# Sistemi Operativi, Secondo Modulo, Canale M–Z
# Riassunto della lezione del 30/03/2017

Igor Melatti

## La Bash, per davvero

- Per le variabili è possibile (ma obbligatorio solo per gli array associativi) specificare un tipo tra:
  - stringa: tipo di default, accetta qualsiasi assegnamento
  - intero: `declare -i`. Se gli si assegna una stringa, la variabile prende il valore 0; per essere precisi: se gli si assegna un valore, allora viene invocata la valutazione aritmetica;
  - array indicizzato da interi: `declare -a`
  - array indicizzato da stringhe (array associativo): `declare -A` (dichiarazione obbligatoria)
  - variabile costante (ossimoro...): `declare -r` (occorre averci assengato un valore in precedenza...); attenzione, non è reversibile!
- Variabili di tipo array
  - `varArray=(`$v_0$ `...`$v_{n-1}$`)`, dove i valori $v_i$ sono separati da spazi (e possono essere eterogenei, con stringhe mischiate ad interi)
  - si accede all'elemento di indice $i$ così: `${varArray[`$i$`]}`
  - si può assegnare anche il singolo elemento: `varArray[`$i$`]=`$v_i$ (non è necessario aver settato i precedenti $i$: gli array sono *sparsi*)
  - si possono specificare anche gli indici: `varArray=([`$i_0$`]=`$v_0$ `...[`$i_{n-1}$`]=`$v_{n-1}$`)`
  - per gli array associativi, gli indici sono stringhe; per il resto è tutto uguale
  - gli indici possono essere espansioni di variabili
  - **esercizio:** creare un array dove sono definiti i seguenti indici: `3`, `5`, `10`, `100`, con valore pari al doppio dell'indice; farsi poi stampare il valore agli indici `3` e `4`; usando la Tabella 3, farsi stampare anche la lunghezza (come numero di caratteri) di tali valori; farsi inoltre stampare tutti gli indici così settati, e farsi anche stampare il numero totale di indici settati

  - **esercizio:** creare un array dove sono definiti i seguenti indici: `3`, `5a`, `10df`, `100`, con valore pari al doppio della parte numerica dell'indice; farsi poi stampare il valore agli indici `5a` e `4`; usando la Tabella 3, farsi stampare anche la lunghezza (come numero di caratteri) di tali valori; farsi inoltre stampare tutti gli indici così settati, e farsi anche stampare il numero totale di indici settati

- Metacaratteri e *quoting*
  - i metacaratteri sono caratteri speciali, che sono interpretati da bash in un modo peculiare per ciascuno di essi
  - sono solo i seguenti: `(`, `)`, `|`, `&`, `;`, `<`, `>`
  - l'interpretazione è già stata discussa: inizio e fine sottoshell, pipelining o esecuzione condizionale, esecuzione in background o esecuzione condizionale o redirezioni, terminatore di comando, redirezioni
  - altri caratteri hanno significato speciale in alcuni contesti: `$`, `!`, `,`, {, }, `*`, `+` (alcuni già visti, altri da vedere)
  - per usarli nel loro senso letterale, occorre il quoting:
    * mettendo davanti a ciascun metacarattere un \; se si va a capo subito dopo un \, il comando continua nella riga sottostante
    * usando la sintassi speciale `$'carattere'`; se si vuole stampare `'`, occorre scrivere `$'\''`; se si vuole stampare \, occorre scrivere `$'\\'`; si possono usare le escape sequence dei comandi `printf`: `\n` è l'andata a capo (carriage return + line feed), `\r` solo il carriage return, `\f` solo il line feed, `\t` è la tabulazione
    * racchiudendoli tra single quote `'`. All'interno, non ci possono essere altri `'`, nemmeno con il backslash davanti; tutti gli altri caratteri speciali perdono il loro essere speciali e vengono scritti così come appaiono
    * racchiudendoli tra double quote `"`. All'interno, alcuni caratteri speciali, come `!` (history expansion, ci ritorneremo), `$` (espansione dei parametri, descritta in questa lezione, più altre espansioni che vedremo in seguito) e `` ` `` (command substitution, ci ritorneremo) mantengono il loro significato speciale; anche \, ma solo se è seguito da uno di questi 3 caratteri o da `"`.
  - **esercizio:** Scrivere uno script che, usando il comando `echo`, si fa stampare tutti i metacaratteri riportati sopra, usando tutti i metodi descritti. Alla fine, farsi stampare anche le seguenti stringhe: `'ciao'`, `"ciao"`, `"'ciao'"`, `'"ciao"'`
  - **esercizio:** come sopra ma, anziché `ciao`, stampare 2 volte `ciao`, con una tabulazione in mezzo
  - **esercizio:** sperimentare la differenza tra carriage return, line feed e la combinazione dei 2

