Come sopravvivere alla shell di Linux
e
vivere felici e contenti

Michele Andreoli

Indice

Capitolo 1

Introduzione

1.1 Scopo del Corso

Così come studiamo la Storia antica per capire meglio la Storia Moderna, è giusto studiare il sistema operativo UNIX, pilastro fondamentale della storia dell'informatica.

Inoltre, si va diffondendo sempre più nel mondo una moderna incarnazione di UNIX, chiamata Linux: un sistema distribuito gratis e sviluppato in maniera collaborativa da persone di varie parti del mondo, che si tengono in contatto grazie ad Internet.

La conoscenza di Linux può fornirti una possibilità in più nell'affrontare il mondo del lavoro, oltre ad essere una valida esperienza culturale.

Non dimenticare, infatti, che nelle attività di ricerca universitaria, il sistema usato è sempre Linux: perché è *gratis*, perché è *aperto,* perché è *espandibile,* e perché gira dal piccolo dispositivo al grande Super Computer.

Il sistema operativo Android è una versione modificata di Linux.

1.2 I componenti di un sistema Linux

Linux è un sistema operativo (OS, Operating System) per Personal Computers (PC), basato sulla struttura e sulla filosofia del originale sistema UNIX, sviluppato al MIT negli anni 60/70.

Linux è stato creato dallo studente finlandese Linux Torvalds, intorno al 1993.

Esso si compone da:

1. un kernel: il cuore del sistema e che gestisce le periferiche del computer.
2. una shell: l'interprete a linea di comando, che permette l'uso interattivo del sistema (copiare files, cancellare, editare, etc)

3. un server grafico: un programma per accedere alla scheda grafica e disegnare finestre, bottoni, etc. (denominato XWindow).

4. un Window Manager: il programma che gestisce le finestre e quindi il desktop (la scrivania). I due desktop più in voga al momento sono KDE (sponsorizzato dal governo tedesco) e GNOME.

5. una distribuzione: cioè un confezionamento su CD-ROM, con manuali, etc (RedHat, Debian, Caldera, Mandrake, Ubuntu, Mint, ...)

1.3 Ulteriore documentazione

Un buon punto di partenza su Linux in Italia è il sito www.linux.it.

1.3.1 Aggiornamenti di questo documento

Puoi scaricare una nuova versione di questo documento da questo sito:
http://micheleandreoli.org/public/Didattica/materiali/corso-unix

1.4 Notazioni

Se davanti ad una riga è presente il simbolo # (cancelletto), vuol dire che si tratta di un comando UNIX che potrebbe essere battuto sulla tastiera.

Naturalmente, lo inserirai SENZA il cancelletto.

Ad esempio, con la riga:

```
# vi prova.txt
```

si vuol dire che devi battere il comando vi prova.txt

Capitolo 2

Il Free software

2.1 Concetto di Software Libero

Il software, anche se forse non ci abbiamo mai fatto caso, è una delle più importanti merci del mondo moderno. Esso influenza la nostra vita, la sicurezza delle nostre comunicazioni e dei nostri dati, in fondo: la nostra libertà.

Software libero significa avere il controllo su quello che gira sul nostro computer; significa poter aprire il cofano della nostra auto.

Software libero significa avere ad disposizione i sorgenti dei programmi, non solo l'eseguibile del programma. Significa poterlo modificare, poterlo redistribuire, poter contribuire al suo miglioramento e vedersene riconosciuto il merito.

Paladino di questo modo di intendere il software è Richard Stallman, ex ricecatore del MIT e fondatore del manifesto GNU.

2.2 Il futuro del Free Software

Il concetto di Free Software sta guadagnando popolarità non solo tra gli appassionati (smanettoni ed hackers di vario genere) ma anche tra aziende ed instituzioni di vari paesi.

Il governo tedesco, ad esempio, sponsorizza KDE, che è un desktop manager molto amato dalla comunità Linux.

Nel parlamento europeo (ma anche in quello italiano) sono all'esame proposte di introduzione di Linux nella didattica e nelle pubbliche amministrazioni.

Per quanto riguarda la ricerca universitaria (specie quella scientifica) Linux è già presente da tempo. Perché è libero, perché i ricercatori già conoscevano UNIX, perché lo si può modificare per scopi particolari senza dover pagare diritti a terzi.

Capitolo 3

La struttura del filesystem

3.1 La gerarchia delle directories

I dischi UNIX sono organizzati in una struttura gerarchica di directory, esattamente come DOS/WIN, a partire da quella che si chiama *root*, ed è indicata con «/».

Eccone un esempio:

```
|-- bin
|-- etc
|   |-- cron
|   `-- ppp
|-- home
|   |-- paperino
|   |-- pippo
|   `-- pluto
|-- lib
|-- sbin
`-- usr
|-- bin
|-- doc
|-- lib
`-- sbin
```

- La directory /bin (*binary*) contiene comandi eseguibili, ad esempio.
- La directory /sbin (*system binary*) contiene comandi per l'amministrazione del sistema.
- /home contiene i dati dei singoli utenti.

- /usr/bin e /usr/sbin contengono i comandi accessibili a tutti gli utenti,e non solo all'amministratore, etc etc.
- /lib contiene le *librerie dinamiche*; /usr/doc la documentazione, etc.

3.2 Nome completo (o assoluto) di un file

Il nome completo di un file deve specificare l'intero percorso (path) che conduce fino al file stesso, a partire dalla radice (*root*) che viene indicata con «/» (*slash*). Esempi:

```
/home/paperino/immagini/auto.jpg
/bin/ls
...  etc  ..
```

Nel Dos invece che / (*slash*) si usa «\» (*backslash*); la root viene invece indicata col nome del disco: C:, D:, A:, etc

Nella notazione del DOS scriveremmo per gli stessi files:

```
c:\home\paperino\immagini\auto.jpg
...  etc  ...
```

3.3 Parte directory e parte file di un file assoluto

In un file assoluto come */home/paperino/immagini/auto.jpg*, la parte directory è */home/paperino/immagini/* e la parte file è auto.jpg

Possono essere estratte con i comandi *dirname* e *basename*.

3.4 Navigare nell'albero delle directory

Per questo esiste il comando cd (change directory), che si usa come l'analogo del DOS.

