

Shell Script

Gli **shell script** sono **programmi** interpretati dalla shell, scritti in un linguaggio i cui costrutti atomici sono i **comandi Unix**. I comandi possono essere combinati in sequenza o mediante i costrutti usuali di un linguaggio di programmazione. La sintassi varia da shell a shell. Faremo riferimento alla shell `bash`. Gli shell script sono la base degli *scripting languages*, come *Perl*.

Uno shell script va scritto in un file utilizzando per esempio il comando `cat` o un editor (`vi`, `emacs`, etc). Per poter eseguire lo script, il file deve essere reso **eseguibile**. Lo script viene eseguito invocando il nome del file.

Esempio

```
> cat >dirsize          # il file dirsize viene editato
ls /usr/bin | wc -w
Ctrl-d                  # fine dell'editing
> chmod 700 dirsize     # dirsize viene reso (in particolare) eseguibile
> dirsize               # viene invocato il comando dirsize
459                     # risultato dell'esecuzione
```

Esecuzione di script

Lo script viene eseguito in una **sottoshell** della shell corrente.

Il comando `set -v/set -x` fa sì che durante l'esecuzione di uno script la shell visualizzi i comandi nel momento in cui li legge/segue (`set -` annulla l'effetto di `set -v/set -x`). Ciò è utile per il debugging.

```
> cat >data
```

```
set -x
```

```
echo the date today is:
```

```
date
```

```
Ctrl-d
```

```
> chmod u+x data
```

```
> data
```

```
++ echo the date today is:
```

```
the date today is:
```

```
++ date
```

```
Tue Oct 25 17:37:52 CEST 2005
```

```
# contenuto dello script data
```

```
# ...
```

```
# lo script viene invocato
```

```
# ... la shell visualizza
```

```
# i comandi mentre
```

```
# li esegue
```

```
# ...
```

... esecuzione di script

```
> cat >sost
```

```
set -v # contenuto dello script sost
```

```
cd TEXT # ...
```

```
ls *.txt
```

```
sed s/'#'/';;;'/ file.txt
```

```
Ctrl-d
```

```
> more TEXT/file.txt
```

```
# questo e' un commento
```

```
# in un programma
```

```
> chmod u+x sost
```

```
> sost # lo script viene invocato
```

```
cd TEXT # ... la shell visualizza
```

```
ls *.txt # i comandi
```

```
file.txt # mentre
```

```
sed s/'#'/';;;'/ file.txt # li legge
```

```
;;; questo e' un commento # output del comando sed
```

```
;;; in un programma # ...
```

Variabili

Le **variabili** della shell sono stringhe di caratteri a cui è associato un certo spazio in memoria. Il **valore** di una variabile è una **stringa di caratteri**. Le variabili della shell possono essere utilizzate sia sulla linea di comando che negli script.

Non c'è dichiarazione esplicita delle variabili.

Assegnamento di una variabile (eventualmente nuova): `variabile=valore`
(Importante: non lasciare spazi a sinistra ed a destra dell'operatore =)

```
> x=variabile
```

```
> y='y e' una variabile'
```

Per **accedere** al valore di una variabile si utilizza il \$:

```
> echo il valore di x: $x
```

```
il valore di x: variabile
```

```
> echo il valore di y: $y
```

```
il valore di y: y e' una variabile
```

```
> echo y
```

```
y
```

Le variabili sono **locali** alla shell o allo script in cui sono definite. Per rendere globale una variabile (**variabile d'ambiente**) si usa il comando `export`:

```
> export x          # promuove x a variabile di ambiente
```

Variabili di ambiente

Le **variabili di ambiente** sono variabili globali. Esiste un insieme di variabili di ambiente speciali riconosciute dalla shell e definite al momento del login:

VARIABILE	SIGNIFICATO
PS1	prompt della shell
PS2	secondo prompt della shell; utilizzato per esempio in caso di ridirezione dell'input dalla linea di comando
PWD	pathname assoluto della directory corrente
UID	ID dello user corrente
PATH	lista di pathname di directory in cui la shell cerca i comandi
HOME	pathanme assoluto della home directory

Esempi d'uso:

```
> echo il valore di PATH: $PATH
```

```
il valore di PATH: /usr/bin:/usr/openwin/bin:/usr/local/bin
```

Le variabili di ambiente possono essere modificate:

```
> PS1='salve: '
```

```
salve:
```

```
> PATH=./bin:$PATH
```

```
> echo il valore di PATH: $PATH
```

```
il valore di PATH: ./bin:/usr/bin:/usr/openwin/bin:/usr/local/bin
```

Parametri

Le variabili \$1, \$2, ..., \$9 sono variabili speciali associate al primo, secondo, ..., nono parametro passato sulla linea di comando quando viene invocato uno script:

```
> cat >copia
mkdir $1
mv $2 $1/$2
Ctrl-d
```

```
> copia nuovadir testo
> ls nuovadir
testo
```

Se uno script ha più di 9 parametri, si utilizza il comando `shift` per fare lo shift a sinistra dei parametri e poter accedere ai parametri oltre il nono:

```
> cat >stampa_decimo
shift
echo decimo parametro: $9
Ctrl-d
> stampa_decimo 1 2 3 4 5 6 7 8 9 10
decimo parametro: 10
```

Variabili di stato automatiche (I)

Sono variabili speciali che servono per gestire lo **stato** e sono aggiornate **automaticamente** dalla shell. L'utente può accedervi solo in lettura.

