

# AMMINISTRAZIONE

DI

# SISTEMI

A.Q.

# Indice

<b>1 SysAdm e Cloud computing</b>	...3
<b>2 Funzionamento GNU/LINUX</b>	...5
<b>2.1 Funzionamento dei moduli</b>	
<b>2.2 Sistemi di storage</b>	
<b>2.3 Processo di avvio</b>	
<b>2.4 Partizionamento da terminale</b>	
<b>3 Pacchetti software</b>	...11
<b>3.1 Installazione</b>	
<b>3.2 Pacchetti</b>	
<b>3.3 Your own repo</b>	
<b>4 Shell</b>	...15
<b>4.1 Processi</b>	
<b>4.2 Processi e segnali</b>	
<b>4.3 Processi in background</b>	
<b>4.4 Shell expansion</b>	
<b>4.5 Filtri</b>	
<b>4.6 Shell scripting: variabili</b>	
<b>4.7 Shell scripting: aritmetica</b>	
<b>4.8 Shell scripting: controllo di flusso</b>	
<b>4.9 Shell scripting</b>	
<b>5 Utenti, Gruppi e File</b>	...39
<b>5.1 Gruppi</b>	
<b>5.2 Permessi</b>	
<b>5.3 Lavorare con i file</b>	
<b>6 Servizi e Logging</b>	...43
<b>6.1 Cron: esecuzione periodica</b>	
<b>6.2 At: posticipare l'esecuzione</b>	
<b>6.3 systemd</b>	
<b>6.4 Monitoraggio e logging</b>	

<b>6.5 Gestione servizi</b>	
<b>7 Vagrant, ssh e Ansible</b>	...49
<b>7.1 Secure Shell</b>	
<b>7.2 Ansible</b>	
<b>7.3 Playbook</b>	
<b>7.4 Roles</b>	
<b>7.5 Integrazione con vagrant</b>	
<b>7.6 Esempi ed esercizi</b>	
<b>7.7 Handlers</b>	
<b>8 Networking</b>	...62
<b>8.1 Configurazione di rete</b>	
<b>8.2 Configurazione di una rete di VM</b>	
<b>8.3 Configurazione della rete tramite Ansible</b>	
<b>8.4 Esercizio riassuntivo</b>	
<b>8.5 DHCP</b>	
<b>9 Monitoraggio centralizzato</b>	...74
<b>9.1 Modello dei dati e operazioni</b>	
<b>9.2 Configurazione SNMP</b>	
<b>9.3 Comandi manager</b>	
<b>9.4 Misurare parametri del sistema</b>	
<b>9.5 Esecuzione codice</b>	
<b>9.6 Esercizio riassuntivo</b>	
<b>10 Configurazione centralizzata</b>	...94
<b>10.1 LDAP</b>	
<b>10.2 Entry e schema</b>	
<b>10.3 Tipi di attributi</b>	
<b>10.4 Classi</b>	
<b>10.5 Gestione schemi e entry</b>	
<b>10.6 LDAP per l'autenticazione LINUX</b>	
<b>10.7 Configurazione per login via LDAP</b>	
<b>10.8 Test del login</b>	
<b>10.9 sudo-LDAP</b>	

# 1 SysAdm e Cloud computing

Tra le necessità più richieste da parte di un'azienda che usa sistemi di elaborazione troviamo:

- Alta disponibilità
- Sicurezza
- Aggiornamento hardware

La collocazione di un ambiente locale (*on premises*) è sempre più rara.

Oggi giorno si tende ad “**esternalizzare**”, ovvero, a commissionare a professionisti la gestione di un ambiente affidabile. Questi luoghi sono i **Data center** o **Server farm**.

Per garantire l'accessibilità senza interruzioni di dati e servizi sono state sviluppate alcune tecniche di base che partono dagli elementi costruttivi e salgono alle architetture.

(RAID, LVM, multipath, clustering, ...)

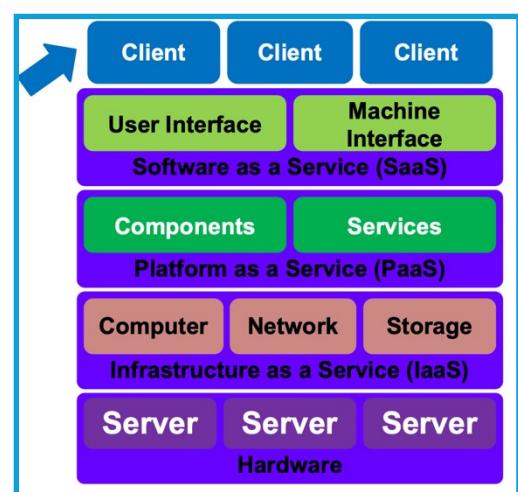
[Queste tecniche non difendono da eventi catastrofici come i cataclismi o i terremoti]

Una semplice soluzione di housing risolve molti problemi ma lascia comunque nelle mani del cliente aspetti sistemistici. Ad esempio, se il mio business è sviluppare e testare un applicativo desktop, l'hardware non mi interessa, mi basta poter configurare liberamente i diversi sistemi operativi su cui distribuirlo.

Il **cloud computing** permette di affrontare con maggior efficienza molte tipologie di progetti. Dal punto di vista economico, permette investimenti più flessibili; dal punto di vista gestionale, esternalizza le attività non-core.

Un cloud provider si fa carico della realizzazione di un (gruppo di) data center: realizza gli edifici, predisponde gli impianti, acquista apparati di calcolo e networking. Il pool di risorse complessive viene utilizzato per far funzionare sistemi virtualizzati. Molti clienti condividono le risorse fisiche (multi-tenancy) spalmando i costi fissi e delegando completamente la loro amministrazione; la configurazione è tramite interfacce che nascondono completamente la struttura fisica.

C'è *provisioning dinamico*: l'avvio e arresto delle risorse è on demand.



# 2 Funzionamento GNU/Linux

Il sistema operativo svolge una varietà di ruoli, sinteticamente...

- Astrazione delle risorse
  - Fisiche: Storage, porte usb, schede di rete, ...
  - Logiche: Filesystem, stack di rete, ...
- Controllo dell'accesso
  - Scheduling CPU
  - Allocazione della memoria
  - Accesso a dispositivi
  - ...

L'accesso all'hardware in particolare è astratto da **moduli** che implementano i **device driver**.



Un **device driver**, o driver, è un programma software che consente al sistema operativo di comunicare con un dispositivo hardware specifico. Questo significa che il driver funge da interprete tra il software e l'hardware, facilitando l'interazione tra di essi. Alcuni driver sono cablati nel kernel, la maggior parte sono implementati da **moduli del kernel** dinamicamente caricabili.

Un **modulo** definisce come farsi trovare e come sono implementate le versioni specifiche di system call per il dispositivo gestito dal modulo stesso.

Il SO definisce tecniche standard di interazione:

- 1) Rilevato dispositivo fisico
- 2) Controller I/O manda un INTERRUPT
- 3) Interrupt handler: dialogo con SO; evento scritto su **dbus** (canale per eventi di sistema)
- 4) **udev** riceve l'evento, e consultando **/lib/modules/<kernel\_version>/modules.alias** individua il modulo in grado di gestire il dispositivo

## 2.1 Funzionamento dei moduli

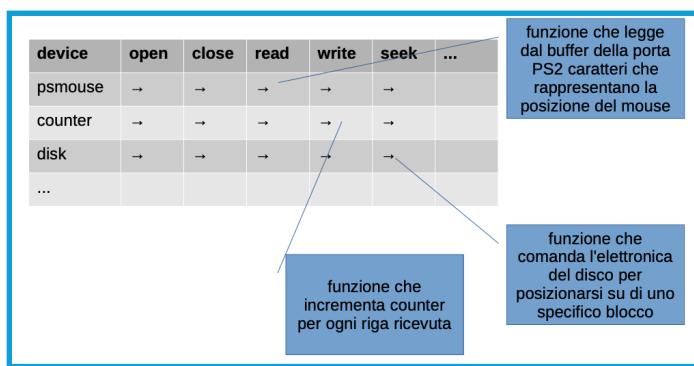
In ogni modulo sono dichiarate le stringhe identificative dei dispositivi fisici gestibili sotto forma di alias. Il comando **depmod** crea un elenco di tutte queste dipendenze e le raccoglie nel file *module.alias*.

Il modulo definisce in che modo vanno implementate le system call per **macrocategorie**:

**Dispositivi a blocchi:** utilizzano buffer e cache per ottimizzare il trasferimento di blocchi di dimensione data (Es. *Dischi di vario tipo*).

**Dispositivi a caratteri:** gestiscono il trasferimento dati un carattere/byte alla volta.

Il codice delle funzioni dei moduli sfrutta una tabella di puntatori specificando per ogni device dove trovare una specifica funzione.  
Serve dunque un sistema di nomi per dire "voglio invocare la read di X".



In Linux, i "**device file**" sono file speciali che consentono agli utenti e ai processi di interagire direttamente con i dispositivi hardware o i driver del kernel come se fossero file standard. Questi file speciali si trovano comunemente nella directory `/dev` del sistema file.

La creazione e la gestione dei device file sono gestite da **udev**, che è il sistema di gestione dei dispositivi per il kernel Linux. Quando un dispositivo hardware viene rilevato dal sistema, udev crea automaticamente il device file corrispondente nella directory `/dev`, consentendo agli utenti e ai processi di accedere al dispositivo.

```
b rw-rw---- 1 root disk 8, 1 Mar 20 11:14 /dev/sda1  
c rw--w---- 1 las tty 136, 0 Mar 20 11:17 /dev/pts/0
```

→ sono memorizzati sul filesystem come normali inode, ma non hanno data block associati

→ su di essi, invocando le system call tipiche dei file si scatenano operazioni definite nel corrispondente device driver

→ **major number**: identifica il tipo di dispositivo, ad esempio se è un dispositivo di blocco

→ **minor number**: identifica un'istanza specifica del dispositivo,  
ad esempio, due dischi rigidi avranno stesso major number e diverso minor number

→ possono essere di tipo **blocco** o **carattere**

<p>Alcuni device files non rappresentano vere periferiche, sono implementati dal kernel e sono utili per lavorare coi processi</p> <ul style="list-style-type: none"> <li><b>/dev/zero</b> produce uno stream infinito di zeri (binari)</li> <li><b>/dev/null</b> ogni read restituisce EOF, ogni write viene scartata</li> <li><b>/dev/random</b> produce byte casuali ad alta entropia → <u>bloccante</u> se non ce n'è a sufficienza</li> <li><b>/dev/urandom</b> produce uno stream pseudocasuale illimitato</li> </ul>	<p>Alcuni device files notevoli che rappresentano vere periferiche:</p> <ul style="list-style-type: none"> <li><b>/dev/tty*</b> terminali fisici del sistema</li> <li><b>/dev/pts/*</b> pseudo-terminali (dentro finestre del sistema grafico)</li> <li><b>/dev/sd*</b> <u>dischi e partizioni</u></li> </ul>
---	---

## 2.2 Sistemi di storage

tipicamente: **dispositivi a blocchi = supporti di storage**

In Linux, un "dispositivo a blocchi" è un tipo di dispositivo hardware o di file system che trasferisce dati in blocchi di dimensioni fisse. Questi dispositivi sono spesso utilizzati per l'archiviazione di dati a lungo termine e includono hard disk, SSD, unità USB e altre forme di memoria di massa.

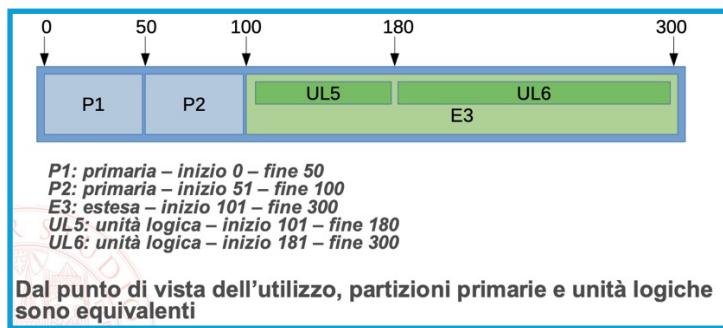
Per renderlo fruibile servono tre operazioni...

### Partizionamento

Consiste nella suddivisione di un disco in sottoinsiemi di blocchi. Ogni separazione si comporta come un dispositivo indipendente! Torna utile per separare spazi con esigenze diverse.

Per un partizionamento esistono vari standard, un esempio è il **Master Boot Record**.

Tabella con max 4 partizioni primarie dove ognuna potrebbe indicare una partizione estesa.



Device file per dischi e partizioni sono unificati sotto il framework **SCSI**:

`/dev/sdXXNN` (ES: `/dev/sda1`)

→ **XX** una o più lettere che identificano il disco

→ **NN** numero della partizione

Notiamo i nuovi dischi NVMe/M.2 con notazione differente:

`/dev/nvmeXnYpN`

Con **Y** identificatore del namespace, una sorta di macro-partizione hardware.

I criteri di partizionamento possono essere vari come partizione di swap (memoria virtuale), collezione dati generale, collezione dati divisi per funzioni.

**Osservazione:** La partizione di swap è una particolare partizione di un disco rigido o di un altro dispositivo di archiviazione che viene utilizzata dal sistema operativo come area di memoria virtuale aggiuntiva. Questo spazio di memoria viene utilizzato per memorizzare temporaneamente i dati quando la RAM (memoria principale del sistema) è piena o quando il sistema necessita di spazio aggiuntivo per l'allocazione di processi.

## Formattazione

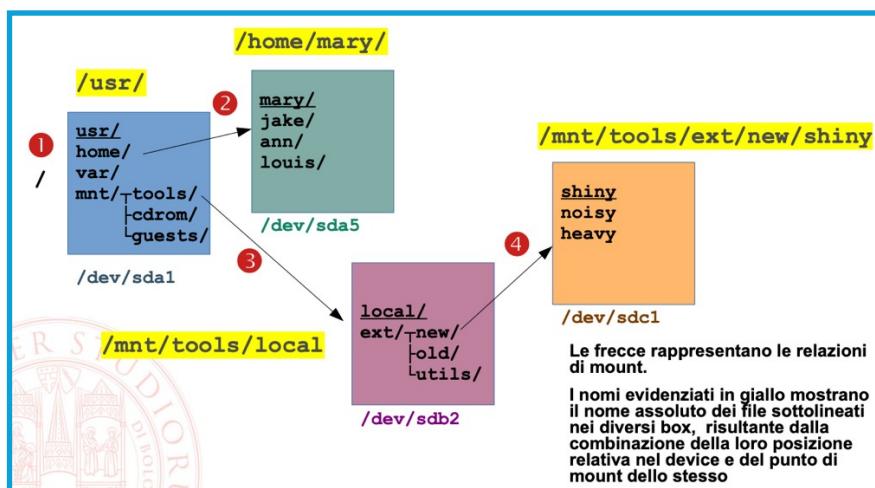
Per utilizzare un dispositivo a blocchi per archiviare dati, è necessario formattarlo con un filesystem. Questo permette l'accesso ai file secondo il modello **Virtual File System**.

Esempio di filesystem è **ext4**:

- limite singolo file 16 TiB
- limite filesystem 1 RiB
- extent: i file grandi sono allocati in modo contiguo
- allocazione dei blocchi a gruppi
- allocatore invocato solo quando necessario
- journal checksumming

## Mount

Prima di poter utilizzare un dispositivo a blocchi, è necessario montarlo nel filesystem. Questo consente al kernel di associare il dispositivo a blocchi con un percorso nel filesystem, consentendo agli utenti e ai programmi di accedervi come se fosse un'altra directory nel sistema.



## 2.3 Processo di avvio

Per andare a regime il sistema attraversa un processo di boot, che può essere diviso in queste fasi:

(1) **BIOS** – Individua i dispositivi di possibile caricamento del boot loader e l'ordine per esaminarli → Molti BIOS prevedono la possibilità di proteggere con password l'avvio o la modifica della configurazione

(2) **Boot Loader** – Sceglie il sistema operativo e gli passa eventuali parametri  
→ Gestione della “maintenance mode”  
→ Stesso tipo di protezione con password come descritto per BIOS

(3) **Sistema operativo** – carica i device driver (da non sottostimare) e avvia il processo init

(4) **init** – gestisce i runlevel o i target per coordinare l'inizializzazione del sistema, cioè avviare i servizi nell'ordine corretto

L'interfaccia UEFI verifica ogni componente software prima di passare il controllo al BootLoader e successivamente al sistema operativo. È possibile istruire il boot loader a caricare sistemi diversi (o più versioni/configurazioni dello stesso sistema).

## 2.4 Partizionamento da terminale

Lavorando in ambiente Linux possiamo usare i seguenti comandi per partizionare un disco.

**df**

Il comando **df** è utilizzato per visualizzare lo spazio su disco utilizzato.

Altro comando di simile utilizzo è **lsblk**.

**fdisk /dev/sdX**

Questo comando serve a manipolare le tabelle di partizioni su disco. Ad esempio, per partizionare il disco sdX posso usare il comando sopra riportato. Una volta dentro a fdisk...

**p** > permette di vedere le partizioni

**n** > creare una nuova partizione

**w** > salva ed esci

**mkfs.ext4 /dev/sdX1**

Per formattare come filesystem secondo ext4

**mount /dev/sdX1 /nuovo/**

Il comando viene utilizzato per montare il filesystem su una specifica directory. Nel nostro esempio montiamo /dev/sdX1 sulla dir /nuovo/

Per smontare il filesystem **umount /dev/sdX1**

# 3 Pacchetti software

Il tipico ciclo di vita del software prevede installazione, aggiornamento e disinstallazione. L'installazione può avvenire da **file binari** (tramite una semplice copia “nei posti giusti”) o da **file sorgente** (con necessità di compilazione).

Nel caso di un'installazione da binari, probabile necessità di disporre non solo dei software indicati come prerequisiti, ma anche che essi siano di una versione specifica.

Nel caso di installazione da sorgente, si ha invece qualche grado di flessibilità (possibilità che i sorgenti dispongano di diverse interfacce per adeguarsi a cosa si trova sul sistema).

In Linux il caso più comune è quello di software distribuito per mezzo di un archivio **tar.gz**, scritto in C e predisposto alla compilazione tramite auto-configurazione. Verifica se sono soddisfatti i prerequisiti, rivela eventuali versioni di pacchetti e la loro collocazione, genera i “Makefile” (file di configurazione) sulla base delle specificità del sistema.

La prima cautela da usare quando si scarica software dovrebbe essere quella di verificarne l'autenticità da una firma digitale. Naturalmente per verificare una firma serve una chiave pubblica fidata. Il comando **gpg** è utilizzato per la crittografia e la firma digitale dei dati.

Ad esempio...

**gpg --verify FILE.asc FILE.tar.gz**  
mostra il key id  
**gpg --keyserver pgpkeys.mit.edu --recv-key <KEY\_ID>**  
scarica una chiave pubblica da un server di chiavi specificato utilizzando l'id della chiave

## 3.1 Installazione

Installazione manuale tipica in Linux prevede l'estrazione di un pacchetto che solitamente si presenta come archivio tar compresso. È buona prassi determinare una collocazione sensata per i sorgenti ed estrarre in tale directory l'archivio.

Testare l'archivio prima dell'estrazione per verificare la gerarchia delle dir che genera...  
**cd /usr/local/src**  
**tar tvzf net-snmp-5.4.tar.gz**  
**tar xvzf net-snmp-5.4.tar.gz**

Di solito il software è accompagnato dai file da leggere README e INSTALL.

Se esiste un eseguibile di nome **configure** lo si lancia con il parametro **--help** per ottenere la lista dei parametri di configurazione disponibili. Successivamente si lancia nuovamente **configure** con i parametri scelti, si risolvono i problemi evidenziati da **configure** (tipicamente assenza di pacchetti necessari come prerequisiti), e si genera il file **config.log**.

Per la compilazione si lancia il comando **make** o si seguono le indicazioni presenti nell'output generato. → **sudo make install**

Ci è subito chiaro che questo tipo di installazione è difficile da manutenere e richiede molti componenti ausiliari. Entrano in scena le distribuzioni e i pacchetti e soprattutto **l'installazione assistita** effettuata per mezzo di software ausiliari.

Un tool di installazione può farsi carico delle verifiche relative alle dipendenze e può generare dinamicamente dati specifici.

## 3.2 Pacchetti

Le distribuzioni di Linux organizzano il **software in pacchetti** e dispongono di un **package manager** per la loro gestione. Un pacchetto si presenta sotto forma di singolo file che contiene in forma compatta l'insieme di software precompilato, criteri per la verifica della compatibilità e dei prerequisiti e procedure di pre/post-installazione.

Tutte le distribuzioni supportano i processori Intel 32bit, la maggior parte quelli a 64bit, alcune sono disponibili per tutte le varietà di processori su cui è stato portato il kernel.

Alcune distribuzioni sono “*versionate*”: durante il ciclo di vita di una versione vengono forniti solo aggiornamenti correttivi, tutte le novità vengono testate e accumulate per la pubblicazione in una nuova versione. Altre sono “*rolling*”: ogni volta che c’è una novità viene testata e distribuita; quindi, in ogni momento il sistema è alla versione più recente.

Per installazioni di tipo server esistono varianti denominate LTS (Long Term Support): per 5/7 anni chi cura la distro garantisce che gli aggiornamenti non modifichino le API.

Due distribuzioni capostipite da cui sono state derivate quasi tutte le varianti più diffuse sono **Debian** e **RedHat**, due sistemi di gestione dei pacchetti con molte somiglianze.

The diagram shows the structure of a .deb package file name: `aptitude-0.2.15.9-2_i386.deb`. It is divided into four main parts: `nome` (name), `versione del software` (software version), `versione del pacchetto` (package version), and `architettura` (architecture). Arrows point from each part to its corresponding segment in the file name.

I pacchetti per le distribuzioni Debian e derivate (es. Ubuntu) sono in formato .deb  
– `aptitude-0.2.15.9-2_i386.deb`

I pacchetti per le distribuzioni RedHat e derivate (es. CentOS, Fedora) sono in formato .rpm  
– `httpd-2.4.6-45.el7.centos.x86_64.rpm`

I pacchetti possono essere scaricati e gestiti singolarmente. Normalmente però si usano i repository (**repo**), raccolte indicizzate di pacchetti, possono essere online o su filesystem locali. I package manager leggono per ogni repo l'indice e i metadati dei pacchetti, conoscono quali versioni sono disponibili per ogni pacchetto, conoscono le dipendenze tra pacchetti (e quindi come risolverle).

#### Gestione dei pacchetti .deb

database location: `/var/lib/dpkg, /var/lib/apt`

sources file: `/etc/apt/sources.list`

update sources: `apt-get update`

key management: `apt-key`

search: `apt-cache search keywords`

install: `dpkg -i filename.deb`

apt-get install packagenames

upgrade: `apt-get upgrade [packagenames]`

remove: `dpkg -r packagename`

apt-get remove packagenames

Deprecato  
Chiavi gpg direttamente  
in `/etc/apt/trusted.gpg.d/`

(i suffissi `-get` e `-cache` possono essere omessi nelle distribuzioni più recenti, in cui il comando `apt` regge tutti i sotto-comandi come `search`, `update`, `install`, ...)