Esempio:

```
# cd /home/paolo
```

ci porta, come directory corrente, nella directory /home/paolo

Se vogliamo salire di un livello di directory, basta scrivere:

```
# cd ..
```

.. è un nome di directory speciale: simboleggia sempre la directory più in alto rispetto a quella dove ci troviamo.

Capitolo 4

Sintassi dei comandi UNIX

Uno dei comandi più importanti (collocato in /bin) è il comando ls (list), ed è l'equivalente del comando DOS dir.

4.1 Argomenti di un comando

La sintassi del comando ls, ma anche quella di tutti gli altri comandi, segue una regola ben precisa:

```
# nome_comando argomenti ...
```

Gli argomenti sono delle stringhe che si appongono, separate da spazi, dopo il nome del comando stesso. Essi possono essere obbligatori o facoltativi.

Per indicare che un argomento è facoltativo, in genere, nella documentazione del comando, lo troviamo racchiuso tra parentesi quadre [...].

Oltre a questa suddivisione, ve ne è un altra: un argomento può essere di tipo opzione (switch) o di tipo dato. Quelli di tipo opzione vengono prima di quelli di tipo dato, ed iniziano sempre con in -, qualche volta con –

4.1.1 Esempi:

————–

```
# ls –l /bin
```

Qui abbiamo richiesto la lista (ls) della directory /bin, specificando che vogliamo il formato lungo (-l, long).

```
# ls –F /bin /usr/bin
```

Qui vogliamo la lista di due directory, specificando il formato su più colonne (-F).

```
# ls --sort=time /bin
```

Qui vogliamo che la lista sia ordina in base alla data di creazione.

```
# ls -R /
```

Qui abbiamo richiesto la lista della directory radice (/), specificando che vogliamo una lista ricorsiva (-R), cioè di tutte le sottodirectory. In pratica, in questo caso, avremo la lista di TUTTI i files presenti nel sistema

```
# ls -l
```

Qui abbiamo omesso il nome della directory. In questo caso, il comando assume che vogliamo listare la directory corrente, cioè quella dove ci troviamo quando abbiamo battuto il comando.

```
# ls -a
```

Mostra tutti (-a, all) i files, anche quelli nascosti.

4.2 Combinare switch

```
# ls -laR
```

In questo caso chiediamo la lista completa, ricorsiva, di tutti i files, dalla directory corrente a scendere giù.

4.3 Paradigma di un comando

Come in latino, anche i comandi UNIX hanno quello che possiamo chiamare paradigma; in sostanza, una breve illustrazione di come usare il comando (in inglese: *sinopsis*)

Per il comando ls il paradigma è del tipo:

```
# ls [-laFR ...] [directory o files]
```

Con questa scrittura[1] si vuol significare che sia gli switch che i dati sono facoltativi, e che alcuni degli switch più usati sono: -l, -a etc

4.3.1 Nota

Il comando ls ha circa 200-300 switch diversi! Io ho illustrato solo le più usate.

[1] cioè con le parentesi quadre []

4.4 Tipici comandi di UNIX

La lista di comandi riportata qui non è completa. Per avere una lista completa, il modo più semplice è controllare il contenuto delle directory dove si trovano in genere i comandi: /bin, /sbin, /usr/bin e /usr/sbin

Ad esempio:

```
# ls /bin
```

Non è completa perché, di volta in volta, io aggiungo/rimuovo altri comandi al sistema.

4.5 Help di un comando

In genere, si ottiene l'help con lo switch –help oppure -h. Esempio:

```
# ls --help
```

4.6 Amministrazione del filesystem

ls (list) lista files/directory

mkdir (make dir) crea directory

rm (remove) cancella

mv (move) sposta e rinomina

cp (copy) copia

find (find) cerca files

cat concatena files] description

cd (change dir) cambia directory

4.7 Cambiare i permessi di un file

```
# chmod (change mode)
```

4.8 Archiviazione

tar (tape archiver) Mette insieme più files in un unico archivio (ma non comprime)

4.9 Compressione

gzip (Gnu zip) Comprime un file (compressione lossless)

zip (pkzip) Compressore e archiviatore, origine DOS

4.10 Estrazione informazioni da files

wc	(word count) conteggi parole e righe
grep	selezione di righe in base a pattern
tr	(translate) rimpiazza un carattere con un altro
sed	(stream editor) rimpiazzia intere parole
dd	estrae pezzi di un file

4.11 Editing

vi (visual editor) Difficile da usare. Tradizionale UNIX edit Facile da usare.

4.12 Matematica

expr semplici calcoli algebrici

bc calcolatore scientifico

4.13 Criptografia

md5sum (Message Digest, Algorithm 5) Generazione di HASH

gpg (Gnu Pretty privacy Guard) criptografia e firma elettronica

4.14 Networking

ping per pingare un computer in rete telnet per aprire una sessione su un computer remoto

ssh per aprire una sessione criptata (s=secure) su un computer remoto

ftp per scaricare files via FTP

wget per scaricare pagine web da un computer remoto curl idem

4.15 Miscellanea

echo stampa a video di una stringa date data corrente

du (disk usage) Informazione sullo stato dei dischi

df Informazione sui punti di montaggio dei dischi

mount/umount Montaggio e smantaggio dischi (avanzato)

dirname estrare la parte directory da un file assoluto basename estrane la parte file da un file assoluto

Altri che mi sono dimenticato di menzionare.

4.16 Una tipica sessione di lavoro

```
# mkdir animali                  creo la directory animali
# cd animali                     vi entro dentro
# mkdir vert invert              creo le sottodirectory vert e invert

# cd vert                        entro nella sottodirectory vert
# edit gatto                     creo ed edito il file gatto
# cp gatto siamese               copio gatto su siamese
# rm gatto                       cancello gatto

# cd ..                          torno su di un livello. Ora sono in anir
# ls                             chiedo la lista.
Dovrei vedere le directory vert e invert
# rm -r invert                   rimuovo la directory invert, con l'opzio
```

```
che significa ricorsiva: rimuovo tutto il
suo contenuto.
# cp -r /etc vert                copio la directory /etc, ricorsivam
nella directory vert. Ora essa contiene un
file (siamese) ed una directory (etc/)

# cd ..                          torno ancora su. Ora sotto di me,
directory animali.
# rm -r animali                  la cancello, insieme a tutte le po
che ho creato in questo esempio
```

4.17 Estensione dei files

Come sapete, nella convenzione DOS i files hanno una parte "nome" e una parte "estensione" di tre caratteri, che segue il punto ".". Esempi:

pippo.jpg nota.txt .....

