Appunti di Bash su Linux
ovvero:
come utilizzare Linux da terminale

mt

Versione:	1.00
Ultima Modifica :	8 febbraio 2009
Data di compilazione :	18 gennaio 2011

Vi veri veniversum vivus vici.

Con la forza della verità, in vita,
ho conquistato l'universo.

Christopher Marlowe
The Tragical History of Doctor Faustus

Preparati ad ascoltare cose
che tra gli uomini
mai sono state udite prima,
e di cui gli Elfi parlano di rado.

J.R.R. Tolkien
Racconti Ritrovati

Abbiamo visto che la programmazione
è un'arte, perché richiede conoscenza,
applicazione, abilità e ingegno,
ma soprattutto per la bellezza
degli oggetti che produce.

Donald E. Knuth

Indice

1 Introduzione **1**
- 1.1 Software Open-Source e formati aperti . . . 3
- 1.2 Sicurezza su Linux . . . 6
- 1.3 Convenzioni usate nel testo . . . 7

2 Bash **8**
- 2.1 Comandi più comuni . . . 8
- 2.2 Operazioni su file . . . 10
- 2.3 Bash Scripts . . . 11
- 2.4 Simboli e caratteri speciali . . . 12
- 2.5 Altre cose importanti sulla shell . . . 22

3 File, directory e permessi **23**
- 3.1 I comandi ls e stat . . . 23
- 3.2 Permessi . . . 24
- 3.3 Tipi di file . . . 26
- 3.4 Albero delle directory . . . 27
- 3.5 Filesystems . . . 28
- 3.6 Localizzazione di file e comandi . . . 30

4 Input ed output **32**
- 4.1 Redirezione dell'input/output e degli errori . . . 32
- 4.2 File speciali . . . 34
- 4.3 Manipolazione input/output da file . . . 35
- 4.4 Sed . . . 38
- 4.5 Awk . . . 38

5 Variabili **39**
- 5.1 eval . . . 39
- 5.2 Argomenti ed opzioni degli script . . . 39
- 5.3 Calcoli matematici . . . 39
- 5.4 Alcune variabili di configurazione . . . 39
- 5.5 Variabili "speciali" di Bash . . . 40
- 5.6 Array . . . 41

6 Condizioni, confronti e test **42**
- 6.1 If - then - else/elif - fi . . . 42
- 6.2 Confronti numerici . . . 42
- 6.3 Test su file . . . 42
- 6.4 Case . . . 42
- 6.5 Select . . . 43

7 Cicli **45**
- 7.1 For - do - done . . . 45
- 7.2 While - do - done . . . 45

8 Alias e funzioni **47**

9 Processi **48**

10 File di configurazione: **51**
10.1 Configurazione globale 51
10.2 Configurazione personalizzata del singolo utente 52

11 Trucchi del mestiere **57**
11.1 Operazioni con file 57
11.2 Caratteri speciali 58
11.3 Web e servizi di rete 58

12 Alcuni esempi di script in Bash **60**
12.1 Rinominare una serie di file 60
12.2 Flip 60
12.3 Fattoriale 61
12.4 Calcolo del pi-greco con un metodo di tipo Montecarlo 61
12.5 Portscanner 64
12.6 Network-Traffic 64

13 Alcuni esempi di script in Perl **65**
13.1 Ricerca di file doppi 65

14 Script in altri linguaggi **68**
14.1 Tcl/Tk 68
14.2 Expect 68

15 Linguaggi compilati **69**

16 Terminali **71**

17 Amministrazione e manutenzione **72**
17.1 Utenti, su, sudo 72
17.2 Caratteristiche hardware del sistema 72
17.3 Data e ora del sistema 72
17.4 Avvio 73
17.5 Installazione di nuovo software 73
17.6 Locale 76
17.7 Tastiera italiana 77
17.8 Accenti 77
17.9 Mouse 77
17.10 Demoni 77
17.11 Formattare dischi/partizioni 77
17.12 Backup 78
17.13 Creazione di CD/DVD-Rom 78
17.14 Altre periferiche 80

18 Internet **81**
18.1 Configurazione internet . . . 81
18.2 Web . . . 82
18.3 Mail . . . 82
18.4 News . . . 83
18.5 Servizi di rete . . . 83
18.6 Sicurezza . . . 83

19 Crittografia **85**

20 OpenPGP **85**

21 Filesystems criptati **85**

22 X11: la modalità grafica **86**
22.1 Avvio da console . . . 86
22.2 Configurazione per il singolo utente . . . 88
22.3 Altri comandi di X e pacchetti utili . . . 90
22.4 Fonts . . . 90
22.5 Multi-schermo . . . 90
22.6 Multimedia . . . 91
22.7 Menu grafici . . . 93

23 Produzione di documenti e conversioni tra formati **94**

A Con cosa apro questo documento? **95**

B Creazione di un Makefile **98**
B.1 Makefile per un tipico progetto in C . . . 98
B.2 Makefile per la compilazione di codice LaTeX . . . 98

C Creazione di una pagina di manuale **102**

D Il comando screen **104**

E L'editor di testi vim **105**
E.1 Perché sembra tutto così difficile? . . . 105
E.2 Comandi principali . . . 105

F Gnuplot **107**

G Creazione di database con PostgreSQL **110**

H mySQL **110**

I Controllo di versione di un software **111**
I.1 CVS . . . 111
I.2 Mercurial (Hg) . . . 111

J Toys **112**

K Riepilogo comandi visti (e non) **115**

L Glossario **123**

Riferimenti bibliografici e siti internet **128**
Manuali 128
Siti Internet 129

Indice analitico **131**

1 Introduzione

Questo piccolo manuale vuole essere uno strumento per imparare ad utilizzare nel modo più efficiente possibile la linea di comando di Linux. Cominciamo dunque facendo un po' di chiarezza sui nomi.

Innanzitutto "Linux" propriamente detto non è un vero sistema operativo, ma un *kernel*, [1] ossia, il cuore di un sistema operativo, che si occupa di interfacciare a basso livello, la macchina con le applicazioni che poi va ad usare l'utente. Nonostante questo, comunemente ci si riferisce a Linux come al sistema operativo formato dal kernel e da tutto il software ad esso annesso, anche se sarebbe più corretto chiamare il tutto "GNU/Linux". [2]

Il progetto GNU (acronimo che sta per GNU is Not Unix) comprende una vasta gamma di software (di cui fanno parte ad esempio gcc, emacs, gnuplot) che dovrebbe creare un giorno un vero e proprio sistema operativo GNU. Esiste anche un kernel GNU (il suo nome è HURD) che al momento è ancora del tutto sperimentale.

Il kernel Linux originario venne sviluppato da Linus Torvalds, allora studente di informatica all'università di Helsinki, nell'oramai lontano 1991. La leggenda narra che il professor Andrew S. Tanembaum [3] lo giudicò un lavoro non molto buono, soprattutto per la decisione di creare un kernel monolitico, anziché scegliere l'architettura a microkernel. Questo significa che il kernel Linux si occupa di ... e non demanda le operazioni ... come è il caso di altri kernel come ... [4] Questa scelta, sebbene in controtendenza con i kernel più moderni, lo rende però più veloce e forse anche più affidabile.

Linux fa parte della famiglia degli Unix, ma ha una caratteristica peculiare che ne ha fatto la fortuna: è "libero", ossia il suo codice sorgente viene distribuito con una licenza (in questo caso la GPL) che lo rende non solo utilizzabile liberamente, ma anche modificabile e ri-distribuibile da tutti. Questo ha fatto si, tra le altre cose, che si venissero a creare moltissime diverse distribuzioni, ognuna corredata da parecchio software (tra le più famose: Slackware, Debian, Suse, RedHat, Fedora, Mandriva, Gentoo, Ubuntu).

Uno dei vantaggi di usare Linux (ma è lo stesso con altri sistemi liberi, come FreeBSD) è quello di poter personalizzare la propria installazione, scegliendo la distribuzione più adatta, l'interfaccia grafica da usare, i programmi da installare e, se uno ne ha le capacità, andando perfino a modificare il codice sorgente, ottenendo così quello che gli serve (e magari condividendolo poi con gli altri, secondo lo spirito del software libero).

Certo è meglio procedere per piccoli passi, dunque prima di arrivare alla ricompilazione del kernel, (cosa non tanto difficile di per sé, quanto critica, visto che un piccolo bug a livello del kernel può creare il blocco del sistema se vi va bene, o molto

[1] parola inglese che significa nocciolo o nucleo

[2] provate infatti a vedere, quando avrete dimestichezza con i comandi, quanti pacchetti dipendono, direttamente o meno, dalla libreria GNU `"glibc"` ...

[3] creatore del sistema Minix, allora usato per scopi didattici e a cui lo stesso Linus si ispirò inizialmente.

[4] molti kernel commerciali, come quelli delle varie versioni di Windows successive alla NT sono in realtà ibridi, ossia una via di mezzo tra i due tipi descritti finora.

peggio . . .) è il caso di imparare ad utilizzare e a sfruttare appieno le potenzialità di questo sistema operativo.

E qui entra in gioco questo manuale. Ci si potrebbe chiedere: perché mai doversi imparare tutta una serie di comandi quando è possibile fare tutto (o almeno così si è portati a pensare all'inizio) da una comoda e intuitiva modalità grafica? Effettivamente un utente Windows che decide di passare (magari solo per provare) a Linux, abituato a fare tutto in modalità grafica, cercherà di continuare a fare altrettanto.

Inoltre bisogna ammettere che, fino a pochi anni fa Linux, a differenza di Windows, non era certo *user-friendly*, ed un utente inesperto poteva facilmente spaventarsi e decidere di tornare indietro. Oggi però alcune distribuzioni hanno fatto enormi passi in avanti in fatto di facilità di installazione e di intuitività nell'utilizzo al primo accesso.

Ma non appena un utente un po' smaliziato si avventura nei meandri del sistema operativo, si accorge dell'esistenza di una creatura molto potente e allo stesso tempo pericolosa: il terminale dei comandi. Sempre facendo (con le dovute cautele) i paragoni col mondo Windows, sarebbe l'analogo del prompt di MS-DOS. Bene: questo basta a far capire come mai il tipico utente proveniente dal mondo Windows disdegna la linea di comando. Non è solo dovuto al fatto che l'interfaccia grafica di Windows sia user-friendly e dunque l'utente medio non trova la necessità di spingersi più in là. Il vero problema è, invece, che il prompt dei comandi di MS-DOS, pur migliorato nelle ultime versioni di Windows, [5] è praticamente inutilizzabile per scopi seri. Su Linux, come vedremo, la situazione è molto diversa.

All'inizio naturalmente è necessario un certo periodo di tempo per familiarizzare con i nomi dei comandi, la sintassi, etc. Ma se si ha la pazienza di cominciare a leggere qualche manuale, e di fare un po' di utili esercizi, e nonostante qualche inevitabile errore (un consiglio: createvi un account apposito per fare le vostre prove, in modo da annullare la possibilità di fare disastri; e non sognatevi neppure di fare le vostre prime prove dall'account di amministratore del sistema) si arriverà al punto in cui ci si accorge che da terminale si possono fare moltissime cose, e (cosa più importante) molto velocemente!

Per fare un esempio pratico, le tipiche operazioni che vale la pena di fare da linea di comando sono quelle che si ripetono uguali su molti file (come cambiarne una parte del nome o l'estensione, spostarli da una directory ad un'altra in base al nome, copiarli o meno in un'archivio in base alla data di ultima modifica). E vale la pena di perdere inizialmente un po' di tempo per trovare ed adattare alle proprie esigenze il comando corretto da dare, poiché la volta successiva in cui ne avremo bisogno lo faremo molto più in fretta; se poi ci capiterà di usarlo molte volte lo impareremo praticamente a memoria. Scopriremo inoltre che non è necessario impararsi nomi eccessivamente complicati visto che esistono gli *alias*, con i quali possiamo definire delle scorciatoie personalizzate a qualsiasi comando. Oppure impareremo a creare dei piccoli *script* (programmini il cui analogo, nel mondo DOS, sono i file *batch*, ossia quelli con estensione `".BAT"`) che possono iterare facilmente le operazioni desiderate.

Concludo questa introduzione con una nota sulla struttura di questo manuale.

[5] mi riferisco alle varie versioni fino almeno a Windows XP

Ho cercato, per quanto ho potuto, di introdurre nozioni, comandi ed esempi in un ordine abbastanza logico, in maniera che leggendo dall'inizio si possa riuscire a seguire il filo anche senza avere tante conoscenze di base.

Tuttavia a volte, questo non è stato del tutto possibile, ed è per questo che nel testo ci sono molti riferimenti a sezioni successive o precedenti. L'idea è che il manuale vada letto sezione dopo sezione, anche senza approfondimenti, fino al punto in cui il lettore si rende conto che non può fare a meno, per capire realmente, di mettere in pratica gli esempi proposti, integrandoli anche con altre letture, e cominciando a cercare ed utilizzare le nozioni sparse qua e là nel manuale; a quale punto del testo succederà questo dipenderà ovviamente dal suo livello di conoscenza iniziale.

1.1 Software Open-Source e formati aperti

Questa sezione, devia un poco dallo scopo principale della guida, ma serve ad evidenziare l'importanza del fare le scelte più opportune nell'installazione ed utilizzo del software.

Innanzitutto, il software, così come accade per i libri o la musica, è protetto da diritti d'autore. Questo si rende necessario per evitare che una qualsiasi persona "copi" un programma da un altro e lo usi senza il permesso dell'autore o lo ridistribuisca a suo nome.

Si potrebbe pensare che il software Open-Source non abbia bisogno di queste protezioni, in quanto "open" e dunque accessibile a tutti. In realtà proprio per il fatto di essere open, necessita di essere protetto, per impedire che qualcun altro se ne appropri e lo trasformari in un software proprietario.

Si rende così necessario che anche il software open abbia qualcuno che ne detenga i diritti, ma allo stesso tempo, per la sua natura, esso dovrebbe appartenere a tutti. Qui entra in gioco il concetto di "copyleft" [6], ossia la volontà da parte dell'autore di condividerlo con tutti e, allo stesso tempo, di impedire che qualcuno in futuro possa appropriasene o ripubblicarlo in forma "un po' meno libera".

Di solito si parla di questi software, protetti da licenze dette libere, indicandoli a loro volta come liberi (o *free*).[7] Ma attenzione al seguente concetto: non tutto il software Open-Source è necessariamente "free".

> L'*Open Source* è una metodologia di sviluppo; il *Software Libero* è un movimento di carattere sociale.

In modo più rigoroso, infatti, è possibile definire un software come "libero", se la sua licenza consente a chiunque di prendere il codice sorgente, di modificarlo per i suoi scopi e di ridistribuirlo liberamente; se la licenza non lo consente esplicitamente, il software anche se open, non è free. Da notare che software libero ma non protetto

[6] il primo ad usare questo termine sembra essere stato ..., tuttavia il concetto è stato poi reso celebre da Richard M. Stallman, fondatore della Free Software Foundation.

[7] spesso invece il termine free (o freeware) viene impiegato per indicare tutti i programmi scaricabili gratuitamente dalla rete; questo genera confusione, in quanto non tutti questi sono necessariamente "liberi", nel senso qui descritto del termine.

da opportune licenze di copyleft potrebbe, un giorno, essere ripubblicato sotto licenze meno libere ...

Esistono varie licenze, sotto le quali è possibile distribuire il codice sorgente di un software; le più note, sono la *GNU-GPL* (nelle sue varie versioni), la *X11* e quelle di tipo *BSD*. Le differenze principali tra queste riguardano la possibilità o meno di poter ridistribuire il software anche sotto licenze diverse, e di consentire ad altri di sfruttare il software a fini commerciali. In questo senso la GPL è abbastanza restrittiva, e non permette al codice, protetto da essa, di essere riutilizzato in software con licenze diverse; questo per poter proteggere il lavoro dei programmatori della comunità Open-Source e garantire che il software da loro sviluppato rimanga per sempre "libero".

Ci sono dei settori nei quali è decisamente importante che il software sia "open". Ad esempio per la crittografia, e la ragione è ottimamente spiegata da Phil Zimmermann, l'autore di `"pgp"`, nel suo lavoro [24]: sostanzialmente come è possibile conoscere la bontà o meno di un programma di crittografia se non possiamo verificare gli algoritmi che esso usa? In questo caso l'assicurazione della sua bontà da parte del suo produttore non può essere sufficiente, specialmente per dati veramente importanti (quali ad esempio segreti industriali, militari o di stato).

Un altro caso è il software usato nella Pubblica Amministrazione. I motivi principali sono più o meno gli stessi per cui anche l'utente comune dovrebbe preferire l'open-source, ossia:

1. è gratis (e anche nel caso della pubblica amministrazione, pur se indirettamente, siamo sempre noi a risparmiare).

2. la sicurezza non è certificata dal produttore (che ha evidentemente i propri interessi) ma dalla comunità che lo sviluppa e lo usa (e che non ha altri interessi se non migliorarlo).

3. se necessario e se ne si hanno le capacità, può essere conveniente investire tempo e risorse per migliorare il software o adattarlo alle proprie esigenze. Nel caso di software proprietario, invece, gli aggiornamenti (anche quelli che riguardano la sicurezza) dipendono dalle scelte del produttore.

4. eventuali bachi, oltre ad essere scoperti più facilmente, vengono corretti in tempi ragionevoli, che dipendono sostanzialmente da quanta gente (e quanto brava) si dedica allo sviluppo di quel software, ma di solito, almeno nel caso dei programmi più diffusi, sono molto brevi. Nel caso del software closed-source solo il produttore può correggere l'errore, e può decidere di farlo come e quando gli pare.

5. è molto più facile scovare l'eventuale presenza di "backdoor". Chi mai si sognerebbe di mettere del codice maligno all'interno di software Open-Source? Il problema, casomai, è assicurarsi che il software precompilato (che è quello che si scarica di solito) provenga effettivamente dal codice da cui si afferma che derivi (ma possiamo anche decidere di prendere tutti i sorgenti e compilarceli da soli). Nel caso del software closed, non potendo il sorgente essere verificato da qualcuno esterno all'azienda, il dubbio sulla presenza di

codice malevolo (nei nostri confronti) nascosto al suo interno, rimane sempre (qualche informazione può ancora essere ottenuta mediante operazioni di *reverse engineering*, comunque normalmente vietate dalle licenze).

6. con la diffusione del sorgente di un programma si spinge quella parte della gente che ne è interessata a studiarne il codice e si contribuisce perciò alla diffusione della conoscenza. Questo punto potrebbe sembrare banale, ma si ricordi che molti programmi del mondo Unix, ancora oggi molto usati, sono stati creati inizialmente nei dipartimenti di informatica di varie università per scopi prevalentemente didattici. Il codice doveva essere aperto proprio perché il suo scopo era quello di "insegnare". Lo stesso concetto di software libero deriva da questa tradizione di condivisione e diffusione del sapere.

L'ampia diffusione di validi prodotti nel mondo del software open-source, inoltre, comporta indirettamente dei benefici anche a chi non li usa, infatti una sana concorrenza spinge i produttori di software proprietario a migliorare i propri programmi per offrire qualcosa in più dei concorrenti o comunque, in genere, ad abbassare i prezzi.[8] Al contrario, in una situazione in cui sono poche case produttrici a farla da padrona, l'evoluzione del software è in genere più lenta e comunque decisa più da politiche aziendali che dalle reali esigenze degli utenti.

In maniera analoga, risulta conveniente per la Pubblica Amministrazione, ma non solo, la scelta di usare esclusivamente formati aperti (ossia di cui si conoscono le specifiche, in modo che sia possibile creare un proprio software per leggerli e/o modificarli). Si possono evidenziare le seguenti motivazioni:

1. non si obbligano gli altri utenti a procurarsi software proprietario (e costoso) per leggere i nostri documenti.

2. portabilità (almeno teorica) su un maggior numero di sistemi.

3. di solito un formato aperto, non dovendo essere protetto dalla lettura, è molto più snello del suo analogo chiuso (vi siete mai chiesti che cosa mai ci sia dentro un file `".doc"` che occupa una considerevole quantità di spazio, ma in cui in realtà avete scritto solo poche righe?).

4. controllo dei dati realmente contenuti nel documento; col formato open, noi, così come tutti gli altri, possiamo andare a controllare tutte le informazioni realmente (fisicamente) presenti nel documento. Nel caso di formato di cui non conosciamo tutte le specifiche, possono capitare delle sorprese; ad esempio il formato `".doc"`, consente di salvare al suo interno le modifiche fatte e poi annullate. Noi possiamo anche non accorgercene ma qualcun altro, con gli strumenti adatti a disposizione, può risalire alle vecchie versioni del documento e a dati che magari non volevamo rendere pubblici.

5. l'accessibilità nel tempo; infatti se un software proprietario non dovesse più essere mantenuto, potrebbe costringerci a rimanere legati a vecchie piattaforme per leggere i documenti creati da quel programma, poiché magari capita che

[8] storicamente questo è accaduto per esempio per Internet Explorer, grazie alla concorrenza di Firefox

essi non siano leggibili con software più recenti o traducibili in altri formati. Con i formati aperti, anche se un programma non venisse più sviluppato, molto probabilmente, ricompilandolo opportunamente funzionerebbe anche su sistemi operativi diversi ed aggiornati, oppure qualcuno potrebbe sempre riprendere in mano il codice ed aggiornarlo o creare dei convertitori verso altri formati.

Tipici esempi di documenti in formato aperto sono i file Html, PDF, Postscript, Dvi, Rtf, ODF; un tipico esempio di formato chiuso è invece il famigerato `".doc"` di Word (l'editor di testi del pacchetto Microsoft-Office).

Per l'utente comune, spesso si presenta la scelta tra usare un programma libero ed uno proprietario sostanzialmente simili tra loro, dei quali però probabilmente andrebbe a sfruttare meno del 10% delle potenzialità. A volte conviene chiederci: di quali funzionalità ho veramente bisogno? Ed il software free me le offre?

Infine, un'ultima osservazione: non vi sembra assurdo scaricare illegalmente programmi proprietari quando esiste una valida alternativa free?

Anche per quanto riguarda i manuali e la documentazione che accompagna il software, possiamo scegliere tra varie licenze libere come le Creative-Commons e la GFDL (quella di Wikipedia, per intenderci).

1.2 Sicurezza su Linux

Le minacce per un sistema sono essenzialmente le seguenti: virus che infettano i programmi, *trojan* (che in genere aprono delle backdoor) bachi (*bugs*) nel software installato ed infine *worm*, che si diffondono via internet.

Non starò qui a spiegarne le varie differenze, ed in ogni caso una breve definizione viene data nel glossario (L).

Innanzitutto sfatiamo una leggenda: non è vero che su Linux non esistono virus. Ogni sistema operativo su cui è possibile installare dei programmi e compilare del codice, può avere dei virus. Tuttavia è vero che, in genere, virus su Linux non ce ne sono. Questo perché ... Per questo motivo, un antivirus su Linux è completamente inutile. Per motivi analoghi, i worm, molto difficilmente ...

I trojan ...

Discorso completamente diverso, invece, per quanto riguarda i bug. Il software open-source, come tutto il software, soffre di bug, ossia errori nel codice, che a volte consentono a dei malintenzionati di accedere senza permesso al sistema, o di acquisire permessi che non dovrebbero avere. Un tipico esempio di falla potenzialmente sfruttabile per fare danni è l'*exploit*. Esiste la concreta possibilità che qualcuno, scoperto un bug, che crea una falla nella sicurezza di un sistema, non lo segnali ma cominci a sfruttarlo per i suoi scopi personali. Del resto, così facendo, la vittima che abbia un minimo di esperienza, si accorgerà del problema e lo segnalerà.

In genere comunque i pericoli più immediati derivano dagli stessi utenti che hanno regolare accesso al sistema. Ad esempio attacchi ti tipo DOS (Denial of Service) intenzionali o meno, saranno sempre possibili, a meno di non limitare le risorse disponibili ai singoli utenti (naturalmente questo ha poco senso in un

sistema casalingo con un singolo utente, ma diventa fondamentale su un server multi-utente).

Le regole per mantenere un sistema sicuro sono dunque aggiornare frequentemente il software, non utilizzare programmi di dubbia provenienza e, soprattutto, sapere quello che si sta facendo!

Oltre a questo

configurare il *firewall* ed eliminare i servizi non indispensabili.

In sostanza, Linux, nella sua configurazione di default è probabilmente molto sicuro, tuttavia nessun sistema operativo è in grado di auto-proteggersi dai danni che può causare l'utente (soprattutto se possiede i privilegi di amministratore).

1.3 Convenzioni usate nel testo

Comandi e parole riservate (di Bash o altri programmi):

Le unità di misura dell'informazione, sono:

- il bit (simbolo: bit) corrispondente all'unità minima di informazione;
- il byte (simbolo B) equivalente a 8 bit.

Esse e i loro multipli seguono la convenzione dei prefissi del Sistema Internazionale per cui, ad esempio, si ha:

- 1 MiB = 1 024 KiB = 1 048 576 B;
- 1 MB = 1 000 kB = 1 000 000 B.

2 Bash

Bash è letteralmente l'acronimo di "Bourne-Again-SHell",[9] ed è uno dei più usati interpreti di comandi dell'ambiente Unix-Linux.

Che cos'è un'interprete di comandi? È sostanzialmente uno dei modi più semplici e molto spesso più efficaci per comunicare le nostre direttive al sistema operativo che gestisce il nostro computer, scrivendole direttamente su un terminale, oppure su un file, (detto script) da eseguire. D'ora in poi indicherò il generico programma che si occupa di tutto questo con il termine"Shell" (ossia conchiglia o guscio).

Bash [10] non è l'unica shell: vengono spesso usate anche la Korn-Shell (`"ksh"`) e la C-Shell (`"csh"` o la sua versione più avanzata `"tcsh"`). Tuttavia essa è quella installata di default nelle distribuzioni Linux più diffuse ed, essendo molto potente e facilmente personalizzabile, è indubbiamente quella maggiormente utilizzata.

La shell può essere interattiva, ossia si aspetta che le vengano comunicati i comandi, uno a uno, da terminale (ossia inserendoli da tastiera), affinché faccia effettivamente qualcosa, oppure non interattiva, nel qual caso solitamente viene lanciata da uno script, ossia un file di testo contenente una certa sequenza di comandi.

Tutto ciò che appare, quando si ha a che fare con una shell interattiva, (nel nostro caso Bash) è il messaggio di "Prompt" ossia qualcosa del genere:

```
[user@host]$ _
```

Il trattino di sottolineatura (l'*underscore*) è un modo come un altro per indicare il cursore del vostro terminale, ossia il punto dove l'output è terminato, e dove potete quindi cominciare a digitare un comando. Il messaggio di prompt può variare a seconda delle vostre impostazioni (basta cambiare la variabile `"PS1"`, come descritto nella sez. 10.2) ma per semplicità d'ora in poi lo indicherò sempre con un `"$"`.

Per lanciare un singolo comando, basta scriverlo sul terminale, seguito da un Invio (*Enter*).

2.1 Comandi più comuni

Per iniziare facciamo una panoramica sui comandi più comuni, senza conoscere i quali sarebbe difficile capire gli esempi che verranno presentati più avanti.

- **ls:** elenca i file in un certo percorso o secondo un certo modello specificato;

  Esempio:
  ```
  $ ls dir
  ```

- **cd:** cambia la directory di lavoro corrente;

  Esempio:
  ```
  $ cd dir
  $ cd ..
  ```

[9] ossia un'evoluzione della shell di Bourne, che è la shell di default dei sistemi Unix, fin dagli anni '70. Spiegherò tra poco che cosa si intende per shell.

[10] Bash è a tutti gli effetti un nome proprio; essendo però una shell, ed essendo la shell nient'altro che una conchiglia, inevitabilmente la associo al genere femminile; spero che non vi dispiaccia!

- **cp:** copia un file;

  Esempio:
  ```
  $ cp file.txt file.bak
  ```

- **rm:** rimuove un file (o una directory con l'opzione `"-r"`);

  Esempio:
  ```
  $ rm file.bak
  $ rm -rf dir/
  ```

- **mv:** sposta o rinomina un file;

  Esempio:
  ```
  $ mv file.txt file.bak
  $ mv file.txt ..
  ```

- **mkdir:** crea una nuova directory;

  Esempio:
  ```
  $ mkdir documenti
  ```

- **cat:** mostra il contenuto di un file;

  Esempio:
  ```
  $ cat file.txt
  ```

- **echo:** stampa un messaggio sullo schermo;

  Esempio:
  ```
  $ echo "ciao a tutti"
  ```

- **man:** mostra il manuale di un comando;

  Esempio:
  ```
  $ man ls
  ```

- **pwd:** mostra il percorso completo dell'attuale directory di lavoro;

Alcune precisazioni: Il comando `"ls"` accetta nomi di file o directory come argomento dalla linea di comando (o anche più di uno). Nel caso in cui gli argomenti siano file elenca i file (se esistono), mentre nel caso di directory, il comportamento di default è di mostrare tutto il contenuto delle directory. Se non viene specificato nessun argomento, viene mostrato il contenuto della directory corrente. Ovviamente, se non esistono file o directory corrispondenti all'argomento, stamperà sul terminale un messaggio di errore.

È possibile inoltre specificare il percorso del file, in due modi: dando quello relativo alla direcory in cui ci troviamo ora (che viene indicata come `"."`), oppure quello completo, rispetto alla directory principale o radice, in gergo *root* (ossia la `"/"`). Il percorso sarà dunque costituito da una serie di directory, innestate una dentro l'altra e separate ciascuna dal carattere `"/"`.[11] Supponiamo ad esempio di trovarci nella directory `"/home/user/"`, ossia la directory in cui ci si dovrebbe trovare all'avvio della shell (ovviamente con al posto di user il vostro username), e che da ora in poi indicherò come `"HOME"`; supponiamo inoltre che in essa ci sia il file `"file.txt"`. I seguenti comandi elencheranno tutti lo stesso file:

[11] mentre nei sistemi Windows le directory vengono separate dal carattere `"\"`.

```
$ ls file.txt
file.txt
$ ls ./file.txt
./file.txt
```

```
$ ls /home/user/file.txt
/home/user/file.txt
$ ls ../user/file.txt
../user/file.txt
```

La directory `".."` è quella immediatamente superiore (nella struttura delle directory) a quella corrente.