21

[Su slide anche gestione .rpm]

La firma dei pacchetti è gestita centralmente. I mantainer di una distribuzione forniscono le chiavi di verifica nei media di installazione ufficiali o sui repository online.

Si può generare confusione se un pacchetto con lo stesso nome è presente in versioni diverse in repository differenti. I package manager, di default, scelgono sempre la versione più avanzata

#### La situazione va controllata e gestita

##### – Controllo della provenienza di un pacchetto

- Yum: `repoquery -i [package name]`

- Apt: `apt-cache showpkg [package name]`

##### – Elenco dei pacchetti provenienti da un repo

- Yum: `yum list installed | grep [repo name]`

- Apt: vari comandi per estrarre manualmente info dai file della cache

Per evitare a priori problemi in sistemi con dipendenze complesse (ad esempio mix di pacchetti installati manualmente e via package manager). Il "version locking" o "version pinning" in Linux si riferisce alla pratica di fissare specifiche versioni di pacchetti software per garantire che un sistema rimanga stabile e coerente, evitando aggiornamenti non desiderati che potrebbero causare problemi di compatibilità o interrompere il funzionamento del sistema.

#### Apt

- editare `/etc/apt/preferences.d/*`

- <https://wiki.debian.org/AptPreferences>

#### Yum

- `yum install yum-plugin-versionlock`  
poi

- `yum versionlock [package name]`  
o editare a mano

- `/etc/yum/pluginconf.d/versionlock.list`

## 3.3 Your own repo

Non è difficile **pacchettizzare** le proprie applicazioni! Nel mondo RPM:

- si configura un ambiente di build per un utente (non root!)
- si preparano i sorgenti e tutti i file che devono essere inclusi in un pacchetto
- si effettua il build del pacchetto
- lo si carica su di un server web
- si generano gli indici che rendono riconoscibile il sito come repository

# 4 Shell

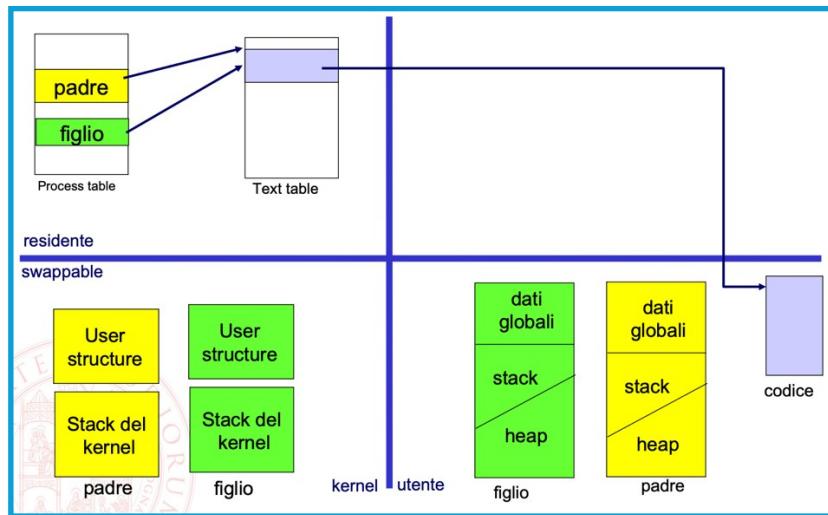
Mostreremo in rosso comandi o nomi di file, in blu l'output dei comandi, in verde l'input (inclusa righe nei file di configurazione)

**Bash** è una shell UNIX e può essere usata per programmare task da eseguire automaticamente anziché dover impartire comandi a mano. Ci sono due aspetti importanti da tenere a mente...

- 1) Gli elementi di base gestiti sono file e processi
- 2) Il linguaggio è interpretato e non compilato (subisce espansione)

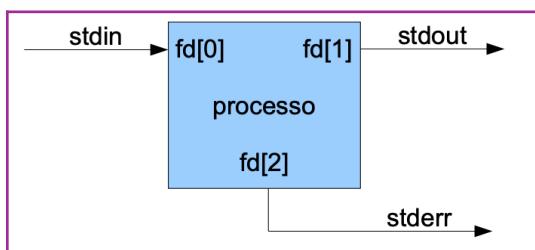
## 4.1 Processi

Ricordiamo che un processo padre può richiedere la creazione di un nuovo processo figlio realizzando così gerarchie di processi. **fork** crea una copia “pesante” del processo corrente; in particolare duplica tutte le risorse e condivide il codice.



Una system-call della famiglia **exec** separa codice padre-figlio caricando uno specifico programma. Quando si lancia un programma *Bash* la prima cosa che accade è dunque la duplicazione delle risorse.

- All'avvio il kernel inizializza i dispositivi HW e li espone come:
  - /dev/tty\* → terminali virtuali che accedono direttamente alla console
  - /dev/pts/\* → terminali che accedono a finestre grafiche
- Il device driver gestisce tali file e rende disponibili per la lettura i caratteri digitati dalla tastiera
- Viene avviato un processo di gestione del terminale (kernel DD: login). Si apre in lettura il file speciale con fd = 0 (std input) e in scrittura fd = 1 (std output) e fd = 2 (std error)
- Bash eredita di default i fd e comunica col terminale
- Dunque, tutti i comandi che operano su stream di testo sono progettati per disporre di tre stream.



Nell'interpretare la riga di comando, *Bash* può disconnettere gli stream predefiniti e far trovare gli stessi fd aperti su un file diverso. Analizziamo diversi tipi di ridirezione...

#### *da/verso file*

##### *ridirezione stdout / stderr:*

**ls > myfile** scrive lo stdout di ls nel file myfile

**ls >> my file** scrive lo stdout di ls nel file myfile (in append)

##### *confluenza stream:*

**ls > myfile 2>&1** ridirige stderr su stdout POI stdout su myfile (ORDINE IMPORTANTE!)

##### *ridirezione stdin:*

**sort < myfile** riversa il contenuto di myfile in input al comando sort

#### *speciali*

##### *ridirezione con MARCATORE (es EOF):*

**sort << EOF**

**prova**

**testo input**

**EOF** riversa il testo in input

##### *ridirezione singola linea:*

**sort <<< "testo da mandare a stdin"**

##### *ridirezione definitiva con exec:*

**exec 2>/dev/null**

*creazione nuovi fd con exec:*

**exec 3< filein 4> fileout 5<> filerw** Ogni lettura fatta con <&3 leggerà da filein; stesso discorso per 4 e 5. Per chiudere **exec 3>&-**

## 4.2 Processi e segnali

Ogni comando lanciato da shell o dal sistema diviene un processo. I processi sono identificati globalmente da uno specifico **PID** e in aggiunta in alcuni casi da un **Job ID**.

Un processo svolge le proprie azioni a nome dell'utente che lo ha lanciato (i processi lanciati da root hanno il potere di assumere l'identità di altri utenti, così facendo si "declassano" e perdono il potere di tornare indietro). I processi anche non lanciati da una stessa pipeline possono comunicare tra loro, ad esempio, usando apposite PIPE o per mezzo di SEGNALI.

I **segnali** sono eventi asincroni notificati dal kernel a un processo. Possono essere generati dal kernel stesso o da un altro processo; il contenuto informativo è limitato ad un numero.

Il controllo dei segnali ricevuti avviene ogni volta che il processo rientra in user space (ad esempio quando schedulato dalla CPU). Se tra un controllo e il successivo sono stati ricevuti più segnali, vengono posti in stato "pending": non una coda ma settato un semplice flag.

Ogni processo può "registrare" presso il sistema operativo una routine di gestione (**handler**) per un segnale. Alla rilevazione di un segnale pending, il flusso di esecuzione del processo a cui è destinato viene interrotto e viene eseguito l'handler.

Quando un processo riceve un segnale può comportarsi in tre diversi modi:

- Gestione di DEFAULT
- IGNORARE il segnale
- Eseguire HANDLER

(ricorda che fanno eccezione KILL e STOP)

**NOTA:** I signal handler non vengono ereditati dai processi figli!!

Il builtin trap permette di definire un'azione da eseguire alla ricezione di un segnale.

**trap [-lp] [[codice] segnale ...]**

```
#!/bin/bash

# Definisci una funzione da eseguire alla ricezione del segnale SIGINT
pulisci() {
    echo "Pulizia in corso..."
    # Altre operazioni di pulizia qui
    exit 0
}

# Intercetta il segnale SIGINT e esegui la funzione di pulizia
trap pulisci SIGINT
```

Oltre ai segnali standard riconosce pseudo segnali:  
**DEBUG** lanciato dalla shell prima di ogni comando  
**RETURN** lanciato dopo il ritorno da una funzione o dopo l'inclusione tramite source  
**ERR** lanciato ad ogni comando che fallisce  
**EXIT** lanciato dalla shell in uscita

Per inviare un segnale ad un processo si può usare kill.

**kill [options] <pid> [...]**

L'opzione -l elenca i segnali supportati **kill -signal\_number <pid>**

Alcune combinazioni di tasti sono trasformate in segnali, tipo...

Ctrl+Z → SIGTSTP

Ctrl+C → SIGINT

Ctrl+\ → SIGQUIT

Il comando sleep innesca un comando per far dormire il processo

**sleep <seconds>**

sono supportati i suffissi **m(minuti)** **h(ore)** **d(giorni)**

## 4.3 Processi in background

Si può usare un'unica shell per l'esecuzione contemporanea di più comandi che non abbiano necessità di accedere al terminale, lanciandoli in **background**.

Questo si ottiene postponendo il carattere **&** alla command line.

La shell risponde comunicando un numero tra parentesi quadre (**job id**) che identifica il job localmente a questa shell. Per usarlo al posto di un PID, si utilizza **%job\_id**

**MOLTO UTILE:** Il PID del processo viene memorizzato nella variabile **\$!**

Se si lancia una command line senza &, e si vuole rimediare, si può dare un segnale di STOP con Ctrl+Z; anche in questo caso si riceve un job id e con il comando **bg %job\_id**, si invia un segnale **CONT** che riavvia il processo e contemporaneamente lo mette in background.

**wait** permette di bloccare l'esecuzione fino al completamento dei job in background.  
Di default attende tutti i job ma si possono anche passare come argomento i *job\_id* specifici.

Se durante l'attesa la shell riceve un segnale per il quale è definito un handler con **trap**, **wait** esce immediatamente con exit code 128 e l'handler viene eseguito.

Se è necessario riportare in foreground (primo piano) un processo ricollegandolo così al terminale, si usa il comando **fg %job\_id**. Il comando **jobs** mostra l'elenco dei job, cioè di tutti i processi avviati dalla shell corrente, indicando il loro stato (attivo o stoppato).

*Modificatori di processi in bg:*  
**nohup <comando>** la shell, alla chiusura, non invia SIGHUP al comando  
**nice <comando>** lancia il comando con una niceness diversa da zero  
**disown** rimuove completamente un job dalla job table della shell

## 4.4 Shell expansion

### 1. Tokenizzazione

La riga viene divisa in token (stringhe, parole o carattere :) usando come separatori un elenco fisso di metacaratteri:

**SPACE TAB NEWLINE ; () < > | &**

Da qui in poi tutti i passi (2-10) sono saltati per le parti di riga racchiuse tra apici singoli.

### 2. alias?

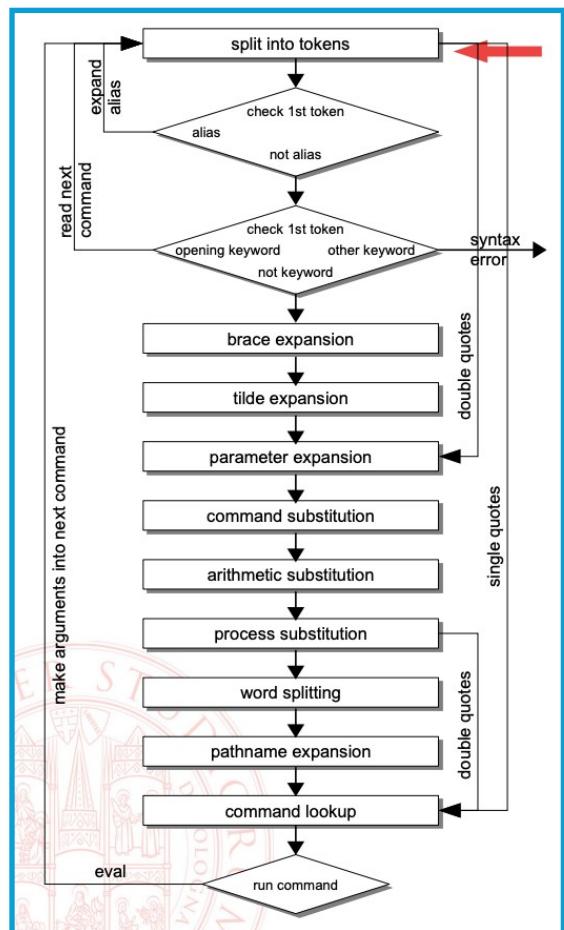
La shell cerca il primo token nella lista degli alias. Se lo trova, lo espande e riparte col processing dal punto 1. Non eseguito sulle parti di riga racchiuse tra doppi apici.

### 3. keyword?

Se il primo token è una parola chiave che dà inizio a un comando composto, ad es:

if – while – function – { – (

la shell predisponde l'ambiente per il comando composto. No se tra doppi apici.



### 4. Brace expansion

Es: Pre{Lista}Post → PreItem1Post PreItem2Post ...

Lista può essere

estensiva → {a,pippo,mamma}

sequenza → {min..max[..incr]}

esistono moltissime altre brace expansion. No se tra doppi apici.

### 5. Tilde expansion

Se un argomento inizia con `~`, la shell lo espande in base alla variabile d'ambiente `HOME`, sostituendo `~` con il percorso della directory home dell'utente corrente. Questa espansione consente di utilizzare rapidamente il percorso della directory home senza doverlo digitare esplicitamente. Ad esempio, se il tuo nome utente è "user123" e digiti `~/Documents`, la shell espanderà questa espressione in `/home/user123/Documents`, assumendo che la tua directory home sia `/home/user123`. No se tra doppi apici.

## 6. Parameter expansion

Il carattere '\$' può marcare l'inizio di diverse espansioni: parameter expansion, command substitution, arithmetic expansion. I passaggi da 6 a 9 sono eseguiti anche se racchiuse tra doppi apici!

## 7. Command substitution

Il token **\$ (comando)** ha questo effetto: viene creata una subshell, vi viene eseguito comando, sdtout di comando viene posto sulla riga di comando al posto del token originale.

## 8. Arithmetic expansion

Il token **((expr))** causa la valutazione di expr, un'espressione aritmetica; se preceduto da \$, il risultato viene posto sulla riga di comando, altrimenti l'unico effetto è eventualmente sulle variabili.

## 9. Process substitution

Il token **<(comando)>** o **>(comando)** ha questo effetto: viene eseguito comando in modo concorrente e asincrono rispetto al resto della riga; il suo input o output appare "come un nome di file" tra gli argomenti di tale comando.

## 10. Word splitting

I risultati dei passi 6..9 sono esaminati, e separati in word indipendenti. Il separatore è un qualsiasi carattere presente nella variabile *IFS* (default: **<space> <tab> <newline>**).

No se tra doppi apici.

## 11. Pathname expansion

Ogni word viene esaminata e se contiene uno dei caratteri \*, ?, [ viene considerata un pattern e sostituita con tutti i nomi di file che concordano. No se tra doppi apici.

## 12. Quote removal

Prima dell'esecuzione vengono rimosse tutte le occorrenze di caratteri di quoting "usate" effettivamente. Vengono impostati gli stream in caso di ridirezione. Viene cercato il comando in quest'ordine > **funzioni** > **builtin** > eseguibili in **\$PATH**

Il meccanismo di espansione di wildcard e variabili è potente ma interferisce con l'interpretazione letterale di alcuni simboli: `[]!*?${}()”‘`\\|><;`

Quando si debbono passare come parametro ad un comando delle stringhe contenenti tali simboli, è necessario proteggerli dall'espansione.

`\` (backslash) inibisce l'interpretazione del solo carattere successivo come speciale  
`'` (apice) protetto dall'espansione e trattato letteralmente, senza eccezioni.

`”` (apice doppio o virgolette) ogni carattere di una stringa racchiusa tra una coppia di virgolette viene protetto dall'espansione, con l'eccezione del `$`, del backtick (```), di `\`, ed altri casi particolari.

## 4.5 Filtri

Il meccanismo di ridirezione è utilizzato da un set di comandi pensati esattamente per elaborare stream di testo ricevuti via stdin, che producono risultati su sdtout, i **filtri**.

Vediamo i più comuni...

### cat

- invocato senza parametri copia stdin su stdout
- invocato con uno o più file ne riproduce in sequenza il contenuto
- il comando **tac** riproduce l'ingresso in ordine inverso

### less

- intercetta l'output e lo impagina permettendone la navigazione attraverso diversi comandi
- utile da porre a fine di una pipeline
- alcuni comandi:
  - F** – follow, scorre fino al termine dell'input e resta in attesa di nuove righe da mostrare
  - <N>g** – si porta alla riga numero <N> (default: 1)
  - G** – si porta al termine del file
  - /<pattern>** - cerca la riga successiva contenente <pattern>
  - ?<pattern>** - cerca la riga precedente contenente <pattern>
  - n** – ripete l'ultima ricerca
  - N** – ripete ma nel verso opposto
  - q** – esce da less

### rev

- inverte l'ordine dei caratteri in input verso lo stream di output

### head

- estrae la parte iniziale di un file (default: 10 righe)
- **-c NUM** produce i primi NUM caratteri (usando **-NUM** “eccetto gli ultimi”)
- **-n NUM** produce le prime NUM righe (usando **-NUM** “eccetto le ultime”)

### tail

- uguale al filtro precedente ma estrae la parte finale (usando **+NUM** “a partire da”)
- **-f** si visualizzano in tempo reale eventuali nuove righe

### cut

- permette di “tagliare” parti di righe, il modo più semplice è il seguente: **-c**  
**cut -cPOSIZIONI\_CARATTERI [file]**  
**cut -c15 solo 15° carattere**  
**cut -c8-20 caratteri da 8° a 20°**  
**cut -c-30 fino al 30°**  
**cut -c8- dopo 8°**
- in caso di campi opportunamente delimitati per file organizzati a record, le opzioni **-d** e **-f** permettono di estrarre uno o più campi  
**cut -dDELIMITATORE -fELENCO\_CAMPPI**

```
cut /etc/passwd | cut -f1 -d: -s  
estratto primo campo usando come delimitatore :  
-s salta le righe che non contengono il delimitatore
```

## sort

- ordina in modo lessicale
- **-u** elimina entry multiple
  - r** reverse
  - R** random
  - b** ignora gli spazi bianchi a inizio riga
- //altro su slide
  - tSEP** imposta SEP come separatori di campi (default: spazi)
- //altro su slide

## uniq

- elimina i duplicati consecutivi

## wc

- filtro di conteggio
- **-c** caratteri
- l** linee
- w** parole (separate da spazi)

## grep

- esamina le righe in ingresso e riproduce in uscita quelle che contengono un pattern corrispondente ad una espressione regolare (passata come argomento)

ls | grep prova  
cerca la stringa prova all'interno dell'output ls

→ **egrep** supporta le espressioni regolari "moderne" (man page regex(7)):

**RE** = uno o più rami non vuoti separati

Ramo = uno o più pezzi concatenati tra loro

Pezzi = atomi con eventuali moltiplicatori

Atomo = **(RE)**

[charset]

^ inizio riga, \$ fine riga, . carattere qualsiasi che non sia riga vuota  
(per usare il punto uso il backslash: \.)

Backslash sequence

Singolo carattere

Backslash sequence:

- \< - \>** la stringa vuota all'inizio – alla fine di una parola.
- \b** la stringa vuota al confine di una parola
- \B** la stringa vuota a condizione che non sia al confine di una parola
- \w** è sinonimo di "una qualsiasi lettera, numero o \_"
- \W** è un sinonimo "un qualsiasi carattere non compreso in \w"

Moltiplicatori:

{n,m}

indica da n a m occorrenze dell'atomo che lo precede

?

indica zero o una occorrenza dell'atomo che lo precede

\*

indica zero o più occorrenze dell'atomo che lo precede

+

indica una o più occorrenze dell'atomo che lo precede

Charset:

[abc] indica UN qualsiasi carattere fra a, b o c

[a-z] indica UN qualsiasi carattere fra a e z compresi

[^dc] indica UN qualsiasi carattere che non sia né d né c

Charset class:

[ :NOME\_CLASSE: ]

dove **NOME\_CLASSE** deve appartenere all'insieme definito in **wctype(3)**  
come tipicamente:

alnum digit punct alpha graph space  
blank lower upper cntrl print xdigit

o eventualmente nel *locale* attivo

→ Nel caso in cui una RE possa corrispondere a più di una sottostringa di una data stringa, la RE corrisponde a quella che inizia per prima nella stringa. Se a partire da quel punto, la RE può corrispondere a più di una sottostringa, selezionerà la più lunga.

→ Controlli di matching:

-E usa la extended RE

-F disattiva RE e usa il paramentro come stringa letterale

-i rende l'espressione insensibile a maiuscole o minuscole

→ Controlli input:

-r cerca ricorsivamente in tutti i file di una cartella

-f file prende le RE da file invece che come parametro

→ Controlli output:

-o restituisce solo le sottostringhe che contengono RE

-v restituisce le linee che non contengono RE

-l passando più file, restituisce i nomi dei file che contengono RE

-n restituisce anche il numero della riga con RE

-c conteggio delle righe con RE

--line-buffered disattiva il buffering

Comandi come **sed** e **awk** consentono operazioni ancora più complesse. Si comportano come veri linguaggi di programmazione.

**tr**

→ sostituisce rapidamente caratteri

tr 'A-Z' 'a-z' trasforma le maiuscole in minuscole

Oltre i filtri...

## xargs

Può essere necessario inserire in una pipeline comandi che non leggono stdin, ma vogliono parametri sulla riga di comando

xargs <comando> si aspetta sullo standard input un elenco di stringhe, ed invoca poi comando con tali stringhe come argomenti.

Es.

```
pipeline | cheproduce | nomi_di_file | xargs ls -l
```

lancerà ls -l per ogni file ricevuto dalla pipeline

Particolarità del comportamento di xargs

- xargs raggruppa le invocazioni in modo da ridurre il carico. Questo può funzionare con comandi come ls, ma non se il comando accetta un singolo parametro
- xargs passa le righe dell'input così come sono sulla riga di comando costruita. La presenza di spaziatori quindi farà percepire al comando invocato una molteplicità di parametri

Opzioni utili:

<b>-0</b> (zero)	utilizza null, non lo spazio, come terminatore di argomento
<b>-L MAX</b>	usa al più <b>MAX</b> linee di input per ogni invocazione
<b>-p comando</b>	chiede interattivamente conferma del lancio di ogni

## process substitution

Supponiamo di avere cmd\_producer\_su\_stdout  
che in "stile filtro" genera dati su stdout

Supponiamo che cmd\_consumer\_da\_file non accetti stdin, e voglia invece come parametro un file da leggere ed elaborare

Non si possono connettere con una pipe

Si potrebbe usare un file temporaneo

```
cmd_producer_su_stdout > tmpfile  
cmd_consumer_da_file tmpfile
```

Svantaggi?