Il DOS tratta in maniera piuttosto diversa le due parti.

In Linux è diverso. Il punto è considerato un carattere come gli altri.

Possiamo ancora chiamare "estensione" quello che viene dopo il punto, ma per Linux essa non ha un significato particolare. Serve giusto a noi, per identificare facilmente il contenuto del file stesso.

Ne consegue, che l'estensione in Linux non è limitata a 3 caratteri. Esempi:

fire.c gtk.cc nota.txt pipo.configurazione

sono tutti nomi validi.

4.17.1 Funzionamento dei wild-char

Per wild-char si intendono alcuni caratteri "jolly" che si possono usare per specificare i files. Funzionano il Linux in maniera analoga al DOS.

In DOS, ad esempio, si può scrivere:

```
# dir *.jpg
```

intendendo "listami tutti i file con estenzione .jpg"

Benchè il significato sia lo stesso in DOS e Linux, si possono facilemente commettere errori. Vedi paragrafo successivo.

4.17.2 Espansione dei wild-char

La shell cerca sempre di "espandere" il simbolo "*", tutte le volte che è possibile.
Se io batto il comando:

```
# ls *
```

e la directory corrente contiene i files "x y z", il simbolo * (che qui significa "tutti i files") verra espanso. Il vero comando che verrà eseguito sarà quindi:

```
# ls x y z
```

Sta quindi al comando "ls" capire che i tre argomenti x,y,z rappresentano files da listare.
Ma che succede se batto:

```
# cp *
```

Questo comando verrà espanso in:

```
# cp x y z
```

Ora viene il bello: il comando "cp" si aspetta che l'ultimo argomento sia la directory dove copiare. Ecco perché si arrabbierà emettendo il misterioso messaggio di errore:

```
cp: copying multiple files, but last argument 'z' is not a directory
```

Capitolo 5

Filtri di ricerca

In questa lezione si imparerà ad estrarre informazioni da un file, sulla base di un pattern di ricerca. Soltanto le righe del file che soddisfano il pattern (mattern matching) saranno selezionate.

5.1 Il comando grep

Il comando UNIX per fare questo si chiama grep ed ha questa sintassi:

```
# grep [opzioni] pattern file
```

Alcune delle opzioni sono:

- -i ignora la differenza maiuscolo/minuscolo
- -v inverti la ricerca (in output solo quelle che NON soddisfano)

5.2 Simulare un DataBase

Edita il file veicoli e copia all'interno le seguenti righe, aggiungendone altre, se vuoi (senza le righe con i puntini!)

```
... contenuto del file veicoli ......
        4       renault         laguna          france
        4       renault         scenic          france
        2       honda           cbr-900         japan
        2       honda           cbr-600         Japan
        4       honda           civic           japan
        4       ferrari         testarossa      italy
        4       fiat            fiat-500        italy
```

```
4       fiat            fiat-600        italy
2       ducato          monster         italy
.....................................
```

Come si vede, si tratta di un data base relazionale, dove ogni record (cioè una linea) è composto da vari field (campi). In questo caso i campi sono: ruote/casa/-modello/nazione.

5.3 Metacaratteri e loro uso

Il pattern di ricerca viene composto mescolando caratteri e metacaratteri. Alcuni dei metacaratteri più comuni sono i seguenti:

```
.       un carattere qualsiasi, ma uno solo
.*      0,1,2,3 .... caratteri qualsiasi
.+      1,2,3 ... caratteri qualsiasi
.?      0,1 caratteri qualsiasi
[abcD]  solo le quattro lettere a,b,c,D
[0-4]   solo i numeri compresi tra 0 e 4
[f-w]   le lettere comprese tra f e w
^       inizio riga
$       fine riga
```

Esempi di pattern:

```
^2.*italy
        cbr-[69]
```

Il primo pattern seleziona le righe che iniziano per 2, hanno un certo numero di caratteri, e poi la string italy

Il secondo pattern seleziona le righe che contengo la stringa cbr- e poi uno dei due numeri 6 o 9.

5.4 Uso di *grep*

(in una riga c'è il comando, nella successiva c'è il risultato)

```
# grep cbr-6 veicoli

        2       honda           cbr-600         Japan

        # grep ^4.*italy veicoli
```

```
4       ferrari         testarossa      italy
4       fiat            fiat -500       italy
4       fiat            fiat -600       italy

# grep italy veicoli

4       ferrari         testarossa      italy
4       fiat            fiat -500       italy
4       fiat            fiat -600       italy
2       ducato          monster         italy
```

5.5 Tocca a te ...

Prova ad estrarre

- le auto non italiane. - le moto giapponesi. - le auto la cui marca inizi per f o t

Capitolo 6

Standard input (stdin) e standard output (stdin)

Ogni comando UNIX suppone di avere un "input" ed un "output".

Spesso l'output è il video, l'input è la tastiera. Ma non è sempre così.

In un comando come grep, ad esempio, l'output è il video. E se volessimo l'output in un file?

6.1 Redirezione l'output

Come anche nel DOS, l'output si può redirezionare verso un file, usando il simbolo ">".

In questo esempio:

```
# grep "italy" veicoli > veicoli_italiani
```

verrà creato il file "veicoli_italiani", contente solo i veicoli fatti in Italia.

6.2 Redirezione l'input

Questo viene fatto in genere usando le "pipes". Vedi in seguito.

6.2.1 Memorizzare il risulto in una variabile

Questa è una delle caratteristiche più importanti di UNIX. Grazie ad essa, l'output di un comando può essere memorizzato in una variabile, invece che in un file.

Se ho usato la variabile X, per stamparne il valore devo fare "echo $X". (notare il segno di $).

```
# x=5
# echo $x (qui ci stamperà 5)
```

La sintassi per farlo è la seguente:

```
# variabile=$(comando UNIX)
```

Esempi:

```
# lista = $(ls /bin)
# oggi = $(date)
# risultato=$(expr 2 + 3)
```

Capitolo 7

Lavorare con le *pipes*

In UNIX l'output di un comando può essere inviato in input ad un altro comando, e così via.

Basta concatenare i vari comandi con il simbolo di pipe "|", che trovi sul tasto a sinistra in altro, sopra la "\".

