
Little John • Manhattan • Shodan • Tx0
Corso
Base
di
UNIX

DISCLAIMER:

QUESTO CORSO DI UNIX VIENE PENSATO, SCRITTO E STAMPATO AD USO E CONSUMO DI CHIUNQUE VOGLIA IMPARARE A CONOSCERE I SISTEMI OPERATIVI UNIX . GLI AUTORI SI RISERVANO IL DIRITTO DI MODIFICARE IL TESTO IN QUALSIASI MOMENTO PER AGGIORNARNE, MODIFICARNE OD ESPANDERNE IL CONTENUTO.

È CONSENTITA ED INCORAGGIATA LA STAMPA DI QUESTO MANUALE PER SCOPO AUTOFORMATIVO, SENZA FINI DI LUCRO.

NON È CONSENTITO IL PASSAGGIO DI QUESTO CORSO A TERZI QUALORA SPROVVISTO DEI RICONOSCIMENTI AGLI AUTORI ED AL LOA HACKLAB DI MILANO NONCHÉ DELLE INFORMAZIONI SULLA REPERIBILITÀ DEL MANUALE AGGIORNATO IN RETE.

QUESTO TESTO È FORNITO "AS IS". GLI AUTORI NON SI ASSUMONO ALCUNA RESPONSABILITÀ SULLE CONSEGUENZE CHE L'USO DELLE TECNICHE E DELLE CONOSCENZE IN ESSO RIPORTATE POSSA CAUSARE.

Piccola Prefazione

Con questo corso si intende dare una visione complessiva dei comandi e delle strutture base dei sistemi operativi della "famiglia Unix".

Partendo da "cos'è un sistema Unix?" fino ad arrivare alla grafica e alle finestre, si cercherà di esaminare tutti gli aspetti dell'utenza Unix, fino a, si spera, creare degli utenti consapevoli, in grado di gestire i propri file, il proprio ambiente e il proprio lavoro in maniera autonoma ed efficiente.

Questo non è un corso per amministratori di sistema o per imparare a "bucare" le macchine altrui. Questo corso e' rivolto a tutte le persone che vengono messe davanti ad un terminale Unix senza che ne sappiano nulla; questo corso è inoltre rivolto a chi vuole imparare qualcosa di nuovo e di diverso dalle solite finestre di Windows e, magari, avere un riconoscimento professionale. Concludendo, con questo corso si intende dare una visione differente dell'informatica imparando ad usare un computer in modo consapevole, senza la mediazione di decorazioni o fronzoli che spesso non danno la possibilità di vedere cosa sta succedendo veramente.

Gli autori

Riferimenti Utili

Web LOA: *http://www.autistici.org/loa/*

Web Dispense Corso: *http://www.autistici.org/loa/web/doc/corsounix.ps*
http://www.autistici.org/loa/web/doc/corsounix.pdf
http://www.autistici.org/loa/web/doc/corsounix.tar.bz2

Email:
<littlejohn@autistici.org>
<manhattan@paranoici.org>
<shodan@autistici.org>
<tx0@autistici.org>

Versione: 1.0 pre1
Ricompilato in data September 22, 2002

Nota sul software utilizzato:

Per la realizzazione di questo testo è stato utilizzato solo Free Software. Le operazioni si composizione del testo sono state eseguite su sistemi **Linux** (http://www.linux.org/, http://www.linux.com/). Per la scrittura del libro gli autori hanno utilizzato **Vim** (http://www.vim.org/) e **Emacs** (http://www.gnu.org/software/emacs/). Per la formattazione del Corso è stato impiegato esclusivamente **TEX** (http://www.tug.org/) con l'ausilio del macropackage **ConTEXt** (http://www.pragma-ade.nl/). Per la realizzazione dei dettagli grafici, della copertina, delle icone è stato impiegato **The Gimp** (http://www.gimp.org/).

Nota sul bollino SIAE:

Il LOA HackLab porta avanti da anni la sua battaglia contro la SIAE, a favore della libera circolazioni dei saperi. Coerentemente con le scelte fatte in passato, pubblichiamo anche su questo Corso il bollino no-siae.

L'esistenza di un organismo con funzioni di controllo poliziesco al quale è obbligatorio iscriversi e registrare qualsiasi tipo di pubblicazione destinata al mercato editoriale nel suo senso più vasto, anche nel caso in cui non si intenda tutelare economicamente il proprio "diritto d'autore", è una delle principali cause della difficoltà di circolazione dell'informazione e della conoscenza.

Questa pubblicazione non si attiene a queste regole, preferendo la coerenza politica alla legalità (complicità).

Convenzioni adottate

Nel corso del testo troverete numerosi comandi UNIX trattati. Abbiamo adottato la seguente convenzione: un comando UNIX nel testo è indicato come `ls -l`. Un sezione riguardante la sintassi di un comando o il contenuto di un file è:

```
gzip [OPZIONI] file1 [ file2 [ fileN ] ]
```

Uno screenshot di una operazione eseguita a video invece viene indicato:

```
$ cat /etc/motd
Linux gromit 2.2.19-ok #2 Thu Feb 7 17:54:31 CET 2002 i686 unknown

Most of the programs included with the Debian GNU/Linux system are
freely redistributable; the exact distribution terms for each program
are described in the individual files in /usr/share/doc/*/copyright

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
$
```

Incontrerete anche alcuni riquadri particolari:

Note:

Vi informano circa un aspetto di un programma o di un comando del quale fareste bene a prendervi nota

Avvertimenti:

Se state per fare quello che è indicato nel riquadro vi state cacciando in un guaio. E noi vi spieghiamo perché

Suggerimenti:

Il riquadro vi spiega qual'è il modo migliore per raggiungere un determinato obiettivo

Indice dei Contenti

Cippalippa code, ovvero un bieco esempio di come dovranno funzionare le pagine in pillola

Questa e' una prova di pillola!!!
Questa e' ancora dentro???

E ora come la mettiamo???

12

Il concetto di Kernel

12

1. # history_

di Tx0

Sembra incredibile, ma anche UNIX non esiste da sempre. Vediamo come è nato. . .

Unix nasce nei primi anni ’70 ad opera del colosso delle telecomunicazioni americane AT&T. Il sistema doveva essere dedicato ai tecnici, ai programmatori, agli sviluppatori, a coloro che lavoravano con i sistemi per creare altri sistemi. Per questo all’utente odierno (abituato al *paradigma grafico* dell’interfaccia all’utente) la *shell* con tutti quei comandi pieni di dozzine di opzioni risulta così ostica.

Per capire gli orientamenti che hanno portato alla creazione di **questo** Sistema Operativo è necessario riflettere sulla situazione dell’*hardware* dell’epoca. I computer erano principalmente grossi *mainframe* (per il primo *personal* occorre attendere altri dieci anni) costituiti da una singola unità centrale e un gruppo di terminali connessi ad essa tramite link seriali a bassissime velocità (anche 75 o 300 bit per secondo). Con una simile *architettura* non aveva senso parlare di interfacce grafiche, quindi tutto il lavoro sulla macchina era realizzato attraverso uno nutrito insieme di comandi dai nomi impronunciabili.[1]

Le prime versioni di UNIX comprendevano già tutte le funzioni di multiutenza e multiprocesso che saranno proprie di tutte le successive. L’interazione con il sistema avveniva solo ed esclusivamente attraverso una *shell* (letteralmente *conchiglia* ma correntemente tradotto in italiano come *interprete comandi*) con limitate funzionalità di supporto della scrittura dei comandi.

Dopo un periodo di uso limitato da parte della AT&T, la comunità informatica comincia ad apprezzare UNIX e le sue caratteristiche più rivoluzionarie rispetto al passato. La più sconvolgente delle quali è la possibilità di ottenere i sorgenti in C del Sistema Operativo per cifre contenute dalla stessa AT&T.

Diversi produttori di hardware (*Sun Microsystems*, *Hewlett Packard*, *IBM*, *Digital Equipment*, *Silicon Graphics*) cominciano a scrivere le rispettive versioni di UNIX (*SunOS*, poi diventato *Solaris*, *HP-UX*, *AIX*, *Digital UNIX*, *IRIX*) per offrire allo stesso tempo un Sistema Operativo standard ma aggiunto di particolari features che gli consentano di affermarsi sugli altri UNIX.[2]

[1] Anche il nome dei comandi ha una ragione di essere nella velocità dei link: oggi sembra assurdo ma era *sensibilmente* più veloce scrivere `cp` che non `copy` o `mv` piuttosto che `move`, perché quello che veniva digitato doveva viaggiare lungo la linea e tornare indietro, i caratteri comparivano davvero uno alla volta!

Questi anni sono caratterizzati della guerra fra i *flavour* (versioni) di UNIX. Negli anni '80 il Nemico (*Microsoft*) effettivamente ancora non esisteva, essendo troppo impegnata nella conquista dell'*home computing* per occuparsi di Sistemi Veri;[3] anche se degno di nota è rilevare come la stessa Microsoft avesse acquisito i diritti di Xenix (flavour di UNIX) per poi rivenderlo in blocco così come era entrato in casa alla *Santa Cruz Operating* (per gli amici *SCO*) senza mai pianificare uno straccio di strategia di mercato su questo materiale. (Oggi *SCO UNIX* è uno dei più importanti flavour di UNIX presenti sul mercato al pari di Solaris e AIX, mentre Xenix ha concluso il suo ciclo vitale da quasi dieci anni).

Intanto l'Università di Berkeley sta scrivendo la propria versione di UNIX. E questo non stupisce affatto dato che le Università all'epoca erano uno dei pochi soggetti che potevano permettersi ricerca seria e libera in campo informatico, mentre i pesci piccoli ancora non avevano a disposizione una piattaforma diffusa e standard per realizzare software di basso costo.

Berkeley Software Distribution (più noto come *BSD*) rappresenta molto nella storia di UNIX: è il primo e l'ultimo flavour che tenta il distacco dall'architettura *System V* concepita da AT&T! A parte BSD nessuno batterà più nuove vie; i sistemi cominceranno a conformarsi a BSD oppure resteranno fedeli a System V. Una terza via non sarà mai cercata.

BSD soprattutto ha significato:

- Nuove chiamate al Kernel[4]
- Nuova concezione dei processi di boot[5], configurazione dei servizi, gestione di account, dischi e dispositivi
- Nuovi comandi

In dettaglio questo comporta:

- i programmi scritti per System V avevano bisogno di modifiche per compilare ed eseguire sotto BSD e viceversa, con ovvio disagio dei programmatori che si trovavano a dover scrivere due versioni del programma quando prima ne occorreva solo una (e

[2] Basti pensare che Digital (che aveva già **VMS** per la sua serie di *VAX Station*) ha deciso di passare a UNIX, per comprendere quanto questo Sistema Operativo sia stato dirompente sulla scena.

[3] Si questo corso è fazioso, lo sappiamo e ne siamo contenti :-)

[4] Il Kernel è il nucleo del Sistema Operativo deputato alla esecuzione dei compiti di più basso livello; costituisce il fondamento di tutto il Sistema

[5] Il boot è la fase di avvio del Sistema Operativo durante la quale non solo viene caricato in memoria il Kernel, ma viene anche eseguita la procedura di configurazione di partenza e vengono eseguiti gli script che lanciano i singoli servizi disponibili sulla macchina

questo è un elemento fortemente negativo per UNIX che aveva sempre privilegiato la *portabilità* del software fra differenti piattaforme).

- una volta scritto un programma questo doveva essere in grado di installare correttamente sul sistema; ma tra i due rami non c'è più la compatibilità necessaria sul formato, il nome, la posizione e la sequenza in chiamata degli *script* di boot. I nomi e le dislocazioni delle directory fondamentali del sistema seguono criteri non comuni. Anche qui è necessaria una specializzazione del software.

- I comandi stessi[6] non sono più uniformi. Anche per l'utente (che comincia a non coincidere più univocamente con lo sviluppatore, ma è anche lo scrittore piuttosto ché il matematico o l'ingegnere nucleare) insorgono le prime forme di disagio quando deve passare da un sistema SysV a uno BSD.

È notevole vedere come BSD intanto abbia già influenzato i sistemi proprietari dei costruttori di hardware. Ad esempio Sun sceglie BSD per il suo SunOS (si ricrederà poi negli anni in favore di SysV quando il sistema cambia il suo nome in Solaris). IBM impronta il suo AIX sempre sul modello si BSD.

A questo punto la storia di UNIX si intreccia intimamente con la storia di un altro importante software: la suite di protocolli di telecomunicazione *TCP/IP*.

La vecchia rete di comunicazione statunitense (ArpaNET) era in grado di accogliere solo 256 computer a livello globale, ed era basata su un protocollo in via di superamento in favore del nuovo *TCP/IP*. Tuttavia un forte limite alla transizione era costituito dall'esiguo numero di Sistemi Operativi già abilitati a "parlare" il *TCP/IP*.

All'Università di Berkeley viene così commissionata l'implementazione dello *stack TCP/IP* sul suo BSD UNIX. Lo stack (letteralmente *pila, serie ordinata*)[7] è la parte del Kernel del Sistema Operativo deputata alla gestione dei pacchetti[8].

Da allora il binomio UNIX — *TCP/IP* sarà indissolubile. Oggi non esiste un singolo flavour che non abbia uno stack TCP/IP disponibile, soprattutto nell'era di *Internet*[9].

Quando questo sodalizio si consuma, sulla piazza UNIX è la sola scelta possibile per la gestione di server. Gioco forza, Internet si forma a immagine e somiglianza di UNIX.[10] I software UNIX si diffondono come standard senza dettare standard. Diviene abituale

---

6 Oltre un certo livello di complessità, s'intente; `cp` è sempre `cp`, ma non serve andare troppo oltre per rilevare differenze

7 Il nome deriva dal fatto che ciascun pacchetto di dati attraversa diversi *layer* (strati) ciascuno dei quali opera delle modifiche o delle scelte sul pacchetto inerenti i vari aspetti del suo invio.

