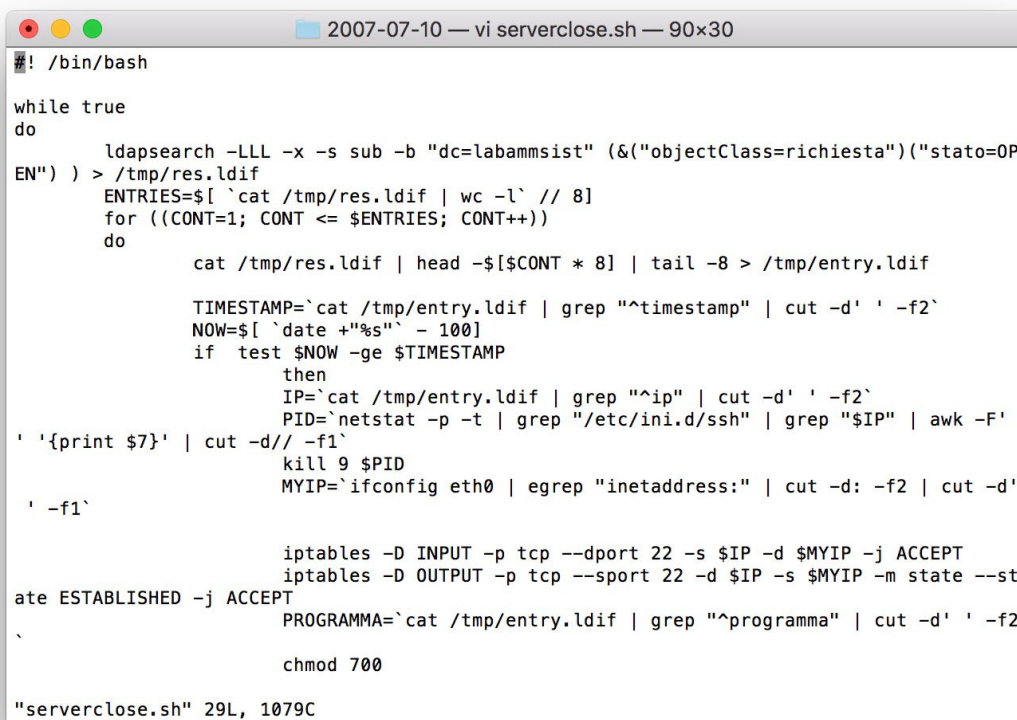


LAS

Guida pratica per l'esame di laboratorio



```
2007-07-10 — vi serverclose.sh — 90x30
#!/bin/bash

while true
do
    ldapsearch -LLL -x -s sub -b "dc=labammsist" (&("objectClass=richiesta")("stato=OPEN")) > /tmp/res.ldif
    ENTRIES=$( `cat /tmp/res.ldif | wc -l` // 8)
    for ((CONT=1; CONT <= $ENTRIES; CONT++))
    do
        cat /tmp/res.ldif | head -${CONT * 8} | tail -8 > /tmp/entry.ldif

        TIMESTAMP=`cat /tmp/entry.ldif | grep "^timestamp" | cut -d' ' -f2`
        NOW=$( `date +%s` - 100)
        if test $NOW -ge $TIMESTAMP
        then
            IP=`cat /tmp/entry.ldif | grep "^ip" | cut -d' ' -f2`
            PID=`netstat -p -t | grep "/etc/ini.d/ssh" | grep "$IP" | awk -F'
' '{print $7}' | cut -d// -f1`
            kill 9 $PID
            MYIP=`ifconfig eth0 | egrep "inetaddress:" | cut -d: -f2 | cut -d'
' -f1`

            iptables -D INPUT -p tcp --dport 22 -s $IP -d $MYIP -j ACCEPT
            iptables -D OUTPUT -p tcp --sport 22 -d $IP -s $MYIP -m state --st
ate ESTABLISHED -j ACCEPT
            PROGRAMMA=`cat /tmp/entry.ldif | grep "^programma" | cut -d' ' -f2`
            chmod 700

"serverclose.sh" 29L, 1079C
```

Introduzione

Ho deciso di scrivere questa guida per unire una miriade di appunti sparsi trovati in rete, nelle slide e in laboratorio in modo da rendere più facile il superamento dell'esame di LAS.

Pur non essendo esaustiva in ogni aspetto, la guida parla della maggior parte delle tematiche affrontate in laboratorio, soffermandosi sui problemi più comuni.

Contribuire alla guida

Se ritieni di poter migliorare la guida o hai trovato un errore, visita la repository GitHub ed apri una *issue*, oppure inviami un messaggio. Ogni contributo è ben accetto :)

Link Repository: <https://github.com/federico-terzi/las-guida-pratica>



Figura 1: QR Code alla repository di GitHub

Fonti

- Molto materiale è tratto dalle slide del Prof. Marco Prandini che sono liberamente disponibili a questo indirizzo: <http://lia.disi.unibo.it/Courses/AmmSistemi1718/>
- Il *cheat sheet* regex alla fine è tratto dal sito: <http://addedbytes.com>

Indice

1	Linguaggio Bash	5
1.1	Variabili speciali	5
1.2	Escaping degli apici	5
1.3	Lettura Argomenti posizionali	6
1.4	Lettura Argomenti variabili	6
1.4.1	Lettura Argomenti variabili tramite array	6
1.5	Assegnamento condizionale ad una variabile	6
1.6	<code>getopts</code> : Definire opzioni da riga di comando	7
1.7	Piping e Ridirezione	7
1.7.1	Pipe	8
1.7.2	Ridirezione su file	8
1.7.3	Ridirigere lo standard error	8
1.7.4	<code>/dev/null</code> : Sopprimere l'output di un comando	8
2	Comandi di Base	10
2.1	<code>ls</code> : Elenca i file in una directory	10
2.2	<code>cd</code> : Cambiare il direttorio corrente	10
2.3	<code>ln</code> : Creare collegamenti a file o direttori	11
2.4	<code>printf</code> : Stampare i testi formattati	11
2.5	<code>pwd</code> : Ottenere il percorso del direttorio corrente	11
2.6	<code>du</code> : Visualizzare l'uso di spazio di un file o directory	11
2.7	<code>find</code> : Trovare un file	12
2.8	<code>history</code> : Richiamare comandi passati	13
2.9	<code>man</code> : Leggere la documentazione di un comando	13
3	Filtri	15
3.1	<code>grep</code> : Filtrare le righe in input che rispettano un pattern	15
3.1.1	Filtrare le righe che contengono un pattern	15
3.1.2	Usare <code>grep</code> in un <code>if</code> , per verificare se una variabile rispetta un pattern	15
3.1.3	Cercare ricorsivamente i file che contengono un pattern	16
3.2	<code>egrep</code> : Filtrare le righe in input con le REGEX	16
3.3	<code>cut</code> : Estrarre parti di righe	16
3.3.1	Estrarre range di caratteri	16
3.3.2	Estrarre dei campi da una riga	16
3.4	<code>sort</code> : Ordinare le righe	17
3.5	<code>uniq</code> : Rimuovere i duplicati (contigui)	17
3.6	<code>head</code> : Mostra le prime N righe di un file	18
3.7	<code>tail</code> : Mostra le ultime N righe di un file	18
3.8	<code>sed</code> : String Replace	18
3.9	<code>awk</code> : Estrarre campi da una riga	19
4	Processi e Segnali	20
4.1	<code>ps</code> : Ottenere informazioni sui processi attivi	20
4.1.1	Ottenere le informazioni a partire dal PID	20
4.1.2	Ottenere le informazioni a partire dal comando	20
4.1.3	Ottenere le informazioni dall'utente che l'ha lanciato	20
4.2	Segnali	21
4.2.1	<code>kill</code> : Inviare segnali ai processi	21
4.2.2	<code>trap</code> : Intercettare i segnali	21

4.3	Job control: Inviare processi in background e viceversa	21
4.3.1	Avviare un processo in background	22
4.3.2	Riportare un processo in foreground	22
4.3.3	Visualizzare tutti i jobs attivi	22
4.4	nohup: Far sopravvivere i processi alla chiusura della shell	22
4.5	fuser: Visualizza i processi che usano un file	22
4.5.1	Inviare un segnale a tutti i processi che usano un file	23
4.6	lsopf: Lista i file aperti nel sistema	23
5	Utenti e Gruppi	24
5.1	adduser: Creare un utente	24
5.1.1	Creare un utente con un gruppo specifico	24
5.2	addgroup: Creare un gruppo	24
5.3	Aggiungere un utente ad un gruppo	24
5.4	id: Visualizzare le informazioni di un utente	25
5.5	Visualizzare i gruppi di cui un utente fa parte	25
5.6	Vedere tutti i gruppi del sistema	25
5.7	passwd: Cambiare la password di un utente	25
5.8	whoami: Ottieni l'username dell'utente corrente	26
5.9	who: Visualizza gli utenti collegati alla macchina	26
6	Permessi	27
6.1	chmod: Cambiare i permessi di un file	28
6.2	chown: Cambiare l'ownership di un file	28
6.3	umask: Settare i permessi di default	29
6.3.1	Visualizzare la maschera corrente	29
6.3.2	Settare la maschera	29
7	Logging	30
7.1	logger: Loggare un messaggio	30
7.2	Impostare la destinazione dei Log	30
7.2.1	Salvare i log su file	30
7.2.2	Inviare i log ad un server remoto	31
8	Cron: Eseguire comandi periodicamente	32
8.1	Configurazioni di base	32
8.1.1	Esegui un job ogni giorno alle 2	32
8.1.2	Esegui un job due volte al giorno	33
8.1.3	Esegui un job ogni minuto	33
8.1.4	Esegui un job ogni domenica alle 5	33
8.1.5	Esegui un job ogni 10 minuti	33
8.1.6	Esegui un job nei mesi selezionati	33
8.1.7	Esegui un job nei giorni selezionati	33
8.1.8	Esegui un job la prima domenica di ogni mese	33
8.1.9	Esegui un job ogni 4 ore	33
8.1.10	Esegui un job due volte ogni domenica e lunedì	33
8.1.11	Esegui un job ogni 30 secondi	34
8.1.12	Esegui un più comandi con un solo job	34
8.1.13	Esegui un job una volta all'anno	34
8.1.14	Esegui un job una volta al mese	34
8.1.15	Esegui un job una volta alla settimana	34
8.1.16	Esegui un job ogni giorno	34
8.1.17	Esegui un job ogni ora	34
8.1.18	Esegui un job al riavvio del sistema	34
8.2	Editare crontab da uno script bash	35
8.2.1	Aggiungere un comando	35
8.2.2	Rimuovere un comando	35
8.3	Correggere il PATH in crontab	35

9 At: Posticipare l'esecuzione di un comando	36
9.1 Visualizzare i task in programma	36
9.2 Rimuovere un lavoro dalla coda	36
10 Configurazioni di Rete	37
10.1 Verificare che un nodo sia connesso ad un altro	37
10.1.1 ping	37
10.1.2 tcpdump	37
10.2 Configurazione IP a Runtime	37
10.2.1 Address	37
10.2.2 Route	38
10.3 Configurazione IP Permanente	38
10.4 Configurazione IP Forwarding	38
10.4.1 Client	38
10.4.2 Router	39
10.4.3 Server	40
10.5 ss: Ottenere informazioni sulle socket	40
11 ssh: Secure Shell	41
11.1 Eseguire remotamente un comando	41
11.2 Variabili d'ambiente SSH	41
11.3 Configurare il login tramite chiave pubblica	41
11.3.1 Configurare la macchina HOST	41
11.3.2 Configurare la macchina REMOTA	42
12 iptables: Stateful packet filtering	43
12.1 Tabella FILTER: Funzionalità di firewall	43
12.1.1 Visualizzare le regole attuali	44
12.1.2 Aggiungere nuove regole	44
12.1.3 Abilitare LOOPBACK e SSH	45
12.1.4 Eliminare una regola	45
12.1.5 Impostare le policy di default	45
12.1.6 Loggare pacchetti con iptables	46
12.1.7 Rilevare inizio e fine di una connessione TCP	46
12.2 Tabella NAT: Effettuare il NAT sui pacchetti in transito	46
12.2.1 Destination NAT	46
12.2.2 Source NAT	47
12.3 Chain personalizzate	48
12.4 Salvare e ripristinare le regole di iptables	48
13 tcpdump: Sniffing dei pacchetti	49
13.1 Principali opzioni	49
13.2 Filtrare i pacchetti	49
13.2.1 Filtrare i pacchetti in base alla destinazione o alla sorgente	49
13.2.2 Filtra i pacchetti in base al network	50
13.2.3 Filtra il traffico legato ad una porta	50
13.2.4 Filtra il traffico legato ad un protocollo	50
13.2.5 Filtra il traffico in base alla dimensione del pacchetto	50
13.3 Combinare i filtri	50
13.4 Catturare i pacchetti con flag TCP particolari	51
14 SNMP: Simple Network Management Protocol	52
14.1 Configurare il demone SNMPD	52
14.2 Visualizzare tutte le entry	52
14.3 Ottenere il valore di una entry	53
14.4 Estendere le funzionalità di SNMP	53
14.4.1 Leggere il valore di un campo personalizzato	53
14.4.2 Estendere la funzionalità con un comando root	53
14.5 Ottenere informazioni riguardo un processo	54
14.5.1 Conteggio del numero di istanze di un processo	54

15 LDAP: Lightweight Directory Access Protocol	56
15.1 Definire uno schema	56
15.1.1 Esempio con classi Strutturali	56
15.1.2 Esempio con classi Ausiliarie	57
15.1.3 Tipi di dato	57
15.2 Eliminare uno schema	58
15.3 Interrogare la directory	58
15.4 Aggiungere una entry	59
15.4.1 Aggiungere entry con classe Strutturale	59
15.4.2 Aggiungere entry con classe Ausiliaria	59
15.5 Modificare una entry	60
15.6 Eliminare una entry	60
Appendice: Cheat sheet Regex	60

Capitolo 1

Linguaggio Bash

1.1 Variabili speciali [↗](#)

All'interno di uno script bash è possibile utilizzare una serie di variabili built-in:

\$\$	PID dello script bash corrente
\$#	Numero di argomenti da riga di comando
\$?	Exit value dell'ultimo comando eseguito command.
\$	PID della shell
\$_	PID dell'ultimo comando eseguito in background
\$0	Nome script search.
\$n	N-esimo argomento posizionale. Il massimo numero è 9
*, @\$	Tutti gli argomenti passati.
"\$*"	Tutti gli argomenti in un'unica stringa "\$1 \$2 ...". I valori sono separati da IFS.
"\$@"	Tutti gli argomenti, separati individualmente ("\$1" "\$2" ...).

