

# Appunti Esame Pratico

## Parameter Expansion:

Sintassi	Uso	Esempio	Output
<code>\$var</code> / <code>\${var}</code>	Esponde la variabile	<code>x=3; echo \$x / \${x}</code>	3
<code>\${#var}</code>	Lunghezza stringa	<code>x=ciao; echo \${#x}</code>	4
<code>\${var:offset:length}</code>	Estrae <b>length</b> caratteri da <b>offset</b> (0-based)	<code>s=abc; echo \${s:1:1}</code>	b
<code>\${VAR#pattern}</code>	Taglia da <b>sinistra</b> , il meno possibile	<code>s="a-b-c"echo "\${s#*-}"</code>	b-c
<code>\${VAR##pattern}</code>	Taglia da <b>sinistra</b> , il più possibile	<code>s="a-b-c"echo "\${s##*-}"</code>	c
<code>\${VAR%pattern}</code>	Taglia da <b>destra</b> , il meno possibile	<code>s="a-b-c"echo "\${s%-*}"</code>	a-b
<code>\${VAR%%pattern}</code>	Taglia da <b>destra</b> , il più possibile	<code>s="a-b-c"echo "\${s%%%-*}"</code>	a

## Command Substitution:

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

## Arithmetic Expansion:

Sintassi	Uso	Esempio	Output
<code>\$((expr))</code>	Valuta espressione aritmetica e restituisce il risultato	<code>echo \$((3+5))</code>	8
<code>((expr))</code>	Valuta ma restituisce l'esito	<code>((x&gt;3))</code>	<i>status</i>

- 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
<code>{a,b}</code>	Espande lista	<code>echo {a,b}</code>	<code>a b</code>
<code>{1..5}</code>	Range numerico	<code>echo {1..3}</code>	<code>1 2 3</code>
<code>{a..c}</code>	Range alfabetico	<code>echo {a..c}</code>	<code>a b c</code>

### Globbering Expansion:

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

### History Expansion:

Sintassi	Restituisce
<code>!!</code>	ultimo comando
<code>!\$</code>	ultimo argomento
<code>!n</code>	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
<code>comando &amp;</code>	Avvia il comando in <b>background</b>	Il PID è in <code>\$!</code> , il job ha un indice <code>%n</code>
<code>Ctrl + Z</code>	<b>Sospende</b> il processo in foreground	Il processo non termina
<code>Ctrl + C</code>	<b>Termina</b> il processo in foreground	Segnale SIGINT
<code>jobs</code>	Elenca i <b>job</b> attivi o sospesi	Mostra <code>%n</code> , <b>non PID</b>
<code>bg %n</code>	Riprende un job sospeso in <b>background</b>	Usa l'indice job
<code>fg %n</code>	Porta un job in <b>foreground</b>	Riattacca stdin
<code>kill PID</code>	Invia un segnale a un processo	PID reale
<code>kill %n</code>	Invia un segnale a un job	<code>%n</code> ≠ PID
<code>disown %n</code>	Sgancia il job dalla shell	Sopravvive alla chiusura
<code>ps -ax</code>	Lista <b>tutti</b> i processi	Non solo quelli della shell

Comando	Effetto	Note importanti
<code>top</code>	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
<b>File</b>	<code>-e</code>	<code>[ -e file ]</code>	file esiste
	<code>-f</code>	<code>[ -f file ]</code>	file regolare
	<code>-d</code>	<code>[ -d file ]</code>	directory
	<code>-h / -L</code>	<code>[ -h file ]</code>	link simbolico
	<code>-r</code>	<code>[ -r file ]</code>	leggibile
	<code>-w</code>	<code>[ -w file ]</code>	scrivibile
	<code>-x</code>	<code>[ -x file ]</code>	eseguibile
	<code>-nt</code>	<code>[ file1 -nt file2 ]</code>	file1 più recente
	<code>-ot</code>	<code>[ file1 -ot file2 ]</code>	file1 più vecchio
	<code>-z</code>	<code>[ -z str ]</code>	lunghezza zero
<b>Stringhe</b>	<code>-n</code>	<code>[ -n str ]</code>	lunghezza non zero
	<code>= / ==</code>	<code>[ s1 = s2 ] / [[ s1 == s2 ]]</code>	uguali
	<code>!=</code>	<code>[ s1 != s2 ]</code>	diversi
	<code>&lt;</code>	<code>[[ s1 &lt; s2 ]]</code>	ordine lessicografico
	<code>&gt;</code>	<code>[[ s1 &gt; s2 ]]</code>	ordine lessicografico

	Operatore / Struttura	Sintassi	Significato
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	<code>cmd &lt; file</code>	input da file
stdout a file	<code>cmd &gt; file</code>	sovrascrive
stdout append	<code>cmd &gt;&gt; file</code>	aggiunge
stderr a file	<code>cmd 2&gt; file</code>	solo stderr
stdout+stderr	<code>cmd &amp;&gt; file</code>	output completo
fd custom read	<code>exec 3&lt; file</code>	fd 3 in lettura
fd custom write	<code>exec 4&gt; file</code>	fd 4 in scrittura
leggere da fd	<code>read -u 3 var</code>	legge da fd
scrivere su fd	<code>echo x &gt;&amp;4</code>	scrive su fd
chiudere fd	<code>exec 3&lt;&amp;-</code>	chiusura
here-document	<code>cmd &lt;&lt;EOF</code>	stdin multilinea
here-string	<code>cmd &lt;&lt;&lt; "str"</code>	stringa → stdin
process subst	<code>cmd &lt; &lt;(cmd2)</code>	output come file
pipe	<code>cmd1   cmd2</code>	stdout → stdin