8 Un *pacchetto* costituisce l'unità in cui viene suddiviso l'insieme dei dati da inviare

9 ...che usa TCP/IP, che è figlio di UNIX, che sta riguadagnando terreno proprio grazie ad Internet!

10 Per citare un dettaglio minore ma significativo, gli URL usano lo *slash* ('/') e non il *backslash* ('\') per separare le directory, come usa UNIX e non DOS/Windows

risolvere i problemi di fornitura di un servizio tramite programmi UNIX. Le alternative faticano ad arrivare.

Passati gli Anni '80 di cose ne succedono. Microsoft ha partorito *WindowsNT* e il suo *DOS + Windows 3.1* ha preso piede diffusamente nelle case e negli uffici. I *personal computer* sono alla portata del pubblico più ampio. Ormai l'informatica **è** un fenomeno di massa.

E questo è un serio problema.

Eh sì! Perché UNIX non è più standard come quindici anni prima, perché le case sono in guerra fra loro per il predominio di un mercato tipicamente UNIX che in realtà WindowsNT erode giorno dopo giorno conquistando affezionati fra quelli che non vogliono essere costretti ad imparare ed a ragionare per mettere in piedi un Server WEB, ma si accontentano di un click!

Sun, HP, IBM, Digital e SCO[11] entrano negli anni '90 convinti che il mercato sia ancora completamente di UNIX e che quindi l'obiettivo rimanga quello di conquistare fra loro fette di questo mercato. Non realizzano invece subito che Microsoft ha lavorato sull'unica parte della *Macchina* che loro non hanno mai vezzeggiato: *l'utente*! Nonostante nei primi anni novanta tutti i flavour citati di UNIX abbiano già sviluppato una comune interfaccia grafica[12], i sistemi di Microsoft devolvono a questa **tutta** la gestione degli aspetti del sistema, trasformandolo in qualcosa di molto simile ad una scatola di mattoncini Lego[13]. Questa innovativa intuitività garantita da mouse, icone e doppi click consente anche a chi ha conoscenza zero della gestione di un server di realizzare installazioni con minimo sforzo e senza dovere ampliare notevolmente il proprio bagaglio di nozioni.

Parallelamente a questi mutamenti radicali sulla scena mondiale stanno avvenendo nuovi eventi che costituiranno la salvezza dei sistemi UNIX dalla conquista di Microsoft e dal morbo della *proprietarizzazione*[14].

Nel 1990 Linus Torvalds, studente universitario finlandese, rilascia la release 0.01 di *Linux*. Il progetto di Torvalds è quello di creare (senza nessuna pretesa di completezza) un Kernel UNIX. Prima di lui Andrew Tanenbaum, docente universitario, aveva compiuto

[11] Escludiamo da questo elenco Silicon Graphics perché il mercato della grafica verrà attaccato da Microsoft più tardi e quindi *SGI* avrà modo di arrivare più preparata allo scontro

[12] Stiamo parlando di *X Window System*. La release 4 risale ai primi anni '80, successivamente evoluta nella 5 ed infine nella 6. Attualmente la release 6.3 è lo *stato dell'arte*

[13] Stiamo semplificando. Gestire un intero sistema con un mouse è più semplice per il neofita, non per l'amministratore esperto che guadagna tempo e flessibilità dalla shell. Ma è proprio l'utente inesperto che rivoluzionerà il mercato anche nel campo dei server di fascia alta

[14] Ossia dal divenire sempre più *proprietà* progettuale di una singola azienda, divergendo inesorabilmente da uno standard comune

la stessa impresa realizzando *Minix*, un piccolo (come suggerisce il nome) UNIX a scopo didattico e dimostrativo.

Intanto da BSD si stacca un filone di sviluppo che dà origine a FreeBSD. Il nome dice già che il proposito del sistema è quello di creare una versione di BSD liberamente distribuibile e rispecchiante il più possibile la versione ufficiale di BSD.

Linux e FreeBSD hanno in comune molto. Diventano rapidamente soggetto di sviluppo collettivo. Non sono ristretti da copyright castranti e possono quindi circolare anche in versione sorgente. Hanno come obiettivo la realizzazione del miglior UNIX possibile, che non è frutto utopico di menti visionarie e avulse dalla realtà del mercato, ma il concreto tentativo di collezionare le migliori caratteristiche progettuali ed implementative dei flavour esistenti e convogliarle in un unico sistema il più condivisibile possibile. Ma hanno alle spalle un modello di sviluppo differente. Linux non chiude a priori il numero di sviluppatori che possono lavorare al Kernel. FreeBSD delinea un gruppo di sviluppatori e con quello evolve.

Tuttavia i due sistemi sono *la cosa giusta al momento giusto*. E il processo di sviluppo prende sempre più piede. Tanto che Linux nel 1994 è già quasi pronto al battesimo della versione 1.0 e costituisce un sistema stabile e tutt'altro che minimale (diversamente dai più sostenibili intenti iniziali di Torvalds, che di sicuro non si attendeva di riscuotere un simile interesse dalla comunità mondiale di Internet).

Il secondo evento del momento si scrive *Richard Stallmann*, si pronuncia *Free Software Foundation* ma se provate a disegnarlo ha la forma di uno *GNU*[15].

Stallmann, fresco di una Laurea in Fisica, lascia l'Università per fondare la *FSF* con lo scopo di realizzare un sistema UNIX interamente *free*. Il suo concetto di *libero* è tendenzialmente dieci volte più esteso di quello che il comune utente attribuisce alla parola, ossia *gratis* ! La gratuità del software è indubbiamente uno dei nodi centrali del pensiero di Stallmann ma non ne è il principale.

Perché un sistema sia free è innanzi tutto necessario che non sia coperto da diritti che ne vincolino la circolazione (quindi la possibilità di copia e riproduzione), la facoltà di esaminare e modificare i sorgenti del programma e l'impegno del software a non invertire queste tendenze *mai* ! La gratuità del software ovviamente discende come conseguenza da questi due precedenti postulati.

Aberrante per Stallmann è l'idea che un gruppo limitato possa imporre degli standard su sistemi, formati o peggio ancora modalità di elaborazione dell'informazione. Per questo il prodotto culturalmente più rilevante della FSF è la *General Public License* (o *GPL*): un documento che permette di proteggere il frutto della propria programmazione con una licenza in grado di tutelare la paternità dell'autore e di garantire al contempo la libera circolazione del software.

Programmatore prima che politico, Stallmann inizia la scrittura del flavour UNIX di FSF. Seguendo un paradosso solo apparente, il primo pezzo del sistema (qui inteso nel senso più completo del termine, includendo anche i tool e il software di base, in questo più simile a BSD che non a Linux come progetto) è il compilatore C `gcc`.

15 GNU è un acronimo ricorsivo per *"GNU is Not Unix"*

Il paradosso non sussiste in quanto la disponibilità di un Kernel come Linux già oggetto di diffusso ed accurato sviluppo da parte di un'altra comunità consente di arrivare più rapidamente ad un sistema UNIX completo, lasciando lo sviluppo di un Kernel come ultima tappa[16]. Per contro Linux viene presto licenziato sotto la GPL, a dimostrazione che gli intenti delle due comunità sono coincidenti.

Il terzo evento (figlio evidente dei primi due e di una maturata cognizione della perdita di presa di UNIX sul mercato) è la creazione dello standard *POSIX*. I produttori di UNIX si impegnano nella definizione di uno standard di sistema che unifichi le chiamate al Kernel e le librerie fondamentali per riguadagnare compatibilità e portabilità fra i singoli flavour. POSIX è un standard *in progress* che continua a maturare con l'evolversi delle esigenze e finora ha portato ad una sostanziale convergenza dei sistemi. Linux stesso ha fatto proprio l'obiettivo di conformarsi a questo standard.

I frutti più immediati di questi sviluppi sono già oggi visibili. Un numero crescente di produttori stanno supportando Linux e la sua filosofia rilasciando i propri sistemi gratuitamente (anche se non free), dimostrando di capire che l'obiettivo è creare una piattaforma di utenza sempre più vasta per UNIX, sapendo di avere rientri economici dalla vendita dell'hardware[17]. Dimostrano anche di sentire l'esigenza di un fronte comune per contrastare il dilagante avanzamento di Microsoft che già oggi conta numerose sconfitte sul fronte delle vendite e delle installazione di pacchetti software. Basti pensare che il 60% dei server Web al mondo è *Apache* e non *IIS* e che sendmail è il più diffuso *mail server* del pianeta con l'80% delle installazioni.

Il prossimo obiettivo è la conquista del *desktop*. UNIX ha sofferto sin dai primi anni '90 del predominio di Windows sui *PC* e quindi nel settore di mercato di base. Questo sia per la diffusione a tutti i livelli di Windows sia per la carenza di una interfaccia grafica sufficientemente completa da contrastare l'assistenza totalizzante che il desktop di Windows dà al proprio utente.

Gli sforzi più sensibili di proporre un'alternativa per UNIX viene da tre maggiori progetti:

16 È solo ultimamente infatti che FSF ha iniziato a sviluppare il suo Kernel noto come *The HURD*

17 Va ricordato che tutti i nomi citati sin qui, ad esclusione di SCO, sono principalmente produttori di *workstation* e di *server* e che i loro flavour UNIX sono solo un supporto alla macchina e non un prodotto a parte. Così Sun rilascia Solaris a prezzo del solo supporto e dona attrezzature *Sparc* agli sviluppatori Linux, così come numerose sono le donazioni di Digital (ora comprata da Compaq), mentre IBM sviluppa sempre più prodotti free o disponibili in binario per Linux. Per contro Linux è ormai stato portato su tutti i processori Sparc, Risc 6000 e PowerPC, Alpha e PA/RISC

- CDE
- KDE
- Gnome

CDE (o *Common Desktop Environment* per esteso) è il frutto del lavoro di diversi produttori (Sun, IBM, HP e altri) di standardizzare un desktop basato su *Motif* (lo storico *toolkit* di UNIX) ed è già oggi disponibile per diversi flavour.

KDE è il primo progetto apparso di desktop evoluto per Linux e UNIX in genere. Consente una più semplice gestione delle finestre e delle applicazioni, automatizza molte procedure e consente una seplice ma efficace configurazione dell'ambiente grafico di lavoro.

Gnome è l'ultimo arrivato ma ha rapidamente guadagnato terreno (soprattutto per Linux). Riprendendo tutti gli obiettivi di KDE, aggiunge una più ambiziosa visione dell'ambiente di lavoro arrivando a definire (per ora solo in fase di sviluppo) *framework* per registrazioni audio a livelli professionali ed altre caratteristiche mai entrate a far parte di un desktop/ambiente UNIX.

La situazione attuale di Linux è composta da un numero notevole di *distribuzioni* del sistema. Dato che Linux (nel senso ampio) è il risultato dell'accorpamento di numerose componenti sviluppate da parti indipendenti sono nate organizzazioni con lo scopo di organizzare questo software, suddividerlo in *packages* e fornire una procedura uniforme di installazione. Fra queste possiamo citare le più note e diffuse:

- *Debian* Ha lo scopo di fornire un sistema UNIX completamente free, completo, SysV compliant, rigorosamente attenente agli standard, con l'impegno di non limitare mai nel futuro questi obiettivi. La versione attuale (2.2, code named Potato) è formata da 3 CD di binari + 2 CD di codice sorgente ottenibili dalla rete senza costo.
- *Red Hat* Distribuzione commerciale formata da una parte scaricabile dalla rete in binario e sorgente e da una parte ottenibile a pagamento, contenente anche pacchetti proprietari sviluppati da *Red Hat Inc.*. È SysV compliant.
- *Slackware* Distribuzione BSD fra le più antiche, incorpora tutti i software più diffusi. Ne esiste una versione estesa di più di 4 CD ottenibile a pagamento.

Per le motivazioni espresse sin qui e per altri motivi di gusto personale, chi scrive preferisce Debian e la sua natura completamente free.

La sfida è aperta. Ma i segnali che arrivano da numerosi produttori di software private come *Corel* e *IBM* portano a prevedere un rapido potenziamento almeno degli UNIX free. Comunque sia il dominio di Microsoft è sempre più debole.

E poi Windows2000 non gira su un 486, Linux 2.4 si!

2. Qualche informazione su UNIX

di Shodan

Per utilizzare al meglio `UNIX` è necessario familiarizzare con alcuni concetti fondamentali: sistema operativo , kernel , multitasking , multiutenza .

2.1 Cos'è un sistema operativo

Con il termine "sistema operativo" si intende quel software che gestisce l'hardware di un computer e che mette quest'ultimo in comunicazione con gli altri programmi. Inoltre permette agli utenti di accedere alle risorse della macchina. Tutti i computer, per funzionare, devono avere un sistema operativo.

2.2 Il Kernel

Il Kernel è il cuore del sistema operativo Unix e viene caricato in memoria all'avvio del computer, questa operazione viene chiamata anche **boot**. Esso gestisce tutte le risorse di un computer e le presenta a tutti gli utenti come un sistema coerente.

2.3 Unix è multitasking

Una traduzione letterale della parola *multitasking* potrebbe essere *multiobbiettivo*. Questo termine si applica ad un sistema operativo che è in grado di gestire più *task* (obbiettivi) contemporaneamente.

A differenza del DOS o di altri vecchi sistemi operativi, Unix è in grado di far funzionare contemporaneamente più programmi contemporaneamente e di fare in modo che questi ultimi, dove necessario, si scambino informazioni. Questa caratteristica permette un utilizzo migliore delle risorse e quindi una velocizzazione dei vari compiti.

