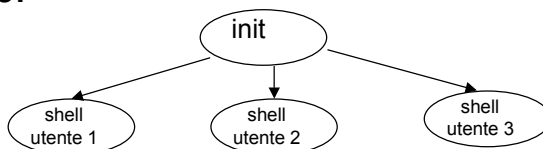


Script di shell (bash)

Shell di Unix

- Esistono diversi shell:
 - Bourne Shell
 - C Shell
 - Korn Shell
 - Tc Shell
 - etc.
- Interfaccia di alto livello tra utente e sistema operativo.



Shell di Unix

- Interpreta e mette in esecuzione comandi da:
 - standard input
 - file comandi
- Linguaggio comandi con elevato potere espressivo.
- Accesso al sistema mediante uno shell dedicato all'utente.

Bash

- Bash è un acronimo per Bourne Again Shell
- È una tra le shell più usate in ambiente UNIX
- Il termine shell (conchiglia) deriva dal fatto che la shell funziona da intermediario tra utente e kernel:
 - Il kernel è "la perla" racchiusa dalla shell

Bash: file di configurazione

- La bash ha 5 file di configurazione:
/etc/profile
/etc/bashrc
~/.bash_profile
~/.bashrc
~/.bash_logout
- I file in etc sono globali, gli altri locali e personali del singolo utente

Bash: variabili

- La definizione di una variabile avviene nel modo seguente:
nome_variabile=valore
- Una volta definita, una variabile può essere resa disponibile ai programmi che l'utente usa con il comando export; esempio:
export nome_variabile
- Per visualizzare il contenuto di una variabile si usa il comando echo e si fa precedere il nome della variabile dal carattere \$:
echo \$nome_variabile

Bash: variabile PATH

- La variabile PATH definisce le directory a cui possiamo accedere da qualunque punto del file system:

```
echo $PATH
```

- Per aggiungere un cammino alla variabile si procede come segue:

```
PATH=$PATH:/usr/games
```

```
export PATH
```

Bash: alias

- Gli alias si possono definire come comandi accorciati
- Un comando lungo può essere associato ad un comando mnemonico più breve detto alias

- Esempio:

```
ls -aF --color
```

- può avere come alias:

```
alias ls=`ls -aF --color`
```

Programmazione nello shell di Unix

- Programmazione attraverso:
 - variabili: `x=$y`
 - comandi condizionali:
 - `if..then..else..fi`
 - `case..in..esac`
 - comandi ripetitivi:
 - `for..in..do..done`
 - `while..do..done`
 - valutazione di condizioni:
 - `test`
 - input:
 - `read`
 - ...

Bash: il primo programma

- La prima riga di un programma di shell bash (detto script di shell) deve essere:
`#!/bin/bash`
- Le righe successive possono essere:
 - Commenti
 - Comandi
 - Costrutti di shell
- Esempio:
`#!/bin/bash`
`echo "Ciao Mondo"`

Bash: commenti e assegnazioni

- Per assegnare un valore ad una variabile è sufficiente usare il segno di =

```
X=12
```

- Le righe che iniziano con il carattere # (eccetto la prima) sono considerate commenti

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

```
x=12
```

```
#Ho assegnato ad x il valore 12
```

```
echo $x
```

Bash: il costrutto IF

- Il costrutto IF permette di modificare il flusso di uno script al verificarsi, o meno, di una certa condizione
- if ...then ... else ... elif ... fi
- Per verificare una condizione si può usare il comando test, oppure le parentesi []
- Esempio:
[-f /etc/miofile];
- Restituisce vero se il file miofile esiste in /etc

Bash: il costrutto IF

- Esempio:

```
#!/bin/bash
if [ -f semafori.c ];
then
    cat semafori.c
else
    echo "File non presente"
fi
```

- Se il file semafori.c non esiste nella directory dove viene eseguito lo script, verrà eseguito il ramo else

Bash: il costrutto IF

- I controlli che la shell ci permette di fare su di un file sono i seguenti:
 - -d controlla se il file e' una directory
 - -e controlla se il file esiste
 - -f controlla se il file e' un file regolare
 - -g controlla se il file ha il bit SGID settato
 - -r controlla se il file e' leggibile dall'utente che esegue lo script
 - -s controlla se la dimensione del file non e' 0
 - -u controlla il file ha il bit SUID settato
 - -w controlla se il file e' scrivibile

Bash: il costrutto WHILE

- È un costrutto ciclico che permette di ripetere una serie di operazioni fino a che la condizione di test è vera. Esempio:

