



## Zab's Linux Manual



based on **Red Hat Linux 7.0**  
by [Zabberoni Simone](#)

Tutte queste pagine sono la trascrizione dei miei appunti sull'utilizzo e la configurazione di Linux. Sono tratti da svariati documenti trovati nella grande rete, dalle man pages, dai vari howto, dal Tomware Penguin e dal Linux Documentation Project. Grazie a tutti! ;-)

### Sections:

[Comandi di "sopravvivenza" nella shell](#), comandi di base per interagire con il sistema  
[Shell e I/O Redirection](#), interpreti dei comandi, stream piping e redirection  
[Shellscript](#), creazione di script della shell e automazione di procedure  
[Utenti e password](#), creazione, gestione e policy di sistema  
[Introduzione ai filesystems](#), creazione e controllo, gerarchia standard, swap filesystems  
[Struttura di un filesystems](#), inodes, superblock, blocchi di memorizzazione, hard e soft link  
[Ricerca e identificazione dei files](#), tools e integrazione  
[Permissions](#), permessi di lettura/scrittura/esecuzione dei file, concetti di owner e group  
[Startup e shutdown del sistema](#), procedura di init, servizi e stop del sistema  
[Networking](#), utilizzo di linux in rete  
[Kernel Compiling](#), guida alla ricompilazione del kernel  
[X window](#), usare una Graphical User Interface sotto linux [NON DISPONIBILE]  
[Linux Tool Guide](#), guide di utilizzo per i vari tool standard di linux [NON DISPONIBILE]  
[Links](#) a vari siti su Linux e sicurezza

## BASIC SHELL INTERACTION

### Interazione con files e directory

<b>ls</b>	[ <i>directory</i> ]	mostra la lista dei file e delle subdirectory della directory corrente o di quella specificata. In genere viene definito l'alias <code>ll = ls -l --color</code> che stampa a colori files, directory e relativi permessi L'opzione <code>-a</code> mostra anche i files nascosti (il cui nome inizia per ".")
<b>pwd</b>		stampa la directory corrente
<b>mkdir</b>	<i>directory</i>	crea la directory specificata
<b>rmdir</b>	<i>directory</i>	elimina la directory specificata a patto che sia vuota
<b>rm</b>	<i>directory/files</i>	elimina files e directory. In genere per gli utenti è impostata l'opzione <code>-i</code> che chiede conferma per ogni file. L'opzione <code>-rf</code> rimuove ricorsivamente tutta la struttura di directory specificata (es.: <code>rm /home/user/old</code> elimina la directory <code>old</code> e tutti i files e le subdirectory contenute). Tale opzione è molto potente e se la si usa da root è necessaria la massima attenzione! Linux presume che root sappia ciò che fa quindi permette anche <code>rm / -rf</code> senza considerare il fatto che il sistema potrebbe risentirne! ;-)
<b>file</b>	[ <i>file</i> ]	mostra il tipo del file specificato o dei files della directory corrente (ASCII, binary, link ecc..)
<b>cat</b>	[ <i>files</i> ]	stampa i files specificati. Si può ridirezionare l'output per concatenare i files specificati con la seguente sintassi: <code>cat file1 file2 &gt; bigfile</code>
<b>more</b>		visualizza una schermata alla volta l'output di altri comandi (es.: <code>cat file   more</code> oppure <code>ls   less</code> )
<b>cp</b>	<i>source dest</i>	copia i files e directory sorgente nella destinazione
<b>mv</b>	<i>source dest</i>	muove i files e directory sorgente nella destinazione. Viene usato anche per rinominare (es.: <code>mv manuale.txt tutorial.txt</code> rinomina in <code>tutorial.txt</code> il file che prima era <code>manuale.txt</code> )

## System information and processes

<b>uname</b>		mostra informazioni sul sistema come architettura del processore, nome e release del sistema operativo, versione del kernel ecc. L'opzione <b>-a</b> le mostra tutte insieme
<b>runlevel</b>		stampa il runlevel del sistema (vedi system startup)
<b>who</b>		mostra gli utenti correntemente loggati nel sistema. L'opzione <b>-TH</b> stampa anche lo stato del flag <i>mesg</i> (vedi sotto)
<b>finger</b>	<i>username</i>	mostra le informazioni personali dell'utente specificato come data e ora dell'ultimo login, shell utilizzata, nome ecc...
<b>last</b>	<i>[-n lines]</i>	stampa la lista degli ultimi accessi al sistema
<b>man</b>	<i>argument</i>	stampa le informazioni relative all'argomento passato. Sono descritti le modalità di utilizzo di quasi tutti i comandi e dei files di configurazione. E' fondamentale anche per i sysadmin!
<b>strace</b>		traccia le chiamate di sistema effettuate da un programma
<b>ltrace</b>		traccia le chiamate alle librerie effettuate da un programma
<b>ps</b>		mostra i processi attivi e i relativi pid. Per pid si intende process id, ovvero un numero univoco nel sistema assegnato al processo per renderlo individuabile (vedi sotto "kill"). Una forma utile di sintassi è <b>ps -ef   grep nome_processo</b> che in genere è sufficiente, comunque <b>ps</b> ha molte opzioni ed è quindi utile controllare la man page di volta in volta in base alle necessità
<b>kill</b>	<i>[-num] pid</i>	Invia un segnale ad un dato processo indicato dal pid (vedi sopra "ps"). In genere viene utilizzato per "uccidere" (da cui kill ;-)) i processi nel caso siano bloccati ( <b>kill -9 pid</b> funziona sempre!) oppure per riavviarli. Per esempio, se cambiamo configurazione di rete è più comodo riavviare solo inetd invece che tutta la macchina quindi, una volta trovato il pid di inetd, possiamo inviare <b>kill -SIGHUP inetd_pid</b> che fa riavviare i servizi di rete

## User functions and accounting

Le funzioni relative all'accounting sono spiegate in breve, per maggiori informazioni vedi [Utenti e password: creazione, gestione e policy di sistema](#)

<b>mesg</b>	<i>[+/-]</i>	stampa o imposta lo stato del flag mesg. Se è + allora il terminale permetterà la ricezione dei messaggi derivati da <b>talk</b> e <b>write</b> (vedi sotto) degli utenti altrimenti potrà ricevere solo i broadcast message inviati da root (vedi <b>wall</b> )
<b>write</b>	<i>username</i>	invia un messaggio al terminale di un utente loggato a patto che egli abbia impostato il flag mesg +. Scrivere il messaggio e terminare con control-d
<b>talk</b>	<i>username</i>	apre una sessione di chat con l'utente specificato
<b>wall</b>		invia un messaggio a tutti gli utenti loggati. In genere è privilegio di root per indicare lo stato del sistema ed eventuali shutdown
<b>su</b>	<i>[username]</i>	permette di "diventare" l'utente specificato assumendone i privilegi (se si conosce la password ;-). Di solito viene usata per diventare root nel caso siano necessarie attività che un normale utente non può svolgere (es.: modificare i files di configurazione) ma può essere fatto per diventare qualsiasi utente. Può essere usato per eseguire un solo comando: <b>su root -c=command</b>
<b>logout</b>		esce dal terminale (o torna all'utenza originale dopo il comando <b>su</b> )
<b>useradd</b>	<i>username</i>	crea un nuovo utente
<b>userdel</b>		elimina un utente
<b>usermod</b>	<i>username</i>	modifica le proprietà di un utente
<b>chsh</b>	<i>[username]</i>	cambia la shell standard dell'utente corrente oppure di quello specificato
<b>passwd</b>	<i>[username]</i>	cambia la password dell'utente corrente o di quello specificato

## SHELLS e STREAMS

### Cos'è una shell?

Per dirla in parole veramente povere è un programma che prende un input, lo passa al computer per l'elaborazione e stampa l'output ed eventuali errori

In realtà una shell (detta anche interprete dei comandi: i "vecchi" del dos ricorderanno command.com!) è un programma molto complesso viste le operazioni che deve gestire:

- **gestione dell'ambiente**, ovvero inizializzazione e gestione delle variabili di ambiente. Si tratta di variabili che descrivono le caratteristiche dell'ambiente di lavoro e vengono utilizzate da vari programmi. Per esempio saranno sempre presenti le variabili \$path (contenente il percorso di ricerca dei comandi) e \$home (percorso della home directory dell'utente corrente)
- **interpretazione dei comandi**, come detto sopra attendere un input, eseguire il programma, stampare output ed errori
- **processi di background**, la loro gestione è tutta in mano alla shell
- **streaming e redirection**, ovvero la manipolazione dei flussi di i/o (vedi sotto)

Vista la loro natura di interpreti dei comandi, le shell possono essere considerati veri e propri ambienti di programmazione per la creazione di script (similmente ai cari batch del dos) veramente potenti.

Molti comandi che utilizziamo normalmente non sono programmi compilati bensì script della shell (detti anche shellscript o shellcode) ben progettati (l'init del sistema è gestito in gran parte da alcuni script!)