1.2 Escaping degli apici [↗](#)

Nella shell bash, i singoli apici e i doppi apici assumono significati particolari e ben distinti:

- **Singoli apici:** Preservano tutto il contenuto, senza effettuare alcuna valutazione.
- **Doppi apici:** Preservano il contenuto, ad eccezione dei caratteri: \$, ', \, ! che vengono invece interpretati.

```
USER=pippo
echo "Ciao $USER"      # Stampa: Ciao pippo
echo 'Ciao $USER'     # Stampa: Ciao $USER
echo "$(echo ciao)"   # Stampa: ciao
echo '$(echo ciao)'   # Stampa: $(echo ciao)
```

Info

Un'interessante proprietà degli apici è che possono essere concatenati in un'unica stringa. Questo può essere molto utile per formare argomenti che includono variabili esterne ma che vanno anche protetti.

```
echo "ciao" 'ciao'    # Stampa: ciaociao

USER=pippo
echo "ciao $USER" 'ciao $USER' # Stampa: ciao pippociao $USER

# Esempio un po' piu' complesso:
ssh "$1" "echo $2 $3 > '/tmp/$(echo $SSH_CLIENT | cut -f1 -d" ")'
```

1.3 Lettura Argomenti posizionali [↗](#)

Per leggere gli argomenti posizionali, è possibile utilizzare la notazione \$1, \$2, ecc.

```
# Stampo il primo argomento
# Invoco il comando come: ./script.sh ciao
echo "Il primo argomento e' $1"      # Output: Il primo argomento e' ciao
```

Un utilizzo interessante degli argomenti posizionali è l'utilizzo di \$0 per ottenere il nome del file bash corrente. Questo può essere utile per chiamarlo ricorsivamente.

1.4 Lettura Argomenti variabili [↗](#)

Può capitare di dover leggere un numero di argomenti variabili. Il miglior modo per farlo è utilizzare \$@ all'interno di un ciclo for

```
# Stampo tutti gli argomenti
for arg in "$@" ; do
    echo $arg
done
```

Info

Da notare un paio di cose:

- I doppi apici intorno a "\$@", che permettono di proteggere l'espansione da effetti indesiderati
- \$@ permette di ottenere anche gli argomenti che contengono degli spazi, a patto che durante la chiamata siano stati utilizzati i doppi apici. Esempio: ./script.sh "ciao ciao" test produrrà due argomenti: "ciao ciao" e "test".

1.4.1 Lettura Argomenti variabili tramite array [↗](#)

È anche possibile leggere gli argomenti trasferendoli direttamente in un array:

```
declare -a ARGS # Dichiaro l'array

# Riempio l'array con gli argomenti
ARGS=("$@")

# Ciclo tra tutti gli elementi
for index in ${!ARGS[@]} ; do
    ARG=${ARGS[$index]} # Elemento corrente
done
```

1.5 Assegnamento condizionale ad una variabile [↗](#)

A volte può essere comodo dare dei valori di default alle variabili, in caso non sia stato passato l'argomento corrispondente:


```
# Se $1 non e' specificato, setta la variabile a "pippo"
NOME=${1:-"pippo"}

# Se HOME non e' settata o e' nulla, la setta a /tmp
cd ${HOME:=/tmp}

# Stampa un errore ed esce se $2 non e' definito
NOME=${2:? "Errore, due non definito"}
```

1.6 getopt: Definire opzioni da riga di comando

`getopts` permette di definire delle opzioni da riga di comando, utilizzabili in uno script bash.

Il fulcro del meccanismo è definire la stringa delle opzioni, ad esempio: "ab:". In questo caso, visto che `b` è seguita da ":", `getopts` si aspetta che **debba avere anche un valore**, che verrà poi salvato nella variabile `$OPTARG`.

Se volessimo che anche `a` abbia un valore, la stringa diventerebbe: "a:b:"

Segue un esempio:

```
#!/bin/bash
aflag=
bflag=
bval=
while getopt 'ab:' OPTION ; do
    case $OPTION in
        a)      aflag=1
                ;;
        b)      bflag=1
                bval="$OPTARG"
                ;;
        ?)      echo "Errore argomenti"
                exit 2
    esac
done
# Per ottenere gli argomenti, shift via le opzioni
shift $(( $OPTIND - 1 ))

if [ "$aflag" ] ; then
    echo "Fai qualcosa con a"
fi
if [ "$bflag" ] ; then
    echo "B valeva: $bval"
fi
```

Ricordati

Le opzioni di `getopts` devono essere **monocarattere**, ovvero le opzioni estese non sono supportate.

1.7 Piping e Ridirezione

Ogni comando unix ha la possibilità di interagire con il sistema attraverso tre flussi:

- Standard Input: Flusso di dati in ingresso
- Standard Output: Flusso di dati in uscita
- Standard Error: Flusso di dati in uscita **che indicano un errore**

1.7.1 Pipe [↗](#)

Su unix è possibile ridirigere il flusso di dati in uscita da un processo all'ingresso di un altro e, per farlo, si usa una **pipe**.

La pipe viene indicata con la barra verticale | e si interpone tra i due processi comunicanti, ad esempio:

```
# Legge il file ciao.txt e ne inoltra il contenuto al comando less,  
# che ne mostra il contenuto in maniera progressiva  
cat ciao.txt | less
```

Info

Se si vuole inviare in piping anche lo standard error di un comando, si può utilizzare 2>&1 in questo modo:

```
# Inoltra sia lo standard output che lo standard error a less  
ls 2>&1 | less
```

1.7.2 Ridirezione su file [↗](#)

Per salvare l'output di un comando in un file, si può utilizzare la ridirezione:

```
# Salvo il risultato del comando ls nel file ciao.txt  
ls > ciao.txt  
  
# Come nel caso precedente, ma in APPEND al file  
ls >> ciao.txt
```

1.7.3 Ridirigere lo standard error [↗](#)

Nel caso in cui si voglia salvare su file lo standard error in uscita da un comando, si può utilizzare la sintassi: 2> oppure 2>>

```
# Salvo il risultato del comando ls nel file ciao.txt  
# ed eventuali errori nel file errori.txt  
ls > ciao.txt 2> errori.txt
```

Info

Nel caso in cui si voglia unire lo standard output allo standard error, si può usare la sintassi: 2>&1

```
# Salvo sia lo standard output che lo standard error nel file ciao.txt  
ls > ciao.txt 2>&1
```

1.7.4 /dev/null: Sopprimere l'output di un comando [↗](#)

Può essere comodo sopprimere l'output di un comando per non mostrarlo a video. Per farlo, si può redirigere il flusso dello standard output al file /dev/null

```
# Sopprimo l'output del comando  
ls 1>/dev/null
```

Info

Analogamente si può sopprimere lo standard error:

```
# Sopprimo lo standard error del comando ( ma stampo l'output )  
ls 2>/dev/null
```

Capitolo 2

Comandi di Base

In questo capitolo verranno spiegati i principali comandi utilizzati nella shell linux.

2.1 ls: Elenca i file in una directory [🔗](#)

Permette di elencare i file o il contenuto della directory specificata (se non specificata, directory corrente).

```
ls # Elenca il contenuto della directory corrente

Prevede molte opzioni:

-l # Abbina al nome le informazioni associate al file
-h # Mostra la dimensione del file in versione piu leggibile
-a # Mostra tutto ( includendo anche i file che iniziano con . )
-A # Come -a, ma escludendo .. e .
-R # Percorre ricorsivamente la gerarchia
-r # Inverte l'ordine dell'elenco
-t # Ordina i file in base all'ora di modifica ( dal piu recente )
-i # Indica gli i-number dei file
-F # Aggiunge alla fine del filename * agli eseguibili e / ai direttori
-d # Mostra le informazioni di un direttorio senza listare contenuto.
-X # Lista alfabeticamente in base all'estensione
--full-time # Mostra la data di modifica completa (data, ora, sec, ecc)
```

Ordinare i file [🔗](#)

Il comando ls offre la possibilità di ordinare i file secondo vari criteri, specificando l'argomento --sort

```
ls --sort=extension # Ordina in base all'estensione
ls --sort=size      # Ordina in base alla dimensione del file
ls --sort=time      # Ordina in base alla data di ultima modifica
```

L'ordine può essere invertito utilizzando l'argomento -r

2.2 cd: Cambiare il direttorio corrente [🔗](#)

Serve per navigare all'interno dei direttori. Può essere utilizzato in vari modi:

```
cd # Torna nella home dell'utente corrente
cd /path/della/directory # Naviga nel path "/path/della/directory"
cd .. # Naviga nel direttorio padre
cd - # Torna nel direttorio precedente
```

Info

In caso abbiate appena creato una directory, è molto comoda la sintassi:

```
mkdir ciao      # Crea la directory
cd !$          # Entra nella directory appena creata
```

2.3 ln: Creare collegamenti a file o direttori [↗](#)

```
# Crea un hardlink
ln /path/to/file /path/to/hardlink

# Crea un link simbolico
ln -s /path/to/file /path/to/link
```

2.4 printf: Stampare i testi formattati [↗](#)

Utilizzando il comando `printf` è possibile stampare dei testi formattati in maniera analoga alla funzione C omonima.

```
# La sintassi del comando e' la seguente:
printf <FORMATO> <ARGOMENTI>

# Output: Nome: pippo, eta: 16
printf "Nome: %s, eta: %d" pippo 16
```

`printf` supporta una notevole quantità di formati, i principali sono:

- `%s` Stringa
- `%d` Intero
- `%x` Intero in forma esadecimale
- `%o` Intero in forma ottale
- `%f` Float, con precisione di 6 caratteri

2.5 pwd: Ottenere il percorso del direttorio corrente [↗](#)

Utilizzato per stampare a video il direttorio corrente, ad esempio:

```
pwd # Stampa in output il percorso assoluto del direttorio corrente
```

2.6 du: Visualizzare l'uso di spazio di un file o directory [↗](#)

Viene utilizzato per ottenere informazioni riguardo all'uso di spazio da parte di un file o di una cartella.

```
du /path/directory      # Stampa la dimensione di ogni nodo dell'albero
Opzioni principali:
-h                      # Mostra le dimensioni in modo facilmente leggibile
-s                      # Mostra il sommario ( totale ) della directory
--max-depth             # Massima profondita' di esplorazione
--exclude="*.txt"      # Esclude tutti i file che rispettano il pattern.
                       # NON supporta le regular expressions
```

Info

Alcuni esempi d'uso con altri comandi:

```
# Mostra i 10 file piu voluminosi nel direttorio corrente
du | sort -nr | head -10
```

2.7 find: Trovare un file [↗](#)

Permette di trovare file e directory all'interno del sistema, in base ad un gran numero di opzioni.

```
# Trovare tutti i file che finiscono per .txt nella directory specificata
find /percorso/directory -name *.txt

# Trovare tutti i file con dimensione superiore a 100k
find -size +100k

# Trovare tutti i file dell'utente las
find -user las

# Trovare tutti i file del gruppo las
find -group las

# Trovare tutti i direttori
find -type d
```

Cercare in base ai permessi [↗](#)

```
# Trovare tutti i file che possono essere eseguiti dal proprietario
# e dal gruppo.
find -type f -perm -110

# Trovare tutti i file che hanno esattamente il permesso 110
find -type f -perm /110
```

Qual'è la differenza tra i due?

La differenza tra i due è che il primo controlla che un file rispetti quelle condizioni, **ma non impone niente sulle altre**. Verrebbero anche indicati i file con permessi di lettura e scrittura. Il secondo invece controlla che i permessi del file siano **esattamente** quelli specificati.

Eseguire un comando per ogni risultato [↗](#)

```
# Stampare le informazioni di ogni file trovato tramite ls
find . -exec ls -lh {} +

# Un altro modo per realizzare questo meccanismo e' utilizzare xargs
find . | xargs ls -lh
```

Info

Per cercare i file che contengono un particolare contenuto di file, vedere il comando `grep` con opzione `-Rl`

Effettuare una ricerca con regex [↗](#)

```
# Cercare tutti i file il cui nome finisce con .jpg, con una regex egrep
find . -regextype posix-egrep -regex '.*\.jpg'
```

Combinare le opzioni [↗](#)

Le opzioni possono essere combinate con operazioni logiche di AND, OR e NOT.

```
# NOT: basta aggiungere ! prima dell'opzione.
# Trovare tutti i file che non finiscono per .txt
find ! -name *.txt

# AND: basta inserire due comandi, sono automaticamente messi in AND.
# Trovare tutti i file con dimensione superiore a 100k e
# che finiscono per .txt
find -size +100k -name *.txt

# OR: bisogna inserire l'opzione -o prima del secondo comando
# Trovare tutti i file dell'utente las o che finiscono per .txt
find -user las -o -name *.txt
```

2.8 history: Richiamare comandi passati [↗](#)

Per visualizzare la lista dei comandi eseguiti precedentemente, può essere utilizzato il comando `history`. A quel punto, basta ricordarsi il numero a fianco del comando desiderato e digitare:

```
!NUMERO # Ad esempio: !5
```

Ricerca interattiva tramite CTRL + R [↗](#)

Per cercare interattivamente attraverso la history è possibile utilizzare la scorciatoia: CTRL + R. Questa mostrerà il comando più recente che contiene il termine di ricerca e premendo nuovamente CTRL + R sarà possibile cercare a ritroso nella cronologia.