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

```
x=0
```

```
while [ "$x" -le 10 ]; do
```

```
    echo "x= $x"
```

```
    x=$((x+1))
```

```
done
```

Bash: il costrutto WHILE

- Come per i file ci sono test predefiniti per i numeri:

- x -eq y Vero se x è equivalente a y

- x -ne y Vero se x non è equivalente y

- x -gt y Vero se x è maggiore y

- x -lt y Vero se x minore y

- x -ge y Vero se è maggiore o equivalente

- x -le y Vero se minore o equivalente

Bash: il costrutto WHILE

- Per confrontare le stringhe di caratteri:
 - `x = y` Vero se x è uguale y
 - `x != y` Vero se x non è uguale a y
 - `-n x` Vero se x non è nullo o vuoto
 - `-z x` Vero se x è nullo o vuoto

Bash: il costrutto UNTIL

- È simile al while, solo che il ciclo termina quando la condizione diventa vera.
- Esempio:

```
#!/bin/bash
x=0
until [ "$x" -ge 10 ]; then
    echo "Current value of x: $x"
    x=$((x + 1))
done
```

Bash: il costrutto FOR

- Il costrutto FOR è usato per ripetere una serie di comandi un numero predeterminato di volte.

```
for ... in ... do ... done
```

- Esempio:

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

```
for mia_var in 1 2 3 4 5; do
```

```
    echo $mia_var
```

```
done
```

Bash: il costrutto FOR

- Esempio:

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

```
for mia_var in *; do
```

```
    echo $mia_var
```

```
done
```

- Il carattere * è un carattere jolly che si espande in: "tutto nella directory corrente"

Bash: il costrutto CASE

- Costrutto di scelta multipla

```
#!/bin/bash
x=5 # assegno a x il valore 5
# incomincio a testare x
case $x in
  0) echo "Il valore di x: 0."
    ;;
  5) echo "Il valore di x: 5."
    ;;
  9) echo "Il valore di x: 9."
    ;;
  *) echo "Valore sconosciuto"
esac
```

Bash: il costrutto CASE

- La stessa cosa poteva essere scritta con il costrutto IF

```
#!/bin/bash
x=5 # assegno a x il valore 5
if [ "$x" -eq 0 ]; then
  echo "Il valore di x: 0."
elif [ "$x" -eq 5 ]; then
  echo "Il valore di x: 5."
elif [ "$x" -eq 9 ]; then
  echo "Il valore di x: 9."
else
  echo "Valore sconosciuto"
fi
```

Bash: virgolette

- Le virgolette doppie `` sono usate per:
 - Impostare un valore stringa preservando gli spazi
 - Espandere il valore delle variabili
- Le virgolette semplici ` non fanno espandere i valori delle variabili.
- Esempio:

```
#!/bin/bash
x=5
# uso le doppie virgolette
echo "Usando le doppie virgolette x : $x"
# uso le virgolette semplici
echo 'Usando le virgolette semplici x : $x'
```

Bash: operazioni con i numeri

- Le operazioni possono essere effettuate in due modi:
 - Inserendo l'espressione tra doppie parentesi tonde
 - Usando il comando expr
- Esempio:

```
x=`expr $x + 1`
```
- Oppure:

```
x=$(( $x + 1 ))
```

Bash: operazioni con i numeri

- Gli operatori aritmetici messi a disposizione sono:
 - Addizione +
 - Sottrazione -
 - Moltiplicazione *
 - Divisione /
 - Resto o Modulo %

Bash: operazioni con i numeri

- Esempio:

```
#!/bin/bash
x=5
y=3
somma=$((x + $y))
sottrazione=$((x - $y))
moltiplicazione=$((x * $y))
divisione=$((x / $y))
modulo=$((x % $y))
echo "Somma: $somma"
echo "Sottrazione: $sottrazione"
echo "Moltiplicazione: $moltiplicazione"
echo "Divisione: $divisione"
echo "Modulo: $modulo"
```

Bash: costrutto AND

- La direttiva AND è indicata con la sequenza di caratteri &&
- La sintassi della direttiva AND è:
prima_condizione && seconda_condizione
- Viene controllata prima la condizione di sinistra e se restituisce vero anche la condizione di destra
- Esempio:
#!/bin/bash
x=5
y=10
if ["\$x" -eq 5] && ["\$y" -eq 10]; then
 echo "Entrambe le condizioni vere"
else
 echo "Le condizioni non sono entrambe vere"
fi

Bash: costrutto OR

- La direttiva OR è indicata con la sequenza di caratteri ||
- La sintassi della direttiva OR è:
prima_condizione || seconda_condizione
- Se la prima condizione è vera allora anche il risultato finale sarà vero, altrimenti viene controllata anche la condizione di destra.

