

# Controllo di flusso negli script: if-then-else

Il comando condizionale

```
if condition_command
then
    true_commands
else
    false_commands
fi
```

esegue il comando `condition_command` e utilizza il suo **exit status** per decidere se eseguire i comandi `true_commands` (exit status 0) od i comandi `false_commands` (exit status diverso da zero).

Ad esempio lo script seguente prende come argomento un login name e stampa a video un messaggio diverso a seconda se il parametro fornito compaia all'inizio di una linea del file `/etc/passwd` oppure no:

```
if grep "^$1:" /etc/passwd >/dev/null 2>/dev/null
then
    echo $1 is a valid login name
else
    echo $1 is not a valid login name
fi
```

```
exit 0
```

# Condizioni: exit status e comando test (I)

Se la condizione che si vuole specificare non è esprimibile tramite l'exit status di un "normale" comando, si può utilizzare l'apposito comando `test`:

`test expression`

che restituisce un exit status pari a 0 se `expression` è vera, pari a 1 altrimenti.

Si possono costruire vari tipi di espressioni:

- espressioni che controllano se un file possiede certi attributi:

- e *f* restituisce vero se *f* esiste;

- f *f* restituisce vero se *f* è un file ordinario;

- d *f* restituisce vero se *f* è una directory;

- r *f* restituisce vero se *f* è leggibile dall'utente;

- w *f* restituisce vero se *f* è scrivibile dall'utente;

- x *f* restituisce vero se *f* è eseguibile dall'utente;

- espressioni su stringhe:

- z *str* restituisce vero se *str* è di lunghezza zero;

- n *str* restituisce vero se *str* non è di lunghezza zero;

- str1* = *str2* restituisce vero se *str1* è uguale a *str2*;

- str1* != *str2* restituisce vero se *str1* è diversa da *str2*;

## Condizioni: exit status e comando test (II)

...

- espressioni su valori numerici:

*num1 -eq num2* restituisce vero se *num1* è uguale a *num2*;

*num1 -ne num2* restituisce vero se *num1* non è uguale a *num2*;

*num1 -lt num2* restituisce vero se *num1* è minore di *num2*;

*num1 -gt num2* restituisce vero se *num1* è maggiore di *num2*;

*num1 -le num2* restituisce vero se *num1* è minore o uguale a *num2*;

*num1 -ge num2* restituisce vero se *num1* è maggiore o uguale a *num2*

- espressioni composte:

*exp1 -a exp2* restituisce vero se sono vere sia *exp1* che *exp2*

*exp1 -o exp2* restituisce vero se è vera *exp1* o *exp2*

*!exp* restituisce vero se non è vera *exp*

La shell fornisce anche la possibilità di costruire espressioni numeriche complesse, da utilizzare con il comando di test, tramite la sintassi seguente:

```
#[expression]
```

Ad esempio:

```
> num1=2
```

```
> num1=${num1*3+1}
```

```
> echo $num1
```

# Controllo di flusso negli script: cicli while

Sintassi:

```
while condition_command
do
    commands
done
```

L'effetto risultante è che vengono eseguiti i comandi `commands` finché la condizione `condition_command` è vera. Esempio:

```
while test -e $1
do
    sleep 2
done

echo file $1 does not exist
exit 0
```

Lo script precedente esegue un ciclo che dura finché il file fornito come argomento non viene cancellato. Il comando che viene eseguito come corpo del `while` è una pausa di 2 secondi.

# Controllo di flusso negli script: cicli until

Sintassi:

```
until condition_command
do
    commands
done
```

L'effetto risultante è che vengono eseguiti i comandi `commands` finché la condizione `condition_command` è **falsa**. Esempio:

```
until false
do
    read firstword restofline
    if test $firstword = end
    then
        exit 0
    else
        echo $firstword $restofline
    fi
done
```

Lo script precedente legge continuamente dallo standard input e visualizza quanto letto sullo standard output, finché l'utente non inserisce la stringa `end`.