Il comando `"rm"` cancella un file. Attenzione: di default esso non chiede nessuna conferma (a meno che non sia stata specificata l'opzione `"-i"`). Per cancellare una directory, bisogna specificare l'opzione `"-r"` (cancella in modo ricorsivo) e `"-f"` (in caso contrario se è stata specificata da qualche parte l'opzione `"-i"` chiede conferma per ogni singolo file contenuto nella directory). Attenzione però! Questo comando cancella senza chiedere mai conferma qualsiasi file e sotto-directory contenuti nella directory da cancellare.[12]

Ho già suggerito di cominciare a dare questi comandi creando un account di prova, per evitare di fare troppi danni. Ma chiaramente non potrete usare in eterno un account di prova. E per quanto diventiate esperti e facciate attenzione, gli errori capitano e la perdita o la corruzione di file sono sempre in agguato. Per prevenirle esiste un solo metodo: il backup, ossia copiare i dati importanti su supporti diversi da quelli su cui si opera di solito (vedi sez. 17.12).

2.2 Operazioni su file

Per leggere il contenuto di un file (di testo), oltre al già citato `"cat"`, possiamo usare i seguenti comandi:

- `"more"` - mostra il contenuto di un file, una pagina per volta (premendo invio avanza di una linea, spazio di una pagina);
- `"less"` - analogo al precedente, ma con la possibilità di scorrere il file avanti e indietro; per uscire premere il tasto 'q'.

```
$ cat file.txt
...
$ more file.txt
...
$ less file.txt
...
```

Osservazione: se provate a leggere un file non di testo (ad esempio un'immagine o un file binario eseguibile), vi accorgerete che non solo è costituito principalmente da caratteri non leggibili, ma anche che (probabilmente) il vostro prompt è stato modificato con caratteri illeggibili. Questo perché sono stati scritti in output dei caratteri particolari (anche detti di controllo), che non dovrebbero essere stampabili (in realtà dovrebbe dipendere anche dal tipo di terminale). Niente

[12] e nel mondo Unix non esiste l'analogo del comando *undel* del DOS.

panico! Per risolvere il problema scrivete il comando "reset" sul terminale, in modo da ripristinare le impostazioni iniziali (anche se non vedete correttamente il comando, non preoccupatevi, scrivetelo e premete Invio).

Se volete per forza vedere il contenuto di un file del genere, evitando però questi problemi, potete usare l'opzione "-v" del comando "cat" il quale, in questo modo, visualizza solo i caratteri Ascii standard e scrive con un "^-..." i caratteri di controllo e con "M-..." i caratteri superiori al 127; il lato negativo è dunque che, a causa di tutto ciò, con questa opzione non vengono visualizzati correttamente gli accenti.

Per modificare il contenuto di un file di testo lo strumento più comodo da usare è un editor di testi. Di default su Linux è presente il comando "ed", che però risulta alquanto ostico da usare. In realtà l'unica cosa che probabilmente dovreste aver bisogno di sapere su "ed" è come riuscire ad uscirne, se per caso lo avviate per sbaglio da terminale: l'unico comando che lo fa uscire, infatti è "q" (seguito da invio). Se proprio volete farvi del male e volete saperne di più, provate a leggerne il manuale (comando: "man ed").

Per iniziare vi consiglio altri editor: "nano" (o "pico") da terminale, "kedit", "gxedit" o altri ancora da modalità grafica (a seconda della distribuzione e dei pacchetti che avete installato troverete uno o più tra questi). Per utilizzi un po' più avanzati, però, vi consiglio di imparare ad usare "emacs" (potente ma allo stesso tempo abbastanza intuitivo al primo approccio), oppure "vi", o la sua evoluzione "vim" (per quest'ultimo vedi la sez. E).

Segnalo anche "xemacs" come versione con alcune migliorie grafiche di emacs.

2.3 Bash Scripts

I "Bash-Scripts" non sono altro che sequenze di comandi per Bash, scritte in un file. Scopriremo come fare per eseguirli nella sez. 3.2. Una caratteristica importante che contraddistingue gli script per Unix è la prima linea del file la quale indica al sistema operativo, quale deve essere l'interprete di comandi (nel nostro caso è "#!/bin/bash" che corrisponde proprio alla Bash, ma potrebbe essere un'altra shell o anche un altro tipo di programma, di cui il resto del file costituirebbe l'input). Un semplice esempio:

Bash Script di Esempio

```
1 #!/bin/bash
2
3 echo "Hello World!"
```

Un esempio di uno script che non usa Bash potrebbe essere il seguente:

File auto-leggente

```
1 #!/bin/cat
2
3 Attenzione: questo script, una volta eseguito,
4  si auto-legge!
5
6 Fine del messaggio.
```

Potrebbe contenere ad esempio la guida di un programma, e per richiamarla, basterebbe eseguire il file! L'unico difetto (estetico) sarebbe che stampando tutto il suo contenuto, mostra anche la riga `"#!/bin/cat"`; questo può essere evitato sostituendola con `"#!/usr/bin/tail -n +2"` la quale sfrutta il comando `"tail"`, che stampa le ultime `"num"` righe del file (con l'opzione `"-n num"`), o come in questo caso le righe a partire dalla seconda (con l'opzione `"-n +num"`).

2.4 Simboli e caratteri speciali

Simb.	Ascii	Nome italiano	Nome inglese
!	33	punto esclamativo	
"	34	virgolette	quotes, inverted commas
#	35	cancelletto	number sign, hash, sharp[13]
$	36	dollaro	dollar
%	37	percento	percent
&	38	"e" commerciale	ampersand (and)
'	39	apice, apostrofo, virgoletta chiusa	apostrophe
(	40	aperta parentesi tonda	open round bracket
)	41	chiusa parentesi tonda	closed round bracket
*	42	asterisco	asterisk
+	43	più	plus
,	44	virgola	comma
-	45	meno	minus
.	46	punto	dot, period, full stop
/	47	barra	slash[14]
:	58	due punti	colon
;	59	punto e virgola	semi-colon
?	63	punto di domanda	question mark
@	64	chiocciola	at
[	91	aperta parentesi quadra	open square bracket
\	92	barra rovesciata[15]	backslash
]	93	chiusa parentesi quadra	closed square bracket
^	94	cappuccio, accento circonflesso	caret, hat, circumflex
_	95	sottolineatura	underscore
`	96	apice inverso, virgoletta aperta	
{	123	aperta parentesi graffa	open brace bracket
|	124	barra verticale	pipe
}	125	chiusa parentesi graffa	closed brace bracket
~	126	tilde	tilde

Alcuni simboli vengono interpretati in maniera particolare da Bash; dunque se vogliamo poter sfruttare il terminale nel modo migliore è necessario iniziare a

[13] ossia diesis.
[14] oppure *forward slash*, per differenziarlo dal *backslash*.
[15] o "fendente inverso".

conoscerli.

La tilde ("∼") sta ad indicare la propria "HOME". Se le si fa seguire il nome di un utente, Bash sostituirà il tutto con il percorso verso la "HOME" di quell'utente. In aggiunta, "∼+" e "∼-" indicano rispettivamente la directory di lavoro corrente (ossia il percorso completo di ".") e quella precedente (se esiste). "∼"

```
$ echo ~
/home/user
$ cd ~
$ pwd
/home/user
$ cd ~user2
$ pwd
/home/user2
$ echo ~root
/root
```

```
$ cd
$ pwd
/home/user
$ echo ~-
/home/user2
$ echo ~+
/home/user
$ cd -
$ pwd
/home/user2
```

Il punto di domanda ("?") e l'asterisco ("*") possono essere usati per indicare rispettivamente un singolo carattere qualsiasi, oppure una qualsiasi sequenza di caratteri. Ecco un esempio, fatto in una directory preparata con vari file dai nomi opportuni: "?" "*"

```
$ ls
1a.jpg  1.jpg   3.jpg   5.jpg    a2.jpg   b.jpg   file.gif
1b.jpg  2.jpg   4.jpg   a1.jpg   a.jpg    c.jpg   file.txt
$ ls *.jpg
1a.jpg  1.jpg   3.jpg   5.jpg    a2.jpg   b.jpg
1b.jpg  2.jpg   4.jpg   a1.jpg   a.jpg    c.jpg
$ ls ?.jpg
1.jpg   3.jpg   5.jpg   b.jpg
2.jpg   4.jpg   a.jpg   c.jpg
$ ls [123].jpg
1.jpg   2.jpg   3.jpg
$ ls [0-9]*.jpg
1a.jpg  1b.jpg  1.jpg   2.jpg    3.jpg    4.jpg   5.jpg
$ ls *.{jpg,gif}
1a.jpg  1.jpg   3.jpg   5.jpg    a2.jpg   b.jpg   file.gif
1b.jpg  2.jpg   4.jpg   a1.jpg   a.jpg    c.jpg
```

Questo tipo di sostituzione (assieme ad altre che vedremo in questa sezione), viene chiamata espansione. Osserviamo anche che l'espansione viene fatta direttamente da Bash, non dal comando "ls"; infatti:

con "ls":

```
$ ls 1*
1a.jpg 1b.jpg 1.jpg 1.txt
```

con "echo":

```
$ echo 1*
1a.jpg 1b.jpg 1.jpg 1.txt
```

Il risultato in effetti sembra lo stesso. In realtà è sempre preferibile usare "ls", che formatta l'output nella maniera più opportuna (infatti nel caso in cui la sostituzione

comprenda molti file, il comando "echo" li elencherebbe tutti di seguito nella stessa riga, senza andare a capo e senza incolonnarli).

"[]"

Nell'esempio precedente si vede un esempio di utilizzo oltre che dell'asterisco e del punto di domanda, anche delle parentesi quadre e graffe. Le parentesi quadre specificano un set di caratteri, le graffe delle alternative separate da una virgola.

"{}"

Il "$" davanti ad una parola indica che essa è una variabile, e che la shell, quando va ad eseguire il comando, deve sostituirla con il suo contenuto, come nel seguente esempio:

"$"

```
$ var="ciao a tutti"
$ echo $var
ciao a tutti
$ echo "Messaggio: $var"
Messaggio: ciao a tutti
$ ls *.txt
file.txt
$ lista=*.txt
$ echo $lista
file.txt
$ ls file*
file.gif  file.txt
$ nome=file
$ ls ${nome}*
file.gif  file.txt
$ echo ${nome}*
file.gif  file.txt
```

Il nome della variabile può essere racchiuso da delle parentesi graffe e questo risulta utile quando per qualche motivo non si può lasciare uno spazio dopo il nome.

"${}"

```
$ nome=a
$ ls ${nome}1.*
a1.jpg
```

Ci sono inoltre dei modi per inizializzare o modificare la variabile in questione:

ad esempio:

```
$ echo $nome

$ echo ${nome:-default}
default
$ echo $nome

$ echo ${nome:=default}
default
$ echo $nome
default
$ echo ${nome:+ok}
ok
$ echo ${#nome}
7
```

ed inoltre:

```
$ nome=/home/user/file.1.2.1
$ echo $nome
/home/user/file.1.2.1
$ echo ${nome#/home/}
user/file.1.2.1
$ echo ${nome#*/}
home/user/file.1.2.1
$ echo ${nome##*/}
file.1.2.1
$ echo ${nome%.*}
/home/user/file.1.2
$ echo ${nome%%.*}
/home/user/file
```

Osservazione: se modificate il valore di una variabile all'interno di uno script, il valore di questa non sarà salvato all'esterno dello script stesso. Ad esempio:

```
$ var="pippo"
$ echo $var
pippo
$ cat script.sh
#!/bin/bash
var="pluto"
echo $var
$ ./script.sh
pluto
$ echo $var
pippo
```

Per far sì che questo accada occorre che lo script venga lanciato in uno dei seguenti modi (come descritto anche nella sezione 3.2):

con un punto:

```
$ . ./script.sh
```

o equivalentemente con:

```
$ source ./script.sh
```

Per far si, invece, che le variabili impostate da linea di comando, siano lette dai programmi lanciati dalla shell, bisogna esportarle col comando `"export"`. Per maggiori informazioni su questo e sulle altre operazioni possibili con le variabili di Bash, vedi la sez. 5. Come possiamo vedere dall'esempio precedente, le virgolette (") racchiudono una stringa di testo. In esse però vengono ancora sostituite le variabili; per cui se si vuole invece una stringa di testo pura e semplice conviene usare gli apici ("'", codice ascii 39):

```
$ var="ciao a tutti"
$ echo "var = $var"
var = ciao a tutti
$ echo 'var = $var'
var = $var
```

"."

"""

"'"

Il punto esclamativo ("!") subisce un'espansione, in modo tale che il comando che segue viene completato nello stesso modo dell'ultima volta che è stato lanciato: "!"

```
$ ls a*.jpg
a1.jpg a2.jpg a.jpg
$ !ls
ls a*.jpg
a1.jpg a2.jpg a.jpg
$ echo !ls
echo ls a*.jpg
ls a1.jpg a2.jpg a.jpg
```

Oltre che in questo modo, si può risalire ai comandi precedenti con le seguenti espressioni:

- "!!"
- "!$"
- "!#"
- "!n"
- "!-n"
- "!?string?"

Ed è anche possibile operare delle sostituzioni sull'ultimo comando contenente una certa stringa, tramite sequenze di "^": "^"

```
$ ls ?.jpg
1.jpg    3.jpg    5.jpg    b.jpg
2.jpg    4.jpg    a.jpg    c.jpg
$ ^.jpg^.txt^
ls ?.txt
1.txt    2.txt    3.txt
```

È possibile fare dei calcoli aritmetici, ed assegnarne il risultato a delle variabili, attraverso il costrutto "$[...]", oppure equivalentemente con "$((...))". Ad esempio: "$[]" "$(( ))"

```
$ declare -i num
$ num=1
$ echo $num
1
$ i=$[num+1]
$ echo $num
2
$ i=$((num*2))
$ echo $num
4
```

```
$ echo $[10/2]
5
$ echo $[10/3]
3
$ echo $[10%3]
1
$ echo $[2**3]
8
$ echo $[-2*5]
-10
```

```
$ echo $[2#11]
3
$ echo $[8#11]
9
$ echo $[0xA]
10
$ echo $[010+01]
9
$ echo $[0xA+2#11]
13
```

L'istruzione "declare -i num" specifica che la variabile "num" può assumere come valori solo dei numeri (interi). In realtà non è necessaria, in quanto Bash eseguirebbe lo stesso le operazioni matematiche che seguono, tuttavia può risultare utile negli script per evitare (o perlomeno aiutare a scoprire) certi errori.

L'apice inverso ("`", codice ascii 96), racchiude un'espressione che viene valutata (esattamente come un comando eseguito a parte) e che viene poi sostituita con l'output ottenuto. In questo modo è facile assegnare ad una variabile l'output di un comando: "`"

```
$ echo `ls a*.jpg`
a1.jpg  a2.jpg  a.jpg
$ var=`ls file.*`
$ echo $var
file.gif  file.txt
$ echo $(cat file.txt)
....
$ echo "$(cat file.txt)"
....
```

La stessa cosa si può fare utilizzando il costrutto: "$(comando)", come negli ultimi due comandi dell'esempio qui sopra. Osservazione: confrontate bene il risultato dei due comandi, solo in apparenza uguali, applicandolo ad un file con varie linee di testo. "$()"

Il fatto che Bash interpreti in maniera speciale tutti questi simboli, crea qualche difficoltà nel caso, ad esempio, in cui sono usati all'interno del nome di un file, o di una directory, o di un comando. Essenzialmente, per eliminare questi problemi, ci sono due modi: far precedere il simbolo da un backslash ("\", codice ascii 92), oppure racchiudere la stringa tra apici ("'"). Uno dei casi tipici (che crea spesso un mucchio di problemi) è il caso di un nome di file contenente degli spazi: "\"

```
$ cp file.txt "file copia"
$ ls file*
file copia  file.txt
$ for file in file* \
> do echo :$i: \
> done
:file copia:
:file.txt:
$ for file in `ls file*` \
> do echo :$i: \
> done
:file:
:copia:
:file.txt:
$ rm -f file\ copia
$ ls file*
file.txt
```

```
$ cp file.txt 'a$ciao!'
$ ls *ciao*
a$ciao!
$ ls *\$*
a$ciao!
$ ls a?ciao?
a$ciao!
$ rm -f a\$ciao\!
```

Il comando "for" verrà descritto nella sezione 7.1. Qui è interessante notare invece l'uso del backslash per separare il comando su più righe. La shell ci fa capire che è in attesa di ulteriori comandi, visualizzando un prompt diverso, in questo caso il simbolo ">".

Infine, a proposito di file con nomi particolari, può esservi utile questa dritta: se avete un file di nome "-nome" e, volendo cancellarlo, lanciate il comando "rm -f", riceverete un errore (vedi l'esempio sotto). Questo perché "rm", come molti altri comandi, considera gli argomenti che iniziano con dei "-" (meno) come delle opzioni. Se volete convincerlo che quella non è un'opzione ma proprio il nome del file, dovete informarlo che le opzioni sono finite con un "--" (doppio meno), ossia nel nostro caso:

```
$ rm -f -nome
rm: invalid option -- n
$ rm -f -- -nome
```

In alternativa basta indicare il file con il suo percorso, in modo che non ci sia un meno davanti; ad esempio, banalmente, basta indicarlo come "./-nome".

Sebbene in genere sia meglio evitare queste complicazioni, assegnando ai file dei nomi standard,[16] in teoria potete inserire praticamente qualsiasi simbolo nel nome di un file, tranne ovviamente il carattere "/" che è dedicato alla separazione tra le directory.

Il punto e virgola (";"), viene usato per separare diversi comandi sulla stessa linea; ad esempio: ";"

```
$ echo "Aspetta 10 secondi"; sleep 10; echo "Stop"; ls
...
```

Se si postpone un "&" ad un comando, questo viene eseguito in *background* (letteralmente "sullo sfondo"), ossia la shell non aspetta che esso termini prima di essere di nuovo pronta a ricevere comandi (il comportamento di default, viene indicato come *foreground*, "in primo piano"). Questo è utile per lanciare programmi che utilizzano la modalità grafica, senza tenere occupato il terminale, come ad esempio: "&"

```
$ firefox &
```

Vedremo però come questo metodo per lanciare comandi in background non sia sempre il migliore. Succede infatti che se si chiude il terminale, anche i programmi ancora in esecuzione, che sono stati lanciati da esso, vengono terminati.

Il simbolo "!" all'inizio di una riga fa in modo che Bash la ignori completamente; se non si trova all'inizio, viene comunque ignorato tutto il resto della riga. Questo risulta utile per inserire dei commenti in uno script. Il simbolo ":" invece fa in modo che gli argomenti della riga vengano correttamente interpretati ed espansi, ma senza eseguire nessun comando. "!" ":"

[16] l'ideale sarebbe senza accenti o altri caratteri speciali, e non troppo lunghi, per evitare problemi nel copiarli su filesystem diversi

Ci sono ancora altri simboli da conoscere; alcuni (`">"`, `"<"`, `"|"` e simili) verranno descritti nella sez. 4.1 a proposito dell'input/output dei comandi. Soffermiamoci invece sui costrutti `"[ ]"` e sull'analogo `"[[ ]]"`, i quali servono a delimitare dei test o dei confronti (matematici e non) e il cui impiego verrà descritto ampiamente nella sez. 6. La differenza tra i due, consiste nel fatto che il primo non è altro che un modo alternativo di richiamare il comando `"test"`, mentre il secondo è un *built-in* (un comando incorporato) di Bash. Questo spiega anche perché il primo ha sempre bisogno di uno spazio tra le parentesi quadre (che sono un comando vero e proprio) e i termini da valutare al loro interno.

Caratteri Speciali - col comando `"echo"`, possiamo stampare sul terminale qualsiasi codice Ascii; alcuni di questi hanno però un significato speciale, e come tali vengono interpretati in maniera diversa dal terminale.

Ecco una tabella che li riassume:

Codice				Significato
0	\000	"\0"	(Null)	carattere nullo
7	\007	"\a"	(Bell)	campanella di sistema
8	\010	"\b"	(BackSpace)	cancella carattere precedente
9	\011	"\t"	(Tab)	tabulazione
10	\012	"\n"	(Newline)	va a capo
11	\013	"\v"	(Vertical Tab)	tabulazione verticale
12	\014	"\f"	(Form Feed)	avanzamento di pagina [17]
13	\015	"\r"	(Carriage Return)	ritorna ad inizio riga
27	\033	"\e"	(Escape)	inizia sequenza di escape

Nota: I sistemi operativi DOS/Windows utilizzano di default, per segnalare l'andata a capo nei file di testo, la sequenza di caratteri `"\n\r"`, mentre su Linux e gli Unix in generale, viene usato il solo carattere `"\n"`. Questo spiega perché capita spesso che un file di testo scritto su Linux, non venga letto correttamente su Windows (a parte il problema degli accenti nel caso vengano usati set di caratteri diversi), mentre i file scritti su Windows, presentano su Linux dei fastidiosi caratteri aggiuntivi a fine riga. Per risolvere il problema basta comunque usare un editor di testi un po' più intelligente del "Blocco Note" di Windows, oppure (come vedremo in un esempio nella sez. 12.2) utilizzare degli strumenti per la conversione automatica tra i due formati.

Colori - Una nota su come generare delle scritte colorate con la shell. Se vogliamo stampare sul terminale una scritta colorata, dobbiamo usare le cosiddette sequenze di "escape"; un esempio: [18]

[17] può essere interpretato in diversi modi a seconda del programma; ad esempio con cat va ad una nuova linea, mentre con more ad una nuova pagina.

[18] i colori effettivi mostrati nel presente esempio sono stati messi solo per dare un'idea, ma non sono certo quelli del terminale e, in ogni caso, anche questi ultimi possono variare in base alla configurazione.

```
$ echo -e "\033[0;31mrosso \033[0;32mverde\033[0m"
rosso verde
$ echo -e "\e[0;34mblu \e[0;35mmagenta\e[0m"
blu magenta
```

Questi codici funzionano sia sulle console non grafiche che sui terminali in modalità grafica, anche se i colori effettivi potrebbero differire (anche radicalmente) tra un caso e l'altro. Da notare che la sequenza `"\033["` (oppure `"\e["`), è sostituibile con il codice `"\233"`. [19]

Ecco una tabella con i codici dei vari colori e delle altre proprietà impostabili per il testo stampato su terminale:

Codice	Effetto	Codice	Colore testo	Codice	Colore sfondo
00	Normale	30	Nero	40	Sfondo nero
01	Grassetto	31	Rosso	41	Sfondo rosso
02	Opaco	32	Verde	42	Sfondo verde
04	Sottolineato [20]	33	Giallo	43	Sfondo giallo
05	Lampeggiante [20]	34	Blu	44	Sfondo blu
07	Inv. testo/sfondo	35	Magenta	45	Sfondo magenta
08	Invisibile	36	Azzurro	46	Sfondo azzurro
		37	Bianco	47	Sfondo bianco
		39	Default	49	Default

In realtà gli effetti descritti sono quelli standard, che di solito funzionano nelle console non grafiche, ma che vengono interpretati in maniera diversa dai terminali grafici. Ad esempio `"xterm"` dispone di tutta una serie di opzioni che specificano come comportarsi nel rendere questi effetti.

Nota: l'attributo del testo "nascosto", non va usato per cose del tipo nascondere una password in quanto questa anche se non visibile direttamente viene comunque passata al terminale e dunque rintracciabile in vari modi. Potete averne una conferma indiretta, osservando il risultato dei seguenti comandi:

```
$ echo -e "\033[8mpassword"

$ pippo=`echo -e "\033[8mpassword"`
$ echo $pippo | cat -v
^[[8mpassword
```

Posizionamento del cursore. È possibile posizionare il cursore in una data riga e colonna, mediante sequenze di escape del tipo:

```
$ echo -en "\033[x;yH"
```

È inoltre possibile salvare la posizione corrente del cursore:

[19] da notare che nelle nuove versioni di bash/echo? ci vuole sempre uno zero davanti alla sequenza numerica, per indicare che il numero è ottale; in questo caso, dunque, \0233.

[20] con i terminali che lo supportano.

```
$ echo -en "\033[s"
```
per poi ripristinarla mediante il comando:
```
$ echo -en "\033[u"
```
Un altro modo per effettuare questi spostamenti è usare il comando "tput" (vedi 16).

Per utilizzi più raffinati, tuttavia, conviene usare la libreria "ncurses" del C.

Allarme/Campanella. Il comando "bell"

```
$ echo -e "\a"
$ echo -e "\007"
```

Codici:
\033[10;nnn] imposta la frequenza (in hertz)
\033[11;nnn] imposta la durata (in millisecondi)
Valori di default:

Mini-Riassunto dell'uso dei simboli e di sequenze di caratteri speciali:

Simbolo	Significato
. (punto)	include del codice ("source")
‘ (apice inverso [21])	sostituzione di comando
” (virgolette [22])	racchiude una stringa
’ (apice [23])	racchiude una stringa costante senza interpretare i simboli speciali
\ (backslash)	precede simboli speciali e sequenze di “escape”
∼ (tilde)	directory HOME
! (punto esclamativo)	sostituisce corrispondenze nella cronologia dei comandi (HISTORY)
? (punto di domanda)	
* (asterisco)	
{ } (parentesi graffe)	
[ ] (parentesi quadre)	
&	comando in background
;	separa lista di comandi
:	
#	commento
$( )	sostituzione di comando
${ }	effettua sostituzione di variabile
[ ]	verifica test/confronti (comando "test")
[[ ]]	verifica test/confronti (built-in Bash)
(( ))	modalità matematica
$[ ]	modalità matematica

[continua . . .]

[21] codice Ascii 96
[22] codice Ascii 34
[23] codice Ascii 39

Simbolo	Significato
$(( ))	modalità matematica
\nnn	inserisce carattere dal codice ascii (in ottale)

2.5 Altre cose importanti sulla shell

Abbiamo visto che un comando può essere lanciato dalla shell in background (con un `"&"` alla fine). Possiamo però anche interrompere provvisoriamente un programma già lanciato (in *foreground*) e mandarlo successivamente in background. Per interromperlo, si deve premere la sequenza di tasti "Control + Z" (spesso essa viene indicata come `^Z`). Attenzione: questo non lo fa terminare, ma solamente bloccare. Per riprenderlo usate il comando `"fg"` se lo rivolete in foreground, oppure `"bg"` se lo volete mandare in background. Per maggiori dettagli sul funzionamento di questi comandi, e su come bloccare, far terminare o far riprendere i comandi lanciati, vedere la sezione 9.

Riassunto dei segnali da tastiera:

- Per interrompere un programma in foreground: premere i tasti "Control + C" (`^C`). In realtà questo è il comportamento di default per molti programmi, ad esempio un normale script, ma molti altri prevengono l'uscita causata dall'invio di questo segnale.
- Sospendere un programma in foreground: `^Z`.
- Inviare un segnale di uscita al processo: `^\` (vedi anche sez. 9).
- Per terminare l'input da tastiera: `^D`, che corrisponde alla funzione "EOF" (*End of File*), che segnala la fine dell'input.
- Per cancellare la schermata: `^L`.
- Cancellare la riga di input corrente: `^U`.
- Cancellare il carattere precedente: `^H` (equivalente al carattere di "Backspace")
- Carattere di "Invio": `^J`.
- Andare a capo: `^M`.
- Produrre un beep (*bell*): `^G`. Da notare che nei terminali in modalità grafica, solitamente non viene prodotto alcun suono, ma viene rinfrescata la schermata. Per cambiare questo comportamento vedi la sezione 22 in cui si tratta la configurazione di X.
- Interrompere l'input da tastiera: `^S`, mentre per riprenderlo `^Q`.

3 File, directory e permessi

Nozione di *filesystem*: (il modo in cui è formattata la partizione dell'hard-disk su cui lavorate).

Nota: quello che sto per dire è pensato specificatamente per un tipico sistema Linux, dunque con un filesystem di tipo ext2 o ext3. Tuttavia quasi tutto sarà ugualmente valido su molti altri filesystem di ambienti simili (FreeBSD, Mac OSX, etc).

3.1 I comandi ls e stat

Innanzitutto ogni file è accompagnato da una serie di informazioni che lo caratterizzano: data di creazione, ultima modifica, ultimo accesso, nome del proprietario, del gruppo, permessi di lettura, esecuzione e modifica.

Potete vedere tutte queste informazioni in vari modi, ma principalmente usando due comandi: ls e stat.

```
$ ls -l file.txt
-rw-r--r--    1 user    group       147 ago  7 10:57 file.txt
```

L'opzione `"-l"` del comando `"ls"`, fa scrivere in un'unica riga i permessi del file, il numero di collegamenti (link) verso di esso, il nome del proprietario e del gruppo, la dimensione (in byte), data e orario dell'ultima modifica ed infine il nome. Se si vuole invece l'orario di ultimo accesso al file, bisogna specificare anche l'opzione `"-u"`; infine se si vuole l'orario di creazione, basta usare l'opzione `"-c"`.

