Bash Command

▼ 1. Sintassi, comandi e path utili

$$comando\ [argomenti...]\ [modificatore]$$

- $apropos [nome\_comando]$: lista i comandi che contengono la stringa passata per parametro nella loro descrizione sintetica;
- $whatis\ [nome\_comando]$: descrizione sintetica del comando;
- $man\ [nome\_comando]$: manuale completo del comando;
- $info\ [nome\_comando]$: manuale alternativo (ottimizzato) del comando;

- $/etc/shells$: posizione delle shell(s) nel SO
- $echo\ \$BASH\_VERSION$: visualizza la versione in uso della shell bash
- $chsh$: cambia la shell
- $export\ [nome\_variabile]$: rende disponibile una variabile (anche quelle definite dalla shell), nei processi figli;

- $echo$: stampa a video;
- cat : stampa il contenuto di un file;
- wc : conta il numero di linee, caratteri o byte... presenti nel file;
- $less$: mostra su una singola pagina i dati ricevuti in input;
- $seq\ [first\_number\ incremento\ last\_number]$: genera una sequenza;
- $find\ pathname(s)\ [argomenti]$: visita ricorsivamente l'albero delle directory a partire dal pathname specificato selezionando e visualizzando i file in base alle specifiche introdotte tramite gli argomenti;

- $[[\ operatore\ operando\ ]]$ e $[[\ operando1\ operatore\ operando2]]$: consente di valutare alcune condizioni che riguardano stringhe o pathname
- $((espressione))$: valutata come espressione aritmetica del C, restituisce 0 (vero) se il valore dell'espressione e' diverso da zero;

▼ 2. Quoting

Forzare la shell a trattare un carattere speciale come se fosse un carattere normale.

- $\backslash[carattere]$: quoting singolo carattere;
- $'[stringa]'$: quoting gruppo di caratteri;
- $"[stringa]"$: quoting gruppo di caratteri ma alcuni caratteri speciali (es. $) rimangono tali;

```
#quoting examples
rm elenco\ gruppi
rm 'elenco gruppi'
rm "elenco gruppi"
```

▼ 3. Caratteri speciali e variabili predefinite

- $\#$: commento
- $\#!$: Indica la shell utilizzata nello script
- $\sim$: home directory
- $?$: Jolly. Durante l'espansione viene sostituito con un qualunque carattere;
- $*$: Jolly. Durante l'espansione viene sostituito con una qualunque stringa;
- $[]$: Jolly. Durante l'espansione viene sostituito con i caratteri presenti all'interno delle parentesi (o il range);
- $\{\}$: Durante l'espansione viene sostituito con la stringa presente all'interno delle parentesi (o il range);

```
rm libby1?.jpg #rimuove da libby10.jpg fino a libby19
rm libby*.jpg #rimuove tutti i file .jpg che il cui nome inizia con "libby"
rm libby1[1-5].jpg #rimuove da libby11.jpg a libby15.jpg
rm libby1[15].jpg #rimuove libby11.jpg E libby15.jpg
rm libby1.{txt,pdf} #rimuove tutti i file chiamati libby1 in formato pdf e txt
```

- $ ($\{\}$) : variabile da espandare
- $[space]$: anche lo spazio e' un carattere speciale e pertanto va quotato

Variabili predefinite:

- $\#$: numero di argomenti passati allo script (il nome dello script non viene contato);
- $* : lista degli argomenti passati, separati da spazi;
- $? : exit status dell'ultimo comando eseguito;
- $$: PID shell in esecuzione
- $! : PID ultimo processo eseguito in background

▼ 4. Redirezione dell'I/O

- $<$: redirect dell'input (stdin) al file specificato, in questo modo la fonte dello stdin e' il file e non la tastiera: $comando\ [argomento] < pathname$;
- $>$: redirect dell'output al file specificato (al posto di stdout). L'output del comando viene salvato nel file specificato. Se il file e' gia' presente viene sovrascritto altrimenti viene creato un nuovo file: $comando\ [argomento] > pathname$;
- $>>$: redirect dell'output al file specificato (al posto di stdout). L'output del comando viene salvato nel file specificato. Se il file e' gia' presente, il nuovo output viene accodato alla fine di esso altrimenti viene creato un nuovo file, $comando\ [argomento] >> pathname$;
- $|$: prende l'output del primo comando e lo usa come input del successivo: $comando1 \mid comando2 \mid ... \mid comandoN$

▼ 5. Costrutti

▼ If-then-else

```
if condizione1; then
        comando ...
elif condizione2; then
        comando ...
...
else
        comando ...
fi
```

▼ While

```
while condizione; do
        comando ...
done
```

▼ For

```
for var in arg ...; do
        comando ...
done
```