Vedi la sezione [shellscrip](#)t per maggiori chiarimenti

Ci sono moltissime varietà di shell, le più famose sono:

<b>Bourne shell</b>	/bin/sh	è la shell originale di UNIX, essenziale ma funzionale
<b>C shell</b>	/bin/csh	è stata costruita dall'università di Berkeley sulla bourne shell, ha una sintassi simile al linguaggio C e gestisce l'history dei comandi
<b>Korn shell</b>	/bin/ksh	shell molto simile alla C shell
<b>Bash</b>	/bin/bash	sviluppata dal progetto Gnu, è basata sulla bourne ma comprende funzionalità sia della C che della Korn shell. Bash sta per Bourne Again Shell, scelto perchè suona simile a born again shell (i programmatori a volte hanno uno strano senso dell'umorismo :-). Non è ancora conforme a Posix ma è presente in tutti i nuovi sistemi linux

### Streams, piping e redirection

Questi concetti sono la carta vincente delle shell sopra descritte poichè permettono di combinare più comandi in modi impensabili per altri linguaggi.

Linux utilizza tre flussi (**stream**) standard per i dati: standard **input**, standard **output** e standard **error**

Ogni programma riceve (sotto il controllo della shell) i dati sul proprio standard input, li elaborano e inviano l'output e l'error sui propri canali standard.

N.B.: sia lo standard output che lo standard error vengono stampati sullo schermo ma sono ben distinti!!

La **redirection** ci permette di prendere uno stream e inviarlo dove desideriamo. Per esempio:

- Se voglio avere su un file il contenuto di una directory basta digitare `ls -l > listfile` per fare in modo che ciò che normalmente viene stampato a schermo dal comando `ls` venga invece scritto su un file
- Se voglio che un programma non stampi messaggi di errore posso inviare lo standard error al "bidone" di sistema con il comando `cmd 2> /dev/null`

Il piping ci permette di collegare lo standard output di un programma allo standard di input di un altro (mediante il simbolo "|" detto pipe) rendendo estremamente flessibile la shell:

- ho un file di testo e voglio vedere tutte le righe che contengono una certa parola. Invece di utilizzare un editor posso scrivere `cat file | grep parola` che invia il contenuto del file non sullo schermo bensì al comando `grep` che cerca riga per riga le istanze della parola

- ho un altro file di testo che voglio leggere ma è più lungo di una pagina. Digitando `cat file | more` visualizzo il file una pagina alla volta

Logicamente si possono usare entrambe le tecniche insieme! Per esempio `ls | sort > file` invia ad un file l'elenco ordinato del contenuto di una directory.

Questi logicamente sono solo esempi banali, con l'esperienza (e imparando anche a scrivere shellscrip) verrete a comporre dei comandi veramente lunghi e articolati! Molti amministratori preferiscono ancora la console alla gui e un motivo ci sarà!

## **SHELLSCRIPT**

Come già detto, una shell può essere utilizzata anche come ambiente di programmazione.

Uno shellscrip può essere considerato una versione molto più potente e flessibile di un batch di msdos in quanto permette l'utilizzo dei costrutti di iterazione e dei controlli come un vero e proprio linguaggio di programmazione. Inoltre si possono utilizzare le caratteristiche più importanti della shell come la ridirezione e il piping unitamente ad altri comandi nati appositamente per la gestione di flussi di dati. Tra i più importanti ci sono **sed** e **awk** per la gestione di flussi di testo, **grep** per la ricerca di una parola in un testo e **wc** per il conteggio di parole, righe e caratteri di un testo. Per creare script della shell è sufficiente qualsiasi editor (io personalmente uso **vi** che colora la sintassi e controlla le parentesi ma vanno bene anche pico, joe, emacs, gnotepad+ ecc...)

### **Hello world Script**

Il tradizionale programma che scrive a video "Hello world" in formato shellscrip:

```
#!/bin/bash
echo Hello world
```

La prima riga di questo script serve a definire il percorso e il nome del file che deve essere utilizzato per l'esecuzione dell script e viene chiamata notazione Shebang (similmente al perl in cui la prima riga di solito è `#!/usr/bin/perl`).

### **Variabili**

Diversamente dai linguaggi classici come il c (e il caro pascal che tutti abbiamo studiato a scuola) nei programmi della shell non è necessario dichiarare né le variabili né i tipi stessi delle variabili.

- Una variabile di uno shellscrip può contenere un numero, un carattere o una stringa
- Assegnare un valore ad una variabile crea la variabile stessa, non è necessario dichiararla prima

Esempio:

```
#!/bin/bash
STR="Hello world"
echo $STR
```

In tal modo viene creata una variabile di nome STR a cui viene assegnato il valore "Hello world". Il \$ viene utilizzato in fase di stampa per accedere al valore della variabile stessa

All'interno delle funzioni è comunque possibile dichiarare delle variabili locali (che possono avere lo stesso nome di qualche variabile globale: in tal modo bisognerà far attenzione al problema della visibilità) con la keyword "local":

```
local VAR=valore
```

### **Catturare l'output di un comando**

E' possibile catturare l'output di un comando per assegnarlo ad una variabile. Se per esempio volessimo inserire in una variabile la lista delle voci di una directory basterebbe la linea:

```
LIST=$(ls)
```

**Attenzione:** Nel caso il \$ venisse omissso il contenuto della variabile list non sarebbe l'output del comando ls bensì la stringa "ls"

### **If .. then .. else**

La struttura del condizionale è semplice:

```
if condizione then operazione1 else operazione2
```

Ovvero: se la condizione è vera esegui il blocco di operazioni 1 altrimenti esegui il blocco di operazioni 2

La sintassi è abbastanza semplice, bisogna ricordarsi di lasciare uno spazio tra le parentesi quadre e la condizione:

```
#!/bin/bash
...
if [ "$var1" = "$var2" ]; then
  echo Condizione vera!
else
  echo Condizione falsa!
fi
```

I blocchi di operazioni possono anche contenere operazioni multiriga, altri condizionali ecc... ma bisogna ricordarsi che troppe operazioni nidificate rendono più complesso il debugging nel caso di errori di sintassi (ricordatevi sempre di mettere "fi" alla fine di un condizionale!!)

### **For**

Il ciclo for nel linguaggio shellscrip è diverso dagli altri linguaggi in quanto permette di eseguire un iterazione in base alle parole contenute in una stringa:

```
#!/bin/bash
for i in $(ls); do
  echo $i
done
```

### **While**

Il ciclo while controlla una condizione ed esegue il blocco istruzioni bloccandosi quando tale condizione è falsa (oppure viene incontrata un'istruzione di break):

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]; do
  echo N= $COUNTER
  let COUNTER=COUNTER+1
done
```

### **Until**

Il ciclo until ripete un'iterazione bloccandosi quando la condizione risulta vera (oppure viene incontrata un'istruzione di break):

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ]; do
  echo N= $COUNTER
  let COUNTER-=1
done
```

## Funzioni

In praticamente tutti i linguaggi è possibile creare funzioni per raggruppare e riutilizzare il codice (e rendere anche più leggibile il programma stesso!). Per dichiarare una funzione si usa la keyword `function` seguita dal nome della funzione e dal codice della funzione stessa. Dopodichè la funzione viene richiamata solo scrivendone il nome, come qualsiasi altro comando

```
#!/bin/bash
function myfunc {codice funzione}
```

*codice programma*

## Parametri

Nel caso si vogliano usare i parametri bisogna sapere che il nome del programma è contenuto nella variabile `$0` e i vari parametri sono contenuti in `$1`, `$2` e così via. Nella variabile  `$#`  è contenuto il numero dei parametri passati mentre la variabile  `$@`  è un array contenente tutti i parametri.

## Valori di ritorno

Nella bash il valore di ritorno di un programma viene memorizzato nella variabile  `$?` . E' quindi possibile lanciare un programma senza visualizzarne l'output (**comando `&> /dev/null`**) e utilizzarne il valore di ritorno.

## Read

Il comando `read` è usato semplicemente per richiedere all'utente un certo input. E' molto importante che tale input sia controllato in quanto un programma sicuro non si aspetta mai che l'input sia corretto! Se un programma che fa alcune semplici divisioni non controlla bene l'input è può essere possibile far leggere uno 0 come divisore e il programma va in crash (questo è un esempio banale ma moltissimi exploit si basano sul passare parametri che mandino in crash un programma)

```
#!/bin/bash
read NOME
echo "Ciao, $NOME"
```

## Select

La keyword `select` è molto utile per la creazione dei menu utente in quanto da una stringa data estrae le possibili scelte e attende un input che verrà poi controllato con una catena di `if ... then`.

```
#!/bin/bash
OPTIONS="Run Quit"
select opt in $OPTIONS; do
if [ "$opt" = "Quit" ]; then
    echo Exiting...
    exit
elif [ "$opt" = "Run" ]; then
    echo "Running..."
    exit
else
    clear
    echo Bad Option!
fi
done
```

## Espressioni aritmetiche

Per far valutare alla shell un'espressione aritmetica bisogna racchiudere l'espressione fra parentesi quadre e utilizzare il \$ per accedere al valore altrimenti verrebbe trattata come una stringa. Ovvero:

```
echo 1+1      stampa la stringa "1+1"  
echo ${1+1}  stampa 2.
```

E' da notare che la bash "risponde" mediante numeri interi quindi echo \${3/4} stampa 0!! E' necessario usare il comando bc per gestire numeri non interi:

```
echo 3/4 | bc -l stampa 0.75
```

Per i formati corretti di bc è opportuno controllarne la man page

### **Operatori aritmetici e di confronto**

Gli operatori aritmetici sono i "soliti": + per la somma, - per la sottrazione, \* per la moltiplicazione e / per la divisione

Sono invece diversi gli operatori di confronto:

```
-lt <  Minore           if [ $V1 -lt $V2 ]; then  
-gt >  Maggiore        if [ $V1 -gt $V2 ]; then  
-le <= Minore o uguale if [ $V1 -le $V2 ]; then  
-ge >= Maggiore o uguale if [ $V1 -ge $V2 ]; then  
-eq =   Uguale         if [ $V1 -eq $V2 ]; then  
-ne !=  Non uguale     if [ $V1 -ne $V2 ]; then
```

### **Operatori di confronto su stringhe**

```
= Stringhe uguali      if [ "$S1"="$S2" ]; then  
!= Stringhe diverse   if [ "$S1"!="$S2" ]; then  
-n La stringa non è vuota if [ -n $S1 ]; then  
-z La stringa è vuota   if [ -z $S1 ]; then
```

E' opportuno utilizzare le stringhe racchiuse fra doppi apici per evitare messaggi di parse error nel caso una stringa sia vuota

## **ACCOUNTING**

Linux è un sistema operativo multitasking e multiutente ed è quindi previsto che più utenti con diversi privilegi (e motivi di lavoro!) possano accedere ed utilizzare il sistema in contemporanea.