Soluzione: process substitution

```
cmd_consumer <(cmd_producer_su_stdout)
```

- Il processo tra parentesi è lanciato concorrentemente all'altro e la shell genera un nome di file (sarà una named pipe) da fornire al primo

Caso simmetrico:

```
cmd_producer_su_file >(cmd_consumer_da_stdin)
```

## tee

tee è un comando utile per duplicare uno stream di output

- invia una copia del proprio stdin a stdout
- invia una copia identica in un file passato come parametro
- opzione **-a**: il file è aperto in append

```
comando1 | tee FILE | comando2
```

si noti che combinato alla process substitution permette anche varianti più complesse

```
ls | tee >(grep foo | wc > foo.count) |  
      tee >(grep bar | wc > bar.count) |  
      grep baz | wc > baz.count
```

## shuf

genera permutazioni random delle righe di stdin o di un file

- esattamente come `sort -r`

ma anche...

- degli argomenti passati, se attiva l'opzione `-e`

Es. `shuf -e uno due tre`

due

uno

tre

- di un range di numeri, con l'opzione `-i`

Es. `shuf -i 1-5`

5

1

3

2

4

ripetendo all'infinito estrazioni casuali degli argomenti, con l'opzione `-r`

## command substitution

La **command substitution** permette di utilizzare direttamente un processo per generare parametri da collocare sulla command line per un altro comando

Due sintassi

``comando``

`$ (comando)`

- equivalenti a meno di piccoli dettagli sul trattamento di alcuni caratteri speciali che potrebbero apparire in `comando`
- `comando` viene eseguito in una subshell
- il suo stdout compare sulla riga di comando al posto del token di command substitution

Es.:

`ls $(cat /etc/passwd | cut -f6 -d:)`

- estrae le home dir degli utenti e le pone come parametri a `ls`

## 4.6 Shell scripting: variabili

Ricordiamo che alcuni simboli: [ ] ! \* ? \$ { } ( ) " ' ` \ | > <; vanno protetti da espansione tramite \ ‘ oppure “....

Ricordiamo inoltre che il comando **echo** stampa i caratteri che lo seguono, utile ad esempio per visualizzare il contenuto di una variabile: echo \$VAR

### Pathname expansion

- \* rappresenta una qualunque stringa di zero o più caratteri
- ? rappresenta un qualunque carattere singolo
- [SET] rappresenta un qualunque carattere appartenente a **SET**
  - SET può essere un elenco, es. **[afhOV]**
  - SET può essere un intervallo, es. **[a-k]**
    - l'ordine dipende dal *locale*
    - più intervalli si possono unire con la virgola es. **[a-d,0-5]**
  - SET può essere negato con ! o ^, es **[!a] [^A-Z]**
  - SET può essere una classe come per egrep, es. **[[:alnum:]]**
  - per includere - o ] nel SET, metterli come primo carattere

ER 37/14

### Brace expansion

- È un meccanismo di espansione per generare sequenze di stringhe secondo un pattern con la stessa sintassi della pathname expansion, ma le stringhe sono generate indipendentemente dal fatto che esistano o meno file che rispettano il pattern
- Sintassi:  
**[PRE]{LISTA}[POST]** oppure **[PRE]{SEQUENZA}[POST]**
- Esempio con lista:  
**a{d,c,b}e** → espanso dalla shell in **ade ace abe**
- Esempi con sequenza:
  - file{9..13..2}.c** → **file9.c file11.c file13.c**
  - doc{009..11}** → **doc009 doc010 doc011**
  - {a..j..3}** → **a d g j**
- “prodotto cartesiano”  
**{a..c}{1,3}** → **a1 a3 b1 b3 c1 c3**
- nesting  
**p{1{a,b},2,3{b,d}}** → **p1a p1b p2 p3b p3d**

incremento  
opzionale

zero-padding

solo singolo  
carattere  
alfabetico

→ Le **variabili** permettono di memorizzare una stringa di testo. La creazione avviene tramite = senza inserire spazi prima o dopo. L'espansione del nome di una variabile con il suo valore avviene tramite \$.

**pippo=valore**

**\$pippo**

→ Le **variabili di ambiente** non rimangono confinate all'interno della shell come le variabili standard. È necessaria la loro esportazione: **export pippo**

Il comando **set** visualizza l'ambiente.

### Variabili notevoli di bash

#### ■ Settate da bash:

- **BASHPID** PID della shell corrente
- **\$** PID della shell "capostipite"
- **PPID** PID del parent process della shell "capostipite"
- **HOSTNAME** nome dell'host
- **RANDOM** un numero casuale tra 0 e 32767
- **UID** id utente che esegue la shell

#### ■ Usate da bash:

- **HOME** home directory dell'utente
- **LC\_vari** scelta dei vari aspetti della localizzazione
  - **man locale(7)**, anche **locale(1)** e **locale(5)**
- **PS0...PS4** prompt in diversi contesti

#### ■ Altre decine

- **man bash** → **Shell variables**

→ Ogni script può accedere agli argomenti indicati sulla propria linea di comando:

**\$0** (comando), **\$1** (arg1), **\$2** (arg2), ...

(se NUM > 9 usare \${NUM})

**"\$\*"** si espande in un'unica stringa con tutti gli argomenti separati da spazi **"\$1 \$2 ..."**

**"\$@"** si espande come lista di argomenti separati **"\$1" "\$2" "..."**

**"\$#"** contiene il numero di argomenti

→ Il comando **shift** sposta a sinistra di uno il valore delle variabili (escluso \$0)

→ Può essere utile saper gestire **variabili di default**:

**FILEDIR=\${1:-"/tmp"}**

Usa di default /tmp se \$1 è null o not set (non altera \$1)

**cd \${HOME:=/tmp}**

Usa di default /tmp se \$HOME è null o not set (poi espansa)

**FILESRC=\${2:?“Error”}**

Se \$2 è null o not set stampa l'errore ed esce

→ Manipolazione **search&replace**:

<i>name:number:number</i>	Substring starting character, length
#name	Return the length of the string
name#pattern	Remove (shortest) front-anchored pattern
name##pattern	Remove (longest) front-anchored pattern
name%pattern	Remove (shortest) rear-anchored pattern
name%%pattern	Remove (longest) rear-anchored pattern
name/pattern/string	Replace first occurrence
name//pattern/string	Replace all occurrences

[Es. cambio estensione da .txt a .page: mv “\${fn}” “\${fn/.txt/.page}”]

→ Accesso **indiretto**, nome di una variabile usata come riferimento indiretto:

**chiave=pippo**

**pippo=valore**

**echo \${!chiave}**

**valore**

→ **Array** con indice numerico; dichiarazione non obbligatoria: **declare -a MYVECTOR**

Assegnamento: **A[0]=valore**

Accesso a celle: **echo \${A[0]}**

Inizializzazione diretta: **MYVECTOR=(el1 el2 el3 ...)**

Inizializzazione con separatori personalizzati: **EL=(el1.el2)**

**IFS='.' MYVECTOR=(\$EL)**

Visualizzare tutti gli elementi: **“\${A[\*]}” o “\${A[@]}”**

NOTIAMO CHE GLI INDICI POSSONO ESSERE NON CONSECUTIVI!

Set di indici di celle assegnate: **echo \${!A[@]}**

Numero di celle usate: **echo \${#A[\*]}**

→ **Array associativi** in cui l'indice può essere una stringa (mappe chiave-valore):

```
declare -A ASAR
ASAR[chiaveuno]=valore
echo ${ASAR[chiaveuno]}
valore
```

```
KEY=chiaveuno
echo ${ASAR[$KEY]}
valore
```

Il builtin **read** legge stringhe da stdin e le assegna a variabili; l'input viene tokenizzato usando IFS. Se ci sono più token che variabili, quelli in eccesso finiscono tutti nell'ultima.

```
read A B C
io mi chiamo Alessandro
echo $A / $B / $C
io / mi / chiamo Alessandro
```

Alcune opzioni:

- p **PROMT** stampa PROMPT prima di accettare input
- u **FD** legge dal file descriptor FD specificato
- a **ARRAY** assegna i token agli elementi di ARRAY

NOTA: attenzione ai sottoprocessi generati dalla pipe...

<pre>echo ciao   read A echo \$A (niente)</pre>	<pre>echo ciao   ( read A ; echo \$A ) ciao</pre>
---	---

## 4.7 Shell scripting: aritmetica

Bash tratta quasi tutto come stringhe, salvo per contesti particolari...

Possiamo dichiarare valori come numeri interi e far avvenire “arithmetic evalutation”:

```
declare -i N
N="3 * (2 + 5)"
echo $N
21
```

Possiamo usare anche il comando **let**:

```
let N++
echo $N
21
```

Oppure **(( ))** che si comporta come “ ” nel proteggere dalle espansioni, ma in più riconosce le variabili senza espansione \$ e le riconosce anche se non sono numeri interi.

Il token **\$(( ))** viene espanso col risultato.

**Esempio**

```
counter=0
declare -p counter
declare -- counter="0"
echo $(( counter++ )) $(( counter++ )) $(( counter++ ))
0 1 2
counter=$(( counter * newvar + 100 ))
echo $counter
100
```

“print” mostra tipo e valore del simbolo  
“--” nessun tipo

Gli operatori sono sostanzialmente quelli del C...

<b>id++ id-- ++id --id</b>	post/pre-incremento/decremento
<b>+ - * /</b>	somma sottrazione prodotto divisione
<b>** %</b>	elevamento a potenza, modulo
<b>! ~ &amp;   ^</b>	NOT NOT AND OR XOR bit-a-bit
<b>&lt;&lt; &gt;&gt;</b>	shift binario a sinistra / destra
<b>= *= /= %= += -= &lt;=&gt; &gt;= &amp;= ^=  =</b>	assegnamento
<b>&lt;= &gt;= &lt; &gt; == !=</b>	confronto
<b>&amp;&amp;   </b>	AND / OR logico
<b>expr1?expr2:expr3</b>	restituisce il risultato di expr2 o expr3 rispettivamente se expr1 è true o false

## 4.8 Shell scripting: controllo di flusso

**( )** raggruppa comandi per trattarli come un singolo processo aprendo una subshell

**{ }** crea una sequenza mantenendo l'esecuzione nello spazio di memoria della shell principale  
ogni comando deve essere terminato da newline o ;

**{ read A; read B }**

Le **funzioni** sono sequenze di comandi con un nome...

**function NOME() { SEQUENZA ; }**

possono ricevere parametri utilizzabili come quelli posizionali dello script (\$1, \$2, ...).

Il nome della funzione viene posto in **\$FUNCNAME**.

Se invocate “semplicemente”, sono eseguite nel contesto del chiamante con stesso spazio di memoria e variabili locali MA ATTENZIONE alle invocazioni in pipeline.

I comandi di controllo di flusso decidono un percorso in base all'exit code di un processo (0 true, altri false). Questo exit code si trova nella variabile speciale settata dalla shell **\$?**.

Esistono vari **interpreti di espressioni logiche**:

→ Builtin: **test**, **[**, **[[** **]]**

→ Comandi: **test**, **[ ]**

I builtin vengono eseguiti prima di cercare altri comandi, fare riferimento a quelli...

*Da sistemi operativi....*

Il comando **test** valuta un'espressione. Restituisce uno stato uguale a zero in caso di "true".

I test possono essere eseguiti su *file*:

- **test -f <nomefile>** esistenza di file
- **test -d <nomefile>** esistenza di directory
- **test -r <nomefile>** diritto di lettura sul file (-**w** e -**x**)

*stringhe:*

- **test -z <stringa>** vero se **stringa nulla**
- **test <stringa>** vero se **stringa non nulla**
- **test <stringa1> = <stringa2>** uguaglianza stringhe
- **test <stringa1> != <stringa2>** diversità stringhe

*valori numerici:*

- **test <val1> -gt <val2>** (**val1>val2**)
- **test <val1> -lt <val2>** (**val1<val2**)
- **test <val1> -le <val2>** (**val1<=val2**)
- **test <val1> -ge <val2>** (**val1>=val2**)

Al posto di test si possono usare **[ ... ]** (builtin) o **[[ ... ]]** (keyword).

La semantica è la stessa... **test -d mydir** equivale a **[ -d mydir ]**

Gli spazi sono ancora significativi... **[ [ a = b ] ]**

I vantaggi sono che si possono usare:

- op. logici **&&** e **||**
- op. **>= < <= !=**
- metacaratteri interni per **[[ ... ]]**...

*Esempio:* **[[ \$a = res\* ]]**

restituisce 0 (=vero) se \$a contiene una stringa che inizia per res seguito da zero o più caratteri.

- **=~** confronta regular expressions complesse

*Esempio:* **[[ \$a =~ ^[0-9]+\$ ]]**

restituisce 0 (=vero) se \$a contiene una stringa con un numero intero positivo

Gli operatori logici possono essere usati anche sulla riga di comando per combinare gli exit code di qualsiasi processo:

**cd "\$MYDIR" && ls "\$MYFILE"**

ritorna true se riesco ad entrare in \$MYDIR ed elencare \$MYFILE

Nota che sia in questo caso che nei precedenti la valutazione è efficiente: se il primo operando basta a determinare il risultato, il secondo non è considerato.

*Da sistemi operativi STRUTTURE DI CONTROLLO....*

Struttura **if**:

```
if <condizioni>
then
<comandi>
[elif <condizioni>
then
<comandi>]
[else <comandi>]
fi
```

```
if <condizioni>; then <comandi>
fi
```

```
if ! [[ $a = no ]]; then echo hai scritto si
else echo hai scritto no
fi
```

Struttura **switch**:

```
case <var> in
<pattern-1> )
<comandi>;
<pattern-2> | <pattern-3>)
<comandi>;
esac
```

Struttura **for**:

Ripetizione del ciclo per ogni stringa nella lista.

```
for <var> [in <list>]
do
<comandi>
done
```

```
for i in $*
do
echo $i
done
```

stampa tutti gli argomenti in input del comando

Struttura **while**:

Ripetizione del ciclo finché la condizione è vero o finché un comando non termina con stato =0.

```
while <condiz. o lista comandi>
do
<comandi>
done
```

```
until <condiz. o lista comandi>
do
<comandi>
done
```

(Valgono: continue, break e return e system call exit[status])

Nelle versioni più recenti di bash esiste anche una sintassi C-LIKE: (START, TEST, EX-END)

```
for (( i=0, j=0 ; i+j < 10 ; i++, j+=2 ))
```

## 4.9 Shell scripting

*Da sistemi operativi ricordiamo infine che....*

È possibile creare file (comandi o shell script) che contengono comandi da eseguire in sequenza.

```
#!/bin/bash // indica di eseguire il file usando shell bash/  
A=5  
B=8  
echo C=`expr $A + $B`
```

} file: somma

```
[MBP-di-apple:Desktop Alessandro$ ./somma  
C=13
```

Prima dell'esecuzione la shell prepara il comando attraverso 3 passi di sostituzione ed espansione:

- 1) sostituzione dei comandi: i comandi tra `` sono eseguiti e sostituiti
- 2) sostituzione variabili: i nomi \$VAR sono espansi con il loro valore
- 3) sostituzione metacaratteri: i metacaratteri sono espansi per matching

**source** comando.sh

- può essere usato per eseguire uno script nel contesto di un altro
- è utilizzato per eseguire uno script e rendere disponibili variabili, funzioni e altre definizioni all'interno del chiamante (utile per creare librerie importabili)

**eval**

- permette di processare un file come se fosse uno script sottoponendolo ai 12 passi di valutazione; permette ad uno script di generare altri script

```
listp="ls | more"  
$listp  
ls: cannot access |  
eval $listp  
...elenco file paginato...
```

- forza l'espansione delle variabili

## select

→ built-in utile per semplificare la selezione tra le alternative; costruisce un menu a scelta multipla all'interno di uno script interattivo.

```
select var in "option1" "option2"
do
  if[ "$var" = "option1" ]; then ...; fi
done
```

## getopts

→ utile per costruire uno script con sintassi: comando OPZIONE\_PREC\_TRATTINO ARGOMENTI

```
While getopts "a:b:" option
do
  if[ $option = a ]; then ...; fi
done
```

## date

→ è un comando ricco di opzioni, può essere utilizzato per interrogare l'orologio e convertire formati. Usando il comando **-s** imposta l'orologio del sistema.

## time

→ è una keyword che anteposta a un comando ne riposta la durata

## mktemp

→ crea un file temporaneo in /tmp, univoco e con non prevedibilità del nome

→ il comando restituisce il nome del file creato

→ opzioni:

**-d** crea una directory

**-p DIR** crea il file all'interno di DIR invece che in /tmp

## basename

→ rimuove il path ed eventualmente un suffisso da un nome di file

## dirname

→ rimuove l'ultimo componente del percorso di un file

## printf

- permette di formattare gli argomenti in modo più complesso rispetto a echo
  - printf [-v var] format [arguments]
  - v var assegna il risultato della formattazione alla variabile invece che riprodurlo su stdout
  - format è una stringa di formato documentata da printf
  - arguments sono gli argomenti a cui applicare la stringa di formato

## script

- crea una copia di tutto ciò che appare sul terminale (comandi, output, ...)
- termina con CTRL-D o exit
- produce un file **typescript** che può essere usato come documentazione o dato come argomento a **scriptreplay**

## watch

- esegue un comando periodicamente mostrando l'output sul terminale (è uno strumento di monitoraggio non di automazione)
- -d evidenzia le differenze dall'ultimo aggiornamento
- -n <INTERVALO> imposta l'attesa tra un'esecuzione e la successiva

Anchors		Sample Patterns	
^	Start of line +	([A-Za-z0-9]+)	Letters, numbers and hyphens
\A	Start of string +	(\d{1,2}\.\d{1,2}\.\d{4})	Date (e.g. 21/3/2006)
\$	End of line +	([^\\s]+(?:\\.(jpg gif png))\\.)\\2)	jpg, gif or png image
\Z	End of string +	(^\\[1-9]\\{1\\}\$ \\^\\[1-4]\\{1\\}\\[0-9]\\{1\\}\\\$ \\^\\[50\\]\$)	Any number from 1 to 50 inclusive
\b	Word boundary +	(#?(\\[A-Fa-f0-9]\\{3\\})\\((\\[A-Fa-f0-9]\\{3\\})\\{3\\})?)	Valid hexadecimal colour code
\B	Not word boundary +	(?=.*\\d)(?=.*[a-z])(?=.*[A-Z]).\\{8,15\\})	8 to 15 character string with at least one upper case letter, one lower case letter, and one digit (useful for passwords).
\<	Start of word	(\\w+@[a-zA-Z_]+?\\.[a-zA-Z]\\{2,6\\})	Email addresses
\>	End of word	(\\<(/?[^\\>]+\\)>)	HTML Tags
Character Classes		<b>Note</b> These patterns are intended for reference purposes and have not been extensively tested. Please use with caution and test thoroughly before use.	
\c	Control character	Quantifiers	
\s	White space	*	0 or more +
\S	Not white space	*?	0 or more, ungreedy +
\d	Digit	+	1 or more +
\D	Not digit	+?	1 or more, ungreedy +
\w	Word	?	0 or 1 +
\W	Not word	??	0 or 1, ungreedy +
\xhh	Hexadecimal character hh	{3}	Exactly 3 +
\Oxxx	Octal character xxx	{3,}	3 or more +
POSIX Character Classes		{3,5}	3, 4 or 5 +
{3,5}?		{3,5}?	3, 4 or 5, ungreedy +
Special Characters		<b>Note</b> Ranges are inclusive.	
		\	Escape Character +
		\n	New line +
		\r	Carriage return +
		\t	Tab +
		\v	Vertical tab +
		\f	Form feed +
		\a	Alarm
		[\b]	Backspace
		\e	Escape
		\N{name}	Named Character
Assertions		String Replacement (Backreferences)	
?=	Lookahead assertion +	\$n	nth non-passive group
?!	Negative lookahead +	\$2	"xyz" in /^(abc(xyz))\$/
?<=	Lookbehind assertion +	\$1	"xyz" in /^(?:abc)(xyz)\$/
?!= or ?<!	Negative lookbehind +	\$`	Before matched string
?>	Once-only Subexpression	\$'	After matched string
?()	Condition [if then]	\$+	Last matched string
?()	Condition [if then else]	\$&	Entire matched string
?#	Comment	\$_	Entire input string
<b>Note</b> Items marked + should work in most regular expression implementations.		\$\$	Literal "\$"
Pattern Modifiers		Metacharacters (must be escaped)	
		^	[
		\$	{
		(	\
		)	+
		)	?
		<	
		>	.

Available free from  
AddedBytes.com

# 5 Utenti, Gruppi e File

Gli utenti sono i soggetti di tutte le operazioni svolte sul sistema, sono utilizzati per determinare i permessi di accesso a qualsiasi risorsa. Ogni utente deve appartenere ad un gruppo, può inoltre appartenere ad un numero arbitrario di gruppi supplementari.

L'utente **root** è l'utente amministratore con privilegi più alti. Per eseguire un comando come root, è necessario anticiparlo con **sudo** oppure aprire una shell root tramite **sudo -i**.

## useradd

- per creare un nuovo utente, per invocare il comando sono necessari permessi root
- caratteristiche di default nel file: **/etc/login.defs**
- **-m** crea la home dell'utente
- **-s** assegna la shell
- **-U** crea un gruppo con lo stesso nome dell'utente
- **-K** per modificare la UMASK=0077

Le credenziali locali sono in generale salvate nei file: **/etc/passwd** e **/etc/shadow**  
(il secondo accessibile solo da root)

## id

- visualizza le informazioni dell'utente corrente come User ID, Group ID, gruppi secondari
- possibile anche verificare le informazioni di un altro utente: **id <nome>**

## usermod

- attraverso i parametri di questo comando è possibile modificare le caratteristiche di un utente
- esistono anche comandi per modificare unicamente certi valori, come la password:  
**passwd**

## whoami

- stampa lo username dell'utente corrente

## who

- elenca gli utenti attualmente collegati alla macchina

## 5.1 Gruppi

**addgroup <nome>**

→ crea un gruppo

**usermod -a -G <nomeGruppo> <nomeUtente>**

→ questo comando specifico aggiunge un utente a un gruppo e lo setta come gruppo secondario, se si vuole cambiare in gruppo primario usare: **-g**

**groups**

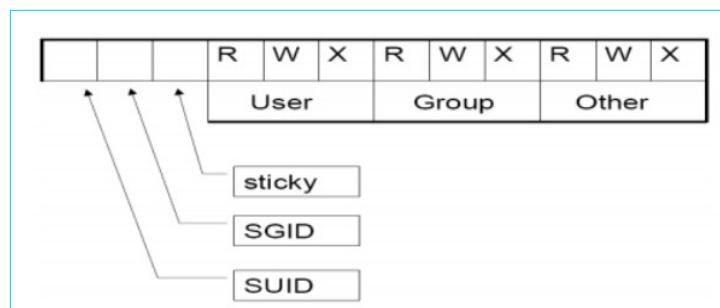
→ elenco dei gruppi dell'utente corrente (altrimenti specificare l'utente)

**getent group**

→ elenco di tutti i gruppi del sistema

## 5.2 Permessi

Su Unix tutto è file ed ognuno di essi è descritto da un identificatore (i-node). Ogni file è associato a 12 bit di permessi di cui gli ultimi 9 relativi a tre a User Group e Other:



*Per i file:*

[ R = permesso di lettura      W = di scrittura      X = di esecuzione ]

*Per le dir:*

[ R = elencare i file della dir    W = modifica file nella dir    X = permesso di entrare nella dir ]

## Da sicurezza...

I tre bit più significativi configurano comportamenti speciali.

