Appunti Esame Pratico

Parameter Expansion:

Sintassi	Uso	Esempio	Output
`$var` / `${var}`	Espande la variabile	`x=3; echo $x` / `${x}`	`3`
`${#var}`	Lunghezza stringa	`x=ciao; echo ${#x}`	`4`
`${var:offset:length}`	Estrae `length` caratteri da `offset` (0-based)	`s=abc; echo ${s:1:1}`	`b`

Command Substitution:

Sintassi	Uso	Esempio
`$(cmd)` / `` `cmd` ``	Sostituisce con output di `cmd`	`x=$(ls)`

Arithmetic Expansion:

Sintassi	Uso	Esempio	Output
`$((expr))`	Valuta espressione aritmetica e restituisce il risultato	`echo $((3+5))`	`8`
`((expr))`	Valuta ma restituisce l'esito	`((x>3))`	*status*

- l'operatore `$(( ))` racchiude solo UNA PARTE di una riga di comando che deve essere una espressione (più un eventuale assegnamento) valutandone aritmeticamente il contenuto
- l'operatore `(( ))` racchiude TUTTA una riga di comando semplice, che deve essere una
  espressione (più un eventuale assegnamento) e ne valuta aritmeticamente gli operandi.

Brace Expansion:

Sintassi	Cosa fa	Esempio	Risultato
`{a,b}`	Espande lista	`echo {a,b}`	`a b`

Sintassi	Cosa fa	Esempio	Risultato
`{1..5}`	Range numerico	`echo {1..3}`	`1 2 3`
`{a..c}`	Range alfabetico	`echo {a..c}`	`a b c`

Globbing Expansion:

Pattern	Espande	Esempio
`*`	qualsiasi stringa	`ls *.txt`
`?`	un carattere	`ls file?.txt`
`[abc]`	uno tra	`ls file[ac].txt`
`[a-z]`	range	`ls [a-z]*`
`[^a]`	negazione	`ls [^a]*`

History Expansion:

Sintassi	Restituisce
`!!`	ultimo comando
`!$`	ultimo argomento
`!n`	comando n

Variabili:

`var=5` / `var="abc cba"`

- non sono consentiti spazi prima o dopo '='
- `var="stringa" comando` → definisce una variabile temporanea utilizzata solo da uno specifico eseguibile (una variabile già definita tornerà ad avere il suo precedente valore subito dopo)
- `export var="abc"` → rende var una variabile d'ambiente e sarà ereditata dalle shell figlie
- per eliminare una variabile si usa `unset <nome variabile>`

Permessi:

- `chmod 777 file.sh` / `chmod u+x file.sh` → cambia i permessi associati a utenti e gruppi

Processi:

- `source <nome script>` / `. <nome script>` → esegue lo script nel processo corrente e non in una subshell (utile per condivisione variabili)
- `$#` → numero di argomenti passati allo script
- `$0` → nome dello script in esecuzione
- `$1`, `$2`, ... → argomenti in ordine (primo, secondo, ecc.)
- `$*` → tutti gli argomenti concatenati in un'unica stringa separata da spazi
- `$@` → tutti gli argomenti, ma ciascuno trattato come stringa separata (quotata)
- `$?` → exit status

Comando	Effetto	Note importanti
`comando &`	Avvia il comando in **background**	Il PID è in `$!`, il job ha un indice `%n`
`Ctrl + Z`	**Sospende** il processo in foreground	Il processo non termina
`Ctrl + C`	**Termina** il processo in foreground	Segnale SIGINT
`jobs`	Elenca i **job** attivi o sospesi	Mostra `%n`, **non PID**
`bg %n`	Riprende un job sospeso in **background**	Usa l'indice job
`fg %n`	Porta un job in **foreground**	Riattacca stdin
`kill PID`	Invia un segnale a un processo	PID reale
`kill %n`	Invia un segnale a un job	`%n` ≠ PID
`disown %n`	Sgancia il job dalla shell	Sopravvive alla chiusura
`ps -ax`	Lista **tutti** i processi	Non solo quelli della shell
`top`	Monitor interattivo processi	Sistema intero

Comandi Strutturati:

- `if [[ ]]` / `(( ))` / `comando; then ..; elif ..; else ..; fi`
- `while [[ ]]` / `(( ))` / `comando` / `read; do ..; done`
- `for name in $DIRS` / `/usr/include/*; do ..; done`

    - `for ((i=0;i<n;i++)); do ..; done`
- `case "$var" in pat) .. ;; pat) .. ;; esac`

Espressioni Condizionali `[[ ]]`:

	Operatore / Struttura	Sintassi	Significato
File	`-e`	`[ -e file ]`	file esiste
	`-f`	`[ -f file ]`	file regolare
	`-d`	`[ -d file ]`	directory
	`-h` / `-L`	`[ -h file ]`	link simbolico
	`-r`	`[ -r file ]`	leggibile
	`-w`	`[ -w file ]`	scrivibile
	`-x`	`[ -x file ]`	eseguibile
	`-nt`	`[ file1 -nt file2 ]`	file1 più recente
	`-ot`	`[ file1 -ot file2 ]`	file1 più vecchio
Stringhe	`-z`	`[ -z str ]`	lunghezza zero
	`-n`	`[ -n str ]`	lunghezza non zero
	`=` / `==`	`[ s1 = s2 ]` / `[[ s1 == s2 ]]`	uguali
	`!=`	`[ s1 != s2 ]`	diversi
	`<`	`[[ s1 < s2 ]]`	ordine lessicografico
	`>`	`[[ s1 > s2 ]]`	ordine lessicografico
Numeri	`-eq`	`[ n1 -eq n2 ]`	uguali
	`-ne`	`[ n1 -ne n2 ]`	diversi
	`-lt`	`[ n1 -lt n2 ]`	minore
	`-le`	`[ n1 -le n2 ]`	minore o uguale
	`-gt`	`[ n1 -gt n2 ]`	maggiore
	`-ge`	`[ n1 -ge n2 ]`	maggiore o uguale

- all'interno di espressioni matematiche è possibile usare gli operatori `<>=` per valutare espressioni matematiche, ma all'interno di `[[ ]]` questi confrontano stringhe

Manipolazione Stringhe:

Comando	Uso principale	Struttura (sintassi)	Flag importanti
`echo`	output	`echo STRINGA`	`-n` , `-e`
`printf`	output preciso	`printf FORMAT ARG`	`%s %d \n`
`read`	input	`read VAR...`	`-r` , `-a` , `-u`
`wc`	conteggio	`wc FILE`	`-c -w -l -m`
`tr`	trasformazione	`tr SET1 SET2`	`-d -s -cd`
`cut`	estrazione	`cut -d D -f N`	`-d -f -b -c`
`sort`	ordinamento	`sort FILE`	`-n -r -k -t -u`
`uniq`	unicità	`uniq FILE`	`-c -d -u`
`grep`	ricerca	`grep PATTERN FILE`	`-E -v -o -i -n`
`sed`	editing	`sed 'SCRIPT' FILE`	`s///g -n -i`
`awk`	processing	`awk 'PROG' FILE`	`-F` , `NR` , `$1`
`head`	prime righe	`head FILE`	`-n`
`tail`	ultime righe	`tail FILE`	`-n -f`
`rev`	inversione	`rev FILE`	—

FD, redirezioni:

Struttura	Sintassi	Effetto
stdin da file	`cmd < file`	input da file
stdout a file	`cmd > file`	sovrascrive
stdout append	`cmd >> file`	aggiunge
stderr a file	`cmd 2> file`	solo stderr
stdout+stderr	`cmd &> file`	output completo
fd custom read	`exec 3< file`	fd 3 in lettura
fd custom write	`exec 4> file`	fd 4 in scrittura
leggere da fd	`read -u 3 var`	legge da fd

Struttura	Sintassi	Effetto
scrivere su fd	`echo x >&4`	scrive su fd
chiudere fd	`exec 3<&-`	chiusura
here-document	`cmd <<EOF`	stdin multilinea
here-string	`cmd <<< "str"`	stringa → stdin
process subst	`cmd < <(cmd2)`	output come file
pipe	`cmd1 | cmd2`	stdout → stdin