# Controllo di flusso negli script: cicli for

Sintassi:

```
for var in wordlist
do
    commands
done
```

L'effetto risultante è che vengono eseguiti i comandi `commands` per tutti gli elementi contenuti in `wordlist` (l'elemento corrente è memorizzato nella variabile `var`). Esempio:

```
for i in 1 2 3 4 5
do
    echo the value of i is $i
done
```

```
exit 0
```

L'output dello script precedente è:

```
the value of i is 1
the value of i is 2
the value of i is 3
the value of i is 4
the value of i is 5
```

# Controllo di flusso negli script: case selection

Sintassi:

```
case string in
expression_1)
    commands_1
    ;;
expression_2)
    commands_2
    ;;
...
*)
    default_commands
    ;;
esac
```

L'effetto risultante è che vengono eseguiti i comandi `commands_1`, `commands_2`,... a seconda del fatto che `string` sia uguale a `expression_1`, `expression_2`,...

I comandi `default_commands` vengono eseguiti soltanto se il valore di `string` non coincide con nessuno fra `expression_1`, `expression_2`,...

I valori `expression_1`, `expression_2`,... possono essere specificati usando le solite regole per l'espansione del percorso (caratteri jolly).

# Esempio d'uso del costrutto di case selection

Supponiamo di avere il seguente script memorizzato nel file `append`:

```
case $# in
1)
    cat >>$1
    ;;
2)
    cat >>$1 <$2
    ;;
*)
    echo "usage: append out_file [in_file]"
    ;;
esac

exit 0
```

Lo script precedente controlla che il numero degli argomenti forniti (variabile `$#`) sia 1 o 2 (a seconda se l'input da accodare al primo argomento debba provenire dallo standard input o da un altro file specificato sulla linea di comando), altrimenti stampa un messaggio che illustra l'utilizzo dello script.

# Command substitution

Il meccanismo di **command substitution** permette di sostituire ad un comando o pipeline quanto stampato sullo standard output da quest'ultimo.

Esempi:

```
> date
```

```
Tue Nov 19 17:50:10 2002
```

```
> vardata='date'
```

```
> echo $vardata
```

```
Tue Nov 19 17:51:28 2002
```

Un comando molto usato con le command substitution è `basename` (restituisce il nome di un file, senza il path):

```
> basefile='basename /usr/bin/man'
```

```
> echo $basefile
```

```
man
```

**Importante:** per operare una command substitution si devono usare gli “apici rovesciati” o backquote (```), non gli apici normali (`'`) che si usano come meccanismo di quoting.

## Esempio (I)

Progettare uno script, chiamato `listfiles`, che prende due parametri, una `directory` e la dimensione di un file in byte. Lo script deve fornire il nome di tutti i file regolari contenuti nella `directory` parametro ai quali avete accesso e che sono più piccoli della dimensione data. Si controlli che i parametri passati sulla linea di comando siano due e che il primo sia una `directory`.

Esempio di soluzione (prima parte: controllo dei parametri):

```
if test $# -ne 2
then
    echo 'usage: listfiles <dirpath> <dimensione>'
    exit 1
fi
if ! test -d $1
then
    echo 'usage: listfiles <dirpath> <dimensione>'
    exit 1
fi
```

## Esempio (II)

Esempio di soluzione (seconda parte: esecuzione del compito stabilito nell'esercizio):

```
for i in $1/*
do
    if test -r $i -a -f $i
    then
        size='wc -c <$i'
        if test $size -lt $2
        then
            echo 'basename $i' has size $size bytes
        fi
    fi
done

exit 0
```

# Esercizi

- Progettare uno script che prende in input come parametri i nomi di due directory e copia tutti i file della prima nella seconda, trasformando tutte le occorrenze della stringa SP in SU in ogni file.
- Progettare uno script `drawsquare` che prende in input un parametro intero con valore da 2 a 15 e disegna sullo standard output un quadrato (utilizzando i caratteri +, - e |) come nel seguente esempio:

```
> drawsquare 4
```

```
+--+
```

```
|  |
```

```
|  |
```

```
+--+
```

- Progettare uno script che prende in input come parametro il nome di una directory e cancella tutti i file con nome `core` dall'albero di directory con radice la directory parametro.