	11 – SUID	10 - SGID	9 – STICKY / TEMP
<b>file</b>	Se settato a 1 (su un file eseguibile) fa sì che al lancio il SO generi un processo con l'identità dell'utente proprietario	Come SUID ma agisce sull'identità del gruppo.	"STICKY" OBSOLETO. Il SO tiene in cache una copia del programma
<b>dir</b>	non usato	Se l'utente appartiene anche al gruppo proprietario della dir, assume come gruppo attivo quello della directory. Nelle aree collaborative i file sono aperti al gruppo mentre in quelle personali sono privati.	"TEMP" Questo bit settato a 1 impone che nella directory i file siano cancellabili solo dai rispettivi proprietari.

Vediamo ora, quali sono i comandi che interagiscono con i permessi.

### chmod

- il seguente comando serve a modificare i permessi e presenta diverse possibili sintassi:
- chi? a (tutti), u (user), g (group), o (other)
- come? = (setta), + (aggiungi), - (sottrai)
- cosa? w, r, x, s (suid)

**chmod u+wx <file>**

- formato numerico: **chmod 0777 <file>**

### chown

- cambia la proprietà di un file
- cambia l'utente proprietario: **chown <username> <file>**
- cambia gruppo proprietario: **chown :<group> <file>**
- cambia utente-gruppo proprietario: **chown <username>:<group> <file>**

### umask

- visualizza l'umask corrente, ovvero una maschera che stabilisce quali permessi verranno sottratti automaticamente alla creazione di un file.
- set in formato numerico: **umask 0007 <file>**

(nota che l'umask sarà valida solo per la sessione corrente)

## 5.3 Lavorare con i file

**rm** → cancella un file (rimuove il link, cancellato se link = 0)

**cp** → copia un file

**mv** → sposta un file

**ln** → crea un link

**mkdir** → crea una dir

**rmdir** → cancella un dir se vuota

**find** → ricerca in tempo reale, ad esempio, file con certo nome

**dd** → lettura a byte da un file (if=<nome>) e scrittura su file (of=<nome>)

**tar** → archiviazione di file

**gzip** → compressione in formato .gz

# 6 Servizi e Logging

Nel gergo Unix vengono definiti “demoni” i servizi di sistema avviati e arrestati automaticamente ed eseguiti a nome di utenti specifici senza accesso a shell interattiva.

Operazioni comuni sui demoni sono:

- Configurare le condizioni per il loro avvio e arresto automatico
- Esaminare lo stato
- Tracciare l'esecuzione
- Avviarli, arrestarli o inviare loro segnali

Il comando **ps** serve a mostrare informazioni sui processi attualmente in esecuzione.

Possiamo inoltre seguire il comando con (**ps aux**):

- a**: Mostra i processi di tutti gli utenti, non solo quelli appartenenti all'utente che ha eseguito il comando.
- u**: Visualizza le informazioni dettagliate sui processi, come il nome dell'utente, la percentuale di utilizzo della CPU e della memoria, il tempo di esecuzione del processo, ecc.
- x**: Include anche i processi che non sono collegati a un terminale (cioè i processi daemon o background).

Notiamo che processi inutili consumano risorse e offrono opportunità di attacco.

Oltre agli utenti, possiamo individuare tre primarie fonti di processi: pianificatori periodici, demoni di gestione degli eventi, procedure di avvio del sistema.

## 6.1 Cron: esecuzione periodica

Per eseguire periodicamente i comandi possiamo usare **Cron (crond)**; ogni utente presenta una propria cron table (**crontab**) visualizzabile in </var/spool/cron>. I task di sistema sono invece raccolti in </etc/crontab>.

A ogni comando corrisponde una riga con il seguente formato:



L'azione viene eseguita quando l'ora corrente fa match con tutti i campi di una riga...

- \* fa match con tutti gli elementi
- si specifica un range di valori con -
- usando la virgola possiamo specificare più range

Esempi:

MINUTO ORA G.MESE MESE G.SETTIMANA <comando>

```
*      *    27    *      *    $/HOME/paga  
30    8-18 *      *    1-5   /bin/backup
```

ATTENZIONE: se sono specificati diversi da \* sia il G.MESE sia G.SETTIMANA, entrambi i giorni sono considerati per eseguire l'azione.

Mentre i task di sistema raccolti in /etc/crontab sono editabili direttamente, per editare le tabelle utente è possibile usare i seguenti comandi:

### crontab

- **-e** modifica tabella dell'utente corrente
- u <username> -e** modifica tabella utente specificato
- l** lista dei cronjob
- r** eliminare la tabella

### systemctl status crond

- verifica lo stato del demone
- **systemctl start|stop|restart crond**

*Esempi di configurazioni...*

Un job ogni minuto:

```
*      *      *      *      *    script.sh
```

Un job ogni domenica alle 17:

```
0     17    *      *      sun  script.sh
```

Un job ogni 10 minuti:

```
*/10  *      *      *      *    script.sh
```

Un job ogni 30 secondi (nativamente impossibile, bisogna usare un trucco):

```
*      *      *      *      *    script.sh  
*      *      *      *      *    sleep 30; script.sh
```

ATTENZIONE: crontab richiede il PATH assoluto di un comando!

*Editare crontab da uno script...*

```
# Salva il crontab attuale in un file temporaneo  
crontab -l > tmpcron.txt  
  
# Aggiungo il nuovo comando in append al file  
echo "* * * * * /bin/echo ciao" >> tmpcron.txt  
  
# Aggiorna il crontab con il nuovo file  
crontab tmpcron.txt  
  
# Rimuovo il file temporaneo  
rm tmpcron.txt
```

## 6.2 At: posticipare l'esecuzione

Il comando **at** in Linux è utilizzato per pianificare l'esecuzione di un comando una singola volta in un momento futuro. Una volta specificato l'orario **at [time]** si aprirà un prompt per l'inserimento dei comandi che saranno da eseguire. Per terminare CTRL+D.

L'orario può essere specificato in diversi formati, per semplicità, ecco alcuni esempi...

**at 16:00**  
**at 16:00 tomorrow**  
**at now + 2 hours** (tra 2 ore)  
**at 12:00 12/10/2027**

(la quantità minima è il minuto)

Possiamo impostare un comando anche usando una pipe: **echo ciao | at 16:00**

Visualizziamo i task in coda con **atq**, una volta scoperto l'id di un job (primo valore) è possibile rimuoverlo con **atrm <id>**.

## 6.3 systemd

**Systemd** è un sistema di init e un gestore di servizi per sistemi operativi basati su Unix (come Linux), progettato per avviare, fermare e gestire servizi e processi di sistema. Sostituisce i tradizionali sistemi di init come *SysVinit* o *Upstart*, migliorando la gestione dei servizi di sistema, riducendo i tempi di avvio e offrendo una gestione più robusta delle dipendenze tra i servizi.

→ **systemd** **gestisce servizi** (detti anche **unit file**) come processi di sistema, demoni o script che devono essere eseguiti al boot o in momenti specifici.

→ tiene traccia delle **dipendenze** tra i servizi e assicura che i servizi richiesti siano avviati nell'ordine corretto.

→ Gli **unit file** sono semplici file di configurazione con estensioni specifiche, come “.service” per i servizi, “.socket” per le socket, “.mount” per i file system, ecc.

Le definizioni delle unit sono reperibili in libreria di riferimento [/lib/systemd/system/](#), in file forniti dai mantainer [/usr/lib/systemd/system/](#) o in file personalizzati [/etc/systemd/system](#).

→ include un componente chiamato **journald**, che raccoglie i log di sistema e di servizio. Questo centralizza e migliora la gestione dei log rispetto ai sistemi tradizionali basati su *syslog*.

**systemctl** è il principale strumento a riga di comando per interagire con *systemd*. Con **systemctl** puoi avviare, fermare, riavviare servizi e ottenere informazioni sullo stato del sistema. Alcuni comandi comuni includono:

Verificare lo stato di un servizio: **systemctl status <nome\_servizio>**

Avviare un servizio: **systemctl start <nome\_servizio>**

Fermare un servizio: **systemctl stop <nome\_servizio>**

Riavviare un servizio: **systemctl restart <nome\_servizio>**

Abilitare un servizio al boot: **systemctl enable <nome\_servizio>**

Disabilitare un servizio al boot: **systemctl disable <nome\_servizio>**

## 6.4 Monitoraggio e logging

Fare “**logging**” serve a lasciare una traccia dettagliata e persistente delle attività dei demoni. Queste permettono di ricavare una diagnostica istantanea per sondare lo stato corrente di elementi come CPU, memoria o disco. I log (diari) sono indispensabili per rilevare attività malevole o sospette.

Tipicamente producono file di testo, senza nessuna garanzia di uniformità di formato (a parte la marcatura temporale). Esempi di sistemi di logging Linux sono **rsyslog**, **syslog-ng** o **journal** di **systemd**.

**rsyslog** è il demone (più moderno rispetto alla vecchia versione **syslog**) responsabile della gestione dei log e che offre registrazione remota avanzata. Un suo messaggio è caratterizzato dalla coppia:

**<facility>.<priority>**

<i>facility</i> rappresenta l'argomento e identifica la sorgente del messaggio: auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, syslog, user, uucp, local0 ... local7	<i>priority</i> la priorità del messaggio: emerg, alert, crit, err, warning, notice, info, debug <i>(ordine decrescente)</i>
--	--

### logger [messaggio]

- inviare un messaggio personalizzato al sistema di logging
- **-p** permette di specificare facility e priority: **logger -p local3.info "ciao"**

Possiamo creare e salvare una certa categoria specifica di log all'interno di un file. I seguenti campi di esempio sono separati da TAB e salvati in un file da aggiungere alla cartella delle configurazioni **/etc/rsyslog.d/**.

tutti i log di local0: **local0.\* /var/log/attivita**  
tutti i log kernel dal livello debug in su: **kern.debug /var/log/attivita**  
tutti i log kernel debug: **kern.=debug /var/log/attivita**

Al termine dell'operazione riavviare: **sudo systemctl restart rsyslog**

Possiamo anche inoltrare i log ad un server remoto. Lato server sarà necessario configurare l'attività come visto sopra, lato client specificheremo l'indirizzo del server:

**local0.\* @192.168.56.203**

## 6.5 Gestione servizi

### at

Attraverso l'uso di **at** possiamo **programmare un servizio** da svolgere in un momento futuro. Per cancellare o verificare lo stato del job può essere utile **estrarne l'ID**.

Mettiamo caso di voler programmare un comando tra 30 minuti:

```
echo "/bin/comando" | at now + 30 minutes 2>&1 | grep '^job' | cut -f2 -d' ' > $ATJOB
```

Tramite **grep** filtriamo l'output del comando **at** cercando le righe con la parola 'job'; successivamente usando **cut** estraiamo il secondo argomento, ovvero, l'ID. Tramite **>** salviamo l'ID nella variabile **ATJOB** (creata precedentemente: **ATJOB=\$(mktemp)**).

Possiamo ora usare questa variabile per gestire il job, ad esempio eliminandolo:

```
atrm $(cat $ATJOB)
```

Possiamo implementare anche un **watchdog temporale per imporre un tempo limite** ad un comando.

Ipotizziamo di lanciare un comando che duri molto in background (**&**) e ne salviamo il PID:  
**comando\_lungo & PID=\$!**

Creiamo una variabile per salvare WD per salvare l'ID del job at come visto prima:

```
WD=$(mktemp)  
echo "/bin/kill $PID" | at now + $LIMITE minutes 2>&1 | grep '^job' |  
cut -f2 -d' ' > $WD
```

Verrà eseguito il comando **kill** sul processo \$PID dopo il \$LIMITE temporale prestabilito. Se il comando termina prima, possiamo disinnescarlo con WD.

### syslog

Per aggiungere facility di log possiamo usare la directory /etc/rsyslog.d che contiene file di configurazione automaticamente inclusi.

Aggiungere un nuovo file:

```
echo "local4.=info  /var/log/myevents.log" > /etc/rsyslog.d/mylog.conf
```

Disattivazione della facility:

```
mv /etc/rsyslog.d/mylog.conf /etc/rsyslog.d/mylog.off
```

Ad ogni modifica: **systemctl restart rsyslog**

# 7 Vagrant, ssh e Ansible

Visto che il corso tratta di amministrazione di sistemi, è necessario ora introdurre **Vagrant**.

**Vagrant** è uno strumento open-source progettato per creare e gestire ambienti virtualizzati in modo facile e ripetibile. Viene utilizzato principalmente per configurare macchine virtuali, automatizzando il processo di configurazione dell'ambiente di sviluppo, eliminando i problemi di dipendenze e configurazioni diverse tra macchine.

Tutto l'ambiente è configurato attraverso un singolo file chiamato **Vagrantfile**, che descrive il sistema operativo, le risorse (come CPU e RAM), le dipendenze e i servizi da installare. Si può anche automatizzare l'esecuzione di script di provisioning per installare pacchetti e configurare l'ambiente.

*Esempio di Vagrantfile:*

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "debian/bookworm64"

  config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.network "public_network"
  config.vm.network "forwarded_port", guest: 80, host: 8080
  config.vm.synced_folder "code/", "/app/code"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = 2048
    vb.cpus = 1
    vb.linked_clone = true
  end

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y apache2
    service apache2 start
  SHELL
end
```

Annotations pointing to specific parts of the Vagrantfile:

- box di base
- vari tipi di interfacce di rete virtuali
- mappatura cartella host-guest
- provider e configurazione hw virtuale
- provisioner: azioni di configurazione da eseguire dentro la VM

Per configurare una VM con Vagrant sono necessari i seguenti passaggi:

1) Installare Vagrant e VirtualBox (o un provider compatibile)

2) Creare una cartella di progetto

```
mkdir vagrant-proj
cd vagrant-proj
```

### 3) Creare il Vagrantfile

all'interno della cartella creare un file chiamato 'Vagrantfile' manualmente o tramite comando **vagrant init**

### 4) Configurare il Vagrantfile

qui è possibile specificare la box del sistema (immagine base del sistema operativo) e altri parametri come nell'immagine sopra riportata

Ecco un esempio:

Creare una VM chiamata "server" con IP statico 10.1.1.2 su una rete interna

```
Vagrant.configure("2") do |config|
  # Scegli la box da utilizzare
  config.vm.box = "debian/bookworm64"
  # diminuzione spazio occupato su disco
  config.vm.provider "virtualbox" do |vb|
    vb.linked_clone = true

  # Configura la rete interna con IP statico
  config.vm.network "private_network", ip: "10.1.1.2"

  # Configura il nome della VM
  config.vm.define "server" do |server|
    server.vm.hostname = "server"
  end

  # Configura altre opzioni se necessario
  # Ad esempio, puoi configurare la memoria, CPU, ecc.
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 1
  end
end

In questo esempio:
- `config.vm.box` specifica l'immagine base della VM.
- `config.vm.network "private_network", ip: "10.1.1.2"` configura una rete interna con l'IP statico 10.1.1.2.
- `config.vm.define "server"` imposta il nome della VM come "server".
- La configurazione per VirtualBox è facoltativa, ma consente di specificare la memoria e i CPU per la VM.
```

### 5) Comandi Vagrant

Per avviare la VM: **vagrant up**

Per connettersi: **vagrant ssh**

Per arrestare la VM: **vagrant halt**

Per distruggere la VM: **vagrant destroy**

All'interno della VM possiamo verificare la configurazione della rete tramite i comandi

**ip a** o **ipconfig**

## 7.1 Secure Shell

SSH (Secure Shell) è un protocollo di rete crittografico utilizzato per consentire connessioni sicure tra due dispositivi su una rete, spesso per l'accesso remoto a server, dispositivi di rete o macchine virtuali. È comunemente usato per accedere a shell remote e permette di eseguire comandi e operazioni su un altro computer in modo sicuro, proteggendo i dati trasmessi tramite crittografia. SSH utilizza una combinazione di crittografia simmetrica, crittografia asimmetrica e algoritmi di hashing per proteggere le comunicazioni. Il processo di connessione SSH avviene in tre fasi principali: negoziazione connessione, autenticazione, trasmissione dati.

```
ssh [ale@192.168.1.100]
```

→ per accedere ad un servizio è solitamente necessario specificare l'utente con cui si vuole accedere e il server a cui si vuole accedere

→ è possibile inviare anche singoli comandi: **ssh ale@192.168.1.100 ls**

Configurare l'accesso tramite chiave pubblica permette di effettuare il login senza digitare password. Sulla macchina HOST generiamo una coppia (pubblica + privata) e inviamo una copia della chiave pubblica alla macchina remota.

```
ssh-keygen -t rsa -b 2048
```

# premere invio a ogni domanda

```
scp .ssh/id_rsa.pub ale@192.168.1.100
```

# invio alla macchina remota 192.168.1.100

Per eseguire correttamente il login assicurarsi che nella cartella .ssh sia presente il file id\_rsa generato automaticamente. Se è necessario specificare una chiave diversa da quella di default è possibile usare -i.

La MACCHINA REMOTA crea la cartella .ssh con i giusti permessi e aggiunge alla coda delle chiavi la chiave passata:

```
mkdir .ssh  
chmod 700 .ssh  
cat id_rsa.pub >> .ssh/authorized_keys
```

Osservazione: lanciando nella stessa dir del Vagrantfile **vagrant ssh-config** otteniamo vari parametri della VM:

HostName 127.0.0.1

User vagrant

Port 2222

IdentityFile /home/.../private\_key

Allora **vagrant ssh** equivale a:

**ssh -p 2222 -i /home/.../private\_key vagrant@127.0.0.1**

## 7.2 Ansible

**Ansible** è uno strumento open-source di automazione IT che consente di gestire, configurare e automatizzare i sistemi IT, compresi server, dispositivi di rete e altre infrastrutture. È ampiamente utilizzato per il provisioning di server, la gestione della configurazione, il deployment di applicazioni e molto altro.

Ansible non richiede l'installazione di agenti sui nodi gestiti (i server su cui operare). Comunica con i nodi tramite SSH (per Linux) o WinRM (per Windows). Usa un approccio dichiarativo. Questo significa che si descrive lo stato desiderato del sistema (es. "Installa il pacchetto NGINX"), e Ansible si occupa di far sì che quel sistema raggiunga quello stato.

La configurazione è scritta in YAML, un formato leggibile e semplice. Non richiede conoscenze avanzate di programmazione.

Vediamo ora la struttura e i vari elementi chiave di Ansible:

### 1) Architettura

**Control Node:** Nodo in cui si eseguono i comandi Ansible, solitamente il proprio computer o un server centrale.

**Managed Nodes:** I sistemi che vengono gestiti da Ansible.

### 2) Inventory

L'inventario è un file che elenca i nodi gestiti. Può essere anche un semplice file di testo che elenca gli IP o nomi host dei server.

```
[web]
```

```
192.168.1.10
```

### 3) Playbook

I Playbook sono il cuore di Ansible. Sono file YAML che contengono una serie di compiti (tasks) da eseguire sui nodi gestiti. Ogni task rappresenta un'operazione, come installare un pacchetto o copiare un file.

### 4) Roles

Le Roles sono una struttura più complessa e modulare per organizzare i Playbook. Consentono di riutilizzare il codice, raggruppando i compiti, i file di configurazione e le variabili necessarie per realizzare un determinato scopo.

### 5) Vars

Le variabili vengono utilizzate in Ansible per memorizzare valori dinamici che possono essere riutilizzati nei playbook, come configurazioni specifiche del sistema o del servizio.

### 6) Templates

Ansible supporta l'uso di template scritti in Jinja2, un potente motore di template Python.

Per inizializzare un progetto, sarà opportuno creare una cartella (`mkdir ansible-proj`), creare un inventory per definire gli host (`touch inventory`) e creare il playbook (`touch playbook.yml`).

Per eseguire un playbook: **ansible-playbook playbook.yml**

Specificare gli host: **ansible-playbook -i playbook.yml** (dove host è il file inventory)

Tramite l'opzione **-l** è possibile specificare un solo host all'interno del file.

## 7.3 Playbook

Un **playbook** in Ansible è un file scritto in **YAML** (Yet Another Markup Language) che definisce una serie di **play** (azioni) da eseguire su gruppi di host o server. All'interno dei play, ci sono delle **task**, ognuna delle quali esegue un'azione specifica. Ogni play specifica gli host su cui eseguire le operazioni e una lista di task da eseguire.

La scrittura YAML è molto sensibile alla posizione di tutti i caratteri, per validarne la scrittura:

**ansible-lint -vvv playbook.yml**

Vediamo ora un esempio per capire il funzionamento:

```
---
```

```
- name: Installazione di pacchetti
  hosts: web
  become: true
  tasks:
    - name: Installare nginx
      apt:
        name: nginx
        state: present
    - name: Avviare il servizio nginx
      service:
        name: nginx
        state: started
```

**hosts:** specifica su quali host eseguire il playbook

**become:** indica che il playbook va eseguito con privilegi elevati

**tasks:** elenca i task da eseguire sugli host

**Moduli:**  
Ogni task utilizza un modulo seguito da vari parametri che lo configurano (come apt, copy, service, ...).

Da notare l'indentazione e l'uso dei trattini.

I **moduli** forniti sono tantissimi e consultabili su documentazione ufficiale:

[https://docs.ansible.com/ansible/latest/collections/index\\_module.html](https://docs.ansible.com/ansible/latest/collections/index_module.html)

Oppure possiamo ottenerli come elenco da terminale: **ansible-doc -l**  
ed ottenere informazioni dettagliate: **ansible-doc <modulo>**

## 7.4 Roles

I **ruoli** (roles) in Ansible sono una potente funzionalità che permette di organizzare e riutilizzare il codice di configurazione in modo modulare e scalabile. Un ruolo è una struttura predefinita di directory e file che raggruppa compiti specifici, variabili, file, modelli, e altre risorse che un ruolo può richiedere. Ogni ruolo è progettato per eseguire una singola funzionalità o configurazione, come l'installazione di un software, la configurazione di un servizio, o la gestione di una risorsa.

La struttura di directory tipica di un ruolo è la seguente:

```
roles/
  nome_del_ruolo/
    defaults/
      main.yml      # Variabili predefinite
    files/
      # File statici che possono essere copiati sui target
    handlers/
      main.yml      # Handlers per gestire gli eventi
    meta/
      main.yml      # Metadati del ruolo, inclusi dipendenze
    tasks/
      main.yml      # Compiti principali da eseguire
    templates/
      # Modelli Jinja2 da utilizzare per generare file
    vars/
      main.yml      # Variabili specifiche del ruolo
```

Per usare un ruolo in un playbook, lo si deve includere sotto la chiave `roles`:

```
- name: Configure web servers
  hosts: webservers
  become: yes
  roles:
    - nome_del_ruolo
```

Per il nome\_del\_ruolo specificato vengono eseguiti i task e gli handler; inoltre sarà possibile definire variabili condivise tra i task e file che potranno essere referenziati col nome relativo. Ovviamente è possibile specificare più ruoli (più cartelle nome\_del\_ruolo).

## 7.5 Integrazione con vagrant

Utilizzeremo l'host Linux come Control Node, e le VM Vagrant come Managed Nodes.

Per integrare Ansible con Vagrant, è sufficiente specificarlo nel **Vagrantfile** come metodo di provisioning, indicando il nome del playbook da eseguire.