Un utente è definito da un nome utente (username) che è pubblico e da una password che è bene sia tenuta segreta a tutti gli altri! La password serve al sistema per verificare che "tu sei tu" ovvero che realmente sei l'utente che hai specificato. Avere la password di un utente vuol dire poter accedere al sistema a suo nome, leggere/modificare/cancellare i suoi dati e agire al posto suo: non molti utenti desiderano questo!

Nella creazione di un sistema linux bisogna tener conto del fatto che più utenti con vari livelli di conoscenza e competenze dovranno accedere, quindi è necessario pianificare una logica di fondo nella creazione di utenti e password definita **policy** di sistema.

### **Username**

Risulta comodo usare nomi utente tutti in minuscolo per evitare errori di battitura o dimenticanze!

Un nome utente è **univoco** nel sistema.

Pensare quindi di assegnare il nome utente in base al nome di battesimo dell'utente non è una grande idea vista la quantità di "nomi comuni ricorrenti". Se viene creato un utente con lo stesso username di un altro

non ci saranno gravi problemi a parte il fatto che solo l'originale funzionerà (i nuovi utenti vengono accodati al file /etc/passwd che in fase di login viene consultato dall'inizio).

Lo stesso vale per l'assegnazione del nome utente in base al cognome dell'utente: molte persone hanno lo stesso cognome (Rossi x esempio!)

Un'idea migliore può essere questa: il primo carattere del nome seguito dal cognome forma il nome utente. Si avrebbero le corrispondenze Mario Rossi = **mrossi**, Carlo Bianchi = **cbianchi**  
Bisogna comunque considerare i casi particolari:

- Mario Rossi e Marco Rossi: che fare? per evitare astrusi meccanismi la cosa più facile è di assegnare lo stesso username seguito da 2. Ovvero se Mario Rossi = **mrossi** allora Marco Rossi = **mrossi2** e così via per eventuali Marcella Rossi ecc...
- Nomi lunghi: e se, citando Fantozzi, dovessimo creare l'username della Contessa Mazzoni Serbellanti Viendalmare? Solo per fare un login ci vorrebbe una vita! Troncare l'username dopo 8 o 10 caratteri (N.B: o 8 o 10! la policy va applicata a tutti gli utenti per tutta la "vita" del sistema) è l'ideale, la maggior parte delle persone rientra nei 10 caratteri.
- Spazi nel cognome: come creare l'username di Mario De Sanctis? Si può risolvere in due modi: si trasforma lo spazio in "\_" e quindi **mde\_sancti** (ricordiamoci di troncarsi nomi lunghi! In tal caso a 10 caratteri) oppure non si considera lo spazio e quindi **mdesanctis** (in tal caso rientra nei 10 caratteri). Personalmente considero più "elegante" e user friendly la seconda soluzione

### **Password**

Dicevamo che dalle password dipende la sicurezza del sistema.

E che ci sono diverse tipologie di utenti con determinate conoscenze.

La password sicura, secondo lo standard Unix, è composta da:

- caratteri maiuscoli e minuscoli
  - almeno un numero
  - almeno un segno di punteggiatura
  - è lunga almeno sei caratteri
- Quindi una password sicura potrebbe essere **cF12\_0**. Le password così create, a meno che non vengano scritte da qualche parte o "spiate" durante la battitura (e non è facile!) sono praticamente impossibili da scoprire con la forza bruta.

In genere molti utenti scrivono la password su foglietti vicino al pc oppure sul monitor o sotto al telefono ecc... anche in caso di password relativamente semplici: se a tutti gli utenti venisse imposta una password simile a **cF12\_0** tutti la terrebbero scritta da qualche parte perchè impossibile da ricordare e molti si lamenterebbero.

Che fare in questi casi? Per prima cosa distinguere gli utenti di sistema (root & company) dagli utenti normali. I primi DEVONO utilizzare password sicure in quanto hanno privilegi mediante i quali un malintenzionato potrebbe danneggiare il sistema.

Per quanto riguarda gli utenti normali è necessario trovare il punto di incontro tra user friendliness e sicurezza. Una password di 6 caratteri o totalmente numerica o totalmente alfabetica (con caratteri tutti minuscoli) non è il massimo della sicurezza ma non dovrebbe essere "malvoluta" degli utenti. Riuscire ad imporre una password alfanumerica con tutti i caratteri minuscoli è una buona cosa per la sicurezza e non dovrebbe causare troppe lamentele.

Eliminare la piaga delle "password sul monitor" e simili è impossibile, bisogna fare in modo che gli utenti normali possano scrivere e leggere in zone altamente limitate (se possibile solo nelle relative home directory) in modo che un qualche malintenzionato non possa fare troppi danni.

Nel sistema la password non viene salvata da nessuna parte (a meno che non sia l'utente a farlo). In /etc/passwd (o in /etc/shadow, dipende da come è configurato il sistema) **NON** è salvata la copia cifrata della password bensì è salvata la chiave di criptazione! Inoltre i primi due caratteri sono una radice di criptazione: ce ne sono 2500 ed è quindi improbabile (impossibile dire!) che due utenti con la stessa password abbiano la stessa chiave di criptazione

## Root & company

Per root e tutti coloro che abbiano privilegi di amministrazione di sistema è **OBBLIGATORIO** impostare una password sicura (Se possibile di 8 caratteri!).

L'accesso di tali utenti al sistema deve sempre essere attraverso una comunicazione cifrata (ssh e simili) in quanto telnet e ftp inviano la password in plain text: in tal modo un qualsiasi furbone armato di tcpdump o sniffit (vedi Linux Tool Guide) potrebbe ottenere accesso totale al sistema.

Sarebbe inoltre opportuno cambiare periodicamente le password dei superusers: un volta ogni mese o due è più che sufficiente.

In linea di principio generale: l'accesso con privilegi di superuser devono essere effettuati solo in caso di effettivo bisogno, la possibilità di fare danni è alta e basta (nel peggiore dei casi! ;-)) un comando sbagliato per eliminare/modificare determinati files o impostazioni.

## Files

In un sistema normale è probabile che le password e le informazioni si trovino in `/etc/passwd` mentre nei sistemi sicuri viene utilizzato `/etc/shadow`

Il file `/etc/passwd` è composta da una riga per ogni utente suddivisa da ":" in sette campi nel formato:  
**username:pwd:uid:gid:name:home:command**

Una riga tipica potrebbe essere:

```
zabbo:0/fdGTk:100:102:Zabberoni Simone:/home/zabbo:/bin/bash
```

- Username: che altro dire :)
- Pwd: chiave di criptazione della password (vedi sopra). Si modifica con **passwd**
- Uid: User Id, è un numero intero che identifica univocamente l'utente all'interno del sistema
- Gid: Group Id, è un numero intero che identifica il gruppo di appartenenza dell'utente
- Name: nome reale dell'utente. Si modifica con **chfn** ma tale comando non è sempre presente
- Home: directory home dell'utente in cui solo lui ha accesso con tutti i diritti
- Command: di solito è una shell dei comandi ma può essere anche un altro comando (es.: **/bin/who** oppure **/bin/exit** se si vuole impedire l'accesso di un utente). Si cambia con **chsh** Logicamente un utente può usare i comandi sopra menzionati (chfn, chsh, passwd) solo relativamente alla propria utenza mentre root può modificare qualsiasi utenza.

Il file `/etc/passwd` è accessibile in lettura da tutti gli utenti e in scrittura solo da root (e dai programmi come **passwd**, **chsh**)

Il file `/etc/shadow` ha circa la stessa struttura ma contiene più informazioni (es.: scadenza dell'account) e non è leggibile dagli utenti

## **INTRODUZIONE AI FILESYSTEMS**

Per filesystem si intende sia un metodo di organizzazione dei dati su un dispositivo (es.: FAT16, Ext2) che lo spazio in cui i dati risiederanno (ovvero le partizioni). Formattare un filesystem significa creare un'organizzazione per i dati che verrà poi utilizzato dal sistema operativo.

