Lezione 1 - Laboratorio SO

Tipologie di VM dell'Atene

Come utilizzare le VM dell'Ateneo (oggi usiamo gitbash):

Ci sono due tipi di VM:

1. La prima ha l'immagine sui PC fisici dell'istituto;
2. La seconda ha l'immagine su un server remoto.

SO -> Hypervisor (VMWare) -> SO "Host"
La macchina virtuale possiede un disco virtuale con ciò che contiene l'OS emulato. Ogni SO virtuale ha una descrizione di come è formato (virtualmente) e ha uno o più file che ne compongono il disco virtuale.

Nel caso 1) quando dei file nella VM vengono modificati essi vengono anche salvati nella macchina fisica; ciò significa che la macchina manterrà le modifiche - cambiare macchina fisica su cui viene avviata la VM non manterrà le modifiche, in quanto le modifiche non sono salvate su tale macchina.

Le VM hanno un solo user "Studente" con password "studente". Ciò significa che qualsiasi studente dell'ateneo può vedere i vari file che sono all'interno della VM. Pertanto NON bisogna salvare le password o modificare o eliminare directory di sistema.

Le VM hostate su sistemi remoti sono personali e privati; inoltre cambiare dispositivo non eliminerà i file che sono stati salvati al suo interno.

Oggi usiamo Git Bash al fine di vedere come funziona la bash:

Uso e assegnazione di variabili

Ogni qual volta runniamo un programma all'interno della bash esso lavora in un ambiente di esecuzione a sè stante. Il programma che viene lanciato eredita una copia della bash da cui è stato avviato; tale copia non è però completa: non vengono copiate tutte le variabili, ma solo quelle "d'ambiente".

Come si assegnano le variabili in bash:

`<nvar>=<args>` - `nvar` ottiene il valore `args`. l'uguale deve stare ATTACCATO a `nvar`, altrimenti viene restituito `"command not found"`. Se invece `args` è preceduto da uno spazio, la bash proverà ad eseguire args come comando;

viene prima eseguita l'assegnazione (`nvar` diventa " ") e poi eseguito il comando `args` .

La variabile PATH

Una variabile molto importante è la variabile `PATH` : essa è formata da tanti percorsi di file system, separati da due punti (:).

Ad esempio: `/bin:/sbin:/usr/bin:/usr/local/sbin` si riferisce alle directory `/bin` , `/sbin` , `/usr/bin` , e `/usr/local/sbin` .
Per visualizzare la variabile, bisogna runnare :

```
echo $PATH
```

Quando viene lanciato un programma senza specificare un percorso specifico, la bash cerca tale programma in alcune directory; tali directory sono appunto specificati all'interno della variabile PATH. Nel caso ci fossero due file chiamati uguale in due directory specificate in PATH, viene eseguito quello che si trova nella directory che sta per prima nella PATH (nel caso precedente) se abbiamo gcc in /bin e in /sbin verrà eseguito prima quello in /bin).
Se un programma viene eseguito senza specificare nella directory e non si trova all'interno della PATH, non verrà eseguito in tale modo: esso va quindi eseguito tramite specifica della directory in cui si trova.

Per sapere dove si trova un programma nel file system, esiste il comando

```
which <cmd>
```

dove cmd è il programma che si sta cercando. which cerca gli SOLO eseguibili presenti all'interno della PATH.

Modificare la PATH comporta che i programmi di uso comune possono non essere eseguibili direttamente come comando: ciò rende il tutto molto più difficile da usare, quindi va modificata con prudenza.

Usare il comando bash all'interno di una bash creerà un'altra bash figlia della bash su cui il comando è stato eseguito. La shell figlia possiede una copia (non completa) della madre. Modificare qualcosa all'interno di una bash figlia non comporterà modifiche all'interno della bash padre, che può essere ripristinata tramite il comando exit, che chiude la bash in utilizzo.

Il comando ps mostra i processi di sistema. La colonna PID riporta l'identificatore del processo (Process IDentifier), mentre la colonna COMMAND ci dice il programma che sfrutta il processo.
PPID è l'ID del processo padre da cui il processo è stato generato. La bash principale ha PPID 1.

Cosa può eseguire una CLI?

La CLI può eseguire 3 tipologie di comandi:

1 - Comandi built-in: sono i comandi il cui interprete è la shell stessa
2 - File binari eseguibili: sono i file che contengono codice macchina e si trovano all'interno del file system.
3 - Script: file di testo che contengono una lista di comandi (dei tipi appena visti) e vengono creati dagli utenti.