```
# NOTA: questa riga (e l'end corrispondente) ci sono già nel Vagrantfile
# a meno che non lo stiate scrivendo a mano da zero, nel qual caso mancano
# dovrete specificare a mano anche config.vm.box (ad es. = "debian/bullseye64")
Vagrant.configure("2") do |config|
#
# Run Ansible from the Vagrant Host
#
  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "playbook.yml"
  end
end
```

Poiché il playbook viene applicato alla VM definita nel Vagrantfile, il file yaml (definito nella stessa directory) in prima battuta può ignorare il concetto di inventory. Dichiareremo che dovrà funzionare per tutte le macchine note (**all**).

```
---
- hosts: all
  become: true
  tasks:
```

ATTENZIONE: il provisioning avviene automaticamente al primo avvio (`vagrant up`), viene memorizzato e NON ripetuto ai successivi lanci.

Possiamo forzarlo all'avvio: **vagrant up --provision**  
o su una VM running: **vagrant provision**

## 7.6 Esempi ed esercizi

**Aggiungere un nuovo utente... impostare una password, disabilitare l'utente per 180 giorni, creare una home dir, creare il gruppo “aleq”**

**tasks:**

```
- name: Aggiungi un nuovo utente
  ansible.builtin.user:
    name: aleq
    comment: Alessandro Quaranta
    create_home: true
    password: pistacchio
    group: aleq
    expires:"{{ (ansible_date_time.epoch | int) + (180 * 24 * 60 * 60) }}"
```

**Nota:** L'opzione expire imposta la data di scadenza per l'account in formato epoch (timestamp UNIX). Usiamo la variabile ansible-date\_time.epoch e sommiamo il numero in secondi corrispondente a 180 giorni.

```
"{{ 'pistacchio' | password_hash('sha512') }}"
```

**Fare in modo che un utente standard possa accedere al Managed Node (VM vagrant) senza bisogno di inserire la password**

1) Creare una chiave ssh per l'utente del Control Node (7.1)

```
cd ~/.ssh
```

```
ssh-keygen -t rsa -b 2048 -f id_rsa_ansible
```

2) Uso il modulo authorized\_key di ansible.posix

```
- name: Carica chiave pubblica SSH nelle chiavi autorizzate
  ansible.posix.authorized_key:
    user: aleq
    state: present
    key: "{{ lookup('file', '~/.ssh/id_rsa_ansible.pub') }}"
```

**Nota:** L'opzione state garantisce che la chiave venga aggiunta qualora non fosse già presente, il plugin lookup serve a leggere il contenuto del file specificato come stringa

### Installare lo script di nome copy.sh che copia tutti i file elencati in save.list

```
- name: Copy script
  ansible.builtin.copy:
    src: /path/myhome/copy.sh
    dest: /home/vagrant/copy.sh
    mode: '0755'

- name: Copy Config File
  ansible.builtin.copy:
    src: /path/myhome/save.list
    dest: /home/vagrant/save.list
    mode: '0644'
```

Per il primo task, salvare in /usr/bin lo script e renderlo di proprietà ed eseguibile solo dal root

```
- name: Copy script root
  ansible.builtin.copy:
    src: /path/myhome/copy.sh
    dest: /usr/bin/copy.sh
    owner: root
    group: root
    mode: '0700'
```

**Nota:** L'owner dello script diventa root (e anche il gruppo proprietario)

### Modifica di un file: assicurare che sia presente la riga /etc/passwd in save.list

```
- name: Assicurare la presenza di una linea
  ansible.builtin.lineinfile:
    path: /etc/save.list
    line: /etc/passwd
```

### **eliminare le righe che iniziano per /etc/passwd o /etc/shadow**

```
- name: Eliminare le righe con regexp
  ansible.builtin.lineinfile:
    path: /etc/save.list
    regexp: '^/etc/(passwd|shadow)'
    state: absent
```

### **aggiungere l'utente johnd nei sudoers**

```
- name: Aggiungi a sudoers
  ansible.builtin.lineinfile:
    path: /etc/sudoers.d/johnd
    line: 'johnd ALL=(ALL) NOPASSWD:ALL'
    create: yes
    owner: root
    group: root
    mode: '0440'
    validate: 'visudo -cf %s'
```

### **Pianificazione periodica di uno job con cron: eseguire come root uno script ogni 2 minuti nei giorni feriali escluso agosto**

```
- name: Uso di cron per eseguire uno script
  ansible.builtin.cron:
    name: "Backup dei file in lista"
    minute: "*/2"
    weekday: "1-5"
    month: "1-7, 9-12"
    job: "/usr/bin/copy.sh 2>/dev/null >/dev/null"
```

### Garantire che il sistema di logging sia attivato

```
- name: Garantire che rsyslog sia attivo
  ansible.builtin.systemd:
    state: started
    name: rsyslog.service
```

### **su <username>**

→ cambia utente senza eseguire nuovo login  
→ per caricare l'ambiente come se avesse fatto login: **su - <username>**

### **sudo -i -u <username>**

→ se si dispone di permessi root si può cambiare utente senza inserire la password

## 7.7 Handlers

Gli **handler** sono un modo per far eseguire un task in funzione dell'esecuzione di un altro task. Due elementi hanno un ruolo fondamentale, il costrutto “**handlers**” e il suo nome. Un handler può essere notificato da uno o più task, tramite **notify**, ma viene eseguito sempre una sola volta.

Ad esempio, vogliamo riavviare un demone se il file di configurazione viene modificato:

```
tasks:  
  - name: Scrivi config file  
    ansible.builtin.template:  
      src: /srv/httpd.j2  
      dest: /etc/httpd.conf  
    notify:  
      - Restart apache  
  
handlers:  
  - name: Restart apache  
    ansible.builtin.service:  
      name: httpd  
      state: restarted
```

# 8 Networking

La componente elementare delle reti internet è la network IP, una sorta di isola che funge da nodo terminale. Ogni isola è connessa da calcolatori specializzati detti router o gateway.

In un computer l'interfaccia di rete (livello fisico) è un vero e proprio dispositivo elettronico, tipicamente identificato dal MAC address. In una VM, invece, l'interfaccia di rete è un artefatto gestito dall'hypervisor. Il sistema operativo guest la percepisce come un dispositivo, in quanto l'hypervisor può impostare il MAC, definire a quale segmento remoto è connessa, gestire l'indirizzamento logico.

VirtualBox permette di attivare molteplici interfacce per ogni VM. Ognuna di queste può essere connessa in una modalità specifica; le principali sono: NAT, Bridged, Host-only, Internal. A default è attiva una sola interfaccia NAT.

**In vagrant:**

**Bridged:** si comportano come se fossero connesse all'interfaccia dell'host attraverso un bridge, quindi come se fossero attestate sulla stessa LAN dell'host.

```
config.vm.network "public_network"
```

è possibile specificare un ip statico per la VM:

```
config.vm.network "public_network", ip: "192.168.0.7"
```

**Host-only:** configurato per usare una specifica subnet IP, genera un interfaccia virtuale sull'host e le assegna un IP della subnet. Connnette l'interfaccia della VM alla corrispondente LAN virtuale in modo che possa comunicare solo con l'host.

```
config.vm.network "private_network"
```

(vale opzione ip)

**Internal:** le interfacce sono assegnate a una LAN totalmente virtuale, solo le interfacce appartenenti alla stessa internal network possono comunicare tra loro.

```
config.vm.network "private_network", virtualbox_intnet: "LAN1"
```

(vale opzione ip)

Per far parlare tra loro le isole è necessario sfruttare la tecnologia IP, che si pone l'obiettivo di far dialogare network IP a prescindere dalla loro implementazione e dalla loro locazione.

Ogni nodo di internet ha una base dati di destinazioni possibili; quando deve essere inviato un datagramma la tecnologia della propria network può essere utilizzata per raggiungere la destinazione finale o per raggiungere il primo ponte.

Definiamo “routing” la scelta del percorso su cui si inviano i dati.

Parliamo di “Direct delivery” se IP sorgente e IP destinatario sono sulla stessa rete fisica, “Indirect delivery” se la sorgente invia il pacchetto ad un router intermedio.

La prospettiva di un esaurimento degli indirizzi IP ha fatto esplodere l'utilizzo di una tecnica che consente di utilizzare un solo indirizzo pubblico per un'intera rete privata, NAT.

SNAT (Source Network Address Translation) e DNAT (Destination Network Address Translation) sono due tipi di NAT utilizzati per modificare gli indirizzi IP nei pacchetti di rete.

SNAT: Solitamente utilizzato quando i dispositivi in una rete privata (come una LAN) devono comunicare con l'esterno (ad esempio, Internet). Cambia l'indirizzo sorgente.

DNAT: Utilizzato per reindirizzare il traffico in arrivo verso un server interno o un servizio specifico. Cambia l'indirizzo destinazione.

## 8.1 Configurazione di rete

In Linux, la configurazione di rete può avvenire manualmente andando a modificare il file `/etc/network/interfaces` o a runtime tramite i seguenti comandi.

### ip a

- mostra una lista di tutte le interfacce di rete del sistema:  
viene riportato il nome dell'interfaccia di rete,  
link/ether specifica l'indirizzo MAC,  
inet indica l'indirizzo IPV4
- per aggiungere un indirizzo IP: **ip a add <ip>/<subnet\_mask> dev <interface>**  
si specifica indirizzo ip e subnet mask che vogliamo assegnare,  
e l'interfaccia di rete a cui vogliamo assegnare l'ip
- per rimuovere un indirizzo IP: **ip a del <ip>/<subnet\_mask> dev <interface>**

**ip r**

→ mostra le rotte dei router che specificano come i pacchetti verranno inviati

**ping <ip>**

→ verifica della connettività di un indirizzo

**traceroute <ip>**

→ verifica del percorso di un pacchetto

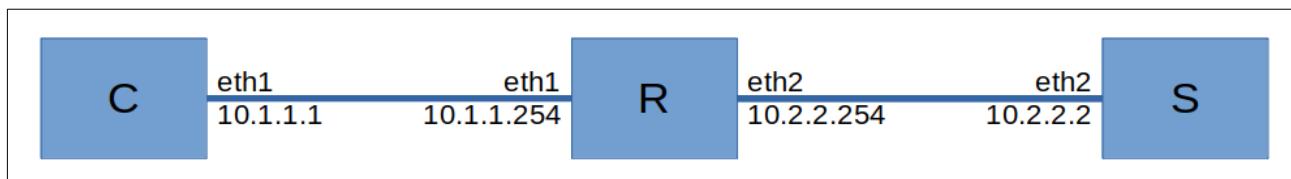
**ss**

→ verifica dello stato delle connessioni

## 8.2 Configurazione di una rete di VM

Gli esercizi che seguono richiedono la generazione di molteplici VM. Vediamo come fare a costruire una propria rete di VM.

Ad esempio, configuriamo staticamente Client Router e Server per ottenere questo layout (oltre alle connessioni di default a internet via eth0) facendo in modo che Client (10.1.1.1) e Server (10.2.2.2) possano scambiare traffico attraverso Router.



Creiamo una rete interna privata, **Internal**, in modo tale che le macchine appartenenti alla stessa rete possano comunicare tra loro.

## **Opzione 1) Lavorare su 3 Vagrantfile differenti**

### Client

```
Vagrant.configure("2") do |config|
  config.vm.provider "virtualbox" do |vb|
    vb.linked_clone = true
  end

  # Configurazione Client
  config.vm.define "client" do |client|
    client.vm.box = "nome_box_client"
    client.vm.network "private_network", virtualbox_intnet: "LAN1", auto_config: false
  end
end
```

Predisposta la VM, possiamo configurare a mano l'**interfaccia** da dentro la VM tramite bash:

**sudo ip a add 10.1.1.1/24 dev eth1**

**sudo ip link set dev eth1 up**

Per rendere persistente la configurazione, andiamo a modificare il file **/etc/network/interfaces**:

```
auto eth1
iface eth1 inet static
  address 10.1.1.1
  netmask 255.255.255.0
  up /usr/sbin/ip route add 10.2.2.0/24 via 10.1.1.254
```

Bisogna inoltre modificare **eth0 allow-hotplug** con **auto**.

Dopo ogni modifica di questo file eseguire il comando: **systemctl restart networking**.

### Server

I passaggi sono gli stessi del Client, il Vagrantfile sarà modificato con:

**client.vm.network "private\_network", virtualbox\_intnet: "LAN2", auto\_config: false**

il file **/etc/network/interfaces**:

```
auto eth1
iface eth1 inet static
  address 10.2.2.2
  netmask 255.255.255.0
  up /usr/sbin/ip route add 10.1.1.0/24 via 10.2.2.254
```

**NOTA: up... serve ad aggiungere una route statica alla rete (verso il sistema) add... passando per il gateway via...**

## **Router**

```
Vagrant.configure("2") do |config|
  config.vm.provider "virtualbox" do |vb|
    vb.linked_clone = true
  end

  # Configurazione Router
  config.vm.define "router" do |router|
    router.vm.box = "nome_box_router"

    # Configura LAN1 per eth1 senza auto-configurazione
    router.vm.network "private_network", virtualbox_intnet: "LAN1", auto_config: false

    # Configura LAN2 per eth2 senza auto-configurazione
    router.vm.network "private_network", virtualbox_intnet: "LAN2", auto_config: false
  end
end
```

il file [\*\*/etc/network/interfaces\*\*](#):

```
# Configurazione statica per eth1 (LAN1)
auto eth1
iface eth1 inet static
  address 10.1.1.254
  netmask 255.255.255.0

# Configurazione statica per eth2 (LAN2)
auto eth2
iface eth2 inet static
  address 10.2.2.254
  netmask 255.255.255.0
```

Dobbiamo ora abilitare il forwarding IP; per farlo decommentiamo la seguente linea del file [\*\*/etc/sysctl.conf\*\*](#): **net.ipv4.ip\_forward=1**

Applichiamo le modifiche: **sysctl -p**

## **Opzione 2) Lavorare su un unico file Vagrant**

```
Vagrant.configure("2") do |config|
  config.vm.provider "virtualbox" do |vb|
    vb.linked_clone = true
  end

  # Configurazione Client
  config.vm.define "client" do |client|
    client.vm.box = "nome_box_client"
    client.vm.network "private_network", virtualbox_intnet: "LAN1", auto_config: false
  end

  # Configurazione Router
  config.vm.define "router" do |router|
    router.vm.box = "nome_box_router"
    router.vm.network "private_network", virtualbox_intnet: "LAN1", auto_config: false
    router.vm.network "private_network", virtualbox_intnet: "LAN2", auto_config: false
  end

  # Configurazione Server
  config.vm.define "server" do |server|
    server.vm.box = "nome_box_server"
    client.vm.network "private_network", virtualbox_intnet: "LAN2", auto_config: false
  end
end
```

## 8.3 Configurazione della rete tramite Ansible

Per modificare il file [/etc/network/interfaces](#) è possibile usare Ansible.

Ipotiziamo di voler modificare il file sostituendo la direttiva per eth0 **allow-hotplug** con **auto**.

Vengono ora proposti due metodi differenti.

### 1) **community.general.interfaces\_file**

Il seguente modulo permette di modificare opzioni in un file di configurazione di rete.

Aggiungeremo una condizione (when) che ci assicura che il sistema operativo sia basato su Debian. Un handler si occuperà di riavviare il servizio di networking.

```
- name: Update eth0 from allow-hotplug to auto
  community.general.interfaces_file:
    iface: eth0
    option: 'auto'
    state: present
    backup: yes
  when: ansible_facts['os_family'] in ['Debian']
  notify: Restart Networking
```

(backup conserva la configurazione precedente)

### 2) **ansible.builtin.copy**

In alternativa possiamo creare un file già formattato con le impostazioni di rete volute.

Possiamo dunque copiarlo nel percorso [/etc/network/interfaces](#) o nella dir

[/etc/network/interfaces.d/](#) (che permette di suddividere la configurazione su più file).

Copiamo il file di configurazione per l'interfaccia eth3.

```
- name: Copy a new network configuration "interfaces" file into
  place, after passing validation with ifup
  ansible.builtin.copy:
    src: /path/to/your/interfaces_file
    dest: /etc/network/interfaces.d/eth3
    validate: /usr/sbin/ifup --no-act -i %s eth3
  notify: Restart Networking
```

## HANDLER

```
handlers:  
  - name: Restart Networking  
    ansible.builtin.service:  
      name: networking  
      state: restarted
```

NOTA: Specificare l'uso di un playbook nel Vagrantfile!

## 8.4 Esercizio riassuntivo

Proviamo ad unire le nozioni appena apprese. Partendo dalle 3 VM tirate su nel capitolo 8.2 (Client, Server e Router), attiviamo un nuova interfaccia interna collocandole tutte e tre su una nuova LAN3.

Notiamo che il nome dell'interfaccia cambierà a seconda di quante sono già attive: se su Client e Server sono presenti eth0 (navigazione) e eth1 (inserita per la configurazione client-router-server) la nuova interfaccia sarà eth2, mentre sul Router che usa due interfacce per l'esercizio già svolto sarà eth3.

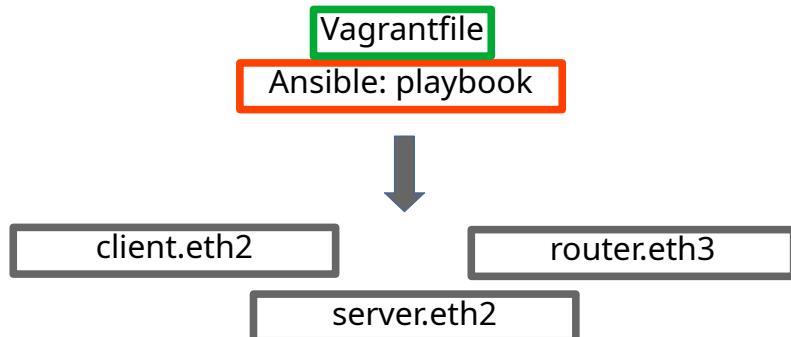
Realizzare il provisioning in modo che le VM assumano automaticamente su tale interfaccia questi indirizzi:

Client: 192.168.56.201

Router: 192.168.56.202

Server: 192.168.56.203

Per la configurazione useremo un unico playbook!



*Vagrantfile*

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/bookworm64"
  config.vm.provider "virtualbox" do |vb|
    vb.linked_clone = true
  end

  config.vm.define "client" do |machine|
    machine.vm.hostname = "client"
    machine.vm.network "private_network", virtualbox_intnet: "LAN1", auto_config: false
    machine.vm.network "private_network", virtualbox_intnet: "LAN3", auto_config: false
    machine.vm.provision "ansible" do |ansible|
      ansible.playbook = "playbook1.yml"
    end
  end

  config.vm.define "server" do |machine|
    machine.vm.hostname = "server"
    machine.vm.network "private_network", virtualbox_intnet: "LAN2", auto_config: false
    machine.vm.network "private_network", virtualbox_intnet: "LAN3", auto_config: false
    machine.vm.provision "ansible" do |ansible|
      ansible.playbook = "playbook1.yml"
    end
  end

  config.vm.define "router" do |machine|
    machine.vm.hostname = "router"
    machine.vm.network "private_network", virtualbox_intnet: "LAN1", auto_config: false
    machine.vm.network "private_network", virtualbox_intnet: "LAN2", auto_config: false
    machine.vm.network "private_network", virtualbox_intnet: "LAN3", auto_config: false
    machine.vm.provision "ansible" do |ansible|
      ansible.playbook = "playbook1.yml"
    end
  end
end
```

*playbook1*

```
---
```

- hosts: client  
become: true  
**tasks:**
  - name: configure host network  
ansible.builtin.copy:
    - src: "client.eth2"
    - dest: "/etc/network/interfaces.d/eth2"
    - mode: '0644'
  - notify: Restart networking
  
**handlers:**

  - name: Restart networking  
ansible.builtin.service:
    - name: networking
    - state: restarted

  
- hosts: router  
become: true  
**tasks:**
  - name: configure host network  
ansible.builtin.copy:
    - src: "router.eth3"
    - dest: "/etc/network/interfaces.d/eth3"
    - mode: '0644'
  - notify: Restart networking
  
**handlers:**

  - name: Restart networking  
ansible.builtin.service:
    - name: networking
    - state: restarted

  
- hosts: server  
become: true  
**tasks:**
  - name: configure host network  
ansible.builtin.copy:
    - src: "server.eth2"
    - dest: "/etc/network/interfaces.d/eth2"
    - mode: '0644'

```
    notify: Restart networking
```

```
handlers:
```

- name: Restart networking
- ```
ansible.builtin.service:  
    name: networking  
    state: restarted
```

*client.eth2*

```
auto eth2  
iface eth2 inet static  
    address 192.168.56.201  
    netmask 255.255.255.0
```

*server.eth2*

```
auto eth2  
iface eth2 inet static  
    address 192.168.56.203  
    netmask 255.255.255.0
```

*router.eth3*

```
auto eth2  
iface eth2 inet static  
    address 192.168.56.202  
    netmask 255.255.255.0
```

## 8.5 DHCP

DHCP (Dynamic Host Configuration Protocol) è un protocollo di rete utilizzato per assegnare automaticamente indirizzi IP ed altre configurazioni di rete ai dispositivi che si connettono ad una rete. Il suo scopo principale è facilitare la gestione degli indirizzi IP, eliminando la necessità di configurare manualmente ogni dispositivo che si collega ad una rete.