7.1 Esempio 1: estrazione righe

```
# cat veicoli | grep "italy"
```

In questo esempio si usa il comando "cat", il quale semplicemente legge il file "veicoli" e lo riscrive in output. L'output viene spedito a "grep", il quale ne spulcia le righe.

In sostanza:

```
# cat veicoli | grep "italy"
```

è perfettamente equivalente al comando:

```
# grep "italy" veicoli
```

solo che, in questo caso, si dice che grep "sta lavorando in pipe".

7.2 Esempio 2: conteggio righe (comando wc)

In questo esempio conosceremo il comando "wc" (word count). Questo comando ha le seguenti opzioni:

- -l conteggio righe (lines)

- -c conteggio caratteri (characters)
- -w conteggio parole (word)

Contare le righe di un file:

```
# cat veicoli | wc -l
```

Contare le macchine italiane:

```
# cat veicoli | grep "italy" | wc -l
```

7.3 Esempio 3: sorting (ordine alfabetico)

Qui impariamo l'uso del comando "sort" (ordina). Questo comando vuole anche sapere quale campo vogliamo usare per l'ordine alfabetico.
Esempio:

```
sort -k 3 ( userà il campo 3)
```

Le moto di ogni marca, ma ordinate per nazione:

```
# cat veicoli | grep "^2" | sort -k 4
```

o per marca:

```
# cat veicoli | grep "^2" | sort -k 2
```

Capitolo 8

Codici *Escape*

Come sai, alcuni dei caratteri della tabella ASCII non sono stampabili e rappresentano "codici di controllo"

8.1 Caratteri ASCII di controllo (codici escape)

I codici escape vengono indicati nella forma \x (col backslash). Eccone alcuni:

\r ritorno carrello (CR, vai a capo)

\n new line (NL, nuova linea)

\a bell (il terminale fa BEEP)

\t tabulazione (tasto TAB)

ed altri ancora.

I più imporanti sono, effettivamente,

\n ottale(12), decimale(10) \r ottale(15), decimale(13)

8.2 Formato dei file di testo

DOS e UNIX utilizzano, purtroppo, una sintassi diversa per segnare la posizione dove la riga è finisce.

Entrambi lo fanno, comunque, inserendo in fondo alla riga dei codici escape, i quali non si vedono, ma che le stampanti o i terminali sanno interpretare come "andata a capo".

8.2.1 Come fa il DOS

Il DOS, in fondo alla riga, inserisce la coppia "\n\r". Quindi, ben due caratteri.

\n "ruota il rullo" di una posizione in avanti \r ritorno a capo del carrello

Effettivamente, se pensiamo a come funzionavano le macchine da scrive, le due operazioni servivano a preparare la stampa della riga successiva.

8.2.2 Come fa UNIX

Purtroppo, UNIX aggiunge solo "\n", e questo effettivamente è un pò illogico. Forse la scelta fu fatta per risparmiare un byte ad ogni riga, dato che i due compaiono sempre insieme.

8.3 Conseguenze

Se scriviamo un file sotto UNIX e poi pretendiamo di vederlo sotto DOS, dato che il DOS non trova lo \r, ci appare come una sola grossa riga :-(

Se scriviamo il file sotto DOS e lo vediamo sotto UNIX, i caratteri spuri \r vengono mostrati da UNIX in fondo ad ogni riga nella forma ^M. Molto spiacevole.

8.3.1 Soluzione

Esistono due programmi *(unix2dos* e *dos2unix*) che fanno la conversione.

Capitolo 9

Stream editors

9.1 Il comando tr

Sostituzione di caratteri singoli

In questa lezione si impara a rimpiazzare caratteri da un file di input, producendo un file di output.

9.1.1 Caratteri ASCII di controllo (codici escape)

Alcuni caratteri ASCII hanno un significato speciale e vengono indicati nella forma \x (col backslash).

- \r ritorno carrello (vai a capo)
- \n new line (nuova linea)
- \a bell (il terminale fa BEEP)
- \t tabulazione (tasto TAB)

ed altri ancora.

9.1.2 Rimpiazzo di singoli caratteri: comando "tr"

Esempi:

1. # cat veicoli | tr i X (2)
2. # cat veicoli | tr -d [0-9] (3)
3. # cat veicoli | tr [a-z] [A-Z] (3)

4. # cat veicoli | tr [\t] X

- Nell'esempio (1) lavorando in pipe, il comando "tr" spulcia le righe del file "veicoli" e traduce tutte le "i" in "X"
- In (2), cancella (-d) tutti i caratteri numerici In (3), mette tutto in maiuscolo In (4), traduce tutti gli spazi e la tabulazione (indicata con \t) con una X

9.1.3 Rimozione caratteri di fine riga

Supponiamo di voler cancellare i codici escape \r e \n che sono presenti in fondo alle righe:

```
# cat veicoli | tr -d '\r' | tr -d '\n'
```

Come vedi, si fanno due "cicli" di lavorazione, perché "tr" cancella un tipo di carattere per volta. In output vedremo una sola grossa riga.

9.1.4 Trasformare un file in singole parole

Parti dal file "veicoli", tanto per usare del materiale. Vogliamo in output un file che abbia soltanto una parola per riga.

Come prima cosa, si potrebbero sostiture tutti gli spazi e i tab con il carattere di andata a capo UNIX \n:

```
# cat veicoli | tr '[ \t]' '\n'
```

Ma il risultato contiene troppe linee vuole. Perchè? Il comando corretto è:

```
# cat veicoli | tr -s '[ \t]' '\n'
```

L'opzione "-s" fa si che più spazi attaccati vengano trattati come un unico spazio.

9.1.5 Ora tocca a te

- Scrivi un testo su più righe dove compare spesso una parola, ad esempio "cane".
- Crea un comando per trasformalo in un file dove ogni parola è collocata su una riga a se stante.
- Fai il grep delle sole righe che contengo la parola scelta.
- Effettua il conteggio di quante volte compare la parola.

Comandi occorrenti: tr, grep, wc.

9.2 Sostituzione di più caratteri (sed)

"sed" è un potente comando che analizza il suo input, effettua delle sostituzioni in base a filtri e produce un suo output.