Tuttavia per ottenere queste ultime informazioni (assieme ad altre che non starò a spiegare) è più conveniente usare il comando `"stat"`:

```
$ stat file.txt
File: `file.txt'
Size: 147         Blocks: 8   IO Block: 4096  Regular File
Device: 302h/770d  Inode: 83680  Links: 1
Access: (0644/-rw-r--r--) Uid: (500 / user) Gid: (501/group)
Access: 2007-08-07 15:48:00.000000000 +0200
Modify: 2007-08-07 10:57:08.000000000 +0200
Change: 2007-08-07 10:57:08.000000000 +0200
```

Interpretiamo le principali informazioni che ci vengono date: il nostro file occupa 147 byte; la sua ultima modifica risale alle ore 10:57 del 7 Agosto 2007; appartiene all'utente di nome `"user"` ed al gruppo (una categoria che può comprendere più utenti) `"group"`.

Per cambiare le informazioni sull'orario dell'ultimo accesso e quello dell'ultima modifica ad un file, si può usare il comando `"touch"`. Ad esempio:

```
$ touch file.txt
```

Se `"file.txt"` esiste, vengono aggiornate all'istante attuale data e ora di ultimo accesso e modifica; se non esiste viene creato un nuovo file (vuoto) con quel nome.

3.2 Permessi

Soffermiamoci adesso sui permessi attribuiti ai file (e alle directory). Essi possono essere visualizzati in due modi, numerico o simbolico. Si ha: 1: Esecuzione, 2: Scrittura, 4: Lettura. Oppure, x: Esecuzione, w: Scrittura, r: Lettura.

I permessi scritti in modo numerico hanno il vantaggio di poter essere sommati tra di loro, per cui il permesso di lettura (4) più quello di scrittura (2) sono rappresentati dal numero 6, mentre quello di lettura più quello di esecuzione (1) corrispondono al numero 5.

Vedendo il tutto come un numero binario è immediato capire che ogni permesso corrisponde ad un preciso bit:

o	b	permessi	
0	000	`"---"`	nessun permesso
1	001	`"--x"`	sola esecuzione
2	010	`"-w-"`	sola scrittura
3	011	`"-wx"`	scrittura + esecuzione
4	100	`"r--"`	sola lettura
5	101	`"r-x"`	lettura + esecuzione
6	110	`"rw-"`	lettura + scrittura
7	111	`"rwx"`	tutti i permessi

La sequenza di permessi di un file, in realtà, è costituita da tre di questi numeri (bit), corrispondenti ai permessi per l'utente proprietario del file (u), per gli altri appartenti al suo stesso gruppo (g) ed infine per tutti gli altri (o).

Ad esempio un file leggibile scrivibile ed eseguibile dal proprietario, e leggibile ed eseguibile dagli altri avrà dei permessi, descritti dall'output di ls, pari a `"rwxr-xr-x"` o equivalentemente con numeri ottali, a 755.

Nel caso delle directory il permesso di esecuzione è equivalente a quello di potervi accedere o meno col comando cd, mentre quello di lettura a quello di poter elencare i file al suo interno (ad esempio con `"ls"`).

Oltre a questi permessi (corrispondenti come abbiamo detto ad un numero a tre bit) ci sono altre proprietà particolari, corrispondenti ciascuna ad un altro bit, come il "suid" (*set user ID*), per cui il file viene eseguito dal proprietario, anche se a lanciarlo è un altro utente, lo "sgid" (*set group ID*) che fa la stessa cosa ma per il gruppo, e il "bit sticky" per le directory destinate a contenere file temporanei, scrivibili da tutti gli utenti (di solito `"/tmp"` e `"/var/tmp"`).

Ovviamente l'amministratore (l'utente `"root"`) non ha bisogno dei permessi per leggere/modificare/cancellare i file degli altri utenti (altrimenti che amministratore sarebbe?).

Per impostare o cambiare i permessi, si usa il comando `"chmod"`. Anche questo può essere usato in due modi, specificando i permessi con il numero ottale corrispondente (di tre cifre), oppure specificando le modifiche da fare con le lettere (`r/w/x` e `u/g/o`) e i simboli `+/-`. Ecco alcuni esempi:

```
$ chmod +x file.sh
```

rende eseguibile il file `"file.sh"`.

```
$ chmod u+x file.sh
```

rende eseguibile (al solo proprietario) il file.

```
$ chmod 755 file.sh
```
imposta numericamente i permessi del file.

Ci possiamo però chiedere una cosa: nel primo dei tre esempi qui sopra, a chi viene dato il permesso di esecuzione, solo al proprietario o a tutti quanti? Per rispondere a questa domanda dobbiamo prima parlare di un altro comando. Quando viene creato un nuovo file (in qualche modo, ma dalla shell), di default i permessi vengono settati in base ad una "maschera" definita dal comando `"umask"`. Ad esempio con il comando:

```
$ umask 027
```

lanciato da terminale, i file successivamente creati in quella sessione avranno permessi `"-rw-r-----"`, mentre le directory `"drwxr-x---"`.

La risposta alla domanda di prima, dunque è che a seconda della maschera, cambia anche il comportamento di default di `"chmod"`. Se ad esempio la maschera è `000`, il permesso verrà aggiunto a tutti, mentre se è `077` verrà aggiunto solo al propietario. Fate delle prove con delle maschere diverse per capire il funzionamento.

I comandi `"lsattr"` e `"chattr"` mostrano e modificano alcune caratteristiche possedute dai file in un filesystem di tipo ext2. Il loro utilizzo va oltre lo scopo di questo manuale; se avete bisogno di maggiori informazioni provate a consultare [1].

Esecuzione - Per eseguire un file (di cui abbiamo i permessi di esecuzione) ad esempio uno script di nome file.sh, basta scrivere: `"./file.sh"`

Il punto iniziale serve per due motivi:

1. I comandi vengono cercati da Bash in un certo elenco di percorsi a lei noti (vedi sez. 5.4); se la directory attuale non è in quell'elenco, Bash non potrà trovare il comando.

2. Inoltre in questo modo possiamo essere assolutamente certi che Bash esegua proprio quel file e non magari un altro con lo stesso nome, presente in qualche altra directory di quell'elenco.

In alternativa, il comando `"sh file.sh"` eseguirà allo stesso modo il file, anche se questo non ha i permessi di esecuzione.

Con `"exec file.sh"`, il file (o il comando) viene eseguito in una sub-shell; se il comando esce con successo, la shell viene chiusa (a meno che il comando non sia contenuto in uno script, ossia una shell non interattiva).

Infine se scriviamo `"source file.sh"` avremo che il file sarà eseguito nella stessa shell e non in una subshell, come se tutto il suo input fosse digitato sul terminale (e anche senza i permessi di esecuzione); la conseguenza principale di questo, come abbiamo già visto prima, è che se vengono modificate delle variabili nello script, il valore di queste verrà salvato anche all'uscita, cosa che invece non succede negli altri casi.

È possibile anche cambiare il proprietario o il gruppo a cui appartiene il file; tuttavia questo, per ovvi motivi di sicurezza potrà essere fatto solamente dall'utente root (l'amministratore del sistema). Ad esempio:

```
$ ls -l file.sh
-rwxr-xr-x   1 user      group      198 ago 26 15:20 file.sh
$ chown new_user file.sh
$ ls -l file.sh
-rwxr-xr-x   1 new_user  group      198 ago 26 15:20 file.sh
$ chgrp new_group file.sh
$ ls -l file.sh
-rwxr-xr-x   1 new_user  new_group  198 ago 26 15:20 file.sh
```

3.3 Tipi di file

Nei sistemi di tipo Unix, come vedremo, quasi tutto è un file: le directory ad esempio sono dei file particolari. I vari dispositivi (dischi, tastiera, mouse, scheda audio, stampante, etc) sono associati a dei file speciali detti device, nella direcory `"/dev"`. Addirittura i processi sono visibili come dei particolari file nella direcory `"/proc"`.

In un file-system (vedi la sez. successiva) di tipo ext2/ext3 possiamo elencare i seguenti tipi di file, distinguibili in base al primo carattere dell'output del comando `"ls -l"`

- (-) File normale
- (d) Directory
- (l) Link simbolico
- (p) Fifo (Pipe)
- (s) Socket
- (c) Character Device
- (b) Block Device

Vedremo le caratteristiche dei vari tipi nelle prossime sezioni. Soffermiamoci qui sui link. Nei filesystem `"ext2"` e simili, ci sono due tipi di link, fisici e simbolici (alcune volte vengono anche definiti rispettivamente *hard* e *soft*).

```
$ touch file1
$ ln file1 file2
$ ln -s file1 file3
$ ls -l file?
-rw-r--r--  2 user  group   0  feb 2 17:53  file1
-rw-r--r--  2 user  group   0  feb 2 17:53  file2
lrwxrwxrwx  1 user  group   0  feb 2 17:53  file3 -> file1
```

In pratica la differenza consiste nel fatto che, nel primo caso, viene creato un file che ha in comune lo spazio occupato sul disco con un altro. Se si cancella uno dei due, l'altro non ne risente. Nel secondo caso, si ha semplicemente un file che

rimanda ad un altro (più o meno come un link in una pagina Html). Se il file originale viene cancellato, o semplicemente spostato, il link simbolico non ha più nessuno a cui fare riferimento, per cui viene definito *orfano*. Qualunque operazione su quest'ultimo (a parte la cancellazione) porterà inevitabilmente ad un errore. Va osservato inoltre che quando si cancella un link simbolico (ad es. tramite il comando `"rm"`), viene cancellato il link, non il file a cui fa riferimento. Invece qualunque altra operazione di lettura/scrittura/modifica sul link simbolico, agisce in realtà sul file fisico a cui il link fa riferimento.

Oltre che link verso dei file, posso creare dei link verso directory; in questo caso però generalmente possono essere solo di tipo simbolico.[24]

Ovviamente non è possibile creare link fisici tra due partizioni o tra due dischi diversi.

3.4 Albero delle directory

Per poter sfruttare appieno la shell su di un sistema Linux, è necessario sapere qualcosa su come sono strutturati i file e le directory.

Innanzitutto avremo una directory principale, in cui sono innestate tutte le altre; essa viene chiamata directory radice (*root*) ed è indicata con `"/"`. Ogni directory del sistema avrà un certo percorso a partire dalla radice; ad esempio se volete vedere il percorso completo della directory di lavoro corrente, lo potete visualizzare col comando `"pwd"`. Se lanciamo il comando `"ls"` all'interno della directory `"/"`, ci accorgiamo che di default (intendo subito dopo l'installazione, senza nostre successive modifiche) sono già presenti molte sotto-directory. Ad esempio:

```
$ ls /
bin     etc     lib     proc    tmp
boot    home    mnt     root    usr
dev     initrd  opt     sbin    var
```

Ognuna di queste è dedicata ad uno scopo ben preciso (o a contenere file di un tipo ben preciso). Per ora ci basta sapere che:

- Le directory `"/bin"`, `"/sbin"`, `"/lib"` e `"/usr"` contengono tutti i vari programmi e librerie installati nel sistema.
- `"/etc"` contiene i principali file di configurazione del sistema. Ad esempio `"/etc/passwd"` ed `"/etc/shadow"` contengono le passord e le altre informazioni sugli utenti (non pensate però di poter vedere così facilmente le password degli altri utenti: sono criptate, ed inoltre solitamente gli utenti normali non hanno il permesso di lettura per questi file). `"/etc/group"` contiene, analogamente, le informazioni sui vari gruppi. `"/etc/issue"` contiene il messaggio mostrato al login, `"/etc/hosts"` contiene una lista dei computer (e dei loro IP) conosciuti.

[24] secondo la pagina di manuale di `"ln"`, l'amministratore può creare anche link fisici tra directory (da verificare).

- "/home" contiene (di solito) le varie "HOME" degli utenti ("/home/nome-utente").
- "/tmp" è una directory scrivibile da tutti gli utenti, per file temporanei (ad esempio la cache di alcuni programmi) e che di solito viene cancellata automaticamente ogni tot tempo (ma questo dipende dalla configurazione).
- "/var" contiene i messaggi di log dei vari programmi, nonché i file in coda per qualche servizio, ad esempio i file che stanno per essere spediti per mail (se è installato un server di posta come sendmail) quelli in arrivo, e i file in coda di stampa.
- Infine quelle che più ci interessano ora, e il cui utilizzo descriverò qui di seguito, sono "/dev" e "/mnt".

Ogni dispositivo (in gergo *device*) (ad es. hard-disk, floppy, zip, CD/DVD, pen-drive) viene associato dal sistema operativo ad un ben preciso file contenuto nella directory "/dev".

Ad esempio: "/dev/hd*", "/dev/sd*", "/dev/fd*". Spesso esistono dei link a questi file, dal nome molto più illuminante: "/dev/cdrom", "/dev/floppy", "/dev/mouse", etc.

Una qualsiasi di queste unità può essere suddivisa in partizioni, ognuna delle quali può essere vista come un disco a parte. Ad esempio se il nostro unico hard-disk ha 4 partizioni:

```
$ ls /dev/hda*
/dev/hda1 /dev/hda2 /dev/hda3 /dev/hda4
```

Le varie partizioni dei vari dispositivi vengono montate attraverso il comando "mount", in una directory detta punto di mount.

Per convenzione i vari dispositivi/partizioni vengono montati in sottodirectory della directory "/mnt/".

```
$ mount -t tipo_fs /dev/dispositivo /mnt/mount-point
```

3.5 Filesystems

Famiglie di filesystems Unix: ext2, ext3, ext4, reiser-fs, ...

Journaled Filesystems

Famiglie di filesystems DOS/Windows: Fat12, Fat16, Fat32, Ntfs.

I filesystems Fat16 e Fat32, sono i filesystem attualmente più diffusi (nel mondo Windows ma non solo); la quasi totalità di hard-disk, pen-drive e memory-card in vendita è formattata in uno di questi modi. Essi hanno comunque dei limiti: Fat16 può gestire delle partizioni da 32 MiB fino ad una grandezza massima di 2 GiB, mentre Fat32 può gestire delle partizioni da un minimo di 512 MiB fino ad un massimo di 2 TiB e file di una dimensione massima di 4 GiB. Quando Fat32 venne creato (più o meno con Windows 98) questi limiti erano inavvicinabili, oggi invece

cominciano a stare stretti ad alcuni utenti con esigenze specifiche (ad esempio la manipolazione di video è in genere l'operazione che occupa più risorse in termini di spazio su disco).

In ogni caso, prima con Windows XP e poi con Vista, si è passati definitivamente al filesystem Ntfs, almeno per gli hard-disk interni, dedicati al sistema operativo.

La buona notizia è che tutti quanti questi filesystem sono attualmente supportati da Linux. Esempi di come montare queste partizioni (da root):

```
$ mount -t vfat /dev/hda2 /mnt/dati
$ mount -t ntfs /dev/hda1 /mnt/vista
```

Famiglie di filesystem Macintosh: Nella preistoria informatica i Macintosh avevano il ... Oggi Macintosh è Mac Os X, ed usa

Gli altri filesystems: partizione di Swap, Cd-Rom/DVD.

Una nota a parte merita la partizione di "swap". Che cos'è? Quanta swap serve?

I CD-Rom e i DVD (contenenti dati, non audio o video), utilizzano comunemente un filesystem standard chiamato "ISO9660", il quale ha dei limiti abbastanza rigidi per quanto riguarda la lunghezza dei nomi dei file, etc. Per fortuna, come vedremo nella sez. 17.13, esistono dei modi per sopperire a queste mancanze. Si tratta di decidere se rispettare tutte le restrizioni per privilegiare la compatibilità con tutti sistemi, anche del passato , o rinunciarvi per sfruttare le caratteristiche non standard.

Ovviamente i filesystem, oltre che montati, possono essere smontati. Per farlo si utilizza il comando "umount".

In generale non sarà necessario montare ogni volta le partizioni che sono fisse sul vostro sistema (ad esempio quelle dei vari hard-disk), in quanto queste possono essere montate automaticamente all'avvio, configurando opportunamente il file "/etc/fstab".

Esempio di file /etc/fstab

```
1  #Device    Mount-Point   Type     Options
2
3  /dev/hda1  /mnt/windows  vfat     iocharset=iso8859-15,\
4                                     umask=0     0 0
5  /dev/hda2  /             ext2     defaults    1 1
6  /dev/hda3  /home         ext2     defaults    1 2
7  /dev/hda4  swap          swap     defaults    0 0
8  /dev/fd0   /mnt/floppy   vfat     sync,nosuid,noauto,user,nodev,\
9                                     iocharset=iso8859-15,umask=0    0 0
10 /dev/hdb4  /mnt/zip      vfat     noauto,owner,user,umask=0       0 0
11 /dev/scd0  /mnt/cdrom    iso9660  noauto,owner,iocharset=iso8859-15,\
12                                    ro,umask=0    0 0
13 /dev/sda1  /mnt/usb      vfat     noauto,owner,user,umask=0       0 0
```

I filesystems che non hanno l'opzione "noauto", vengono montati tramite il comando:

```
$ mount -a
```

che di solito viene lanciato all'avvio del sistema (di sicuro è in qualche file di configurazione che dipende però dalla distribuzione).

Nel file "/etc/mtab", sono elencati i file-systems attualmente montati sul sistema. Essi sono elencabili da terminale con il comando "mount" (senza argomenti).

Floppy:

Cd-Rom/Dvd:

HD-Dvd/Blu-Ray Disc:

Zip: "/dev/hdd4"

Penna Usb/Lettori Mp3/Hard-Disk esterni:

La maggior parte delle distribuzioni di Linux recenti, possiede un qualche strumento per gestire l'inserimento di nuove partizioni (tipicamente una pen-drive, un hard-disk esterno o una fotocamera digitale) in modo automatico, almeno nella modalità grafica.

Su molte di queste (ad esempio Ubuntu) è Gnome che si occupa di tutto. Può essere utile tuttavia sapere come fare tutto da shell.

Esistono i comandi "gnome-mount", "gnome-umount" e "gnome-eject", utilizzabili da linea di comando (anche da modalità non grafica).

Prerequisiti: hald, gnome

Attraverso il programma "ivman", è possibile gestire il mount automatico di nuovi dispositivi, anche da modalità non grafica. In pratica il dispositivo non appena viene inserito, sia esso CD o penna usb, viene automaticamente montato in una certa directory (in "/mnt" o "/media").

Per informazioni su come creare/modificare le partizioni e formattarle (ossia crearvi un certo file-system) vedere la sezione 17.11.

3.6 Localizzazione di file e comandi

Come faccio a sapere dovè un certo file? E se un certo comando è installato oppure no?

Il primo passo è sfruttare il completamento automatico di Bash: si scrive il nome del comando o parte di esso, se esso viene completato o è compreso nella lista presentata dalla shell, allora esso è installato e si trova in uno dei percorsi in cui Bash cerca i comandi (elencati nella variabile "PATH", vedi sez. 5.4). A questo punto se si vuole sapere qualè esattamente il percorso in cui si trova il file, si può utilizzare il comando "whereis". Se ci sono più file eseguibili con quel nome, per sapere quale sarà esattamente quello che verrà eseguito (quello nella directory che viene per prima nel "PATH"), si può usare il comando "which".

Esempio:

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.bz2
$ whereis mpg123
mpg123: /usr/local/bin/mpg123
 /usr/local/share/man/man1/mpg123.1.bz2
$ whereis startx
startx: /usr/X11R6/bin/startx  /usr/bin/X11/startx
$ which startx
/usr/X11R6/bin/startx
```

Come si può vedere "whereis" mostra non solo il percorso dell'eseguibile, ma anche eventualmente della pagina di manuale. Tuttavia se il comando è stato installato in qualche percorso non standard, "whereis" molto probabilmente non riuscirà a trovarlo. In questo caso si può tentare con il comando "find":

```
$ for dir in `echo $PATH | sed "s/:/ /g"`; do
> find $dir -name gnuplot; done
/usr/bin/gnuplot
```

in cui viene cercato il file "gnuplot" in ogni directory contenuta in "PATH". Attenzione però, il comando così com'è scritto darà errore se in PATH ci sono nomi di directory con degli spazi. Per prevenire questo problema si può sfruttare la variabile "IFS".

```
$ OLDIFS="$IFS"; IFS=:
$ for dir in $PATH; do find $dir -name gnuplot; done
/usr/bin/gnuplot
$ IFS="$OLDIFS"
```

Naturalmente si può usare la stessa tecnica per cercare un file anche negli altri percorsi definiti dal sistema, ad esempio "LD_LIBRARY_PATH", o in "MANPATH" (vedi sez. 5), oppure inserendo a mano un qualsiasi altro percorso. Inoltre find permette di cercare file corrispondenti ad un certo modello; ecco alcuni esempi:

```
$ find /usr/bin -name gnuplot
/usr/bin/gnuplot
$ find /usr/local/bin -name "*123"
/usr/local/bin/mpg123
$ find /usr/local/ -type f -perm 755
...
```

L'ultimo comando mostra tutti i file eseguibili contenuti in "/usr/local/" e nelle sue sotto-directory.

Il comando "xargs" serve per eseguire un certo comando su una lista di argomenti in input. È molto utile in combinazione con "find". Esempio:

...

Opzioni principali di find:

4 Input ed output

Ogni programma ha a disposizione di default tre canali di comunicazione.

- L'input, ossia i dati in ingresso del programma, è costituito normalmente da quanto viene inserito da tastiera da parte dell'utente (standard input).
- L'output, ossia i dati in uscita, normalmente vengono stampati sul terminale (standard output).
- Infine c'è un ulteriore flusso di dati in uscita, lo standard error che, sebbene normalmente venga stampato a terminale, è distinto dallo standard output.

4.1 Redirezione dell'input/output e degli errori

Lanciando un qualsiasi comando è sempre possibile ridirigere il suo input od output da e verso un file. Analizziamo i seguenti comandi:

```
$ ls > elenco_file.txt
$ ls >> elenco_file1.txt
$ echo "ciao a tutti" > file.txt
$ echo "da pippo" >> file.txt
```

Il primo fa in modo che l'elenco dei file della directory corrente venga salvato nel file `"elenco_file.txt"`; se il file esiste verrà sovrascritto, senza chiedere conferma (per cui attenzione nell'utilizzo di questi comandi!). Il secondo comando, analogamente, manda l'output al file specificato, ma se questo esiste, anziché sovrascriverlo, attacca l'output (*append*) alla fine del file. Gli ultimi due comandi fanno sì che il file `"file.txt"` venga prima (eventualmente) sovrascritto con la riga "ciao a tutti" e poi ampliato con la seconda.

Se si vuole invece ridirigere lo standard input di un programma, si hanno a disposizione le tre seguenti modalità:

```
$ cat < file.txt
ciao a tutti
da pippo
$ cat <<EOF
> ciao a tutti
> da pippo
> EOF
ciao a tutti
da pippo
$ cat <<<ciao
ciao
```

Nella prima, l'input richiesto dal comando `"cat"` viene letto dal file `"file.txt"`, con l'effetto di stampare il suo contenuto sul terminale. Nella seconda, l'input viene invece battuto tutto di seguito da tastiera, e concluso da una certa stringa

specificata (la riga conclusiva deve contenere esclusivamente quella data stringa). Infine nella terza l'input viene specificato direttamente sulla linea del comando (deve però essere contenuto su una sola riga, o al massimo spezzettato tramite backslash).

Osserviamo che è tranquillamente possibile unire i due reindirizzamenti, di input e di output:

```
$ cat < file.txt > copia.txt
```

in cui il comando cat prende l'input dal file `"file.txt"` e manda l'output (ossia il contenuto del primo file) in `"copia.txt"`; il risultato finale è che viene creata una copia del file (esattamente come col comando `"cp"`).

Infine si può ridirigere anche lo standard error:

```
$ ls file.txt > output.txt 2> error.txt
```

Esistono altre combinazioni che permettono di duplicare un canale, reindirizzandolo verso un altro (tipicamente lo standard error verso lo standard output o viceversa). All'interno di uno script, possiamo sfruttare questo per stampare messaggi di errore:

```
$ echo "Messaggio di errore" >&2
```

Nel caso di più reindirizzamenti, bisogna fare attenzione all'ordine con cui li si fa. Ad esempio:

```
$ ./script.sh > error.txt 2>&1
```

il quale reindirizza sia l'error che l'output verso il file `"error.txt"`, mentre il contrario non funzionerebbe.

Nei sistemi di tipo Unix, inoltre, questi tre canali sono rappresentati da tre file speciali: `"/dev/stdin"`, `"/dev/stdout"` e `"/dev/stderr"`.

In realtà è possibile avere ben più di tre descrittori di file:

```
$ exec 3> out.txt
$ echo ciao
ciao
$ echo ciao >&3
$ cat out.txt
ciao
```

4.2 File speciali

Possiamo riferirci "fisicamente" a tutti i canali di input/output descritti prima con le varie device `"/dev/fd/n"`, dove n di default è 0 per lo standard input, 1 per lo standard output e 2 per lo standard error.

Esistono altri file *speciali* che possono tornare utili; tra questi:

- File speciali di Input:

  /dev/random se letto fornisce una serie di caratteri (bytes) generati in modo pseudo-casuale

  /dev/urandom come il precedente, ma restituisce sempre tanti byte quanti ne sono stati chiesti

  /dev/zero se letto fornisce una serie di caratteri `"NULL"` (codice Ascii 0)

- File speciali di Output:

  /dev/null si comporta come un pozzo senza fondo (utile per eliminare una serie di errori o avvertimenti inutili creati da certi programmi)

  /dev/full si comporta come un dispositivo (disco) sempre pieno, per cui qualsiasi scrittura su di esso ritorna un errore

  /dev/dsp rappresenta la scheda audio (reindirizzandogli un file `"WAV"` opportunamente costruito, lo si dovrebbe sentire)

  /dev/lp0 è la (prima) stampante

Esempi di utilizzo:

```
$ firefox 2> /dev/null >&2 &
[1] nnnn
$ dd if=/dev/urandom bs=512 count=1 of=file.dat
entrati 1+0 record
usciti 1+0 record
```

Il primo comando reindirizza lo standard error verso `"dev/null"`, poi duplica lo standard output reindirizzandolo verso lo standard error (e dunque sempre verso `"/dev/null"`), col risultato netto di non avere nessun output né su terminale né su disco (ottimo per quei programmi che hanno l'abitudine di stampare un mucchio di avvertimenti e/o errori non fondamentali per noi). Risultato analogo poteva essere ottenuto con il comando `"firefox > /dev/null 2>&1"`, oppure semplicemente con `"firefox > /dev/null 2> /dev/null"`.

Il secondo usa il comando `"dd"` (di cui vedremo alcune applicazioni più avanti) per leggere dal file speciale `"/dev/urandom"`, con l'effetto di riempire il file `"file.dat"` con byte casuali.

Un altro modo per ridirigere l'output di un comando, verso l'input di un altro, è creare un file speciale, detto *pipe*:

```
$ mkfifo file_io
$ cat file.txt > file_io &
[1] .....
$ cat < file_io
...
[1]+ Done .....
```

Infine il comando "mknod" serve per creare particolari file a caratteri o a blocchi, che di solito sono strettamenti correlati con il sistema operativo, ossia con il kernel.[25]
Esempi:

```
$ mknod -m 666 /dev/null c 1 3
$ mknod -m 666 /dev/zero c 1 5
$ mknod -m 666 /dev/full c 1 7
$ mknod -m 644 /dev/random c 1 8
$ mknod -m 644 /dev/urandom c 1 9
$ mknod -m 666 /dev/dsp c 14 3
```

Se avete installato sul vostro sistema la documentazione correlata al kernel, potrete trovare queste informazioni nel file "/usr/src/linux/Documentation/devices.txt" o con nome e percorso più o meno simili, a seconda della distribuzione.
blockdev

4.3 Manipolazione input/output da file

Innanzitutto i modi più semplici per creare un nuovo file di testo sono i seguenti:

```
$ echo -e "Testo\n...\nPenultima riga\nFine." > 1.txt
$ echo "" > 2.txt
$ echo -n > 3.txt
$ > 4.txt
$ ls -l ?.txt
-rw-r--r--    1 user    group       31 nov 25 16:20 1.txt
-rw-r--r--    1 user    group        1 nov 25 16:20 2.txt
-rw-r--r--    1 user    group        0 nov 25 16:20 3.txt
-rw-r--r--    1 user    group        0 nov 25 16:20 4.txt
```

Notiamo che il secondo file non è vuoto come ci si potrebbe aspettare: il comando "echo" infatti stampa comunque un carattere di a capo ("\n") nel file. Per avere un file vuoto bisogna usare l'opzione "-n", oppure scrivere semplicemente " > 4.txt", che manda un output nullo (poiché nessun comando lo precede) al file,[26] od infine sfruttare il comando "touch". Questo comando normalmente aggiorna

[25] il carattere occulto del funzionamento di questi dispositivi è sottolineato dal fatto che i permessi, dovendo essere per lo più di lettura/scrittura per tutti, sono rappresentati dal numero 666. Non c'è altro da aggiungere!

[26] da notare che se il file esiste, esso viene sovrascritto senza chiedere conferma; su Bash e su Linux in generale, bisogna sempre fare attenzione ai comandi che si scrivono.

l'orario di ultima modifica di un file (lo *tocca*, appunto) ma ha un interessante effetto collaterale: se gli si fornisce il nome di un file non esistente, ne crea uno con quel nome ma vuoto.

```
$ ls -l empty.txt
ls: empty.txt: No such file or directory
$ touch empty.txt
$ ls -l empty.txt
-rw-r--r--    1 user    group        0 nov 25 16:20 empty.txt
```

L'utilizzo tipico di "touch" è, come vedremo, quello di sfruttare i suoi effetti per forzare la ricompilazione di un qualche software, accompagnato dall'opportuno "Makefile" (vedi sez. B). Ora vediamo alcuni strumenti utili per visualizzare o manipolare il contenuto dei file:

colrm: rimuove le colonne specificate.

column:

csplit: spezza un file nei punti corrispondenti ad un certo modello.

cut: stampa le sezioni specificate (ad esempio l'ennesima colonna o l'ennesimo carattere di una riga) da un file o dallo standard input:

```
$ echo -e "primo\tsecondo"
primo   secondo
$ echo -e "primo\tsecondo" | cut -f 2
secondo
$ echo "parola_lunga" | cut -c 3-6
rola
```

fmt: formatta l'input suddividendolo (se può) in righe di pari lunghezza, ma senza spezzare le parole.

fold: suddivide le righe in input, in modo che vengano spezzate esattamente ad una certa lunghezza massima (pari di default a 80 caratteri).

head: mostra le prime n righe o i primi n caratteri di un file.