Possiamo abilitare nel Router il servizio DHCP permettendo al Client di ricevere automaticamente un indirizzo IP andando a modificare il file **/etc/dnsmasq.conf**:

```
interface=eth1
dhcp-range=10.1.1.10,10.1.1.20,12h
dhcp-option=3
dhcp-option=option:ntp-server,10.1.1.254
dhcp-option=option:dns-server,10.1.1.254
dhcp-option=121,10.2.2.0/24,10.1.1.254
```

- Configuriamo la dnsmasq per rispondere alle richieste in arrivo da eth1.
- Definisce l'intervallo di indirizzi IP (10.1.1.10 - 10.1.1.20) che il server DHCP può assegnare ai client, con validità di 12 ore.
- Lasciamo al client l'accesso a internet predefinito fornito da VirtualBox (dhcp-option=3)
- Definiamo inoltre una route statica (121) per passare dal router (10.1.1.254) per raggiungere il server (10.2.2.0/24)

Per configurare il Client bisogna fare in modo che riceva l'IP tramite l'interfaccia eth1.

Aggiorniamo il file **/etc/network/interfaces** sostituendo la configurazione di eth1 con:

```
auto eth1
iface eth1 inet dhcp
```

(N.B il lease del DHCP lato Router è stato impostato ogni 12h, qualora voleste fare delle prove continue per forzare il DHCP dal client ed ottenere un nuovo indirizzo è possibile tramite privilegi di amministratore usare il comando:

```
sudo dhclient -v
```

# 9 Monitoraggio centralizzato

Ogni apparato managed dispone di strumenti proprietari per configurazione e monitoraggio.

**SNMP** (Simple Network Management Protocol) è un protocollo standard utilizzato per gestire e monitorare dispositivi di rete come router, switch, server, stampanti, dispositivi IoT, e altro ancora. È progettato per facilitare la raccolta e la gestione delle informazioni di stato e delle prestazioni dei dispositivi di rete in modo centralizzato. Standardizza il modello dei dati e il modello di interazione.

I tre componenti SNMP sono i seguenti:

- **Managed Devices**

*Sono i dispositivi che vengono monitorati e gestiti da SNMP. Ogni dispositivo gestito deve avere un agente SNMP installato.*

- **SNMP Agent**

*È un software che gira sul dispositivo gestito. L'agente raccoglie le informazioni sullo stato e le prestazioni del dispositivo e le memorizza in una struttura dati chiamata MIB.*

- **SNMP Manager**

*È il sistema che invia richieste agli agenti SNMP per raccogliere informazioni o impostare parametri di configurazione.*

Esistono tre principali versioni di SNMP, ognuna con caratteristiche e livelli di sicurezza differenti:

## 1. **SNMPv1:**

- La prima versione del protocollo, molto semplice ma con limitate funzionalità di sicurezza.
- Comunicazione in chiaro, senza autenticazione.

## 2. **SNMPv2:**

- Versione migliorata di SNMPv1, con un aumento delle funzionalità, ma con la stessa limitata sicurezza (usa una stringa di comunità come password).
- Miglioramenti nella gestione degli errori e nelle prestazioni rispetto a SNMPv1.

## 3. **SNMPv3:**

- La versione più sicura e avanzata del protocollo, introduce autenticazione, crittografia e controllo degli accessi. È considerata la versione preferita per ambienti di produzione moderni.
- **Supporta:**
- Autenticazione: Verifica dell'identità tramite hash come MD5 o SHA.
- Crittografia: Protezione dei dati trasmessi tramite algoritmi come DES o AES.
- Controllo di Accesso: Autorizzazioni specifiche per utenti o gruppi.

## 9.1 Modello dei dati e operazioni

Elementi chiave del modello dei dati sono gli **OID** (Object Identifiers) e **MIB** (Management Information Base).

Gli **OID** rappresentano singoli oggetti gestiti in un dispositivo di rete, identificati tramite una struttura gerarchica simile a un albero. Ogni OID è un percorso numerico unico nella struttura, composto da una sequenza di numeri separati da punti.

Esempi di OID:

- `1.3.6.1.2.1.1.1` può rappresentare il nome di un dispositivo.
- `1.3.6.1.2.1.2.2.1.10` può rappresentare i byte ricevuti su un'interfaccia di rete.

La **MIB** è una sorta di database che definisce e organizza gli oggetti gestibili (le variabili) tramite SNMP. In sostanza si presenta come un catalogo che associa ad ogni oggetto un OID, una sintassi (tipo di dato) e una codifica (descrizione della rappresentazione materiale).

Le **operazioni** che un manager SNMP può eseguire sono principalmente:

**Get:**

Il manager invia una richiesta per ottenere il valore di una determinata variabile MIB da un agente.

**GetNext:**

Il manager richiede il valore della variabile successiva nella gerarchia MIB. È utile per iterare su una tabella di valori.

**Set:**

Il manager invia una richiesta per modificare il valore di una variabile MIB su un dispositivo. Ad esempio, può essere usato per configurare un dispositivo di rete.

**GetBulk:**

Una versione più efficiente di `GetNext`, usata per recuperare grandi quantità di dati in un'unica richiesta.

**Trap:**

L'agente invia una notifica non richiesta al manager SNMP quando avviene un evento specifico (es. un'interfaccia di rete va offline). Le trappole SNMP sono utili per il monitoraggio in tempo reale degli eventi critici.

**Inform:**

Simile a `Trap`, ma richiede una conferma di ricezione dal manager SNMP, garantendo che l'informazione sia stata ricevuta.

## 9.2 Configurazione SNMP

Tramite apt è necessario innanzitutto scaricare i seguenti pacchetti:

- Lato Client (Manager) : *snmp, snmp-mibs-downloader*
- Lato Server (Agent): *snmpd*

Osservazione: Per installare i file MIB su sistemi Debian, è necessario abilitare i repository **contrib** e **non-free** poiché i file MIB di SNMP sono distribuiti con una licenza non compatibile con la distribuzione predefinita di Debian. Ecco i passaggi:

1) nel file /etc/apt/sources.list aggiungere '**contrib non-free**' alla fine di ogni linea che finisce con *deb*.

2) installare il pacchetto che scarica i MIB con i comandi

**apt update**

**apt install snmp snmp-mibs-downloader**

3) attivare l'utilizzo editando il file /etc/snmp/snmp.conf commentando la riga '*mibs* :'

**sudo sed -i 's/mibs :/# mibs :/g' /etc/snmp/snmp.conf**

Prima di poter eseguire delle richieste è necessario editare il file /etc/snmp/snmp.conf come descritto sopra (commentando *mibs*:).

L'agent è rappresentato dal demone **snmpd** pacchetto da installare sulla VM Server:

**apt update**

**apt install snmpd**

Eseguire poi i seguenti passaggi sul file /etc/snmp/snmpd.conf:

1) per la socket, sostituire **agentAddress udp:127.0.0.1:161**  
con **agentAddress udp:161**

2) definire una vista per il MIB: **view all included .1** (separare con TAB)

3) inserire le righe per abilitare le community per SNMPv1:

**rocommunity public default -V all**  
**rwcommunity supercom default -V all**

(controllare che non ci siano altre righe che definiscono rocommunity e rwcommunity)

4) Dopo la configurazione riavviare il demone: **systemctl restart snmpd**

## 9.3 Comandi manager

Essenziali in ogni invocazione i parametri: **-v** versione e **-c** community.

### snmpwalk

→ utilizza ricorsivamente getNext per navigare un intero sottoalbero del MIB, utile per visualizzare gli oggetti di un server

*Esempio:* Visualizza tutti gli oggetti del server 10.9.9.1

**snmpwalk -v 1 -c public 10.9.9.1 .1**

Oppure

**snmpwalk -v 1 -c public 10.9.9.1 .1 | less**

*Esempio:* Visualizza tutti gli oggetti sotto .1.3.6.1.2.1.1 del server \$IP\_SERVER

**snmpwalk -On -v 1 -c public \$IP\_SERVER .1.3.6.1.2.1.1**

(-On formatta l'output con notazione numerica OID)

### snmpget

→ recupero di un singolo oggetto

*Esempio:* Visualizza il valore di una specifica entry senza notazione OID

**snmpget -v 1 -c public 10.9.9.1 .1 'UCD-SNMP-MIB:memAvailReal.0'**

(Da in output il valore presente nel file di configurazione)

*Esempio:* Visualizza il valore di una specifica entry con notazione OID

**snmpget -v 1 -c public \$IP\_SERVER .1.3.6.1.2.1.1.4.0**

(Da in output il valore presente nel file di configurazione)

*Esempio:* se il valore non è impostato? Proviamo ad eseguire...

**snmpget -v 1 -c public \$IP\_SERVER .1.3.6.1.2.1.1.6.0**

ma questo valore non è impostato, modifichiamo il valore di un OID come segue

**snmpset -v 1 -c supercom \$IP\_SERVER .1.3.6.1.2.1.1.6.0 s "proprio qui"**

usando la community di scrittura impostiamo la stringa (s) "proprio qui" come valore di sysLocation (rappresentato dall'OID scritto sopra)

Dopo aver avviato tcpdump, eseguire snmpwalk ci permette di osservare in tempo reale come SNMP invii una serie di richieste getNext: **tcpdump -nlp -i lo udp port 161**  
ripetiamo poi **snmpwalk**

## 9.4 Misurare parametri del sistema

Nel file `/etc/snmp/snmpd.conf`, è possibile configurare SNMP per monitorare automaticamente alcuni parametri di sistema tramite le estensioni UCD-SNMP. Queste direttive permettono di controllare variabili come il carico del sistema, lo spazio disponibile su disco e i processi in esecuzione.

### 1) Monitoraggio carico del sistema

`load [max-1][max-5][max-15]`

Indichiamo il carico massimo degli ultimi 1, 5, 15 minuti. Se il carico del sistema supera una delle soglie indicate si attiverà la flag associata. Questa funzionalità viene esposta sotto la tabella `.1.3.6.1.4.1.2021.10`.

Ad esempio impostiamo:

`load 5 4 3`

Otteniamo le tre righe con il carico attuale e le eventuali flag attivate:

`snmpwalk -v 1 -c public $IP_SERVER .1.3.6.1.4.1.2021.10`

### 2) Monitoraggio delle partizioni

`disk [partizione] [minfree|minfree%]`

Indichiamo la partizione e lo spazio minimo in MB o in percentuale sotto cui viene attivata la flag. Esposta sotto la tabella `.1.3.6.1.4.1.2021.9`.

### 3) Monitoraggio dei processi

`proc [nometproc] [maxnum [minnum]]`

Indichiamo il nome del processo specifico, il numero massimo di istanze (eventualmente anche minimo) per il quale il flag verrà attivato. La tabella `.1.3.6.1.4.1.2021.2` espone il numero di istanze attive e se il flag è stato attivato.

ATTENZIONE: ricorda sempre che dopo ogni cambio di configurazione il demone va riavviato (`sudo systemctl restart snmpd`)

## 9.5 Esecuzione codice

In **/etc/snmp/snmpd.conf**, la direttiva **extend-sh** permette di specificare il comando che l'agent SNMP deve eseguire quando riceve una richiesta (una `getRequest`).

La sintassi è la seguente:

**extend-sh <etichetta> <comando>**

L'etichetta identifica il comando SNMP.

Ad esempio: **extend-sh test echo Ciao**

L'output del comando viene esposto sulla tabella con OID:

**NET-SNMP-EXTEND-MIB::nsExtendOutputFull."etichetta"**

Per l'esempio precedente: **NET-SNMP-EXTEND-MIB::nsExtendOutputFull."test"**

Quando un client invia una richiesta ad esempio usando `snmpget` all'OID descritto sopra, viene eseguito il comando e il risultato diventa il valore associato all'OID.

```
snmpget -v 1 -c public $IP_SERVER NET-SNMP-EXTEND-MIB::nsExtendOutputFull."test"
```

---

L'agent gira con identità standard, nelle VM del corso [Debian-snmp](#). Se vogliamo eseguire un comando con privilegi di root dobbiamo ricorrere a sudo.

Modifichiamo allora il file sudoers usando il comando **visudo**.

Ad esempio per far eseguire a Debian-snmp il comando `ss -lntp` senza password:

**Debian-snmp ALL=NOPASSWD:/usr/bin/ss -lntp**

Configuriamo ora **/etc/snmp/snmpd.conf**:

```
extend-sh sshd /usr/bin/sudo /usr/bin/ss -lntp | egrep '0\.0\.0\.0:22.*sshd'
```

(Questa configurazione esegue il comando `ss` con sudo e filtra l'output per mostrare solo le righe che contengono 0.0.0.0:22)

## 9.6 Esercizio riassuntivo

Utilizzare Ansible per creare a configurare un ambiente multi-machine con Vagrant. La topologia della rete prevede l'utilizzo di due macchine virtuali usate come Agent SNMP ed una macchina da dove poter controllare i due nodi (denominata da adesso in poi Controller), quindi in tutto è necessario creare nel Vagrantfile 3 macchine virtuali. Impostare correttamente gli hostname delle machine virtuali tramite o Vagrantfile oppure tramite DNSMASQ, rispettivamente **agent1**, **agent2** e **controller**.

La configurazione di rete deve avvenire tramite DHCP erogato da DNSMASQ sul nodo da cui è possibile interrogare i due Agent SNMP, ovvero il Controller.

Dopo aver installato tutti i pacchetti del caso e configurato i due demoni SNMPPD tutto tramite Ansible, è necessario aggiungere ad entrambi i demoni, nel giusto file di configurazione, la possibilità di lanciare due comandi custom quando viene richiesto il corrispondente OID, impostando correttamente i permessi e i meccanismi con cui effettuare l'inalzamento dei privilegi.

Il primo comando in questione (script **createdump.sh**) deve poter generare un PCAP, utilizzando TCPDUMP, che termini automaticamente dopo aver catturato 10.000 pacchetti, sull'interfaccia di rete interna tra la macchina virtuale con l'Agent e il Controller. Utilizzando Rsyslog (configurato tramite Ansible) gli agent segnalano al Controller la disponibilità del PCAP e quindi la possibilità di poterlo recuperare da remoto. Il secondo comando custom (script **deletedump.sh**) invece deve poter cancellare i file PCAP che il Controller ha già recuperato, i cui nomi vengono inviati via syslog dal Controller agli Agent.

Periodicamente, ogni 3 minuti alternativamente su un agent e sull'altro (es: minuto 3 agent1, minuto 6 agent2, minuto 9 agent1, ...), il Controller invocherà uno script (**managedump.sh**) per eseguire il protocollo di generazione, recupero e cancellazione dei PCAP. Lo script deve immediatamente uscire senza fare nulla se ce n'è già un'istanza lanciata in precedenza e non ancora terminata, mentre in caso contrario deve

- avviare la generazione del PCAP usando la richieste del corrispondente oggetto SNMP
- mettersi in attesa che sul log compaia la conferma di generazione avvenuta, che include il nome del file da scaricare
- scaricare col metodo più sicuro che conoscete il file dall'agent in una directory locale
- inviare conferma all'agent loggando il nome del file scaricato
- invocare via SNMP lo script di cancellazione del PCAP

L'esercizio prevede anche lo scambio delle chiavi SSH ovvero: la generazione di una coppia di chiavi nel controller oppure nel host da dove si lancia Ansible. Successivamente la chiave pubblica generata deve essere installata nei due agent; questa parte viene lasciata da completare.

## ATTO I : creare le VM

Tramite Vagrant creiamo le 3 VM con i nomi richiesti; queste saranno sulla stessa rete (LAN1) [8.2].

*Vagrantfile:*

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/bullseye64"

  config.vm.provider "virtualbox" do |vb|
    vb.linked_clone = true
  end

  config.vm.define "controller" do |machine|
    machine.vm.hostname = "controller"
    machine.vm.network "private_network", virtualbox_intnet:
"LAN1", auto_config: false
  end

  config.vm.define "agent1" do |machine|
    machine.vm.hostname = "agent1"
    machine.vm.network "private_network", virtualbox_intnet:
"LAN1", auto_config: false
  end

  config.vm.define "agent2" do |machine|
    machine.vm.hostname = "agent2"
    machine.vm.network "private_network", virtualbox_intnet:
"LAN1", auto_config: false
  end

  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "site.yml"
  end
end
```

## ATTO II : roles e playbook

Il Vagrantfile sarà configurato per usare il playbook site.yml che proporrà a sua volta una configurazione basata su roles [7.4]. Le directory avranno seguente struttura:

```
roles/
  common/
    files/ # File statici che possono essere copiati sui target
    handlers/ main.yml      # Handlers per gestire gli eventi
    tasks/ main.yml        # Compiti principali da eseguire
  controller/
    files/
    handlers/ main.yml
    tasks/ main.yml
  agent/
    files/
    handlers/ main.yml
    tasks/ main.yml
```

### common/

Quali sono le operazioni che dovranno svolgere tutte le VM?

#### tasks/ main.yml

##### 1) Aggiornamento di tutti i pacchetti

```
---
- name: Update all packages
  ansible.builtin.apt:
    update_cache: yes
```

##### 2) Cambiare “allow-hotplug” [8.2]

```
- name: Change allow-hotplug with auto
  ansible.builtin.replace:
    path: '/etc/network/interfaces'
    regexp: '^allow\-\hotplug'
    replace: 'auto'
    validate: /usr/sbin/ifup --no-act -i %s eth0
```

##### 3) Impostare la configurazione di rsyslog che permette **logging remoto**

```
- name: Copy rsyslog configuration enable remote log
  ansible.builtin.copy:
    src: rsyslog.conf
    dest: '/etc/rsyslog.conf'
    owner: 'root'
    group: 'root'
    mode: '0644'
  notify: Restart rsyslog
```

```

handlers/ main.yml
---
- name: Restart rsyslog
  ansible.builtin.service:
    name: rsyslog
    state: restarted

files/
  rsyslog.conf: file di configurazione che permette logging remoto

```

## **controller/**

### **tasks/ main.yml**

1) Configurare la propria **interfaccia di rete** [8.2, 8.3] tramite copy.

```

---
- name: Copy a new network configuration "interfaces" file into
place, after passing validation with ifup
  ansible.builtin.copy:
    src: eth1_cfg
    dest: /etc/network/interfaces.d/eth1_cfg
    validate: /usr/sbin/ifup --no-act -i %s eth1
  notify: Restart Networking

```

2) Configurare il **servizio DHCP** [8.5], dunque, installare dnsmaq, copiare la sua configurazione, attivarlo al boot del sistema.

```

- name: Install Dnsmasq
  ansible.builtin.apt:
    name: dnsmasq
    update_cache: true

- name: Copy Dnsmasq configuration
  ansible.builtin.copy:
    src: dnsmasq.conf
    dest: '/etc/dnsmasq.conf'
    owner: 'root'
    group: 'root'
    mode: '0644'
  notify: Restart Dnsmasq

- name: Enable Dnsmasq at boot
  ansible.builtin.systemd:
    name: dnsmasq.service
    state: started
    enabled: yes

```

3) Configurare il **servizio di log** [6.4], nella dir /etc/rsyslog.d/ possiamo caricare configurazioni aggiuntive a quelle base. Vogliamo che tutti i messaggi per la <facility> local1 e con priorità info (o superiore) siano salvati in dumps.log. L'esercizio chiedeva infatti che uno dei comandi scrivesse su log quando il file PCAP è pronto.

```
- name: Copy syslog configuration
  ansible.builtin.copy:
    src: createdumps.conf
    dest: '/etc/rsyslog.d/'
    owner: 'root'
    group: 'root'
    mode: '0644'
  notify: Restart rsyslog
```

4) **Configurare SNMP su Controller** [9.2]. Dobbiamo installare SNMP, dato che le macchine tirate su sono Debian, seguire i passaggi visti nei capitoli precedenti.

```
- name: Enable Contrib Non-Free
  ansible.builtin.replace:
    path: /etc/apt/sources.list
    regexp: 'main$'
    replace: 'main contrib non-free'

- name: Install SNMP and Mibs
  ansible.builtin.apt:
    name: [ snmp, snmp-mibs-downloader ]
    update_cache: true

- name: Enable MIBS
  ansible.builtin.replace:
    path: /etc/snmp/snmp.conf
    regexp: '^mibs :'
    replace: '# mibs :'
```

5) **Copiare lo script.** Il Controller fa uso dello script managedump.sh. Questo deve essere invocato ogni 3 minuti.

```
- name: Copy managedump.sh script in Controller
  ansible.builtin.copy:
    src: managedump.sh
    dest: '/usr/bin/'
    owner: 'root'
    group: 'root'
    mode: '0740'

- name: Run managedump.sh every 3 min
  ansible.builtin.cron:
    name: "managedump"
    minute: "3"
    job: "/usr/bin/managedump.sh"
```

## handlers/ main.yml

Restart dei servizi: networking, dnsmasq, rsyslog

## files/

*eth1\_cfg*: configurazione interfaccia di rete per il controller

auto eth1

iface eth1 inet static

    address 10.1.1.254

    netmask 255.255.255.0

*dnsmasq.conf*: configurazione servizio DHCP

dhcp-range=interface:eth1,10.1.1.5,10.1.1.10,12h

dhcp-option=3

# inibisce il comportamento di default, che indicherebbe

# a Client di prendere come default gateway Router

# mentre noi vogliamo che resti quello di VirtualBox

#dhcp-host="INSERTYOURCLIENTMACADDRESS",10.1.1.1

#dhcp-host="INSERTYOURSERVERMACADDRESS",10.2.2.2

*createdump.conf*: configurazione per creare il file di log

local1.info /var/log/dumps.log

*managedump.sh*: script

## agent/

### tasks/ main.yml

1) Configurare l'**interfaccia di rete** [8.2, 8.3] tramite copy.

---

```
- name: Copy a new network configuration "interfaces" file into
place, after passing validation with ifup
  ansible.builtin.copy:
    src: eth1_cfg
    dest: /etc/network/interfaces.d/eth1_cfg
    validate: /usr/sbin/ifup --no-act -i %s eth1
  notify: Restart Networking
```

2) **Installare tcpdump e il demone snmpd** [9.2], poi, copiare il suo file di configurazione.

```
- name: Install tcpdump and snmpd
  ansible.builtin.apt:
    name: [ snmpd, tcpdump ]
    update_cache: true

- name: Copy snmp agent configuration
  ansible.builtin.copy:
    src: snmpd.conf
    dest: '/etc/snmp/snmpd.conf'
    owner: 'root'
    group: 'root'
    mode: '0644'
```

3) **Copiare gli script**. Nota l'uso di un loop.

```
- name: Copy snmp agent bash script createdump and deletedump
  ansible.builtin.copy:
    src: "{{ item }}"
    dest: '/usr/bin/'
    owner: 'Debian-snmp'
    group: 'root'
    mode: '0740'

  loop:
    - createdump.sh
    - deletedump.sh
  notify: Restart snmpd
```

4) Configurare l'esecuzione di **codice tramite SNMP con privilegi** [9.5].