2.9 man: Leggere la documentazione di un comando [↗](#)

Il comando `man` permette di consultare la documentazione di un qualunque comando linux. Per utilizzarlo, bisogna digitare nel terminale:

```
man <NOME_COMANDO >
```

Sezioni del man

Il `man` contiene varie sezioni di interesse per differenziare omonimi appartenenti a contesti diversi. Le sezioni principali sono:

1. User commands
2. Chiamate al sistema operativo
3. Funzioni di libreria
4. File speciali (`/dev/*`)
5. Formati dei file, protocolli e strutture C
6. Giochi
7. Macro, header, filesystem, concetti generali
8. Comandi amministrativi riservati a root

Per visualizzare l'entry di una particolare sezione, il formato sarà:

```
man <NUMERO_SEZIONE> <NOME_COMANDO>
```

È anche possibile cercare contemporaneamente in tutte le sezioni, digitando:

```
man -a <NOME_COMANDO>
```


Capitolo 3

Filtri

3.1 grep: Filtrare le righe in input che rispettano un pattern

`grep` è uno strumento estremamente potente per filtrare e cercare dei pattern all'interno di file o dello standard input. Possiede anche un *fratello maggiore*, `egrep`, totalmente compatibile con `grep` ma con l'aggiunta del supporto alle *regex*.

Opzioni:

```
-v      # Inverti il match ( escludi le righe che soddisfano il pattern )
-c      # Conteggio del numero di righe che soddisfano il pattern
-i      # Effettua la ricerca CASE-INSENSITIVE
-w      # Cerca parole intere e NON sottostringhe
-l      # Mostra solo i nomi dei file ( senza righe con match )
-n      # Mostra i numeri di riga dei match
-o      # Mostra solo la stringa che fa match, e non la riga intera
-B <N> # Stampa anche le N righe precedenti al match
-A <N> # Stampa anche le N righe successive al match
--line-buffered # Disabilita l'output buffer e stampa subito le righe
```

3.1.1 Filtrare le righe che contengono un pattern

```
# Stampo le righe di un file che contengono la parola "ciao"
grep -e "ciao" filename.txt
# Completamente equivalente a questo
cat filename.txt | grep -e "ciao"

# Utilizzando egrep e' possibile utilizzare le regex per la ricerca
# Cerca tutte le righe che iniziano con "ciao"
egrep -e "^ciao.*" filename.txt
```

3.1.2 Usare grep in un if, per verificare se una variabile rispetta un pattern

Il valore di ritorno di `grep` è 0 il pattern è stato trovato, 1 altrimenti. E' quindi possibile inserirlo direttamente in un `if` per verificare un pattern. In questo caso tuttavia è comoda l'opzione `-q` che permette di **non** stampare a video l'output di `grep`.

```
A=ciaone

# Controllo se la variabile A contiene "ciao"
if echo $A | grep -q "ciao" ; then
echo A contiene ciao
fi
```

3.1.3 Cercare ricorsivamente i file che contengono un pattern [↗](#)

Uno degli usi più comuni di `grep` è quello di effettuare una ricerca ricorsiva per trovare tutti i file che contengono un particolare pattern.

```
# Elenca tutti i file che contengono "ciao" analizzando
# ricorsivamente il direttorio corrente.
grep -r "ciao" .

# Per restituire solo i nomi dei file senza le righe
# che fanno match, usare l'opzione -l
grep -r -l "ciao" .
```

3.2 `egrep`: Filtrare le righe in input con le REGEX [↗](#)

`egrep` è il *fratello maggiore* di `grep` e, pur rimandando compatibile con suoi comandi e opzioni, aggiunge il supporto alle `regex`.

Esempio:

```
# Elenca le righe esattamente uguali a "ciao" nel file ciao.txt
egrep '^ciao$' ciao.txt
```

Info

In fondo alla guida è incluso un **cheat sheet** Regex, molto comodo per costruire le varie espressioni.

3.3 `cut`: Estrarre parti di righe [↗](#)

Il comando `cut` permette di estrarre parti di righe.

3.3.1 Estrarre range di caratteri [↗](#)

```
# Restituisci il decimo carattere di ogni riga del file
cut -c10 file.txt

# Restituisci i caratteri dal quinto al decimo
cut -c5-10 file.txt

# Restituisci i caratteri dall'inizio fino al 20esimo
cut -c-20 file.txt

# Restituisci i caratteri dal quinto in poi
cut -c5- file.txt
```

3.3.2 Estrarre dei campi da una riga [↗](#)

`cut` può essere utilizzato per estrarre dei campi da una riga se questi sono separati da un carattere:

```
cut -d<carattere_delimitatore> -f<elenco_campi>

# Estraggo il primo campo, separati da una virgola
cut -d, -f1

# Estraggo il primo ed il terzo campo, separati da uno spazio
cut -d' ' -f1,3
```

Info

Aggiungendo l'opzione `-s` a `cut`, si possono escludere tutte le righe che non contengono il delimitatore.

Info

Se si vogliono estrarre dei campi da una riga, può essere più comodo utilizzare il comando `awk`

3.4 sort: Ordinare le righe [↗](#)

Per ordinare una serie di righe di un file (o ricevute dallo standard input), si può utilizzare il comando `sort`:

```
# Ordina alfabeticamente le righe del file
sort file.txt

# Ordina numericamente le righe del file
sort -n file.txt

# Ordina le righe del file ed elimina i duplicati
sort -u file.txt

# Ordina in ordine inverso
sort -r file.txt
```

`sort` può anche essere utilizzato per ordinare delle righe secondo dei campi e non l'intera riga:

```
sort -t<delimitatore> -k <campo>,<campo>[n]

# Ordina degli indirizzi IP
sort -t . -k 1,1n -k 2,2n -k 3,3n -k 4,4n
```

3.5 uniq: Rimuovere i duplicati (contigui) [↗](#)

Il comando `uniq` permette di rimuovere le righe duplicate **contigue**.

```
# Elimina i duplicati CONTIGUI
uniq

# Indica il numero di occorrenze per ogni duplicato
uniq -c

# Mostra SOLO i duplicati
uniq -d
```

Info

È importante ricordarsi che `uniq` **non ha memoria**, questo implica che le righe, per essere rilevate come duplicate, devono essere **contigue**. Ad esempio:

```
ciao
pippo
ciao
```

Verrebbe lasciato passare inalterato da `uniq`, perchè i due `ciao` non sono contigui.

3.6 head: Mostra le prime N righe di un file [↗](#)

```
# Mostra solo le prime 2 righe del file
head -2 file.txt

# Mostra solo i primi 5 caratteri del file
head -c 5 file.txt
```

3.7 tail: Mostra le ultime N righe di un file [↗](#)

```
# Mostra le ultime 3 righe del file
tail -3 file.txt
```

Info

Può essere molto comoda l'opzione `-f` che permette di lasciare aperto un file e di visualizzare in tempo reale le aggiunte ad esso.

A riguardo, è comodo un altro *trucchetto* nel caso sia necessario visualizzare solo le righe che sono state aggiunte dall'avvio del comando in poi, e **non quelle già presenti nel file**:

```
# Non leggo nessuna riga dal file, ma ricevo quelle nuove
tail -n 0 -f file.txt
```

Inoltre può anche essere utile che `tail` termini quanto un altro processo termina, per questo usare l'opzione `--pid`

```
# Continua a leggere fino a che il processo 1234 non termina
tail -f --pid 1234
```

3.8 sed: String Replace [↗](#)

```
# Sostituisci la prima occorrenza di pippo con pluto per ogni riga
sed 's/pippo/pluto/' file.txt

# Sostituisci TUTTE le occorrenze di pippo con pluto
sed 's/pippo/pluto/g' file.txt

# Sostituisci tutte le occorrenze di pippo con pluto CASE INSENSITIVE
sed 's/pippo/pluto/gi' file.txt

# Aggiungi all'inizio di ogni riga "INIZIO: "
sed 's/^/INIZIO:/' file.txt
```

3.9 awk: Estrarre campi da una riga

awk è come cut, ma sotto steroidi. Permette di estrarre dei campi da una riga usando stringhe arbitrarie come separatori.

```
# Estrai il primo campo dalla riga, separato da uno spazio
awk '{ print $1 }'

# Estraggo il secondo campo, separato da una virgola
awk -F "," '{ print $2 }'

# Esempio di estrazione piu' complessa:
# INPUT: campo1=pippo campo2=pluto
# Vogliamo estrarre "pippo"
awk -F "campo1=" '{ print $2 }' | awk -F "campo2=" '{ print $1 }'
```

Info

In caso si vogliono filtrare dei campi in **real-time**, può essere comoda l'opzione `-W interactive` che evita il buffering.

Capitolo 4

Processi e Segnali

4.1 ps: Ottenere informazioni sui processi attivi [↗](#)

Il comando `ps` permette di ottenere tutte le informazioni legate ai processi correntemente in esecuzione. Principali opzioni:

```
ax    # Visualizza tutti i processi, anche non propri
u     # Mostra gli utenti proprietari del processo
w     # Visualizza la riga di comando completa che
      # ha originato il processo
f     # Visualizza i rapporti di discendenza tra i processi
-p    # Solo il processo con questi pid
```

4.1.1 Ottenere le informazioni a partire dal PID [↗](#)

`ps` permette di specificare uno o più PID relativi ai processi da visualizzare.

```
# Visualizza le informazioni relative al processo con PID=1234
ps -p 1234

# Visualizza le informazioni relative a piu processi
ps -p 1234,1235
```

4.1.2 Ottenere le informazioni a partire dal comando [↗](#)

A volte può essere più comodo cercare un processo a partire dal suo comando.

```
# Visualizza le informazioni relative al processo chrome
ps -C chrome
```

4.1.3 Ottenere le informazioni dall'utente che l'ha lanciato [↗](#)

Utilizzando l'opzione `U` è possibile visualizzare tutti i processi di un dato utente.

```
# Visualizza i processi dell'utente pippo
ps U pippo

# Visualizza i processi di pippo e pluto
ps U "pippo pluto"
```

4.2 Segnali [↗](#)

I segnali sono uno strumento che permette a due processi di comunicare. Un processo può infatti inviare ad un altro un segnale che lo avvisa di un determinato evento.

Inoltre, un processo può *registrare* presso il sistema operativo un *handler* apposito che, quando viene ricevuto il segnale associato, viene eseguito.

Nella tabella sottostante sono riportati i principali segnali:

SIGINT	2	Interrupt a process (used by Ctrl-C)
SIGHUP	6	Hang up or shut down and restart process
SIGKILL	9	Kill the process (cannot be ignored or caught elsewhere)
SIGTERM	15	Terminate signal, (can be ignored or caught)
SIGTSTP	20	Stop the terminal (used by Ctrl-z)
SIGSTOP	23	Stop execution (cannot be caught or ignored)

4.2.1 kill: Inviare segnali ai processi [↗](#)

Il comando `kill` può essere utilizzato per inviare un determinato segnale ad un processo.

```
# Invia un segnale SIGKILL al processo con PID=150
kill -SIGKILL 150
```

4.2.2 trap: Intercettare i segnali [↗](#)

`trap` è un built-in della shell che permette agli script `bash` di intercettare dei segnali e di definire degli *handler* personalizzati per gestirli.

La sintassi d'uso è:

```
# Esegue il comando echo quando lo script riceve il segnale USR1
trap 'echo ciao' USR1
```

Spesso si utilizza una funzione per creare l'handler, segue un esempio:

```
#!/bin/bash

# Definisco l'handler personalizzato
function logging () {
    case "$1" in
        start) echo started ;;
        stop)  echo stop      ;;
    esac
}

# Stoppa il logging quando viene ricevuto SIGUSR1
trap 'logging stop' USR1
logging start

while :
do
    # Eseguo le varie elaborazioni
    echo aspetto...
done
```

4.3 Job control: Inviare processi in background e viceversa [↗](#)

Da shell può essere molto comodo lanciare dei comandi in background ed eventualmente riportarli in foreground.

4.3.1 Avviare un processo in background [↗](#)

Per avviare un processo in background da shell, basta aggiungere `&` al termine del comando:

```
# Avvia il comando in background
vim &
```

La shell ci restituisce un **job id**, che potremo utilizzare successivamente per riportare il processo in foreground.

Info

Si può anche portare un processo in background dopo averlo avviato, basta premere `CTRL+Z` e successivamente digitare `bg <job_id>`

Spiegazione

Con `CTRL+Z` si invia al processo un segnale di `SIGSTOP` che lo blocca. A questo punto, con il comando `bg` si invia un segnale di `SIGCONT` che riavvia il processo in background.

4.3.2 Riportare un processo in foreground [↗](#)

Si può facilmente riportare un processo in foreground con il comando `fg`.

```
# Riporta il processo con il jobid specificato in foreground
fg <job_id>
```

4.3.3 Visualizzare tutti i jobs attivi [↗](#)

Per visualizzare l'elenco di tutti i jobs attivi con il loro stato (attivo o stoppato), si può utilizzare il comando `jobs`

```
# Elenca tutti i jobs con il loro stato.
jobs
```

4.4 nohup: Far sopravvivere i processi alla chiusura della shell [↗](#)

Normalmente, alla chiusura della shell, tutti i processi da lei generati ricevono un segnale di `SIGHUP` e vengono terminati. Per evitare questo comportamento si può utilizzare il comando `nohup`, che rende immune il processo all'hangup ed inoltre provvede a scollegare l'output di un processo.

```
# Esegue il comando in background e lo
# rende immune alla chiusura della shell
nohup <command> &
```

Info

A default, `nohup` ridirige l'output di un processo nel file `nohup.out`

4.5 fuser: Visualizza i processi che usano un file [↗](#)

```
# Visualizza i PID dei processi che usano un file
fuser /path/to/file

# Visualizza i processi che usano un intero filesystem
fuser -m /var
```


4.5.1 Inviare un segnale a tutti i processi che usano un file [↗](#)

Tramite l'opzione `-k` è possibile inviare un segnale a tutti i processi che usano un determinato file:

```
# Invia un segnale di SIGUSR1 a tutti i processi che usano il file
fuser -k -USR1 /path/to/file
```

Info

A causa di una formattazione particolare, analizzare l'output di `fuser` può essere particolarmente problematico usando `awk` o `cut`. In quei casi, conviene usare il comando `lsof`.