Per fare un esempio pratico, con Unix è possibile leggere la posta, mentre si scaricano dei file dalla rete, mentre un programma di rendering 3D calcola un'immagine e cosi' via...

L'unico limite al numero di processi (questo è il nome dei vari task) è dato dalla memoria e dalla dimensione della *process table*; un utente normale di solito non riesce a riempire la prima, tanto meno la seconda.

2.4 Unix è multiutente

Qui è un po' più facile di prima: con *multiutente* si intende un sistema operativo che può ospitare processi di più persone contemporaneamente senza "confondersi" tra i vari utenti. L'importanza di questa caratteristica è che così si permette a più persone di lavorare simultaneamente sullo stesso computer. Inoltre questo permette ad ogni singolo utente di avere il proprio spazio personale su disco. Unendo queste due caratteristiche potremmo immaginare che le cose che vengono fatte nell'esempio precedente, possono essere fatte da più persone in contemporanea.

3. Entriamo nel sistema

di Shodan e Tx0

Impariamo a muoverci nel sistema, a cambiare la directory corrente, a copiare un file, a cambiarne il proprietario e i permessi di accesso...

Si ringrazia icemaze per la stesura dei paragrafi indicati con (*)

3.1 Cos'è un account

L'*account* possiamo dire che è il "diritto ad accedere al computer" e viene dato solo dall'**Amministratore di Sistema**. Si può pensare all'account come ad un proprio ufficio dentro l'ambiente Unix; altri utenti hanno il loro ufficio e il lavoro fatto da ognuno non influenza quello degli altri. Come il proprio ufficio, l'account è personalizzabile in modo da garantire facilità di uso e di accesso al proprietario a seconda delle necessità personali. Più avanti vedremo come modificare le impostazioni dell'ambiente in cui lavoriamo. Ogni ufficio è chiuso a chiave, solo il proprietario può accedervi; questa chiave è la *password*, spesso viene scelta dall'utente, ma succede anche che venga data d'ufficio. La password è modificabile, e, per motivi di sicurezza, sarebbe meglio cambiarla ogni tanto, vedremo più avanti come.

Consigli per una password sicura 3.1.1

Più la password è complicata, più risulta difficile che qualcuno possa indovinarla. Generalmente è buona regola non usare parole comuni o nomi propri: molte persone usano come password il proprio nome o cognome, altri usano il nome del compagno/a, gatto, etc...

Un altra regola che sarebbe bene seguire è quella di mischiare vari tipi di carattere nella password in modo da renderla difficile da scovare: se noi mischiassimo le normali lettere minuscole con lettere maiuscole, numeri e caratteri non alfanumerici, avremmo un buon grado di sicurezza. Molti sistemi non permettono l'immissione di password troppo semplici.

Avere una password difficile è importante per tenere persone non autorizzate fuori dal proprio spazio di lavoro. Se dei maleintenzionati fossero in grado di penetrare nel sistema potrebbero fare danni ai dati del proprietario incauto e potrebbero pure danneggiare il sistema, quindi tutti gli utenti; questo perché è più facile causare problemi da dentro un sistema piuttosto che da fuori.

3.2 La procedura di login e logout

Una volta ottenuto l'account, procederemo ad utilizzarlo.

Dinnanzi a noi vediamo una scritta fatta così:

```
UNIX(r) System V Release x.y (host.domain.it)

login:
```

Questa è la richiesta da parte del sistema di immettere la *login*. La login è il nome che è stato scelto di dare all'account. Ogni login è unica nel sistema, non possono esistere due utenti con lo stesso nome. Una volta inserita la login ci troveremo davanti una cosa così:

```
UNIX(r) System V Release x.y (host.domain.it)

login: user1
Password:
```

A questo punto ci viene richiesta la password per etrare nel sistema. Per motivi di sicurezza la password non viene visualizzata a schermo quando la si digita, non viene neppure visualizzato il cursore dato che potrebbe dare un suggerimento sulla lunghezza della password. Sempre per motivi di sicurezza il sistema risponderà solo con un "Login incorrect" qualora si sbagliasse la login o la password; dare indicazioni in merito al tipo di errore (se si è sbagliata la login o la password) darebbe suggerimenti su quali utenti esistono e quali no.

Per eseguire la procedura di logout (ossia di abbandono del sistema, di disconnessione dal terminale[1], basta digitare `logout` (o in alcuni casi anche `exit`) oppure premere la combinazione di tasti `^d`.

3.3 Comandi di Base

Bene. Siamo entrati nel nostro account. Ora vediamo cosa fare.

Oggi è un bel giorno da ricordare: `date`, `cal` 3.3.1

Ci troviamo dinnanzi a una riga con dei caratteri e una cosa lampeggiante tipo questa:

```
UNIX(r) System V Release x.y (host.domain.it)

login: user1
Password:

Last login: Mon Oct 02 2000 12:33:06
Sun Microsystems Inc.   SunOS 5.5.1     Generic May 1996
You have mail.

%
```

Questo è quello che si chiama *prompt dei comandi*, il quadratino lampeggiante è il cursore che ci indica in quale posizione andremo ad immettere i comandi. Proviamo a immettere un comando: *date*

```
% date
Mon Oct  2 14:56:41 MET DST 2000
%
```

[1] o di fuga da questo mostro pieno di caratteri `:-)`

Questo è come funziona l'interfaccia testo: c'è un *prompt* che attende il comando, si inserisce il comando con eventuali parametri, con [INVIO] o [ENTER] si comunica al sistema che il comando è completo e che deve elaborare la nostra richiesta. Nel caso dell'esempio il comando in questa forma chiede al sistema la data e l'ora, il comando in questione, inoltre, ci comunica anche il fuso orario (MET DST).

Un altro comando interessante da vedere è `cal`. Senza parametri ci fornisce il calendario del mese e anno correnti, volendo si possono specificare altri mesi e altri anni.

```
$ cal

      May 2002
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

$
```

Dove sono finito: `hostname`, `uname` 3.3.2

Ora che abbiamo capito come inserire i comandi, procediamo a scoprire il mondo in cui siamo entrati; iniziamo con l'ottenere informazioni riguardo alla macchina sulla quale ci siamo loggati[2].

La prima informazione che cercheremo è il nome della macchina: ogni workstation Unix ha un nome che serve per distinguerla dalle altre; generalmente il nome in questione è quello che viene usato per identificare il computer in rete, ma esso esiste anche quando non c'è connessione in rete.

Per ottenere questa informazione si usa il comando *hostname*.

```
% hostname
bulk01
%
```

[2] Con *loggarsi* si intende in gergo l'eseguire la procedura di login, entrare nel sistema

Dove sono finito: `hostname`, `uname`

Come si può notare questa informazione comunica solo il nome del host, non quello del *dominio*[3], quest'ultimo, infatti non è cosa che riguardi il sistema per le sue funzioni normali se non per quelle inerenti la rete.

Un altro comando importante per ottenere informazioni riguardo alla workstation sulla quale stiamo lavorando è *uname*; questo comando ci può dare informazioni su:

- Sistema Opertivo (quale Unix si sta usando?)[4]
- Nome del computer (come con *hostname*
- Versione del S/O o del kernel (a volte con data di creazione)
- Tipo di architettura e/o modello di workstation

Questa volta inizieremo a vedere un po' le differenze che ci sono tra i vari Unix: se, per esempio, la nostra macchina fosse una macchina Linux su un processore x86[5] e dessimo il comando *uname* senza parametri otterremmo una cosa come questa:

```
% uname
Linux
%
```

Questo perché senza parametri *uname* assume che si stia chiedendo solo il nome dello Unix. Usando vari parametri si possono avere altre informazioni, per comodità useremo il parametro *-a*, che in questo caso significa "tutti i parametri insieme", ecco il risultato di tale operazione:

```
% uname -a
Linux copkiller 2.2.17 #1 Wed Sep 13 13:39:09 CEST 2000 i586 unknown
%
```

Questa volta abbiamo un po' più di dati, infatti oltre al nome dello Unix, ci viene dato il nome della macchina, la versione del kernel, data e ora di creazione del kernel e l'architettura di sistema. Ora vediamo gli output dello stesso comando su altri due Unix:

3 Se, per esempio fossimo loggati sulla macchina copkiller.autistici.org e usassimo il comando *hostname* la risposta sarebbe `copkiller`

4 Come detto in precedenza ci sono vari tipi di Unix, generalmente ogni architettura ha il suo, capita anche che ci siano più Unix per la stessa architettura

5 Con "x86" si intendono tutti i processori Intel e compatibili

Solaris:

```
% uname -a
SunOS copkiller 5.5.1 Generic_103640-29 sun4u sparc SUNW,Ultra-1
%
```

e *HP/UX*

```
% uname -a
HP-UX copkiller B.10.20 A 9000/777 2002963839 two-user license
%
```

Vediamo qui che le informazioni sono quasi le stesse: il nome dello Unix (SunOS o HP/UX), il nome della macchina e la versione del sistema operativo[6]; le notizie qui iniziano ad essere diverse anche di posizione: con Solaris[7] abbiamo in ordine il patchlevel, il tipo di processore, l'architettura di sistema e il modello della workstation; Con HP/UX, invece troviamo: il modello di workstation, il numero di identificazione della macchina[8] e il tipo di licenza del sistema operativo. Ovviamente alcune informazioni saranno utili solo all'amministratore di sistema, altre anche agli utenti: se per esempio un utente avesse scritto un programma e lo avesse compilato[9] su una macchina Solaris, non potrebbe far girare lo stesso eseguibile su un'HP/UX, si rende quindi necessario conoscere il tipo di macchina che si accinge ad usare.

◻ C'è nessuno?: `who`, `w` 3.3.3

Come detto in precedenza, Unix è un sistema operativo *multiutente*, potrebbe quindi esserci qualcun'altro insieme a noi sul sistema, per vedere se così è ci sono due comandi: *who* e *w*. Il primo di questi se usato con i parametri *am i* (in modo da diventare "who am i"), riporta una riga come questa:

6 Negli Unix commerciali il kernel non è distribuito gratuitamente come con Linux, quindi si indica la versione del sistema operativo e non quella del kernel

7 SunOS e Solaris sono entrambi nomi degli Unix della Sun, Solaris è stato adottato per motivi commerciali, SunOS è il nome storico

8 Il numero di identificazione, generalmente si usa come codice di autenticazione per le licenze di software commerciali

9 *Compilato*=tradotto da codice scritto dall'uomo a codice eseguibile dal computer

C'è nessuno?: `who`, `w`


```
% who am i
copkiller!user1    tty1  Oct 11 11.39 (copkiller)
%
```

L'output del comando ci dice le seguenti cose: la macchina sulla quale siamo loggati, il nome utente[10], il terminale, la data e ora in cui ci siamo loggati e la macchina dalla quale ci siamo loggati; in questo caso la macchina sulla quale siamo e quella da dove veniamo coincidono perché non abbiamo usato connessioni di rete o collegamenti simili che vedremo in seguito. Proviamo ora il comando senza parametri:

```
% who
user1    tty0    Oct  9 16:58 (copkiller)
user2    ttyp1   Oct  9 16:58 (automa)
%
```

In questo caso vediamo che oltre a noi (user1, da copkiller) c'è anche un altro utente (user2, da automa), le informazioni che vediamo qui riportate sono circa le stesse dell'esempio precendente.

L'altro comando che andremo a vedere è *w*. La funzione di quest'ultimo è simile a quella di *who*, ma ci vengono date più informazioni: oltre al logname di chi è collegato, ci otteniamo alcune notizie sullo stato della macchina e sugli altri utenti:

```
% w
  2:54pm  up 323 days,  1:55,  2 users,  load average: 0.17, 0.06, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
user2    ttyp0    copkiller.autist Mon 2am  2.43s  0.25s  0.25s  -tcsh
user1    tty0     copkiller.autist Mon 5pm  0.00s  0.61s  0.23s  w
%
```

Andando in ordine vediamo: nella prima riga, l'ora corrente, da quanti giorni e ore la macchina è accesa senza interruzioni e quanto è il *carico* della CPU[11], nella seconda una

[10] il nome utente e il nome della macchina sono separati da un !, questa è una vecchia notazione che veniva usata per distribuire la posta quando non esistevano le reti LAN, ma i computer erano collegati con collegamenti punto-punto (seriale, modem, etc...); la notazione di comune utilizzo oggi è utente@macchina.

legenda per quello che viene sotto, nelle restanti i dati sugli utenti: nome, tty, da dove sono collegati, da quando, da quanto tempo sono *idle*, cioé senza fare nulla, la quantità di tempo macchina usata da tutti i processi e relativi "figli" in quel terminale, la quantità di tempo macchina dei processi attivi e cosa sta facendo l'utente sulla data tty.

