga dal proprio standard input, la deve mettere in output, aggiungendovi all'inizio la stringa "UNO ", poi manda il segnale SIGUSR2 al processo due.sh di cui possiede il pid. E così via. Leggendo la fine file. lo script uno.sh prima manda il segnale SIGUSR2 al processo due.sh e poi termina.

- Lo script **due.sh** inizia scrivendo in un file due.pid.txt il proprio process identifier. Poi aspetta 2 secondi. Poi legge dal file uno.pid.txt il process identifier del processo bash che sta eseguendo due.sh. Poi entra in un loop infinito contenente sleep 1. Successivamente, ogni volta che lo script due.sh riceve il segnale SIGUSR2 da uno.sh, due.sh deve leggere una riga dal proprio standard input, la deve mettere in output, aggiungendovi all'inizio la stringa "DUE ", poi manda il segnale SIGUSR2 al processo uno.sh di cui possiede il pid. E così via. Leggendo la fine file lo script due.sh termina.

# Familiarizzarsi con processi PID e signal (1)

## Soluzione

### 1. lanciaPipeSignal.sh

```
#!/bin/bash
cat /usr/include/stdio.h | \
./uno.sh | ./due.sh
```

### due.sh

```
#!/bin/bash
ricevutoSIGUSR2() {
    if read RIGA ; then
        echo "DUE ${RIGA}";
        kill -SIGUSR2 ${PIDuno}
    else
        exit 1
    fi
}

trap ricevutoSIGUSR2 SIGUSR2

echo ${BASHPID} > due.pid.txt
sleep 2
read PIDuno < uno.pid.txt
while true; do
    sleep 1
done
```

# Familiarizzarsi con processi PID e signal (1)

## Continuazione Soluzione

### **uno.sh**

```
#!/bin/bash
ricevutoSIGUSR2() {
    if read RIGA ; then
        echo "UNO ${RIGA}";
        kill -s SIGUSR2 ${PIDdue}
    else
        kill -s SIGUSR2 ${PIDdue}
        exit 0
    fi
}
trap ricevutoSIGUSR2 SIGUSR2
echo ${BASHPID} > uno.pid.txt
sleep 2
read PIDdue < due.pid.txt

# leggo prima riga e mando segnale
continua qui a destra
```

### **continuazione di uno.sh**

```
# leggo la prima riga e mando
# il primo segnale

if read RIGA ; then
    echo "UNO ${RIGA}";
    kill -s SIGUSR2 ${PIDdue}
else
    kill -s SIGUSR2 ${PIDdue}
    exit 1
fi

while true; do
    sleep 1
done
```

# comandi condizionali (1)

2. Capire che exit status viene restituito dal seguente script **bastardo.sh**

```
( sleep 2; ls -d /usr/include/ ) && { [[ (! ( "false" > "true" )) ||  
      ( 3 -le 5 ) ]] ; } && if [[ $? < "01" ]] ; then exit 0; else exit 1 ; fi ;  
exit $?
```

# comandi condizionali (1)

## Soluzioni di esercizi slide precedente

### **2. bastardo.sh**

l'esecuzione produce exit status 0

# esercizio 95 - **discendenti.sh**

Scrivere uno script bash **discendenti.sh**, che prende un argomento intero a riga di comando. L'intero indica il numero di script figli da lanciare.

Ad esempio, all'inizio lo script potrebbe essere lanciato passandogli come argomento "3".

Lo script controlla l'argomento che gli è stato passato.

- Se il valore dell'argomento è maggiore di 0, lo script lancia in background lo script stesso tante volte quanto il valore dell'argomento intero e passa come argomento a ciascuno script proprio quel valore intero diminuito di 1. Poi lo script attende la fine di tutti i suoi processi figli. Poi stampa a video l'argomento che gli è stato passato. Infine termina restituendo 0.
- Se invece il valore dell'argomento è uguale a zero, allora lo script stampa a video l'argomento che gli è stato passato e poi termina restituendo 0.

# **soluzione esercizio 95 - discendenti.sh**

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

```
if (( "$#" != "1" )) ; then echo "serve un argomento intero" ; exit 1 ; fi  
if (( "$1" < "0" )) ; then echo "serve un argomento intero maggiore o uguale a 0" ;  
exit 1 ; fi
```

```
NUMFIGLI=$1
```

```
for (( i=0; $i < ${NUMFIGLI}; i=$i+1 )) ; do  
    ./discendenti.sh ${NUMFIGLI}-1 ) &  
done
```

```
#for (( i=0; $i < ${NUMFIGLI}; i=$i+1 )) ; do  
#    wait
```

```
#done
```

```
echo ${NUMFIGLI}
```

```
exit 0
```

# Esercizio 41 - script cercarecente

Scrivere uno script bash **cercarecente.sh** che comincia cercando tutti i file con estensione .h in **tutte le sottodirectory** della directory /usr/include/linux/ escludendo i files che si trovano direttamente nella directory /usr/include/linux/

Confrontare la data di ultima modifica dei file così trovati e stampare a video il nome del file modificato più recentemente.