4.6 lsof: Lista i file aperti nel sistema [↗](#)

`lsof` permette di vedere tutti i file aperti nel sistema.

```
# Visualizza tutti i file aperti dal sistema
lsof

# Visualizza i file aperti da uno specifico user
lsof -u <username>

# Visualizza i file aperti da uno specifico processo
lsof -p <PID>

# Limitare l'output di lsof ad una sola directory
lsof +D /path/to/dir

# Listare tutti i file in base al tipo di connessione (tcp o udp)
lsof -i tcp

# IMPORTANTE: Visualizza le porte numeriche e non simboliche
lsof -i tcp -P
```

Capitolo 5

Utenti e Gruppi

5.1 adduser: Creare un utente [↗](#)

Per aggiungere un utente al sistema, si può utilizzare il comando `adduser`

```
# Aggiungi un utente al sistema
adduser <nomeutente>
```

Ricordati

Per utilizzare `adduser` sono necessari i permessi di root.

5.1.1 Creare un utente con un gruppo specifico [↗](#)

Normalmente, quando si crea un nuovo utente, questo viene aggiunto ad un nuovo gruppo chiamato come l'utente stesso. Per cambiare questo comportamento, si può utilizzare l'opzione `--ingroup` di `adduser`

```
# Se non esiste già', bisogna creare il gruppo
addgroup <nomegruppo>

# A questo punto possiamo creare un nuovo utente ed aggiungerlo al gruppo
adduser --ingroup <nomegruppo> <nomeutente>
```

5.2 addgroup: Creare un gruppo [↗](#)

Per creare un nuovo gruppo all'interno del sistema si può utilizzare il comando `addgroup`

```
# Creo un gruppo
addgroup <nomegruppo>
```

5.3 Aggiungere un utente ad un gruppo [↗](#)

Per aggiungere un utente ad un gruppo, si può utilizzare il comando `usermod` in questo modo:

```
usermod -a -G <nomegruppo> <nomeutente>
```

Info

Quando si usa `usermod` con opzione `-G`, si aggiunge un gruppo ai **gruppi secondari** dell'utente. Se si vuole cambiare il **gruppo primario**, bisogna utilizzare l'opzione `-g`

```
usermod -g <nomegruppo> <nomeutente>
```

5.4 id: Visualizzare le informazioni di un utente [↗](#)

Si può utilizzare il comando `id` per ottenere alcune informazioni di un utente, in particolare:

- User ID
- Group ID
- Gruppi secondari

```
# Visualizzo le info dell'utente corrente
id

# Visualizzo le info di un utente specifico
id <nomeutente>
```

5.5 Visualizzare i gruppi di cui un utente fa parte [↗](#)

Per farlo ci sono due modi, il più semplice è utilizzare `groups`

```
# Elenco i gruppi dell'utente corrente
groups

# Elenco i gruppi di un utente specifico
groups <nomeutente>
```

In caso si vogliono avere più informazioni (come i **group id**), è possibile utilizzare il comando `id` (vedi 5.4).

5.6 Vedere tutti i gruppi del sistema [↗](#)

```
# Stampa l'elenco di tutti i gruppi del sistema
getent group
```

5.7 passwd: Cambiare la password di un utente [↗](#)

Per cambiare la password di un utente, si può usare il comando `passwd`

```
# Cambia la password dell'utente corrente
passwd

# Cambia la password di un utente specifico
# NOTA: Necessari i permessi di root
passwd <nomeutente>
```

Info

Utilizzando rispettivamente le opzioni `-l` e `-u` è possibile settare un account in stato di lock o unlock. Solo root può farlo.

5.8 whoami: Ottieni l'username dell'utente corrente [↗](#)

Per ottenere l'username dell'utente corrente, si può utilizzare il comando `whoami`

```
# Stampa l'username dell'utente corrente
whoami
```

5.9 who: Visualizza gli utenti collegati alla macchina [↗](#)

Serve ad ottenere un elenco degli utenti correntemente collegati alla macchina

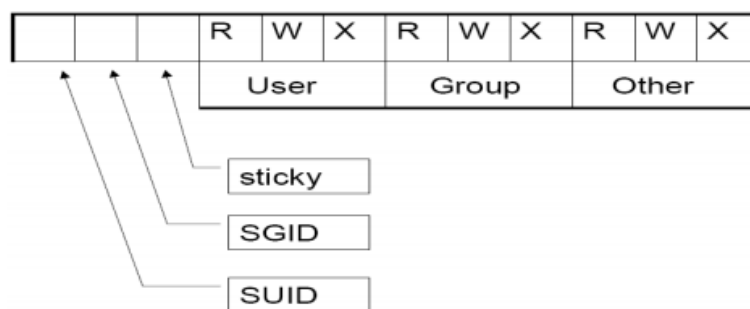
```
# Elenca gli utenti correntemente collegati alla macchina
who
```

Capitolo 6

Permessi

Su Unix **tutto è file** ed ognuno di essi è descritto da un identificatore, l'**i-node**. Tutti sono associati ad un set di 12 bit che rappresentano i vari permessi.

La seguente figura illustra la disposizione di questi 12 bit:



Dove, nel caso di **file**:

- R = read: Permesso di lettura di un file
- W = write: Permesso di scrittura dentro un file
- X = execute: Permesso di eseguire il file come programma

Mentre, nel caso di una **directory**:

- R = read: Permesso di elencare i file contenuti al suo interno.
- W = write: Permesso di aggiunta/cancellazione/rinomina di un file al suo interno.
- X = execute: Permesso di entrare nella directory (comando `cd`)

Info

Il permesso **write** in una directory permette ad un utente di cancellare qualunque file contenuto in essa, anche se l'utente non ha diritti sul file.

I 3 bit più significativi sono quelli **speciali** e configurano aspetti particolari:

Nel caso di **file**:

- SUID = Set User ID: In un file eseguibile, se settato a 1, permette di configurare il sistema operativo per **lanciarlo con l'identità dell'utente proprietario** invece che quella dell'utente che l'ha eseguito.
- SGID = Set Group ID: Analoga a SUID ma per i gruppi.
- STICKY BIT: Obsoleto al giorno d'oggi, suggerisce al sistema operativo di tenere una copia del programma in memoria.

Nel caso di **directory**:

- SUID = Set User ID: Non viene usato con le directory.

- SGID = Set Group ID:
 - Precondizioni:
 - * Un utente appartiene anche al gruppo proprietario della directory
 - * il SGID è settato a 1 sulla directory.
 - Effetto:
 - * L'utente assume come gruppo attivo il gruppo proprietario della directory
 - * I file creati nella directory hanno quello come gruppo proprietario.
 - Vantaggi (mantenendo umask 0006):
 - * Nelle aree collaborative i file sono automaticamente resi leggibili e scrivibili da tutti i membri del gruppo
 - * Nelle aree personali i file sono privati perchè di proprietà del gruppo principale dell'utente, che contiene solo l'utente medesimo.
- STICKY Temp: Se settato a 1 su una directory, impone che i file siano cancellabili solo dal rispettivo proprietario.

6.1 chmod: Cambiare i permessi di un file [↗](#)

Per impostare i permessi di un file o di una directory è possibile utilizzare il comando `chmod`

```
# Diamo a tutti il permesso di lettura
chmod a=r <nomefile>

# Aggiungiamo il permesso di scrittura ed esecuzione al proprietario
chmod u+wx <nomefile>

# Rimuoviamo i permessi di scrittura al gruppo proprietario
chmod g-w <nomefile>

# E' anche possibile utilizzare il formato numerico
chmod 0777 <nomefile>
```

6.2 chown: Cambiare l'ownership di un file [↗](#)

Per cambiare l'utente o il gruppo proprietario di un file si può utilizzare il comando `chown`.

```
# Cambia l'utente proprietario di un file
chown <username> <nomefile>

# Cambia il gruppo proprietario di un file
# NOTA: Attenzione ad aggiungere :
chown :<group> <nomefile>

# Cambia l'utente ed il gruppo proprietario
chown <username>:<group> <nomefile>

# Cambia la proprietà di tutti i file contenuti nella directory
# in maniera RICORSIVA
chown -R <username>:<group> <nomedirectory>
```

Ricordati

Per utilizzare con successo il comando `chown` in generale **servono i permessi di root.**

6.3 umask: Settare i permessi di default [↗](#)

Per fare in modo che una certa configurazione di permessi venga usata a default per tutti i nuovi file creati, è possibile utilizzare il comando `umask`. Questo permette di specificare la *maschera* di bit che verrà sottratta ai 12 bit dei permessi.

$$\begin{array}{r} 777 \\ -027 \text{ (umask)} \\ \hline 750 \text{ resulting permission} \end{array}$$

Diagram illustrating the calculation of the resulting permission (750) from the default permission (777) and the umask (027):

- 7 = read, write, execute permission for user
- 5 = read, execute permission for group
- no permission for other

6.3.1 Visualizzare la maschera corrente [↗](#)

```
# Visualizza la umask corrente
umask

# Per visualizzare i permessi a default correnti in maniera
# piu' comprensibile, si puo' usare l'opzione -S
umask -S
```

6.3.2 Settare la maschera [↗](#)

Per settare la maschera, si può utilizzare sempre il comando `umask`

```
# Imposta la maschera in modo che i nuovi file abbiano permesso 775
umask 0002
```

Ricordati

L'impostazione della `umask` è valida solo per la sessione di shell corrente. Per fare in modo che l'impostazione sia persistente, bisogna aggiungere il comando `umask` al file `/etc/bash.bashrc`

Capitolo 7

Logging

Per effettuare logging su linux, si può utilizzare `rsyslog`. Ogni messaggio di log è caratterizzato dalla coppia:

`<facility>.<priority>`

Dove `facility` rappresenta l'argomento e può essere:

- `auth`
- `authpriv`
- `cron`
- `daemon`
- `ftp`
- `kern`
- `lpr`
- `mail`
- `news`
- `syslog`
- `user`
- `uucp`
- `local0 .. local7`

Mentre `priority` rappresenta la priorità del messaggio e può essere:

- | | | | |
|-----------------------|----------------------|-------------------------|-----------------------|
| 1. <code>emerg</code> | 3. <code>crit</code> | 5. <code>warning</code> | 7. <code>info</code> |
| 2. <code>alert</code> | 4. <code>err</code> | 6. <code>notice</code> | 8. <code>debug</code> |

Dove il valore più basso rappresenta la priorità massima. Esempio: `local0.info`

7.1 logger: Loggere un messaggio [↗](#)

Utilizzando il comando `logger` è possibile loggare manualmente un messaggio, specificando la `facility` e la `priority`:

```
# Logga il messaggio "ciao" nella facility local3 con priority info
logger -p local3.info "ciao"
```

7.2 Impostare la destinazione dei Log [↗](#)

I vari log possono essere rediretti verso diverse destinazioni. Per farlo, bisogna creare un file `conf` all'interno della cartella `/etc/rsyslog.d/`, ad esempio `/etc/rsyslog.d/lab.conf`. All'interno di questo file è possibile specificare come destinazione dei log sia un file che un server remoto.

7.2.1 Salvare i log su file [↗](#)

E' possibile salvare una certa categoria di log all'interno di un file:


```
# Salva tutti i log di tipo local0 con qualunque priorit 
local0.*          /var/log/attivita

# Salva i log kernel da livello debug in su
kern.debug        /var/log/attivita

# Salva i log kernel con livello ESATTAMENTE UGUALE a debug
kern.=debug       /var/log/attivita

# NOTA: I due campi sono separati da un TAB
```

Ricordati

Al termine dell'operazione, riavviare rsyslog con il comando: `sudo systemctl restart rsyslog`

Ricordati

Se elimini direttamente un file di log, potrebbe capitare che il file non venga pi  ricreato. Per risolvere il problema, puoi riavviare il demone di rsyslog: `sudo systemctl restart rsyslog`

7.2.2 Inviare i log ad un server remoto [↗](#)

E' possibile inviare una certa categoria di log ad un server remoto, per farlo bisogna impostare correttamente sia il client che il server:

Configurare il Client

```
# Invia all'indirizzo 192.168.56.203 tutti i log di tipo local0
# con qualunque priorit 
local0.*          @192.168.56.203

# NOTA: I due campi sono separati da un TAB
```

Ricordati

Al termine dell'operazione, riavviare rsyslog con il comando: `sudo systemctl restart rsyslog`

Configurare il Server

Nel server pu  essere utile salvare i log ricevuti in un file, come nell'esempio prima:

```
# Salva tutti i log di tipo local0 con qualunque priorit 
local0.*          /var/log/attivita

# NOTA: I due campi sono separati da un TAB
```

Bisogna anche modificare il file `/etc/rsyslog.conf` per abilitare la ricezione dei log tramite UDP, decommentando:

```
$ModLoad imudp
$UDPServerRun 514
```

Ricordati

Al termine dell'operazione, riavviare rsyslog con il comando: `sudo systemctl restart rsyslog`

Capitolo 8

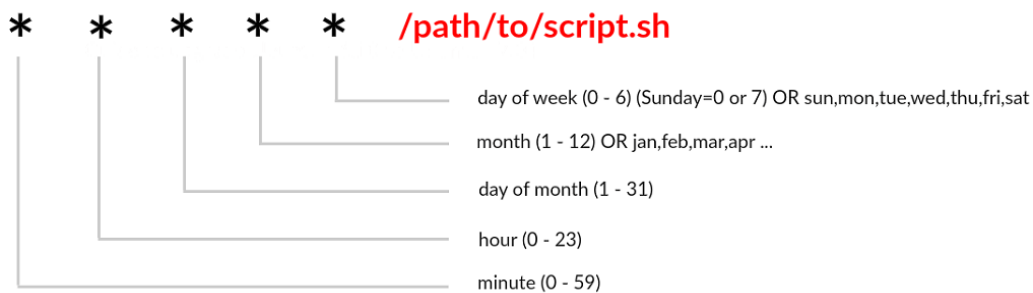
Cron: Eseguire comandi periodicamente

Per eseguire dei comandi periodicamente, è possibile utilizzare **Cron**. Per editare i cronjob da eseguire, si utilizza `crontab`

È possibile editare il file di configurazione ed aggiungere i vari comandi da eseguire utilizzando il comando:

```
crontab -e
```

Ad ogni comando corrisponde una riga, con il seguente formato:



- L'**asterisco** (*) fa match con tutti gli elementi
- E' possibile definire un **range di valori** con il simbolo -, come ad esempio: 1-10 oppure sun-fri
- E' possibile definire **multipli range di valori** utilizzando la virgola come separatore: jan-mar, jul-sep

Info

Di default, usando `crontab -e` viene editata la configurazione dell'utente corrente. Per **editare la configurazione di un altro utente** è possibile utilizzare: `crontab -u username -e`

Per listare tutti i cronjob, si può utilizzare il comando:

```
crontab -l
```

8.1 Configurazioni di base [↗](#)

Segue un elenco delle configurazioni di `crontab` più comuni:

8.1.1 Esegui un job ogni giorno alle 2 [↗](#)

```
0 2 * * * /bin/sh backup.sh
```

8.1.2 Esegui un job due volte al giorno [↗](#)

In questo caso, alle 5 di mattina e alle 5 di pomeriggio.

```
0 5,17 * * * /scripts/script.sh
```

8.1.3 Esegui un job ogni minuto [↗](#)

```
* * * * * /scripts/script.sh
```

8.1.4 Esegui un job ogni domenica alle 5 [↗](#)

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

8.1.5 Esegui un job ogni 10 minuti [↗](#)

```
*/10 * * * * /scripts/monitor.sh
```

8.1.6 Esegui un job nei mesi selezionati [↗](#)

In questo caso a gennaio, maggio ed agosto.

```
* * * jan,may,aug * /script/script.sh
```

8.1.7 Esegui un job nei giorni selezionati [↗](#)

In questo caso, la domenica ed il venerdì alle 5 di pomeriggio.

```
0 17 * * sun,fri /script/script.sh
```

8.1.8 Esegui un job la prima domenica di ogni mese [↗](#)

```
0 2 * * sun [ $(date +%d) -le 07 ] && /script/script.sh
```

8.1.9 Esegui un job ogni 4 ore [↗](#)

```
0 */4 * * * /scripts/script.sh
```

8.1.10 Esegui un job due volte ogni domenica e lunedì [↗](#)

```
0 4,17 * * sun,mon /scripts/script.sh
```

8.1.11 Esegui un job ogni 30 secondi [↗](#)

Eeguire un job con intervalli inferiori al minuto non è possibile nativamente con cron, ma si può usare questo trucco:

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

8.1.12 Esegui un più comandi con un solo job [↗](#)

Con il punto e virgola si possono concatenare più comandi.

```
* * * * * /scripts/script.sh; /scripts/scrit2.sh
```

8.1.13 Esegui un job una volta all'anno [↗](#)

```
@yearly /scripts/script.sh
```

8.1.14 Esegui un job una volta al mese [↗](#)

```
@monthly /scripts/script.sh
```

8.1.15 Esegui un job una volta alla settimana [↗](#)

```
@weekly /bin/script.sh
```

8.1.16 Esegui un job ogni giorno [↗](#)

```
@daily /scripts/script.sh
```

8.1.17 Esegui un job ogni ora [↗](#)

```
@hourly /scripts/script.sh
```

8.1.18 Esegui un job al riavvio del sistema [↗](#)

Questo job verrà eseguito all'avvio del sistema.

```
@reboot /scripts/script.sh
```

8.2 Editare crontab da uno script bash [↗](#)

8.2.1 Aggiungere un comando [↗](#)

La strategia migliore è quella di utilizzare un file temporaneo

```
# 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
```

8.2.2 Rimuovere un comando [↗](#)

La strategia si avvale di `grep` per filtrare via la riga da rimuovere.

```
# Filtro il crontab con grep -v
# poi lo invio a crontab -
# Che legge da standard input ed aggiorna cron
crontab -l | grep -v "echo ciao" | crontab -
```

8.3 Correggere il PATH in crontab [↗](#)

Quando si esegue uno script su crontab, è importante ricordarsi di scrivere i percorsi assoluti dei comandi, perchè cron non ha la conoscenza dei PATH dell'utente.

Un modo per superare questo inconveniente è definire il PATH all'interno del crontab file stesso. Editando crontab con `crontab -e` bisogna inserire le due variabili SHELL e PATH.

```
This has the username field, as used by /etc/crontab.
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the 'crontab'
# command to install the new version when you edit this file.
# This file also has a username field, that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
42 6 * * *    root    run-parts --report /etc/cron.daily
47 6 * * 7    root    run-parts --report /etc/cron.weekly
52 6 1 * *    root    run-parts --report /etc/cron.monthly
01 01 * * 1-5 root    python /path/to/file.py
```

Capitolo 9

At: Posticipare l'esecuzione di un comando

Tramite `at` è possibile posticipare un comando in modo che venga eseguito successivamente. La sintassi di base è la seguente:

```
# Apre un prompt interattivo per inserire il comando
# da chiudere con Ctrl+D
at <time>

# Da notare che per impostare il comando si puo' usare una pipe
echo 'echo ciao' | at <time>
```

Il valore del tempo può assumere diversi valori, seguono alcuni esempi:

- "9:00 AM" : alle 9 di mattina
- "now + 30 minutes" : 30 minuti da adesso
- "now + 30 minutes" : 30 minuti da adesso
- "now + 2 days" : 2 giorni da adesso

9.1 Visualizzare i task in programma [↗](#)

Utilizzando il comando `atq` si possono visualizzare i programmi correntemente in coda.

Info

Normalmente `atq` restituisce i comandi nella coda dell'utente ma, se **eseguito con permessi di root**, mostra i comandi nelle code di tutti gli utenti.

9.2 Rimuovere un lavoro dalla coda [↗](#)

Utilizzando `atrm` si può rimuovere un comando dalla coda, la sintassi è la seguente:

```
# Rimuovo il job con Id 7
atrm 7
```

Info

Per ottenere l'id del job da eliminare si può utilizzare `atq` per visionare l'elenco.

Capitolo 10

Configurazioni di Rete

10.1 Verificare che un nodo sia connesso ad un altro [↗](#)

10.1.1 ping [↗](#)

Il modo più semplice per verificare che un nodo sia connesso ad un altro, e quindi che lo possa raggiungere, è utilizzare il comando `ping`

Se ad esempio volessimo contattare il server che si trova all'indirizzo `192.168.56.203`, potremmo, dal client, digitare il comando: `ping 192.168.56.203`

Se il comando fornisce un output continuo, significa che il nodo è raggiungibile.

10.1.2 tcpdump [↗](#)

Allo stesso modo, nel nodo server, è possibile visualizzare le richieste in arrivo con il comando `tcpdump`

Individuata l'interfaccia di rete del server, in questo caso `eth3`, possiamo digitare `tcpdump -i eth3 -l -n` per visualizzare in tempo reale le richieste ICMP del ping.

Info

Per maggiori informazioni su `tcpdump`, guardare il capitolo 13.

10.2 Configurazione IP a Runtime [↗](#)

10.2.1 Address [↗](#)

```
# Visualizzazione configurazioni IP
ip a

# Assegnazione IP
ip a add <address>/<mask> dev <interface>

# Rimozione IP
ip a del <address>/<mask> dev <interface>
```

10.2.2 Route [↗](#)

```
# Visualizzazione di tutte le Routes
ip r

# Routing via Gateway
ip r add <dst_net>/<mask> via <gw_addr>

# Routing via Interfaccia di rete
ip r add <dst_net>/<mask> dev <interface>

# Rimozione
ip a del <address>/<mask>
```

10.3 Configurazione IP Permanente [↗](#)

Bisogna editare il file `/etc/network/interfaces` ed un tipico esempio è il seguente:

```
auto eth0                                # Attiva con ifup -a
iface eth0 inet static                   # Ip statico definito manualmente
                                         # se usate dhcp invece che static,
                                         # fa in automatico.

    address 192.168.56.203
    netmask 255.255.255.0
    gateway 192.168.56.1
    up /path/to/command                  # Eseguito dopo la configurazione
```

Ricordati

Dopo la modifica, bisogna riavviare il modulo di networking.

```
sudo systemctl stop networking
sudo systemctl start networking
```

Se qualche interfaccia non riparte, usare:

```
ifup <nome_interfaccia>
```

Se, dopo averle rimosse, restano delle interfacce ancora attive, utilizzare:

```
ip addr del <indirizzo>/<mask> dev <nome_interfaccia>
```

10.4 Configurazione IP Forwarding [↗](#)

E' possibile configurare l'IP forwarding in maniera tale che il **client** possa comunicare con il **server** tramite il **router**, come in figura 10.1.

Per farlo, è necessario configurare le tre macchine come segue:

10.4.1 Client [↗](#)

Editando il file `/etc/network/interfaces`

```
auto eth2
iface eth2 inet static
    address 10.1.1.1
    netmask 255.255.255.0
    up ip route add 10.9.9.0/24 via 10.1.1.254
```

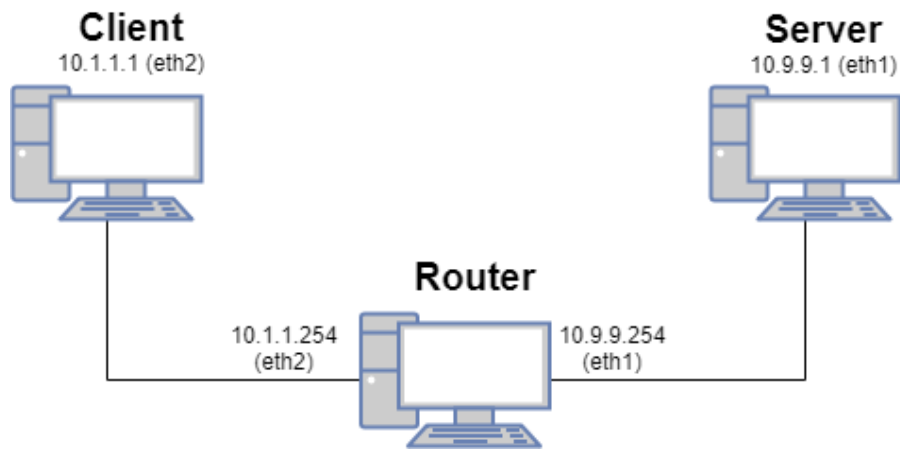



Figura 10.1: Architettura forwarding

Ricordati

Dopo la modifica, bisogna riavviare il modulo di networking.

```
sudo systemctl stop networking
sudo systemctl start networking
```

Se qualche interfaccia non riparte, usare:

```
ifup <nome_interfaccia>
```

Se, dopo averle rimosse, restano delle interfacce ancora attive, utilizzare:

```
ip addr del <indirizzo>/<mask> dev <nome_interfaccia>
```

10.4.2 Router [↗](#)

Editando il file `/etc/network/interfaces`

```
auto eth1
iface eth1 inet static
    address 10.9.9.254
    netmask 255.255.255.0

auto eth2
iface eth2 inet static
    address 10.1.1.254
    netmask 255.255.255.0
```

Ricordati

Dopo la modifica, bisogna riavviare il modulo di networking.

```
sudo systemctl stop networking
sudo systemctl start networking
```

Se qualche interfaccia non riparte, usare:

```
ifup <nome_interfaccia>
```

Se, dopo averle rimosse, restano delle interfacce ancora attive, utilizzare:

```
ip addr del <indirizzo>/<mask> dev <nome_interfaccia>
```

Bisogna abilitare il forwarding, editando il file `/etc/sysctl.conf` ed assicurandosi che sia presente, non commentato, questo comando:

```
net.ipv4.ip_forward=1
```

Ricordati

Dopo la modifica, bisogna lanciare:

```
sudo sysctl -p
```

10.4.3 Server [🔗](#)

Editando il file `/etc/network/interfaces`

```
auto eth1
iface eth1 inet static
    address 10.9.9.1
    netmask 255.255.255.0
    up ip route add 10.1.1.0/24 via 10.9.9.254
```

Ricordati

Dopo la modifica, bisogna riavviare il modulo di networking.

```
sudo systemctl stop networking
sudo systemctl start networking
```

Se qualche interfaccia non riparte, usare:

```
ifup <nome_interfaccia>
```

Se, dopo averle rimosse, restano delle interfacce ancora attive, utilizzare:

```
ip addr del <indirizzo>/<mask> dev <nome_interfaccia>
```

10.5 ss: Ottenere informazioni sulle socket [🔗](#)

`ss` è un comodo strumento che permette di ottenere delle informazioni riguardo all'utilizzo di socket nel sistem:

```
# Visualizza le connessioni tcp nel sistema, visualizzando anche
# i processi e gli utenti coinvolti
ss -tpn

# Con l'udp invece
ss -upn
```

Capitolo 11

ssh: Secure Shell

Il comando `ssh` permette di ottenere il terminale di una macchina remota.

11.1 Eseguire remotamente un comando [↗](#)

Oltre a poter aprire una shell interattiva, `ssh` permette anche di inviare singoli comandi alla macchina remota:

```
# Eseguo il comando ls sulla macchina remota
ssh las@192.168.56.201 ls
```

11.2 Variabili d'ambiente SSH [↗](#)

All'interno di una shell `ssh` sono presenti alcune variabili di sistema che permettono di ottenere delle informazioni riguardo alla connessione corrente:

```
# Ad esempio
SSH_CLIENT='192.168.56.1 56156 22'
SSH_CONNECTION='192.168.56.1 56156 192.168.56.201 22'
SSH_TTY=/dev/pts/0
```

11.3 Configurare il login tramite chiave pubblica [↗](#)

Configurando l'accesso con chiave pubblica permette di effettuare il login con `ssh` senza digitare la password.

11.3.1 Configurare la macchina HOST [↗](#)

Sulla macchina HOST dovremo generare una coppia di chiavi (pubblica + privata) e poi inviare la propria chiave pubblica alla macchina remota.

```
# Genero la coppia di chiavi
# NOTA: premere invio ad ogni domanda
ssh-keygen -t rsa -b 2048

# Invio alla macchina remota la chiave pubblica
# NOTA: Cambiare l'indirizzo di destinazione all'occorrenza
scp .ssh/id_rsa.pub las@192.168.56.201:
```

Ricordati

Per eseguire correttamente il login remoto, assicurati di avere nella cartella `.ssh` il file `id_rsa`. Questo viene automaticamente generato da `ssh-keygen` e rappresenta la tua chiave privata.