Linux può gestire un gran numero di tipi di filesystem tra cui:

<b>ext, ext2</b>	filesystems nativi di linux (Extended e second extended)
<b>vfat, msdos</b>	filesystem Microsoft per dos e Windows 9x
<b>ntfs</b>	filesystem Microsoft per Windows NT/2000 (per ora con il kernel 2.4 il supporto è stabile solo in lettura, è ancora sconsigliabile la scrittura da linux in una partizione ntfs perchè potrebbe causare dei danni)
<b>iso9660</b>	filesystem utilizzato nei CD
<b>nfs</b>	filesystem di rete (network filesystem) per la condivisione files
<b>hpfs</b>	filesystem nativo OS/2
<b>sysv</b>	filesystem di system V e Xenix

La lista dei filesystem gestibili dal sistema si trova in **/etc/filesystems**

Nel file **/etc/fstab** ci sono le definizioni dei filesystem locali per facilitarne il **mount** (vedi sotto)

Nel file **/etc/mstab** sono indicati i filesystem montati al momento

### **Mounting / Unmounting filesystems**

Per leggere i dati da un filesystem (che sia un disco o un cd) è necessario "montarlo" in una directory (detta punto di innesto o **mount point**):

```
mount /dev/hda2 /mnt/d -t vfat
```

In questo modo i dati della seconda partizione del primo disco IDE (primary master) sono accessibili dalla directory /mnt/d e il filesystem utilizzato è di tipo vfat (windows fat).

Se i dati relativi al filesystem (ovvero dispositivo, tipo di filesystem e directory di innesto/**mount point**) fossero inseriti nel file /etc/fstab potremmo semplicemente dare il comando:

```
mount /dev/hda2 oppure mount /mnt/d
```

Quando un filesystem non serve più possiamo "smontarlo" dal suo mount point:

```
umount /mnt/d oppure umount /dev/hda2
```

### **Creating filesystems**

I filesystems vengono creati ed inizializzati con il comando **mkfs** (o mke2fs per creare un filesystem ext2):

**mkfs -t type** specifica il tipo di filesystem da creare

L'opzione **-c** attiva il conteggio dei bad blocks mentre l'opzione **-l file** legge la lista dei badblocks da un file esterno creato in precedenza con badblocks

Ad esempio, per creare un floppy ext2:

```
fdformat -n /dev/fd0 formatta il floppy senza verificare i bad blocks
```

```
mkfs -t ext2 -c /dev/fd0 crea un filesystem ext2 conteggiando i bad blocks.
```

### **Filesystem information / checking**

Possiamo sapere quanto spazio libero (o occupato) ci sia nei filesystem montati attualmente con i comandi **du** e **df**

Più utili sono i comandi **fsck** e **e2fsck** che permettono di verificare l'integrità fisica e logica di un filesystem. Se non ci sono guasti hardware e se gli shutdown vengono effettuati in maniera corretta non ci saranno (quasi!) mai problemi ai filesystem.

Fate subito un backup e un check ad un filesystem che credete corrotto anche solo in parte, continuare ad usare un filesystem corrotto porta solo a perdite di dati aggiuntivi!

**N.B.:** i comandi di controllo filesystem vanno usati **ASSOLUTAMENTE** su filesystems smontati! Fare un check su un filesystem montato potrebbe causare gravi perdite di dati!

E' utile anche il comando **badblocks** che scrive una lista dei settori danneggiati di un disco. Se usato con **fsck** si può evitare che il sistema scriva sui blocchi danneggiati semplicemente ignorandoli:

```
badblocks /dev/fd0 > badlist
```

```
fsck -t ext2 -l badlist /dev/fd0
```

## Ext2 Tools

<b>tune2fs</b>	migliora i parametri del filesystem: max mount number, root reserved blocks ecc...
<b>dumpe2fs</b>	fornisce informazioni tecniche (troppo!) prese dal superblock
<b>debugfs</b>	permette di agire direttamente sulla struttura del filesystem, da usare solo se si conosce bene ext2

## Filesystem hierarchy standard

<b>/</b>	root directory (radice) del sistema, contiene tutte le altre
<b>/bin</b>	comandi di base del sistema (accessibile a tutti gli utenti)
<b>/sbin</b>	comandi di gestione del sistema (non accessibile agli utenti)
<b>/dev</b>	dispositivi (devices) del sistema. ad ogni device viene associato un file
<b>/etc</b>	files di configurazione del sistema e dei relativi softwares installati (es.: apache, samba, vnc ecc...)
<b>/home</b>	home directory degli utenti, ad ogni user è assegnata una subdirectory di (es.: /home/jmullins) da lui accessibile con tutti i permessi. Agli altri utenti può essere possibile o meno accedervi con determinati permessi (in base alla policy di sistema o a specifiche necessità)
<b>/usr</b>	contiene la maggior parte dei programmi installati sulla macchina
<b>/lib</b>	contiene le librerie condivise da più programmi
<b>/tmp</b>	usata per scrivere files temporanei. E' bene limitarne le dimensioni per evitare attacchi mirati allo riempimento del disco
<b>/proc</b>	filesystem virtuale contenente informazioni sui processi e sui vari dispositivi (es.: processore, memoria ecc...)
<b>/var</b>	dati variabili, mail, code di stampa

La directory `/dev`, come detto sopra, contiene dei files speciali che rappresentano i dispositivi del sistema. Tra questi è opportuno ricordare `/dev/null` (**command 2** `> /dev/null` non stampa eventuali messaggi di errore) e `/dev/zero` (utilizzato di solito con `dd` per creare files compatti e per "piallare" i dischi: **dd if=/dev/zero of=/dev/hda** elimina TUTTO il contenuto di hda riempiendolo di 0)  
Gli altri special devices verranno trattati di volta in volta in concomitanza di argomenti correlati.

## Swap filesystem

Ogni sistema deve usare una partizione oppure un file di swap come appoggio per la ram. In genere è preferibile usare una partizione (molte distribuzioni in fase di installazione ne creano una). Tale spazio è bene sia contiguo e "senza buchi" per cui è comodo utilizzare **dd**:

```
dd if=/dev/zero of=/swapfile count= 1024 bs= 1024
```

Crea un file di 1024 blocchi ciascuno dei quali è grande 1024 bytes.

```
mkswap /swapfile
```

Setta /swapfile come area di swap

```
swapon /swapfile
```

Comunica al sistema che /swapfile è utilizzabile per swappare

```
swapoff /swapfile
```

Comunica al sistema che /swapfile non è utilizzabile per swappare

Per visualizzare i vari swap e il relativo utilizzo si usa il comando **free**

## FILESYSTEMS STRUCTURE

In questo capitolo viene trattata la struttura di un filesystem e la sua gestione da parte del kernel. I comandi di creazione e gestione vengono trattati nel capitolo [Filesystem Intro](#)

Un filesystem costruito su un disco è costituito da 3 componenti:

- superblock**
- tabella inode**
- blocchi di memorizzazione**

Il superblock controlla il filesystem e (di solito) non è accessibile direttamente ma solo mediante determinate chiamate di sistema. Tra le cose più importanti, il superblock tiene traccia di:

- inodes liberi**
- blocchi liberi**
- read-only flag** (N.B.: se il filesystem è montato in sola lettura neanche root può scrivere!)
- modified flag** (impostato quando il filesystem cambia: il superblock deve quindi essere scritto sul disco pena la perdita delle modifiche)

Un superblock separato risiede nella memoria del kernel per ogni filesystem montato localmente e ciascun filesystem montato risiede in una tabella di dispositivi nel kernel a cui è associato un identificatore univoco

### Gli inodes

Ogni voce del filesystem viene associata ad un inode. Gli inodes gestiscono le voci del disco e contengono tutte le voci amministrative dei files. Tutti questi dati vengono utilizzati nella i-list del kernel che è l'elenco di tutti gli inode associati ai file aperti del sistema

I primi quattro elementi di un inode sono puntatori ad altri inodes:

- i primi due gestiscono una **catena hash** (residente nel kernel) per individuare gli inode
- gli altri due gestiscono una lista di **inodes liberi** per la creazione dei files

Quando una voce inode non è più necessaria viene tolta dalla lista del kernel e accodata alla lista degli inodes liberi. Quando un inode è necessario viene prelevato il primo dalla lista dei liberi

**N.B.:** ogni voce del disco è un file e ogni dispositivo **ha** un file. I dispositivi possono essere di tipo **raw** (trasferimento a blocchi di I/O: disco) oppure **speciale per caratteri** (trasferimento di singoli caratteri: terminale, tastiera).

I dispositivi raw in genere hanno due nomi di file, uno per il trasferimento sequenziale e uno per l'accesso casuale.

Ad ogni dispositivo è associato un driver e ogni inode di dispositivo ha due indici

- un indice che punta alla tabella del driver del dispositivo
- un identificatore univoco per il dispositivo specifico del driver

Seguono quindi le informazioni sul tipo di file relativo all'inode: i tipi di files più comuni sono **files standard** (insiemi di dati memorizzati su disco) e le **directory**. Una directory è un file speciale che fa riferimento ai dati e ha nomi associati agli inode: quando si accede ad un nome viene restituito un inode per accedere al file. Nell'inode è anche memorizzato il conteggio delle voci della directory (ogni voce può essere considerata come un hard link, vedi sotto). Tenendo traccia del numero di link il sistema può sapere se una richiesta di eliminazione riguarda un file oppure una directory. Quando il numero dei link scende a 0 il file viene contrassegnato per l'eliminazione

Altri tipi di file sono i *named pipe*, i *link simbolici* e i *file di dispositivo*

Sono poi memorizzati i dati relativi ai permessi del file per i quali rimando al capitolo [Permissions](#)

L'inode tiene traccia delle dimensioni dei file. Questo è comodo poichè permette di evitare l'apertura e il calcolo delle dimensioni. La dimensione è memorizzata in due modi:

- inbyte**
- inblocchi** (se i dispositivi usano blocchi di diverse dimensioni allora viene memorizzata anche la dimensione di tali blocchi.)