```
- name: Add privileged actions for snmp agent scripts
  ansible.builtin.copy:
    src: Debian-snmp
    dest: '/etc/sudoers.d/'
    owner: 'root'
    group: 'root'
    mode: '0440'
    validate: '/usr/sbin/visudo -csf %s'
```

5) Abilitare il demone al boot

```
- name: Enable snmpd at boot
  ansible.builtin.systemd:
    name: snmpd.service
    state: started
    enabled: yes
  notify: Restart snmpd
```

## 6) Configurare il servizio di log [6.4]

```
- name: Copy syslog configuration
  ansible.builtin.copy:
    src: deletedump.conf
    dest: '/etc/rsyslog.d/'
    owner: 'root'
    group: 'root'
    mode: '0644'
  notify: Restart rsyslog
```

### files/

*eth1\_cfg*: configurazione interfaccia di rete per gli agent che usano DHCP  
auto eth1  
iface eth1 inet dhcp

*snmpd.conf*: configurazione demone [9.2], specificando anche gli script [9.5].

master agentx  
agentaddress udp::161

view all included .1

rocommunity public default -V all  
rwcommunity supercom default -V all

extend-sh createdump /usr/bin/createdump.sh &!  
extend-sh deletedump /usr/bin/deletedump.sh

*Debian-snmp*: configurazione per eseguire il codice con privilegi [9.5], tcpdump e chown necessitano privilegi di root. tcpdump permette di intercettare fino a 10000 pacchetti su eth1 e salvarli nel file con prefisso /tmp/dump\_. chown di cambiare il proprietario del file che inizia con /tmp/dump\_.

```
Debian-snmp ALL=(ALL) NOPASSWD: /usr/bin/tcpdump -n -U -i eth1 -c 10000 -w /tmp/dump_*
Debian-snmp ALL=(ALL) NOPASSWD: /usr/bin/chown Debian-snmp /tmp/dump_*
```

*deletedump.conf*: configurazione per creare il file di log.  
local1.info /var/log/deletedump.log

*createdump.sh*, *deletedump.sh*: script

### ATTO III : gli script

Capiamo prima il funzionamento di tcpdump...

#### **tcpdump [opzioni] [filtro]**

→ è uno strumento usato per catturare ed eseguire diagnostica sui pacchetti che attraversano la rete in Unix; le [opzioni] specificano come visualizzare i dati, il [filtro] quali pacchetti catturare.

→ NECESSARI I PERMESSI sudo

→ principali opzioni:

-i specifica l'interfaccia di rete (*Es: -i any, -i eth1*)

-c [N] visualizza i primi N pacchetti

-w [output.pcap] scrive i pacchetti nel file di output

-r [input.pcap] legge i pacchetti dal file

-l -n per rilevare in real-time i pacchetti senza essere bufferizzati e senza risolvere hostname

-U per rilevare i pacchetti in output senza che vengano bufferizzati (come -l)

→ principali filtri:

- filtrare un host: **tcpdump host 10.0.5.2**

- filtrare una porta: **tcpdump port 80**

- filtrare per protocollo: **tcpdump icmp**

- filtrare per sorgente: **tcpdump src 192.168.1.1**

- filtrare per destinazione: **tcpdump dst 192.168.1.1**

- filtrare per dimensione pacchetto (byte): **tcpdump less 100 (/greater)**

*"Il primo comando in questione (script **createdump.sh**) deve poter generare un PCAP, utilizzando TCPDUMP, che termini automaticamente dopo aver catturato 10.000 pacchetti, sull'interfaccia di rete interna tra la macchina virtuale con l'Agent e il Controller. Utilizzando Rsyslog (configurato tramite Ansible) gli agent segnalano al Controller la disponibilità del PCAP e quindi la possibilità di poterlo recuperare da remoto."*

```
#!/bin/bash

ORA=$(date +%s)

IPV4ETH1=$(ip a | grep "10.1.1." | awk '{ print $2 }' | awk -F '/' '{ print $1 }')

# salvo il file pcap usando tcpdump
/usr/bin/sudo /usr/bin/tcpdump -n -U -i eth1 -c 10000 -w /tmp/dump_${ORA}.pcap &&

# cambio il proprietario del file
/usr/bin/sudo /usr/bin/chown Debian-snmp /tmp/dump_${ORA}.pcap &&

# && concatena i comandi in modo che il secondo è eseguito solo se il
# primo ha avuto successo
```

```

# uso logger per inviare il messaggio di log al controller
# in particolare sfrutto l'opzione -n per indicare la macchina remota
/usr/bin/logger -n 10.1.1.254 -p local1.info "READY $IPv4ETH1 /tmp/dump_${ORA}.pcap"

# in alternativa a -n potevo:
# configuro rsyslog dell'agent con
# local1.info @10.1.1.254
# e rsyslog del controller con
# local1.info /var/log/dumps.log
# e uso
# logger -p local1.info "READY /tmp/dump_${ORA}.pcap"

```

*"Il secondo comando custom (script **deletedump.sh**) invece deve poter cancellare i file PCAP che il Controller ha già recuperato, i cui nomi vengono inviati via syslog dal Controller agli Agent. "*

```

#!/bin/bash

( tail -f /var/log/deletedump.log | /usr/bin/grep -E --line-buffered 'DONE
DOWNLOAD' | while read -r TAG TAG FILEPATH ; do

    /usr/bin/rm -f "$FILEPATH"

    exit 0

done ) &

# tail -f /var/log/deletedump.log :legge continuamente il file di log in
# tempo reale

# grep filtra le linee che contengono 'DONE DOWNLOAD', --line-buffered
# garantisce che le linee vengano inviate senza attesa alla pipe

# il ciclo while per ogni riga passata da grep assegna i primi 3
elementi alle variabili TAG TAG e FILEPATH (primi due ignorati)

# cancelliamo allora il file in modo forzato

# exit 0 termina il ciclo (fa sì che lo script termini dopo la
cancellazione del primo file che corrisponde al filtro)

```

*"Periodicamente, ogni 3 minuti alternativamente su un agent e sull'altro (es: minuto 3 agent1, minuto 6 agent2, minuto 9 agent1, ...), il Controller invocherà uno script (**managedump.sh**) per eseguire il protocollo di generazione, recupero e cancellazione dei PCAP. Lo script deve immediatamente uscire senza fare nulla se ce n'è già un'istanza lanciata in precedenza e non ancora terminata, mentre in caso contrario deve*

- *avviare la generazione del PCAP usando la richieste del corrispondente oggetto SNMP*
- *mettersi in attesa che sul log compaia la conferma di generazione avvenuta, che include il nome del file da scaricare*
- *scaricare col metodo più sicuro che conoscete il file dall'agent in una directory locale*
- *inviare conferma all'agent loggando il nome del file scaricato*
- *invocare via SNMP lo script di cancellazione del PCAP"*

```
#!/bin/bash

TEMP="/tmp/running"
PCAPFOLDER="/tmp/pcap"
# quale agente ha eseguito l'ultimo operazione?
LASTAGENTRUNNING="/tmp/lastagent"

# creo le dir (assicurandomi che non venga prodotto errore se già esistono -p) e il file dumps.log
mkdir -p "$TEMP"
mkdir -p "$PCAPFOLDER"
touch /var/log/dumps.log

# se il file LASTAGENTRUNNING esiste già lo leggo e inserisco il suo contenuto nella variabile LASTAGENT (che sarà o agent1 o agent2), in questo modo so chi è l'ultimo agente attivo
LASTAGENT=""

if [[ -f "$LASTAGENTRUNNING" ]]; then
    LASTAGENT=$(cat "$LASTAGENTRUNNING")
else
    touch "$LASTAGENTRUNNING"
fi
```

```

# dobbiamo ora recuperare gli IP dei due agent, questi sono stati
assegnati tramite DHCP. Il file dnsmasq.leases registra gli indirizzi ip
assegnati in rete.

DHCPLEASE="/var/lib/misc/dnsmasq.leases"
AGENT1IPv4=""
AGENT2IPv4=""

# per ogni riga (while read) del file $DHCPLEASE mappo ogni elemento
sulla corrispondente variabile (ID, ...) e ricavo gli IP di agent1 e
agent2

while read -r ID MACADDRESS IPv4 HOSTNAME IPv6 ; do
    if [[ "$HOSTNAME" == agent1 ]] ; then
        AGENT1IPv4="$IPv4"
    elif [[ "$HOSTNAME" == agent2 ]] ; then
        AGENT2IPv4="$IPv4"
    fi
done < "$DHCPLEASE"

# creiamo una funzione per il dump remoto
function remotedump {
# creiamo un file temporaneo con il pid ($$) del processo corrente
    /usr/bin/echo $$ > "$TEMP"/$$_"$1"_running

# tramite tail leggiamo le linee di dumps.log in tempo reale e filtriamo
solo quelle con READY seguito dall'IP passato alla funzione remotedump
    tail -n 0 -f /var/log/dumps.log | /usr/bin/grep -E --line-
buffered "READY $1" | while read -r TAG IPv4 REMOTEFILERPATH ; do

# copio (scp) il file remoto (REMOTEFILERPATH) da IP ($1) alla mia
cartella (PCAPFOLDER)
    /usr/bin/scp root@"$1":'$REMOTEFILERPATH' "$PCAPFOLDER"
# invio un log per indicare che ho finito il download
    /usr/bin/logger -n "$1" -p local1.info "DONE DOWNLOAD $REMOTEFILERPATH"
# lancio deletedump tramite SNMP sull'agent
    /usr/bin/snmpget -v 1 -c public "$1" 'NET-SNMP-EXTEND-
MIB::nsExtendOutputFull."deletedump"'
# cancello il file temporaneo usato per indicare il processo in
esecuzione
    /usr/bin/rm -f "$TEMP"/$$_"$1"_running

```

```

        exit 0
done
}

# controllo se ci sono processi in esecuzione
PROCESS_RUNNING=$(/usr/bin/find "$TEMP" -type f)

# se l'ultimo agente era 1 e ci sono meno di due processi in esecuzione (-lt 2) (-ne evita il newline finale) (meno di due perché agent1 e agent2 possono alternarsi)
if [[ "$LASTAGENT" == "agent1" ]] && [[ $(/usr/bin/echo -ne "$PROCESS_RUNNING" | /usr/bin/wc -l) -lt 2 ]] ; then
    LASTAGENT="agent2"
    /usr/bin/echo "$LASTAGENT">>"$LASTAGENTRUNNING"
# lancio la nostra funzione remotedump per agent2 (in background) e lancio il comando createdump
    remotedump "$AGENT2IPv4" &
    /usr/bin/snmpget -v 1 -c public "$AGENT2IPv4" 'NET-SNMP-EXTEND-MIB::nsExtendOutputFull."createdump"'

# se l'ultimo agente era 2 e ci sono meno di due processi in esecuzione (-lt 2) (-ne evita il newline finale)
elif [[ "$LASTAGENT" == "agent2" ]] && [[ $(/usr/bin/echo -ne "$PROCESS_RUNNING" | /usr/bin/wc -l) -lt 2 ]] ; then
    LASTAGENT="agent1"
    /usr/bin/echo "$LASTAGENT">>"$LASTAGENTRUNNING"
# lancio la nostra funzione remotedump per agent2 (in background) e lancio il comando createdump
    remotedump "$AGENT1IPv4" &
    /usr/bin/snmpget -v 1 -c public "$AGENT1IPv4" 'NET-SNMP-EXTEND-MIB::nsExtendOutputFull."createdump"'

```

```
# primo lancio

elif [[ $(/usr/bin/echo -ne "$PROCESS_RUNNING" | /usr/bin/wc -l) -lt 2 ]] ; then
# NON abbiamo nessun processo che sta andando quindi lanciamo e salviamo che stiamo andando
    LASTAGENT="agent1"
    /usr/bin/echo "$LASTAGENT">>"$LASTAGENTRUNNING"
    remotedump "$AGENT1IPv4" &
    /usr/bin/snmpget -v 1 -c public "$AGENT1IPv4" 'NET-SNMP-EXTEND-MIB::nsExtendOutputFull."createdump"'
fi

exit 0
```

# 10 Configurazione centralizzata

Un sistema correttamente funzionante dipende da parecchi file di configurazione, ma in una rete possono esserci centinaia di macchine che hanno gli stessi file. Aggiungere un utente, un host, cambiare un parametro di un demone, vorrebbe dire cambiare coerentemente centinaia di file di configurazione. Nasce, dunque, l'esigenza di condividere queste informazioni tra sistemi diversi.

In questo corso non vedremo sistemi di descrizione delle infrastrutture cloud, sistemi di configuration management, sistemi di supporto a DevOps.

Ci concentreremo invece, nella gestione dell'autenticazione centralizzata definendo due aspetti fondamentali: come scegliere localmente la sorgente dei dati e come trasferire dati dal server centrale ai client.

Scelta della sorgente: NSS / PAM.

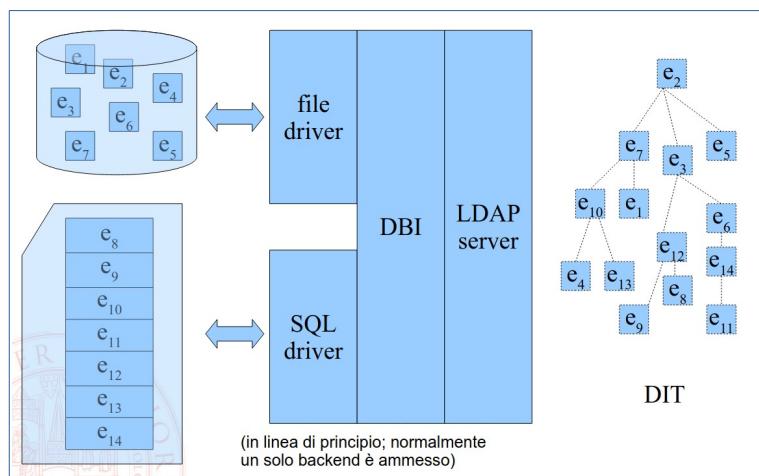
Protocollo di distribuzione: LDAP.

## 10.1 LDAP

**LDAP** (Lightweight Directory Access Protocol) è un protocollo standard usato per accedere e mantenere informazioni in una directory, principalmente in un contesto di rete. Deriva dal modello x.500 visto nel corso di Reti di Calcolatori.

La base di dati della directory può essere un qualsiasi sistema di archiviazione. Viene definita un interfaccia **DBI** che gestisce una serie di entry che formano i nodi del sistema.

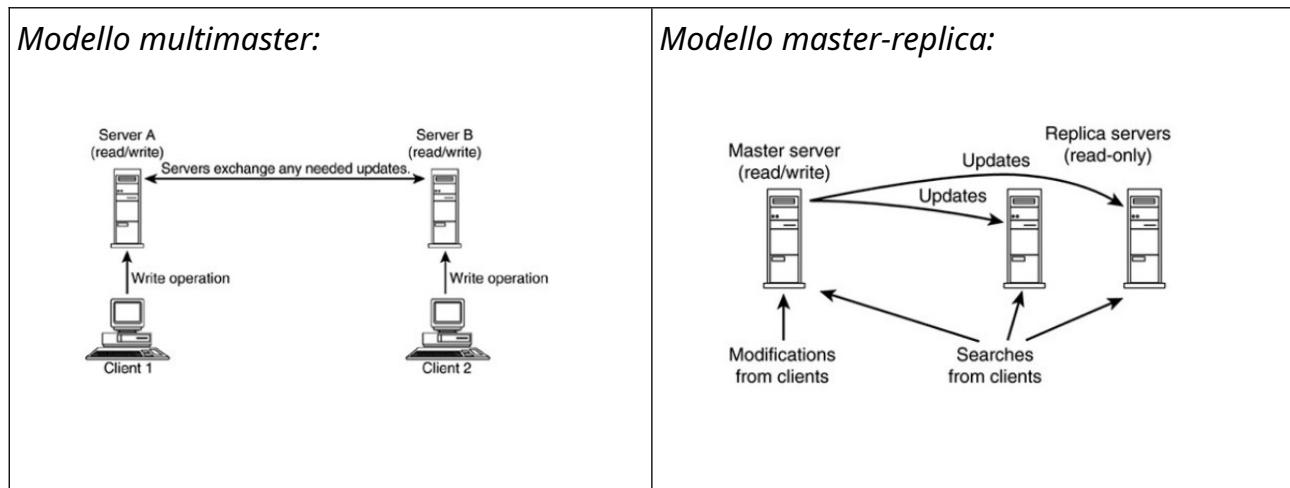
LDAP organizza le informazioni in una struttura gerarchica simile a un albero, nota come **DIT** (Directory Information Tree). I nodi principali di questo albero rappresentano oggetti come utenti, gruppi o dispositivi e sono identificati con un **DN** (Distinguished Name) univoco. La gerarchia permette di segmentare e organizzare le informazioni in modo logico.



LDAP è un protocollo leggero e poco impegnativo in termini di risorse. È basato su TCP/IP (porta 389, TLS 636), il che lo rende facilmente integrabile in reti esistenti. Prevede nativamente meccanismi di elevata disponibilità e la sua natura permette di implementare efficacemente la ridondanza.

L'architettura LDAP è costituita da:

- Client LDAP: Un'applicazione o un utente che esegue richieste al server LDAP per recuperare o modificare informazioni.
- Server LDAP: Gestisce la directory e risponde alle richieste del client.



## 10.2 Entry e schema

Una **entry** è una **collezione di attributi**, non si definisce un tipo, ma si specifica direttamente il valore etichettato dal tipo. Tra gli attributi ce ne sono due sempre presenti: **dn** (distinguished name) e **objectClass** (una o più).

*Esempio: [attributo: tipoAttributo=valoreAttributo]*

```

dn: dc=labammsis
objectClass: dcObject
objectClass: organization
dc: labammsis

```

Questo formato è usato per scambiare entry tra client e server e prende il nome di LDIF. Un file di questo tipo può contenere più entry, semplicemente separate da una virgola.

L'inserimento di una entry deve essere disciplinato, si usa dunque uno **schema**.

Lo **schema** definisce la struttura e le regole dei dati archiviati nel server LDAP. Esso stabilisce quali tipi di oggetti (come utenti, gruppi, dispositivi) possono essere presenti nella directory e quali attributi ogni oggetto può o deve avere. Questo è scritto in un file **ldif**, salvato nella sezione dei metadati del directory **ldap**.

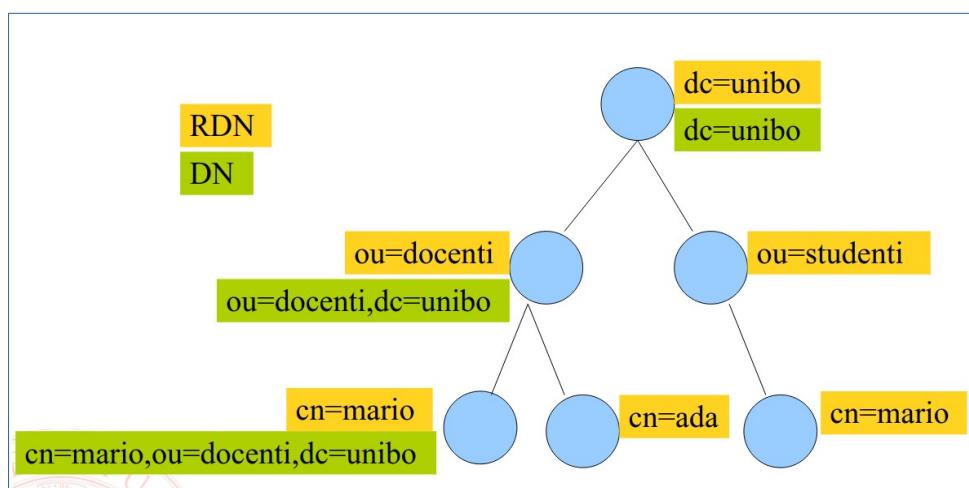
Nello schema sono presenti attributi e classi:

- **ObjectClass**: Definisce il tipo di entry (ad esempio, person, organizationalPerson, inetOrgPerson per gli utenti). L'ObjectClass stabilisce quali attributi sono permessi per ciascuna entry.

- **Attributi**: Gli attributi sono campi che contengono le informazioni specifiche dell'entry, come il nome (cn), il cognome (sn), l'indirizzo email (mail), ecc. Ogni attributo è associato a una **syntax** che specifica il tipo di dati ammessi (stringhe, numeri, valori booleani).

In ogni entry si sceglie un attributo rappresentativo.

La stringa **tipoAttributo=valoreAttributo** scelta è il **RDN**, mentre il nome del nodo a cui si appende è il **BDN**. Il nome **univoco** della entry è dato dal **DN**, ovvero RDN+BDN.



Ogni RDN deve essere localmente unico. A volte un singolo attributo non è sufficiente a garantire questa unicità (ad esempio nel caso dei nomi); ldap propone dunque la concatenazione (+) di più attributi. (Es: cn=Alessandro+sn=Quaranta).

## 10.3 Tipi di attributi

Un **attributeType** specifica separatamente come i valori possono essere memorizzati e confrontati:

- **SYNTAX**: rappresenta il tipo di dato nativo associato all'attributo.
- **Matching Rules**: definiscono i criteri con cui i valori degli attributi vengono confrontati durante le operazioni di ricerca e confronto.

- **EQUALITY**: Determina come verificare se due valori sono uguali. Ad esempio, caseIgnoreMatch per il confronto case-insensitive delle stringhe.
- **ORDERING**: Specifica l'ordinamento dei valori, come ad esempio confronti alfabetici o numerici.
- **SUBSTRING**: Permette di cercare valori parziali all'interno di un attributo, ad esempio per trovare tutte le stringhe che contengono un determinato sottostringa.

- **Vincoli d'uso**: restringono come e quante volte un attributo può essere utilizzato in una entry.

- **SINGLE-VALUE**: Restringe l'attributo a un singolo valore per entry, impedendo di avere più valori per quel campo (ad esempio, un utente può avere un solo uid).
- **NO-USER-MODIFICATION**: Indica che il valore dell'attributo non può essere modificato dagli utenti, ma solo dal sistema.
- **USAGE**: Specifica lo scopo dell'attributo. Ad esempio, può essere userApplications (per dati utente generici) o directoryOperation (per scopi di gestione della directory).

- **Dipendenze gerarchiche**: permettono di creare attributi basati su altri tipi di attributo esistenti, utilizzando la direttiva **SUP**.

Esistono numerosi attributeType già definiti da vari internet standard:

<https://oav.net/mirrors/LDAP-ObjectClasses.html>

Esempi: *cn = common name, dc = domain component, ....*

Per definire un nuovo attributeType in LDAP, è necessario aggiungere la sua descrizione nella configurazione del server LDAP. Questo viene fatto inserendo la definizione sotto forma di un attributo di una entry speciale (**olcAttributeTypes**) utilizzata proprio per la configurazione degli attributi. Ogni attributeType ha un Object Identifier (OID) univoco.

*Esempio 1 creo un nuovo attributo chiamato fn o filename:*

```
olcAttributeTypes: ( 1000.1.1.1 NAME ( 'fn' 'filename' )
DESC 'nome del file'
EQUALITY caseExactMatch
SUBSTR caseExactSubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

*Esempio 2 creo un nuovo attributo chiamato telephoneNumber:*

```
olcAttributeTypes: ( 2.5.4.20 NAME 'telephoneNumber'  
    DESC 'Telephone Number'  
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.50  
    EQUALITY telephoneNumberMatch  
    SUBSTRING telephoneNumberSubstringsMatch  
    SINGLE-VALUE )
```

## 10.4 Classi

Lo scopo delle **objectClass** è di elencare quali attributi una entry deve avere (**MUST**) e quali può (**MAY**). Le classi possono essere di tre tipi...