```
$ cat file1.txt
ciao a tutti
bla bla bla
continua ...
fine del file.
```

```
$ head -n 2 file1.txt
ciao a tutti
bla bla bla
$ head -c 10 file1.txt
ciao a tut
```

join: stampa l'output di due diversi file, ma unendo le righe con delle chiavi in comune. Queste chiavi tipicamente sono delle etichette numeriche ad inizio riga in stile Basic. Le etichette devono già essere ordinate. Ad esempio:

Presi due file:

```
$ cat 1.dat
100 farina
200 uova
300 latte
```

```
$ cat 2.dat
100 500 gr
200 4
300 1 l
```

```
$ join 1.dat 2.dat
100 farina 500 gr
200 uova 4
300 latte 1 l
```

nl: stampa le righe di un file numerandole progressivamente:

```
$ nl file1.txt
   1 ciao a tutti
   2 bla bla bla
   3 continua ...
   4 fine del file.
```

paste: stampa le righe di diversi file affiancandole su diverse colonne.

pr: formatta l'input suddividendolo in pagine, pronte per la stampa su carta.

rev: rovescia ogni riga di un file

```
$ rev file1.txt
ittut a oaic
alb alb alb
... aunitnoc
.elif led enif
```

sort: ordina alfabeticamente le righe in input.

```
$ sort file1.txt
```

split: spezza il file in input in n parti di lunghezza fissa; per riconcatenare si può usare `"cat"`.

tac:

tail: mostra le ultime n righe di un file (con sintassi analoga al comando `"head"`).

```
$ tail -n 1 file1.txt
fine del file.
$ tail -n +2 file1.txt
bla bla bla
continua ...
fine del file.
```

tee: duplica l'input, copiandolo oltre che sull' output, su un altro file

tr: traduce un set di caratteri in un'altro (oppure li elimina); ad esempio:

wc: conta righe, parole e caratteri in input

```
$ wc file1.txt
   4     9    45 file1.txt
$ wc -l file1.txt
   4 file1.txt
```

può essere interessante l'opione `"-L"` che stampa la dimensione della riga più lunga.

Controllare se due file (di testo) sono uguali:

```
$ cmp file1.txt file2.txt
```

Se invece vogliamo vedere in dettaglio le differenze, riga per riga, tra i due file:

```
$ diff file1.txt file2.txt
```

L'output del comando `"diff"` può essere usato per creare delle patch, come si vedrà nella sez. 17.5. Può inoltre essere usato per confrontare il contenuto di due directory:

```
$ diff -r -P dir1/ dir2/
```

Mostra tutte le differenze per ogni file ...

4.4 Sed

Un tipico esempio dell'utilizzo di `"sed"`, è sostituire tutte le occorrenze di una certa parola con un'altra:

```
$ sed -e 's/vecchio/nuovo/g' <<EOF
> un vecchio vestito
> EOF
un nuovo vestito
```

4.5 Awk

Sulla sintassi di awk c'è ancora moltissimo da sapere, ma va oltre lo scopo di questo manuale. Per maggiori informazioni potete leggere [8].

5 Variabili

```
$ var="Ciao a tutti"
$ echo $var
Ciao a tutti
```

```
$ export var=pippo
$ echo "var=$var"
var=pippo
```

5.1 eval

5.2 Argomenti ed opzioni degli script

Come avrete già intuito da precedenti esempi, definiamo "argomenti" tutte le parole passate da linea di comando ad un certo programma, che può essere sia uno script Bash che un qualsiasi altro tipo di file eseguibile.

Nel caso degli script Bash essi vengono salvati nelle variabili ...; in Perl invece sono contenuti nell'array `"ARGV[]"`, mentre in C nell'analogo `"argv[]"`.

Come conviene trattarli?

Comandi utili:

shift

getopts

getopt

xargs

5.3 Calcoli matematici

```
$ i=0
$ i=$[i+1]
$ echo $i
1
```

Il comando bc permette di fare calcoli più sofisticati (anche con i reali), di usare funzioni matematiche e anche di definire nuove funzioni (vedi ad esempio la definizione di una funzione per il calcolo del fattoriale nella sez. 12.3).

5.4 Alcune variabili di configurazione

USER: indica il nome dell'utente; di solito viene impostata in `"/etc/profile"` al momento del login; vedi anche la sez 10.1, per maggiori dettagli sulle configurazioni (globale e personale).

HOME: la directory dell'utente (di solito impostata al momento del login)

PATH:

LD_ LIBRARY_ PATH:

MANPATH: indica la lista di percorsi in cui il comando `"man"` cerca le pagine di manuale.

INFOPATH: indica la lista di percorsi in cui il comando `"info"` cerca le pagine di documentazione.

LS_ COLORS: se impostata fa sì che il comando `"ls"` elenchi i file con colori diversi a seconda del tipo (per maggiori dettagli vedi la sez. 10.2).

GREP_ COLOR: se impostata, fa sì che il comando `"grep"` colori i termini che corrispondono alla ricerca effettuata (per un esempio vedi la sez. 10.2).

COLUMNS: questa variabile viene di solito settata dal terminale (come ad es. `"xterm"`) e ne indica il numero di colonne.

5.5 Variabili "speciali" di Bash

Le variabili elencate qui di seguito o sono settate automaticamente all'avvio di Bash, oppure vengono dichiarate (o modificate) in un qualche file di configurazione (globale o personale dell'utente, vedi la sez. 10.1) e vanno in ogni caso a modificare il comportamento della shell. Ciascuna di esse ha infatti un significato o uno scopo ben preciso:

PWD: indica la directory corrente (ossia il percorso completo di `"."`).

OLDPWD: indica la directory precedente, a cui si può tornare col comando `"cd -"`.

?: indica lo stato di uscita (rappresentato da un numero compreso tra 0 e 255) dell'ultimo comando lanciato. Convenzionalmente è zero se il comando ha avuto successo mentre è diverso da zero se c'è stato un errore. Alcuni codici di errore, convenzionalmente, hanno un significato particolare. Sarebbe dunque bene che gli i valori di uscita ritornati dai nostri script non coincidano proprio con quei valori, per non confondere le idee.

Ad esempio:

...

0, 1 . . . n: sono i parametri passati da linea di comando ad uno script (per un esempio del loro utilizzo vedi la sez. 12). Attenzione: per n > 9, bisogna racchiudere il numero tra parentesi graffe. Ad esempio: `${10}`.

#:

@:

$:

!:

IFS: definisce il separatore di campo interno (*Internal Field Separator*), ossia la lista di quei caratteri, che Bash interpreterà come separazione tra un argomento e l'altro per la funzione `"read"`. Il valore predefinito è *space/tab/newline*.

RANDOM: fornisce un numero casuale compreso tra 0 e 32767 ($2^{15} - 1$).

PS1: definisce il prompt di Bash.

PS2: Il suo valore di default è `">"`.

PS3:

PS4:

REPLY: variabile di default in cui viene salvato il valore letto dal comando `"read"`.

SECONDS:

TMOUT:

PPID: l'ID del processo del genitore della shell (vedi sez. 9).

PROMPT_COMMAND: vedi l'esempio dell'orologio sul terminale (10.2).

HISTSIZE:

HISTFILE:

HISTFILESIZE:

HISTCONTROL:

HISTIGNORE:

HISTCMD:

HISTTIMEFORMAT Esempio: HISTTIMEFORMAT=

HOSTFILE:

5.6 Array

Bash può definire anche degli array.

6 Condizioni, confronti e test

L'istruzione "if" permette di eseguire delle istruzioni solo se una certa condizione è verificata.

```
$ if condizione; then istruzione; istruzione; ...; fi
```

La condizione deve essere racchiusa tra parentesi quadre ("[ ]"). In realtà possiamo vedere che questo è un vero e proprio comando (esterno a Bash):

```
$ whereis [
[: /usr/bin/[ /usr/share/man/man1/[.1.bz2
$ ls -l /usr/bin/[
lrwxr-xr-x  1 root  root   4 ott 14 2007 /usr/bin/[ -> test
```

Dunque "[" non è che un altro modo di chiamare il comando "test" il quale verrà descritto fra poco.

6.1 If - then - else/elif - fi

Possono tornare utili anche le istruzioni "true" e "false", dall'ovvio significato:

```
$ if true; then echo vero; else echo falso; fi
vero
$ if false; then echo vero; else echo falso; fi
falso
```

6.2 Confronti numerici

6.3 Test su file

```
$ if [ -f file.txt ]; then echo "OK"; else echo "Errore"; fi
OK
$ if [ -r file.txt ]; then echo "OK"; else echo "Errore"; fi
OK
$ if [ -x file.txt ]; then echo "OK"; else echo "Errore"; fi
Errore
```

6.4 Case

```
$ ls | wc -l
14
$ case `ls | wc -l` in 1) echo "1 file";; 2) echo "2 file";;
> *) echo "vari file";; esac
vari file
```

Ma il suo utilizzo principale è negli script; ad esempio può essere utilizzato per controllare gli argomenti passati da linea di comando:

Script Indent-1

```
#!/bin/bash

PROGNAME="script.sh"
VERSION=0.1
i=""

while [ "$1" != "" ]
do
  case $1 in
  "-h")
        echo "Usage: $PROGNAME [-i|-v|-h]"
        echo "..."
        exit 0
        ;;
  "-v")
        echo "Version: $VERSION"
        shift
        ;;
  "-i")
        echo "Option: -i (indent text)"
        i="      "
        shift
        ;;
  *)
        echo "$1: Option not recognized!" >&2
        exit 1
        ;;
  esac
done

sed "s/^/$i/"
```

Come si può intuire questo script indenta o meno il testo in input di un certo numero di spazi, a seconda delle opzioni specificate da linea di comando.

Vedremo successivamente come funziona esattamente il ciclo creato dall'istruzione `"while"`. L'istruzione `"shift"` fa slittare di un numero verso il basso la numerazione degli argomenti passati allo script da terminale, cosicché `"$2"` diventa `"$1"` e così via.

6.5 Select

```
$ select a in "qui" "quo" "qua"; do
> echo "Hai scelto $a"
> done
1) qui
2) quo
3) qua
#? 1
```

```
Hai scelto qui
#? 2
Hai scelto quo
...
```

Il ciclo è però infinito; per uscirne bisogna mettere esplicitamente l'istruzione `"break"` prima di `"done"`, oppure premere "Ctrl+D" in input (End Of File). È inoltre possibile cambiare il prompt di default che ci propone select, modificando la variabile `"PS3"`.

Un esempio:

7 Cicli

Ci sono sostanzialmente due modi di creare un ciclo, con comandi Bash, ossia usando "for", oppure "while":

```
for variabile in lista
do
  istruzione
  istruzione
  ....
done
```

```
while condizione
do
  istruzione
  istruzione
  ....
done
```

Come si può notare le istruzioni del ciclo devono essere racchiuse dalle istruzioni "do" e "done". Se si vuole scrivere un comando su un'unica riga, bisogna utilizzare i ";", come nei seguenti esempi:

```
$ for variabile in lista ; do istruzione; istruzione; ....; done
$ while condizione; do istruzione; istruzione; ....; done
```

Ci sono inoltre due istruzioni specifiche che hanno senso solo all'interno di un ciclo. La prima è "continue", la quale passa direttamente all'iterazione del ciclo successiva sempre che la condizione rimanga valida (nel caso di "while"), o che la lista non sia finita (nel caso di "for"), oppure ovviamente esce dal ciclo e passa all'istruzione successiva (quella dopo "done"). La seconda è "break", la quale esce immediatamente dal ciclo, senza ulteriori controlli, e passa all'istruzione successiva. Un altro modo per far terminare il ciclo è usare "return", che però funziona solo negli script, ed inoltre esce non dal ciclo ma dal programma (o dalla funzione in cui viene chiamato).

7.1 For - do - done

Adesso vediamo maggiormente in dettaglio, mediante alcuni esempi, l'uso dell'istruzione "for".

Da notare che posso ridiregere input ed output, non solo di un comando alla volta, ma anche di un ciclo, o una sequenza di comandi delimitata in qualche modo. Nel caso di "for":

```
$ for i in ``; do echo "$i: ciao"; done | tr
```

7.2 While - do - done

La condizione ...

Come esempio, riprendiamo lo script visto nella sez. 6.4, migliorandolo ulteriormente:

Script Indent-2

```
1 #!/bin/bash
2
3 PROGNAME="indent"
```

```
VERSION=0.2
i=""

while [ "$1" != "" ]; do
  case $1 in
  "-h")
        echo "$PROGNAME - Version $VERSION"
        echo "Usage: $PROGNAME [-i|-h]"
        echo "  -i <num>: indent text with <num> spaces"
        echo "  -h      : print this help message"
        exit 0
        ;;
  "-t")
  "-i")
        shift
        if [ "$1" == "" ]; then
          echo "Error." >&2
          exit 1
        fi
        if [ "$1" -gt 10 ] || [ "$1" -lt 1 ]; then
          echo "Error." >&2
          exit 1
        fi
        for j in `seq 1 $1`; do
          i="${i} "
        done
        shift
        ;;
  *)
        echo "Error: Option $1 not recognized!" >&2
        exit 1
        ;;
  esac
done

sed "s/^/$i/"
```

8 Alias e funzioni

Gli alias possono essere creati al volo da terminale, ma ovviamente non verranno salvati per sessioni future. Alcuni degli alias più comunemente usati sono:

```
$ alias rm='rm -i'
$ alias ls='ls --color'
```

Il primo fa sì che venga sempre chiesta conferma quando si cerca di cancellare un file, mentre il secondo fa sì che l'output del comando ls sia "colorato" in base alle impostazioni della variabile "LS_COLORS".

Funzioni ricorsive:

Fork Bomb:

`bomb() { bomb | bomb & }; bomb`

Per evitare problemi di sicurezza (attacchi DOS) configurare opportunamente il file "/etc/security/limits.conf", ad esempio aggiungendo una riga del tipo:

```
@users  hard    nproc   100
```

la quale setta un massimo di 100 processi per utente.

`declare -f` elenca le funzioni definite.

`unset -f`

Le definizioni di alias e funzioni vengono inserite in uno dei file di configurazione letti da Bash al suo avvio, tipicamente "/etc/bashrc" per quelle globali e "~/.bashrc" per il singolo utente.

9 Processi

Processi in foreground e in background:

Elenco processi attivi messi in background:

È possibile far dimenticare alla shell di aver lanciato un certo job, utilizzando (almeno nel caso di Bash) il comando interno `"disown"`.

Se si vuole eseguire un programma in background, per potersi poi scollegare dal sistema e lasciare il programma in esecuzione, si può dare un comando del genere:

```
$ nohup comando > out.txt 2> err.txt &
```

Questo ci assicura che:

1. il processo non è *figlio* della shell attuale, per cui non viene chiuso al termine di questa;
2. nemmeno i canali di comunicazione in output verranno chiusi, poiché sono reindirizzati verso dei file; questo risulta molto utile se successivamente abbiamo bisogno di rintracciare eventuali errori. Se invece l'output non ci dovesse interessare per niente, allora ci conviene reindirizzare entrambi verso `"dev/null"` (al riguardo sono già stati fatti degli esempi nella sez. 4.1).

Elenco dei processi lanciati dalla shell corrente: `"ps"`.

Elenco dei processi dell'utente: `"ps x"`.

Elenco dei processi di tutti gli utenti: `"ps aux"`.

Struttura ad albero dei processi:

Priorità di un processo:

Monitoraggio continuo dei processi: `"top"` (mostra varie informazioni, tra cui nome utente, percentuale carico sistema e memoria occupata, priorità).

Ogni processo può ricevere dei segnali; essi sono inviati dal comando `"kill"`. È possibile visualizzarne un elenco col comando:

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      32) SIGRTMIN    33) SIGRTMIN+1
...
```

Qui di seguito elenco i segnali più frequentemente usati e il loro significato (attenzione però, ogni singola applicazione può trattare alcuni segnali in modo diverso; quelli che non possono essere cambiati sono il 9 e il 19).

1/HUP (HANGUP) fa sì che il processo venga ripreso dopo essere stato bloccato

2/INT (INTERRUPT) fa terminare il processo, prima cercando di interromperlo (come con `"-15"`), e poi, se non ha successo, facendolo uscire forzatamente.

3/QUIT (QUIT)

6/ABRT (ABORT) fa uscire forzatamente il processo (come con `"-9"`) e genera, se possibile, un *core dump*.

9/KILL (KILL) indica che quel processo deve essere terminato immediatamente. Questo segnale non è bloccabile (a parte la mancanza dei privilegi necessari: non si possono uccidere applicazione di altri utenti se non si è root), in modo da consentire che un singolo processo che rallenta il sistema possa sempre venire fermato.

15/TERM (TERMINATION) è il segnale di default; cerca di far terminare il processo, ma non sempre può farlo (ad esempio quando questo è bloccato per qualche motivo).

18/CONT (CONTINUE) indica che quel processo, se è stato bloccato (ad esempio con `"kill -19"`, oppure con Ctrl+Z da tastiera) deve riprendere.

19/STOP (STOP) il processo deve essere bloccato; anche questo segnale non è bloccabile, ma attenzione: di default il processo non esce ma rimane sospeso in attesa di successivi segnali.

Come esempio possiamo utilizzare `"kill"` per verificare l'esistenza di un processo:

```
$ kill -0 $$
$ echo $?
0
$ echo -n &
[1] 4614
$ kill -0 $!
bash: kill: (4296) - No such process
$ echo $?
1
```

Lo stesso si può fare con `"killall"`, specificando però il nome del processo, col vantaggio che non serve conoscerne il PID, ma con lo svantaggio di non poter distinguere tra più processi con lo stesso nome.
Supponiamo ad esempio di avere varie shell aperte:

```
$ ps x | grep bash
 1792 pts/0    S      0:00 bash
 2102 pts/1    S      0:00 bash
 3270 pts/2    S      0:00 bash
 5978 pts/0    R      0:00 grep bash
$ killall -0 bash
$ killall -19 bash
```

A questo punto ci ritroviamo con tutte le shell aperte bloccate, compresa quella da cui abbiamo lanciato il comando. Per sbloccarle, aprite un'altra shell (un altro terminale se siete in modalità grafica o un'altra console in modalità non grafica) e lanciate il comando `"killall -18 bash"`.

Similmente il comando `"skill"` invia un segnale ad un gruppo di processi, specificandone il PID, il nome, l'utente, o il terminale (*tty*). Una sua opzione interessante è `"-n"`, che elenca semplicemente i segnali corrispondenti al modello richiesto. Ad esempio:

```
$ skill -n bash
1795
2926
9274
```

```
$ skill -v bash
136,0   user   1795 bash
136,1   user   2926 bash
136,2   user   9274 bash
```

Il primo comando elenca i processi corrispondenti a tutte le bash attive. Il secondo mostra anche altre informazioni, tra cui l'utente, il nome del comando ed il terminale a cui appartiene.

Se voglio specificare un certo user, devo usare l'opzione `"-u"`. Osserviamo i seguenti esempi:

```
$ skill -n xterm -u user
1749
2912
9235
```

```
$ pidof xterm
1749 2912 9235
```

Il primo elenca tutti i processi appartenenti all'utente `"user"` ed associati al comando `"xterm"`. Il secondo, sfrutta il comando `"pidof"`, il quale a sua volta elenca i processi corrispondenti ad un certo programma.

Il comando `"trap"` permette allo script di stabilire come procedere quando il processo riceve un segnale dall'esterno (tipicamente mediante il comando `"kill"`).

Il comando `"strace"` ...

Il comando `"fuser"` ...

Il comando `"truss"` consente di monitorare tutte le chiamate a sistema effettuate da un processo. Per ovvie ragioni di sicurezza, solo l'amministratore può seguire processi di altri utenti.

10 File di configurazione:

10.1 Configurazione globale

File /etc/profile

```
# /etc/profile -*- Mode: shell-script -*-
# (c) MandrakeSoft, Chmouel Boudjnah <chmouel@mandrakesoft.com>

loginsh=1

# Users generally won't see annoyng core files
[ "$UID" = "0" ] && ulimit -S -c 1000000 > /dev/null 2>&1

if ! echo ${PATH} |grep -q /usr/X11R6/bin ; then
    PATH="$PATH:/usr/X11R6/bin"
fi

if [ "$UID" -ge 500 ] && ! echo ${PATH} |grep -q /usr/games ; then
    export PATH=$PATH:/usr/games
fi

umask 022

USER=`id -un`
LOGNAME=`logname`
MAIL="/var/spool/mail/$USER"
HISTCONTROL=ignoredups
HOSTNAME=`/bin/hostname`
HISTSIZE=1000

if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ]; then
    INPUTRC=/etc/inputrc
fi

# some old programs still use it (eg: "man"), and it is also
# required for level1 compliance for LI18NUX2000
NLSPATH=/usr/share/locale/%l/%N

export PATH PS1 USER LOGNAME MAIL HOSTNAME INPUTRC NLSPATH
export HISTCONTROL HISTSIZE
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig/

for i in /etc/profile.d/*.sh ; do
        if [ -x $i ]; then
                . $i
        fi
done

unset i
```

File /etc/bashrc

```
# /etc/bashrc

# System wide functions and aliases
```

```
# Environment stuff goes in /etc/profile

# by default, we want this to get set.
# Even for non-interactive, non-login shells.
if [ `id -gn` = `id -un` -a `id -u` -gt 99 ]; then
        umask 002
else
        umask 022
fi

# are we an interactive shell?
if [ "$PS1" ]; then
    case $TERM in
        xterm*)
            PROMPT_COMMAND='echo -ne "\033]0;${USER}@"\
            "${HOSTNAME}:${PWD}\007"'
            ;;
        *)
            ;;
    esac
    [ "$PS1" = "\\s-\\v\\$ " ] && PS1="[\u@\h \W]\\$ "

    if [ -z "$loginsh" ]; then # We're not a login shell
        for i in /etc/profile.d/*.sh; do
            if [ -x $i ]; then
                . $i
            fi
        done
    fi
fi

unset loginsh
```

10.2 Configurazione personalizzata del singolo utente

Ogni utente nella propria "HOME" dovrebbe avere i seguenti file di configurazione: ".bash_profile", ".bashrc" e ".bash_logout".

Riporto qui sotto un possibile esempio di configurazione per un generico utente:

File .bash_profile

```
# Get the aliases and functions

if [ -r /etc/profile ]; then
  . /etc/profile
fi

if [ "$HOME" = "" ]; then
  export HOME=~
fi
export PATH=$PATH:~/bin/
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/lib/
export CFLAGS="$CFLAGS -I $HOME/include/"
```

File .bashrc

```
# Source global settings

if [ -r /etc/bashrc ]; then
   . /etc/bashrc
fi

eval `dircolors /etc/DIR_COLORS`

PS1="\[\033[1;34m\][\u\[\033[1;31m\]@\h "\
    "\[\033[1;34m\]\w]\$\[\033[0;0;0m\] "

alias rm="rm -i"
alias cp="cp -i"
alias mv="mv -i"

alias ls="ls --color"
alias grep="grep --colour"
umask 022
export GREP_COLOR="31"
```

File .bash_logout

```
# Source global settings

if [ -r /etc/logout ]; then
   . /etc/logout
fi

clear
fortune
```

Precedentemente è già stato spiegato l'utilizzo degli alias. Di norma questi vengono definiti nel file ".bashrc". Una configurazione tipica prevede degli alias per evitare di cancellare i file sovrascrivendoli (con i comandi "cp" e "mv") o per avere un'output colorato del comando "ls".

Volendo personalizzare i colori, bisogna impostare opportunamente la variabile "LS_COLORS"; solitamente per farlo si include nel file ".bashrc" il comando:

```
$ eval LS_COLORS=`dircolors ~/.coloursrc`
```

il quale va a leggere un file, in questo caso nella nostra "HOME", opportunamente formattato come nel seguente esempio:

File .coloursrc

```
# Elenco terminali
TERM linux
TERM console
TERM xterm

# Tipi di file
NORMAL 00       # global default
```

```
FILE   00       # normal file
DIR    01;34    # directory
LINK   01;36    # symbolic link
FIFO   33;40    # pipe
SOCK   01;35    # socket
BLK    01;33;40 # block device driver
CHR    01;33;40 # character device driver
ORPHAN 01;31;40 # symlink to nonexistent file
EXEC   01;32    # files with execute permission

# Elenco estensioni
.tar 01;31
.gz  01;31
.jpg 01;35

 ...
 ...
```

Ovviamente conviene copiare il file usato di default nella propria "HOME":

```
$ cp /etc/DIR_COLORS ~/.coloursrc
```

per poi modificarne solo le cose che ci interessano.

Col comando "alias" vengono elencati tutti gli alias correntemente impostati. Inoltre è possibile togliere un alias col comando unalias. Ad esempio se non si vuole più che "ls" sia lanciato con l'opzione "--color", basta fare:

```
$ unalias ls
```

Se però si vuole impedire a Bash di fare l'alias di un comando solo in una certa occasione, è più conveniente o racchiudere il nome del comando tra apici, o farlo precedere da un backslash: "\".

La variabile "PS1", come già visto descrive il messaggio del prompt; esso è configurabile, oltre che con un messaggio di testo personalizzato, con delle stringhe che vengono sostituite automaticamente da Bash al momento dell'esecuzione; ad esempio:

Il comando umask, come già detto, serve a fissare la maschera con cui vengono creati i nuovi file dalla shell.

La variabile "GREP_COLOR" fissa il colore della parte di output di grep che corrisponde al modello cercato.

Il comando di Bash shopt ...

Per concludere il nostro discorso su bashrc vediamo come configurare il completamento automatico che opera Bash sul nostro input quando premiamo "tab"...

Oltre a quelli già elencati, ci possono essere altri file (e directory) di configurazione nella nostra "HOME" (in realtà molti altri, ve ne potete rendere conto col comando "ls -a"); alcuni di questi vengono generati automaticamente al primo avvio di qualche programma, per cui la loro cancellazione accidentale non arreca un grave

danno per l'utente (e comunque esiste il backup per questo!); alcuni però contengono informazioni utili per la configurazione personalizzata scelta dall'utente, per cui prima di cancellarli è bene sapere quello che si sta facendo. Qui ne descriverò (brevemente) alcuni.

Il file ".inputrc" descrive come Bash deve gestire la lettura dell'input. Si può specificare un file diverso da quello di default (di solito "/etc/inputrc") nella variabile d'ambiente "INPUTRC".

I file ".xsession", ".xinitrc", ".Xresources", ".Xdefaults", ".Xauthority" e ".Xmodmap" sono file di configurazione di "X" (vedi la sez. 22).

Le directory ".vim", ".ssh", ".mozilla", ".gimp-*" contengono le configurazioni relative ai corrispondenti programmi. Attenzione ".mozilla" serve non solo a Mozilla ma anche a Firefox. Questa directory spesso col tempo tende a diventare piuttosto grossa, tuttavia vi conviene agire su di essa tramite mozilla e/o firefox, ad esempio cancellando la cache e le copie di backup dei bookmarks. Cancellando invece tutta la directory perdereste la vostra configurazione e tutti i bookmarks, oltre alla cronologia, etc. etc.

Le direcory ".kde" e ".gnome" (o simili) contengono ovviamente le configurazioni dei rispettivi window manager. Se non li usate potete anche cancellarle, poiché verrebbero comunque ricreate ad un nuovo avvio del rispettivo window manager, con le impostazioni di default; se invece li usate e avete personalizzato in vari modi sfondi, icone, colori, menu etc, non cancellatele o perderete tutta la vostra configurazione.

I file ".pinerc" e .muttrc, contengono la confurazione per i programmi di posta elettronica "pine" (o la sua nuova versione "alpine") e "mutt".

La directory "GNUstep" contiene la configurazione del Window Manager WindowMaker (comando "wmaker"). Può esservi utile sapere che la directory può anche essere spostata da un'altra parte, purché venga impostato correttamente il nuovo percorso nella variabile "GNUSTEP_USER_ROOT" inserendo l'opportuno comando nel file ".bash_profile".

In linea di principio potete cancellare quei file e quelle directory i cui nomi sono associati a programmi che siete sicuri non userete mai (magari li avete usati una sola volta per prova), o che addirittura avete ormai disinstallato.

Se non siete sicuri su un file, adottate questo trucco: non cancellatelo, ma spostatelo in qualche altra directory, poi uscite dalla sessione e riaccedete al sistema; se tutto funziona correttamente (e ne siete assolutamente certi), potete anche fidarvi di cancellarlo. Io comunque nel dubbio, lo conserverei ancora per un po', non si sa mai.

In ogni caso la maggior parte di essi non occupa molto spazio, per cui potete benissimo tenerli.

Infine analizziamo le funzioni definite dall'utente ...

Esempio: orologio

Esempio di funzione Bash

```
1 # codice da aggiungere al file ~/.bashrc
2
3 PROMPT_COMMAND=bashclock
4
```

```
bashclock(){
        x=`tput cols`
        x=$((x-3))
        tput cup 0 $x
        echo -en "\033[1;31m[]\033[0;0m"
        tput cup 0 0
}
```

11 Trucchi del mestiere

Questa sezione racchiude una serie di comandi con uno scopo ben preciso; tutti questi sono in linea di principio scrivibili su una sola riga di terminale. Per operazioni più complicate che necessitano di comandi più elaborati, vedere la prossima sezione che mostra alcuni esempi di script.