Table 1: I tipi di espansione di bash, e loro ordinamento. Nella terza colonna sono riportati i casi in cui l'espansione avviene anche se all'interno di double quotes `""`; se vengono usate le single quotes, allora non avviene mai alcuna espansione (a meno che non siano a loro volta dentro le double quotes...)

| **Espansione** | **Ordine** | **Double quotes** |
|---|---|---|
| Brace expansion | 1 | NO |
| Tilde expansion | 2 | NO |
| Parameter and variables expansion | 3 | SÌ |
| Arithmetic expansion | 4 | SÌ |
| Command expansion (o substitution) | 5 | SÌ |
| Word splitting | 6 | NO |
| Filename expansion (o globbing, o wildcarding) | 7 | NO |
| (Process substitution, non sempre disponibile) | 8 | NO |
| (Quote removal) | 9 | |

- Funzionamento di bash: l'*expansion* (vedere anche `http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_04.html`)
  - alcune già viste: il globbing o filename expasion (espansione delle wildcard per formare un filename), l'espansione delle variabili, la tilde che indica la home directory
  - ce ne sono anche altre; l'elenco completo è in Tabella 1, che mostra anche l'ordine con cui vengono applicate
  - quello che accade è che la bash, prima di eseguire un comando, effettua in ordine le espansioni riportate in Tabella 1
  - quindi, quale comando viene effettivamente eseguito (e con quali argomenti/opzioni) dipende da tali espansioni
  - per esempio: il comando `ls ~/*.txt` dapprima diventa `ls /home/utente/*.txt` (per la tilde expansion) e poi `ls /home/utente/file.txt` (filename expansion, assumendo che l'unico file con estensione `.txt` nella home di `utente` sia `file.txt`)
  - se l'ordine di espansione fosse stato l'inverso, si sarebbe ottenuto lo stesso risultato?
  - le espansioni possono essere in cascata, ovvero al risultato di una espansione si applicano le seguenti (ma nell'ordine di Tabella 1)
    - eccezione: non può succedere di applicare la variable expansion al risultato di una tilde expression, perché la tilde expression dentro delle parentesi graffe (senza spazi) non è riconosciuta come tale (deve essere staccata da metacaratteri o caratteri speciali)
- Brace expansion: espansione delle parentesi graffe (*brace expression*)

Table 2: Tilde expressions e tilde expansions

| **Tilde expression** | **Espansione** |
|---|---|
| (stringa vuota) | `$HOME` |
| `nomeutente` | Home directory di `nomeutente` (se esistente) |
| `+` | `PWD` |
| `-` | `OLDPWD` |
| `+n` | output di `dirs +n` |
| `-n` | output di `dirs -n` |