9.2.1 Sed: uso elementare

Il comando "sed" ha una sintassi del tipo:

sed "s/cosa/con/g"

"s" significa "sostituisci"; "g" significa "globalmente".
Esempi:

cat veicoli | sed "s/cbr-/CBR /g"

9.2.2 Sed: uso avanzato

Nel fare la sostituzione, sed può usare parte delle stringhe che ha letto, se racchiuse tra \(.... \), ponendole dove avete messo il segnaposto "&".
Osservate questi criptici comandi e provate a vedere cosa fanno:

- cat veicoli | sed "s/\(.\)/(&)/g"
- cat veicoli | sed "s/\([0-9]\)/00&/g"
- cat veicoli | sed "s/\(.\)$/& -fine/g"
- cat veicoli | sed "s/\(+\)$/-/g"

Prova anche questi, su qualche file:

- sed "s/aab/ab/g"
- sed "s/pier\(.\)/Pier&/g"
- sed "s/mi chiamo \(.+\)/si chiama &/g"

Capitolo 10

L'algoritmo MD5

10.1 Definizione semplificata di funzione hash

Per funzione hash si intende una funzione "non invertibile" (*one-way* function), ma che possa essere considerata "quasi iniettiva".

In sostanza, si richiede che se x_1 ed $x2$ sono diversi, allora, con grande probabilità, anche hash(x_1) sia diverso da hash(x_2).

Col termine "grande probabilità" si intende che i casi in cui il valore hash di due oggetti diversi coincidano sia un fatto raro, o comunque trascurabile.

10.2 Message digest (riassunto di un messaggio)

L'algoritmo hash denominato MD5 (message digest number 5) è progettato per prendere in input un messaggio grande quanto si vuole e crearne un "riassunto" di esattamente 16 bytes.

In Linux è disponibile a questo scopo il comando "md5sum". Sintassi:

```
# md5sum file1 file2 ....
```

Esempio:

Supponiamo di voler calcolare l'hash del file *01-intro.txt*. Ecco cosa ottengo al momento in cui scrivo:

```
# md5sum 01-intro.txt
1761d89790dd5b8ade1054809837c821 01-intro.txt
```

Come vedi, l'output del programma consiste in una stringa di 32 caratteri esadecimali, formanti appunti 16 bytes, con in più il nome del file processato.

è possibile processare più files, usando l'asterisco. Esempio:

- # md5sum *.exe (processa tutti i .exe)
- # md5sum * (processa tutti i file in questa dir)
- # md5sum /bin/* (processa tutti i file in /bin)

Io, per esempio, ho processato tutti gli eseguibili contenuti nella cartella c:\windows di Windows XP e ho trovato questo:

```
35b2560b60d2649ead8b9d56cb3c1121  c:/windows/IsUn0410.exe
515e4684008e955de0c81e6a7aea1c2a  c:/windows/IsUninst.exe
96ff03f5e325328c2b34112ee0e25705  c:/windows/NDNuninstall450.exe
205f4b55e601fd6f7be341922615 87a4  c:/windows/NSUninst.exe
de605de82c02fa006975336cf4f71e74  c:/windows/RSETPATH.exe
b9917fc4c836776765e311fff84dd534  c:/windows/Setup1.exe
536cc99602811f8c2597fabd7979610b  c:/windows/Unnero.exe
1597bc081cd26a36d727887279429c7a  c:/windows/explorer.exe
2acc867d1df9b41dc39d38ab3c24cf7b  c:/windows/hh.exe
28766412898912b9452e384dae205a9c  c:/windows/iun505.exe
bcbbce4c2209491af29dce3ee58a08a9  c:/windows/psuninst2.exe
b27a87f284cf79b119f8dcc43a9df9e1  c:/windows/regedit.exe
beda4634f0669332e25f526c870e380f  c:/windows/setdebug.exe
f36a271706edd23c94956afb56981184  c:/windows/twunk16.exe
435963f03a8d6d022a258b63176dc612  c:/windows/twunk32.exe
9ca884a033a9013ef939ff6554586200  c:/windows/uninst.exe
84b4f61f59a421bd85d97b35d194b42b  c:/windows/unvise32.exe
23a458e8eb269a71a29ada0cb3e22e65  c:/windows/unvise32qt.exe
12e4c95eb860031602b8ecc444350e35  c:/windows/winhelp.exe
9f77595d57e5980765cd5cd037373490  c:/windows/winhlp32.exe
```

10.3 A cosa può servire?

1. Caso 1: Beh, te lo puoi immaginare! Se ora un virus infettasse uno dei miei files eseguibili contenuti in c:\windows, rifacendo il check con *md5sum*, l'hash del file modificato sarebbe ora diverso, ed io potrei scoprirlo.
2. Caso 2: Nella trasmissione di un grosso file X su internet, spesso si aggiunge ad esso anche il file X.md5, con dentro la segnatura MD5 di X. Il destinatario può scoprire se il download ha avuto qualche problema, semplicemente ricalcolandolo e confrontandolo con quello che l'autore ha memorizzato in X.md5.

10.4 Lavorare in pipe con md5sum

md5sum può lavorare anche senza avere direttamente il file, ma leggendolo dallo stdin. Esempi:

```
# echo "ciao, mondo" | md5sum
```

Qui ha codificato la stringa "ciao, mondo".

```
# date | md5sum
```

Qui ha codificato la data corrente.

10.5 Ora tocca a te ...

1. Cerca un grosso file di testo o un documento in C: Calcola il suo hash. Modifica il file anche di un solo carattere, con Word ad esempio. Ricalcola il nuovo hash. Sono diversi?
2. Può succedere che MD5 "non si accorga" di una modifica? Perché?
3. md5sum processa solo files, le directory le salta.
4. E se volessi processare *tutto* l'hardisk?

Capitolo 11

Archivi compressi

Il comando tar (*tape archiver*) prende come input una serie di files o directories e crea un unico archivio, in cui i files sono messi uno di seguito all'altro.

La dimensione finale è praticamente pari alla somma delle dimensioni. Quindi, lo scopo di tar NON è la compressione dei dati.

11.1 Creare un archivio

Sintassi:

```
# tar -cf NOME-DEL-ARCHIVIO FILE1 FILE2 DIR1 FILE3 ...
```

Qui la "c" dei "-cf" significa "create".

In genere, per distinguerli, si fa in modo che il nome del file archivio finisca con .tar.

Esempi:

1. # tar -cf a.tar /bin /etc
2. # tar -cf b.tar /bin/*
3. # tar -cf c.tar /bin

Che differenza c'è tra il caso (2) e il caso (3)?