L' inode tiene anche traccia di tre tipi di orari relativi al file:

- ora di creazione**
- ora dell'ultimo accesso**
- ora dell'ultima modifica**

Gli ultimi dati memorizzati in un inode sono l'elenco dei blocchi dei dati (senza i quali il file non esisterebbe :) I primi blocchi sono detti **standard** e di solito sono una decina. Se non sono sufficienti vengono utilizzati i blocchi di **indirezione** che sono altri elenchi di blocchi liberi. I blocchi di indirezione possono gestire 128 o 256 voci (in relazione alle dimensioni del filesystem) e un filesystem deve supportare 3 livelli di indirezione: si può avere un file al massimo di 16gb (che non è poco!). In tal modo un inode può tenere traccia dei file di grandi dimensioni senza creare problemi di spazio per i files più piccoli. L'indirezione aumenta la flessibilità del sistema!

## Links

Linux supporta i link ai file differenziandoli in due tipi:

**hard link** un hard link è una voce di directory che punta ad un inode già esistente. Sono efficaci ma limitati allo stesso filesystem del file da "linkare", tale file deve già esistere e non è possibile eseguire il link delle directory

**soft link** un link di questo tipo (detto comunemente **symbolic link**) è un file speciale che contiene il percorso al file a cui ci si riferisce. Tale percorso può puntare a tutto: directory, filesystems remoti, altri link e file inesistenti  
In tal modo non è necessario conoscere il percorso di un dato file ma si ha lo svantaggio della lentezza in quanto è necessario aprire il file link e risolvere il percorso e il nome del file a cui si punta (e quando si devono risolvere delle catene di link si nota veramente la lentezza!)

Logicamente in un percorso è possibile combinare link hard e symbolic

## Tips

Se si dispone di due o più dischi fissi è opportuno suddividere i filesystems più utilizzati, in genere /e /usr: in tal modo l'I/O dei file è più veloce

E' opportuno assegnare delle quote di disco ai vari utenti sia per limitare lo spazio che per evitare attacchi DoS mirati al riempimento del disco

## **RICERCA e IDENTIFICAZIONE DEI FILES**

Linux mette a disposizione svariati tools di ricerca e identificazione. Di seguito vengono mostrati i più utili e soprattutto il metodo di utilizzo più efficace. E' da notare che la potenza dei tools non dipende solo dai tool stessi ma soprattutto dal modo in cui li si usa combinati (Integrazione degli strumenti). Esistono infatti comandi come xargs che a se stanti non fanno nulla, sono stati creati appositamente per manipolare l'output di altri comandi

## Utilizzo di ls

Il comando **ls** è uno dei primi comandi unix (nomi brevi e facili da ricordare, keep it easy!) ed è uno dei comandi più utili e utilizzati. Ci sono vari flag utili per la modalità di visualizzazione:

- l stampa un file per riga con i relativi permessi e altre informazioni. Di solito viene creato l'alias **ll** equivalente a `ls -l` in quanto è il flag più usato
- i stampa il numero inode dei file
- R esamina ricorsivamente le sottodirectory
- a mostra anche i files nascosti (ovvero i files che iniziano con ".")
- r ordina l'output in senso inverso
- t ordina in base all'ultima modifica dei file (il primo sarà il file modificato più di recente)
- ct ordina in base all'ultima modifica delle proprietà o dei permessi dei file
- ut ordina in base all'ultimo accesso ai file

## Utilizzo di find

Find è un comando molto potente ma utilizza molte risorse ed ha una sintassi "oscura". Permette l'utilizzo dei caratteri jolly e permette la costruzione di espressioni (a patto di entrare nella perversa logica del comando stesso ;-)

- name *filename* trova le corrispondenze in base al nome del file
- xdev la ricerca viene effettuata solo all'interno del filesystem corrente
- perm *permission* trova le corrispondenze in base ai permessi dei file. Si può usare la codifica u+r,g+w oppure la codifica ottale
- type *filetype* trova le corrispondenze in base al tipo di file (b=file speciale a blocchi, c=file speciale per caratteri, d=directory, f=file standard, p=named pipe)
- user *username* trova le corrispondenze in base al nome utente/user id specificato (il nome utente viene "tradotto" in uid consultando il file /etc/passwd)
- group *groupname* trova le corrispondenze in base al nome del gruppo/group id specificato (il nome del gruppo viene "tradotto" in gid consultando il file /etc/group)
- size *blocks* trova le corrispondenze in base alla dimensione in blocchi dei files (la dimensione dei blocchi è 512 byte)
- atime *time* trova le corrispondenze in base alla data/ora dell'ultimo accesso ai files
- ctime *time* trova le corrispondenze in base alla data/ora dell'ultima modifica delle proprietà/permissions dei files
- mtime *time* trova le corrispondenze in base alla data/ora dell'ultima modifica dei files
- exec *command* esegue una serie di comandi usando l'output di find come input (es.: `find / -name core -mtime +7 -exec rm { } \;` ; elimina i file core vecchi di una settimana )
- ok richiede conferma prima dell'esecuzione dei comandi specificati con exec

Questi flag rendono estremamente potente il comando find ma è anche possibile costruire espressioni logiche per rendere le ricerche ancora più efficaci:

**Parentesi:** usate per raggruppare le espressioni, se le espressioni all'interno sono vere allora tutta l'espressione fra parentesi è vera. Le parentesi devono essere precedute da "\" poichè sono considerate caratteri speciali dalla shell: \  
( espressioni )

**Negazione:** si può negare il valore di un'espressione con "!" (nella shell C si usa \!)

**And:** si può esplicitare l'and fra le espressioni elencate con -a, in tal modo se la prima espressione è falsa la seconda non viene controllata

**Or:** l'opzione -o fa un'operazione or tra le espressioni ovvero se la prima è vera la seconda non viene controllata

Esempi:

- Eliminare tutti i file oggetto e i core dalla directory corrente:  
`find . \(-name '*.o' -o -name core\) -exec rm { } \;`

## Utilizzo di xargs

Pur essendo molto potente il flag `-exec` del comando `find` ha un problema: crea un processo per ogni corrispondenza che trova anche se il comando da eseguire è sempre lo stesso. Quindi, oltre al "normale" utilizzo di risorse usate da `find`, si ha un ulteriore spreco.

Per risolvere questo inconveniente è stato creato il comando **xargs** il cui scopo è quello di creare elenchi di parametri da passare ad un comando una creando un solo processo. Quindi l'esempio di prima per la rimozione dei file oggetto e dei core diventa:

```
find . \( -name '*.o' -o -name core \) | xargs rm
```

creando un solo processo per il comando `rm` invece di crearne uno per ogni file oggetto o core trovato con un buon risparmio di risorse

## Utilizzo di which

Il comando **which** si applica solo agli eseguibili e ritorna con esattezza il percorso del comando specificato (es.: il comando `which bash` stamperà a video `/bin/bash`)

**Utilizzo di file** Il comando **file** riceve in input un elenco di file che prova ad identificare controllandone il tipo, la dimensione e il primo blocco di dati. Viene poi formulata un'ipotesi sul contenuto.

**N.B.:** "Ipotesi" vuol dire che in genere il comando `file` è veritiero ma **non è garantito che lo sia!**

## PERMISSIONS

Memorizzati nell'inode di ogni file ci sono i permessi di accesso che definiscono che tipo di interazione è permessa ad un utente rispetto ad un certo file

I permessi consistono in 12 bit e ognuno di essi abilita o meno un determinato permesso. I primi 3 determinano dei casi speciali di esecuzione e verranno trattati di seguito. Gli altri nove bit possono essere divisi in tre gruppi di tre bit relativamente al tipo di utenza di cui definiscono i permessi:

- owner:** permessi relativi al possessore del file ovvero colui che l'ha creato
- group:** permessi relativi al gruppo di appartenenza dell'owner
- world:** permessi relativi a tutti gli altri utenti

I tre bit che formano i permessi hanno lo stesso ordine e significato per tutti e tre i tipi di utenze:

- read:** permesso di lettura
- write:** permesso di scrittura
- execute:** permesso di esecuzione

Utilizziamo quindi il comando **ls -l** e otterremo la lista delle voci di una directory con i relativi permessi.

Una riga tipica potrebbe essere:

```
- rwx r-- --- 1 zabbo sysusers 2009241 gen 26 15:07 myfile.tar.gz
```

Il primo "trattino" indica che si tratta di un file standard (es.: se fosse stata una directory allora il primo carattere sarebbe stato "d"). Le tre terne seguenti indicano i vari permessi:

- rwX:** l'owner (ovvero l'utente "**zabbo**") ha tutti i tipi di accesso
- r--:** il gruppo di appartenenza dell'owner (ovvero "**sysusers**") ha accesso solo in lettura al file
- : tutti gli altri utenti non possono accedere in nessun modo al file

Per cambiare i permessi dei file si usa il comando **chmod** con la seguente sintassi:

**chmod permessi file**

es.: **chmod +rx myfile.tar.gz** rende leggibile ed eseguibile a tutti il file `myfile.tar.gz`

Al posto di usare questa sintassi consiglio di utilizzare i permessi in formato decimale ovvero di considerare ogni terna di bit un numero in base allo stato del permesso: permesso settato=1. Quindi con una semplice conversione si hanno le corrispondenze:

- $rwX = 111 = 7$
- $rX = 101 = 5$
- $r- = 001 = 1$

es.: **chmod 750 myfile.tar.gz** imposta i permessi  $rwX r-x ---$  al file `myfile.tar.gz`

Un file non può essere eliminato a meno che non si abbia il permesso di scrittura nella directory genitrice (oltre al permesso in scrittura sul file stesso :-). In questi casi è comunque possibile eliminare o alterare il contenuto del file senza eliminarlo

Se si imposta solo il permesso  $x$  riguardo ad una directory si rende possibile la visualizzazione dei nomi delle voci ma non è possibile visualizzare il contenuto dei file

Impostare il permesso di scrittura su un file per tutti gli utenti è un vero e proprio invito all'abuso: un eventuale "simpaticone" può sfruttarlo a proprio vantaggio in vari modi!

