Lezione 1 - 8/3/21 (Bash)

Comandi base Bash

- Ogni riga è un comando
- Gli spazi sono ignorati
- Posso mettere più comandi in fila (?)
- Se metto **"\"** dopo un comando premendo invio posso **scrivere altri comandi a capo**
- **Commenti** con "#"
- Il simbolo **'$'** serve a fare un get della variabile
- Scrivendo '**$$'** ci restituisce il **PID** del processo attuale
- Scrivendo **'$?'** ci restituisce il **codice di ritorno dell'ultimo comando** (non l'output ma lo stato di esecuzione: con successo o errore)Se
- Scrivendo di fila due comandi diversi ma separati da un solo ';' ci permette di eseguire quei due **comandi in sequenza**
- Se scrivo: **"comando1 && comando2"** allora verrà eseguito il secondo comando solo se il primo riesce ad essere eseguito correttamente
- Se scrivo: **"comando1 || comando2"** allora viene eseguito il secondo comando solo se il primo fallisce
- **Piping**: comunicazione tra processi e quindi comandi
- Se scrivo: **"comando1 | comando2"** allora vengono eseguiti entrambi i comandi con **piping** (cattura solo lo **stdout**)
- Se scrivo: **"comando1 |& comando2"** allora vengono eseguiti entrambi i comandi con **piping** (cattura **anche** lo **sterr**)
- In pratica con la singola pipe: **'|' l'output** del primo comando viene dato **in pasto al secondo** comando

- **Subshell:** sotto-ambiente della **shell** attuale, lo si utilizza usando "**(...comandi...)**" e si prende l'**output di una subshell** con '**$**'
- **0 = true**
- **1 = false**

Canali:

- Canale 0 → **stdin**
- Canale 1 → **stdout**
- Canale 2 → **stderr**

Alias:

"alias lsss=ls"

Argomenti:

`ls -alh /temp`

- Hanno ordine di precedenza

Comandi:

- **ls** → Elenca le sottocartelle e file della cartella corrente
- **ls** -v → "Verbous" da informazioni più complete (tre livelli: "-v", "-vv", "-vvv")
- **clear** → Serve a cancellare tutto ciò che c'è sul terminale (in realtà volta pagina)
- **pwd** → "Print working directory" Ci mostra il path completo della cartela corrente
- **cd** → Ci consente di spostarci tra le cartelle (".." per andare in dietro)
- **wc** → "What Count" Da dettagli sui file es: grandezza, quante righe, quanti caratteri...
- **date** → Restituisce la data attuale
- **cat** → Concatena dei file
- **echo** → Possiamo scrivere del testo come output

- **alias** → Permette di creare degli alias tra due comandi (anche con parametri!!)
- **test** →
- **read** → Consente l'acquisizione di variabili
- **file** → Da informazioni sul singolo file (Es: "file.txt" → ASCII text)
- **chown** → Consente di cambiare il proprietario del file e quindi i permessi dei vari utenti su un file
- **chmod** → Consente di modificare i permessi che si hanno su quel file come lettura, scrittura o entrambi
- **cp** → Permette di copiare un file (o una cartella?)
- **mv** → Permette di spostare file in un altra posizione (o una cartella?)
- **type** → Restituisce informazioni su un singolo comando cioè che tipo di comando è
- **grep** → Permette di fare la ricerca all'interno di file
- **truncate** → Permette di limitare o espandere il contenuto di alcuni file come la grandezza
- **function** → Permette di definire funzioni

Canali

Per redirezionare un comando su un altro canale posso usare la sintassi:

`" ls 1>/tmp/out.txt 2>/tmp/err.txt"`

Come si vede sopra i numeri stanno ad indicare il canale di output e si può fare un redirezionamento anche in cascata come mostrato.

N.B. → Se ometto il canale in automatico sarà quello di output (1).

Con il "<" invece cambio il canale di input es:

`" mail -s "Subject" rcpt < content.txt "`

(anziché interattivo)

">>" operator

Funziona come un append, cioè se scrivo:

"echo ciao mondo>file.txt"

sovrascrive su file.txt il testo che lo segue.

Invece con ">>":

"echo ciao mondo>>file.txt"

...aggiunge il testo che scriviamo a file.txt

"<<" operator

Ad es:

"echo << EOF"

Così viene dato un prompt interattivo che accetta ed esegue gli input dati fino al carattere 'EOF'.