11.1 Operazioni con file

- Elencare solo le sotto-directory (anche quelle nascoste):

```
$ ls -d */ .*/
```

  oppure, ricorsivamente:

```
$ find . -type d
```

- Elencare gli n file più grossi presenti in una certa directory:

```
$ find . -printf "%s %h/%f\n" | sort -n | tail -n 5
```

  Se non lo si vuole fare ricorsivamente, bisogna specificare l'opzione di find: `"-maxdepth 1"`

- Mostrare le sotto-directory ordinandole a seconda dello spazio occupato:

```
$ du -k --max-depth=1 | sort -n
```

- Cancellare un certo file di configurazione, di cui conosciamo il nome ma non sempre il percorso completo; esempio tipico è un file che blocca un nuovo avvio di firefox quando questo viene chiuso non correttamente:

```
$ find . -name .parentlock -exec rm -f {} \;
```

- Rinominare una serie di file.

  Supponiamo ad esempio di voler rinominare i file `".jpeg"` in `".jpg"`:

```
$ for file in *.jpeg; do \
> new=${file%.jpeg}.jpg; \
> mv $file $new; \
> done
```

  Così com'è il comando `"mv"` potrebbe dare qualche problema con dei file con nomi particolari. Dovrebbe andare (quasi sempre) tutto bene sostituendo il comando in questione con `"mv -- "$file" "$new""`.

  In ogni caso, lo stesso risultato poteva essere ottenuto in modo molto semplice utilizzando il comando `"rename"`:

```
$ rename .TXT .txt *.TXT
```

11.2 Caratteri speciali

- Metodo alternativo per cancellare la schermata del terminale (senza usare l'apposito comando "clear"):

```
$ echo -en "\014" | more
```

- Conto alla rovescia, sovrascrivendo i numeri su una singola linea:

```
$ echo -n "10 "; for i in `seq 9 -1 0`; do \
> sleep 1; echo -en "\r $i "; done; echo
```

- Riempire un file con n caratteri uguali; ad esempio 100 caratteri 'a':

```
$ dd if=/dev/zero bs=100 count=1 | tr "\0" a > file.txt
```

11.3 Web e servizi di rete

- Download di una serie di file dal Web. Nel caso specifico, supponiamo di voler scaricare una serie di pagine html caratterizzate da un indirizzo con un numero incrementale; ad esempio quelle nell'intervallo:
"www.url.com/page1.html" – "www.url.com/page99.html"

```
$ for i in `seq 1 99`; do \
> wget www.url.com/page${i}.html; done
```

  Se i primi numeri avessero uno zero davanti, basterebbe modificare il comando in questo modo:

```
$ for i in `seq -w 1 99`; do \
> wget www.url.com/page${i}.html; done
```

  Tuttavia non capita spesso che le pagine che ci interessa salvare seguano uno schema numerico così preciso; spesso conviene allora sfruttare l'opzione "-r" di "wget" che scarica ricorsivamente le pagine, seguendo i link all'interno delle pagine. Bisogna però fare attenzione, perché rischiamo facilmente di scaricare molti file che non ci interessano, e magari di tralasciare altri che invece vorremmo. Il seguente comando, sfrutta alcune opzioni di questo comando, allo scopo di scaricare unicamente file di tipo html, jpeg e gif, rimanendo sempre all'interno di un certo dominio e fino ad un livello massimo di ricorsività (il numero di passaggi da seguire per arrivarci; se è 0 viene considerata la sola pagina iniziale) pari a 2:

```
$ url="dominio.it/pagina.html"
$ wget -r -l2 -A.html,.jpg,.gif -Ddominio.it $url
```

- Stampa file locale su computer remoto (ossia su un computer su cui è possibile accedere tramite ssh, e sui cui è attivo il servizio della coda di stampa, "lpd" o "cups"):

```
$ cat nome_file_locale.pdf | ssh user@host lpr
```

oppure, al contrario, se si vuole stampare un file presente su un computer remoto, in locale:

```
$ ssh user@host cat /path/nome_file_remoto.pdf | lpr
```

- Si è logati su un server remoto, ma non si sa che distribuzione/s.o vi siano installati. Il comando:

```
$ uname --sysname --release
```

fornisce nome e versione del kernel. Ma per sapere la distribuzione installata? A volte questa informazione è contenuta nel messaggio di login, ossia nei file `"/etc/issue"` e `"/etc/issue.net"`, ma non sempre è così. In linea generale, quasi tutte le principali distribuzioni possiedono un file di nome /etc/release o /etc/*nomedistro*-release con le informazioni cercate. Dunque basta dare un comando del tipo:

```
$ cat /etc/*release
```

12 Alcuni esempi di script in Bash

Per approfondire questo tema si consiglia vivamente la lettura del libro "Advanced Bash-Scripting Guide" [6]. Qui di seguito sono riportati alcuni esempi, scelti non tanto per la loro effettiva utilità, quanto piuttosto per offrire una panoramica delle possibilità di Bash.

Bisogna sempre ricordarsi però che per alcune situazioni (calcolo numerico pesante, algoritmi un po' più complessi dei soliti cicli for-while, presenza di una grossa mole di dati) Bash, nonostante possa riuscire a portare a termine il compito assegnatole, non è sicuramente il mezzo più adatto. Vedremo nella prossima sezione alcuni esempi nei quali uno strumento decisamente più efficiente è rappresentato da uno script in Perl.

12.1 Rinominare una serie di file

Supponiamo di voler rinominare tutti i file che hanno estensione `".TXT"` in `".txt"` (ricordiamoci che nei filesystems ext2/ext3 e simili i nomi dei file sono case-sensitive!).

Script Rinomina file

```
#!/bin/bash

for file in *.TXT
do
  new=`basename $file TXT`txt
  if [ ! -f "$new" ]
  then
    echo "$file -> $new"
    mv -- "$file" "$new"
  else
    echo "Attenzione: non posso rinominare $file in $new"
    echo "  Il file esiste!"
  fi
done
```

Osservazione: Abbiamo visto nella sez. precedente che per farlo bastava un'unica riga da terminale, grazie al comando `"rename"`, tuttavia il procedimento tramite Bash-script può essere generalizzato per fare un mucchio di cose diverse e più complicate. Ad esempio possiamo voler scrivere esplicitamente quali file vengono rinominati, o quali non possono esserlo perché verrebbe sovrascritto un altro file.

12.2 Flip

Esiste già un comando del genere che funziona molto bene; cercate "flip Unix Windows" su `http://www.google.it`.

12.3 Fattoriale

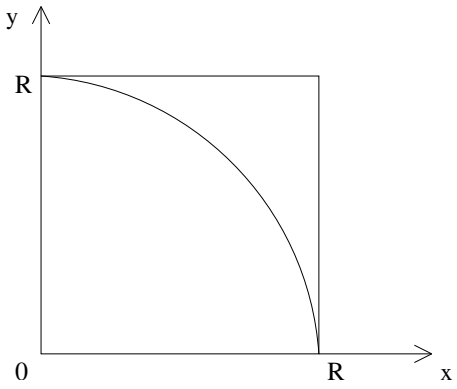
Il seguente script calcola il fattoriale di un numero (passato come argomento da linea di comando), facendo uso di "bc".

Script Fattoriale

```
#!/bin/bash

max=50

declare -i num
num=$1
# Attenzione se $1 non e` un numero $num diventa 0

if [ "$1" != "$num" ] || [ $num -lt 0 ] ; then
  echo "Errore!"
  echo "L'argomento deve essere un numero intero positivo."
  exit 2
elif [ $num -gt $max ]; then
  echo "Errore!"
  echo "Valore Massimo per l'argomento: $max."
  exit 2
fi

bc << EOF
define f (x) {
  if (x <= 1) return (1);
  return (f(x-1) * x);
}
f($num)
EOF
```

12.4 Calcolo del pi-greco con un metodo di tipo Montecarlo

Il seguente metodo cerca di calcolare "brutalmente" il valore di π, sfruttando la generazione di numeri pseudo-casuali (random), secondo il seguente ragionamento: associo ad una coppia di numeri un punto nel quadrato compreso in $[0, R] \times [0, R]$; poiché i numeri vengono generati in maniera apparentemente casuale, con una distribuzione pressoché uniforme nell'intervallo $[0, R]$ (ergodica), posso supporre che la frazione di punti nel quadrato che cadrà anche all'interno del cerchio di raggio R centrato su un vertice del quadrato, sarà pari al rapporto tra le due aree, come visualizzato nella seguente figura:



$$x, y \in [0, R]$$
$$0 \leq x^2 + y^2 \leq R^2$$
$$A_Q = R^2$$
$$A_C = \frac{\pi R^2}{4}$$

Dunque si ottiene:

$$\pi = 4 * \frac{A_Q}{A_C} \simeq 4 * \frac{N_{\text{in}}}{N_{\text{tot}}}$$

Un esempio di codice che concretizza il calcolo è il seguente:

Script Pi-Greco

```
#!/bin/bash
# Calcolo del Pi-Greco con metodo
# di tipo Montecarlo

def=5        # Numero di passi di default (100000)
max=10       # Numero di passi massimo (1E10):
randm=32767 # Massimo valore di RANDOM:

r2=$((randm*randm))
declare -i n tot i in x y z

n=$1
if [ $n -le 0 ] || [ $n -gt $max ]; then
        n=$def
fi

tot=1; i=0
while [ $i -lt $n ]; do
        tot=$((tot*10)); i=$((i+1))
done

echo "Numero di passi: $tot"
in=0; i=1
while [ $i -le $tot ]; do
        x=$RANDOM; y=$RANDOM; z=$((x*x+y*y))
        if [ $z -lt $r2 ]; then
                in=$((in+1))
        fi
        i=$((i+1))
done

```

```
p=$(bc <<< "scale=$n; 4*$in/$tot")
echo "Pi = $p"
```

Oppure in una versione più efficiente che fa uso di "awk".

Script Pi-Greco 2

```
#!/bin/bash
# Calcolo del Pi-Greco - Versione 2
# (fa utilizzo di awk)

def=5   # Numero di passi di default (100000)
max=10  # Numero di passi massimo (1E10):

declare -i n tot i

n=$1
if [ $n -le 0 ] || [ $n -gt $max ]; then
        n=$def
fi

tot=`echo $n | awk '
{n=$1}
END {
  tot=1;
  for (i=0; i < n; i++) {
    tot=tot*10;
  }
  print tot;
}'`

echo $tot | awk '
{tot=$1}
END {
  print "Numero di passi: "tot;
  srand(); num=0;
  for (i=0; i < tot; i++) {
    x=rand(); y=rand();
    if (x*x+y*y < 1) {num++}
  }
  print 4*num/tot;
}'
```

Ovviamente Bash ed Awk non sono i linguaggi più adatti per compiere questo tipo di operazioni (e la generazione dei numeri random da parte di Bash non è sufficientemente ergodica per utilizzi seri); conviene sicuramente scrivere un programma in un linguaggio tipo Fortran, Pascal o C (e sfruttare librerie esterne come la *drand48* o altre), per poi compilarlo ed avere un eseguibile efficiente e veloce.

12.5 Portscanner

12.6 Network-Traffic

13 Alcuni esempi di script in Perl

13.1 Ricerca di file doppi

Un esempio di codice che può tornare utile: uno script che cerca in una directory, anche ricorsivamente, i file doppi. Ovviamente se i file venissero comparati uno a uno, in una directory con moltissimi file il tempo di esecuzione potrebbe diventare molto lungo. L'idea di questo script è di ordinare i file inizialmente in base alla dimensione in byte, e poi di andare a confrontare (tramite il comando esterno "cmp") tra loro solo i file della stessa grandezza.

Script Ricerca Doppioni

```
#!/usr/bin/perl

# Versione 0.5a
# Ultima modifica: 26/02/2009
# by mt

# Scopo: ricerca di file doppi
#
# Caratteristiche:
# - esclude directory e link simbolici dal confronto
# - non ha problemi con nomi di file con spazi
# - procede ricorsivamente nelle subdir solo su
#   esplicita richiesta (opzione -r)
# - considera i file vuoti solo su esplicita richiesta
#   (opzione -e)

$dir=".";
$rec="-maxdepth 1";
$i=0;
$only_no_empty=1;

sub my_help{
  print "Usage: $0 [-r/-e] <dir>\n";
  print "  Default dir is the current one (.)\n\n";
  print "  Options:\n";
  print "   -r : recursively\n";
  print "   -e : consider also empty file\n";
  print "   -h : print this message and exit\n";
}

# Start: Check options and arguments
for ($i = 0; $i <= $#ARGV; $i++) {
  if ($ARGV[$i] eq "--help" || $ARGV[$i] eq "-h") {
    &my_help;
    exit(0);
  }
  if ($ARGV[$i] eq "-r") {
    $rec = "";
```

```
39      next;
40    }
41    if ($ARGV[$i] eq "-e") {
42      $only_no_empty = 0;
43      next;
44    }
45    if (-d $ARGV[$i]){
46      $dir = $ARGV[$i];
47      last;
48    }
49    print "Error with the options!\n";
50    $my_help;
51    exit(1);
52  }
53
54  # elenco file e loro dimensioni
55  @file=`find $dir $rec -type f`;
56  $n = $#file;
57  $n++;
58  print "Number of files: $n\n\n";
59
60  for ($i = 0; $i < $n; $i++){
61    $indiceold[$i] = $i;
62    $chk[$i] = 1;
63    chop($file[$i]);
64    $size[$i]=(stat($file[$i]))[7];
65  }
66
67  # ordinamento, in base alla dimensione del file, di un
68  # vettore di indici anziche' degli array originali
69  @indice = sort {$size[$a] <=> $size[$b]} @indiceold;
70
71  # segna i file vuoti
72  if ($only_no_empty == 1){
73    for ($i = 0; $size[$indice[$i]] == 0; $i++){
74      $chk[$i] = 0;
75    }
76  }
77
78  # cerca file uguali
79  $i = 0;
80  while ($i < $n - 1) {
81    $s = $size[$indice[$i]];
82    for ($j = $i + 1; ($j1 = $indice[$j]) && $j <= $n &&
83            $size[$j1] == $s; $j++) {
84      for ($k = $i; $k < $j; $k++) {
85        $k1 = $indice[$k];
86        if ($chk[$k] == 1 &&
87              `cmp "$file[$k1]" "$file[$j1]"` eq ""){
88          print "$file[$k1]  ==  $file[$j1]\n";
```

```
89            $chk[$j] = 0;
90          }
91        }
92      }
93      $i = $j;
94    }
```

Questo script presenta inevitabilmente dei bug. Uno dei più facili da risolvere è il caso in cui i nomi dei file abbiano caratteri speciali, in particolare "$" e "`"; infatti l'istruzione che sfrutta il comando esterno "cmp" fallirà in quanto cercherà di interpretare (come in Bash) quei simboli. Per eliminare il problema basta aggiungere tra le righe 64-65 le seguenti istruzioni:

```
  $file[$i] =~ s/\`/\\\`/g;
  $file[$i] =~ s/\$/\\\$/g;
```

14 Script in altri linguaggi

Vedremo alcuni esempi di script per "gnuplot" nella sez. F.
python

14.1 Tcl/Tk

Portscanner

```
#! /usr/bin/tclsh
#Portscanner 0.1

if { "$argc" != 1 } {
  puts "Inserire l'host\n"
} else {
  set host $argv
  puts "Portscanner 0.1\n"
  puts "Verifichiamo il server $argv\n"
}

foreach p {7 21 22 23 25 43 79 80 110 111 115 119 143} {

  global host

  catch {
    if {[string compare [socket $host $p] "sock"] != "-1"} {
      switch $p {
          7 {puts " *** La porta echo    (  7) e` aperta" }
         21 {puts " *** La porta ftp     ( 21) e` aperta" }
         22 {puts " *** La porta ssh     ( 22) e` aperta" }
         23 {puts " *** La porta telnet  ( 23) e` aperta" }
         25 {puts " *** La porta smtp    ( 25) e` aperta" }
         43 {puts " *** La porta whois   ( 43) e` aperta" }
         79 {puts " *** La porta finger  ( 79) e` aperta" }
         80 {puts " *** La porta http    ( 80) e` aperta" }
        110 {puts " *** La porta pop3    (110) e` aperta" }
        111 {puts " *** La porta portmap (111) e` aperta" }
        115 {puts " *** La porta sftp    (115) e` aperta" }
        119 {puts " *** La porta nntp    (119) e` aperta" }
        143 {puts " *** La porta IMAP    (143) e` aperta" }
      }
    }
  }
}
puts " "
```

14.2 Expect

15 Linguaggi compilati

Compilare un file di codice sorgente in C: Il comando "gcc source.c", crea di default l'eseguibile "a.out". Ora per eseguirlo basta dunque fare: "./a.out". Se si vuole specificare il nome dell'eseguibile:

```
$ gcc -o nome source.c
```

A volte però sono necessarie altre opzioni. Le più comuni sono, l'inclusione di librerie (con "-lnomelib"), il path di ricerca per le librerie ("-L path"), il path di ricerca per i file da includere: ("-I path").

Osservazione: una volta che il file viene correttamente compilato, specificando però dei percorsi per le librerie diversi dai soliti "/usr/lib" e "/usr/local/lib", molto probabilmente all'esecuzione il programma non troverà la libreria dando un errore del tipo:

Per evitare questo bisogna impostare il percorso della libreria nella variabile "LD_LIBRARY_PATH", come nel seguente esempio:

```
$ gcc -I . -L /home/user/lib -lnomelibreria -o run source.c
$ ./run
Error ...
$ export LD_LIBRARY_PATH=/home/user/lib
$ ./run
...
```

Il comando "ld" unisce gli oggetti compilati per creare l'eseguibile finale (di solito se ne occupa il compilatore e noi non ce ne accorgiamo). Le cose che ci interessano sono il file di configurazione "/etc/ld.so.conf" e la variabile d'ambiente "LD_LIBRARY_PATH" che indicano a "ld" i percorsi dove cercare i file.

Se avete una directory che contiene delle librerie che usate solo voi, come utente, vi conviene impostare la "LD_LIBRARY_PATH" una volta per tutte nel vostro file ".bashrc" o ancora meglio nel ".bash_profile". L'alternativa è mettere una ???

L'output di "gcc", contenente errori e avvertimenti, può essere colorato per evidenziarne meglio i messaggi. Per attivare questa opzione (se non è già attiva) bisogna installare "colorgcc", mentre per personalizzare i colori bisogna creare/modificare il file di configurazione globale "/etc/colorgccrc" oppure, da utente, creare il proprio "$HOME/.colorgccrc".

Altri Linguaggi: bisogna distinguere tra i linguaggi che solitamente sono compilati, come il C, e quelli che vengono usati negli script, e "compilati" solo al momento dell'utilizzo da un'interprete di comandi, proprio come nel caso di Bash.

Al primo tipo appartengono il C++ (compilatore "g++"), il Fortran ("f77", "f90" o successivi a seconda della versione del linguaggio), il Pascal ("fpc"), il Basic ("???"), l'Haskell ("") ed altri ancora.

Al secondo tipo appartengono ad esempio il Perl, il Python, PHP, Ruby, e in parte anche il Qbasic, che però può anche essere compilato. In un certo senso vi fa parte anche il TEX/LATEX, che però non può essere considerato un vero linguaggio di programmazione, ma è piuttosto un misto tra questo ed un linguaggio di formattazione del testo, quali ad esempio i vari HTML, SGML, XML e simili.

Java: Infine il Java ...

16 Terminali

differenza tra console ed emulatori di terminale.

Menu di xterm Configurazione in .xresources (vedi ...)

vt100

reset

tput

tset

17 Amministrazione ed utilizzo pratico del sistema da linea di comando

Ovviamente molto dipende dalla distribuzione in uso e dai pacchetti installati; tuttavia la maggior parte di quello che seguirà vale quasi sempre.

17.1 Utenti, su, sudo

Possiamo sfruttare il file `"/etc/passwd"` per creare appositamente utenti finalizzati ad un unico scopo. Ovviamente questi utenti non dovranno realmente accedere al sistema, ma solo eseguire un determinato comando per poi uscire; per questo motivo al posto di `"/bin/bash"` metteremo il percorso completo del comando:
Una curiosità: il Sistema Operativo FreeBSD (molto simile a GNU/Linux) adotta la seguente tecnica: toor vs root.

password: per cambiare la password bisogna usare il comando `"passwd"`, il quale chiede prima la password attuale (per evitare che altri possano farlo se lasciamo una shell aperta) e poi chiede di ripetere per due volte quella nuova (per evitare possibili errori di battitura). Ovviamente l'amministratore può cambiare la password agli altri utenti senza bisogno di conoscere quella vecchia. A seconda della configurazione, è possibile che una password troppo semplice (ad esempio una parola di uso comune, come gatto, oppure con troppi caratteri uguali, come 00000) non venga accettata.

sudo: a volte potrebbe essere necessario permettere ad alcuni utenti di compiere alcune azioni che normalmente richiedono i privilegi di amministratore, senza voler però dare la password di root (per evitare che possano fare troppi danni). Per risolvere questo problema è nato il comando `"sudo"`.
Nota: la distribuzione Ubuntu nella sua configurazione di default sfrutta sudo, facendo in modo che non sia mai necessario accedere come amministratore (il login dell'utente root viene proprio disabilitato).

quota:

Aggiungere/Rimuovere utenti: usare i comandi (da root) `"useradd"` e `"userdel"`; analogamente per aggiungere/rimuovere gruppi, usare `"groupadd"` e `"groupdel"`. Per effettuare delle modifiche sulle informazioni basilari dell'utente (nome, home-directory, scadenza account, etc), usare il comando `"usermod"` (e analogamente `"groupmod"` per i gruppi).

17.2 Caratteristiche hardware del sistema

17.3 Data e ora del sistema

`"date"`

Tempo Universale: si riferisce al meridiano di Greenwich. Misurato come numero di secondi passati da una certa data ad oggi.

Tempo Unix: numero di secondi passati dalla mezzanotte del 1° gennaio 1970 ad oggi.

differenza tra orologio software ed hardware.
regolazione via internet.
"cal"
Programma minimale in C per estrarre le informazioni su data e ora dato il Tempo Unix.

Data e ora del sistema in C

```c
#include <stdio.h>
#include <time.h>

int main()
{
  time_t t, *p = NULL;
  struct tm tempo;

  t = time(p);
  tempo = *gmtime(&t);

  printf("contatore: %d\n", (int) t); /* secondi dal 1970 */

  printf("ora:    %d\n", tempo.tm_hour); /* 0 -- 23  */
  printf("min:    %d\n", tempo.tm_min);  /* 0 -- 59  */
  printf("sec:    %d\n", tempo.tm_sec);  /* 0 -- 59  */

  printf("giorno: %d\n", tempo.tm_mday); /* 1 -- 31  */
  printf("mese:   %d\n", tempo.tm_mon);  /* 0 -- 11  */
  printf("anno:   %d\n", tempo.tm_year); /* anni dal 1900 */

  return 0;
}
```

17.4 Avvio

Linux Boot Loaders: lilo, grub
Spegnimento del computer:
`$ shutdown -f +2`
dopo due minuti, oppure immediatamente:
`$ shutdown -h now`.
Per fermare uno spegnimento:
`$ shutdown -c`

17.5 Installazione di nuovo software

Il metodo più *classico* per installare un pacchetto, è compilarne il codice sorgente.

Un pacchetto classico viene distribuito nei formati ".tar.gz" o ".tar.bz2". Nel primo caso lo si scompatta con "tar -xvzf nomepacchetto.tar.gz", nel secondo con "tar -xvjf nomepacchetto.tar.gz".

All'interno della directory scompattata normalmente sono presenti i file "README", "INSTALL", "configure", ... o a volte direttamente il Makefile.

Il "configure" è uno script che crea ed auto-configura (impostando percorsi ed opzioni di compilazione) il "Makefile", il quale a sua volta conterrà le istruzioni necessarie per compilare il pacchetto. Spesso esso ha delle opzioni, che consentono di cambiare ad esempio la directory in cui il pacchetto verrà installato, o i percorsi in cui cercare le librerie. Di solito tutte le opzioni vengono elencate col comando: "./configure --help". Se tutto è andato bene (ossia il comando "./configure" non ha dato errori evidenti) vi basta ora lanciare "make" (vedi sez. B).

A volte (nei pacchetti più semplici) non c'è neppure il configure ma direttamente il "Makefile", che potrebbe però necessitare di essere configurato a mano. Leggete sempre il file "INSTALL" (o "README") per sapere cosa dovete fare esattamente.

Se anche la compilazione è andata a buon fine, ora non vi resta che installare veramente il pacchetto. Per farlo dovrete innanzitutto diventare l'amministratore (root) col comando su, e poi lanciare il comando "make install".

```
$ ls
package.tar.gz
$ tar -xvzf package.tar.gz
...
$ cd package
$ ls
... configure INSTALL README ...
$ ./configure --help
...
$ ./configure --eventuali_opzioni
...
$ make
...
$ su
Password:
$ make install
...
```

A volte su internet, vengono rese disponibili delle patch al codice sorgente di alcuni software. In questo caso bisogna innanzitutto applicare la patch al codice sorgente tramite il comando patch, e poi procedere con l'installazione come prima:

```
$ patch -p1 < patch.diff
$ ./configure
...
$ make
...
```

Compilare il codice a volte può diventare frustrante per la mancanza nel vostro sistema di vari file che vanno inclusi nel codice, e di cui non c'è invece bisogno se avete direttamente il file eseguibile (e le librerie).

Il problema dell'installare software copiando direttamente i file eseguibili consiste principalmente nel fatto che essi sono strettamente correlati con la vostra particolare architettura (hardware), e a volte con la vostra particolare distribuzione/configurazione.

Molte distribuzioni (Fedora/RedHat/Mandriva), sfruttano i pacchetti rpm, per cui se avete l'rpm giusto (adatto alla vostra architettura/distribuzione), basta lanciare (da root) il comando:

```
$ rpm -Uvh file_pacchetto.rpm
```

tuttavia a volte questo non funziona a causa delle dipendenze tra un pacchetto e l'altro.

Per ovviare a questo problema (e al problema di dover prima cercare il giusto pacchetto su internet) la maggior parte delle moderne distribuzioni ha implementato un proprio comando, il quale si occupa di cercare da solo tutte le dipendenze e di installare i pacchetti necessari, cercandoli su un certo sito autenticato ed eliminando così anche l'eventuale problema di sicurezza che può nascere dallo scaricare pacchetti rpm (contenenti file binari, quindi senza la possibilità di vedere il codice sorgente) di dubbia provenienza.

- Gentoo: `"emerge"`
- Fedora/Redhat: `"yum"`
- Mandriva:
- Ubuntu: `"apt-get"`
- Debian:
- Slackware:
- Suse/OpenSuse:
- Archlinux: `"pacman"`

Di solito questi stessi comandi hanno la possibilità di cercare su internet gli aggiornamenti disponibili (per tutto il software installato), facendo così in modo di avere un sistema costantemente aggiornato (e dunque con minori rischi per la sicurezza).

Nota: in realtà Gentoo, non ha un sistema di pacchettizzazione con i file binari, come gli `"rpm"` (o i `"deb"` della Debian) ma semplicemente compila tutti i pacchetti necessari a partire dal codice sorgente. Se da un lato questo rallenta (di molto) le operazioni di installazione/aggiornamento, dall'altro si ha (in teoria) un sistema molto efficiente poiché tutto il software installato è stato ottimizzato proprio per la nostra architettura/configurazione.

Per maggiori informazioni su ciascun singolo comando, si consiglia di leggere la rispettiva pagina di manuale (`"man nomecomando"`) oppure di cercare sul sito della propria distribuzione.

Oltre ai comandi citati, le maggiori distribuzioni adottano comunque delle interfacce grafiche per gestire l'installazione di nuovo software. Ubuntu, ad esempio, usa `"synaptic"`, mentre Fedora ...

Citiamo infine l'esistenza delle distribuzioni cosiddette "live" (ad esempio Knoppix, ma altre distribuzioni recenti tra quelle citate prima hanno anche la versione live), nelle quali il sistema operativo non viene realmente installato sull'hard-disk, ma viene avviato da un supporto esterno (di solito CD-Rom o DVD, ma è possibile anche da una pen-drive) e caricato interamente nella RAM. Questo offre l'opportunità, tra l'altro, di poter recuperare i propri dati in sistemi che non riescono pi‘ăd autoavviarsi normalmente (se ad esempio è stato corrotto il Master Boot Record o avete cancellato dei file fondamentali per l'avvio del sistema operativo). Ma sono anche utilizzabili da quegli utenti che desiderano provare Linux, ma non lo fanno per paura di causare danni o rischiare di cancellare i propri dati. Esiste, riguardo a questo, un'opzione all'avvio di Knoppix, che assicura che l'hard-disk non verrà assolutamente toccato in scrittura.

Per finire, analizziamo un problema solo in apparenza banale: come facciamo a sapere se un certo software o un certo file (che può essere un eseguibile, una libreria, documentazione, codice sorgente, un file di configurazione, etc) è già installato, e soprattutto, dove si trova? Possono risultare utili i seguenti comandi:

- Per vedere la lista dei pacchetti installati:

```
$ rpm -q -a
```

- Per vedere l'elenco dei file appartenenti a un certo pacchetto (installato):

```
$ rpm -q -l nome_pacchetto
```

- Per trovare il pacchetto a cui appartiene un certo file:

```
$ rpm -q -f /path/nomefile
```

Se non si trova niente mediante questi metodi, (probabilmente non avete installato il pacchetto con l'utility `"rpm"`) provare a seguire i suggerimenti della sezione 3.6.