# Soluzione Esercizio 41 - script cercarecente

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

```
FILES=`find /usr/include/linux/ -mindepth 2 -name "*.h" -print`  
if [[ -z ${FILES} ]] ; then  
    echo "nessun file trovato"  
else  
# assegno a RECENTE il primo nome di file  
    for RECENTE in ${FILES} ; do  
        break  
    done  
  
    for NAME in ${FILES} ; do  
        if [[ ${RECENTE} -ot ${NAME} ]] ; then  
            RECENTE=${NAME}  
        fi  
    done  
fi  
echo "il file piu' recente e' ${RECENTE}"
```

# Esercizio 113 - insuff2.sh

I due file RisultatiProvaPratica1.txt e RisultatiProvaPratica2.txt contengono in ciascuna riga di testo il Nome, il Cognome, la Matricola e il Voto ottenuti dallo studente nella prova pratica N° 1 e N° 2 rispettivamente. Ciascun Nome e ciascun Cognome è composto da una sola parola. Il numero di matricola è univoco. Il Cognome e il nome, invece, potrebbero essere ripetuti. Il voto può essere non sufficiente (voto < 18).

Scrivere uno script bash **insuff2.sh** che metta in output l'elenco dei soli studenti che rispettano TUTTE le seguenti caratteristiche:

- Hanno sostenuto la **seconda** prova pratica, ottenendo un voto NON sufficiente,
- **Non** hanno sostenuto la **prima** prova pratica.

L'output deve essere formattato su più righe di testo. Ciascuna riga contiene le informazioni su uno studente, in particolare **la Matricola, il Nome, il Cognome ed il voto ottenuto nella seconda prova pratica, in quest'ordine**. Le righe dell'output devono essere **ordinate secondo il Cognome**, in senso crescente.

Esempio di file:

**RisultatiProvaPratica1.txt**

Avio Verdi 876754 21

Dee Bord 666666 20

Rino Ceronte 222222 13

Caio Baro 777777 27

**RisultatiProvaPratica2.txt**

Carmine Ati 8888 23

Paolo Venzi 333333 9

Dee Bord 666666 12

Sante Bo 888888 14

**OUTPUT**

888888 Sante Bo 14

333333 Paolo Venzi 9

# SOLUZIONE Esercizio 113 - insuff2.sh

UN PO' BIZZARRA, USA UN TRUCCO: AGGIUNGE IL CAMPO COGNOME A INIZIO RIGA PER ORDINARE LE RIGHE PRODOTTE, E POI ELIMINA QUEL PRIMO CAMPO COGNOME PRIMA DI MANDARE LE RIGHE IN OUTPUT

```
#!/bin/bash

while read NOME COGNOME MATRICOLA VOTO ; do
    if (( ${VOTO} < "18" )) ; then
        LINES=`grep ${MATRICOLA} RisultatiProvaPratica1.txt | wc -l`
        if [[ "${LINES}" == "0" ]] ; then
            echo ${COGNOME} ${MATRICOLA} ${NOME} ${COGNOME} ${VOTO}
        fi
    fi
done < RisultatiProvaPratica2.txt | sort | cut -d ' ' -f2-
```

Il sort ordina in base al campo COGNOME a inizio riga.

Il cut finale elimina il campo COGNOME a inizio di ciascuna riga prima di mandarla in output

# ALTRA SOLUZIONE MENO BIZZARRA

## Esercizio 113 - insuff2b.sh

```
#!/bin/bash

while read NOME COGNOME MATRICOLA VOTO ; do
    if (( ${VOTO} < "18" )) ; then
        LINES=`grep ${MATRICOLA} RisultatiProvaPratica1.txt | wc -l`
        if [[ "${LINES}" == "0" ]] ; then
            echo ${MATRICOLA} ${NOME} ${COGNOME} ${VOTO}
        fi
    fi
done < RisultatiProvaPratica2.txt | sort -k 3
```

Il sort effettua l'ordine in base al terzo campo di ciascuna riga, il cognome

# Processi e altro (1)

**3. MOLTO DIFFICILE, SOLO PER FUORI DI TESTA:** Scrivere un nuovo script **lanciapuntini.sh** trova il modo di: a) lanciare in foreground lo script puntini.sh (già realizzato in una lezione precedente e descritto qui sotto) passandogli come parametro intero il numero 30, di farlo eseguire per circa 5 secondi e di sospenderne l'esecuzione senza usare la tastiera. b) poi stampa in output "sospeso". c) poi attende circa 3 secondi. d) poi riporta in esecuzione in foreground lo script puntini.sh e trova il modo di stampare a video, dopo circa 4 secondi, la stringa hello. Lo script puntini non deve essere modificato.

