Sistemi Operativi, Secondo Modulo, Canale M–Z
Riassunto della lezione del 06/04/2017

Igor Melatti

Gli Script Bash

- Si possono anche dichiarare e chiamare delle funzioni
  - al solito, la sintassi è alquanto particolare: per la *dichiarazione*, occorre scrivere `function nomefunzione () { commands; }`
  - la parte tra parentesi graffe è esattamente un group command, con tutte le limitazioni sintattiche del caso (spazi obbligatori dopo { e prima di }, occorre il `;` o l'andata a capo prima di })
  - è possibile aggiungere delle redirezioni; verranno effettuate solo al tempo della chiamata
  - non si scrivono mai gli argomenti: dopo `nomefunzione` c'è sempre `()` (tra l'altro, sono sintatticamente opzionali)
  - gli argomenti, come anticipato in altre lezioni, sono gestiti come gli argomenti degli script, usando i parametri posizionali
  - quindi, i parametri posizionali, quando usati nel corpo di una funzione, non sono più gli argomenti dello script ma quelli della funzione
  - una volta terminata una chiamata, i parametri posizionali tornano ad essere quelli dello script (a meno che non si trattasse di una funzione che ne aveva chiamata un'altra..)
  - sintassi per la *chiamata*: `nomefunzione arg1 ...argn` (separati da spazi)
  - quindi nessun controllo sugli argomenti, né su quanti sono né sul tipo
  - il nome di una funzione è un identificatore, ma si tratta di identificatori separati da quelli per le variabili
  - si possono definire funzioni con lo stesso nome dei comandi; nel qual caso, hanno precedenza le funzioni (per ripristinare la situazione precedente, usare `unset`)
  - si possono fare funzioni ricorsive: `function fib () { local fm1; local fm2; if [ $1 -lt 3 ]; then return 1 ; else fib $(($1 - 1)) ; fm1=$?; fib $(($1 - 2)); fm2=$?; return $((fm1 + fm2)); fi; }`

  - notare le parole chiave `return` e `local`
  - senza `local`, si suppone che le variabili siano globali, quindi la funzione di cui sopra non funzionerebbe
  - il `return` è come l'`exit` degli script (in effetti, può essere usato anche dentro uno script, ed è uno dei pochi casi, se non l'unico caso, di comando che si può usare solo in uno script e non nella shell interattiva); provare quindi a chiamare `fib 13` e `fib 14`...
  - **esercizio:** usando una variabile d'appoggio `funres` anziché `return`, fare in modo che la funzione `fib` possa andare oltre il valore 13 (ovviamente, si arriva fino ad un certo massimo che dipende dal sistema, e dovrebbe essere tra 90 e 100...)
  - **esercizio:** riconsiderare l'esercizio sul `case` dato nella lezione 14, e far sì che il parsing della riga di comando (opzioni ed argomenti) sia fatto da una funzione

- Nota: il comando `source` può essere usato, in pratica, per includere uno script dentro un altro

- *Here documents*
  - altra tecnica di redirection, vale solo per l'input
  - anziché leggere da file, si legge direttamente dalla riga di comando
  - sintassi:

    ```
    comando << parolaDiFineInput
    riga di input 1
    ...
    riga di input n
    parolaDiFineInput
    ```

  - semanticamente, è come se si fosse fatto il redirect in input di un file che contiene le n righe:

    ```
    riga di input 1
    ...
    riga di input n
    ```

  - con una importante differenza: nell'here documents avvengono le espansioni dei parametri (anche aritmetiche) e del command substitution
  - ma in modo "strano": sia i single che i double quotes restano così come sono (niente quote removal), quindi ad esempio se `var` vale 10 allora `'$var'` viene espansa in `'10'`
  - l'unico modo per evitare l'espansione dei parametri è escapparli con il backslash
  - ovviamente, 2 backslash risultano in un backslash solo

  - solitamente, `parolaDiFineInput` viene scelta in modo che non appaia (come riga singola) all'interno dell'input voluto (altrimenti, l'input verrebbe troncato...)
  - scelte comuni sono `EOF`, o il comando al contrario (ad esempio, `tac` se il comando è `cat`)

- Vedere esempi allegati