3.4 Elencare i file: `ls`

Diamo ora una prima occhiata ai file e a come si comportano e che proprietà hanno su Unix. Il comando per visualizzare una lista di file è *ls*, senza parametri ci darà una lista dei soli nomi dei file:

```
% ls
messaggio  prova
%
```

Per avere una lista dei file più dettagliata, cioé per scoprire che proprietà hanno i file bisogna usare il parametro *-l*, questo ci darà una lista in formato "lungo":

```
% ls -l
total 2
-rw-rw-r--   1 user1  users    72 Oct 12 09:52 messaggio
-rwxrwxr-x   1 user1  users     0 Oct 12 09:51 prova
%
```

Il primo campo indica lo stato del file: una **r** significa lo stato di *leggibilità*, una **w** quello di *scrivibilità* e una **x** quello di *eseguibilità*; ignorando il primo carattere (in questi casi un "-") che verrà visto più avanti, occupiamoci degli altri nove: dividiamo questi ultimi in tre gruppi da tre caratteri, questi indicheranno i vari stati (rwx) applicati, partendo da sinistra, al proprietario, al gruppo e a tutti gli altri utenti. Il numero subito dopo indica il

11 Con *carico* si intende il grado di occupazione del processore; con un carico ¡ 1 il processore riesce a gestire tutto il lavoro e ha dei cicli in cui non fa nulla, con carico = 1, tutti i cicli del processore sono utilizzati per i processi, con carico > 1 il lavoro è superiore alle capacità della CPU e i processi vengono fermati a rotazione. Questo vale dividendo il carico per il numero di processori.

numero di link, ma questo non ci interessa ora, i campi importanti sono qelli che seguono: indicano (da sinistra) rispettivamente l'utente e il gruppo proprietari del file. A questo punto è necessario notare che Unix ha due database, uno degli utenti e uno dei gruppi, ogni utente appartiene ad almeno un gruppo, si può anche cambiare gruppo al quale si appartiene come si può diventare un altro utente, ma questo lo vedremo in seguito. Gli ultimi tre campi sono rispettivamente, la dimensione in Byte del file, la data (e ora) dell'ultima modifica e il nome del file.

Se ci fossero dei file *nascosti* non li vedremmo con il semplice parametro -l, bisognerebbe aggiungere un "a" per poterli visualizzare:

```
% ls -la
total 18
drwxrwxr-x   2 user1  users  1024 Oct 12 14:15 .
drwxr-xr-x  11 root   sys    3072 Oct 12 10:07 ..
-rw-r--r--   1 user1  users   814 Oct 12 14:15 .cshrc
-rw-r--r--   1 user1  users   347 Oct 12 14:15 .exrc
-rw-r--r--   1 user1  users   341 Oct 12 14:15 .login
-rw-r--r--   1 user1  users   446 Oct 12 14:15 .profile
-rw-rw-r--   1 user1  users    72 Oct 12 09:52 messaggio
-rwxrwxr-x   1 user1  users     0 Oct 12 09:51 prova
%
```

Si può notare che i file nascosti sono quelli il cui nome inizia con un "."; la funzione dei file **.** e **..** sarà spiegata in seguito.

3.5 Visualizzare i file: `cat`, `more` e `less`

Abbiamo appreso come elencare i file contenuti in una o più directory. Ma come è possibile visionarne il contenuto.

Esistono diversi comandi che permettono di far questo. Il primo che vediamo è `cat`. La sua sintassi è:

```
cat nomefile
```

Niente di più semplice. Ad esempio per vedere il contenuto del file `/etc/passwd` usiamo:

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
man:x:6:100:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
identd:x:100:65534::/var/run/identd:/bin/false
telnetd:x:101:101::/usr/lib/telnetd:/bin/false
gdm:x:102:102:Gnome Display Manager:/var/lib/gdm:/bin/false
guest:x:1003:1003:Samba Guest account,,,:/home/guest:/bin/true
nobody:x:65534:65534:nobody:/home:/bin/sh
$
```

Semplice. E utile. Ci sono alcune circostanze in cui `cat` può essere impiegato. Ad esempio per ricostruire file divisi in più parti.[12] Tuttavia questo comando presenta alcune limitazioni. La più evidente è che se il numero di linee del file supera quello del terminale, vedremo scorrerci davanti agli occhi il file e potremo visionarne solo l'ultima parte.

Proviamo allora ad usare `more`:

```
$ more /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
man:x:6:100:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
--More--(65%)
```

All'ultima linea `more` ci informa che ha mostrato circa il 65% del file e che attende input per sapere come comportarsi. Premendo `[ENTER]` `more` avanza di una linea. Premendo `[SPACE]` avanza di una intera paginata.

12 Vedi il comando `split` nel capitolo "Puntiamo più in alto"

Visualizzare i file: cat, more e less

Se provate a premere h vi comparirà una schermata di aiuto con l'elenco dei comandi disponibili:

```
Most commands optionally preceded by integer argument k.  Defaults in brackets.
Star (*) indicates argument becomes new default.
-------------------------------------------------------------------------------
<space>                 Display next k lines of text [current screen size]
z                       Display next k lines of text [current screen size]*
<return>                Display next k lines of text [1]*
d or ctrl-D             Scroll k lines [current scroll size, initially 11]*
q or Q or <interrupt>   Exit from more
s                       Skip forward k lines of text [1]
f                       Skip forward k screenfuls of text [1]
b or ctrl-B             Skip backwards k screenfuls of text [1]
'                       Go to place where previous search started
=                       Display current line number
/<regular expression>   Search for kth occurrence of regular expression [1]
n                       Search for kth occurrence of last r.e [1]
!<cmd> or :!<cmd>       Execute <cmd> in a subshell
v                       Start up /usr/bin/vi at current line
ctrl-L                  Redraw screen
:n                      Go to kth next file [1]
:p                      Go to kth previous file [1]
:f                      Display current file name and line number
.                       Repeat previous command
-------------------------------------------------------------------------------
```

Ad esempio con la lettera v si chiama vi e lo si posiziona alla linea corrente del file.[13] Premendo il tasto / si può immettere un pattern di ricerca con la sintassi delle Regular Expression.[14]

more è uno dei comandi standard di UNIX. Lo troverete su qualsiasi sistema. GNU ha scritto un programma molto simile che si chiama less. Le funzioni basilari sono più o meno analoghe. Tuttavia se chiamate un help con la lettera h vi accorgerete che less è molto più ricco di funzioni.

13 Vedi il capitolo "Editor di testo"
14 Vedi il capitolo "Regular Expression"

```
                  SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.

  h  H                 Display this help.
  q  :q  Q  :Q  ZZ     Exit.
 ---------------------------------------------------------------------------

                           MOVING

  e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
  y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
  f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
  b  ^B  ESC-v      *  Backward one window (or N lines).
  z                 *  Forward  one window (and set window to N).
  w                 *  Backward one window (and set window to N).
  ESC-SPACE         *  Forward  one window, but don't stop at end-of-file.
  d  ^D             *  Forward  one half-window (and set half-window to N).
  u  ^U             *  Backward one half-window (and set half-window to N).
  ESC-(  RightArrow *  Left  one half screen width (or N positions).
  ESC-)  LeftArrow  *  Right one half screen width (or N positions).
  F                    Forward forever; like "tail -f".
  r  ^R  ^L            Repaint screen.
  R                    Repaint screen, discarding buffered input.
        ---------------------------------------------------
        Default "window" is the screen height.
        Default "half-window" is half of the screen height.
 ---------------------------------------------------------------------------

                          SEARCHING

  /pattern          *  Search forward for (N-th) matching line.
  ?pattern          *  Search backward for (N-th) matching line.
  n                 *  Repeat previous search (for N-th occurrence).
  N                 *  Repeat previous search in reverse direction.
  ESC-n             *  Repeat previous search, spanning files.
  ESC-N             *  Repeat previous search, reverse dir. & spanning files.
  ESC-u                Undo (toggle) search highlighting.

        ---------------------------------------------------
        Search patterns may be modified by one or more of:
        ^N or !  Search for NON-matching lines.
        ^E or *  Search multiple files (pass thru END OF FILE).
        ^F or @  Start search at FIRST file (for /) or last file (for ?).
        ^K       Highlight matches, but don't move (KEEP position).
        ^R       Don't use REGULAR EXPRESSIONS.
 ---------------------------------------------------------------------------
```

Cambiare e creare directory: cd, pwd e mkdir

Non vi lasciate spaventare da questa mole di informazioni. Non è necessario che le ricordiate tutte, altrimenti quale sarebbe la funzione dell'help :-).

Ricordatevi però che una delle funzioni più interessanti di more e di less è quella di filtri. Ad esempio, diciamo che state elencando una directory di alcune centinaia di file (ad esempio /usr/bin/) con ls e non sapete come vedere i primi file che scorrono via velocemente. È molto semplice:

```
$ ls -l /usr/bin/ | more
total 119712
-rwxr-xr-x    2 root     root        20092 Jul 26  2001 [
-rwxr-xr-x    1 root     root         5144 Aug 18  2001 3ds2m
-rwxr-xr-x    1 root     root         7612 Aug 18  2001 3ds2rib
-rwxr-xr-x    1 root     root         5732 Aug 18  2001 3dsdump
-rwxr-xr-x    1 root     root         5380 Apr  1 20:00 411toppm
-rwxr-xr-x    1 root     root         1649 Mar 17 10:54 822-date
-rwxr-xr-x    1 root     root        29080 Jan 28 00:04 9wm
-rwxr-xr-x    1 root     root        94264 Jan 10 18:20 a2p
-rwxr-xr-x    1 root     root       276600 Apr 14 20:28 a2ps
lrwxrwxrwx    1 root     root            4 Apr  4 10:06 aaaa -> host
-rwxr-xr-x    1 root     root        15512 Feb  9 23:12 aaxine
lrwxrwxrwx    1 root     root            7 Apr 11 09:59 abiword -> AbiWord
-rwxr-xr-x    1 root     root         1971 Apr  4 04:34 AbiWord
lrwxrwxrwx    1 root     root           27 Mar 21 09:53 AbiWord_d -> /etc/altern
atives/AbiWord_d
-rwxr-xr-x    1 root     root      4143780 Apr  4 04:35 AbiWord_d.gtk
-rwxr-xr-x    1 root     root         3645 Apr  4 04:34 abw2html.pl
-rwxr-xr-x    1 root     root         3844 Apr 27 07:05 access
-rwxr-xr-x    1 root     root        10742 Oct 19  2001 aclocal
-rwxr-xr-x    1 root     root          236 Feb  4 21:44 acroread
-rwxr-xr-x    1 root     root        19268 Mar 17 05:16 addftinfo
--More--
```

Notate che more non ci stà presentando la percentuale di linee lette. Questo non è possibile in quanto riceve i dati da una pipe e quindi non può sapere in anticipo quante linee leggerà in totale.[15]

[15] Ma noi lo sappiamo grazie ad ls. Sarebbero state 119712. Meglio aver usato more

3.6 Cambiare e creare directory: `cd`, `pwd` e `mkdir`

Il comando `cd` è un acronimo di *change directory* e serve a spostare la propria posizione da una directory ad un'altra. La sintassi è elementare:

```
cd nuovadirectory
```

Le directory possono essere indicate in percorsi assoluti (ossia con uno slash davanti) oppure in percorsi relativi. Per sapere in quale directory ci troviamo possiamo usare il comando `pwd` ossia *print working directory* :

```
$ pwd
/home/tx0/TeX/conTeXt/UB
$ cd ..
$ pwd
/home/tx0/TeX/conTeXt
$ cd /tmp
$ pwd
/tmp
$ cd
$ pwd
/home/tx0
$ cd ~/Mail
$ pwd
/home/tx0/Mail
$ cd -
$ pwd
/home/tx0
$
```

Nel primo caso ci siamo spostati con un percorso relativo (`..`) nella directory padre, ossia in quella di livello precedente. Con il secondo `cd` invece ci siamo spostati su un percorso assoluto (`/tmp` comincia con uno slash e quindi è relativo all'intero filesystem anziché alla directory corrente). Con il terzo comando (`$ cd`) ci siamo spostati nuovamente nella nostra home directory. Infatti il comando `cd` senza argomenti riporta nella propria home.

Cambiare e creare directory: cd, pwd e mkdir

Seguono due notazioni speciali. Quando si indica una tilde (~) in un path, questa simboleggia la propria home directory. In questo caso quindi, essendo la home di tx0 la directory /home/tx0/, la scrittura ~/Mail è una forma contratta di /home/tx0/Mail/.

La seconda scrittura (-) indica la penultima directory visitata. Infatti nell'esempio cd - ci riporta in /home/tx0, ossia la penultima directory nella quale ci siamo spostati.

Come fare però a creare una directory? Il comando che assolve questa funzione è mkdir. La sintassi è:

```
mkdir [OPZIONI] directory
```

Ad esempio per creare la directory /tmp/prova possiamo eseguire:

```
$ cd /tmp
$ mkdir prova
$
```

oppure:

```
$ mkdir /tmp/prova
$
```

La directory /tmp deve però esistere, altrimenti mkdir ci restituirà un errore. Una opzione molto utile in questo caso è -p. Se le directory di livello superiore (*parent directory*, da cui -p) non esistono, mkdir provvede a crearle. Ad esempio per creare la directory /tmp/directory/con/un/path/molto/lungo possiamo usare:

```
$ mkdir -p /tmp/directory/con/un/path/molto/lungo
$
```

molto più comodo di:

```
$ mkdir /tmp/directory
$ mkdir /tmp/directory/con
$ mkdir /tmp/directory/con/un
$ mkdir /tmp/directory/con/un/path
$ mkdir /tmp/directory/con/un/path/molto
$ mkdir /tmp/directory/con/un/path/molto/lungo
$
```

3.7 Copiare e muovere i file: cp e mv (*)

cp e mv sono entrambi comandi utilizzati per la manipolazione di file e directory, e sono molto semplici:

```
cp [OPZIONI] [file1 [file2 [fileN]]] [destinazione]

mv [OPZIONI] [file1 [file2 [fileN]]] [destinazione]
```

L'origine può essere rappresentata da più file tramite l'utilizzo di wildcard o può essere un elenco dei file da copiare o spostare; in questi casi, però, è necessario che la destinazione sia rappresentata da una directory.

Sotto UNIX lo stesso comando che si usa per cambiare posizione ad un file si usa anche per cambiarne il nome. Basterà usare mv per "spostarlo" in un file con un nome diverso.

cp, inoltre, supporta un'opzione -R che risulta utilissima per copiare interi pezzi di filesystem in quanto esplora ricorsivamente le directory. Così un comando come