17.6 Locale

ora/fuso local_messages

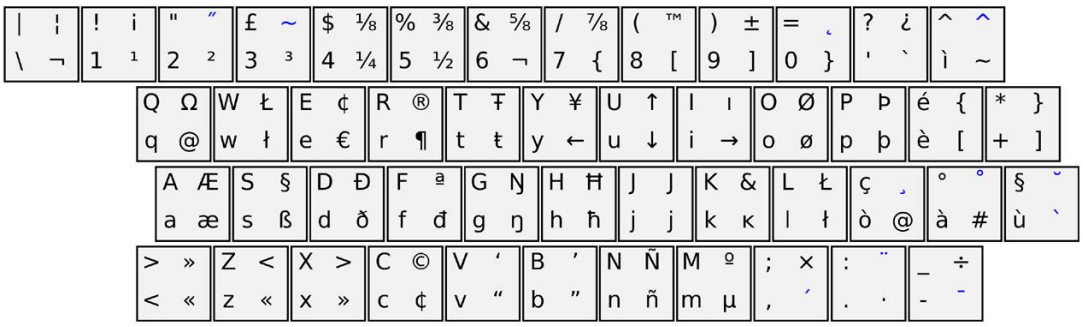
I file che verranno cercati potrebbero essere ad esempio nei seguenti percorsi:

- `"/usr/share/locale/it/LC_MESSAGES"`
- `"/usr/local/share/locale/it/LC_MESSAGES"`

Applicazione di tutto questo ai nostri script in Bash:

17.7 Tastiera italiana

```
$ loadkeys it
```



17.8 Accenti

Tutto quello che bisogna sapere per usare gli accenti.
Non avete la tastiera italiana? Nessun problema! ... anzi forse meglio così!
Modalità Grafica: vedi xmodmap sezione ...
Da Console:

17.9 Mouse

Scelta tra vari tipi di mouse: a rotella o a tre tasti.
Mouse in modalità non grafica: installare "gpm".

17.10 Demoni

Coda di stampa, talk, etc.

Stampa: Sui computer in cui è attivo il server di stampa CUPS, si può digitare l'indirizzo: "http://localhost:631/help".

Chat locale: Vedremo a parte i servizi di rete nella sez. 18.5.

17.11 Formattare dischi/partizioni

Abbiamo già parlato dei diversi filesystems nella sez. 3.5. In essa però abbiamo dato per scontato che le partizioni venissero create una volta per tutte all'installazione del sistema (argomento che non fa parte di questa guida) per poi rimare immutate. In realtà è probabile che capiti la necessità di dover formattare nuovi hard-disk (interni o esterni) oppure chiavette USB, schede di memoria etc.
Copia di sicurezza della propria tabella delle partizioni:

```
$ dd if=/dev/hda bs=512 count=1 of=hda-ptable-backup.data
```

17.12 Backup

Prima di addentrarci nei meandri dell'arte del backup da terminale, conviene forse rispondere ad una serie di domande, che ci indirezzeranno verso il metodo di backup più adatto ai nostri scopi.

1. Che cosa archiviare?

   Tutto quello che, nel caso andasse perso, vi farebbe disperare.

2. Ogni quanto farlo?

3. Quante copie fare?

4. Che supporti usare?

   Da evitare i floppy-disk (si usano ancora?). Tape? Gli Hard-Disk esterni sono Possono andare bene CD-Rom e DVD, con la consapevolezza però che la loro durata dipende in maniera sensibile da come vengono trattati, ed in ogni caso non è eterna. Fino a poco tempo fa venivano ancora usate le unità Zip-Iomega, ma oramai sono state soppiantate dalle pen-drive e dalle memory card. Entrambe sono piuttosto affidabili, ma

   Infine esiste la possibilità di archiviare i dati su qualche server su internet. (ad esempio sfruttando lo spazio offerto dalla casella di posta di Gmail) Questa però può essere una soluzione non sempre ottimale, in quanto

Creazione di un archivio di file:
tar.gz
tar.bz2
zip
Controllo: checksum
md5sum
sha1sum
gpg

17.13 Creazione di CD/DVD-Rom

Limitazioni del filesystem ISO 9660

- lunghezza dei nomi
- caratteri speciali
- link fisici e simbolici
- permessi
- orari

Metodo generico:

```
$ mkisofs -v -R -J -D -T -V "Etichetta" -o immagine.iso \
>  /path/dir
...
$ cdrecord -v speed=10 dev=0,1,0 -data immagine.iso
```

Dove, in questa sintassi, 0,1,0 sono i numeri corrispondenti a `"bus,target,lun"` che identificano il masterizzatore; potete scoprire quali sono sul vostro sistema tramite il comando:

```
$ cdrecord -scanbus
```

e cercando nella lista che vi viene fornita, il nome (marca e/o modello) del vostro masterizzatore. Nelle distribuzioni più recenti, spesso cdrecord è un link a `"wodim"` e si comporta in maniera leggermente diversa. Ad esempio nell'opzione della device va specificato il file nella directory `"/dev"` corrispondente al masterizzatore (potrebbe essere ad esempio `"dev=/dev/cdrom"`).

Cd Audio con cdrecord

Problema: gap tra le tracce audio. Soluzione: usare `"cdrdao"`

Creare un file `".toc"` che contiene le informazioni sulle tracce audio da masterizzare, in un formato del tipo:

```
CD_DA

// prima traccia audio
TRACK AUDIO
TWO_CHANNEL_AUDIO
AUDIOFILE "file_1.wav" 0

// seconda traccia audio con due secondi di pre-gap
TRACK AUDIO
TWO_CHANNEL_AUDIO
PREGAP 0:2:0
AUDIOFILE "file_2.wav" 0

...

// traccia dati finale
TRACK MODE1
DATAFILE "immagine.iso"
```

```
$ cdrdao write --device /dev/cdrom --driver generic-mmc file.toc
```

DVD: usare il comando `"growisofs"` del pacchetto `"cd-rw-tools"`.

Controllo del disco: Problema: settori di "padding" alla fine del Cd.

```
$ size=`isoinfo -s -i immagine.iso`
$ dd if=immagine.iso bs=2048 count=$size | md5sum
$ dd if=/dev/cdrom bs=2048 count=$size | md5sum
```

In modalità grafica, vanno citati i programmi "xcdroast" e "brasero".

17.14 Altre periferiche

Scanner: È necessario installare i pacchetti "libSANE" e "xsane".
È utile anche il plugin per Gimp ("xsane-gimp" o qualcosa del genere) che permette di acquisire l'immagine direttamente dal programma Gimp.

Webcam:

Scheda TV: xawtv, video4linux, streamer

18 Internet

18.1 Configurazione internet

Modem: Software: pppd, wvdial, kppp, gnome-ppp.

I winmodem (modem interni, visibili su Windows attraverso porte seriali emulate via software) spesso non vengono riconosciuti su Linux per la mancanza degli opportuni driver.

Se il produttore non ha sviluppato un suo driver proprietario appositamente per Linux (a volte capita infatti che questo esista ma che non sia incluso direttamente nelle distribuzioni per problemi di licenza) l'unica speranza resta il software smartlink che ...

Per maggiori informazioni al riguardo, si segnalano i seguenti siti:

- http://linmodems.org
- http://www.smlink.com
- http://

Scheda di rete: innanzitutto bisogna vedere se la vostra scheda di rete è stata correttamente riconosciuta dal sistema; per vederlo provate i comandi:

```
$ dmesg | grep eth
...
$ lspci
...
```

in entrambi i casi, se la vostra scheda è stata riconosciuta dovrebbe comparire nell'output di questi comandi. Ora, supponendo di essere fisicamente connessi ad una rete (LAN o WLAN), non resta che vedere, se è stato configurato (correttamente o meno) un indirizzo IP; per farlo:

```
$ ifconfig eth0
inet addr:127.0.0.1
Mask:255.0.0.0
...
```

dove al posto di "eth0" ci potrebbe essere "eth1", etc; attenzione però: di solito il comando "ifconfig" è in "/sbin" che molto spesso non è tra i percorsi inseriti nel "PATH"; se è così, sostituitelo con "/sbin/ifconfig eth0".

File di configurazione:
/etc/sysconfig/network-scripts/ifcfg-eth
o, a seconda della distribuzione,
/etc/sysconfig/network/ifcfg-eth0
/etc/network/interfaces
traceroute

netstat
mpstat
Il comando "snoop" è un semplice ma potente analizzatore di rete, in grado di controllare tutti i pacchetti ed eventualmente di filtrare il traffico.
whois
ping
telnet
ftp
ssh
finger
Esempi:
`finger user@host` informazioni su un utente, con account su un certo computer;
`finger @host` utenti logati su un certo computer;
`finger user` informazioni su di un utente, in locale.
"finger" mostra anche le informazioni contenute nei file "~/.plan" e "~/.project".
Solitamente i firewall vengono configurati per impedire l'accesso a questo tipo di informazioni, almeno dall'esterno.
Sincronizzazione di directory tra computer remoti: "rsync" (oppure in modalità grafica, "grsync").

18.2 Web

18.3 Mail

mutt/alpine/mail/sendmail
pop
"fetchmail"
Si può usare mutt per mandare una mail da linea di comando:

```
$ mutt -x -s "Subject" address@mail.com
Scrivo qui il testo da spedire nella mail,
e lo termino con <Ctrl+D> (EOF)
ciao a tutti
...
EOF
```

oppure per mandare una mail con un file allegato:

```
$ mutt -x -a allegato1.pdf -s "Subject" address@mail.com
Ti mando in allegato il file allegato1.pdf
ciao
EOF
```

Metodo "artigianale":

```
$ telnet smtp.mailserver.com 25
> HELO
> MAIL FROM <mionome@servizio.posta.it>
> RCPT TO <address@mail.com>
> DATA
```

```
bla bla bla
....
.
$
```

18.4 News

Un news aggregator è un programma studiato per tenersi aggiornati con tutti i siti e i blog che pubblicano un canale RSS. Quando un sito ha uno o più link che si presentano come RSS, RDF, XML, Atom o Syndicate, vuol dire che da quegli indirizzi potete seguire comodamente le novità in tempo reale con il vostro programma preferito.

18.5 Servizi di rete

Apache

Boa: web server molto leggero

ftp/httpd/mail

Webmin (gestione grafica dei servizi di rete come Apache, SSH e Samba, oltre che di MySQL).

18.6 Sicurezza

Elenco delle porte maggiormente usate:

Numero	Servizio	Descrizione
7	echo	
15	netstat	
21	ftp	
22	ssh	
23	telnet	
25	smtp	Invio mail
43	whois	
79	finger	
80	http	Web
110	pop3	Mail
111	portmap	
115	sftp	
119	nntp	UseNet News
143	imap2	Mail
220	imap3	Mail
443	https	Web
517	talk	
518	ntalk	
554	rtsp	RealTime Stream Control Protocol
563	nntps	NNTP Over SSL

873	rsync	
993	imaps	IMAP Over SSL
995	pop3s	POP-3 Over SSL
1755	mms	
3128	squid	Web Proxy
8080	http-proxy	Web Proxy

ssh vs telnet.
pericolo porte aperte.
firewall.
iptables
`$ iptables -L`
sicurezza password: john the ripper
16^{40} combinazioni

19 Crittografia

Crittografia a chiave privata. Crittografia a chiave pubblica.

20 OpenPGP

pgp/gpg
  firma digitale (ad es. per le mail), verifica, etc.

21 Filesystems criptati

22 X11: la modalità grafica

In questa sezione non si parlerà tanto dei vari window manager disponibili sotto Linux (per citarne alcuni: Gnome, KDE, Window Maker, Xfce4, BlackBox), ma piuttosto di come avviare ed utilizzare X ed alcune applicazioni grafiche, partendo dalla console e senza bisogno di avviare troppi (pesanti) processi.

22.1 Avvio da console

Supponiamo che abbiate accesso ad un sistema Linux, attraverso la console nella modalità non grafica (a cui si accede premendo i tasti `"Ctrl + Alt + Fn"`, con n normalmente da 1 a 6), e che il server X attualmente non sia in funzione. Allora per avviare la modalità grafica basta fare:

```
$ startx &
```

Molto probabilmente però, il server è già in funzione e sta occupando quella modalità grafica, o meglio quel `"DISPLAY"`. Forse però non sapete che potete avviarne un'altra. Il comando:

```
$ xinit -- :1 &
```

avvierà una nuova modalità grafica sul `"DISPLAY"` n. 1.

Per chiudere la modalità grafica (ossia uccidere il server X), premere contemporaneamente `"Ctrl + Alt + BackSpace"` (normalmente questo ne provoca semplicemente il riavvio, almeno nella modalità di default).

Potete anche collegarvi ad un altro computer ed esportare sul vostro `"DISPLAY"` le finestre aperte in remoto, attraverso il comando ssh.

```
$ ssh -X host.domain.com
```

Un modo alternativo consiste nell'accedere al sistema da remoto e poi impostare la variabile `"DISPLAY"` in modo opportuno:

```
$ export DISPLAY=localhost:0.0
```

Mouse. Non molti sanno che con X è possibile utilizzare il mouse anche ... senza mouse! È infatti possibile emularlo da tastiera premendo i seguenti tasti:[27]

`"Alt + Shift + Bloc-Num"`

A questo punto il vostro tastierino numerico fungerà da mouse. I numeri da `"1"` a `"9"`, escluso il `"5"`, corrispondono a movimenti secondo le direzioni indicate nella figura (e comunque seguendo una logica piuttosto intuitiva), mentre il `"5"` corrisponde ad un singolo clic del mouse. Il tasto corrispondente, però, va scelto precedentemente tramite i tre simboli in alto: `"/"` sta per il tasto sinistro, `"*"` per quello centrale e `"-"` per quello destro. Il `"+"` sta per il doppio clic ed infine `"0"` e `"."` stanno ad indicare rispettivamente che il tasto viene considerato tenuto premuto (durante gli spostamenti) oppure rilasciato. Tutto questo viene riassunto nella figura 1.

Questo non risulta utile solo in una situazione di emergenza, in cui o non si ha a disposizione il mouse o per qualche motivo esso non funziona, ma può essere un

[27] è possibile farlo in maniera analoga anche in Windows, anche se con una disposizione di tasti diversa, attivando la funzione di "Accesso Facilitato".

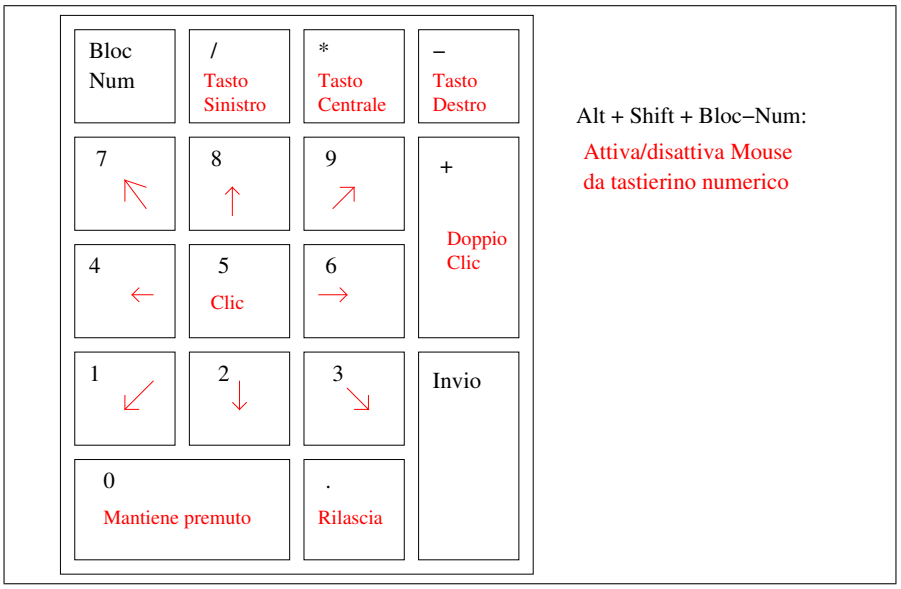


Figura 1: Emulazione del mouse dal tastierino numerico.

modo semplice per far fare degli spostamenti precisi e rettilinei al puntatore, come capita spesso di dover fare in un programma di grafica come `"gimp"` (cosa non sempre facilissima con il mouse).

Cattura dello schermo. È possibile *catturare* lo schermo e salvare l'immagine su un file tramite il comando `"xwd"`.

```
$ xwd > window.data
$ xwd -root > screen.data
```

Lanciato il comando, bisogna cliccare col mouse sulla finestra che si vuole catturare. Un suono viene emesso quando la finestra viene selezionata ed un altro quando viene completata la procedura di salvataggio. Se si vuole salvare l'intero schermo, bisogna specificare l'opzione `"-root"`, ma in questo caso non viene richiesto di cliccare col mouse. Per vedere il contenuto del file, bisogna ricorrere al comando `"xwud"`:

```
$ xwud < window.data
```

oppure convertire l'immagine ottenuta in un formato più consueto (ad esempio `"JPEG"` o `"PNG"`), tramite il comando `"convert"` cha fa parte del pacchetto `"Image-Magick"`:

```
$ convert window.data window.jpg
$ convert window.data window.png
```

Window Manager. Alcuni Window Manager:

KDE: probabilmente il più famoso (e per un certo periodo anche il più usato) di tutti. Ha un duplice vantaggio: essere *user-friendly* ed essere, nella sua configurazione di default, abbastanza simile all'interfaccia grafica di Windows. Questo lo rende facilmente usabile da quegli utenti che passano da Windows a Linux. Si basa sulle librerie Qt.

Gnome: in realtà non è un vero Window Manager nel senso che, a gestire veramente le finestre, si occupa un altro programma nascosto. Gnome è invece un ambiente grafico che si occupa di un po' di tutto, dal lanciare le applicazioni, al montaggio e smontaggio delle pen-drive, allo screensaver, etc ... (tutto questo attraverso una serie di servizi attivati in background e in linea di principio lanciabili anche da altri Window Manager). Si basa sulle librerie GTK+, proprio come Gimp.

Xfce4:

WindowMaker:

BlackBox:

dwm: scarno ed essenziale, ma anche il più veloce e leggero in assoluto (almeno di mia conoscenza). Presenta una caratteristica che lo differenzia da tutti i Window Manager precedentemente descritti: dispone in maniera automatica le finestre sullo schermo, suddividendolo in parti ben precise e definibili dall'utente. Può risultare poco intuitivo all'inizio e l'eventuale personalizzazione richiede una configurazione da fare direttamente sui sorgenti in "C" (questo per renderlo ancora più efficiente). Ma leggendo le pagine su internet e la pagina di manuale, ci si riesce a raccapezzare senza dover essere per forza un guru del "C".

Per il codice sorgente e maggiori informazioni: http://www.suckless.org/dwm.

wmii: il fratello maggiore di dwm; meno scarno e maggiormente configurabile.

Indirizzo: http://www.suckless.org/wmii

22.2 Configurazione per il singolo utente

xsetbg/xli

Esempio di come configurare il proprio file ".xsession" per utilizzare dwm (ma, a parte qualche riga, può andare bene con molti altri window manager alternativi).

File .xsession

```
1
2 # Personal Configuration
3
4 if [ -f ~/.Xresources ]; then
```

```
   xrdb -merge ~/.Xresources
fi

if [ -f ~/.Xmodmap ]; then
   xmodmap ~/.Xmodmap
else
   xmodmap -e "keycode 117 = Multi_key"
fi

xset +fp ~/.font_dir

# Screensaver

xscreensaver -nosplash &
scrsvr_pid = $!

# Background

if [ -f ~/background.jpg ]; then
   width=`xwininfo -root | grep Width | awk '{print $2}'`
   height=`xwininfo -root | grep Height | awk '{print $2}'`
   display -window root -size ${width}x${height}\! \
     ~/background.jpg &
else
   xsetroot -solid "#4477AA"
fi

# Window Manager

while true
do
   echo `date "+%a %_d %B %Y - %H:%M:%S"`
   sleep 1
done | dwm

# Exit

kill -9 $scrsvr_pid
```

Osservazione: il file ".xsession", deve essere reso eseguibile poiché viene lanciato dal gestore del login grafico ("xdm", "gdm" oppure "kdm", a seconda di quello che è stato installato). Se invece si vuole ottenere lo stesso risultato anche quando si fa partire la modalità grafica da console tramite il comando "xinit", il metodo migliore è creare un link simbolico:

```
$ cd
$ ln -s .xsession .xinitrc
```

22.3 Altri comandi di X e pacchetti utili

Alcuni programmi utili per testare la modalità grafica:

ical: un calendario grafico;

rxvt: terminale;

wish: sfrutta le librerie Tcl/Tk e può essere usato per creare degli script (vedi ad es. la sez. 14.1);

wmagnify: apre un riquadro in cui la porzione di schermo vicina al mouse viene mostrata ingrandita;

xclock: mostra un orologio analogico;

xkill: uccide l'applicazione corrispondente ad una certa finestra, cliccandoci sopra con il mouse;

xlock: blocca lo schermo chiedendo la password per ripristinarlo; (analogamente a `"xscreensaver-command -lock"`, ma a differenza dello screensaver non richiede un servizio in background);

xmag: ingrandisce una porzione di schermo;

xmessage: scrive un messaggio su una finestra grafica;

xon: ???;

xprop: mostra le proprietà di una finestra;

xscreensaver: avvia il servizio di screensaver;

xterm: terminale in modalità grafica che avvia la shell di default per l'utente;

xwininfo: mostra infromazioni su di una finestra (dimensioni, posizione, ...)

22.4 Fonts

L'importanza della scelta dei fonts.
Font a larghezza fissa vs font a larghezza variabile.
larghezza fissa: Courier, Monaco, Bera Mono
larghezza variabile: Helvetica, Palatino, Times
- con grazie (*serif*): Times - senza grazie (*sans serif*): Helvetica, Palatino
Font per LaTeX: Computer-Modern/Palatino/Times. Minion-Pro
I font "Computer-Modern" sono suddivisi in quattro famiglie, tra cui una a lunghezza fissa e una "corsiva", usata per la modalità matematica.
Font per il Web: Courier/Helvetica/New Times.

22.5 Multi-schermo

Xinerama.

22.6 Multimedia

Immagini: per visualizzare immagini, si ha a disposizione una vasta serie di comandi, tra i quali `"ee"`, `"xv"`, `"display"` (pacchetto Image-Magick), `"gqview"` e `"qiv"`, ciascuno dei quali apre più o meno tutti i tipi di immagine. Per visualizzare le gif animate, è possibile utilizzare il comando `"animate"`, del pacchetto Image-Magick, oppure `"xanim"` (o in alternativa un browser, ad esempio `"firefox"`).

Per modificare o creare immagini: `"gimp"` (che apre e salva in quasi tutti i formati), `"xfig"` (che utilizza un formato proprio, ma può esportare l'immagine creata in vari formati).

I tag "Exif" contenuti nelle immagini JPEG (creati dalle macchine fotografiche, contengono informazioni tipo data, ora e modalità di scatto), possono essere letti da shell tramite il comando `"jhead"`.

Audio: amarok, audacious, banshee, xmms.

Da terminale: mpg123/mpg321, mpd;

Infine `"mplayer"` apre anche altri formati come i WMA.

Mixer/Sound Server (oss o alsa/pulseaudio):

alsamixer

Per estrarre le tracce (*rippare*) da un cd audio, si possono usare i comandi `"cdda2wav"` o `"cdparanoia"`. Ad esempio:

```
$ cdparanoia -d /dev/cdrom -Owav -B 1-15
```

Questo estrae le 15 tracce del cd, salvandole in file WAV. Per poi convertire da WAV in MP3 è possibile utilizzare il comando `"lame"`:

```
$ lame -h -b 192
```

Per leggere i TAG contenuti nei file MP3, risulta utile il comando da terminale `"mp3info"`.

Se invece si preferisce codificare le tracce in formato OGG, basta usare l'analogo comando `"oggenc"`.

Sia gli MP3 che gli OGG, per quanto possano essere di buona qualità, perdono comunque delle informazioni rispetto al WAV originale. Esistono invece dei formati "lossless", ossia che mantengono inalterata la qualità originale. Tra questi uno dei più noti è il formato FLAC. Il comando `"flac"` è in grado di codificare e riprodurre questi file.

I file in formato MIDI sono completamente diversi: ...

mikmod

Il pacchetto "sox" contiene vari comandi, tra cui `"play"` che permette di suonare un file WAV e `"rec"` che registra direttamente dalla scheda audio (microfono o "line"). Il comando `"sox"`, invece, permette di convertire e fare varie operazioni sui file WAV.

Esempio: convertire tra formati wav (da compresso a normale):

```
$ sox -r 8000 -c 1 -L -i -2 rec001.wav -u rec002.wav
$ lame --bitwidth 16 -m m -s 8 -b 128 rec002.wav rec002.mp3
$ sox -r 8000 -c 1 -L -i -2 rec001.wav -u -r 32000 rec003.wav
$ lame --bitwidth 16 -m m -s 32 -b 128 rec003.wav rec003.mp3
```

Altri programmi utili:

audacity: programma di editing per l'audio, permette di fare varie operazione e di utilizzare filtri esterni; una particolarità: permette di ascoltare l'audio all'indietro (modalità *reverse*).

mpd:

Infine una curiosità: abbiamo visto come esportare il display su di un computer remoto. Ma per esportare il suono? Non esiste un modo nativo per farlo; bisogna sfruttare un qualche programma che faccia da server per il suono sulla macchina remota.

Se è stato installato `"esd"` allora basta digitare da locale, dove presumibilmente si vuole ascoltare fisicamente il suono, il comando:

```
local$ esd -public -tcp -port 5001
```

Mentre sul sistema remoto, dove sono presenti i file multimediali:

```
remote$ export ESPEAKER=<LOCAL_IP>:5001
remote$ mplayer -ao esd file.type
```

o in alternativa, se `"mplayer"` non è stato compilato per supportare l'output esd:

```
remote$ esddsp -v --server=<LOCAL_IP>:5001 mplayer file.type
```

Osservazione: `"esd"` fa parte del pacchetto `"esound"`. Se non è stato installato, ma si sta utilizzando `"kde"`, allora non ce n'è bisogno in quanto quest'ultimo può fare tutto da solo.

Video: per i filmati, i comandi *storici* sono: `"plaympeg"` per gli MPEG, e `"aviplay"` per gli AVI. Tuttavia poiché questi ultimi esistono in una gamma vastissima di codec diversi, sono nati dei programmi che, utilizzando librerie e plugin esterni, dovrebbero aprirli tutti (tranne alcuni WMV di Windows e RM di Real-Media). Tra questi:

mplayer: il quale funziona da linea di comando (anche se esistono delle interfacce grafiche, tipo `"gmplayer"`);

totem:

vlc:

xine:

Per poter invece manipolare/convertire i video tra vari formati, risulta utile/necessario installare i seguenti pacchetti/comandi:

mencoder: viene abitualmente installato assieme a `"mplayer"`.

transcode: effettua conversione tra vari formati (in realtà dipende da molti altri comandi esterni).

avidemux: permette la manipolazione di file avi e conversione da MPEG in AVI.

ffmpeg: codifica e decodifica vari formati video.

Web Radio: pacchetti necessari/software utile:
soma

Voip: Skype (non free); Asterisk.

22.7 Menu grafici

Il comando "zenity" permette di creare facilmente menu grafici.

```
$ mpg123 -v "`find . -type f -name '*.mp3' | zenity --list \
   --title 'List of MP3' --column 'Files'`"
```

dmenu: http://www.suckless.org/dmenu

23 Produzione di documenti e conversioni tra formati

Documenti LaTeX e simili.

Installazione di pacchetti LaTeX aggiuntivi: una volta copiati i file nel giusto percorso, va dato il comando `"mktexlsr"` (o eq. `"texhash"`) per aggiornare il database dei file di TeX.

Standard:

HTML.

SGML.

Postscript.

PDF `"xpdf"`.

DVI. `"xdvi"`, `"dvips"`

Software:

Ghostscript: `"gs"`, `"gv"`.

`"mpage"`

`"psmerge"`

`"ps2ps"`

`"psnup"`

`"psbook"`

`"latex2html"`

Open Office (alternativa open di Microsoft Office), Koffice, SoftMaker (quest'ultimo non free).

AbiWord e GNumeric.

Inkscape (grafica vettoriale).

IBM Lotus Symphony.

A Con cosa apro questo documento?

L'obbiettivo di questa sezione è elencare una serie di comandi in grado di aprire i vari tipi di file in cui possiamo facilmente imbatterci.

Strumenti utili: il comando `"file"`.

Estensione	Tipo	Comandi
.asf	Video	mplayer
.avi	Video AVI	aviplay, mplayer
.bas	Codice Basic	(*qualsiasi editor*)
.blend	File di Blender	blender
.bmp	Immagine bitmap	display, gqview, xview
.bz2	File compresso	bunzip2
.c	Codice C	(*qualsiasi editor*)
.cpp	Codice C++	(*qualsiasi editor*)
.doc	Documento di MS-Office	ooffice[28]
.docx	Ms Office / Open XML / Text	
.djvu	Documento DeJaVU	djview
.dll	Libreria Windows	–
.dvi	Documento DVI	xdvi
.dwg	File di AutoCad	???
.ecw	Immagine ECW	??????
.eps	Encapsulated Postscript	gv
.f	Codice Fortran (77/90/95)	(*qualsiasi editor*)
.fig	Immagine XFig	xfig
.flac	Audio FLAC	`"flac -d"`
.flv	Video	mplayer
.gif	Immagine GIF	display, gqview, xview
.gif	Gif animata	animate, firefox, xanim
.gz	File compresso	gunzip
.h	Sorgente C	(*qualsiasi editor*)
.html	Documento HTML	firefox, lynx
.iso	Immagine ISO	isoinfo, mount
.jpg	Immagine JPEG	display, ee, gqview, xv, xview
.lyx	Codice LaTeX/lyx	lyx
.mdb	Database	ooffice[28]
.mf	Codice MetaFont	(*qualsiasi editor*)
.midi	Audio MIDI	playmidi, timidity
.mov	Video QuickTime	mplayer ??
.mp	Codice MetaPost	(*qualsiasi editor*)
.mp3	Audio MP3	amarok, mpg123, xmms
.mpg	Video MPEG-1/2	mplayer, plaympeg
.nb	Mathematica	mathematica
.odb	Open Document Database	ooffice

continua ...