Info

In caso sia necessario utilizzare una chiave diversa da quella di default: `id_rsa` è possibile utilizzare l'opzione `-i` e specificare un file per la chiave privata.

11.3.2 Configurare la macchina REMOTA [↗](#)

Una volta ricevuta la chiave pubblica, bisogna aggiungerla all'elenco delle chiavi autorizzate:

```
# Se non esiste, creo la cartella .ssh con i giusti permessi
mkdir .ssh
chmod 700 .ssh

# Aggiungo alla coda del file .ssh/authorized_keys la chiave
cat id_rsa.pub >> .ssh/authorized_keys
```

Capitolo 12

iptables: Stateful packet filtering

IPTABLES implementa la funzionalità di stateful packet filtering a livello di kernel ed ha il controllo dei pacchetti in transito sulle interfacce di rete. I pacchetti sono sottoposti a diverse modalità di elaborazione chiamate **table**, ciascuna delle quali composta da gruppi di regole chiamate **chain**.

Netfilter Packet Traversal

<http://linux-ip.net/nf/nfs-traversal.png>
Martin A. Brown, martin@linux-ip.net

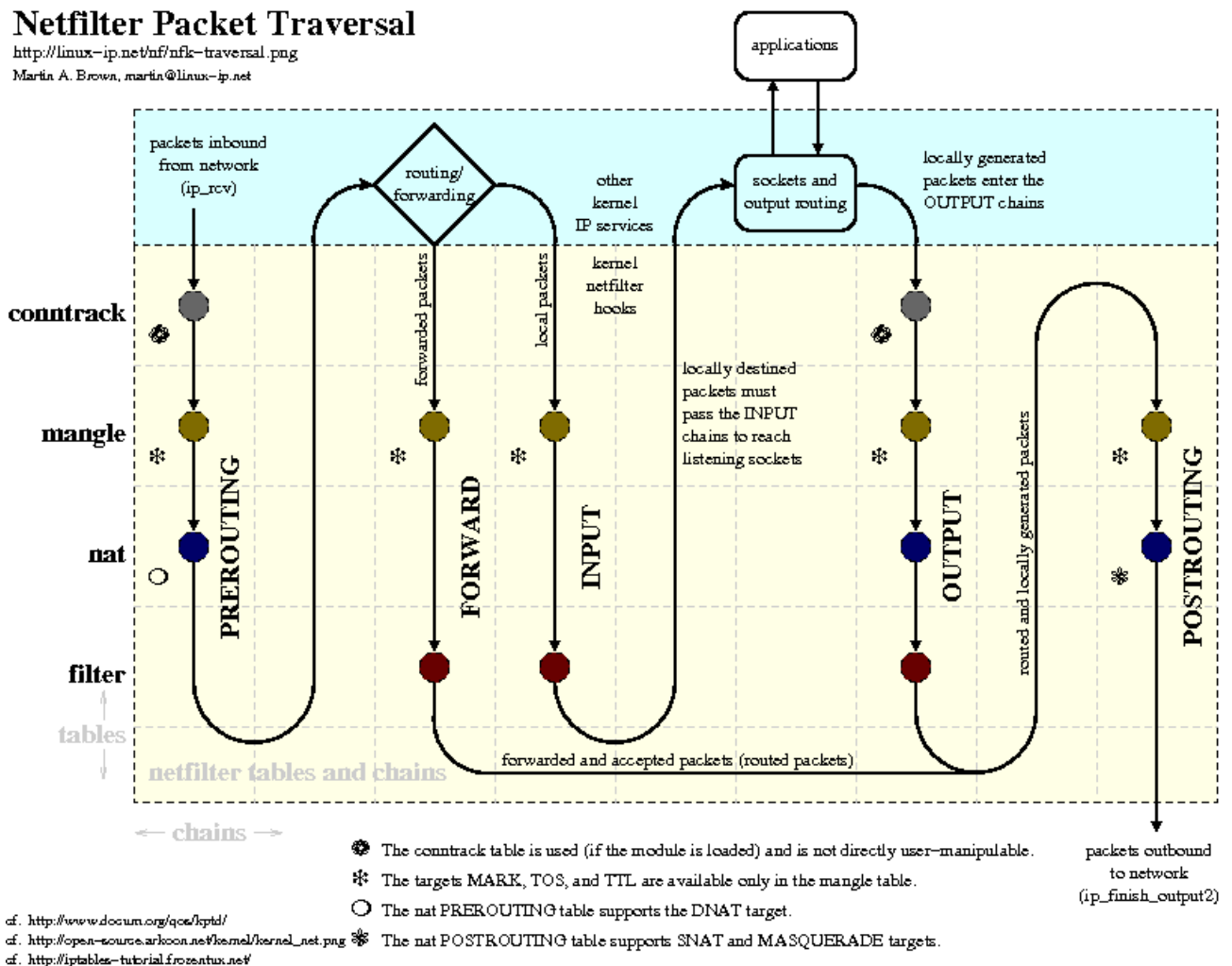


Figura 12.1: Schema architetturale delle IPTABLES

12.1 Tabella FILTER: Funzionalità di firewall

La tabella di FILTER permette di definire le regole di firewall vere e proprie. Al suo interno sono contenute tre **chain** predefinite:

- INPUT: contiene le regole di filtraggio da applicare ai pacchetti in arrivo.

- **OUTPUT:** contiene le regole di filtraggio da applicare ai pacchetti in uscita
- **FORWARD:** contiene le regole di filtraggio da applicare ai pacchetti in transito per il firewall (inoltrate da interfacce diverse)

12.1.1 Visualizzare le regole attuali [↗](#)

```
# Visualizza le configurazioni di iptables
iptables -vnxL --line-numbers
```

Ricordati

Per utilizzare il comando `iptables` bisogna avere permessi di **root**.

12.1.2 Aggiungere nuove regole [↗](#)

L'amministratore può personalizzare il comportamento del firewall aggiungendo nuove regole che, alla fine dell'elaborazione, dovranno stabilire uno dei tre possibili esiti:

- **DROP:** Scarta il pacchetto
- **REJECT:** Scarta il pacchetto ed invia un pacchetto ICMP per segnalare l'errore al mittente.
- **ACCEPT:** Accetta il pacchetto.

Una regola può essere aggiunta sia **in testa** che **in coda** alle regole già presenti nella chain:

```
# Aggiungo una regola in coda ( APPEND ) alla chain FORWARD
iptables -A FORWARD <options> -j ACCEPT

# Aggiungo una regola all'inizio della coda ( INSERT )
iptables -I FORWARD <options> -j ACCEPT
```

Si possono specificare un gran numero di opzioni per migliorare il filtraggio dei pacchetti, le più importanti sono:

```
-i eth3      # Solo pacchetti in ingresso dall'interfaccia eth3
-o eth3      # Solo pacchetti in uscita dall'interfaccia eth3
-s <ip>      # Pacchetti che provengono dall'ip specificato
-d <ip>      # Pacchetti destinati all'ip specificato
-p tcp       # Solo pacchetti TCP
# Specificando il protocollo tcp o udp, si possono selezionare le porte:
--dport <prt> # Pacchetti con porta di destinazione == <prt>
--sport <prt> # Pacchetti con porta di partenza == <prt>
# Nel caso del protocollo TCP, anche lo stato della connessione:
-m state --state NEW,ESTABLISHED
```

Ricordati

Se un pacchetto non soddisfa nessuna regola, **viene applicata la *policy* di default**.

Ricordati

Le regole `iptables` vengono resettate allo spegnimento della macchina, se si vogliono rendere persistenti è quindi importante aggiungerle al file `.bashrc`

12.1.3 Abilitare LOOPBACK e SSH [↗](#)

Alcune regole sono molto importanti e vanno sempre specificate prima di introdurre delle nuove policy di default.

```
# Accetta tutti i pacchetti dell'interfaccia di loopback
iptables -I INPUT -i lo -s 127.0.0.0/8 -j ACCEPT
iptables -I OUTPUT -o lo -d 127.0.0.0/8 -j ACCEPT

# Consenti le connessioni SSH ( in questo caso verso router )
iptables -I INPUT -i eth3 -s 192.168.56.1 -d 192.168.56.202 \
-p tcp --dport 22 -j ACCEPT
iptables -I OUTPUT -o eth3 -d 192.168.56.1 -s 192.168.56.202 \
-p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT
```

12.1.4 Eliminare una regola [↗](#)

Per eliminare una regola, si può far uso dell'opzione -D, specificando il numero di regola oppure riportando la regola per intero.

```
# Elimina la regola numero 2 della chain INPUT
iptables -D INPUT 2

# Se ad esempio aggiungiamo una regola del tipo
iptables -A INPUT -s 10.1.1.1 -j ACCEPT
# Possiamo eliminarla con
iptables -D INPUT -s 10.1.1.1 -j ACCEPT

# E' inoltre possibile eleminare tutte le regole
# di una chain ( FLUSH )
iptables -F INPUT
```

Pattern Ricorrenti

Spesso è comodo definire una funzione per aggiungere un set di regole a iptables parametrizzando il tipo di operazione. Questo permette di aggiungere e rimuovere queste regole con facilità.

```
# ARGOMENTI: $1 A oppure D per aggiungere o togliere la regola.
```

```
function gestisciRegola () {
    iptables -$1 INPUT -s 10.1.1.1 -j ACCEPT
}
```

```
# Esempio d'uso
```

```
gestisciRegola A    # Aggiungo la regola
```

```
gestisciRegola D    # Elimino la regola
```

12.1.5 Impostare le policy di default [↗](#)

Per impostare la policy di default, si può utilizzare l'opzione -P

```
# Imposto tutte le policy di default droppando i pacchetti che non
# soddisfano le regole
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

Ricordati

Quando vengono definite le policy di default a DROP è molto importante aggiungere delle regole per l'interfaccia di loopback e per la connessione ssh, altrimenti il sistema potrebbe non funzionare correttamente.

12.1.6 Loggare pacchetti con iptables [↗](#)

Definendo come policy LOG è possibile loggare i pacchetti in transito:

```
# Inserisce una regola che logga i pacchetti in transito nella chain
# specificata con prefisso e livello di log specificati
iptables -I <chain> <options> -j LOG --log-prefix=" prefisso " \
        --log-level <livello_log>
```

Info

I log hanno *facility* = kernel, questo implica che se ad esempio specifichiamo log-level=debug, i log verranno salvati con livello kernel.debug

E' possibile specificare un file di destinazione apposito, per ulteriori informazioni guardare il capitolo 7.

12.1.7 Rilevare inizio e fine di una connessione TCP [↗](#)

Un caso di notevole interesse è la rilevazione dell'inizio e la fine di una connessione tcp. Nell'esempio sottostante, vengono loggati l'evento di fine ed inizio.

```
# Logga l'inizio di ogni connessione inoltrata da 10.1.1.1 a 10.9.9.1
# con livello debug e prefisso NEWCON
iptables -I FORWARD -i eth2 -s 10.1.1.1 -d 10.9.9.1 -p tcp \
        --tcp-flags SYN SYN -j LOG --log-prefix " NEWCON " --log-level debug

# Logga la fine di ogni connessione inoltrata da 10.1.1.1 a 10.9.9.1
# con livello debug e prefisso ENDCON
iptables -I FORWARD -i eth2 -s 10.1.1.1 -d 10.9.9.1 -p tcp \
        --tcp-flags FIN FIN -j LOG --log-prefix " ENDCON " --log-level debug
```

12.2 Tabella NAT: Effettuare il NAT sui pacchetti in transito [↗](#)

La tabella NAT permette di effettuare delle modifiche ai pacchetti in modo che risultino traslate a livello di indirizzo. Lavora sui pacchetti *forwarded*, questo implica che **per utilizzare il NAT è necessario abilitare le regole di FORWARD** esposte in precedenza.

12.2.1 Destination NAT [↗](#)

Il *Destination NAT* permette di intercettare le connessioni da Client a Router, ridirigendole automaticamente al Server. In pratica Router *impersona* Server ed il Client non si accorge di nulla.