Variabili

Per dichiarare una variabile basta mettere: 'nome'='valore' come di seguito:

"var=val"

Inoltre le variabili valgono solo nella stessa istanza di cmd in cui sono state dichiarate

Variabili di sistema

- **shell** → Contiene il riferimento all'applicazione corrente (alla shell corrente)
- **path** → Contiene i file eseguibili dei vari comandi in ordine di priorità
- **term** → Contiene il tipo di terminale corrente
- **pwd** → Contiene la cartella corrente (modificata quando uso "cd")
- **ps1** → Contiene il prompt e si possono usare marcatori speciali
- **home** → Contiene la cartella principale dell'utente corrente

Array

Definizione: lista=("a" 1 "b" 2 "c" 3)

Output: ${lista[@]}

Accesso singolo: ${lista[x]} (0-based)

Lista indici: ${!lista[@]}

Dimensione: ${#lista[@]}

Set elemento: lista[x]=value

Append: lista+=(value)

Looping:

```
for i in ${!lista[@]}; do
  echo ${lista[$i]}
done
```

```
while [ $i -le 2 ]
do
  echo Number: $i
  ((i++))
done
```

Sottolista: ${lista[@]:s:n} (from index s, length n) per stampare scrivo prima "echo"

Espansione aritmetica

- L'espansione aritmetica non è da confondere con la subshell e si differenzia per l'utilizzo di "(())" al posto della singola
- All'interno delle doppie parentesi tonde si possono rappresentare varie espressioni matematiche inclusi assegnamenti e confronti
- Serve per eseguire varie espressioni matematiche senza che vengano fraintese

Es:

((a = 7)) ((a++)) ((a < 10)) ((a = 3<10?1:0))

Confronti logici

I costrutti fondamentali per i confronti logici sono "test" e i raggruppamenti tra parentesi quadre singole o doppie:

`test ...` , `[ ... ]` , `[ [ ... ] ]`

- `test ...` e `[ ... ]` sono built-in equivalenti
- `[ [ ... ] ]` è una coppia di shell-keywords

In tutti i casi il blocco di confronto genera il codice di uscita 0 in caso di successo, un valore differente (tipicamente 1) altrimenti.

Esempio:

Es1

```
echo "/tmp" > /tmp/tmp.txt ; ls $(cat /tmp/tmp.txt)
```

come eseguire:

```
ls /tmp
```

Confronti logici - interi e stringhe

Interi

Aa Operator	[...]	((..))
Uguale-a	-eq	==
Diverso-da	-ne	!=
Minore-di / minore-o-uguale-a	-lt / -le	< / <=

Stringhe

Aa Operator	[...]	((...))
Uguale-a	= o ==	= o ==
Diverso-da	!=	!=
Minore-di (ordine alfabetico)	\<	<
Maggiore-di (ordine alfabetico)	\>	\>

N.B.

Occorre lasciare uno spazio prima e dopo i "simboli" (es. non "=" ma " = ")

Confronti logici - operatori unari

Esistono alcuni operatori unari ad esempio per verificare se una stringa è vuota o meno oppure per controllare l'esistenza di un file o di una cartella.

Alcuni esempi:

[[-f /tmp/prova]] o [[! -f /tmp/prova]] → Ci dice se "prova" è un file

[[-e /tmp/prova]] o [[! -e /tmp/prova]] → Ci dice se esiste

[[-d /tmp/prova]] o [[! -d /tmp/prova]] → Ci dice se è una cartella

Script/Bash

E' possibile raccogliere sequenze di comandi in un file di testo che può poi essere eseguito.

- Hanno estensione .sh
- Per aggiungere la possibilità di essere eseguito devo fare: "chmod +x ./directory/filename.sh"
- Si avviano scrivendo il percorso e il nome del file es: "./script.sh" e verranno cercati nella cartella specificata ma anche nelle cartelle contenute nella variabile bash
- I commenti si iniziano con '#'
- Il primo commento (prima riga) soprannominato shebang e si scrive: "#!/bin/bash" la shell lo tratterà come un file che va eseguito con questo percorso/applicazione
- Per salvare dentro una variabile del codice un valore passato per parametro userò la sintassi: "VAR=$0" "VAR1=$1" ...
- Per eseguire uno script .sh viene usata una sotto shell