[28] non essendo un formato aperto, ooffice non ne conosce esattamente tutte le specifiche, per cui ci possono essere delle incompatibilità, soprattutto con le versioni più recenti del formato.

Estensione	Tipo	Comandi
.odg	Open Document Graphics	ooffice
.odp	Open Document Presentation	ooffice
.ods	Open Document Spreadsheet	ooffice
.odt	Open Document Text	ooffice
.ogg	Audio OGG-Vorbis	ogg123
.ogv	Ogg/Theora Video	mplayer, totem
.pas	Codice Pascal	(*qualsiasi editor*)
.pdf	Adobe PDF	acroread, gv, xpdf
.pdf.gz	PDF compresso	gv
.pfa	Font Postscript	???
.pfb	Font Postscript	???
.pl	Codice Perl	(*qualsiasi editor*)
.png	Immagine PNG	display
.pnm	Immagine PNM	display
.ppm	Immagine PPM	???
.pps	Power Point Slideshow	ooffice[28]
.ppt	Presentazione Power-Point	ooffice[28]
.pptx	Ms Office / Open XML / Presentation	
.ps	File Postscript	gv
.ps.gz	Postscript compresso	gv
.py	Codice Python	(*qualsiasi editor*)
.qt	Video QuickTime	mplayer
.ram	Audio Real-Media	mplayer, realplay
.rar	Archivio RAR	unrar
.rm	Video Real-Media	mplayer, realplay
.rgb	Immagine RGB	display
.rtf	Documento “Rich-Text-Format”	ooffice
.shtml		firefox
.svg	SVG	????
.swf	Flash	firefox
.tar	Archivio Tar	`"tar -xf"`
.tar.bz2	Archivio compresso	mc, `"tar -xjf"`
.tar.gz	Archivio compresso	mc, `"tar -xzf"`
.tcl	Script TCL/TK	(*qualsiasi editor*)
.tex	Codice LaTeX	(*qualsiasi editor*)
.tiff	Immagine TIFF	display
.ttf	True-Type Font	
.txt	Testo Ascii	(*qualsiasi editor*)
.vrml	Modello 3D	tidarque
.wav	Audio WAV	play
.wma	Windows Media Audio	mplayer
.wmf	Immagine Windows	wmf2x
.wmv	Windows Media Video	mplayer
.xcf	Immagine creata da GIMP	gimp
.xls	Foglio di Calcolo Excel	ooffice[28]

continua . . .

Estensione	Tipo	Comandi
.xlsx	Ms Office / Open XML / Spreadsheet	
.xml	Documento XML	
.xpm	X11 Pixmap	???
.zip	Archivio di file compressi	unzip

Altri tipi di file:

- TEX/LATEX ed affini: .aux, .idx, .ilg, .ind, .log, .toc, .tof (file creati durante la compilazione), .ist (personalizzazione dell'indice), .bib (bibliografia per BibTEX), .ins, .dtx, .sty (pacchetti LATEX).

Se si ha a che fare con un file con estensione sconosciuta, che non è stata descritta qui, molto probabilmente è possibile trovare qualche informazione sul sito http://filext.com.

B Creazione di un Makefile

Questa sezione vuole essere solo una panoramica sul funzionamento del comando "make" e sulla creazione di un tipico (e non troppo complicato) "Makefile". Per approfondire l'argomento si consiglia di leggere [21].

Scopo della creazione di un Makefile.

Vantaggi.

Comandi tipici.

B.1 Makefile per un tipico progetto in C

Utilizzo di librerie non standard:

Esempio1: ncurses
esempio concreto: menu su terminale

Esempio2: md5
esempio concreto: md5sum di un file

Esempio3: pdf
esempio concreto: creare un testo in pdf

Esempio4: Mesa/OpenGL
esempio concreto: finestra grafica con un ...

B.2 Makefile per la compilazione di codice LaTeX

Qui non parleremo di come scrivere codice in LaTeX, ma ci limiteremo a discutere della compilazione dei file. Per informazioni sul linguaggio o sull'installazione del software necessario, potete leggere i seguenti manuali: [16], [17], [18], [19], [20].

Attenzione: a seconda della distribuzione di LaTeX che avete installato, il formato predefinito del documento prodotto dalla compilazione del sorgente, mediante il comando "latex", può essere il DVI oppure il PDF. Qui di seguito suppongo che esso sia il DVI, e che il PDF venga invece prodotto dal comando "pdflatex".

Comando	Cosa produce
"latex documento.tex"	documento.dvi (ed altri file)
"dvips documento.dvi -o documento.ps"	documento.ps
"pdflatex documento.tex"	documento.pdf (ed altri file)
"makeindex documento.idx"	documento.ind (indice analitico)
"thumbpdf"	

La procedura standard consiste nel compilare una volta il file, producendo il documento in formato dvi, e altri file contenenti informazioni tipo l'indice, i link interni, la bibliografia, a seconda del codice e dei pacchetti utilizzati. Come minimo vengono prodotti il file con estensione .log (contenente le informazioni sulla

compilazione, compresi avvertimenti ed errori) e quello .aux (). Poi eventualmente va dato il comando "makeindex" che crea l'indice analitico, e quindi va ricompilato il file. Siccome alcuni pacchetti del LaTeX richiedono di essere compilati almeno due volte per funzionare correttamente, la sequenza di comandi che si consiglia di dare è:

```
$ latex documento.tex
$ makeindex documento.idx
$ latex documento.tex
$ latex documento.tex
$ dvips documento.dvi -o documento.ps
```

Applichiamo tutto questo, scrivendo il seguente opportuno Makefile:

Makefile LaTeX

```
#
# Personal Configuration
#

# Name of the latex source file to compile
source=prova.tex

# Your text editor (pico/nano/vim/emacs)
TE=vim

# Append here name of files eventually included in source
input_list=

# Specify your makeindex configuration file
# IDX_OPT=-s idx_style.idx

#
# Other (don't change!)
#

LC=latex
PDFLC=pdflatex
MKIDX=makeindex
COL="\033[1;31m"
BL="\033[0;0;0m"
name=${source:.tex=}
source_list=Makefile $(name).tex $(input_list)
arx_dir=$(name)_backup

help:
	@echo -e $(COL)"Usage:"$(BL)" make <Action | Target>"
	@echo
	@echo -e $(COL)"Actions:"$(BL)
	@echo -e "  help  (default)\t\t : print this message"
	@echo -e "  view          \t\t : compile and show pdf"
	@echo -e "  print         \t\t : compile and show ps"
	@echo -e "  clean         \t\t : clean all temporary files"
	@echo -e "  clean_all     \t\t : clean pdf/dvi/ps files also"
	@echo -e "  backup        \t\t : create archive (.tar.gz)"
```

```
40      @echo -e "  edit          \t\t : edit source file with $(TE)"
41      @echo -e "  touch         \t\t : touch source file"
42      @echo -e "                \t\t  (and so force recompilation)"
43      @echo
44      @echo -e $(COL)"Targets:"$(BL)
45      @echo -e "   $(name).pdf  \t\t : compile with pdflatex"
46      @echo -e "   $(name).dvi  \t\t : compile with latex"
47      @echo -e "   $(name).ps   \t\t : compile and create ps"
48      @echo -e "   $(arx_dir).tar.gz\t : create archive (.tar.gz)"
49      @echo
50
51  all: $(name).pdf $(name).ps
52
53  $(name).pdf: $(source_list); make clean
54      @echo
55      @echo -e $(COL)"Running PDF-LaTeX..."$(BL)
56      $(PDFLC) $(name).tex
57      @echo
58      @if [ -f $(name).idx ]; then \
59        echo -e $(COL)"Making Index ..."$(BL); \
60        $(MKIDX) $(IDX_OPT) $(name).idx; \
61        echo; \
62      fi
63      @echo -e $(COL)"Re-running PDF-LaTeX..."$(BL)
64      @$(PDFLC) $(name).tex
65      @$(PDFLC) $(name).tex
66      @echo
67
68  $(name).dvi: $(source_list); make clean
69      @echo
70      @echo -e $(COL)"Running LaTeX..."$(BL)
71      $(LC) $(name).tex
72      @echo
73      @if [ -f $(name).idx ]; then \
74        echo -e $(COL)"Making Index ..."$(BL); \
75        $(MKIDX) $(IDX_OPT) $(name).idx; \
76        echo; \
77      fi
78      @echo -e $(COL)"Re-running LaTeX..."$(BL)
79      $(LC) $(name).tex
80      $(LC) $(name).tex
81      @echo
82
83  $(name).ps: $(name).dvi
84      @echo -e $(COL)"Converting from Dvi to PostScript..."$(BL)
85      @echo
86      dvips -o $(name).ps $(name).dvi
87
88  view: $(name).pdf
89      xpdf $(name).pdf
90
91  print: $(name).ps
92      gv $(name).ps
93
94  edit:
95      vim $(name).tex
```

```
96
97  touch:
98       touch $(name).tex
99
100 backup: $(arx_dir).tar.gz
101
102 $(arx_dir).tar.gz: $(source_list)
103      @echo "Archive directory: $(arx_dir)"
104      @echo "Archive file     : $(arx_dir).tar.gz"
105      @echo "File List        : $(source_list)"
106      @echo
107      @echo -e $(COL)"Creating temporary directory..."$(BL)
108      mkdir -p $(arx_dir)
109      cp $(source_list) $(arx_dir)
110      @echo -e $(COL)"Creating backup archive and md5sum..."$(BL)
111      rm -f $(arx_dir).tar.gz
112      tar -cvzf $(arx_dir).tar.gz $(arx_dir)
113      md5sum $(arx_dir).tar.gz > $(arx_dir).md5
114      @echo -e $(COL)"Deleting temporary directory ..."$(BL)
115      rm -rf $(arx_dir)
116
117 clean:
118      @echo -e $(COL)"Cleaning old TeX compilation files..."$(BL)
119      rm -f $(name).aux $(name).log $(name).toc \
120      $(name).idx $(name).ind $(name).out $(name).ilg
121      @echo
122
123 clean_all: clean
124      @echo -e $(COL)"Cleaning pdf, dvi, and ps file..."$(BL)
125      rm -f $(name).pdf $(name).ps $(name).dvi
126      @echo
127
128 .PHONY: all backup clean clean_all default edit help \
129      print touch view
```

Basterà sostituire il vero nome del nostro file sorgente, e cambiare, se necessario la configurazione nelle righe iniziali, per avere un Makefile corretto nella stragrande maggioranza dei casi.

C Creazione di una pagina di manuale

Supponiamo di aver scritto una serie di Bash-script, o di programmi in qualsiasi altro linguaggio, e di voler fare un po' di ordine, in modo che anche dopo qualche tempo, io possa ritrovare facilmente le seguenti informazioni:

- Scopo del programma
- Funzionamento a grandi linee
- Opzioni da linea di comando
- Eventuali Bug

Scoprirete che questo può diventare molto importante, quando la vostra produzione di programmi aumenta e la memoria comincia a fare brutti scherzi ...

Ovviamente posso creare un file di testo con queste informazioni, e salvarlo da qualche parte, ma poi quando serve, dovrò andare a cercarlo. Perché allora, già che ci siamo, non facciamo un piccolo sforzo in più e creiamo la pagina di manuale per il nostro programma? Quando ne avremo bisogno, ci basterà digitare il classico `"man comando"` e avremo in maniera veloce le informazioni che cerchiamo, e per di più formattate molto meglio che in un banale file di testo.

Da notare che in un sistema multi-utente,[29] non c'è bisogno di inserire le pagine di manuale dei nostri comandi, nella directory comune (che di solito è `"/usr/share/man"`) ma possiamo creare una directory apposita nella nostra `"HOME"` (ad esempio `"$HOME/man"`) ed aggiungere questa nella lista di percorsi della variabile `"MANPATH"`.

Le pagine di manuale sono divise in sezioni, ossia un numero da 1 a 9 che catalogano i comandi secondo il seguente criterio:

Sez.	Utilizzo
1	
2	
3	
4	
5	
6	
7	

Inoltre i file possono essere compressi con `"gzip"` o con `"bzip2"`; sarà il programma `"man"` a decomprimerli automaticamente.

Nel nostro caso, dunque, potremmo creare la directory `"$HOME/man/man7"` e inserirvi i file creati.

Vedremo adesso quali sono le istruzioni necessarie per formattare correttamente il testo.

`"groff"`.

Lista dei comandi:

[29] nel senso che vi accedono persone estranee, per cui teniamo maggiormente alla privacy!

Codice	Effetto
.TH	titolo della pagina
.SH	nome della sezione
.SS	nome della sotto-sezione
.TP	indenta un paragrafo
.P	interruzione di riga
.BR	
.B	testo in grassetto (**bold**)
.I	testo in corsivo (*italic*)

Ecco un esempio molto semplice:

Esempio di pagina di manuale

```
.TH MY_BACKUP 7 "2008-03-17" "" "My Toolkit"
.SH NAME
my_backup - Crea un backup della propria home e ne fa il checksum
.SH SYNOPSIS
.BI my_backup

.SH DESCRIPTION
This is the new Version By \fImt\fP!!

Crea, un archivio tar, compresso con gzip, della propria home
(prendendo di default il percorso dalla variabile $HOME), ne
fa il checksum tramite md5sum/sha1sum e mette i file nella
directory /tmp, senza i permessi di lettura ad altri utenti.

.SH OPTIONS
Non ce ne sono (per ora!)

.SH BUGS
User o home con gli spazi ....

.SH AUTHOR
mt
.SH SEE ALSO
.BR my_arxiv (7)
```

Se volessimo infine esportare questa pagina in un formato più consueto anche per altre piattaforme? Ci sono varie possibilità; la più comoda probabilmente è sfruttare il comando `"man2html"` che converte la pagina in formato HTML.

D Il comando screen

Il comando "screen" consente di .. Risulta molto utile quando ci si connette ad un computer da remoto (ad esempio tramite "ssh").

Comandi principali:

Comando	Effetto
Ctrl+A	

E L'editor di testi vim

Leggere [22].

E.1 Perché sembra tutto così difficile?

```
Esc
  :comando
```

E.2 Comandi principali

Spostarsi nel file: in modalità di visualizzazione, i tasti `h`, `j`, `k`, e `l` spostano il cursore rispettivamente, un carattere a sinistra, una riga in basso, una riga in alto, e un carattere a destra. Lo stesso può comunque essere fatto anche mediante le quattro frecce, in modo più intuitivo. Il tasto 0 o Inizio-Riga, fa posizionare il cursore all'inizio della riga, mentre `$` o Fine-Riga fa posizionare il cursore a fine riga. per posizionarsi esattamente alla riga n, digitare `:n`.

Cancellare una riga: digitando `dd` la riga corrente viene cancellata dal testo e messa nel buffer senza nome.

Copiare una riga: con `yy` la riga viene copiata nel buffer senza-nome; se volete riusarla più volte in futuro, può essere conveniente memorizzarla in un buffer associato ad un nome; ce ne sono 26, ciascuno associato ad una lettera. Ad es. `"ayy` (attenzione: ci vanno le virgolette davanti) memorizza la riga nel buffer a.

Incollare una riga: una volta memorizzato il buffer, basta digitare `p`; premendo invece `<n>p` con n un numero anche di più cifre, l'ultimo testo memorizzato viene ricopiato esattamente n volte. Se vogliamo incollare il testo memorizzato nel buffer `a`, basta digitare `"ap`.

Cercare del testo: `/testo` premendo n si passa alla successiva occorrenza; premendo N, si torna a quella precedente.

Sostituire del testo: col comando `:10,20s/vecchio/nuovo/` viene sostituita la parola vecchio con nuovo, ma solo in ogni prima occorrenza per riga, nelle righe dalla 10 alla 20 del file. Se si vuole invece farlo per tutte le occorrenze, anche più di una volta per riga, bisogna specificare l'opzione `g` alla fine del comando. Ad esempio: `:1,$s/vecchio/nuovo/g` sostituisce in tutto il file vecchio con nuovo.

Registrare comandi: `q"addjpq` registra la sequenza di comandi dd, "a, j, p ossia taglia la riga di testo corrente, memorizzandola nel registro a, e la ricopia una riga più in basso. D'ora in poi per richiamare il comando basterà digitare: `@a`

Includere un file: `:r nomefile`

Lavorare su più file contemporaneamente:
Splittare la finestra in 2: `:split`
mentre per potersi poi spostare tra le finestre: Ctrl+w ⟨Up/Down⟩.
Comandi esterni:
Usare `"aspell"` come correttore ortografico:
Formattare il paragrafo in righe di uguale lunghezza: `!}fmt`

F Gnuplot

Gnuplot è un programma che permette di fare grafici a partire da delle funzioni o fornendogli i dati in un file opportunamente formattato (di solito due colonne di dati rappresentano punti 2-D; ma è possibile fornire anche le incertezze, o scegliere l'input tra più colonne). Si possono inoltre fare grafici 3-D, istogrammi, grafici in coordinate polari, etc. L'output può essere visualizzato in una finestra (in modalità grafica) oppure può essere salvato su un file, in diversi formati (postscript (eps) / png / svg).

Avviando gnuplot da terminale si ottiene il seguente prompt:

```
gnuplot>
```

Scrivendo `"help"` otteniamo una guida che descrive l'utilizzo di tutti i comandi e delle varie opzioni.

In alternativa si può scrivere la serie di comandi in un file (che possiamo chiamare ad esempio `"plot.gnu"`) e lanciare gnuplot da terminale, oppure aggiungere come prima riga del file il classico `"#/usr/bin/gnuplot"`, [30] poi rendere eseguibile il file e lanciarlo come uno script. In pratica:

```
$ gnuplot plot.gnu
```

```
$ chmod 755 plot.gnu
$ ./plot.gnu
```

Questo comporta l'evidente vantaggio di avere tutti i comandi salvati, e dunque già pronti per futuri riutilizzi e modifiche.

Comandi principali: Per fare il grafico di una funzione si usa il comando `"plot"`. La variabile da cui dipende la funzione è la x; quello che si intende col comando `"plot f(x)"` è disegna il grafico della funzione y = f(x).

```
plot x
pi=4*atan(1)
plot [-pi:pi] sin(x)
```

Per un grafico in 3-D si usa il comando `"splot"`. In questo caso le variabili sono x e y.

```
splot [-10:10][-10:10] x**2 - y**2
splot [-10:10][-2:2] sin(x) + y
```

Si possono mettere nel grafico anche i dati contenuti in un file. Di norma i dati devono essere suddivisi in due colonne (x e y) per i grafici in 2-D, in tre (x, y e z) per quelli in 3-D. È possibile, comunque, specificare quali colonne usare tramite l'opzione `"using"`:

[30] attenzione ad inserire il corretto percorso di gnuplot della vostra installazione.

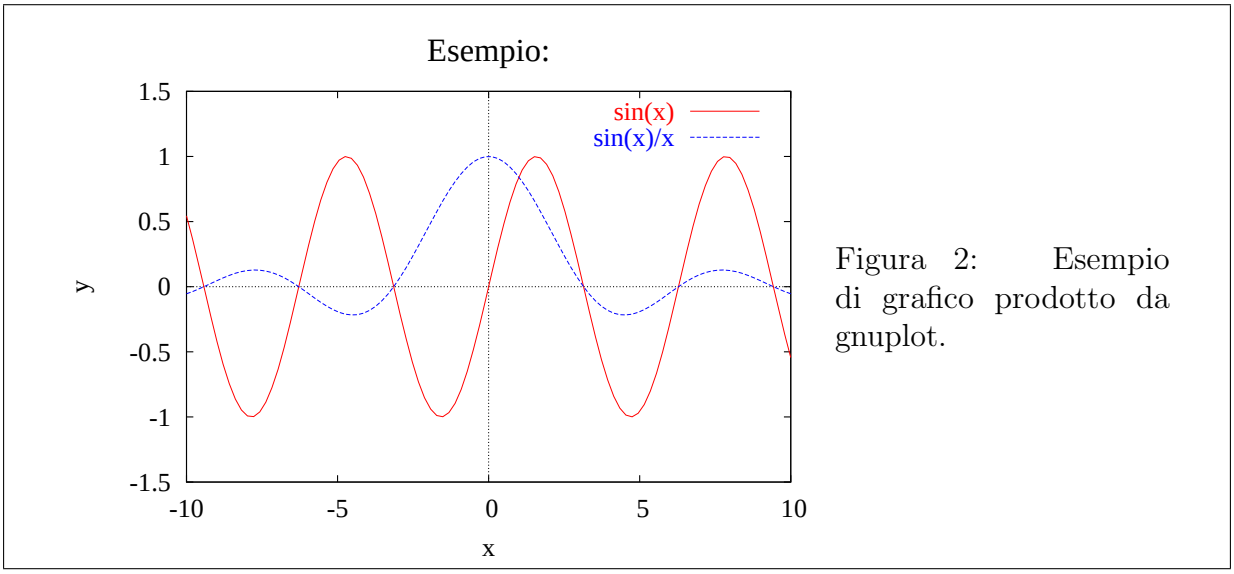


Figura 2: Esempio di grafico prodotto da gnuplot.

```
plot 'file_dati1.dat'
plot 'file_dati2.dat' using 2:3 with lines
```

Con l'opzione `"with"` si può decidere se graficare i dati come singoli punti o interpolandoli con delle linee (per visualizzare una funzione continua). La scelta dell'una o dell'altra opzione va fatta considerando unicamente l'estetica del risultato finale. È possibile anche fare un "fit" dei dati raccolti; ecco ad esempio come fare un fit a retta:

```
f(x) = a*x + b
fit f(x) 'file_dati1.dat' via a, b
```

L'istruzione `"set terminal"` permette di scegliere il tipo di terminale/formato di output e di specificare alcune altre opzioni. Di default è `"x11"`, ossia il grafico viene visualizzato in una finestra grafica. Le varianti che risultano maggiormente utili sono `"set terminal postscript"` e `"set terminal png"`.

Giocando con le opzioni, oltre a cambiare colori, stili, fonts etc, si arriva anche a fare grafici in coordinate polari (o sferiche in 3-D), istogrammi, grafici a torta ed altri ancora.

Ecco un semplice esempio di codice, il quale produce il risultato visibile nella figura 2.

Esempio di utilizzo di Gnuplot

```
1 #!/usr/bin/gnuplot
2
3 set terminal postscript eps color "Times-Roman" 24
4 set output "graf.eps"
5
6 set title "Esempio:" "Times-Roman,30"
7 set xlabel "x"
```

```
8  set ylabel "y"
9  set zeroaxis
10
11 plot [-10:10] [-1.5:1.5] sin(x) title "sin(x)",\
12  sin(x)/x title "sin(x)/x"
```

Per maggiori informazioni consultate il sito http://www.gnuplot.info.

Altri programmi: sebbene "gnuplot" possa fare molte cose, ha dei limiti. Un programma professionale che può fare praticamente tutto è "Mathematica" che però è proprietario (per cui non verrà descritto in questa guida).

In ambito open source, va citato "octave" che permette di fare calcoli numerici e di fare dei grafici di funzioni (utilizzando per questi ultimi lo stesso "gnuplot").

Un altro programma interessante (ed open source) è "Root" che è stato sviluppato presso il Cern (vedi il sito http://root.cern.ch).

Altri programmi che, sebbene nati con scopi diversi, hanno alcune caratteristiche in comune sono, infine, "IDL" e "Matlab".

G Creazione di database con PostgreSQL

H mySQL

I Controllo di versione di un software

Esistono molti programmi che gestiscono automaticamente le varie versioni di un software nelle varie fasi del suo sviluppo. Tra di essi vanno citati:

- arch
- bazaaar VCS
- CVS
- git
- Mercurial Hg
- RCS
- subversion
- SVN

Analizzeremo brevemente solo alcuni tra questi.

I.1 CVS

I.2 Mercurial (Hg)

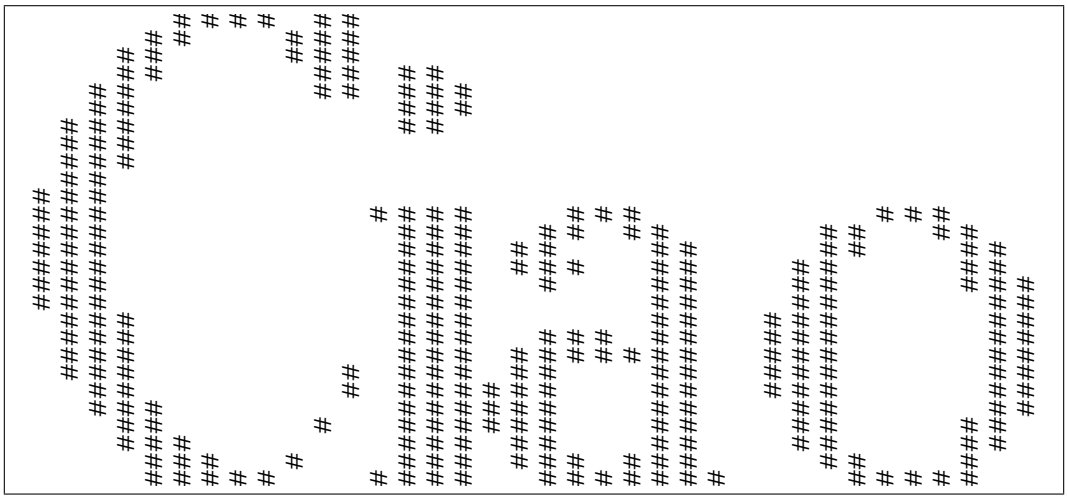


Figura 3: Output prodotto dal comando `"banner -w 40 2> /dev/null <<<Ciao | mpage -1 -o"`, ruotato di 90°.

J Toys

aalib: è una libreria che permette di visualizzare delle immagini approssimandole nel modo più fedele possibile con dei caratteri ascii (in bianco e nero, ovviamente). Il comando per visualizzare un file PNG è `"aview"`; per poter visualizzare anche formati più diffusi (come le immagini JPEG) è possibile sfruttare l'utility `"asciiview"` che opera la conversione in modo automatico; infine il comando `"apron"` (un pacchetto a parte) permette di visualizzare allo stesso modo dei video in formato MPEG.

banner: scrive un messaggio (testo ASCII) usando delle grandi lettere composte da dei `"#"`. È possibile vedere un esempio in figura 3.

blender: programma di rendering e animazione 3D, in grado di creare immagini, filmati e giochi (http://www.blender.org).

cal: mostra il calendario del mese, dell'anno in corso o di un qualunque mese/anno specificato. Ne esiste anche una versione migliorata, di nome `"gcal"` (ftp://ftp.gnu.org/gnu/gcal).

calendario

```
$ date
mar ott 20 18:47:41 CEST 2009
$ cal -3
    settembre 2009         ottobre 2009          novembre 2009
do lu ma me gi ve sa  do lu ma me gi ve sa  do lu ma me gi ve sa
       1  2  3  4  5              1  2  3   1  2  3  4  5  6  7
 6  7  8  9 10 11 12   4  5  6  7  8  9 10   8  9 10 11 12 13 14
13 14 15 16 17 18 19  11 12 13 14 15 16 17  15 16 17 18 19 20 21
20 21 22 23 24 25 26  18 19 20 21 22 23 24  22 23 24 25 26 27 28
27 28 29 30           25 26 27 28 29 30 31  29 30

$ cal -y
                                2009

       gennaio               febbraio                 marzo
```

```
do lu ma me gi ve sa   do lu ma me gi ve sa   do lu ma me gi ve sa
             1  2  3    1  2  3  4  5  6  7    1  2  3  4  5  6  7
 4  5  6  7  8  9 10    8  9 10 11 12 13 14    8  9 10 11 12 13 14
11 12 13 14 15 16 17   15 16 17 18 19 20 21   15 16 17 18 19 20 21
18 19 20 21 22 23 24   22 23 24 25 26 27 28   22 23 24 25 26 27 28
25 26 27 28 29 30 31                          29 30 31

       aprile                 maggio                 giugno
do lu ma me gi ve sa   do lu ma me gi ve sa   do lu ma me gi ve sa
          1  2  3  4                   1  2        1  2  3  4  5  6
 5  6  7  8  9 10 11    3  4  5  6  7  8  9    7  8  9 10 11 12 13
12 13 14 15 16 17 18   10 11 12 13 14 15 16   14 15 16 17 18 19 20
19 20 21 22 23 24 25   17 18 19 20 21 22 23   21 22 23 24 25 26 27
26 27 28 29 30         24 25 26 27 28 29 30   28 29 30
                       31
       luglio                 agosto                settembre
do lu ma me gi ve sa   do lu ma me gi ve sa   do lu ma me gi ve sa
          1  2  3  4                      1          1  2  3  4  5
 5  6  7  8  9 10 11    2  3  4  5  6  7  8    6  7  8  9 10 11 12
12 13 14 15 16 17 18    9 10 11 12 13 14 15   13 14 15 16 17 18 19
19 20 21 22 23 24 25   16 17 18 19 20 21 22   20 21 22 23 24 25 26
26 27 28 29 30 31      23 24 25 26 27 28 29   27 28 29 30
                       30 31
       ottobre               novembre               dicembre
do lu ma me gi ve sa   do lu ma me gi ve sa   do lu ma me gi ve sa
             1  2  3    1  2  3  4  5  6  7          1  2  3  4  5
 4  5  6  7  8  9 10    8  9 10 11 12 13 14    6  7  8  9 10 11 12
11 12 13 14 15 16 17   15 16 17 18 19 20 21   13 14 15 16 17 18 19
18 19 20 21 22 23 24   22 23 24 25 26 27 28   20 21 22 23 24 25 26
25 26 27 28 29 30 31   29 30                  27 28 29 30 31
```

celestia: simulatore delle orbite dei principali corpi celesti del sistema solare.

cowsay: inserisce la frase in input all'interno di un simpatico fumetto associato ad un certo animale, il quale può essere scelto dall'utente; il tutto viene mostrato in output sul terminale in codice ASCII.

ekiga: software free per il *VOIP*.

espeak: legge un messaggio tramite un sintetizzatore vocale.

festival: sintetizzatore vocale, ampliabile con librerie, in varie lingue e toni di voce.

figlet: simile a `"banner"`, ma con più opzioni per lo stile e i caratteri usati; esiste anche un altro comando, `"toilet"` che permette anche di colorare l'output e di salvarlo in vari formati, tra cui l'Html. Va citato anche `"cowsay"` che racchiude il testo in ...

```
$ figlet ciao a tutti
      _                       _         _   _   _
  ___(_) __ _  ___     __ _  | |_ _   _| |_| |_(_)
 / __| |/ _` |/ _ \   / _` | | __| | | | __| __| |
| (__| | (_| | (_) | | (_| | | |_| |_| | |_| |_| |
 \___|_|\__,_|\___/   \__,_|  \__|\__,_|\__|\__|_|
```

fontforge: permette di creare, direttamente da una finestra grafica, dei nuovi font in vari formati.

fortune: stampa un simpatico messaggio scelto casualmente all'interno di un database.

genius: conti numerici a precisione arbitraria (come "bc").

gnuchess: motore per il gioco degli scacchi.

istanbul: programma che permette di registrare un video direttamente dal vostro schermo (operazione detta anche *screencast*).

streamtuner: internet radio.

sunclock: orari di alba e tramonto.

xboard: permette di giocare a scacchi contro il computer (se viene installato un motore come "gnuchess" o "crafty") oppure contro altri utenti su un server internet (ad esempio: http://www.chessclub.com). Se siete invece interessati a creare un server sul vostro computer provate con "chessd" (http://chessd.sourceforge.net).