```
$ cp -R /home /backup
$
```

sarà più che sufficiente per portare in salvo su di una directory di backup tutti i file di tutti gli utenti della macchina. Vi sono ovviamente altre opzioni, ma sono riservate per casi particolari. I più curiosi possono naturalmente consultare le manpage dei due comandi.

3.8 Cancellare file e directory: rm e rmdir (*)

Per cancellare file e directory sotto Unix si utilizza invece il comando rm (remove). Questo comando non va assolutamente sottovalutato! Gli utenti abituati ai più apprensivi DOS e Windows potrebbero rimanere un po' spiazzati dalla leggerezza con la quale, sotto Unix, sia possibile spazzare via un intero sistema.

Questo Sistema Operativo, infatti, si basa sul presupposto che l'utente sappia ciò che stà facendo. Dimenticatevi quindi le rassicuranti conferme, scordatevi di ricevere avvisi nel caso stiate tentando di cancellare file importanti, lasciate alle spalle il cestino e la comoda possibilità di recuperare anche quei file che sembravano perduti. Sotto Unix, infatti, nulla di tutto ciò è valido. Un file cancellato lo è *per sempre* e non c'è nulla (o quasi)[16] che potrà riportarlo alla vita. Tutto ciò premesso, ecco a voi la sintassi di questo semplice quanto distruttivo comando:

```
rm [file]
```

Come per cp vale l'utilizzo delle wildcard. Infatti

```
$ rm *
$
```

distruggerà inesorabilmente ogni file nella directory corrente. I più timorosi (magari in transizione da un altro Sistema Operativo), potrebbero cadere nella tentazione di utilizzare l'opzione "-i", che fa in modo che il sistema chieda conferma prima di cancellare ogni singolo file. Tuttavia esistono molte buone ragioni per non utilizzare una simile opzione. Una fra tutte, quando cancellate molti file vi capiterà sicuramente di annoiarvi di premere il tasto "y" ed inizierete a mitragliarci sopra senza neanche leggere quello che state cancellando. Sarà proprio allora che vi accorgerete di un file importante, tre righe più in alto, che rm ha già portato con sé nell'oblio.

Molto meglio prendersi le proprie responsabilità, non affidarsi ai falsi sensi di sicurezza e pensarci due o tre volte prima di premere il tasto "Invio". Se, al contrario, siete amanti del pericolo, non potrete non apprezzare l'opzione "-R" (a volte anche "-r"), analoga a quella spiegata per cp. Quindi il seguente comando:

16 In effetti il filesystem ext2 di Linux supporta la possibilità di recuperare file cancellati

```
rm -R /
```

cancellerà l'intero filesystem in brevissimo tempo! L'opzione suddetta risulta molto utile per cancellare intere directory ed il loro contenuto. Tuttavia il comando `rmdir` risulta molto più sicuro nell'amministrazione quotidiana, in quanto cancella una directory solo se vuota. Prendendo l'abitudine di impiegarlo al posto del più comodo `rm -R` si possono evitare errori che potrebbero costarvi parecchi file preziosi. Eccone il formato:

```
rmdir [directory]
```

Chi ha seguito la descrizione di `mkdir` si ricorderà di un'opzione abbastanza utile chiamata '-p', che ha la funzione di seguire il percorso specificato. Questa opzione esiste anche su `rmdir`, quindi per cancellare una directory 'documenti' contenente SOLO la directory 'personali', a sua volta vuota, potremo usare:

```
$ rmdir -p documenti/personale
$
```

3.9 Cambiare i permessi e la proprietà: `chmod`, `chown`

Vediamo ora come si fa a cambiare i permessi e la proprietà dei file. Il comando `chmod` è quello che ci permette di modificare lo stato di un file, renderlo leggibile, scrivibile, eseguibile, non leggibile e così via; ci sono due modi per farlo, ma qui vedremo solo quello più semplice: i parametri devono essere la somma dei valori di ogni singolo modo che si vuole assegnare per ogni entità (proprietario, grupppo, tutti gli altri):

r→4 w→2 x→1

dove *r* stà per lettura (read), *w* stà per scrittura (write) e *x* stà per esecuzione (execution).

Volendo settare i permessi di un file a `-rwxr-xr--` (leggibile, scrivibile ed eseguibile per il proprietario, leggibile ed eseguibile, per il gruppo e solo leggibile per gli altri) dovremmo immettere il seguente comando:

Cambiare i permessi e la proprietà: chmod, chown


```
$ chmod 754 nomefile
$
```

Da dove arrivano questi numeri?

Ogni cifra si riferisce ad una categoria di utenti del sistema. La prima riguarda il proprietario del file. La seconda il gruppo principale del quale fà parte, La terza riguarda tutti gli utenti che non sono parte di quel gruppo.

Nel nostro caso abbiamo: Utente → 7, ossia 4+2+1, Gruppo dell'utente → 5, ossia 4+1. Altri utenti → 4, ossia 4 da solo. 4+2+1 attribuito all'utente proprietario significa *lettura* (4) + *scrittura* (2) + *esecuzione* (1) ossia 7 in totale. Così il 5 del gruppo significa *lettura* (4) + *esecuzione* (1). Il resto degli utenti del sistema hanno permessi di sola lettura quindi 4.

`chmod` ha anche una sintassi mnemonica basata su lettere anziché su numeri ottali. Per quanto questa sintassi possa essere più facile per alcuni in quanto ciascuna lettera viene presa dall'iniziale della parola alla quale si riferisce (*u* per l'Utente, *a* per All users, *r* per Read permission), riteniamo essere più immediata la scrittura numerica una volta che si è familiarizzato con essa.

Talvolta può capitare di trovare dei permessi strani, come:

```
$ ls -l ~/perms
-rwsr-s--T  1    tx0     tx0     0 May 27 14:28 /home/tx0/perms
$
```

Cosa significano questi permessi?

Al posto della x di esecuzione, sia nel campo utente `rws` che nel campo gruppo `r-s` troviamo la lettera *s* . Cosa significa?

Questa configurazione dei permessi prende il nome di *set uid bit* e *set gid bit* rispettivamente, a seconda che faccia riferimento all'utente o al gruppo. La funzione di questa impostazione significa: *quando il file viene acceduto, a prescindere da quale utente stia operando su esso, il sistema operativo agirà come se l'utente fosse il proprietario (set uid bit) o l'utente fosse parte del gruppo proprietario (set gid bit)* .

Il bit uid vale 4, mentre il bit gid vale 2. Il meccanismo con il quale questi numeri si combinano è analogo a quello dei permessi di scrittura, lettura ed esecuzione. Se ad esempio vogliamo un file leggibile (4), scrivibile (2) ed eseguibile(1) dall'utente proprietario, leggibile (4) ed eseguibile (1) dagli utenti del gruppo del proprietario, leggibile (4) ed eseguibile (1) da tutti gli altri utenti del sistema e vogliamo che questo file venga acceduto ogni volta come se l'utente che vi accede fosse parte del gruppo del proprietario, potremmo usare la sintassi:

```
$ chmod 2755 nomefile
$
```

Il primo "2" che compare è proprio il set gid bit di cui parlavamo prima. Se avessimo voluto che il file fosse acceduto come l'utente proprietario (4) e come parte del gruppo dell'utente proprietario (2) avremmo usato:

```
$ chmod 6755 nomefile
$
```

Un'ultima nota: `ls` mostra questa proprietà particolare dei file sovrascrivendo la `x` nei permessi. Come è possibile dunque sapere se un file, oltre ad avere il set uid bit attivo è anche eseguibile? `ls` adotta questa convenzione: se un file ha anche il corrispettivo permesso di esecuzione attivo, la lettera `s` sarà minuscola. Se il permesso di esecuzione non è stato concesso, la lettera sarà maiuscola.

Ad esempio il file `/perms` mostrato prima ha una lettera `T` maiuscola nel campo esecuzione degli altri utenti di sistema. Questo significa che il permesso di esecuzione a quegli utenti che NON fanno parte del gruppo del proprietario è stato negato. Già, ma cosa significa la lettera `t`?

È una storia che risale a molto tempo fà. Quando i computer erano meno potenti delle calcolatrici che trovate nel fustino del detersivo, i sistemi UNIX davano la possibilità di appoggiarsi allo *swap space* ossia alla memoria virtuale per eseguire i programmi. Oggi non è più necessario, così questa informazione ha perso di significato. Alcuni sistemi operativi hanno tuttavia ripreso questa opzione e le hanno attribuito nuovi significati. Consultate la man page di `chmod` del vostro sistema UNIX per avere informazioni in proposito. Su Linux questo bit è semplicemente ignorato.

E per quanto riguarda la proprietà? Qui il discorso è molto più semplice. Prima di tutto dovete sapere che un utente viene descritto in due o tre file. Quello più importante è il file `/etc/passwd` che contiene la userid, l'UID, il GID, la password, il path alla home directory e la shell di ciascun utente. Subito dopo segue il file `/etc/group` che descrive i gruppi e chi ne faccia parte. Se un utente è registrato in più di un gruppo, è qui che questa registrazione è avvenuta. L'ultimo file è `/etc/shadow`. Non esiste in tutti i sistemi. Serve a rendere più sicure le password.

Quando la password viene salvata nel file `/etc/passwd` viene crittata. Ma il file è leggibile a tutto il mondo per le *altre* informazioni che contiene. Così la *hash*[17] della password può essere prelevata. Su questa hash si possono far girare programmi di cracking delle password come "john the ripper" e tentare di indovinare la password di un utente.

[17] Si chiama hash il risultato di una funzione di crittazione.

Cambiare i permessi e la proprietà: chmod, chown

Per ovviare a questo problema è stato inventato il file /etc/shadow. Questo file è leggibile *solo* a root. Nel /etc/passwd viene posta una 'x' al posto della password. Questo significa: cerca la password nel file /etc/shadow. Siccome i programmi che autenticano[18] gli utenti all'atto della connessione al sistema girano tutti come root, non avranno problemi a leggere /etc/shadow mentre gli utenti comuni non potranno più recuperare la hash delle password.

Detto questo, come gestisco la proprietà dei miei file? Il comando chown (change owner, cambia proprietario) ci viene in aiuto. La sintassi è elementare:

```
chown [opzioni] proprietario[:gruppo] file1 [ file2 [ fileN ] ]
```

L'elenco dei file può essere lungo quanto volete, basta che lo regga la shell. proprietario è l'unica informazione obbligatoria. Si può indicare tanto in UID che in userid. Così se l'UID del mio account è 1045 posso usare indifferentemente:

```
$ chmod tx0 myfile
$ chmod 1045 myfile
$
```

Per cambiare tutti i file in una directory usiamo l'opzione -R:

```
$ chmod -R tx0 mydirectory
$
```

Tutti i file contenuti nella directory mydirectory, incluse le sottodirectory ed i loro file, saranno di proprietà di tx0. Ovviamente[19] si può cambiare proprietà ai soli file di cui già si è poprietari. Di conseguenza se un utente normale (come tx0) assegna la proprietà dei suoi file a shodan non gli sarà più possibile tornare indietro. Dovrà essere shodan a ridargliela. Oppure root.

Aggiungendo un punto o un due punti dopo il nome del proprietario si può specificare in un colpo solo anche il gruppo. Ad esempio:

[18] Nel nostro caso autenticare significa verificare che una persona che si presenta come una data utenza ne possegga la password corretta

[19] Sii dai l'avevate già capito da soli, no?

```
$ chown tx0.staff mydocument
$
```

assegna all'utente `tx0` ed al gruppo `staff` il file `mydocument`. Per cambiare solo il gruppo senza toccare l'utente si può usare il comando `chgrp`, gemello di `chown`. La sintassi è la solita:

```
chgrp [opzioni] gruppo file1 [ file2 [ fileN ] ]
```

3.10 Scorciatoie: i link e il comando `ln` (*)

Benché al neofita possa sembrare una bizzarria, sotto Unix è possibile creare dei file speciali che in realtà hanno il contenuto di qualche altro file: sono i cosiddetti file "link". Questi si suddividono in "*hard link*" e "*soft link*".

I primi sono più radicali, e sono una copia indistinguibile dal file originale: una volta creati non è più possibile dire quale sia l'originale, e benché siano visti come file distinti hanno in realtà il medesimo contenuto. Modificando uno si modifica anche l'altro, occupano lo spazio necessario per una sola copia e il file non viene cancellato veramente finché tutti i link che lo puntano sono stati eliminati. Gli altri, i "soft link" sono solo dei file contenenti il riferimento al file vero e proprio. Il soft link è un file ben identificabile, ed una volta eliminato il file originale esso perderà di significato: ogni riferimento ad esso genererà un errore di "file non trovato". Inoltre è possibile creare soft link di directory, cosa non possibile con gli hard link. Ma passiamo alla pratica: per creare i link si usa la seguente sintassi:

```
ln [-s] [file/directory] [link]
```

L'opzione `-s` viene utilizzata per specificare che si desidera creare un soft link. Per distruggere un link basta un buon vecchio `rm`. I link vengono utilizzati per condividere informazione fra utenti o processi, ma anche per altri motivi inerenti l'amministrazione di sistema.

3.11 Qualcuno mi aiuti! (man, info e l'opzione -h)

A volte capire quello che UNIX vuole da noi non è la cosa più immediata di questo mondo. Ma cè un modo chiedere aiuto. Anzi più di uno.

Il primo, quello storicamente più diffuso è il comando `man` che è l'abbreviazione di *man nual page*.