- **ABSTRACT**: servono per creare una tassonomia di categorie di oggetti, ancora troppo astratte per poter originare entry.
- **STRUCTURAL**: servono per descrivere categorie di oggetti concreti (la più usata).
- **AUXILIARY**: servono per descrivere collezioni di attributi non direttamente collegati a specifiche categorie di oggetti, ma che possono essere aggiunti per arricchire il contenuto informativo di una entry.

Anche in questo caso ne esistono di già definite:

<https://oav.net/mirrors/LDAP-ObjectClasses.html>

Per definire una nuova **objectClass** in LDAP, proprio come per gli attributi, è necessario aggiungere la sua descrizione nella configurazione del server sotto forma di un attributo in una entry speciale. Questo avviene tipicamente tramite l'attributo **olcObjectClasses**.

*Esempio nuova classe chiamata dir:*

```
olcObjectClasses: ( 1000.2.1.1 NAME 'dir'  
    DESC 'una directory'  
    MUST fn  
    MAY fs  
    AUXILIARY )
```

**ATTENZIONE:** in ogni entry deve essere nominata una sola classe strutturale e possono essere nominate più classi ausiliari.

## 10.5 Gestione schemi e entry

Per riconfigurare un server LDAP sarà dunque necessario inserirgli un entry in formato ldif.

Mettiamo di aver creato lo schema filesystem.ldif:

```
dn: cn=filesystem,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: filesystem
olcAttributeTypes: ( 1000.1.1.1 NAME ( 'fn' 'filename' )
DESC 'nome del file'
EQUALITY caseExactMatch
SUBSTR caseExactSubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
olcAttributeTypes: ( 1000.1.1.2 NAME ( 'fs' 'filesize' )
DESC 'dimensioni del file'
EQUALITY integerMatch
ORDERING integerOrderingMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
olcObjectClasses: ( 1000.2.1.1 NAME 'dir'
DESC 'una directory'
MUST fn
MAY fs
AUXILIARY )
olcObjectClasses: ( 1000.2.1.2 NAME 'file'
DESC 'un file'
MUST ( fn $ fs )
AUXILIARY )
```

```
sudo ldapadd -Y EXTERNAL -H ldap:/// -f filesystem.ldif
```

→ comando apposito per inserire uno schema scavalcando l'autenticazione ldap

Passaggi per eliminare uno schema:

```
sudo systemctl stop slapd
```

→ Navigo come root nella dir degli schemi e cerco il file con il nome del mio schema. Una volta ottenuto posso eliminarlo.

```
cd "/etc/ldap/slapd.d/cn=config/cn=schema"
rm "cn={4}filesystem.ldif"
sudo systemctl start slapd
```

## **ldapsearch -x -b "base\_dn" [-s scope] [filtro]**

→ questo è il comando generale per interrogare una directory LDAP  
“**base\_dn**” rappresenta il punto di partenza da cui effettuare la ricerca (*dc=labammsis*)  
**scope** imposta il livello di profondità della ricerca: **base, one, sub**  
il **filtro** specifica eventuali criteri  
(possibile migliorare la leggibilità con -LLL)

→ filtri:

uguaglianza = (Es: cn=Ale Quaranta, tutte le entry con cn Ale Quaranta)

presenza \* (Es: mail=\*, tutte le entry con attributo mail)

confronto < = > (Es: age<=30)

sottostringa =\*a\*(Es: cn=\*Ale\*)

negazione ! (Es: !(cn=Ale))

AND & (Es: (&(cn=Ale)(age=22)))

OR | (Es: (|(cn=Ale)(age=22)))

Esempio di combinazione: (&(|(cn=Ale)(cn=Livio))(!(status=disoccupato)))

## **ldapadd -x -D "cn=admin, dc=labammsis" [-f file.ldif]**

→ questo è il comando per aggiungere una entry  
-D specifica l'utente DN (Distinguished Name) che effettua l'operazione  
-w fornisce la password in chiaro per cn=admin  
-f specifica il file, se omesso apre STDIN

Esempio file:

```
dn: cn=John Doe,dc=labammsis
objectClass: inetOrgPerson
cn: John Doe
sn: Doe
mail: john.doe@example.com
```

## **ldapmodify -x -D "cn=admin, dc=labammsis" [-f file.ldif]**

→ modifica una entry esistente; funzionalità **add, replace, delete**

Esempio file:

```
dn: cn=John Doe,dc=labammsis
changetype: modify
replace: mail
mail: john.doe@example.com
```

## **ldapdelete -x -D "cn=admin, dc=labammsis" -w "dn\_da\_rimuovere"**

→ elimina un entry

## 10.6 LDAP per l'autenticazione Linux

Partiamo generando una VM “**server**” con rete internal “LAN1” e ip statico 10.1.1.2...

Il pacchetto che contiene il demone che eroga i servizi LDAP è **slapd**.

**sudo apt install -y slapd ldap-utils**

All'atto dell'installazione verrà richiesto di impostare la password dell'amministratore della directory. Non importa cosa si sceglie in questa fase, perché l'installazione di default utilizza *nodomain* come dominio radice del DIT, e vogliamo comunque riconfigurare il servizio successivamente.

Utilizziamo per questo **sudo dpkg-reconfigure slapd**

- **No** alla prima domanda
- **labammsis** alla seconda domanda, sarà il dominio radice
- **Unibo** alla terza domanda (non essenziale)
- **gennaio.marzo** come password, per essere consistenti nei prossimi esempi.
- **Yes / Yes** alle successive e ultime due domande

Al solito, una volta installato può essere controllato con

**systemctl { start | status | stop } slapd**

---

Lato **client** il pacchetto con la strumentazione necessaria è semplicemente **ldap-utils**.

Le connessioni tra un client (come il comando `ldapsearch`) e un server LDAP sono *autenticate*, il che significa che richiedono credenziali valide o altri metodi di autenticazione.

Ci sono vari metodi di autenticazione disponibili, ecco i principali:

- **-Y EXTERNAL -H ldap://**

Usa il metodo EXTERNAL, che sfrutta l'utente del sistema operativo che esegue il comando per autenticarsi automaticamente tramite un *socket Unix locale*. È utile quando il comando viene eseguito direttamente sul server LDAP stesso, piuttosto che da un client remoto.

- **-x -H ldap:// IP.DEL.SERVER /**

Indica che si sta usando l'autenticazione semplice, che invia le credenziali in chiaro. Di default si autentica in modo anonimo come utente ospite ma può essere seguito da

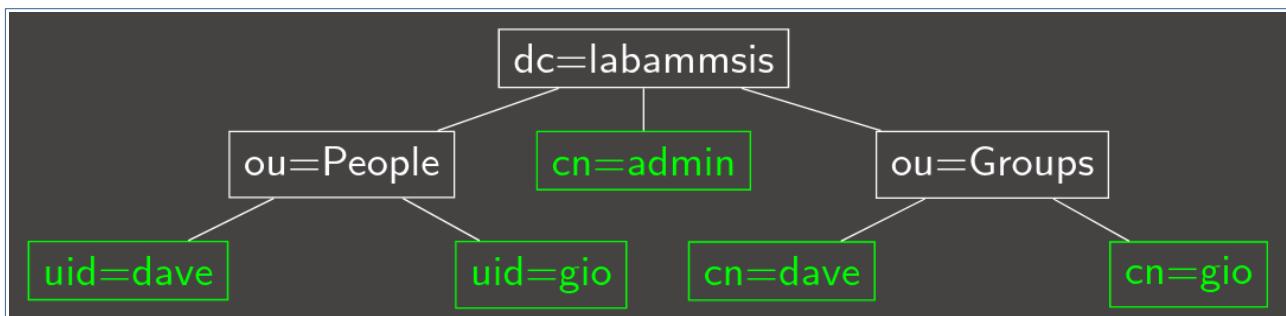
**-D DN.UTENTE -w PASSWORD** per autenticarsi come utente specifico. Usando **-W** verrà chiesta la password in modo interattivo.

```
ldapsearch -x -D "cn=admin,dc=labammsis" -w "gennaio.marzo" -H ldap://10.0.2.15/ -b "dc=labammsis" -s sub
```

Per creare utenti e gruppi all'interno di un server LDAP e replicare gli attributi di un utente UNIX, come indicato, è necessario seguire questi passaggi chiave. In LDAP, gli utenti e i gruppi sono organizzati in **Organizational Units (OU)**, che sono come cartelle all'interno della directory in cui vengono archiviati oggetti correlati.

La struttura tipica include:

- **ou=People, dc=labammsis** per gli **utenti**: dove vengono archiviati tutti gli utenti.
- **ou=Groups, dc=labammsis** per i **gruppi**: dove vengono definiti i gruppi di utenti.



Per aggiungere un'Organizational Unit, si utilizza un file in formato LDIF. Il file **people.ldif** rappresenta l'OU `People`, sotto la quale saranno collocati tutti gli utenti del sistema:

```
dn: ou=People,dc=labammsis
objectClass: organizationalunit
ou: People
description: system users
```

Come già visto, per aggiungere la entry:

```
ldapadd -x -D "cn=admin,dc=labammsis" -w "gennaio.marzo" -H ldap:/// -f people.ldif
```

Possiamo verificare anche se l'inserimento ha avuto successo:

```
ldapsearch -x -LLL -H ldap:/// -b "dc=labammsis" -s sub
```

In modo del tutto analogo definiamo **groups.ldif**:

```
dn: ou=Groups,dc=labammsis
objectClass: organizationalunit
ou: Groups
description: system groups
```

Gli **utenti** sono definiti con vari attributi che mimano quelli di un account utente UNIX. L'entry in formato LDIF per un utente include dettagli come il nome, il cognome, la mail, l'UID, la directory home e la shell di login. Il seguente esempio esplicativo usa dunque classi come posixAccount o shadowAccount che permettono di usare attributi stile LINUX.

```
dn: uid=dave,ou=People,dc=labammsis
objectClass: top
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
givenName: Davide
cn: Davide
sn: Berardi
mail: davide.berardi@unibo.it
uid: dave
uidNumber: 10000
gidNumber: 10000
homeDirectory: /home/dave
loginShell: /bin/bash
gecos: Davide "Dave" Berardi
userPassword: {crypt}x
```

L'ultimo attributo è semplicemente un placeholder; la password dovrà essere impostata con un comando apposito (**ldappasswd**).

```
ldappasswd -x -D "cn=admin,dc=labammsis" -w "gennaio.marzo" -s
"nuovaPassword" "uid=dave,ou=People,dc=labammsis"
```

Ogni **gruppo** deve anch'esso essere definito in LDAP. L'entry per un gruppo include il nome comune (CN) del gruppo e l'identificativo numerico associato. Per creare il gruppo "dave":

```
dn: cn=dave,ou=Groups,dc=labammsis
objectClass: top
objectClass: posixGroup
cn: dave
gidNumber: 10000
```

Esempio: Ricerca specifica su un server LDAP con un certo //IP

```
ldapsearch -x -LLL -H ldap://IP -b 'DC=STUDENTI,DC=DIR,DC=UNIBO,DC=IT' -W
'(&(objectClass=user)(employeeID=0000712698))' SAMAccountName
```

SAMAccountName è l'attributo usato tipicamente per memorizzare il nome utente di accesso nei sistemi basati su Active Directory.

## 10.7 Configurazione sistemi per il login via LDAP

Perchè un sistema possa integrare LDAP come fonte di informazioni sugli utenti e come metodo per la loro autenticazione, servono le librerie contenute nei pacchetti **libnss-ldapd** e **libpam-ldapd**.

**sudo apt install libnss-ldapd libpam-ldapd**

Nota: Durante l'installazione, verrà installato automaticamente anche il pacchetto nslcd (Name Service LDAP Client Daemon), necessario per gestire le connessioni con il server LDAP.

**1)** Durante l'installazione dei pacchetti, ti verranno richieste alcune informazioni di configurazione:

- **URI del server LDAP:** L'indirizzo del server LDAP, ad esempio ldap://IP\_DEL\_SERVER\_LDAP.
- **Dominio radice:** Il DN di base del dominio, ad esempio dc=STUDENTI,dc=DIR,dc=UNIBO,dc=IT

Se vuoi modificare queste impostazioni in seguito: **sudo dpkg-reconfigure nslcd**  
Durante la reconfigurazione vengono offerte opzioni aggiuntive:

- **Account per le Ricerche:** È possibile specificare un DN e una password per un account LDAP dedicato alle ricerche. **ATTENZIONE:** Per il nostro caso, se il server accetta ricerche anonime, evita di inserire le credenziali dell'amministratore LDAP, poiché verrebbero salvate in chiaro nel file /etc/nslcd.conf.
- **Connessioni Cifrate:** In questo caso, l'uso di connessioni cifrate viene disabilitato.

**2)** Dopo l'installazione, LDAP potrebbe non essere ancora attivo su NSS. Apriamo /etc/nsswitch.conf e configuriamo i servizi passwd, group e shadow:

```
passwd: files systemd ldap
group: files systemd ldap
shadow: files ldap
```

**3)** Durante l'installazione e configurazione di libnss-ldapd, ti verrà chiesto di specificare quali fonti dati indirizzare su LDAP. Seleziona: **passwd, group, shadow**.

Riavviamo il demone: **sudo systemctl restart nscd.service**

Testate se con **getent passwd** vengono mostrati gli utenti inseriti nella directory in precedenza. Testate una query su un utente presente solo in LDAP, come **getent passwd dave**. Se non funziona una delle due, provate a riconfigurare il pacchetto nss con **sudo dpkg-reconfigure libnss-ldapd**.

4) Per configurare **PAM (Pluggable Authentication Modules)** per utilizzare LDAP per l'autenticazione degli utenti su Debian, occorre integrare PAM con ns lcd, in modo che PAM possa autenticare gli utenti basandosi sulle informazioni archiviate nel server LDAP.

Nella versione corrente di Debian (Bookworm), non è necessario configurare separatamente PAM con pam\_ldap.conf o pam\_ldap.secret, poiché i parametri sono già presenti in /etc/nslcd.conf. Le informazioni di base sono:

BASE dc=labammsis  
URI ldap://

Se tutti i passaggi sono stati eseguiti correttamente, ora dovresti poter utilizzare LDAP come fonte per gli utenti e l'autenticazione nel sistema.

---

I pacchetti che abbiamo installato, come molti altri, dispongono di una procedura di post-installazione per impostare i parametri di configurazione di base. Ansible ovviamente non può interagire con le maschere interattive, e quindi possono succedere due cose:

- il provisioning fallisce tentando di installare il pacchetto
- vengono impostati parametri di default, che poi bisogna cambiare a mano con **dpkg-reconfigure**

Però c'è un modo generale di automatizzare il provisioning di pacchetti Debian usando **debconf**, e c'è un modulo Ansible che ne può trarre vantaggio.

Col comando **debconf-show nome\_pacchetto** si può vedere quali sono le domande che il configuratore farebbe, e le risposte di default (si vedano le varie man page per una panoramica sul sistema). Perché funzioni, il pacchetto deve essere installato, quindi almeno una volta va fatto a mano. Si può poi usare il **modulo Ansible deb-conf** per preconfigurare le risposte. Prerequisito è installare i pacchetti **debconf debconf-utils**. Si può quindi usare un task che per ogni domanda specificata nel parametro question fornisce la risposta nel parametro value del modulo (specificando anche il tipo di dato nel parametro vtype).

**SU VIRTUALE DISPONIBILI DUE ESEMPI: ldap\_login, ldap\_server**  
[21/5/24 Integrazione di LDAP: 12. Automatizzare il provisioning]

## 10.8 Test del login

Come già anticipato, per testare il funzionamento di un utente ed effettuare il login, si deve attribuire una password agli utenti creati in precedenza, per i quali si era usato un placeholder.

```
ldappasswd -D cn=admin,dc=labammsis -w gennaio.marzo  
uid=dave,ou=People,dc=labammsis -s pistacchio
```

Il comando qui sopra riportato specifica l'utente che sta eseguendo il comando, la sua password, per quale utente sta configurando la password ed infine tramite opzione **-s** l'effettiva password (in questo caso pistacchio).

Configuriamo allora la macchina “**client**” connessa alla stessa rete LAN1 del server visto in precedenza [10.6], questa avrà ip statico 10.1.1.1. Fatto ciò possiamo seguire i passaggi visti nel capitolo precedente [10.7] per far in modo che il client utilizzi il server per il login dei propri utenti.

[21/5/24 Integrazione di LDAP: 14. client-server : **ldap\_lab**] CONFIGURAZIONE TOTALE:

**/Vagrantfile** → tira su VM server ip “10.1.1.2”  
tira su VM client ip “10.1.1.1”

**/site.yml** → serve per includere i roles  
per l'host client e l'host server

### **/roles**

- / **common** → update dei pacchetti e cambio allow-hotplug
- / **set\_proxy** → configurazione del proxy lab 3 unibo
- / **ldap\_server**
  - / task → installazione di debconf e debconf-utils [10.7]
    - installazione di slapd
    - setup di debcof per slapd
    - copia dei file people.ldif groups.ldif [10.6] + handler
  - / handlers → l'handler legato ai file ldif aggiunge le entry a ldap
  - / vars → variabili per il loop debcof
- / **ldap\_login**
  - / task → installazione di debconf e debconf-utils [10.7]
    - setup di debconf per nslcd e per libnss-ldap
    - installazione di ldap-util, libpam-ldap e libnss-ldap
  - / vars → variabili per il loop debcof

**/test.ldif** → entry di esempio per l'aggiunta di un account

Dopo aver aggiunto l'entry utente `test` al server, ed aver specificato una password [10.8] . Possiamo tentare il login...

```
:$ vagrant ssh client  
vagrant@client:$ sudo -i  
root@client# login  
client login: test  
password:  
test@client$
```

Allora il client sta effettivamente usando il server per i propri login.

Nota: è possibile una modifica (o altri comandi) delle entry anche da client... semplicemente unendo i concetti appresi [comando 10.5, opzione -H 10.6]

```
ldapmodify -x -H ldap://10.1.1.2/ -D cn=admin,dc=labammsis -w  
gennaio.marzo -f modifica.ldif
```

## 10.9 sudo-LDAP

Per configurare l'integrazione di sudo con LDAP e fare in modo che il server LDAP fornisca le informazioni sui permessi sudo è necessario installare sul client il pacchetto **sudo-ldap**:

```
sudo apt install sudo-ldap
```

L'installazione di questo pacchetto aggiornerà il file `/etc/nsswitch.conf` per includere `sudoers: files ldap`.

Lo schema `sudo-ldap`, che definisce attributi e classi necessari per gestire i permessi sudo tramite LDAP, si trova in `/usr/share/doc/sudo-ldap/schema.olcSudo`. Per permettere al server LDAP di riconoscere e gestire gli attributi e le classi sudo dobbiamo aggiungere il seguente schema:

sul server LDAP con privilegi di amministrazione:

```
sudo ldapadd -Y EXTERNAL -H ldapi:/// -f /usr/share/doc/sudo-ldap/schema.olcSudo
```

verifichiamo l'aggiunta:

```
sudo ldapsearch -Y EXTERNAL -H ldapi:/// -b "cn=schema,cn=config" "(objectClass=*)"
```

Fatto questo, diventa possibile inserire nel DIT standard la Organizational Unit specifica per sudo, esattamente come abbiamo fatto per inserire le OU "People" e "Groups":

```
dn: ou=SUDOers,dc=labammsis
objectClass: organizationalunit
ou: SUDOers
description: sudo enabled users
```

(aggiungere la entry ldif)

A questo punto si può informare il sistema di quale parte del DIT utilizzare per sudo, inserendo in /etc/ldap/ldap.conf la variabile:

```
SUDOERS_BASE ou=SUDOers,dc=labammsis
```

Logicamente, sotto la OU appena creata, si devono aggiungere entry con contenuto informativo equivalente a quello del file /etc/sudoers.

Se vogliamo convertire il file /etc/sudoers in formato LDAP possiamo usare il seguente comando che salva la formattazione nel file sudoers.ldif:

```
cvtsudoers -b "ou=SUDOers,dc=labammsis" -f ldif -o sudoers.ldif /etc/sudoers
```

Aprendo il file possiamo notare una struttura simile alla seguente:

```
dn: cn=example_user,ou=SUDOers,dc=labammsis
objectClass: top
objectClass: sudoRole
cn: example_user
sudoUser: example_user
sudoCommand: ALL
sudoHost: ALL
```

Ogni sudoRole specifica un set di permessi, come quali comandi (sudoCommand) e su quali host (sudoHost) un utente (sudoUser) può eseguire comandi con sudo.

Aggiungiamo la entry ldif:

```
ldapadd -x -D "cn=admin,dc=labammsis" -w "gennaio.marzo" -H
ldap:/// -f sudoers.ldif
```

Ora puoi testare l'accesso sudo per uno degli utenti configurati nel server LDAP eseguendo un comando con sudo su un client configurato per usare LDAP per sudo.

## Come usare il comando awk

Il comando awk è uno strumento potente e versatile in Unix/Linux per la manipolazione e l'elaborazione di testi.

La sintassi base è: **awk 'pattern {azione}' <file>**

dove l'azione è eseguita su ogni riga che soddisfa il pattern.

Concetti:

- **Campi:** Ogni colonna in awk è rappresentata come un "campo" (\$1, \$2, ... \$NF dove NF rappresenta il numero di campi). Il delimitatore di default è lo spazio o il tab, ma può essere modificato.
- **Modello:** Un'espressione booleana che determina se eseguire o meno l'azione su una data riga. Per esempio, NR == 1 è vero solo per la prima riga.
- **Azioni:** awk supporta azioni come `print` (per stampare dati).

Esempi esplicativi:

Per stampare solo la prima e la terza colonna di un file:

```
awk '{print $1, $3}' file.txt
```

Modificare il delimitatore:

```
awk -F ',', ''{print $1, $2}' file.txt
```

Per stampare solo le righe che contengono una determinata parola:

```
awk '/parola_cercata/ {print}' file.txt
```

L'esempio sopra citato accetta regex, parole che iniziano con ERROR:

```
awk '/^ERROR/ {print}' system.log
```

Per stampare solo le righe in cui il valore della seconda colonna è maggiore di 100:

```
awk '$2 > 100 {print}' file.txt
```

awk ha alcune variabili speciali utili per l'elaborazione:

- **NR:** Numero della riga corrente.
- **NF:** Numero di campi (colonne) nella riga corrente.

Il blocco BEGIN viene eseguito una sola volta prima che awk inizi a processare le righe, mentre END viene eseguito una sola volta al termine:

```
awk 'BEGIN {print "Inizio del file"} {print $1} END {print "Fine del file"}' file.txt
```