I permessi di tipo world non prevalgono su quelli di group e owner, e quelli di group non prevalgono sui permessi di owner. E' quindi possibile creare un file con permessi 077: tutti tranne l'owner hanno accesso al file: mah...

Quando viene creato un nuovo file il sistema applica i permessi di base. Il comando **umask** permette di impostare questi permessi: per esempio potremmo impostare 700 se siamo veramente bisognosi di privacy!

E' possibile modificare anche owner e group di un file con i comandi **chown** e **chgrp**. La sintassi è **chown user file** oppure **chgrp group file**

### Set UID

Spesso è necessario che un utente debba accedere a risorse che normalmente gli sono negate e devono quindi utilizzare permessi "superiori". Il comando **su** è un esempio, permette di "prendere in prestito" i permessi di un altro utente a patto di conoscerne la password. Il problema è che un utente può dover accedere ad alcuni file importanti che hanno root come owner e la password di root è bene rimanga il più segreta possibile!

I primi due bit dei permessi vengono chiamati **set UID** e **set GID** e servono a risolvere questo problema: quando un utente lancia un comando con tali permessi ottiene i permessi dell'owner e/o del group del comando stesso.

Come esempio lampante prendiamo il cambio di password utente mediante il comando **passwd**: tale comando appartiene a **root.root** (ovvero owner=root e group=root) e ha impostati i bit set UID. La password utente si trova nel file `/etc/passwd` che non permette l'accesso in scrittura da parte di nessun utente che non sia root (per fortuna!!).

Quando un utente qualsiasi lancia **passwd** la shell denota che il programma è set UID e quindi imposta i permessi di root e permette di scrivere sul file `/etc/passwd`: voilà!

In ogni caso i programmi di questo tipo (specialmente quando l'owner è root) sono potenzialmente pericolosi: un attaccante può cercare eventuali bug in programma set UID e ottenere accesso al sistema (es.: provocare uno stack overflow su un programma che gira con i permessi di root per ottenere una shell con permessi di root)

### Sticky

Il terzo bit dei permessi è detto sticky e lascia il programma in memoria per risparmiare il tempo di avvio quando esso viene richiamato. E' raramente usato in quanto riduce la memoria disponibile del sistema in base alla dimensione del file. Si può creare un programma con impostato questo bit che non fa altro che replicarsi rimanendo in memoria fino al riempimento delle aree di swap, il che porta ad un denial of service

## SYSTEM INIT

Lo standard per i sistemi linux è il **System V Init**. Si tratta di un programma lanciato dal kernel all'avvio del sistema che si occupa di lanciare tutti i processi di base. System V introduce il concetto di **runlevel** ovvero di modalità di funzionamento:

- 0 Sistema spento
- 1 Single user (utilizzato da root per la manutenzione del sistema)
- 2 Multiuser senza nfs
- 3 **Multiuser con servizi di rete (Default)**
- 4 Non utilizzato
- 5 Modalità grafica
- 6 Reboot

Ogni runlevel definisce una diversa modalità di funzionamento quindi un set diverso di processi lanciati all'avvio.

Tutte le informazioni di configurazione del sistema si trovano in */etc/rc.d*. La sottodirectory *init.d* contiene gli script di avvio di tutti i processi che **possono** essere lanciati all'avvio. I processi che verranno lanciati dipendono dal runlevel, per ognuno dei quali è presente una directory nella forma *rcn.d*. All'interno di queste directory ci sono i link ai files della directory *init.d* il cui nome è composto in tal modo:

- S/K**: la prima lettera sarà una **S** se il link è usato per lanciare un processo, da una **K** se il link è usato per uccidere un processo
- nn**: il numero di due cifre serve per dare un ordine allo start/kill dei processi
- nome**: il nome del comando linkato

Questi sono alcuni dei files presenti nella mia directory */etc/rc.d/rc3.d*: **S85httpd**, **S86samba**, **K20nfs** ecc...

Ma chi (e in che ordine!) lancia i processi?? Il primo script lanciato dal programma *init* è **rc.sysinit** che lancia i processi validi per tutti i runlevel (viene eseguito una sola volta). Dopodichè viene lanciato lo script **rc** che lancia i servizi in base al runlevel (viene chiamato ogni volta che cambia runlevel poichè è possibile cambiare runlevel senza riavviare il sistema!) ed infine viene lanciato *rc.local*

Il runlevel di default si trova nel file */etc/inittab* nel quale ci sarà una riga simile:

*id:3:initdefault* che indica che il sistema parte in runlevel 3. Mi raccomando: non settate questo valore a 0 oppure a 6! Il sistema non partirà più! Per avere informazioni aggiuntive rimando alla man page di *init*, è completa in tutti i passaggi

## NETWORKING

Linux è "nato e cresciuto" in rete ed è soprattutto nel campo del networking che mostra le sue migliori capacità. Che si parli di server o client le applicazioni e la documentazione non mancano, si tratta di fare un po' di pratica prima con la configurazione della scheda di rete e con i vari tools a disposizione (ping, traceroute, tcpdump ecc...) per poi passare ai vari servizi che un server può gestire

Sections:

[Configuration](#), impostare i vari parametri della connessione

[Networking tools](#), utilizzo dei vari tool standard

[Protocols](#), conoscere i protocolli di rete

## NET CONFIGURATION

La prima cosa (e anche la più semplice!) da impostare è il nome di rete della macchina. Se in fase di installazione non è stato settato o modificato sarà localhost, ma non è molto carino :-)

Per impostare il nome di rete della macchina si usa il comando **hostname** seguito dal nome che vogliamo usare: nulla di più facile!!

Per impostare i settaggi del tcp/ip si usano due comandi: **ifconfig** e **route**

- **ifconfig** serve a configurare l'indirizzo ip della scheda (interfaccia) di rete, la netmask ecc...
- **route** serve a configurare le route statiche e i gateway

**N.B.:** le modifiche fatte con questi due comandi sono attive finchè la macchina rimane accesa. Al riavvio verranno reimpostati i setting precedenti.

All'avvio vengono utilizzati uno o più files (in base al numero di schede di rete) per impostare i settings. Nella RedHat 7.0 questi files si trovano in `/etc/sysconfig/network-script` e si chiamano **ifcfg-eth0**, **ifcfg-eth1** e così via per le altre schede di rete (anche se è poco probabile averne più di due o tre!)

Esempio di ifcfg-eth0:

```
DEVICE=eth0
IPADDR=192.168.1.120
NETMASK=255.255.255.0
NETWORK=192.168.2.0
BROADCAST=192.168.1.255
ONBOOT=yes
```

**Ifconfig** permette di impostare ip address, netmask, broadcast e altri settings sul protocollo. Permette inoltre di attivare o disattivare una determinata interfaccia. Questi sono i parametri più importanti e la sintassi del comando:

<b>interface</b>	indica l'interfaccia di rete a cui riferirsi (es.: <b>eth0</b> , <b>eth1</b> , <b>ppp0</b> ecc...)
<b>address</b>	imposta l'indirizzo ip dell'interfaccia specificata
<b>up / down</b>	attiva o disattiva l'interfaccia specificata (vedi anche <b>ifup</b> e <b>ifdown</b> )
<b>netmask x.x.x.x</b>	imposta la netmask
<b>broadcast x.x.x.x</b>	imposta l'indirizzo di broadcast
<b>-broadcast</b>	non utilizza il broadcast
<b>arp / -arp</b>	utilizza o meno l'arp
<b>mtu n</b>	imposta a n il Maximum Transfer Unit
<b>multicast</b>	imposta il flag di multicasting
<b>promisc</b>	utilizza l'interfaccia in modalità promiscua (vedi Protocolli di rete)

Qualche esempio:

- **ifconfig eth0 192.168.0.56 up netmask 255.255.255.0**

Semplicemente imposta ip address e netmask della prima scheda di rete (eth0) e la attiva

- **ifconfig eth1 down**

Disattiva la seconda scheda di rete (eth1)

- **ifconfig eth0 192.168.0.1 up netmask 255.255.255.0 -broadcast -arp**

Imposta e attiva la scheda di rete eth0 senza utilizzare broadcast e arp

**Route** permette di stampare e impostare le route statiche verso le altre reti. I parametri più usati sono i seguenti:

<b>del n</b>	elimina la route indicata
<b>add</b>	aggiunge una route alla lista
<b>default</b>	la route che si sta aggiungendo sarà quella di default
<b>gw x.x.x.x</b>	imposta l'indirizzo del gateway

Qualche esempio:

- **route del 2**

Elimina la seconda route della tabella

- **route add default gw 192.168.2.254**

Aggiunge la route verso il gateway 192.168.2.254 e la imposta come predefinita (impostare il default gateway :-)

Abbiamo così una (o più) schede di rete configurate e funzionanti a livello tcp/ip. Una cosa che manca e che può servire (o essere indispensabile: ad esempio se si vuole navigare in internet) è la specifica dei server dns. Bisogna aggiungere la direttiva `servername x.x.x.x` nel file `/etc/resolver.conf` per ogni dns che vogliamo utilizzare. Si può specificare due o tre volte lo stesso server per fare in modo da bypassare errori di connection timeout

Dopodichè si può impostare anche il file `/etc/hosts` (con righe del tipo `192.168.2.1 zyxel`) in base ai vari hosts che potrebbero servire.