```
man [-k] [SEZIONE] <COMANDO>
```

Se ad esempio vogliamo avere la pagina di manuale del comando `ls` dovremo usare il comando:

```
$ man ls
```

La man page ci viene mostrata attraverso `more` o `less` o quello che abbiamo impostato nella variabile `$PAGER`.

Prima di continuare, bisogna che vi dica che il manuale di UNIX (ossia l'insieme delle man page) è suddiviso in sezioni:

Sezione	Comandi contenuti
1	Programmi o comandi di shell
2	System calls (funzioni del kernel)
3	Library calls (funzioni delle librerie)
4	File speciali (ad esempio `/dev/*`)
5	Formati dei file (come `/etc/passwd`)
6	Giochi `:-)`
7	Macro package (come `groff`)
8	Comandi riservati all'amministratore di sistema
9	Kernel routine (ma non è standard)

Ci sono situazioni in cui avere aiuto da `man` comporta dare un aiuto a `man :-)` Ad esempio nel caso del comando `sleep`[20] `man sleep` ci dà la man page giusta. Ma se vogliamo ottenere la man page della funzione `sleep`?

[20] Il comando `sleep` serve a rendere dormiente il proprio processo per qualche secondo. Il numero di secondi lo specificate subito dopo il comando

Dalla tabella risulta evidente che la man page della funzione `sleep` si deve trovare nella sezione 3:

```
$ man 3 sleep
```

Et voilá!

E se non sappiamo cosa cercare? Allora l'opzione `-k` diventa comoda. `man` cercherà in tutte le man page l'argomento che faremo seguire e darà una lista di tutte le pagine che pertinevano in qualche modo. Un modo forse più facile da ricordare per ottenere questa funzione è il comando `apropos`.

Spesso tuttavia è sufficiente chiedere direttamente al nostro comando di darci una mano. Normalmente l'opzione `-h` (o `--help`) chiede al comando una strisciata delle opzioni (o almeno delle più importanti).

Ultimamente FSF ha introdotto un nuovo sistema di documentazione dei comandi che preferisce alle manual page. Per ottenere informazioni su un comando digitate:

```
info <COMANDO>
```

come ad esempio `info tar`:

Qualcuno mi aiuti! (man, info e l'opzione -h)


```
File: tar.info,  Node: Top,  Next: Introduction,  Up: (dir)

GNU tar: an archiver tool
*************************

   GNU `tar' creates and extracts files from archives.

   This manual documents version 1.13.24 of GNU `tar'.

   The first part of this master menu lists the major nodes in this Info
document.  The rest of the menu lists all the lower level nodes.

* Menu:

* Introduction::
* Tutorial::
* tar invocation::
* operations::
* Backups::
* Choosing::
* Date input formats::
* Formats::
--zz-Info: (tar.info.gz)Top, 265 lines --Top------------------------------

Welcome to Info version 4.1. Type C-h for help, m for menu item.
```

All'interno di `info` potete saltare da un nodo ad un altro premendo semplicemenete INVIO . Il vantaggio di `info` è sicuramente l'ipertestualità ma tenete sempre presente che sui vecchi UNIX `man` è probabilmente la vostra unica risorsa.

4. La Shell

di Tx0

La shell (letteralmente conchiglia, ma correntemente tradotto interprete comandi) è il programma che viene lanciato dal login appena verificato che l'utente è chi dichiara di essere. Deve il suo nome al fatto che racchiude l'utente in una sorta di ambiente (da cui il concetto di conchiglia) fornendogli una omogeneo insieme di comandi per ottenere le funzioni più elementari dal sistema, permettendogli di configurare valori una volta per tutte in variabili visibili a tutti i programmi e consentendogli di utilizzare tutti i comandi aggiuntivi che l'installazione del sistema operativo mette a disposizione. Da qualsiasi shell odierna potete comunque sempre aspettarvi:

- *Gestione dei processi in background*
- *Completo linguaggio di programmazione*
- *Completazione della riga di comando*
- *Redirezione di input e output e piping dei comandi*

4.1 Differenti shell

La shell è un programma a tutti gli effetti. Così, come per qualsiasi tipologia di programma che si rispetti, anche di shell ne esistono diverse versioni. Le differenze sono soprattuto nel *linguaggio* con il quale si *programma* il comportamento della shell. Tutte le shell sono infatti dotate di costrutti logici che permettono scelte condizionali. Questo significa che è possibile pilotare il comportamento della shell al verificarsi o meno di una certa condizione, oppure ripetutamente per *n* volte. Questo tipo di *programmabilità* della shell consente addirittura di scrivere dei veri e propri *mini programmi* anche con compiti e capacità non banali.

A differenza però dei linguaggi di programmazione compilati come il **C**, la shell si programma anche *runtime* ossia mano mano che si usa. Da un punto di vista formale immettere un singolo comando significa (dal punto di vista della shell) avere eseguito un programma di una linea di codice.

Non è solo a questo che si limitano le differenze fra le varie shell. Il prompt, la gestione dei processi, le variabili d'ambiente, numerosi comandi differenziano fra loro le varie shell.

La prima shell era `sh`.[1] Questa shell non aveva controllo dei processi e non completava la linea di comando, tanto per dirne un paio. Era la più elementare delle shell possibili. In fondo era anche la prima!

Anche se non esistevano altre shell per confondercisi, questa aveva un nome particolare: *Bourne shell*, dal nome del programmatore che l'aveva scritta. Più tardi GNU di FSF produrrà una sua versione potenziata e migliorata di *sh* chiamata *bash* (*Bourne Again SHell*, come sempre un acronimo che è più una scusa per se stesso che non per un reale significato, ma questo è il nostro mondo e i nomi li diamo noi `:-)`. `bash` è migliore di `sh` perché offre un ambiente migliorato ed un prompt molto più estensibile, recupera nuove caratteristiche da altre shell e ne introduce di nuove.

`sh` richiama con la sua sintassi di programmazione il Pascal. Decisamente lontano dall'amato C degli hacker di UNIX. Per questo venne creata `csh`, ossia *la shell C*. Intendiamoci: `csh` non è diversa da `sh` solo nel suo linguaggio di programmazione, ma anche in molte altre caratteristiche.

In tempi recenti (ossia una decina di anni fa), è comparsa `tcsh`, versione rinnovata ed espansa di `csh`. Riportata così può sembrare una rincorsa alla shell perfetta. In realtà `sh` e `csh` sono così profondamente diverse che il loro sviluppo raramente si incrocia, salvo quando qualcuno esterno fa sì che questo succeda.

È il caso di `zsh` e di `pdksh`, shell ibride che fondono elementi da `bash` e `tcsh` e ne aggiungono di propri. Il panorama è insomma vasto: per darne una panoramica generale ma non faziosa, descriviamo la sintassi e il funzionamento delle shell *sh* e *csh* in generale.

Elementi di una shell 4.1.1

Qualsiasi shell ha comunque alcuni elementi in comune. Ogni shell presenta all'utente una sequenza di caratteri chiamata *prompt* all'inizio di ogni nuova linea. Questa sequenza (a seconda di come è stata configurata) può fornire all'utente informazioni circa il suo *userid*, il nome della workstation sulla quale si trova (molto utile in caso di connessioni remote come vedremo più avanti) e la *directory corrente*.

Un'altro elemento comune è il linguaggio di programmazione della shell. Ogni shell è prima di tutto un interprete di comandi. Quando richiediamo interattivamente l'esecuzione di un comando come `ls -la` è quasi come se stessimo scrivendo un programma di una sola linea di codice. La shell tuttavia provvede meccanismi più evoluti di programmazione come i cicli condizionali e i costrutti logici.

[1] e come altro poteva chiamarsi? `shell`? No, ben cinque caratteri, troppo lungo per una linea seriale a 75 bps

Altre caratteristiche ormai presenti in molte shell (se non in tutte) sono caratteristiche di assistenza alla scrittura dei comandi, come la *completazione automatica dei comandi* o la possibilità di *editing della linea di comando*.

4.2 sh e derivate

Esaminiamo per prime la Bourne Shell e le sue derivate...

Il Prompt 4.2.1

Le *Bourne Shell* utilizzano la variabile PS1. Esiste poi la variabile PS2 che definisce una altro prompt usato quando la shell si aspetta ulteriore input dentro un contesto che non è quello normale (maggiori chiarimenti in seguito).

Il contenuto del prompt è costituito di lettere e numeri e di caratteri speciali con significati speciali. Li riassumiamo nella tabella seguente:

Carattere	Significato
`\a`	il carattere ASCII del segnale acustico
`\d`	la data in formato "Giorno/sett. Mese Giorno/mese" (es. "Tue May 26")
`\e`	un carattere escape ASCII (033)
`\h`	l'hostname fino al primo punto
`\H`	l'hostname completo
`\n`	"a capo"
`\r`	"ritorno di carrello"
`\s`	il nome della shell, il nome senza directory di $0
`\t`	l'ora corrente in formato europeo 24 ore
`\T`	l'ora corrente in formato anglosassone 12 ore
`\@`	l'ora corrente in formato anglosassone 12 ore am/pm
`\u`	lo username
`\v`	la versione della shell (proprietario di bash
`\V`	la versione della bash comprensivo di patchlevel
`\w`	la directory corrente
`\W`	il "basename" della directory corrente
`\!`	il numero progressivo dell'ultimo comando dato
`\#`	il numero del comando attuale
`\$`	un # se l'UID vale zero, altrimenti un $
`\nnn`	il carattere corrispondente al numero ottale nnn

`\\`	un backslash
`\[`	inizia una sequenza di caratteri non visualizzabili

Impostare il prompt è possibile tanto da riga di comando che da file di configurazione. Eseguiamo una semplice assegnazione interattiva del prompt:

```
$ export PS1="\u@\h:\w\$ "
tx0@defiant:/tmp$
```

Un prompt come quello appena impostato è molto utile in parecchie circostanze. Ad esempio ci ricorda con quale account siamo collegati sulla macchina (ammettendo di avere più account). Ci ricorda se siamo o meno root anche tramite il tipo di terminatore `$` oppure `#`. Ci ricorda sua quale host siamo collegati (il che in caso si sia root e si stia per lanciare uno shutdown è utile per evitare di spegnere un altro computer per sbaglio, parlo per esperienza personale).

> Notate che il carattere speciale `\w` cambia ad ogni cambio di directory.

Il prompt impostato in PS2 viene invece riportato in caso ci si trovi in un contesto aperto e la shell stia attendendo altro input oppure la chiusura del contesto. Per default il prompt è impostato a "> ". Vediamo un esempio:

```
$ echo "
> ciao
> "

ciao

$
```

Variabili notevoli 4.2.2

Le variabili sotto le Bourne Shell si impostano con la diretta assegnazione di un valore alla variabile (es. `PATH="/usr/local/bin:$PATH"`). Perché una variabile sia utilizzabile dai programmi che girano in quell'ambiente, questa deve essere "*esportata*". L'esportazione avviene attraverso il comando `export`. Con la shell di GNU (ricordiamo che si chiama bash) è possibile unire le due operazioni in un'unica linea come fatto precedentemente con PS1 e PS2. Le vecchie bourne invece non consentivano questo, per cui nello scrivere uno script che deve girare anche su una vecchia shell ricordate di spezzare i comandi:

```
$ PATH="/usr/local/bin:$PATH"
$ export PATH
$
```

Come avete notato si fa riferimento ad una variabile con semplicemente con il suo nome quando le deve essere assegnato un valore. Viene invece riferita con un dollaro ($) davanti quando la si vuole *"dereferenziare"*, ossia se ne vuole estrarre il valore contenuto. Nell'esempio precedente abbiamo scritto `PATH="/usr/local/bin:$PATH"` che si legge "*assegna alla variabile PATH il valore costituito da /usr/local/bin e il precedente contenuto della variabile stessa, indicato con $PATH*".

Oltre ai due prompt già incontrati (e badate che esistono anche PS3 e PS4 ma il loro significato ve lo leggete nella man page della shell :-)vediamo un po' di altre variabili notevoli.

Variabile	Significato
PPID	il PID del processo che ha chiamato la shell
PWD	la directory corrente
UID	l'UID dell'utente
BASH	il filename completo usato per chiamare la shell (solo bash ovviamente)
SHLVL	contiene il numero di shell (inclusa quella attuale) che sono state chiamate a catena
RANDOM	ad ogni riferimento, questa variabile cambia valore a caso fra 0 e 32767
HOSTNAME	il nome dell'host
HOSTTYPE	contiene una descrizione dell'architettura dell'host come i386
OSTYPE	contiene una descrizione del tipo di sistema operativo come `linux-gnu`
MACHTYPE	contiene una descrizione generale del sistema come `i386-pc-linux-gnu`
SHELLOPTS	contiene una lista delle opzioni selezionate della shell
PATH	contiene un elenco di directory chiamato *"PATH"all'interno delle* quali cercare gli eseguibili
HOME	la home directory dell'utente
MAIL	il percorso completo alla mailbox
MAILPATH	un elenco separato da : di mailbox
MAILCHECK	specifica l'intervallo in secondi fra un check della mail e il successivo
HISTSIZE	la dimensione dell'history in comandi
HISTFILE	il file nel quale viene salvata la history fra una sessione e l'altra
HISTFILESIZE	il massimo numero di linee contenute nel file di history

LANG	la categoria linguistica da usare (vale in caso manchi la definizione di LC_* appropriata (vedi man page)

Questo elenco non è esaustivo. Per un elenco completo delle variabili importanti consultate la man page della shell. Troverete altre variabili ancora, molte delle quali complesse e esoteriche.

4.3 Controllo dei processi

La gestione dei processi nella shell presenta alcuni vantaggi rispetto a quella basata solo sui tool di sistema. Per ogni processo lanciato in background la shell tiene traccia del PID e lo collega ad uno speciale simbolo che inizia con un percento (%) seguito da un numero progressivo. Vediamo un esempio:

```
$ updatedb &
[1] 1711
$ tar cf /tmp/inutile.tar *
[2] 1713
$ kill %1
[1]-  Done                    updatedb
$
```

Come avrete capito, lanciando un processo in background la shell fornisce prima il progressivo del processo e di seguito il PID. Il progressivo può essere usato per operare sul processo da dentro la shell. Il `kill` che abbiamo impiegato nell'esempio non è evidentemente quello di sistema. Si tratta di un comando builtin nella shell, differente rispetto a `/bin/kill`, in grado di accettare i progressivi dei processi.

È possibile interrompere un processo che sta girando in *"foreground"* con la pressione dei tasti `Control` e `c` insieme. Per sospendere l'esecuzione (senza terminarla) di un processo in foreground si usa invece la combinazione `Control-z`. Una volta sospeso un processo è possibile riportarlo in esecuzione in foreground con il comando `fg` o in background con il comando `bg`.

Per avere un elenco dei processi che stanno girando in background utilizzate il comando `jobs`.