11.2 Ispezionare un archivio

```
# tar -tf a.tar
```

Si ottiene la lista dei file contenuti. Il -t significa "test"

11.3 Estrarre da un archivio

Per estrarre nella directory corrente:

```
# tar -xf a.tar
```

Per estrarre nella directory /home/michele/prova

```
# tar -C /home/michele/prova -xf a.tar
```

Qui -x signifca "extract".

Capitolo 12

Compressione dei files

Si usa il comando *gzip*.

gzip comprime solo files e non directory. Ecco perché per comprimere una intera directory, prima la archiviamo in un singolo file (con TAR) e poi comprimiamo il risultato ottenuto.

12.1 Uso di gzip

Sia «x» un file.

```
# gzip x
```

Ora nella directory avremo sia "x" che "x.gz", la versione compressa. L'estensione .gz è aggiunta automaticamente dal programma.

12.2 Comprimere una intera directory

```
# tar -cf tutta.tar /usr
# gzip tutta.tar
```

Cosa abbiamo ora nella directory? L'archivio non compresso tutta.tar e il nuovo archivio compresso: tutta.tar.gz.

Spesso, per comodità, questo tipo di archivio viene rinominato:

```
# mv tutta.tar.gz tutta.tgz
```

in modo che l'estensione "tgz" stia per "tar.gz".

è questa la forma più comune con cui si distribuiscono i programmi in Internet.

12.3 Lavorare con archivi compressi TGZ

Per poter listare il contenuto del file x.tgz dovremmo prima decomprimerlo; lo stesso anche per estrarne il contenuto. In sostanza, dovremmo fare il processo inverso:

creazione TGZ : archiviazione, compressione apertura TGZ : decompressione, dearchiviazione

Per fortuna, il comando "tar" ha l'opzione "-z" che gli permette di decomprimere "al volo".

Quindi:

- # tar -ztf tutta.tgz (lista)
- # tar -zxf tutta.tgz (estrazione)

12.3.1 Altri compressori

Ve ne sono tantissimi e tutti usano variante dello stesso algoritmo LZH. Il più antico di tutti è "compress". Il più potente è *bzip2*. UPX è molto potente ma comprime solo gli eseguibili.

Cercali su internet.

12.4 Archiviare e comprimere in pipe

è possibile usare i comandi tar e gzip in pipe, passandosi cioè il materiale da comprimere lungo il canale di input-output. Esempi:

Es1)

Qui comprimo una stringa e pongo il risultato in un file:

```
# echo "ciao, mondo" | gzip -c > ciao.gz
```

Es 2)

Qui suppondo di avere il file x.tgz (archivio compresso).

Potrei listarlo semplicemente così:

```
# tar -ztf x.tgz
```

Ma anche così:

```
# cat x.tgz | gzip -d | tar -tf-
```

Questo secondo modo è *istruttivo*, perché fa vedere i tre cicli di lavorazione a cui è sottoposto x.tgz: cat lo apre e lo stampa; gzip lo decomprime, tar lo lista.

12.5 Ora tocca a te

Crea un directory nuova nella tua area di lavoro. Copiaci dentro altri files o directories. Produci un archivio TGZ di questa directory.

Capitolo 13

Criptografia

13.1 Importanza della criptografia

Nel passato, quando gli scambi di informazione procedevano mediante documenti cartacei, intercettare un messaggio era questione di spie e di intrighi internazionali che non toccavano le grandi masse, ma piuttosto soltanto i governi.

Ai tempi nostri, dove la maggior parte delle informazioni vengono scambiate con mezzi elettronici, è fin troppo facile intercettare o alterare i nostri dati privati, anche da parte di Organizzazioni con mezzi economici non eccezionali.

Se un tempo la questione della sicurezza dei dati era materia che riguardava soprattutto gli Stati, ora riguarda il singolo. Ognuno di noi è a rischio.

Purtroppo però, la criptografia non solo non è incoraggiata dai governi ma, addirittura, da qualche parte è ancora considerata illegale.

E si può capire perché: la criptografia potrebbe essere usata dalle Organizzazioni Criminali per proteggere i loro traffici.

Infatti: dato che i codici a cifratura asimmetrica sono considerati praticamente inviolabili, neanche la Magistratura potrebbe aprire il messaggio, senza possedere la chiave.

13.2 Meccanismo di funzionamento

Questo tipo di cifratura si definisce "a chiave asimmetrica" o anche "a chiave pubblica". Si basa su due chiavi: una detta "pubblica" (P) , e serve per cifrare; l'altra è detta "segreta" (S), e serve per decifrare. In simboli:

$$S*P = P*S = I$$

e cioè le due chiavi sono, in un certo senso, una l'inversa dell'altra.

Chi vuole scrivermi, dovrà cifrare con la mia chiave P; io decifrerò con la mia chiave S, nota solo a me.

Una comunità che voglia comunicare in questo modo, deve depositare le proprie chiavi pubbliche presso una Autorità di Certificazione.

Se voglio, perciò, scrivere a Paolo devo procurami la chiave pubblica di Paolo e criptare con questa il messaggio.

In passato invece si usava soltanto la "cifratura simmetrica": una sola chiave serviva per cifrare e decifrare. è ovvio quindi che doveva essere spedita al destinatario con qualche altro mezzo più sicuro (ad esempio, a mano).

13.3 gpg: parte I Cifratura simmetrica

In questo corso non useremo "pgp" (Pretty Good Privacy, di P. Zimmerman), perché non è software libero. Useremo invece "gpg" (GNU Privacy Guard), che ha le stesse funzionalità di PGP, senza i problemi di brevetti ad esso collegati.

Notare l'inversione delle lettere tra gpg e pgp. Anche se dovessi usare il termine PGP mi riferirò sempre a gpg.

gpg si può usare sia con la cifratura tradizionale (simmetrica) che con quella moderna (asimmetrica)

13.3.1 Generare le mie chiavi S e P

Batti:

```
# gpg --gen-key
```

e segui le istruzioni. Ti verrà richiesta una passphrase: TIENILA A MENTE! Per questi esempi, sceglila corta, di un paio di parole.