L'ultimo file da controllare è `/etc/` in cui è specificato l'ordine di ricerca per la trasformazione da hostname a ip address. La riga di default è `order=hosts,bind` che impone di ricercare prima all'interno del file hosts e poi richiedere al dns

### **Esempio di configurazione**

Ecco un esempio di come ho configurato il mio server di prova secondo le caratteristiche della rete:

- Ho una sola scheda di rete
- Ci sono due dns (primario e secondario)
- C'è un gateway per "uscire" verso internet

```
# echo servername 172.30.0.1 > /etc/resolver.conf
# echo servername 172.30.0.2 >> /etc/resolver.conf
# echo order bind,hosts > /etc/host.conf
# hostname zabsrv
# route add default gw 172.30.0.254
# ifconfig eth0 up 172.30.0.10 netmask 255.255.255.0
```

Le prime due righe creano un nuovo `resolver.conf` (fate un backup di sicurezza prima!) con indicati i due dns della rete. Logicamente si può fare anche con un editor ma così ho fatto prima!

La terza riga imposta crea un nuovo `host.conf` (backup come sopra!) impostando che vengano consultati i dns prima del file hosts (che a me non interessa!)

Non mi soffermo sulla quarta riga :-)

Le ultime due righe impostano il default gateway, i settings ip della scheda di rete e l'attivano. Faccio qualche ping, provo a navigare e testo i servizi che mi servono: se funziona tutto salvo i settaggi in `ifcfg-eth0` come spiegato in precedenza

## **NETWORKING TOOLS**

Linux mette a disposizione molti tools per la gestione e il controllo della rete. In questo capitolo verrà mostrato il funzionamento e le potenzialità dei più importanti. Vengono trattati anche softwares non standard (vedi `nmap`) che sarebbero da integrare nel sistema viste le funzionalità.

### **Ping**

Credo che **ping** sia il comando più usato in assoluto da chiunque lavori con le reti! Serve a verificare che un determinato computer sia raggiungibile o meno. Quindi lo si usa per verificare possibili errori di configurazione di un client oppure il crash di un server (giusto per citare due cose "poco" rare :-): se tre client pingano il server ma un altro no possiamo dire che c'è un problema sul client. Se al contrario nessuno pinga il server possiamo credere che sia (magari!!!) un hub o uno switch spento o guasto oppure che il server che cerchiamo di pingare sia "giù" :-)

Come funziona ping? Semplicemente invia all'indirizzo indicato un messaggio **icmp** (vedi protocolli di rete e firewall) di tipo *Echo Request* e aspetta la risposta che è un icmp di tipo *Echo Reply*

Ping ha svariate funzioni ma l'essenziale è ping *ipaddress*. E' possibile specificare un hostname al posto dell'ip address ma bisogna verificare che il dns funzioni e sia raggiungibile (o che il file hosts contenga gli ip delle macchine che ci interessa pingare)

### **Trace Route**

A causa di errori di configurazione di routers o gateway è possibile che i pacchetti che devono andare da una parte all'altra di più reti "muoiano", ovvero che il Time To Live superi il limite massimo. Il TTL è il numero di salti (hops) che un pacchetto può fare fra i router prima di raggiungere a destinazione. Quindi è possibile (anzi frequente!) che , per esempio, due routers si palleggino pacchetti fino a ucciderli. Se provassimo a fare

un ping avremmo la triste risposta TTL exceeded o qualcosa di simile :-)

Ci viene in aiuto **tracert** che, mediante pacchetti icmp, serve appunto a farci vedere tutti gli hops che i nostri pacchetti compiono fino a raggiungere la destinazione o il max TTL. Quindi risulta facile vedere fra quali hosts ci sia il palleggio e correggere l'errore (anche se la seconda parte della frase non è sempre facile!!)

### **NameServer Lookup**

Quando lavoriamo in rete la conversione dei nomi host in indirizzi ip avviene mediante il dns in modo trasparente, senza dover far nulla. Se invece volessimo sapere quale ipaddress corrisponde ad un dato hostname o viceversa grazie a **nslookup** possiamo interrogare un dns. Questo tool si appoggia al primo dns che trova in `/etc/resolver.conf` ma è comunque possibile interrogare un qualsiasi altro server (basta saperne l'ip address :-)

### **TCP Dump**

Può capitare che alcuni pacchetti indesiderati arrivino verso una o più macchine. Il tool **tcpdump** ci permette di monitorare il traffico su un'interfaccia di rete del sistema (eth o ppp) relativamente a tutti i pacchetti tcp o solo ad alcuni in base alla porta di comunicazione. In tal modo possiamo sapere chi stia inviando un certo pacchetto oppure verso chi il nostro linux stia inviando. N.b.: non è uno sniffer! Agisce sulle interfacce della macchina su cui viene utilizzato

### **Sniff It**

Questo è un tool molto utile! E' necessario settare l'interfaccia di rete in modalità promiscua (vedi ifconfig) in modo che riceva tutti i pacchetti. Ora, controllando l'header ip dei pacchetti, possiamo sapere quale host sta comunicando con un certo server/sito e nel caso i dati siano in plain text è possibile leggere la comunicazione. Per esempio, controllando i pacchetti che si dirigono alla porta 23 di un host è possibile controllare il traffico di tipo Telnet (e quindi anche la password che in Telnet non è cifrata!) oppure sniffando i pacchetti destinati alla porta 110 di un pop server possiamo leggere la posta ricevuta e così via! E' quindi **importantissimo** che le comunicazioni importanti siano cifrate (pgp per le mail, ssh per la gestione remota ecc...).

Usare uno sniffer non è moralmente corretto, comunque è facile reperire dei software sentinella che in base ai ritardi di rete possono rilevarne la presenza (quindi l'ip address sorgente e lo "sniffatore")

### **Nmap**

Tool non standard, **nmap** è utilissimo per vedere quante porte aperte abbia un host. E' quindi molto usato sia dopo la creazione di un server (si trovano le porte aperte e vulnerabili) sia in fase di "attacco". Ha molte opzioni e per comodità è stato creato un frontend (**nmapfe**) veramente molto utile. Si possono scaricare i sorgenti o l'rpm da [www.insecure.org/nmap](http://www.insecure.org/nmap). A mio parere è un must have!

## **PROTOCOLS**

In questi capitoli vengono trattati i protocolli di comunicazione maggiormente diffusi discutendone gli utilizzi e le debolezze.

La pila protocollare di riferimento è la seguente:

**FISICO - DATALINK - IP - TCP - APPLICATIVI**

In ogni caso consiglio di leggere Computer Networks di Tanenbaum o qualche libro simile!

## **TCP/IP**

Con questa sigla si definisce l'accoppiata Transfer Control Protocol/Internet Protocol che in realtà sono due protocolli diversi che lavorano a livelli diversi della pila protocollare. Tale accoppiata è diventato lo standard mondiale per la comunicazione (Internet :-)

e sopra di essa sono stati costruiti i protocolli applicativi. L'Internet Protocol definisce che ogni host in rete deve essere individuato da un indirizzo IP ovvero da 4 numeri (compresi tra 1 e 254) separati da punti: 192.168.2.3, 10.10.1.10 ecc...

Il Transfer Control Protocol (che lavora a livello più alto) si occupa di trasferire i dati spezzandoli in pacchetti più piccoli e di assicurare che tutti i pacchetti arrivino in ordine corretto ovvero che l'informazione sia

trasmessa integra (protocollo connesso)

Il tcp introduce il concetto di porta: come identificare i dati relativi ad una mail ricevuta dai dati ricevuti da un sito internet? Ad ogni servizio è assegnato un numero (incluso nell'header tcp) per cui non c'è confusione tra i flussi di comunicazione. Una lista di servizi si trova in */etc/services*, in ogni caso ce ne sono molte altre e se si vuole creare un server in ascolto su una determinata porta bisogna assicurarsi che non sia già "occupata". Un header contiene le informazioni relative al pacchetto: mac address, ip address e porta di origine, mac address, ip address e porta di destinazione. Tali informazioni sono importantissime (sia per il controllo che per lo sniffing!) per questo motivo: nelle reti ethernet tutti gli host ricevono tutti i pacchetti ("parla" un host alla volta) e solo se il mac address di destinazione è giusto il pacchetto viene accettato. E' comunque possibile impostare la propria scheda di rete in modalità promiscua ovvero fare in modo che legga tutti i pacchetti: un software di analisi potrebbe quindi loggare i dati interessanti (es.: copiare su un file tutti i pacchetti che hanno come destination ip un server e come porta la 23: si ottengono le password e i comandi digitati). Nel capitolo Networking Tools alla voce sniffit ci sono maggiori chiarimenti sull'uso e sulla "moralità" dello sniffing.

## UDP

User Datagram Protocol è simile al tcp, con la fondamentale differenza che i pacchetti non è detto che arrivino!