```
$ jobs
[1]+  Running                    xdvi corsounix &
$
```

Nel nostro esempio esiste un solo *"task"* in background (quindi numerato 1) attualmente in corso (`Running`, quindi non sospeso con un `Control-z`) che sta eseguendo il comando ``‘‘xdvi corsounix &’’``[2].

Input e Output, Redirezione, Cilindri, Conigli Bianchi... 4.3.1

È possibile *"redirigere"* l'output della shell e l'input verso la shell per e da differenti fonti. È anche possibile concatenare più comandi fra di loro con un meccanismo detto *"piping"* in modo che l'output di un comando diventi l'input del comando successivo. Questo meccanismo è molto utile nel caso si vogliano utilizzare dei filtri.

La redirezione dell'input e dell'output avviene rispettivamente con i simboli *"minore"* (<) e *"maggiore"* (>). Se vogliamo ad esempio salvare l'output di `ls -l` all'interno di un file possiamo usare la forma:

[2] La presenza della "e commerciale" in fine riga è una conferma del fatto che il processo gira in background

```
$ ls -l > files
$ cat files
total 9032
-rw-r--r--    1    tx0      tx0      19685 Mar  9  2001 corso_perl_loa.tgz
-rw-r--r--    1    tx0      tx0       4424 Feb 14  2001 corsoperl
-rw-r--r--    1    tx0      tx0          0 Oct 21 11:49 files
drwx--S---    2    tx0      tx0       4096 Feb  5  2001 mail
drwx--S---    2    tx0      tx0       4096 May  9  2000 nsmail
-rw-r--r--    1    tx0      tx0       2379 Feb 25  2001 perl.mappings.vim
-rw-r--r--    1    tx0      tx0     265014 Jan 21  2001 underground.txt.bz2
-rw-r--r--    1    tx0      tx0       8650 Dec  3  2000 wangtek5525es.htm
-rw-r--r--    1    tx0      tx0      11824 Feb  7  2001 yprefcard.ps
$
```

Notate che anche il file *"files"* è contenuto nel listato. Questo perché prima la shell crea il file e poi esegue il comando (`ls -l`) il cui output andrà rediretto nel file

Il meccanismo di piping dell'output si realizza utilizzando il carattere `|`, detto appunto *"pipe"*. La sintassi è la seguente:

```
$ ls -la | wc -l
      19
$
```

Il comando non è forse dei più riusciti[3], ci dà sicuramente un'idea del funzionamento del meccanismo.

[3] L'idea sarebbe quella di contare il numero dei file presenti nella directory corrente tramite il conteggio delle linee (`wc -l`) date come output da `ls -la`; il punto è che l'output contiene anche la linea con la dimensione totale della directory e le due linee iniziali inerenti la directory corrente e quella precedente

4.4 Sintassi di programmazione

Le Bourne Shell hanno alcune parole chiave riservate per la programmazione della shell.

La keyword `if` server a eseguire uno o più comandi se una condizione è verificata. Se la condizione è verificata, la keyword `then` delimita l'inizio dei comandi da eseguire. L'elenco dei comandi dura sino alla keyword `fi` (ossia `if` al contrario...). La condizione è compresa fra parentesi quadre. Se si vuole fornire un elenco di comandi da eseguire in caso in cui la condizione non fosse verificate, si può usare la keyword `else`.

```
$ if [ $UID == 0 ]
> then
>   echo "Oh magnifico root"
> else
>   echo "Ma tu non sei root..."
> fi
Ma tu non sei root...
$
```

La keyword `elif` serve ad introdurre una nuova condizione da verificare. La sua sintassi è identica a quella di `if`.

```
$ if [ $UID == 0 ]
> then
>      echo "Oh, magnifico root"
> elif [ $UID -lt 1010 ]
> then
>      echo "Oh, rispettabile membro del gruppo staff"
> else
>      echo "Ma tu non sei nessuno..."
> fi
Oh, rispettabile membro del gruppo staff
$
```

La keyword `while` serve ad eseguire ciclicamente un set di comandi finché la condizione specificata rimane valida. I comandi sono compresi fra la keyword `do` e la keyword `done`. La keyword `until` usa la stessa sintassi di `while` ma la condizione da valutare è negata.

```
$ c=0;
$ while [ $c -lt 10 ]
> do
>        echo -n "$c "
>        let c=$c+1
> done
0 1 2 3 4 5 6 7 8 9 $
```

La keyword `for` consente di iterare una variabile su un set di valori, passato come elenco di elementi a seguito della keyword `in`:

```
$ for counter in 1 2 3 4 7 8 31337
> do
>   echo $counter
> done
1
2
3
4
7
8
31337
$
```

L'ultima keyword che esaminiamo è `case` che permette di eseguire set di istruzioni in base alla corrispondenza fra una variabile e uno o più pattern (o stringhe di testo).

```
$ c='test_variable'
$ case $c in
>   'failtest')
>      echo "non funziona"
>      ;;
>   'test_variable')
>      echo "yes, it's working"
>     ;;
> esac
yes, it's working
$
```

Notate che non viene testata la variabile `c` bensì il suo contenuto `$c`.

I file di configurazione 4.4.1

Il file storico da cui tutte le Bourne Shell leggono un elenco di comandi per predefinire l'ambiente di lavoro dell'utente è `~/.profile`. La `bash` usa un set più ampio di file per garantire maggiore flessibilità. Il file `~/.bashrc` viene letto sempre[4]. Il file `~/.bash_profile` invece viene letto solo in caso la shell sia chiamata interattivamente[5]. Esiste inoltre il file `~/.bash_logout` che viene letto ed interpretato quando l'utente chiude la shell.

[4] ...quindi anche quando la shell viene usata come interprete di comandi e non interattivamente dall'utente

[5] È facile notare nel file `~/.bash_profile` la riga `source ~/.bashrc` in modo da importare tutte le definizioni per la modalità non interattiva senza doverle ridefinire

4.5 Una possibile alternativa: `zsh`

Una shell particolarmente interessante sviluppata in tempi relativamente recenti è `zsh`. Alcune delle sue caratteristiche più impressionanti sono la sua ampliabilità attraverso una struttura modulare e la possibilià di modificare a piacere il comportamento dell'espansione della linea di comando, arrivando a completare persino le opzioni dei comandi!

Espansione della linea di comando 4.5.1

L'espansione della linea di comanda viene controllata con il comando `compctl`. La sintassi del comando è sostanzialmente questa:

```
compctl [OPZIONI] COMANDO [ .. COMANDO ]
```

`COMANDO` è un comando al quale applicare i criteri di espansione specificati con le `OPZIONI`. Vediamone alcune:

Opzione	Significato
`-k` *array*	Specifica un elenco di testi tra i quali scegliere per la completazione
`-K` *funzione*	Specifica di usare il risultato della funzione *funzione* per completare il comando
`-g` *globstring*	Specifica una stringa da espandere con i criteri di globbing della shell sui file presenti nella directory corrente
`-s` *string*	Specifica una stringa semplice di testo con la quale completare il comando

Ad esempio:

```
$ compctl -s "-9" gzip
$
```

imposta come espansione automatica la stinga `-9` al comando `gzip`. Ora, dopo aver digitato `gzip` premendo una volta il tasto TAB si otterrà automaticamente la combinazione `-9`.

Proviamo invece:

```
$ compctl -k "(xzf tzf zf tf)" tar
$
```

Dopo aver scritto `tar` una pressione del TAB ci darà l'elenco completo di tutte queste quattro possibili espansioni. Alla seconda pressione verrà utilizzata la prima, alla terza la seconda e così via.

Struttura modulare 4.5.2

Un'altra eccitante[6] caratteristica di `zsh` è la struttura modulare con la quale è possibile espanderne le funzionalità.

Se andate a sbirciare nella directory `/usr/lib/zsh/[versione della shell]/zsh/` noterete la presenza di molti file `.so` ossia *shared object* . Questi sono moduli che `zsh` può caricare per espandere le sue potenzialità. Vediamone alcuni:

Modulo	Funzione
clone	Consente di creare una copia della shell attaccandola ad un altro terminale
compctl	Come abbiamo già visto, espande le possibilità di espansione della linea di comando
example	Un modulo assolutamente inutile `:-)`che serve a dimostrare come scrivere un modulo per `zsh`
sched	Un concorrente di `at` e `cron`
zftp	Integra un client FTP sulla linea di comando
zle	Lo Zsh Line Editor
zprof	Un profiler per misurare le prestazione della shell, molto utile nel caso si scrivano script complessi

Per avere maggiori informazioni sul funzionamento dei moduli di `zsh` consultate la man page *zshmodules* . Vi sono spiegati tutti i moduli che fanno parte della distribuzione standard di `zsh` ed il loro utilizzo.

[6] Come dici? Mi perverto con poco?

5. Puntiamo più in alto

di Tx0

Trovare file con `which`, `find`, `locate` – Creare archivi con `tar` – Comprimere file con `gzip` e `bzip` – Dividere gli archivi con `split` – Ai piedi dei file e oltre: `tail`, `sort` – Ricerche su testo con `grep` – I processi e la loro gestione: `ps`, `top`, `kill` e `uptime` – I Device

5.1 Trovare file con `which`, `find` e `locate`

Il comando `which` consente di ricavare il percorso completo di un comando. Se volessimo sapere il percorso del comando `ls` sarebbe sufficiente dare il comando:

```
$ which ls
/bin/ls
$
```

Il comando deve trovarsi in una delle directory incluse nel path della shell (ossia nella variabile `$PATH`). Se invece è necessario cercare un file generico su tutto il filesystem è più opportuno utilizzare `locate`; questo cerca dentro un database creato in precedenza il file. In questo modo non viene eseguita una ricerca su disco (più lunga). Il sistema è configurato per eseguire (nei momenti prevedibilmente di minor carico) il comando `updatedb` che si occupa di creare il database. In caso sia necessario lavorare su un database più aggiornato (in quanto è stata installata una gran quantità di file e non si può attendere il prossimo update) si può lanciare il comando di update a mano (con i premessi di root ovviamente, altrimenti il database realizzato sarà personale ed incompleto, a causa dell'impossibilità di accedere a tutto il filesystem da parte di un utente).

Il comando `updatedb` si avvale di un altro comando per creare il database: il comando `find`. Il campo d'azione di questo comando riguarda la ricerca dei file. La sua peculiarità sta nel fatto che la ricerca non è limitata ai soli nomi di file, ma è consentita anche per *propriet a e permessi, tempi di creazione e ultimo accesso, tipo di file e altro ancora*.

La sintassi del comando è la seguente:

```
find [percorso] [schema di ricerca].
```

Il *percorso* indica le directory all'interno delle quali cercare, lo *schema di ricerca* indica i criteri con i quali operare la ricerca. Un'applicazione elementare di `find` è creare un listato di tutti i file contenuti sul filesystem, con il comando:

```
$ find / > /tmp/whole_disk
```

(Attenzione: è un task altamente dispendioso in termini di tempo.) Sul file `/tmp/whole_disk` sarà ora possibile eseguire ricerche evitando di sovraccaricare il disco. Tuttavia l'uso più flessibile di `find` si ottiene indubbiamente con l'uso di uno schema di ricerca. Riassumiamo in breve le opzioni:

Opzione	Significato
-amin *n*	Il file ha avuto l'ultimo accesso *n* minuti prima
-anewer *file*	Il file è più recente di *file*
-atime *n*	Il file ha avuto l'ultimo accesso *n* ore prima
-cmin *n*	Lo stato del file è stato modificato *n* minuti prima
-cnewer *file*	Lo stato del file è stato modificato più recentemente di quello di *file*
-ctime *n*	Lo stato del file è stato modificato *n* ore prima
-empty	Il file non contiene nulla
-gid *n*	Il gruppo del file è *n*
-group *name*	Come il precedente ma per nome di gruppo
-mmin *n*	I contenuti del file sono stati modificati *n* minuti prima
-mtime *n*	I contenuti del file sono stati modificati *n* ore prima
-name *pattern*	Il nome del file è *pattern* (vedi anche -path)
-newer *file*	Il file è stato modificato più recentemente di *file*
-nouser	Il proprietario del file non corrisponde ad alcun utente del sistema
-nogroup	Il gruppo del file non corrisponde ad alcun utente del sistema
-path *pattern*	Il nome del file corrisponde a *pattern*[1]

continua. . .

Opzione	Significato
-perm *(+/-)perms*	I permessi del file sono *perms*. Se i permessi sono preceduti da - tutti i bit dei permessi debbono essere settati come *perms*; se preceduti da + qualsiasi bit può essere impostato come *perms*.
-regex *pattern*	Il nome del file è rappresentato dalla regular expression *pattern*
-size *n[ckw]*	La dimensione del file è *n* blocchi da 512 byte. Se segue una delle lettere indicate *n* è indicato in byte (**c**), kylobyte (**k**) o parole (**w**).
-uid *n*	Il file appartiene all'utente *n*
-used *n*	Il file ha avuto un accesso *n* giorni dopo la modifica del suo contenuto
-user *name*	Il file appartiene all'utente *user*
-type *[bcdflps]*	Il tipo del file corrisponde a quello indicato dalla lettera come segue:

b	block special	l	symbolic link
c	character special	p	named pipe
d	directory	s	socket
f	regular file		

Vediamo qualche esempio utile dell'uso di `find`. Ammettiamo di voler trovare tutti i file dentro la nostra directory il cui nome finisca per ".tex".