Creazione di script

La prima riga di uno script è speciale, in quanto serve all'esecutore per capire qual è l'interprete di comandi che deve eseguire ciò che scriviamo. Essa è

```
#!/bin/bash
```

che ci dice:
- `#` implica un commento
- `!` implica, se messo nel commento in prima riga, che dobbiamo usare l'interprete che specifichiamo
- `/bin/bash` è l'interprete che usiamo, in questo caso la bash.

Se non specificato, lo script viene interpretato da un processo figlio di quello che lo ha avviato.

Per eseguire un programma/script presente all'interno della directory corrente bisogna usare `./<name>`, dove `./` rappresenta la directory attuale mentre `name` è il programma da eseguire.

Permessi, utenti e gruppi

Non è detto che un programma può essere eseguito: ciò dipende dall'utente che lo esegue e dal suo gruppo di appartenenza.

- Un utente è un entità, umana o digitale, che rappresenta l'esecutore di un dato programma in un dato momento. Ogni utente ha una stringa univoca detta "username" e un

identificativo, l'userID.
- Un gruppo è caratterizzato da una stringa, detta groupname, e un identificativo, il groupID. Un utente può appartenere a uno o più gruppi: uno di questi gruppi di appartenenza è il suo gruppo primario, mentre gli altri sono gruppi secondari

Quando un utente crea un file o una directory, a tale file viene associato il gruppo primario a cui l'utente appartiene e l'utente proprietario di quel file. Entrambe le caratteristiche possono essere modificate con `chgrp` e `chown` rispettivamente. L'utente proprietario può cambiare i permessi relativi al file che ha creato, relativi a modifica, lettura, scrittura o, in caso di directory, accesso. Un utente qualunque che vuole accedere ad un file si chiama "effective user".

I permessi relativi ad un file si riferiscono al proprietario (user), agli appartenenti di uno specifico gruppo (group) e agli utenti non appartenenti alle due classi (others).
Lo script che abbiamo creato prima, se eseguiamo

```
ls -al primo.sh
```

riceveremo come output:

```
-rwxr-xr-x 1 STUDENTI+lorenzo.mazzini2 4266656257 27 Sep 23 10:52 primo.sh*
```

La prima stringa rappresenta i permessi e il tipo del file/directory a cui ci stiamo riferendo.

- Il primo carattere rappresenta il tipo di file:
  - `d` rappresenta le directory
  - `c` rappresenta i dispositivi a carattere, ossia le linee di connessione (es. le TTY).
  - `b` rappresenta i dispositivi che possono essere letti a blocchi
  - `-` sono i file eseguibili
- I successivi 9 caratteri sono da osservare a blocchi di 3:
  - I primi 3 sono i permessi del proprietario
  - I secondi 3 sono i permessi del gruppo di appartenenza del file
  - Gli ultimi 3 sono i permessi che degli others.
  In particolare i permessi si suddividono in 4 caratteri:
  - `r` rappresenta i permessi di lettura - ha valore 4. Per i file, ne rende possibile la lettura e l'eventuale modifica (tramite comandi come `cat`), per le directory ne permette il listing (tramite `ls`)
  - `w` rappresenta i permessi di scrittura - ha valore 2. Per i file, rende possibile la loro modifica, per le directory rende possibile la creazione di file / subdirectory al suo interno.
  - `x` rappresenta i permessi di esecuzione - ha valore 1. Per i file, rende possibile l'esecuzione, per le directory rende possibile l'accesso.

- `-` rappresenta che tale permesso è mancante e l'azione relativa a tale permesso non può essere eseguito
  La somma dei valori corrisponde all'insieme di permessi che un utente/gruppo/other ha sul file. Per esempio, se il proprietario del file ha valore 6 nel file, esso avrà i permessi di scrittura e lettura (`r` e `w`).
  Il comando

```
chmod <user|group|other> <nomefile>
```

permette di cambiare i permessi di un file relativo al proprietario, al gruppo di appartenenza, e agli altri utenti, specificando i valori numerici per ciascuno.
Ad esempio

```
chmod 640 primo.sh
```

Cambierà i permessi nel seguente modo:

- Il proprietario avrà i permessi di lettura e scrittura, ma non di esecuzione
- Il gruppo di appartenenza del file avrà solo i permessi di lettura
- Gli altri utenti non possono interagire con il file / accedere alla directory.