K Riepilogo comandi visti (e non)

Qui di seguito sono elencati tutti i comandi visti (e anche alcuni di cui non si è parlato) tranne quelli riguardanti specificatamente X, già descritti nella sezione 22, e quelli descritti nella sezione J.

adp: assembler debugger in grado di eseguire del codice un passo alla volta;

alias: crea un alias per un certo comando;

apropos: mostra i comandi inerenti ad una certa parola;

at: permette di eseguire un comando ad una certa ora prefissata;

awk: potente strumento per la manipolazione di testi/dati;

basename: compie delle operazioni su dei nomi di file;

bc: calcolatrice in modalità testuale;

bg: manda un certo job in background;

bind: associa delle combinazioni di tasti a delle funzioni di readline;

bunzip2: decomprime file compressi col programma bzip2;

bzip2: programma di compressione;

cat: mostra il contenuto di un file;

cd: cambia la directory;

chage: mostra e/o cambia le informazioni sulla password;

chfn: cambia le proprie "finger information";

chgrp: cambia il gruppo proprietario di un file;

chmod: cambia permessi ad un file;

chown: cambia il proprietario di un file;

chsh: cambia la shell di default dell'utente;

clear: pulisce lo schermo (su terminale);

cmp: compara il contenuto di due file;

colrm: rimuove le colonne specificate;

column: sistema su colonne l'input proveniente da uno o più file;

cp: copia un file;

crontab: permette di pianificare dei comandi che devono essere ripetuti quotidianamente, settimanalmente o mensilmente ...

csplit: suddivide un file, in blocchi determinati da un pattern;

cut: rimuove sezioni da ogni riga di file;

date: mostra data e ora;

dd: copia un file dallo stdin allo stdout;

declare: dichiara una variabile;

df: visualizza l'ammontare di spazio libero su disco;

diff: mostra le differenze tra due file (o directory);

dirs: mostra la lista delle directory ricordate;

dmesg: mostra le informazioni generate dal kernel;

du: mostra lo spazio occupato da una directory sul disco;

echo: stampa il messaggio specificato;

ed: editor di testi *user-unfriendly*;

enable: abilita/disabilita i comandi incorporati della shell;

eval: esegue il comando specificato come argomento e ne riporta lo stato di uscita;

exec: esegue un comando ed esce dalla shell interattiva (a meno che non si possa eseguire il comando);

export: esporta una variabile (in eventuali subshell);

factor: scompone un numero nei suoi fattori primi;

fg: rimette un certo job in foreground;

file: determina il tipo di file;

find: permette di fare ricerche di file;

finger: mostra informazioni sugli utenti;

firefox: browser internet;

flac: codifica o decodifica un file audio nel formato flac (*Free Lossless Audio Codec*);

fmt: formatta l'input suddividendolo (se possibile) in righe di pari lunghezza;

fold: formatta l'input suddividendolo in righe di pari lunghezza;

free: mostra le quantità di memoria RAM e swap libere ed occupate;

ftp: sistema di trasferimento file tra diversi computer;

gimp: programma di grafica e fotoritocco;

groff: ???

gunzip: decomprime i file compressi con gzip;

gzip: programma di compressione;

halt: ferma il sistema e spegne il computer;

hash: mostra il numero di occorrenze dei comandi usati nella sessione corrente;

head: stampa le prime n righe di un file;

history: elenca i comandi memorizati nella HISTORY, con un numero progressivo;

hostname: mostra o assegna il nome al computer;

id: mostra User ID e Group ID;

info: legge la documentazione di un comando;

inkscape: programma di grafica vettoriale;

ispell: controlla se un testo contiene parole appartenenti ad un certa lingua;

jobs: elenca i jobs attivi;

join: stampa le righe di due file che hanno un campo in comune;

kill: manda dei segnali a dei processi;

lame: codifica un file audio (WAV) nel formato MP3;

last: mostra l'elenco degli ultimi accessi (login);

lastb: mostra l'elenco degli ultimi tentativi di accesso falliti;

lastlog: mostra l'ultimo login per ogni utente;

ld: unisce gli oggetti compilati per creare l'eseguibile finale;

ldd: elenca le librerie richiamate da un file eseguibile;

less: mostra il contenuto di un file, permettendo lo scorrimento sia in avanti che all'indietro;

links: browser internet testuale;

ln: crea link (simbolici o fisici) verso dei file;

locate: elenca i file contenuti in un database, che corrispondo al modello richiesto;

logname: mostra il nome di login (non cambia se si diventa un altro utente tramite su);

logout: disconnette l'utente dal sistema;

lpq: controllo coda di stampa;

lpr: comando di stampa;

ls: elenca i file;

lynx: browser internet testuale;

man: mostra la pagina di manuale di un comando;

man2html: converte una pagina di manuale nel formato HTML;

mc: Midnight-Commander è un "browser" in modalità testuale per file e directory, con un suo prompt dei comandi e in grado di aprire vari tipi di file (immagini ed archivi);

mesg: mostra/cambia gli accessi in scrittura sul proprio terminale;

mkdir: crea una nuova directory;

mkfifo: crea un tipo speciale di file, che funge da pipe;

mknod: come mkfifo, ma per file speciali, definiti dal kernel;

mktemp: crea un file temporaneo dal nome univocamente definito;

more: mostra il contenuto di un file una pagina per volta;

mount: monta un filesystem in una certa directory (mount-point);

mp3info: legge o cambia le informazioni (Tag ID3) di un file MP3;

mpage: converte il testo ascii in input in un file postscript;

mpg123: legge i file audio in formato MP3;

mplayer: lettore di video di vari formati e codec;

mutt: programma per il controllo della posta elettronica in modalità testuale;

mv: sposta o rinomina un file;

nano: semplice editor di testi;

nc: utility che legge/scrive dati da una connessione TCP/IP;

nice: avvia un comando con una certa priorità;

nl: stampa le righe di un file numerandole progressivamente;

nohup: lancia un comando in modo che questo sia immune dai segnali di hangup e non venga terminato dalla disconnessione dell'utente;

od: stampa una rappresentazione in ottale del contenuto di un file;

ogg123: legge i file audio Ogg-Vorbis;

oggenc: codifica un file audio (WAV) nel formato Ogg-Vorbis;

ogginfo: legge le informazioni di un file audio Ogg-Vorbis;

passwd: cambia password dell'utente;

paste:

pico: semplice editor di testi;

pine: programma per il controllo della posta elettronica in modalità testuale;

ping: controlla se un computer remoto è raggiungibile;

play: suona un file audio WAV;

pr: prepara l'input per essere stampato, suddividendolo in pagine e numerandole;

ps: elenca processi attivi;

ps2ps:

psbook: sistema delle pagine per essere stampate in un file postscript;

psmerge: unisce diversi documenti Postscript in un unico file;

psnup: sistema le pagine di un file postscript, anche in più di una per foglio, per essere stampate;

pstree: mostra l'albero dei processi;

pwd: mostra l'attuale percorso;

quota: mostra la quantità libera e occupata di spazio su disco assegnata all'utente;

read: legge da standard input valore da assegnare ad una variabile;

reboot: riavvia il sistema;

rename: rinomina serie di file seguendo un modello;

renice: altera la priorità di processi in esecuzione;

reset: reinizializza i parametri di un terminale; vedi anche i comandi `"tset"` e `"tput"`;

rev: stampa un file invertendolo da destra a sinistra riga per riga;

rgrep: grep ricorsivo per le sotto-directory;

rm: rimuove un file;

rmdir: rimuove una directory (vuota);

scp: permette il trasferimento di file, usando il protocollo `"ssh"`;

screen: permette di aprire più finestre su di un unico terminale;

sed: strumento per la modifica di testi;

seq: stampa una sequenza di numeri;

set: setta una variabile;

setleds: imposta i led della tastiera (modalità non grafica);

setterm: setta gli attributi del terminale;

shred: cancella un file in modo sicuro, sovrascrivendolo (ormai obsoleto, non funziona con i file-system *journaled*);

sg: analogo al comando `"su"`, ma assume l'identità di un gruppo;

shutdown: spegne o riavvia il sistema;

sleep: attende per n secondi;

sort: ordina alfabeticamente le righe in input;

split: divide il file di input in pezzi di dimensione prefissata;

ssh: permette l'accesso di un utente ad altri computer;

stat: mostra informazioni su un file;

stty: mostra/cambia impostazioni della linea di prompt del terminale;

su: assume l'identità di root o di un altro utente;

sudo: esegue un comando con i privilegi di root;

tac: stampa il contenuto di un file procedendo dall'ultima riga alla prima;

tail: stampa le ultime n righe di un file;

talk: permette la conversazione su terminale tra due utenti;

tar: crea degli archivi di file;

tee: legge dallo standard input e scrive nello standard output e su file;

telnet: permette l'accesso ad altri computer [31];

[31] per motivi di sicurezza dovrebbe essere evitato, e rimpiazzato con `ssh`.

test: effettua vari tipi di test;

time: calcola il tempo di esecuzione di un comando;

top: monitora processi attivi;

touch: cambia gli orari di accesso ed ultima modifica di un file;

tput: mostra i parametri di un terminale;

tr: traduce o cancella un set di caratteri specificato;

trap: fa eseguire alla shell il comando specificato quando ricece un certo segnale;

tree: mostra albero delle directory;

troff: ???

tset: inizializza i parametri di un terminale;

ulimit ??

umask: cambia maschera che definisce i permessi dei nuovi file;

umount: smonta un filesystem;

unalias: toglie un alias;

uname: mostra informazioni sul sistema operativo e sul computer;

uniq: rimuove le linee doppie da un file già ordinato;

unlink: rimuove un singolo file;

unset: elimina una variabile;

unzip: estrae file da archivi di tipo zip;

uptime: mostra informazioni sul sistema;

usleep: attende per n microsecondi;

vi: editor di testi;

wait: attende lo stato di uscita di un processo;

wall: manda un messaggio a tutti gli utenti (vedi il comando `"write"`);

wc: conta il numero di righe, parole e byte in input;

wget: scarica un file, dato l'indirizzo internet (URL), in modalità non interattiva;

whatis: cerca un comando o una pagina di manuale nel suo database;

which: mostra qual'è il percorso di un comando;

whereis: mostra il percorso di un comando o di una pagina di manuale;

who: mostra utenti presenti sul sistema;

whoami: mostra il proprio username;

whois: interroga il ??? a proposito di un dominio internet registrato;

write: scrive un messaggio ad un altro utente con una shell aperta sulla stessa macchina;

xargs: esegue un certo comando sulla lista di argomenti in input;

yes: restituisce una serie infinita di caratteri `'y'` come output;

zip: programma di archiviazione e compressione;

L Glossario

Questa sezione fa un elenco dei termini tecnici (o comunque largamente diffusi nel mondo informatico), per lo più in inglese, incontrati nel manuale, dandone una breve definizione.

Ascii:

Backdoor:

Background:

Backup:

Bios:

Bluetooth:

Boot: termine con cui si indica l'avvio del sistema operativo, caratterizzato dal caricamento del kernel e successivamente dall'avvio dei vari servizi e dal caricamento delle varie impostazioni.

Boot loader: programma che si occupa all'avvio di caricare il kernel del sistema operativo. Deve trovarsi nel Master Boot Record dell'Hard-Disk, in modo che venga lanciato automaticamente dal Bios, oppure può trovarsi in un apposito dischetto di avvio o in un dispositivo analogo (ad es. una pen-drive, se l'hardware consente l'avvio direttamente da questa).

Browser: programma che consente la navigazione nel Web. La parte di esso (ma può anche essere un programma esterno) che si occupa di tradurre le informazioni contenute nella pagina Html disponendole in un formato grafico, viene detto "motore di rendering". Per anni il browser di default del mondo Unix è stato Netscape, poi sostituito dalla famiglia Mozilla/Firefox. Vanno menzionati anche Opera, Konqueror, Galeon e Links.

Bug: (letteralmente pulce, ma tradotto solitamente in baco)

Bus error:

Compilatore: programma che si occupa di convertire il codice scritto dal programmatore in un certo linguaggio, in codice binario, capibile dalla macchina.

Console:

Core dump:

CPU: unità di elaborazione centrale (*Central Processing Unit*).

Crack: (in inglese – rompere) si può riferire a varie cose. Può indicare la rottura di un qualche algoritmo di crittografia, così da consentire l'accesso ai dati da esso protetti anche senza la chiave. Spesso indica un programma in grado ti togliere la protezione ad un software che per funzionare necessita dell'acquisto

della licenza (e della relativa password) ma questo, oltre che illegale, è estraneo al mondo free. Oppure significa superare le misure anticopia che proteggono certi formati o supporti multimediali (ad esempio i file WMV o i dischi DVD-Video). Secondo le licenze, questa operazione sarebbe illegale, anche se viene fatta non per copiare contenuti da altri, ma solo per accedere a ciò che abbiamo regolarmente acquistato da una piattaforma che non potrebbe farlo. Un altro problema è il diritto di backup, che l'utente vorrebbe esercitare, ma che non sarebbe possibile effettuare senza aggirare le protezioni. In questi casi è difficile capire quale sia esattamente il limite giuridico di ciò che possiamo e non possiamo fare, ed oltretutto le norme variano nel tempo e di nazione in nazione.

Demone:

Device:

Distribuzione:

Dos: (*Denial of Service*).

DOS: (*Disk Operating System*).

Editor:

Environment: l'insieme delle variabili d'ambiente.

Ethernet:

Exploit:

File-system:

Firewall:

Firewire: vedi IE....

Font:

Foreground:

Fork: Fork Bomb:

GUI: (*Graphical User Interface*)

Hacker:

IEE 1394:

Interprete:

IP: indirizzo assegnato ad un computer nell'ambito di una rete, sia essa locale o globale.

ISO 8859:

ISO 9660:

ISO 10646: (*Universal Character Set*) ampliamento di Unicode, comprendente fino a 2^{31} diversi caratteri.

Journaling:

Kernel: nucleo del sistema operativo. Ha il compito di permettere ai processi in esecuzione l'accesso all'hardware di sistema.

LAMP: Acronimo di Linux–Apache–MySQL–PHP.

Librerie:

Modem:

MUA: (*Mail User Agent*)

Multitasking:

Partizione: parte in cui può essere suddiviso un supporto di memoria (di norma un hard-disk), vista e gestita come un dispositivo indipendente.

Pattern:

Pipe: (lett.) indica

Porta parallela:

Prompt: messaggio del terminale che ci indica che la shell è pronta per ricevere un nuovo comando. Esistono dei prompt secondari, che normalmente richiedono il completamento di un certo comando, o prompt di altri programmi che richiedono dell'input da tastiera.

RAM:

Random: numero pseudo-casuale generato dal computer. Nessun algoritmo è in grado, da solo, di generare una sequenza di numeri che sia realmente casuale, in quanto essa risulta sempre e comunque periodica, ossia prima o poi i numeri cominceranno a ripetersi, nello stesso ordine. Si può ottimizzare il procedimento allungando il più possibile il periodo e creando diverse sequenze associate ciascuno ad un numero che le genera (seme). Nell'utilizzo pratico, ad esempio nei metodi di tipo Montecarlo, è importante diversificare i risultati scegliendo un seme veramente casuale; di solito per fare questo si sfrutta un qualche dato esterno (ad esempio i centesimi di secondo dell'ora corrente, o una qualche combinazione di altre informazioni). Caratteristiche importanti, dell'algoritmo di generazione di numeri random, oltre all'assenza di correlazioni tra un numero e l'altro (comunque non sempre facili da rintracciare) sono l'ergodicità (ossia tutti i numeri dell'intervallo considerato devono poter essere generati) e l'uniformità della distribuzione (ossia ogni numero dell'intervallo, deve avere la stessa probabilità degli altri di essere generato).

Reboot:

Repository: archivi di software online, da cui normalmente vengono scaricati i pacchetti di aggiornamento.

Reverse engineering: branca dell'informatica che si occupa di invertire il procedimento effettuato dal compilatore, ossia dato il codice binario, cercare di risalire al codice sorgente che lo ha generato.

RFC: (*Request for Comments*) insieme di documenti che descrivono le specifiche di alcuni protocolli o comandi.

RS232:

Scanner:

Script:

Segmentation fault: tipico errore con cui termina un programma compilato quando non riesce a soddisfare una certa richiesta di lettura/scrittura della memoria (tipicamente quando si ha a che fare con degli array).

Shell: programma che si occupa di interpretare i comandi scritti dall'utente e di farli eseguire al sistema operativo. Esistono diversi tipi di shell, tra cui Bash.

Shutdown: indica il procedimento attraverso il quale il sistema operativo si prepara per lo spegnimento, arrestando prima i vari servizi, in un ordine prestabilito, ed eseguendo altre eventuali operazioni necessarie.

Social engineering: (spesso tradotto in ingegneria sociale)

Socket:

Tools: pacchetto di strumenti (software) dedicati ad un certo scopo, come ad esempio testare la sicurezza del sistema o compiere delle conversioni tra vari formati di documento.

Tape:

Trojan:

Usb:

UTF-8: codifica di caratteri Unicode implementata in un sistema Unix. I caratteri Unicode sono a 16 bit, e questo li renderebbe completamente incompatibili con il set di caratteri Ascii, con l'evidente vantaggio tuttavia di poter rappresentare un numero molto maggiore di caratteri. UTF-8 ha però la caratteristica di poter usare per un singolo carattere uno, due oppure tre byte; i primi 128 caratteri (a singolo byte) sono ovviamente in comune con il codice Ascii (e con lo standard ISO 8859), ripristinando così la compatibilità (almeno parziale) con esso, mentre quelli successivi differiscono, di solito, per un byte aggiuntivo di prefisso (nel caso dell'UTF-8). L'eventuale implementazione

tramite UTF-8, dell' UCS (ISO 10646) arriverebbe ad occupare fino ad un massimo di 6 byte.

Virus:

VOIP: (*Voice Over Internet Peer*) skype, asterisk.

Wi-Fi: (o più genericamente wireless).

Wi-Max:

Window-Manager:

Worm:

Zip: unità di memorizzazione su disco magnetico, prodotti dalla Iomega, simili al floppy-disk ma con capacità che vanno dai 100 MB (scarsi) ai circa 750 MB. Sono oramai caduti in disuso.

Riferimenti bibliografici e siti internet

Manuali

Linux

[1] Daniele Giacomini, *Appunti di Informatica Libera* (http://linuxdidattica.org/a2/) (http://www.informaticalibera.it/a2/).

[2] S. Kochan e P. Woods, *Unix Shell Programming*, Hayden 1990.

[3] N. Matthew e R. Stones, *Beginning Linux Programming*, Wrox Press, 1996.

Bash

[4] Brian Fox e Chet Ramey, *Bash Man Page* (comando "man bash", o "info bash").

[5] Brian Fox e Chet Ramey, *Bash Features*.

[6] Mendel Cooper, *Advanced Bash-Scripting Guide* (http://personal.riverusers.com/~thegrendel/abs-guide-1.7.tar.bz2), tradotto in italiano da Emilio Conti, membro dell'ILDP, col nome di *Guida avanzata di Bash-Scripting* (http://www2.pluto.it/ildp/guide/abs/).

Linguaggi

[7] D. Dougherty e A. Robbins, *Sed and Awk*, 2nd edition, O'Reilly and Associates, 1997.

[8] A. Aho, B. Kernighan e P. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1984.

[9] *The GNU Awk User's Guide* (http://www.gnu.org/software/gawk/manual/gawk.html).

[10] L. Wall e R. L. Schwartz, *Programming Perl*, O'Reilly and Associates, 1991.

[11] R. L. Schwartz, *Learning Perl*, O'Reilly and Associates, 1993.

[12] B. W. Kernighan e D. M. Ritchie, *The C Programming Language*, 2nd edition, Prentice Hall, 1988.

[13] B. Stroustrup, *The C++ Programming Language*, 3rd edition, Addison-Wesley, 1997.

[14] B. Eckel, *Thinking in C++*, 2nd edition, Prentice Hall, 2000 (http://www.bruceeckel.com).

[15] D. E. Knuth, *The TEXbook*, Addison-Wesley, 1986.

[16] L. Lamport, *LATEX - A Document Preparation System*, 2nd edition, Addison-Wesley, 1994.

[17] M. Goossens, F. Mittelbach, A. Samarin, *The LATEX Companion*, Addison-Wesley, 1994.

[18] M. Goossens, F. Mittelbach, S. Rahtz, *The LATEX Graphics Companion*, Addison-Wesley, 1997.

[19] T. Oetiker, H. Partl, I. Hyna, E. Schlegl, *The not so short Introduction to LATEX 2ε* di cui esiste anche la traduzione italiana: *Una (mica tanto) breve introduzione a LATEX 2ε*, 2000 (ftp://ftp.uniroma2.it/TeX/info/lshort/).

[20] L. Pantieri, *L'arte di scrivere con LATEX, un'introduzione a LATEX 2ε*, 2009 (http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf).

Programmi/Altro

[21] Richard M. Stallman e Roland McGrath, *The GNU Make Manual* (`"info make"`).

[22] *The VIM Book*.

[23] Richard M. Stallman, *GNU Emacs Manual*, 15th edition, Free Software Foundation Inc, 2002 (`"info emacs"`).

[24] Phil Zimmermann, *An Introduction to Cryptography*, ??

Siti Internet

- The Linux Documentation Project http://www.tldp.org
- Linux Information Portal http://yolinux.com/
- Vari Linux-How-To http://www.ibiblio.org/pub/Linux
- Piccole Guide d'Informatica http://www.webalice.it/lgcrsll/index.html
- Piccole Guide Linux http://www.webalice.it/lgcrsll/pgl/index.html
- Pluto http://www.pluto.linux.it
- Pagina del progetto VIM http://www.vim.org
- The Comprehensive Perl Archive Network http://www.cpan.org
- The LATEX Project http://www.latex-project.org

- The Comprehensive TEX Archive Network http://www.ctan.org
- TEX User Group Home Page http://www.tug.org
- Gruppo Utilizzatori Italiani di TEX http://www.guit.it
- Guide di Html/Css/PHP ... http://www.html.it
- Software Open Source:
  - ⋆ http://freshmeat.net
  - ⋆ http://sourceforge.net
- Se cercate qualsiasi altra cosa ... http://www.google.it

Indice analitico

Simboli

~ 13
.. .. 10
. 9
: 18
; 18
* 13
/ 9, 27
? 13
#! 11
$( ) 17
$(( )) 16
$? 40
$# 40
$$ 40
${ } 14
$! 40
$[ ] 16
$ 14
& 18, 48
{ } 14
| 18
! 16, 18
" 15
, ... 15
< 18
> 18
[ ] 18, 42
[[ ]] 18, 42
\n 19
\ 17
^ 16
` 17

A

alias 47
alpine 82
amarok 91
awk 38

B

bash 8
~/.bash_logout 53
~/.bash_profile 52
~/.bashrc 52
bc 39
bg 22, 48
blockdev 35
boot 73
break 45

C

cal 73
case 42
cat 9, 33
cd 8
cdda2wav 91
cdparanoia 91
cdrecord 78
CFLAGS 69
chattr 25
chmod 24
chown 25
cmp 38
~/.coloursrc 53
colrm 36
column 36
COLUMNS 40
complete 54
configure 74
continue 45
cp 8
csplit 36
cups 77
cut 36

D

date 72
dd 78
declare 47
/dev 28
/dev/dsp 34
/dev/full 34
/dev/null 34
/dev/random 34
/dev/urandom 34
/dev/zero 34
diff 38

dircolors 53
disown 48
DISPLAY 86
display 91
do 45
done 45

E

echo 9
elif 42
else 42
enable 47
esac 42
esd 92
/etc/bashrc 51
/etc/DIR_COLORS 54
/etc/fstab 29
/etc/group 27, 72
/etc/hosts 27
/etc/init.d 73
/etc/inittab 73
/etc/issue 27, 72
/etc/issue.net 72
/etc/ld.so.conf 69
/etc/motd 72
/etc/mtab 30
/etc/passwd 27, 72, 84
/etc/profile 51
/etc/protocols 82
/etc/rc 73
/etc/rc.d 73
/etc/resolv.conf 81
/etc/shadow 27, 72, 84
/etc/sudoers 72
eval 39
exec 25
expect 68
export 39
ext2 28
ext3 28

F

f77 69
false 42
fetchmail 82
fg 22, 48
fi 42
filesystem 23
find 31
finger 82
firefox 82
firewall 84
fmt 36
fold 36
for 45
fpc 69
ftp 82
ftpd 83
fuser 50

G

g++ 69
gcc 69
getopts 39
gimp 91
gnuplot 107
gpg 78, 85
gqview 91
GREP_COLOR 40, 54
groff 102

H

head 36
HISTCMD 41
HISTCONTROL 41
HISTFILE 41
HISTFILESIZE 41
HISTIGNORE 41
HISTSIZE 41
HISTTIMEFORMAT 41
HOME 9, 39
HOSTFILE 41
httpd 83

I

ical 90
if 42
ifconfig 81
IFS 31, 40
INFOPATH 40
init 73
~/.inputrc 55

J

jobs 48

join 36

K

kill 48
killall 49
ksh 8

L

lame 91
ld 69
LD_LIBRARY_PATH 39, 69
less 10
let 39
links 82
loadkeys 77
local 39
login 72
lpd 77
lpq 77
lpr 77
ls 8
LS_COLORS 40, 47
lsattr 25
lynx 82

M

mail 82
make 74, 98
Makefile 98
man 9, 102
man2html 103
MANPATH 40
md5sum 78
mingetty 73
mkdir 9
mkfifo 34
mkfontdir 88
mkfs.ext2 28
mkfs.vfat 28
mkisofs 78
mknod 35
mkswap 29
more 10
mount 28
mp3info 91
mpage 94
mplayer 92
mpstat 82
msgfmt 76
mutt 82
mv 9

N

netstat 81
nice 48
nl 37
nohup 48
ntfs 28

O

OLDPWD 40
opera 82

P

passwd 72
paste 37
patch 74
PATH 39
pidof 50
pine 82
ping 82
pop 82
PPID 41
pr 37
print-pipe 88
/proc 50
prompt 8
PROMPT_COMMAND 41
ps 48
PS1 ..... 8, 41, 54
PS2 41
ps2ps 94
PS3 41
PS4 41
psbook 94
psmerge 94
psnup 94
pstree 48
PWD 40
pwd 9

Q

qiv 91
quota 72

R

RANDOM 41
read 39
reiser-fs 28
renice 48
REPLY 41
return 45
rev 37
rm 9
root 72
rxvt 90

S

screen 104
SECONDS 41
sed 38
sendmail 82, 83
set 39
sha1sum 78, 84
shell 8
shift 39
shopt 54
skill 50
snice 48
snoop 82
sort 37
source 25
sox 91
split 37
ssh 82
startx 86
strace 50
su 72
sudo 72
swap 29

T

tac 37
tail 37
tcsh 8
tee 38
telnet 82
test 42
TEXTDOMAIN 76
TEXTDOMAINDIR 76
then 42
TMOUT 41
/tmp 28
top 48
totem 92
tput 71
tr 38
traceroute 81
trap 50
true 42, 45
truss 50
typeset 39

U

umask 25, 54
umount 29
unalias 54
unset 47
USER 39

V

vfat 28
vim 105
vlc 92

W

wait 50
wc 38
wget 82
whereis 30
which 30
while 45
whois 82
wish 68, 90
wmagnify 90

X

X 86
xargs 31, 39
~/.Xauthority 88
xclock 90
~/.Xdefaults 88
xfig 91
xhost 88
xine 92
xinerama 90
xinetd 77
xinit 86
~/.xinitrc 89
xkill 90
xli 88

xlock 90
xlsclients 88
xlsfonts 88
xmag 90
xmessage 90
xmms 91
~/.Xmodmap 88
xon 90
xprop 90
xrandr 88
xrdb 88
~/.Xresources 88
xscreensaver 90
~/.xsession 88
xset 88
xsetbg 88
xsetroot 88
xterm 90
xv 91
xwininfo 90

Z

zenity 93