Per avere la lista delle tue chiavi, batti:

```
# gpg --list-keys
```

13.3.2 Cifratura tradizionale (simmetrica)

Supponiamo di avere un file xyz e di volerlo cifrare, onde sia leggibile solo da noi. xyz potrebbe ad esempio contenere lettere che compromettono, o la nostra vera pagella :-)

Il comando:

```
# gpg -c xyz
```

produrra il file xyz.gpg, mantenendo l'originale, che possiamo anche cancellare.

Solo noi potremo decifrare il file, perché solo noi sappiamo la passphrase e abbiamo la chiave segreta, lo faremo col comando:

```
# gpg -d xyz.gpg > xyz
```

La cifratura tradizionale è quindi utile quando l'obbiettivo non è "lo scambio" di informazioni sicure, ma "l'archiviazione sicura", sul proprio computer, di materiale che vogliamo tenere protetto da occhi indiscreti.

13.4 gpg: parte II cifratura asimmetrica

Per usare questo tipo di cifratura, i due protagonisti *devono* scambiarsi le loro chiavi pubbiche P.

13.4.1 Pubblicazione della chiave

Supponiamo che io, Michele, voglia comunicare con Karl. Ho bisogno della chiave pubblica di Karl.

Karl dev quindi "exportare" la sua chiave in file "karl.asc" e spedirmela per posta elettronica.

Lo stesso dovrò fare io, se voglio che lui mi risponda in criptato.

13.4.2 Export delle chiavi

Ecco come Karl creerà il file paolo.asc

```
# gpg --export -a > karl.asc
```

Analogamente, io imposterò:

```
# gpg --export -a > michele.asc
```

Io spedirò a Karl il file michele.asc e lui spedirà a me il file karl.asc

13.4.3 Osservazione

E se un intruso intercetta *proprio* queste due email? Poco male, per due motivi:

1. esse contengono il codice MD5 (vedi lezione su MD5). Se qualcuno tenta di modificarle, gpg se ne accorge, perché l'hash è diversa

2. l'intruso potrebbe usare le chiavi per spacciarsi per Michele o Karl, questo è vero.

Ma sono solo chiavi pubbliche! Con esse si può si spedire, ma non si possono aprire i messaggi diretti a Michele o Karl!

13.4.4 Import delle chiavi

Dopo che io e Karl ci siamo spediti i due file, dobbiamo inserire le chiavi "amiche" nel nostro data-base:

(Michele)# gpg --import karl.asc
(Karl) # gpg --import michele.asc

A questo punto, battendo

 # gpg --list-keys

vedremo nella lista le chiavi nostre e quelle del nostro amico.

13.4.5 Ok, criptiamo ora ...

Io ho nel mio data-base la chiave pubblica di Karl-Heinz Zimmer, un mio amico tedesco. Quando batto "gpg --list-keys", tra le tante righe, mi appare questa:

pub 1024D/BBE08037 2000-09-08 Karl-Heinz Zimmer <khz@snafu.de>

è di questa chiave che ho bisogno, se voglio scrivere a Karl. Come farò a dire a gpg che voglio usare *proprio* questa?

Semplice:

```
# gpg -e -r khz@snafu.de mio_file
```

L'opzione "-e" signifa criptare (encrypt) mentre "-r khz@snafu.de" significa che il "recipiente" è khz@snafu.de

Verrà prodotto il file mio_file.gpg che potrò spedire a Karl via email.

13.4.5.1 Ma, posso decifrare i messaggi per Karl?

Non ci crederai, ma nessun altro che Karl, nemmeno tu, potra più aprire il file mio_file.gpg.

Quindi, se cancelli la lettera mio_file, non potrai più effettuare la decifrazione mio_file.gpg -> mio_file.

Solo Karl, infatti ha la chiave segreta necessaria a farlo.

Capitolo 14

Connettersi a sistemi remoti

Linux dà il megli sé nelle attività di rete. In Linux è facilissimo connettersi ad un sistema remoto Linux (ad esempio la propria università) e lavorarci «in remoto» senza notare nessuna differenza.

14.1 Sessioni interattive remote

Per connettersi «in interattivo» ad un sistema linux remoto c'è il comando ssh (*secure shell*), che ha la sintassi ssh user@host. Esempio, il comando

```
# ssh zuckerberg@facebook.com
```

aprirà una sessione «shell» sull'host remoto facebook.com

Il sistema remoto chiederà una password, e tutto il traffico verrà criptografato.

14.2 Eseguire comandi su un sistema remoto

Sintassi:

```
# ssh [-X] user@host command
```

Esempio:

```
#ssh bill@microsoft.com du -s /home/bill
```

il sistema si connette a «microsoft.com» come user «bill» e calcola lo spazio occupato dalla cartella di Bill: /home/bill.

L'opzion -X, invece, attiva il cosidetto «X forwarding»: in sostanza, il comando remoto, invece che creare una finestra sul PC remoto, la crerà sul nostro monitor![1]

14.3 Copiare files e cartelle da un computer all'altro

Per copiare materiale da un computer ad un altro, c'è il comando scp (*secure copy*). Questo comando è la versione di rete del compando UNIX «cp».

La sua sintassi è un po' più complessa, ma è più o meno del tipo:

```
# scp [-r] user1@host1:/a/b/c user2@host2:/e/f/g
```

L'opzione «-r», se presente, significa «ricorsivo», per cui copia l'intera cartella e tutte le sue sottocartelle. Questo vuol dire che si può copiare un intero hard-disk da un continente all'altro, senza alcun problema.

Esempi:

```
# scp -r /home/michele/natale zuckerberg@facebook.com:regali
```

copia tutta la cartella «natale» nella cartella «regali» del

```
# scp -r zuckerberg@facebook.com:regali bezos@amazon.com:regali-restituiti
```

copia tutta la cartella «regali» sul computer destinazione, ma nella cartella «regali-restituiti»

[1] Ma l'utente remoto ci deve abilitare con «xhost +»

Capitolo 15

Programmazione: gli scripts di shell

Esattamente come coi .bat del DOS, anche in UNIX è possibile raccogliere un insieme collaudato di comandi dentro un file, e chiamarlo "programma", o meglio "script".

Uno script è, quindi, un programma interpretato dalla Shell di UNIX, che è l'interprete con cui si dialoga quando si è connessi.