```
# Intercetta le connessioni SSH da Client a Router e le ridirige a Server
iptables -t nat -A PREROUTING -i eth2 -s 10.1.1.1 -d 10.1.1.254 \
        -p tcp --dport 22 -j DNAT --to-dest 10.9.9.1
```

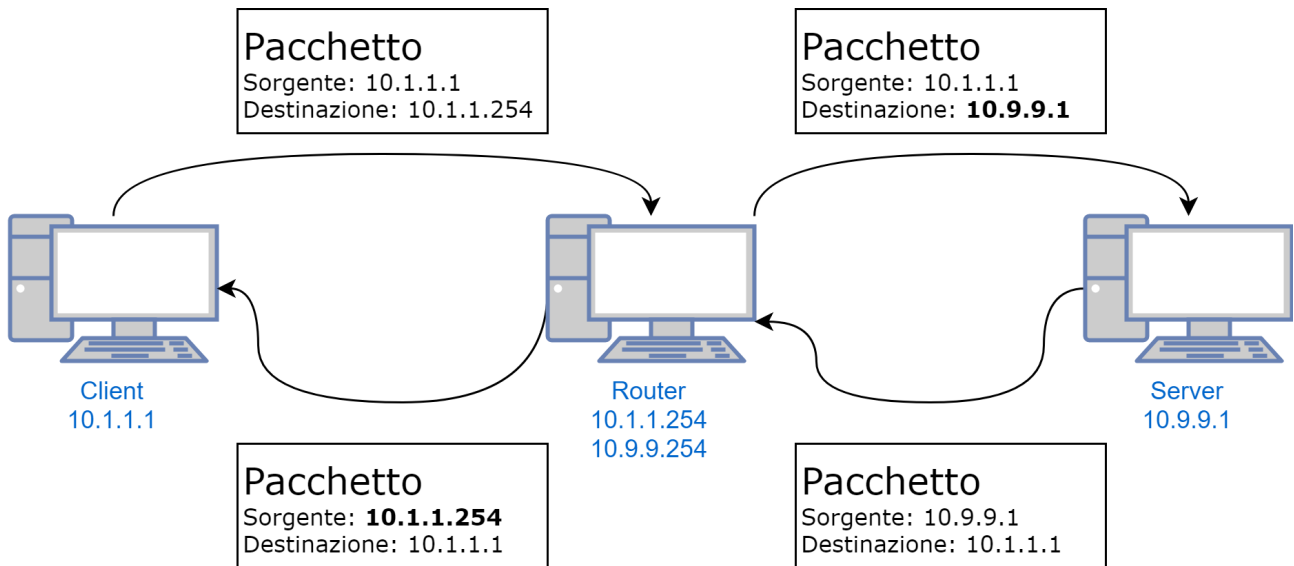



Figura 12.2: Esempio di Destination NAT. In grassetto gli indirizzi modificati.

Ricordati

Ricordati di impostare le regole di FORWARD di iptables in modo che i pacchetti possano essere inoltrati da Router, altrimenti il NAT non funzionerà. Per testare la configurazione iniziale, puoi impostare momentaneamente le policy di FORWARD come ACCEPT

```
iptables -P FORWARD ACCEPT
```

12.2.2 Source NAT [🔗](#)

Il *Source NAT* permette di mascherare la sorgente di un pacchetto, in modo che **la destinazione** non veda il mittente originale. In pratica, quanto il Client invia un pacchetto al Server, questo lo riceve come se **lo avesse mandato il Router**. Quando poi il Server invierà la risposta al Router, questo si occuperà di inoltrarla al Client.

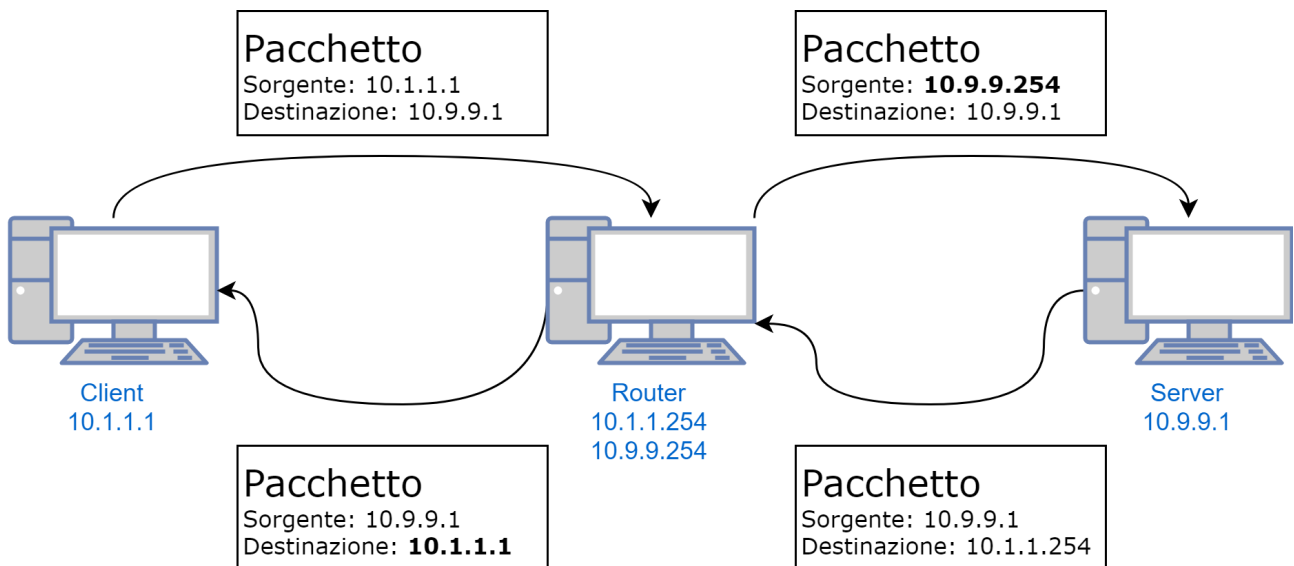


Figura 12.3: Esempio di Source NAT. In grassetto gli indirizzi modificati.

```
# Espone le connessioni dal Client al Server come se venissero
# da Router stesso
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.1 -d 10.9.9.1 \
-j SNAT --to-source 10.9.9.254
```

Ricordati

Ricordati di impostare le regole di FORWARD di iptables in modo che i pacchetti possano essere inoltrati da Router, altrimenti il NAT non funzionerà. Per testare la configurazione iniziale, puoi impostare momentaneamente le policy di FORWARD come ACCEPT

```
iptables -P FORWARD ACCEPT
```

12.3 Chain personalizzate [↗](#)

Con iptables è possibile creare delle chain personalizzate:

```
# Crea la chain PIPPO
iptables -N PIPPO

# Inserisco una regola all'interno di PIPPO che faccia direttamente il
# RETURN ( ovvero ritorni alla chain che l'ha invocata )
iptables -I PIPPO -j RETURN

# Elimino tutte le regole all'interno di PIPPO ( FLUSH )
iptables -F PIPPO

# Elimino la chain PIPPO
iptables -X PIPPO
```

12.4 Salvare e ripristinare le regole di iptables [↗](#)

Le regole di iptables vengono perse ad ogni riavvio della macchina. Per renderle persistenti è quindi necessario ricaricarle ad ogni avvio. Gli strumenti iptables-save e iptables-restore permettono di salvare e in seguito ripristinare la configurazione corrente.

```
# Salva la configurazione corrente sul file
iptables-save > output.txt

# Ripristina la configurazione salvata
iptables-restore < output.txt
```

Capitolo 13

tcpdump: Sniffing dei pacchetti

tcpdump è uno strumento che permette di intercettare e leggere i pacchetti TCP/IP in transito.

13.1 Principali opzioni [↗](#)

```
-i any          # Ascolta da tutte le interfacce di rete
-i eth0        # Ascolta sull'interfaccia eth0
-l            # IMPORTANTE Line-buffered: Stampa un pacchetto appena
              # lo riceve senza bufferizzare.
-n            # IMPORTANTE Non risolvere gli hostname, lascia numerico
-t            # Stampa il tempo in un formato human-friendly
-c [N]        # Legge solo N pacchetti e poi termina
-w output     # Scrive i pacchetti nel file PCAP di output
-r file       # Legge i pacchetti dal file PCAP
-p            # No promiscuous mode
-A            # Stampa i pacchetti in ASCII
-X            # Stampa i pacchetti in ASCII ed esadecimale
```

Ricordati

Per rilevare in real-time i pacchetti è molto importante specificare le opzioni `-l` e `-n` che permettono rispettivamente di non bufferizzare e di evitare di risolvere gli hostname.

13.2 Filtrare i pacchetti [↗](#)

tcpdump offre una grande quantità di filtri per scremare i pacchetti.

Info

Per leggere tutte le informazioni sui filtri di tcpdump, guardare la man page relativa ai `pcap-filter`

13.2.1 Filtrare i pacchetti in base alla destinazione o alla sorgente [↗](#)

```
# Leggi i pacchetti in arrivo da 10.9.9.1
tcpdump src 10.9.9.1

# Leggi i pacchetti destinati a 10.9.9.1
tcpdump dst 10.9.9.1
```

13.2.2 Filtra i pacchetti in base al network [↗](#)

```
# Leggi i pacchetti relativi al network 10.9.9.0/24
tcpdump net 10.9.9.0/24
```

13.2.3 Filtra il traffico legato ad una porta [↗](#)

```
# Leggi i pacchetti relativi alla porta 1234
tcpdump port 1234

# Leggi i pacchetti in arrivo dalla porta 1234
tcpdump src port 1234
```

13.2.4 Filtra il traffico legato ad un protocollo [↗](#)

```
# Leggi i pacchetti relativi al protocollo icmp
tcpdump icmp

# Leggi i pacchetti relativi a udp
tcpdump udp
```

13.2.5 Filtra il traffico in base alla dimensione del pacchetto [↗](#)

```
# Leggi i pacchetti con dimensione minore di 32 byte
tcpdump less 32

# Leggi i pacchetti con dimensione maggiore di 32 byte
tcpdump greater 32

# Si possono concatenare
tcpdump less 32 and greater 20
```

13.3 Combinare i filtri [↗](#)

Per combinare i filtri si possono utilizzare `and`, `or` e `not`.

```
# Traffico destinato a 192.168.0.2 e non di tipo icmp
tcpdump dst 192.168.0.2 and not icmp

# Traffico da un host ma non su una specifica porta
# Utile per filtrare connessione SSH
tcpdump src 10.9.9.1 and not dst port 22

# Con le parentesi si possono anche creare regole piu complesse
tcpdump src 10.9.9.1 and (dst port 1234 or 22)
```

13.4 Catturare i pacchetti con flag TCP particolari [↗](#)

```
# Cattura i pacchetti con dei flag tcp settati:  
tcpdump 'tcp[tcpflags] == tcp-syn'  
tcpdump 'tcp[tcpflags] == tcp-fin'
```

Capitolo 14

SNMP: Simple Network Management Protocol

Il protocollo SNMP permette di gestire in maniera semplificata diverse risorse di rete. Ogni oggetto è rappresentato da un *OID* univoco ed incluso in un database detto *MIB*.

14.1 Configurare il demone SNMPD [↗](#)

Prima di poter eseguire delle richieste è necessario impostare il demone snmp del server. In particolare:

```
# Nel file: /etc/snmp/snmp.conf
# Commentare la riga "mibs :" per avere i nomi simbolici
```

```
# Nel file: /etc/snmp/snmpd.conf
# Sostituire
agentAddress udp:127.0.0.1:161
# con
agentAddress udp:161

# Definire una vista che includa tutto il MIB
view all included .1

# Abilitare le community ad operare su quella vista
rocommunity public default -V all
rwcommunity supercom default -V all
```

Ricordati

Dopo ogni cambio di configurazione, bisogna riavviare il demone SNMP tramite il comando:

```
sudo systemctl restart snmpd
```

14.2 Visualizzare tutte le entry [↗](#)

Per visualizzare tutti gli oggetti di un server, si può utilizzare il comando `snmpwalk`:

```
# Visualizza tutti gli oggetti del server 10.9.9.1
snmpwalk -v 1 -c public 10.9.9.1 .1

# Pue' essere utile utilizzare in piping il comando less
snmpwalk -v 1 -c public 10.9.9.1 .1 | less
```

14.3 Ottenere il valore di una entry [↗](#)

```
# Visualizza il valore di una specifica entry
snmpget -v 1 -c public 192.168.56.203 'UCD-SNMP-MIB:memAvailReal.0'
```

14.4 Estendere le funzionalità di SNMP [↗](#)

È possibile estendere le funzionalità di SNMP aggiungendo delle direttive `extend` nel file di configurazione `/etc/snmp/snmpd.conf`

```
# Da aggiungere al file /etc/snmp/snmpd.conf

# Restituisce la data corrente
extend currdate /bin/date

# Restituisce il numero di processi in esecuzione sulla macchina
extend-sh sshnum ps haux | wc -l
```

Info

Utilizzando la direttiva `extend-sh` è possibile invocare direttamente dei comandi di shell.

Ricordati

Per vedere gli effetti della modifica al file di configurazione, bisogna riavviare il demone SNMP tramite:

```
sudo systemctl restart snmpd
```

Ricordati

I valori definiti tramite `extend` non sono aggiornati ad ogni richiesta, ma vengono salvati in **cache**, generalmente aggiornata ogni 5 secondi.

14.4.1 Leggere il valore di un campo personalizzato [↗](#)

Per ottenere il valore di un campo creato tramite `extend`, si può utilizzare:

```
# Legge il valore di "sshnum" definito precedentemente
snmpget -v 1 -c public <indirizzo_macchina> \
'NET-SNMP-EXTEND-MIB::nsExtendOutputFull."sshnum"'

# Per ottenere il valore puro si puo' utilizzare AWK
snmpget -v 1 -c public <indirizzo_macchina> \
'NET-SNMP-EXTEND-MIB::nsExtendOutputFull."sshnum"' | \
awk -F 'STRING: ' '{ print $2 }'
```

14.4.2 Estendere la funzionalità con un comando root [↗](#)

Per fare in modo che SNMP possa essere esteso per eseguire un comando di root sono necessari alcuni passaggi:

1. Editare il file di configurazione `/etc/snmp/snmpd.conf` per includere la nuova regola:

```
# NOTARE la presenza di /usr/bin/sudo
# Restituisce le connessioni correntemente attive
extend      activeconn /usr/bin/sudo /bin/ss -ntp
```

2. Aggiungere una regola in `/etc/sudoers` (tramite il comando `sudo visudo`) per permettere al demone `snmp` di eseguire il comando come `root`:

```
# NOTA: Per editare il file conviene usare: sudo visudo
# Permette all'utente snmp di eseguire il comando ss
# senza digitare la password
snmp      ALL=NOPASSWD:/bin/ss
```

Info

Si può testare l'esito di questa operazione impersonando l'utente `snmp` e cercando di eseguire il comando che richiede permessi di `root`. Per farlo bisogna diventare `root` (usando il comando `sudo -i`) e successivamente aprire una shell come utente `snmp`:

```
# Apre una shell come utente snmp
# NOTA: Bisogna essere root per riuscirci
su -s /bin/bash - snmp
```

14.5 Ottenere informazioni riguardo un processo

SNMP può essere configurato per monitorare un processo in esecuzione e fornire diversi parametri a riguardo. Il primo passo è modificare il file di configurazione: `/etc/snmp/snmpd.conf`

```
# Aggiungere al file: /etc/snmp/snmpd.conf
proc <nome_processo> <max_numero> <min_numero>

# Ad esempio, per monitorare il demone di log
proc rsyslogd

# Al massimo 4 istanze del processo
proc rsyslogd 4

# Almeno un istanza del processo, ma non piu' di 10
proc rsyslogd 10 1
```

Ricordati

Dopo ogni cambio di configurazione, bisogna riavviare il demone SNMP tramite il comando:

```
sudo systemctl restart snmpd
```

14.5.1 Conteggio del numero di istanze di un processo

Il numero di istanze di un processo è molto utile per determinare se un particolare processo è in esecuzione oppure no.

Il problema principale è ottenere l'id della tabella corrispondente al processo desiderato. Per leggere il conteggio sarà necessario effettuare prima una `snmpwalk` per ottenere l'id e successivamente una `snmpget` per il conteggio.