Bash: input

- Il comando "read" è usato per inserire dati da tastiera; è un comando interno della shell che copierà l'input dell'utente in una variabile.
- Esempio:

```
#!/bin/bash
# programma che prende l'input dall'utente e lo saluta
echo -n "Inserisci il tuo nome: "
read nome_utente
echo "Ciao $nome_utente !"
```
- Il comando "read" aspetta che l'utente immetta qualcosa e prema ENTER

Bash: le funzioni

- Le funzioni rendono la programmazione meno ripetitiva e più semplice
- Il programma è suddiviso in una serie di sottoprogrammi (moduli o funzioni)
- Esempio

```
#!/bin/bash
#dichiariamo la funzione ciao()
ciao
{
    echo "Sono nella funzione ciao"
}
echo "Sto per chiamare la funzione ciao"
ciao
```

Bash: le funzioni

- Per richiamare una funzione bisogna averla dichiarata in precedenza
- La dichiarazione di funzione può anche avvenire mediante la parola chiave function

- Esempio:

```
function ciao()
```

```
{
```

```
    echo "Sono nella funzione ciao"
```

```
}
```

- La parola chiave function è facoltativa

Bash: le funzioni

- Tutto quello che si trova tra le parentesi graffe (istruzioni, variabili) è detto ambito locale:
- Esempio:

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

```
function nuovo_utente()
```

```
{
```

```
    echo "Stai per aggiungere un nuovo utente"
```

```
    adduser
```

```
}
```

```
#iniziamo lo script
```

```
echo "1. Aggiungi un utente"
```

```
echo "2. Esci"
```

```
read scelta
```

```
case $scelta in
```

```
1) nuovo_utente
```

```
;;
```

```
*) exit
```

```
;;
```

```
esac
```

Bash: le funzioni

- Può accadere che sia necessario utilizzare all'interno di una funzione un nome di variabile già utilizzato nel codice dello script.
- Per evitare di modificare il valore della variabile globale la shell Bash consente di dichiarare delle variabili che "vivono" solo all'interno del corpo della funzione, tali variabili vengono dette locali e vengono dichiarate attraverso l'istruzione local: local [VARIABILE[=VALORE]].

Bash: le funzioni

```
#!/bin/bash
#
# Uso di local

EXIT_SUCCESS=0
funzione () {
    local VARIABILE="Sono all'interno della funzione"
    echo -e "\t$VARIABILE"
}
VARIABILE="Sono all'esterno della funzione"
echo $VARIABILE
funzione
echo $VARIABILE
exit $EXIT_SUCCESS
```

Bash: segnali di sistema

- È possibile intercettare i segnali di sistema mediante il comando trap
- La sintassi di trap è:
trap azione segnale
- Il parametro azione indica l'azione che si vuole venga intrapresa a fronte dell'arrivo del segnale specificato
- Il parametro segnale può essere indicato in forma numerica come valore intero o in forma letterale:
 - se si vuole intercettare l'interrupt proveniente da tastiera (CTRL-C) si usa la stringa INT o il codice numerico 2

Bash: segnali di sistema

- Esempio:

```
#!/bin/bash
#catturiamo CTRL-C con trap
trap spiacente INT
function spiacente()
{
    echo "Spiacente ma non puoi farlo"
    sleep 3
}

#contiamo alla rovescia da 10 a 1
for i in 10 9 8 7 6 5 4 3 2 1; do
    echo "Mancano $i secondi"
    sleep 1
done
```

Bash: segnali di sistema

- Se si vuole far semplicemente ignorare il segnale al programma è sufficiente indicare, al posto dell'azione, due virgolette semplici vuote:

```
trap `` INT
```

- Se si vuole annullare l'effetto del comando trap si può resettarlo passandogli il carattere

-

```
trap - INT
```

Bash: argomenti

- Ad un comando, e quindi ad uno script, possono, in generale, essere passati dei parametri da linea di comando.