La shell è essa stessa un programma: per la precisione si tratta del programma /bin/sh (*bash*)

15.1 Creazione dello script

è molto facile. Basta aprire un file con l'editor e fare in modo che la prima linea sia questa

#!/bin/sh

Le righe seguenti conterranno i vostri comandi, ma anche dei commenti, preceduti dal carattere speciale #

15.2 Rendere eseguibile uno script

Affinchè uno script sia "lanciabile", deve avere i permessi di esecuzione. Supponiamo che lo script si chiami "stat". Per dargli il permesso devo battere:

```
# chmod +x stat
```

15.3 Lanciare uno script

Lo script, appena creato, entra a far parte dei comandi di UNIX. Basta entrare nella directory dove si trova lo script e battere il nome dello script. Ad esempio «stat», premettendo la directory corrente «./»:

```
# ./stat
```

(lanciamo così lo script "stat", nella directory ccorrente).

15.4 Argomenti di uno script

Come per gli altri comandi UNIX, anche i nostri script possono avere argomenti. All'interno dello script, gli argomenti verrano visti come variabili: $1, $2, $3 ....

15.4.1 Esempio: lo script "somma", con due argomenti.

Apri il file "somma" e mettici dentro questo:

```
#!/bin/sh
expr $1 + $2
```

Se ora fai:

```
# chmod +x somma (devi farlo una sola volta)
# ./somma 2 3
avri come output 5
```

15.5 Altri esempi

Invece di descrivere nel dettaglio tutti i costrutti della shell, ho creato una directory demo/, con una serie di esempi.

La shell, come linguaggio di programmazione, ammette tutti i costrutti più importanti (for, while, ...), ma non ammette vettori o array.

Gli script di shell vengono usati essenzialmente per l'amministrazione dei sistemi UNIX, per installare pacchetti e per automatizzare la gestione delle varie funzionalità del sistema stesso.

La programmazione della shell è descritta in molti buoni libri presenti in commercio. Linux segue esattamente lo standard UNIX. Per cui, non devi cercare Linux, ma UNIX.

Capitolo 16

Esempi di *scripts*

16.1 Condizionali

```
#!/bin/sh
# demo if
echo "Strutture condizionali"
read -p "Dammi un numero compreso tra 0 e 9: " n
if [ $n -lt 5 ] ; then
echo "mi hai dato un numero minore di 5."
else
echo "mi hai dato un numero maggiore o uguale a 5."
fi
echo "demo finito."
```

16.2 Scelte multiple

```
#!/bin/sh
# demo "multi" , scelte multiple
cat <<END
Ti piace Linux?
(puoi rispondere: molto, poco, per niente, affatto, pochino)
END
read -p "Ti piace Linux? " quanto
case "$quanto" in
molto) echo "me ne compiaccio.";;
poco) echo "si, qualche volta non mi amano";;
per*niente|affatto) echo "ahia, mi spiace";;
pochino) echo "non ti sbilanci, eh? ";;
*)
echo "questa risposta non la sapevo." ;;
esac
```

16.3 Cicli finiti ed infiniti

```
#!/bin/sh # demo cicli
echo "Conteggi e Tabelle:"
# ciclo for
for i in 1 2 3 4 5 ; do
echo "$i $(expr $i \* $i)"
done
cat << END
Per interrompere un programma, devi battere CONTROL-C END
read -p "batti return per iniziare ..." return
# un ciclo infinito
while [ 1 ] ; do
echo "Sono passati $(date +%s) secondi dal 1 Gennaio 1970 ..."
sleep 1
done
```

16.4 Input string e sostituzioni

```
#!/bin/sh
# programma dimostrativo
# sugli script di shell in Linux
# scritto da M.Andreoli, (C)2002
#----------------------
# input di stringhe
#----------------------
# La variabili "nome" conterra'
# il nome dello studente
clear
read -p "Come ti chiami? " nome
echo "Piacere, $nome. Io mi chiamo Linux."
echo "Oggi e': $(date)"
#----------------------
# sostituzioni e sort
#----------------------
read -p "Demo sostituzioni. Batti INVIO " return
clear
NOME1=$( echo $nome | tr [a-z] [A-Z] )
NOME2=$( echo $nome | sed "s/\(.\)/(&)/g" )
NOME3=$( echo $nome | sed "s/\(.\)/&_/g" )
cat <<END
In maiuscolo, tu ti chiami $NOME1!
Che e' come dire $NOME2, oppure $NOME3,
non e' cosi'?
END
```

16.5 Tempo e date

```
#!/bin/sh
# demo date
giorno=$(date +%d)
mancano=$(expr 31 - $giorno)
natale=$( date -d "12/25" +%A)
cat <<END
Oggi e' $(date +%d) del mese $(date +%m) ( $(date +%B)).
Mancano $mancano giorni alla fine del mese.
Natale, quest'anno (siamo nel $(date +%Y)), capita di $natale.
Il Natale dell'anno 2025 capitera' di $(date -d "12/25/2025" +%A),
se ti interessa.
Ehm, sono passati $(date +%s) secondi dal 1 gennaio del 1970,
che e' l'ora zero del sistema operativo Linux.
END
```

16.6 Calcoli semplici (expr)

```
#!/bin/sh
# programma dimostrativo
# sugli script di shell in Linux
# scritto da M.Andreoli, (C)2002
#---------------------
# Calcoli numerici
#---------------------
read -p "Dammi un numero: " x
read -p "Dammene un altro: " y
echo "La somma di $x e di $y fa, guarda caso, $(expr $x + $y)."
echo "La differenza di $x e di $y fa, guarda caso, $(expr $x - $y)."
echo "Il prodotto di $x e di $y fa, guarda caso, $(expr $x \* $y)."
```

16.7 Calcoli complessi (bc)

```
#!/bin/bc -l
/*
calcolatrice scientifica BC
*/
x=0.1;
y=2*x^2+1;
for (x=0.0; x<=6.28; x=x+0.4) {
print "sin(",x,")= ",s(x),"\n";
}
quit
```

16.8 Lavorare con funzioni

```
#!/bin/sh
# demo: funzioni
f()
{
echo "Mi hai passato questi argomenti: $*"
echo "Erano in tutto $# argomenti."
echo "Il secondo era [$2]. Mi e' piaciuto."
echo
echo "Sono buoni i tuoi argomenti! "
}
modulo()
{
expr $1 % $2
}
clear
f 1 ciao 4 -b
echo "25 mod 3 fa $(modulo 25 3). Cosi', tanto per dire."
echo "Ieri era [ $(date -d "yesterday") ]."
```