```

# Ottengo l'id del processo "rsyslogd"
ID=$(snmpwalk -v 1 -c public 10.9.9.1 "UCD-SNMP-MIB::prNames" \
  | grep rsyslogd | awk -F "prNames." '{ print $2}') \
  | awk -F " = " '{ print $1 }'

# Utilizzo l'id per ottenere il conteggio
snmpget -v 1 -c public 10.9.9.1 "UCD-SNMP-MIB::prCount.$ID" \
  | awk -F "INTEGER: " '{ print $2 }'

```

Info

Il modulo UCD-SNMP-MIB offre molte informazioni riguardo ai processi registrati, per vederle tutte si può utilizzare:

```

# Visualizza tutti i valori di UCD-SNMP-MIB
snmpwalk -v 1 -c public 10.9.9.1 .1 | grep UCD-SNMP-MIB

```

Funzioni Utili

```

# Ottiene il numero di istanze di un processo registrato tramite SNMP
# ARGOMENTI: $1 nome processo, $2 indirizzo macchina
function getProcessCount () {
  # Ottengo l'id del processo
  ID=$(snmpwalk -v 1 -c public $2 "UCD-SNMP-MIB::prNames" \
    | grep "$1" | awk -F "prNames." '{ print $2}' \
    | awk -F " = " '{ print $1 }' )

  # Utilizzo l'id per ottenere il conteggio
  NUM=$( snmpget -v 1 -c public $2 "UCD-SNMP-MIB::prCount.$ID" \
    | awk -F "INTEGER: " '{ print $2 }' )

  # Stampo il risultato
  echo $NUM
}

# Esempio d'utilizzo:
NUMERO=$( getProcessCount "rsyslogd" "10.9.9.1" )
echo $NUMERO

```

Capitolo 15

LDAP: Lightweight Directory Access Protocol

Il protocollo LDAP permette di interagire con una Directory.

15.1 Definire uno schema [↗](#)

Uno schema LDAP è una particolare definizione, scritta in un file `ldif`, salvata nella sezione dei metadati del direttorio `ldap`. Nello schema sono presenti le definizioni degli **attributi** e delle **classi** che si vogliono introdurre. Le classi possono essere di due tipi:

- **Strutturali:** Sono caratterizzate dal tag `STRUCTURAL` e sono le uniche classi che possono essere presenti direttamente nel direttorio `ldap`.
- **Ausiliarie:** Sono caratterizzate dal tag `AUXILIARY` e possono essere **aggiunte** a delle classi strutturali, per aumentarne gli attributi. **Non possono essere direttamente aggiunte al direttorio `ldap`**, ma devono sempre essere associate ad una classe strutturale.

15.1.1 Esempio con classi Strutturali [↗](#)

```
dn: cn=count,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: count
olcAttributeTypes: ( 1000.1.1.1 NAME ( 'indirizzo' )
  DESC 'indirizzo IP di un host in forma di stringa'
  EQUALITY caseExactMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
olcObjectClasses: ( 1000.2.1.1 NAME 'client'
  DESC 'un client'
  MUST indirizzo
  STRUCTURAL )
```

Info

A meno che non sia esplicitamente richiesto il contrario, è consigliato utilizzare sempre le classi strutturali per definire nuovi schemi.

15.1.2 Esempio con classi Ausiliarie [↗](#)

```
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 )
```

Ricordati

Nella definizione dello schema **non utilizzare TAB per indentare il codice ma esclusivamente degli spazi**, ldap ha dei seri problemi mentali e se non fai così inizia a lanciare errori a caso. Scusate lo sfogo, ci ho perso un ora.

Una volta definito lo schema, questo può essere aggiunto tramite il comando:

```
# Aggiungo lo schema definito nel file "filesystem.ldif"
sudo ldapadd -Y EXTERNAL -H ldapi:/// -f filesystem.ldif
```

15.1.3 Tipi di dato [↗](#)

In uno schema LDAP è possibile definire vari tipi di dato utilizzando la sintassi OID.

Tabella 15.1: Commonly Used Syntaxes

Name	OID	Description
boolean	1.3.6.1.4.1.1466.115.121.1.7	boolean value
directoryString	1.3.6.1.4.1.1466.115.121.1.15	Unicode (UTF-8) string
distinguishedName	1.3.6.1.4.1.1466.115.121.1.12	LDAP DN
integer	1.3.6.1.4.1.1466.115.121.1.27	integer
numericString	1.3.6.1.4.1.1466.115.121.1.36	numeric string
OID	1.3.6.1.4.1.1466.115.121.1.38	object identifier
octetString	1.3.6.1.4.1.1466.115.121.1.40	arbitrary octets

Tabella 15.2: Commonly Used Matching Rules

Name	Type	Description
booleanMatch	equality	boolean
caseIgnoreMatch	equality	case insensitive, space insensitive
caseIgnoreOrderingMatch	ordering	case insensitive, space insensitive
caseIgnoreSubstringsMatch	substrings	case insensitive, space insensitive
caseExactMatch	equality	case sensitive, space insensitive
caseExactOrderingMatch	ordering	case sensitive, space insensitive
caseExactSubstringsMatch	substrings	case sensitive, space insensitive
distinguishedNameMatch	equality	distinguished name
integerMatch	equality	integer
integerOrderingMatch	ordering	integer
numericStringMatch	equality	numerical
numericStringOrderingMatch	ordering	numerical
numericStringSubstringsMatch	substrings	numerical
octetStringMatch	equality	octet string
octetStringOrderingStringMatch	ordering	octet string
octetStringSubstringsStringMatch	ordering	octet string
objectIdentifierMatch	equality	object identifier

15.2 Eliminare uno schema [↗](#)

In caso sia necessario modificare uno schema definito precedentemente, il modo più semplice per farlo è quello di eliminarlo e successivamente ridefinirlo.

Ogni schema non è altro che un file `ldif` all'interno della cartella `/etc/ldap/slapd.d/cn=config/cn=schema/`. Nel caso dell'esempio `filesystem` trattato in precedenza, per eliminarlo bisognerà procedere in questo modo:

```
# Stoppo il demone slapd
sudo systemctl stop slapd

# A questo punto navigo nella directory degli schemi ( come root )
cd "/etc/ldap/slapd.d/cn=config/cn=schema/"

# Dovrebbe essere presente un file: cn={n}filesystem.ldif
# con n variabile in base allo schema
# Elimino il file
rm "cn={4}filesystem.ldif"

# Riavvio il demone slapd
sudo systemctl start slapd
```

A questo punto è possibile riaggiungere lo schema modificato in maniera analoga alla sezione 15.1.

15.3 Interrogare la directory [↗](#)

Per interrogare la directory è possibile utilizzare il comando `ldapsearch` con la seguente sintassi:

```
ldapsearch -x -b "dc=labammsis" [ -s base | one | sub ] [ filtri ]
```

```
# Visualizza l'entry e tutti i figli a partire da "dc=labammsis"
ldapsearch -x -b "dc=labammsis"

# Visualizza solo i figli di primo livello di "fn=home,dc=labammsis"
ldapsearch -x -b "fn=home,dc=labammsis" -s one

# Visualizza solo l'entry "fn=home,dc=labammsis"
ldapsearch -x -b "fn=home,dc=labammsis" -s base
```

Funzioni Utili

```
# Legge uno specifico attributo di una entry ldap
# ARGOMENTI: $1 Distinguished Name, $2 Nome attributo
# RETURN: 0 se l'attributo e' stato trovato, 1 altrimenti
function readldapattr () {
    RES=$(ldapsearch -x -b "$1" -s base | egrep "^$2" \
| awk -F ":" '{ print $2 }')
    if test -z "$RES" ; then
        return 1
    else
        echo "$RES"
        return 0
    fi
}

# Esempio d'uso:
ATTRVAL=$( readldapattr "ind=10.9.9.1,ind=10.1.1.1,dc=labammsis" "cnt" )
echo "RETURN VALUE: $?"
echo "ATTR VALUE: $ATTRVAL"
```

15.4 Aggiungere una entry [↗](#)

Per aggiungere una entry alla directory, si utilizza il comando `ldapadd`. Il primo passo è definire un file `ldif`, contenente gli attributi della entry. In seguito vengono mostrati due casi di aggiunta, a seconda che la classe sia *strutturale* o *ausiliaria*.

15.4.1 Aggiungere entry con classe Strutturale [↗](#)

Seguendo l'esempio del count sopra riportato, potrebbe essere:

```
# Definizione di una entry per lo schema count
dn: indirizzo=123.123.123.123,dc=labammsis
objectClass: client
indirizzo: 123.123.123.123
```

15.4.2 Aggiungere entry con classe Ausiliaria [↗](#)

Seguendo l'esempio del filesystem sopra riportato, potrebbe essere:

```
# Definizione di una entry per lo schema filesystem
dn: fn=ciao,dc=labammsis
objectClass: file
objectClass: organization
o: Laboratorio
fn: ciao
fs: 1234
```

Info

Dato che `file` è una classe **ausiliaria**, è necessario aggiungere alla entry una classe strutturale per supportarla, in questo caso `organization`.

A questo punto è possibile aggiungere la entry alla directory

```
# Carica la entry dal file ldif e la inserisce nella directory
ldapadd -x -c -D "cn=admin,dc=labammsis" -w admin -f file.ldif

# E' anche possibile caricarla direttamente da standard input
./mioscript.sh | ldapadd -x -c -D "cn=admin,dc=labammsis" -w admin
```

15.5 Modificare una entry [↗](#)

Per modificare una entry si può utilizzare il comando `ldapmodify` a cui bisogna "dare in pasto" un file ldif con le differenze:

```
# File differenze.ldif che modifica l'attributo fs dell'entry
# definita in precedenza con dn: fn=ciao,dc=labammsis
dn: fn=ciao,dc=labammsis
changetype: modify
replace: fs
fs: 2345
```

A questo punto si può invocare il comando `ldapmodify` per effettuare le modifiche:

```
# Invia le modifiche
ldapmodify -x -D "cn=admin,dc=labammsis" -w admin -f differenze.ldif
```

Funzioni Utili

```
# Modifica uno specifico attributo di una entry ldap
# ARGOMENTI: $1 Distinguished Name, $2 Nome attributo, $3 Nuovo valore
# RETURN: 0 se l'attributo e' stato modificato, 1 altrimenti
function modldapattr () {
    echo -e "dn: $1\nchangetype: modify\nreplace: $2\n$2: $3" \
    | ldapmodify -x -D "cn=admin,dc=labammsis" -w admin
    return $?
}

# Esempio d'uso:
modldapattr "ind=10.9.9.1,ind=10.1.1.1,dc=labammsis" "cnt" "2"
echo "RETURN VALUE: $?"
```

15.6 Eliminare una entry [↗](#)

Per eliminare una entry si può utilizzare il comando `ldapdelete`

```
# Elimina la entry con dn: fn=ciao,dc=labammsis
ldapdelete -x -D "cn=admin,dc=labammsis" -w admin "fn=ciao,dc=labammsis"
```

Ricordati

Per eliminare una entry, questa deve essere una *leaf*, ovvero non avere entry figlie. Per ovviare al problema, si può utilizzare il comando `ldapdelete` con opzione `-r` che permette di **eliminare ricorsivamente anche i figli**.

Anchors	
^	Start of line +
\A	Start of string +
\$	End of line +
\Z	End of string +
\b	Word boundary +
\B	Not word boundary +
\<	Start of word
\>	End of word

Character Classes	
\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\xhh	Hexadecimal character hh
\Oxxx	Octal character xxx

POSIX Character Classes	
[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions	
?=	Lookahead assertion +
?!	Negative lookahead +
?<=	Lookbehind assertion +
?!= or ?<!	Negative lookbehind +
?>	Once-only Subexpression
?()	Condition [if then]
?()	Condition [if then else]
?#	Comment

Note Items marked + should work in most regular expression implementations.

Sample Patterns		
([A-Za-z0-9-]+)	Letters, numbers and hyphens	
(\d{1,2}\V\d{1,2}\V\d{4})	Date (e.g. 21/3/2006)	
([\^s]+(?:\.(jpg gif png))\.\2)	jpg, gif or png image	
(^[1-9]{1}\$ ^[1-4]{1}[0-9]{1}\$ ^50\$)	Any number from 1 to 50 inclusive	
(#?([A-Fa-f0-9]){3}([A-Fa-f0-9]){3})?)	Valid hexadecimal colour code	
((?=[*\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).	
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Email addresses	
(\<(/?[^\>]+)\>)	HTML Tags	

Note These patterns are intended for reference purposes and have not been extensively tested. Please use with caution and test thoroughly before use.

Quantifiers	
*	0 or more +
*?	0 or more, ungreedy +
+	1 or more +
+?	1 or more, ungreedy +
?	0 or 1 +
??	0 or 1, ungreedy +
{3}	Exactly 3 +
{3,}	3 or more +
{3,5}	3, 4 or 5 +
{3,5}?	3, 4 or 5, ungreedy +

Special Characters	
\	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

String Replacement (Backreferences)	
\$n	nth non-passive group
\$2	"xyz" in /^(abc(xyz))\$/
\$1	"xyz" in /^(?:abc)(xyz)\$/
\$`	Before matched string
\$'	After matched string
\$\$	Last matched string
\$\$	Entire matched string
\$_	Entire input string
\$\$	Literal "\$"

Ranges	
.	Any character except new line (\n) +
(a b)	a or b +
(...)	Group +
(?:...)	Passive Group +
[abc]	Range (a or b or c) +
[^abc]	Not a or b or c +
[a-q]	Letter between a and q +
[A-Q]	Upper case letter + between A and Q +
[0-7]	Digit between 0 and 7 +
\n	nth group/subpattern +

Note Ranges are inclusive.

Pattern Modifiers	
g	Global match
i	Case-insensitive
m	Multiple lines
s	Treat string as single line
x	Allow comments and white space in pattern
e	Evaluate replacement
U	Ungreedy pattern

Metacharacters (must be escaped)		
^	[.
\$	{	*
(\	+
)		?
<	>	