  - si tratta di una mini-espressione regolare, che riconosce un linguaggio finito, ma con una sintassi diversa da quella vista finora
  - due forme sintattiche: $\{p_1\texttt{,}\ldots\texttt{,}p_n\}$ e $\{v\texttt{..}w\}$ (attenzione: niente spazi all'interno delle parentesi)
  - prima forma: le $p_i$ sono stringhe qualsiasi; se serve che contengano { o `,` occorre quotarli
  - seconda forma: $v$ e $w$ devono essere valori interi (altrimenti, non è più una brace expression)
  - la prima forma viene espansa in una stringa che contiene $p_1 \ldots p_n$, separati da uno spazio
  - la seconda forma viene espansa in una stringa che contiene $v\ v+1 \ldots w$ (sempre separati da uno spazio), oppure $v\ v-1 \ldots w$, a seconda se $v < w$ oppure il contrario
  - c'è una terza forma, variante della seconda: $\{v\texttt{..}w\texttt{..}k\}$, che viene espansa in $v\ v+k \ldots v+ik$ (con $i$ più grande intero tale che $v+ik \leq w$), oppure $v\ v-k \ldots x-ik$ (con $i$ più grande intero tale che $v-ik \geq w$), a seconda se $v < w$ oppure il contrario
  - **esercizio:** usando la brace expansion, farsi stampare tutti i numeri tra 0.0 e 10.0, con step di 0.1
  - **esercizio:** usando la brace expansion, farsi stampare tutti i numeri tra 0.0 e 10.0, con step di 0.5

- Tilde expansion: espansione del carattere tilde (*tilde expression*)
  - non solo home: vedere Tabella 2
  - `dirs` è un comando usato in concomitanza con `pushd` e `popd`, che mantengono uno stack di directory (vedere `man bash`)
  - **esercizio:** assegnare ad una variabile la home di `utente1`, e poi stamparla

- Espansione di parametri e variabili, è quella con più possibilità
  - in Tabella 3 vengono riportate le principali

Table 3: Variable and parameter expressions e expansions

| **Espressione** | **Espansione (var definita)** | **Espansione (var non definita)** |
|---|---|---|
| `$nomevar` | valore di `nomevar` | stringa vuota o 0 |
| `${nomevar}` | valore di `nomevar` | stringa vuota o 0 |
| `${nomevar:-stringa}` | `${nomevar}` | `stringa` |
| `${nomevar:+stringa}` | `stringa` | stringa vuota o 0 |
| `${nomevar:?stringa}` | `${nomevar}` | scrive `stringa` su stderr; se la shell è interattiva, abortisce la parte restante del comando; se la shell non è interattiva, abortisce l'intera shell |
| `${nomevar:=stringa}` | `${nomevar}` | prima assegna `stringa` a `nomevar`, e poi espande `nomevar` |
| `${nomevar:offset}` | `${nomevar}` dalla posizione `offset` in poi (si comincia da 0); se `offset` è negativo (attenzione: allora mettere uno spazio prima del `-`) allora la posizione è data dalla lunghezza di `${nomevar}` cui si sottrae offset | stringa vuota o 0 |
| `${nomevar:offset:length}` | come sopra, ma espande al più `length` caratteri | come sopra |
| `${#nomevar}` | lunghezza dell'espansione di `${nomevar}`; se si tratta di un array, allora lunghezza dell'espansione di `${nomevar[0]}` | 0 |
| `${#nomevar[i]}` | lunghezza dell'espansione di `${nomevar[i]}` | 0 |
| `${!nomevar[*]}` | lista degli indici assegnati nell'array `nomevar` | 0 |
| `${!nomevar[@]}` | lista degli indici assegnati nell'array `nomevar`; come per `$@` e `$*`, la differenza la fa il word splitting (vedere lezione 13) | 0 |
| `${!pref*}` | lista dei nomi di variabili definite che iniziano con `pref` | |
| `${!pref@}` | lista dei nomi di variabili definite che iniziano con `pref`; come per `$@` e `$*`, la differenza la fa il word splitting (vedere lezione 13) | |

Continuazione di Tabella 3.