## ARP

Ogni scheda di rete è individuabile da un MAC address costituito da 6 byte (secondo quanto definito a livello Data Link). Arp (Address Resolution Protocol) viene utilizzato per associare un MAC address ad un IP address. Questi dati si trovano nella arp cache di ogni host che è manipolabile dal comando arp (è presente in tutti i sistemi ma non è mai necessario :-)

La arp cache viene manipolata anche in base ai messaggi arp esterni e gli IP address vengono assegnati "a gara": più un host con un determinato MAC afferma di avere un determinato IP più il nostro host crederà che sia vero: la tecnica di "fingere" di avere un altro IP address (e quindi ricevere i pacchetti destinati all'originale) è chiamata **IP Spoofing**, una ricerca veloce su [PacketStorm](#) troverà documenti relativi all'argomento

## ICMP

Internet Control Message Protocol è un protocollo che si trova a livello IP ed è utilizzato (come dice il suo nome :-)) per lo scambio di informazioni di controllo sugli host in rete. Non ci sono le porte ma i tipi di messaggio: per esempio un ping invia un messaggio di tipo *echo-request* e la risposta sarà di tipo *echo-reply*.

## SIMPLE MAIL TRANSFER PROTOCOL

Il protocollo smtp (e relative estensioni come ESMTP) è il protocollo standard utilizzato per l'inoltro di posta elettronica. Il demone smtp sotto unix e linux è **sendmail** che è in ascolto alla porta **25**. Sendmail è sia il più utilizzato che il più "bacato" al mondo (basta cercare su un database di sicurezza e si troveranno un buon numero di exploit!) e la conoscenza dei comandi del protocollo smtp permette sia di inviare/rilevare le fake mail che di trovare banchi nel server.

A differenza del protocollo pop non è richiesta autenticazione ma in base all'ip address con cui ci si connette sarà impedito o meno di inviare mail (errore di *Relaying not permitted*)

Una volta connessi si riceve il daemon banner (una stringa contenente varie informazioni sul server, molto utile in quanto è possibile rilevare la versione del demone e quindi trovarne exploit). Segue un esempio esplicativo di una sessione smtp:

```
helo name
  nome che verrà visualizzato come sender
mail from: sender address
  indirizzo e-mail del sender
reply_to other address
```

indirizzo a cui inviare la risposta (diverso dal sender address ;-)  
rcpt to: *destination address*  
indirizzo e-mail del destinatario, può essere ripetuto per tutti i destinatari che si vuole  
cc: *other dest address*  
destinatari in carbon copy  
bcc: *other dest address*  
destinatari in blinded carbon copy  
data  
inizia a scrivere la mail  
subject: *subject*  
inserisce il subject della mail  
<corpo della mail>  
terminare la mail con un "." in una riga vuota

## POST OFFICE PROTOCOL

Il pop è il protocollo più usato per il controllo e la gestione degli account di posta elettronica. Il demone pop è in ascolto sulla porta **110** del server e mediante il programma telnet è possibile accedere al server e "fingere" di essere un client di posta. In tal modo da qualsiasi parte del mondo e da qualsiasi sistema che abbia almeno telnet è possibile accedere alla propria posta senza pericoli di password dimenticate nelle cache, nel completamento automatico ecc... oppure si può ripulire piuttosto facilmente un mail bombing

Una volta connessi si riceve il daemon banner (una stringa contenente varie informazioni sul server). E' quindi necessario autenticarsi:

```
user username
pass password
```

Se le informazioni inserite sono corrette si riceverà un messaggio di conferma indicante il numeri di messaggi giacenti altrimenti si riceverà un messaggio di bad login  
Per leggere i messaggi si usano i seguenti comandi:

E' quindi necessario autenticarsi:

```
list mostra l'elenco delle mail
retr # stampa il messaggio numero #
dele # cancella il messaggio #
```

Ci sono un paio di cose di cui tenere conto:

- quando si viene loggati dal sistema viene creato un file lock che impedisce altri accessi con la stessa utenza per tutta la durata del collegamento
- è utile impostare la visualizzazione dell'eco locale per visualizzare ciò che si scrive (di default non è impostato)
- tutta la comunicazione (quindi anche la password) avviene in plain text

## COMPILARE IL KERNEL

Il kernel è il cuore del sistema operativo in quanto tutte le operazioni passano attraverso esso. Un sistema linux funzionante consiste di un kernel, una shell e qualche applicazione di controllo.

In genere tutte le distribuzioni installano un kernel precompilato a cui vengono aggiunti i moduli in base all'installazione.

Perchè ricompilare il kernel?

- aggiornare il sistema senza reinstallare da capo
- aggiunta di nuove funzionalità
- implementazione di nuovi drivers

Su [www.kernel.org](http://www.kernel.org) è possibile scaricare l'ultima release del kernel o le relative patch. Il kernel è una tarball zippata (ovvero con estensione .tar.gz oppure .tar.bz in base all'utility usata per la compressione) che

contiene tutto il codice sorgente di linux e le dimensioni sono generalmente abbastanza elevate (il kernel 2.4.1 compresso con bzip occupa circa 24mb).

Tale file avrà un nome nel formato **linux-x.y.z.tar.gz** in cui x.y.z indicano la major version, la stabilità (se y è pari il kernel è stabile e testato altrimenti è sperimentale e potrebbe dare dei problemi: consigliato a chi è esperto e non si scandalizza davanti a qualche blocco di sistema) e la release.

Nel caso non vogliate scaricare tutto il kernel dovete scaricare TUTTE la patch uscite dalla release del vostro kernel fino alla release che volete "raggiungere" e applicarle in ordine una dopo l'altra (es.: se io possiedo il kernel 2.2.12 ma voglio passare al 2.2.14 devo scaricare i files **patch-2.2.13.gz** e **patch-2.2.14.gz**).

### Procedura di ricompilazione

Per prima cosa copiate il file **linux-x.y.z.tar.gz** in */usr/src* e decomprimetelo con il comando **tar xvfz linux-x.y.z.tar.gz**

Poi create i seguenti link:

**In -s /usr/src/linux/include/asm-i386 /usr/include/asm,**

**In -s /usr/src/linux/include/linux /usr/include/linux,**

**In -s /usr/src/linux/include/scsi /usr/include/scsi**

Spostatevi quindi in */usr/src/linux* e usate il comando **make mrproper** per eliminare eventuali files che potrebbero interferire

Ora si può passare alla configurazione dei componenti del kernel. Ci sono tre tool utilizzabili:

- make config** che funziona su linee di testo, è obsoleto!
- make menuconfig** che funziona in modalità testo con le curses
- make xconfig** che funziona in modalità grafica

Gli ultimi due tools si equivalgono, scegliete quello che preferite!

Una volta selezionati tutti i componenti che volete salvate la configurazione e uscite

Tip: Dopo questa operazione viene creato un file di nome Makefile che contiene tutti i settaggi e le indicazioni per la compilazione: sarebbe opportuno copiarlo da qualche parte nel caso dovesse servire ancora

Tutte le componenti realmente essenziali devono essere incluse nel kernel, quelle che non sono necessarie possono essere installate come moduli: questo fa sì che il kernel monolitico abbia le funzionalità necessarie al sistema e che le altre possano essere richiamate quando servono. Logicamente bisogna trovare un compromesso: nè un kernel monolitico enorme nè troppi moduli!

Passiamo ora alla compilazione del kernel:

**make dep** per compilare le dipendenze

**make clean**

**make zImage** per creare l'immagine del kernel (oppure **bzImage** per avere un'immagine compressa)

Nel caso abbiate selezionato dei moduli allora è necessario compilarli ed installarli:

**make modules** per compilare i moduli selezionati

**make modules\_install**

Alla fine di questo processo troveremo il file **zImage** (o **bzImage**) nella directory */usr/src/linux/arch/i386/boot* che è il nostro kernel monolitico. Per testarne il funzionamento abbiamo due possibilità:

- Copiare il kernel su un floppy (con il comando **dd if=/usr/src/linux/arch/i386/boot/zImage of=/dev/fd0**) da usare per avviare il sistema
- Spostare il kernel in */boot* (magari rinominandola per maggior chiarezza!) e aggiornare **lilo.conf** inserendo le linee:

```
image=/boot/newkernel
```

```
label=newkernel
```

```
root=/dev/hda1
```

Ricordate di lanciare lilo per aggiornare il master boot record!

Ora bisogna riavviare il sistema e incrociare le dita! Nel peggiore dei casi sarà da fare tutto da capo! Nel caso ci fossimo dimenticati qualche impostazione sarà sufficiente usare make xconfig, caricare il Makefile e aggiungere/rimuovere le configurazioni e di nuovo compilare

<a href="http://www.kernel.org">www.kernel.org</a>	Sito ufficiale per la distribuzione del Linux Kernel (sia le versioni stabili che quelle in via di sviluppo)
<a href="http://www.pluto.linux.it">www.pluto.linux.it</a>	Italian Linux Documentation Project
<a href="http://www.linuxgazette.com">www.linuxgazette.com</a>	Sito ufficiale della Linux Gazette
<a href="http://www.gnu.org">www.gnu.org</a>	Sito (con vari mirror tra cui quello italiano!) in cui viene spiegata la filosofia del progetto Gnu (Gnu's Not Unix!)
<a href="http://www.apache.org">www.apache.org</a>	Sito ufficiale del webserver Apache
<a href="http://packetstorm.securify.com">packetstorm.securify.com</a>	Database sulla sicurezza