- Le variabili speciali utilizzate per memorizzare i parametri passati sono:

- \$# Contiene il numero dei parametri passati
- \$1-\$n Contiene il valore singolo dei parametri
- \$? Contiene il valore di uscita dell'ultimo comando
- \$0 Contiene il nome del programma
- \$* Contiene tutti i parametri ordinati per immissione
- "\$@" Contiene tutti i parametri ordinati per immissione inclusi in doppie virgolette

Bash: argomenti

- Esempio:

```
#!/bin/bash
#script per visualizzare il primo argomento
if [ "$#" -ne 1 ] then
    echo "uso corretto: $0 argomento"
fi
echo "L'argomento passato e' $1"
```

Bash: valori di ritorno

- Molti programmi restituiscono dei valori di ritorno che ci informano sul risultato della loro esecuzione
- Ad esempio, il comando grep restituisce un 1 se non trova nulla, 0 altrimenti
- La variabile \$? ci permette di controllare il valore di ritorno dell'ultimo programma eseguito; esempio:

```
#!/bin/bash
grep "pippo" /etc/passwd
if [ "$?" -eq 0 ]; then
    echo "Utente trovato"
    exit 0
else
    echo "Utente non trovato"
    exit 1
fi
```

Bash: ricorsione

- La shell Bash consente di scrivere funzioni che richiamano se stesse. Tali funzioni vengono dette ricorsive.
- Una funzione ricorsiva è utile quando all'interno di una funzione occorre applicare ad un dato la funzione stessa, in questo modo si può semplicemente risolvere il problema consentendo alla funzione di richiamarsi.
- Consideriamo l'algoritmo di Euclide per il calcolo del massimo comun divisore (mcd) di due numeri.
- Questo algoritmo consiste nella relazione per ricorrenza (valida per m ed n naturali):
$$\text{mcd}(0; n) = n$$
$$\text{mcd}(m; n) = \text{mcd}(n \bmod m; m) \text{ per } m > 0$$
- dove con mod abbiamo indicato genericamente l'operatore %.

Bash: ricorsione

```
#!/bin/bash
EXIT_SUCCESS=0
EXIT_FAILURE=1
utilizzo () {
    echo "$(basename $0) <primo numero>
    <secondo numero>"
    exit $EXIT_FAILURE
}
mcd () {
    m=$1
    n=$2
    if [ $n -eq 0 -a $m -eq 0 ]; then
        echo "mcd(0,0) non e definito!"
        exit $EXIT_FAILURE
    elif [ $m -eq 0 ]; then
        return $n
    elif [ $n -eq 0 ]; then
        return $m
    fi
    mcd $(( $n % $m )) $m
}
if [ $# -ne 2 ]; then
    utilizzo
fi
mcd $1 $2
MCD=$?
echo "Il massimo comun divisore dei numeri
    \"$1\" \"$2\" e: $MCD"
exit $EXIT_SUCCESS
```

Bash: vettori

- Bash consente di dichiarare esplicitamente un vettore attraverso il comando `declare -a`, tuttavia qualunque riga di codice del tipo:

```
ARRAY[INDICE]=VALORE
```

- creerà automaticamente un vettore. Un'altra forma legale per dichiarare un vettore è l'assegnazione esplicita di tutti i suoi elementi nella forma:

```
ARRAY=(VALORE 1 VALORE 2 ... VALORE N)
```

- Occorre fare molta attenzione al fatto che l'indice di un vettore parte sempre da zero!

Bash: vettori

- Usando `declare`

```
declare -a VETTORE
```

```
declare -a ALTRO_VETTORE[10]
```

- Nel secondo caso abbiamo creato un vettore di 10 elementi, il primo è `ALTRO_VETTORE[0]`, mentre l'ultimo `ALTRO_VETTORE[9]`.
- Mediante assegnazione di un elemento, in questo caso il primo per ricordare che L'INDICE PARTE DA 0!

```
VETTORE[0]=10
```

- Attraverso l'assegnazione di tutti i suoi elementi:

```
VETTORE=(11 24 32 88 55)
```



Bash: vettori

```
#!/bin/bash

declare -a VETTORE[3]
VETTORE[0]="Tutto e andato bene"
VETTORE[1]="Hai inserito un numero
  maggiore di 20"
VETTORE[2]="Hai inserito un numero
  minore di 10"

inserisci_numero () {
  echo "Inserisci un numero compreso tra
  10 e 20"
  read NUMERO
  if [ "$NUMERO" -lt 10 ]; then
    return 2
  elif [ "$NUMERO" -gt 20 ]; then
    return 1
  else
    return 0
  fi
}

inserisci_numero
RITORNO=$?
echo ${VETTORE[$RITORNO]}
exit $RITORNO
```