| **Espressione** | **Espansione (var definita)** | **Espansione (var non definita)** |
|---|---|---|
| `${nomevar#pattern}` | prima espande `nomevar`, poi toglie la più corta sequenza iniziale che faccia match con `pattern` (i pattern solo gli stessi per il file globbing, vedere fine della lezione 5) | stringa vuota o 0 |
| `${nomevar##pattern}` | prima espande `nomevar`, poi toglie la più lunga sequenza iniziale che faccia match con `pattern` | stringa vuota o 0 |
| `${nomevar%pattern}` | prima espande `nomevar`, poi toglie la più corta sequenza finale che faccia match con `pattern` | stringa vuota o 0 |
| `${nomevar%%pattern}` | prima espande `nomevar`, poi toglie la più lunga sequenza finale che faccia match con `pattern` | stringa vuota o 0 |
| `${nomevar/pattern/string}` | simile al replace di `sed`: prima espande `nomevar`, poi sostituisce con `string` la più lunga sottostringa che faccia match con `pattern` | stringa vuota o 0 |
| `${nomevar//pattern/string}` | prima espande `nomevar`, poi sostituisce con `string` tutte le sottostringhe che facciano match con `pattern` | stringa vuota o 0 |
| `${nomevar/#pattern/string}` | prima espande `nomevar`, poi sostituisce con `string` la più lunga sottostringa iniziale che faccia match con `pattern` | stringa vuota o 0 |
| `${nomevar/%pattern/string}` | prima espande `nomevar`, poi sostituisce con `string` la più lunga sottostringa finale che faccia match con `pattern` | stringa vuota o 0 |

Continuazione di Tabella 3.

| **Espressione** | **Espansione (var definita)** | **Espansione (var non definita)** |
|---|---|---|
| `${nomevar^pattern}` | prima espande `nomevar`, poi converte il primo carattere del risultato, trasformandolo in maiuscolo, purché faccia match con `pattern`. Attenzione, questi pattern devono fare match con un singolo carattere (ad esempio, un pattern che fa la concatenazione di due letterali non avrà effetto) | stringa vuota o 0 |
| `${nomevar,pattern}` | come sopra, ma trasforma in minuscolo | stringa vuota o 0 |
| `${nomevar^^pattern}` | prima espande `nomevar`, poi converte tutti i caratteri che facciano match con `pattern`, trasformandoli in maiuscolo | stringa vuota o 0 |
| `${nomevar,,pattern}` | prima espande `nomevar`, poi converte tutti i caratteri che facciano match con `pattern`, trasformandoli in minuscolo | stringa vuota o 0 |

- **esercizio:** verificare che una variabile intera, se non assegnata precedentemente, viene espansa con 0
- **esercizio:** assegnare alle variabili `v1` e `v2` i valori 1 e 2, rispettivamente, e poi farsi stampare il loro valore separato da _
- **esercizio:** farsi stampare il valore della variabile `v1`, oppure `la variabile v1 non e' definita` se alla variabile non è stato ancora assegnato un valore (attenzione al quoting...)
- **esercizio:** farsi stampare `la variabile v1 e' definita` se alla variabile è già stato assegnato un valore, e la stringa vuota altrimenti
- **esercizio:** scrivere uno script che stampa per 3 volte, su 3 righe consecutive, il valore di una variabile d'ambiente `var_ambiente`, preceduta dalla stampa di una riga `inizio` e seguita dalla stampa della parola `fine`; se tale variabile non è stata definita, allora solo la prima riga dev'essere scritta. Non devono essere stampati altri messaggi. Verificare il corretto funzionamento dello script.
- **esercizio:** come sopra, ma se la variabile d'ambiente `var_ambiente` non è definita, allora assegnarle il valore della variabile `PATH`
- **esercizio:** scrivere uno script che stampi, del valore di `PATH`: i) gli ultimi 10 caratteri; ii) i caratteri dal decimo in poi; iii) i caratteri dal ventesimo al quarantesimo (compresi); iv) la lunghezza (come numero di caratteri)

– **esercizio:** farsi stampare i nomi di tutte le variabili d'ambiente che cominciano con `BASH_`

– **esercizio:** farsi stampare `PATH`, ma: i) senza la prima directory; ii) senza l'ultima directory; iii) con la sola prima directory; iv) con la sola ultima directory; v) con `lib` al posto di `usr`; vi) con `lib` al posto di `usr`, tranne che per la prima occorrenza di `usr`, che dev'essere sostituita con `var`; vii) solo l'ultima directory, e al posto di quelle precedenti deve scrivere `altre`; viii) solo la prima directory, e al posto di quelle precedenti deve scrivere `altre`; ix) con tutti i caratteri maiuscoli; x) con le sole lettere `u, s, r` messe in maiuscolo