**USARE LO SCRIPT** puntini.sh **GIA PROPOSTO IN LEZIONE PRECEDENTE.**

Scrivere uno script **puntini.sh** che prende come argomento a riga di comando un intero positivo che rappresenta un certo numero di secondi. Lo script deve rimanere in esecuzione per quel numero di secondi e, ad ogni secondo, stampare a video un punto . seguito dal proprio PID. Ma senza andare a capo.

## **51.puntini.sh**

```
#!/bin/bash
NUM=0
while (( ${NUM} <= $1 )) ; do
    sleep 1; echo -n ". ${BASHPID}";
    ((NUM=${NUM}+1))
done
```

# Processi e altro (2)

## Soluzione di esercizio fuori di testa

### 3. lanciapuntini.sh

```
#!/bin/bash
( sleep 5 ;
RIGA=`ps | grep puntini.sh | grep -v lanciapuntini.sh` 
# tolgo eventuali spazi iniziali prima del pid nella riga di testo
while [[ "${RIGA}" != "${RIGA# }" ]]; do RIGA="${RIGA# }" ; done
PIDPUNTINI=${RIGA%% *}
kill -s SIGTSTP ${PIDPUNTINI} ;
echo lanciata sospensione
sleep 4
kill -s SIGCONT ${PIDPUNTINI} ;
echo lanciata ripresa puntini
(sleep 4 ; echo hello ) &
) &
./puntini.sh 15
```

### 1. puntini.sh

```
#!/bin/bash
NUM=0
while (( ${NUM} <= $1 )) ; do
    sleep 1
    echo -n ". ${BASHPID}"
    ((NUM=${NUM}+1))
done
```

# usare iterazioni bash e file

## Esercizio1: piu' for per tutti

In una propria directory, creare 10 directory avente nome

1.0 1.1 1.2 1.3 1.4 ..... 1.9

Utilizzare il comando **for** ed il comando **mv** della bash, per cambiare i nomi delle directory rispettivamente in :

2.0 2.1 2.2 2.3 2.4 ..... 2.9

Suggerimento:guardare le slide su bash scripting, dove si parla di Estrazione di sottostringhe da variabili.

## Esercizio2: e ancora un po' piu' di for per tutti

In una propria directory, creare 10 directory avente nome

1.0 1.1 1.2 1.3 1.4 ..... 1.9

Utilizzare il comando **for** ed il comando **mv** della bash, per cambiare i nomi delle directory rispettivamente in

2.9 2.8 2.7 2.6 2.5 ..... 2.0

Notare che, ad esempio, 1.1 deve diventare 2.8 e 1.3 deve diventare 2.6

In generale, 1.X deve diventare 2.(9-X)

# usare iterazioni bash e file

## **Soluzione Esercizio1: piu' for per tutti**

```
for (( NUM=0 ; ${NUM}<10 ; NUM=${NUM}+1 )) ; do mv 1.${NUM} 2.${NUM} ; done
```

## **Soluzione Esercizio2: : e ancora un po' piu' di for per tutti**

```
for (( NUM=0 ; ${NUM}<10 ; NUM=${NUM}+1 )) ; do ((NEWNUM=9-${NUM})) ; mv 1.${NUM} 2.${NEWNUM} ; done
```

# usare moduli, man, gcc, Makefile

## Esercizio3: correggere errori nei Makefile e nei moduli C

All'indirizzo

[http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/ESERCIZI\\_CORREGGERE\\_ERRORI\\_1.tgz](http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/ESERCIZI_CORREGGERE_ERRORI_1.tgz)

c'e' un archivio in formato tar gz contenente una directory 1 che a sua volta contiene delle sottodirectory 1.1 1.2 1.3 1.4 ..... 1.8 1.9

In ciascuna sottodirectory c'e' il necessario per creare un eseguibile, ovvero i codici sorgenti in linguaggio ANSI C (esageratamente semplici) ed un Makefile.

Purtroppo 😊 sorgenti e Makefiles possono contenere degli errori.

I sorgenti e i makefile sono talmente semplici che DOVETE essere in grado di capire cosa fanno, anche se contengono errori.

Quindi, scaricate l'archivio, decomprimetelo in una vostra directory, e poi entrate in ciascuna delle directory in ordine crescente di secondo indice, cioe' prima 1.1 poi 1.2 poi 1.3

In ciascuna directory provate a generare l'eseguibile, correggendo gli eventuali errori.

Poi provate ad eseguire l'eseguibile, correggendo eventuali errori.

NB: decomprimere l'archivio fa parte dell'esercizio.