Al termine dell'esecuzione di ogni comando unix, viene restituito un valore di uscita, **exit status**, uguale a 0, se l'esecuzione è terminata con successo, diverso da 0, altrimenti (codice di errore).

La variabile speciale `$?` contiene il valore di uscita dell'ultimo comando eseguito.

```
> cd
> echo $?
0

> copia nuovadir testo
> echo $?
0

> copia nuovadir testo
mkdir: Failed to make directory "nuovadir"; File exists
mv: cannot access testo
> echo $?
2
```

Il comando `exit n`, dove `n` è un numero, usato all'interno di uno script, serve per terminare l'esecuzione e assegnare alla variabile di stato il valore `n`.

Variabili di stato automatiche (II)

La shell `bash` mette a disposizione numerose variabili di stato; le principali sono:

Variabile	Contenuto
<code>\$?</code>	<i>exit status</i> dell'ultimo comando eseguito dalla shell
<code>\$\$</code>	PID della shell corrente
<code>\$!</code>	il PID dell'ultimo comando eseguito in background
<code>\$-</code>	le opzioni della shell corrente
<code>\$#</code>	numero dei parametri forniti allo script sulla linea di comando
<code>*\$</code> , <code>\$@</code>	lista di tutti i parametri passati allo script sulla linea di comando

In particolare `$$` viene usata per generare nomi di file temporanei che siano unici fra utenti diversi e sessioni di shell diverse, e.g., `/tmp/tmp$$`.

Login script

Il **login script** è uno script speciale eseguito **automaticamente** al momento del login. In (alcune versioni di) Unix/Linux tale script è contenuto in uno dei file `.bash_profile`, `.bash_login`, `.bashrc`, `.profile`, memorizzati nella home directory degli utenti.

Il login script contiene alcuni comandi che è utile eseguire al momento del login, come la definizione di alcune variabili di ambiente.

Ciascun utente può modificare il proprio login script, ad esempio (ri)definendo variabili di ambiente e alias “permanenti”.

Esiste anche uno script di login *globale* contenuto nel file `/etc/profile` in cui l'amministratore di sistema può memorizzare dei comandi di configurazione che valgano per tutti (tale script è infatti eseguito prima di quelli dei singoli utenti).

Lo script di logout eseguito al momento dell'uscita dalla shell, si chiama solitamente `.bash_logout`.

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 (I)

- Creare una sottodirectory `bin` all'interno della propria home directory in cui mettere gli script. Fare in modo che gli script contenuti in `bin` possano essere invocati da qualunque directory con il nome del file, senza dover specificare l'intero pathname.
- Qual è l'effetto della seguente sequenza di comandi? Perché?

```
> cat >chdir  
cd ..  
Ctrl-d  
> chmod 700 chdir  
> chdir  
> pwd
```

- Creare un alias permanente `lo` per il comando `exit`.
- Progettare uno script che prende come parametro una stringa e un file di testo e controlla se la stringa compare nel file.

Esercizi (II)

- Il comando `read` assegna alla variabile speciale `REPLY` un testo acquisito da standard input. Qual è l'effetto dello script `words` contenente i seguenti comandi?

```
echo -n 'Enter some text: '  
read one two restofline  
echo 'The first word was: $one'  
echo 'The second word was: $two'  
echo 'The rest of the line was: $restofline'  
exit 0
```

- Qual è l'effetto della seguente sequenza di comandi? Perché?

```
> cat >data  
echo -n the date today is:  
date  
Ctrl-d  
> chmod 700 data  
> data
```

Esercizi (III)

- Scrivere uno script che estragga soltanto i commenti dal file con estensione `java` fornito come primo argomento, sostituendo `//` con la stringa `linea di commento del file <nome del file>:.`. Inoltre i commenti estratti devono essere salvati nel file fornito come secondo argomento.
- 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.

Esercizi (IV)

- 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.

Schema riassuntivo shell Bash

Argomento	Comandi visti a lezione
Shell e sessione	sh, bash, csh, tcsh, ksh, logout, exit, history, alias, unalias, man, who, date
File e directory	pwd, cd, ls, mkdir, rmdir, cp, mv, rm, ln, chmod, touch, mount, df, du, basename, cmp, diff, find
Processi e job	ps, top, kill, jobs, fg, bg
Visualizzazione	echo, cat, more, less, tail, head
Filtro	wc, uniq, grep, fgrep, egrep, sort, tr, cut, paste
Editor	vi, emacs, xemacs, mc, sed
Script	set, shift, if then else fi, while do done, until do done, for in do done, case in esac, read

Nota: Lo schema riassuntivo relativo ai **metacaratteri** si trova nelle slide della Lezione 3.