```
$ find ~/ -name "*.tex"
/home/tx0/LaTeX/corsoUnix/corsoUnix.tex
/home/tx0/LaTeX/corsoUnix/storia_Unix.tex
/home/tx0/LaTeX/corsoUnix/regexpr.tex
/home/tx0/LaTeX/corsoUnix/la_shell.tex
/home/tx0/LaTeX/corsoUnix/piu_in_alto.tex
$
```

`find` lista il nome di ciascun file soddisfi i criteri di ricerca. Avrete notato che il pattern di ricerca è stato incluso in una coppia di virgolette. Questo accorgimento serve a evitare che la shell *interpoli* l'asterisco espandendolo in *tutti i nomi dei file della directory corrente*. L'asterisco fà infatti parte del *pattern* passato a `find` e non è un metacarattere per la shell.

Altro caso: vogliamo cercare tutti i file nella nostra directory che sono leggibili, scrivibili ed eseguibili per noi.[2]

```
$ find ~/ -perm 700
/home/tx0/bin/logger
/home/tx0/bin/script.pl
/home/tx0/bin/send_mail
/home/tx0/bin/parser.pl
$
```

Abbiamo trovato un buon numero di file. Vogliamo sapere ora quali sono anche eseguibili al nostro gruppo ed al resto degli utenti della macchina.

```
$ find ~/ -perm +777
/home/tx0/bin/script.pl
/home/tx0/bin/send_mail
$
```

Attenzione al più!! Il simbolo + indica a `find` che un file corrisponde ai criteri di ricerca se qualsiasi combinazione dei *bit di stato* coincide con quella espressa. Questo significa che nonsolo saranno positivi i file con permessi 755, ma anche quelli con permessi 700, quelli con permessi 750, 754, 755 e anche 007![3]

L'utilizzo del più ci permette di giocare con le combinazioni imponendo dei limiti a quali elementi possono essere usati per costruire la combinazione senza però obbligare la presenza di alcuno di essi. Ad esempio +755 consente di usare i permessi `-rwxr-xr-x` ma non ne richiede nessuno in particolare quindi include `-rwx------`, include pure `-rwxr-x--x` e `-r-xr-xr--`.

L'utilizzo di un meno al posto di un più inverte invece il significato: i permessi specificati sono tutti richiesti. Potrebbe sorgere il dubbio che il meno sia equivalente all'omissione di qualsiasi segno, ma così non è. Infatti `find ~/ -perm` 700 cerca solo i file `-rwx------`, mentre `find ~/ -perm` -700 cerca i file che abbiano permessi `rwx` per il proprietario, senza imporre limiti sugli altri permessi, quindi trova anche `-rwxrwx---` e `-rwxr-xr-x` ma non i file `-r-xr-xr-x` ad esempio in quanto questi non hanno permessi `rwx` per il proprietario.

Facciamo un ultimo esempio. Decidiamo di volere un listato di tutte le directory contenute nella nostra home directory.

---

2 Ricordiamo che la lettura vale 4, la scrittura vale 2 e l'esecuzione vale 1, quindi 7 per tutti e tre gli attributi

3 A dispetto della combinazione, questi file saranno ben poco segreti! :-)

```
$ find ~/ -type d
/home/tx0/
/home/tx0/mail
/home/tx0/LaTeX/corsoUnix
/home/tx0/gtk_perl
/home/tx0/gtk_tutorial
/home/tx0/Mail
/home/tx0/Perl
/home/tx0/sawfish_themes
$
```

Decidiamo di cercare fra queste quelle che sono anche leggibili ad altri utenti:

```
$ find ~/ -type d -perm -055
/home/tx0/
/home/tx0/mail
/home/tx0/Mail
$
```

Uhm, la nostra home è leggibile al resto del mondo, e così pure due directory che contengono posta elettronica. Sarà meglio cambiare i permessi se non vogliamo che occhi indiscreti vengano a curiosare nella nostra corrispondenza!

Forse qualcuno si chiederà perché impostare -055 invece che -044 come permessi di letture. Unix in effetti richiede che una directory sia leggibile ed eseguibile per poter fornire un contenuto di essa. O meglio: una directory 700, ad esempio, non consente nemmeno il listato dei file. Una directory 744 *"consente"* il listato dei file nel senso che permette di *tentare di leggere i dati generali per ciascun file contenuto nella directory* risultando in una serie di errori, come in:

```
$ ls -la ~/directory/
total 24
drwxr-sr-x    5 tx0   tx0       4096 Dec  4 17:55 .
drwxr-sr-x   53 tx0   tx0       8192 Dec  4 17:55 ..
drwx------    2 tx0   tx0       4096 Dec  4 17:55 dir1
drwxr--r--    2 tx0   tx0       4096 Dec  4 17:56 dir2
drwxr-xr-x    2 tx0   tx0       4096 Dec  4 17:56 dir3
$
$ ls -la ~/directory/dir1
ls: directory/dir1/: Permission denied
$
$ ls -la ~/directory/dir2
ls: directory/dir2/.: Permission denied
ls: directory/dir2/..: Permission denied
ls: directory/dir2/file3: Permission denied
ls: directory/dir2/file4: Permission denied
ls: directory/dir2/file5: Permission denied
total 0
$
$ ls -la ~/directory/dir3
total 8
drwxr-xr-x    2 tx0   tx0       4096 Dec  4 17:56 .
drwxr-sr-x    5 tx0   tx0       4096 Dec  4 17:55 ..
-rw-r--r--    1 tx0   tx0          0 Dec  4 17:56 file6
-rw-r--r--    1 tx0   tx0          0 Dec  4 17:56 file7
$
```

Notate che `~/directory/dir1` non ci consente nemmeno di tentare la lettura generando un errore sulla directory stessa; 2 ci consente il tentativo, ma per ciascun file o directory contenuto genera un errore; 3 invece ci permette il listato dei file. I permessi delle 3 directory sono infatti nell'ordine 700, 744 e 755 (l'ultima leggibile ed *eseguibile* a tutti).

5.2 Creare archivi con tar

Un archivio è un file che contiene più file al suo interno, organizzati in modo da preservarne contenuto, dimensione, permessi, proprietà e dati di creazione e di accesso. È l'equivalente Unix di un file prodotto con `pkzip` o `arj` sotto DOS ma con una differenza: gli archivi Unix non sono compressi.[4]

Il più comune programma per la creazione di archivi sotto Unix è `tar`.[5] La sintassi è la seguente:

```
tar <crtux> [OPZIONI] [file da inserire]
```

Prima di qualsiasi altra eventuale opzione specificata, `tar` richiede che sia specificato un comando che lo istruisca su come comportarsi. Non più di un comando alla volta. Quindi si possono specificare una serie di opzioni per modificare il comportamento dell'archiviatore. Infine è necessario comunicare a `tar` l'elenco dei file da includere nell'archivio. Riassumiamo i comandi e le opzioni fondamentali:

Comando	Azione collegata
-c	Crea un nuovo archivio con i file specificati
-r	Appende i file ad un archivio esistente
-t	Mostra il contenuto dell'archivio elencando o file uno ad uno
-u	Aggiorna l'archivio includendo solo i file modificati più di recente rispetto alla copia presente nell'archivio
-x	Estrae l'archivio

Opzione	Cosa modifica
-f *file*	Scrive l'archivio in *file* anziché sul device specificato da `$TAPE`
-h	Anziché inserire nell'archivio il contenuto dei file che puntati da link, scrive il link al file
-v	Aumenta il livello di messaggi diagnostici forniti
-w	Chiede conferma per ogni azione
-X *file*	Esclude i file listati in *file*

[4] Questo non vuol dire che non possano essere compressi. Vedi a proposito di seguito

[5] Il cui nome è la contrazione di *t ape a rchiver* in quanto in origine pensato per la produzione di archivi solo su unità a nastro *(tape appunto)* e successivamente modificato per poter generare anche archivi dentro file su disco

-g	Comprime l'archivio usando `gzip`
-b	Comprime l'archivio usando `bzip2`
-Z	Comprime l'archivio usando `compress`

Facciamo alcuni esempi:

```
$ tar -cf works.tar works/
$ ls -1
works/
works.tar
$
```

Analizziamo la sintassi usata. La prima lettera è correttamente un comando. **-c** crea un nuovo archivio da zero. Di seguito abbiamo **-f**, che è un'opzione, che imposta il nome dell'archivio a "works.tar".[6] Infine chiudiamo la riga con l'elenco dei file da includere nell'archivio: la directory `works/`. Includere una directory significa includere anche tutti i file e le directory in essa contenuti. Un `ls` sulla directory corrente ci confermerà la creazione dell'archivio.

Se volessimo essere sicuri di quello che stiamo facendo potremmo includere l'opzione `-v` per ottenere un listato di tutti i file contenuti nell'archivio appena creato:

```
$ tar -cvf works.tar works/
works/file1
works/file2
works/file3
works/file4
works/file5
$
```

Vogliamo ora controllare il contenuto dell'archivio creato:

[6] Notate il suffisso o estensione ".tar" per indicare che il file è un archivio creato con `tar`

```
$ tar tf works.tar
works/file1
works/file2
works/file3
works/file4
works/file5
$
```

`tar` ci mostra il contenuto (con il comando `-t`) dell'archivio contenuto nel file works.tar. L'assenza del carattere "-" davanti alla lista di opzioni non è una svista. I parametri possono essere passati a `tar` anche senza questa notazione.

Tempo dopo abbiamo creato anche il file *file6* e vogliamo aggiungerlo all'archivio:

```
$ tar rf works.tar works/file6
$ tar tf works.tar
works/file1
works/file2
works/file3
works/file4
works/file5
works/file6
$
```

In seguito sarà sufficiente dare il comando `u` per aggiornare il contenuto dell'archivio con i file modificati dall'ultima archiviazione. Se fosse invece necessario estrarre il contenuto dell'archivio sarebbe sufficiente il comando `r`. I file verranno estratti nella directory in cui ci si trova, dentro la quale sarà creata la directory `works` e qui posizionati i file. Non pensate che `tar` estragga i file lì dove li avete presi per creare l'archivio. **Nota di compatibilità:** `tar` di Linux rimuove automaticamente lo slash iniziale dai nomi completamente qualificati (i nomi che iniziano con uno slash), mentre altri Unix non lo fanno. Quindi quando estraete un archivio che non avete creato voi o del quale non siete sicuri, eseguite *sempre* un `tar tf` *archivio.tar* per essere certi che i vostri file vengano scritti nel punto giusto del filesystem.

5.3 Comprimere file con gzip e bzip2

gzip e bzip2 sono due programmi di compressione. Il loro scopo è quello di applicare ad un file un algoritmo che ne ricavi una versione di dimensioni ridotte[7], più veloci da trasferire e meno ingombranti da salvare altrove, dai quali si possa riottenere il file originale tramite l'applicazione di un algoritmo inverso *(decompressione)*. L'uso combinato di un programma di compressione ed un programma di archiviazione consente di ottenere *backup* precisi, completi, comodi da usare e soprattutto compatti.

gzip è il programma di compressione ufficiale del progetto GNU. bzip2 è un compressore più recente è più potente (può dare differenze del 5% sul risultato finale) anche se leggermente meno diffuso. Le opzioni sono tuttavia talmente simili che imparare ad usare entrambi i programmi non è complesso e confusionario, ed è conveniente avere una copia di ciascuno sul proprio computer.

Vediamo come è possibile comprimere un archivio:

```
$ tar cf works.tar works/
$ gzip works.tar
$ ls -1
works/
works.tar.gz
$
```

Il file *works.tar.gz* è un archivio tar compresso con gzip (suffisso *".tar.gz"* a volte contratto in *".tgz"*). Un metodo alternativo per comprimere un archivio senza passare per il file ".tar" è il seguente:

```
$ tar cf - works.tar | gzip > works.tar.gz
$ ls -1
works/
works.tar.gz
$
```

[7] Il compresso piò risultare grande dal 1% al 99.9%

Comprimere file con `gzip` e `bzip2`

Il risultato è lo stesso ma con un comando in meno, un quanto abbiamo usato la redirezione della shell per accorparne due. Ora vediamo in dettaglio come la cosa abbia funzionato.

`tar cf - works/`
: Il comando `tar` crea un archivio contenente i file works/* e lo redirige allo STDOUT. Il simbolo "-" usato in una riga di shell significa STDIN/STDOUT a seconda della direzione che i dati assumono. `tar` crea file in questo caso quindi la direzione è STDOUT.

`gzip > works.tar.gz`
: `gzip` accetta come dati sui quali lavorare l'output prodotto da `tar` e ne crea una versiona compressa. La shell pensa quindi a redirigere quello che altrimenti andrebbe a video verso un file di nome *works.tar.gz*

Terza possibilità:

```
$ tar czf works.tar.gz works/
```

L'opzione `-z` di `tar` usa in automatico `gzip` per comprimere il file risultante. A noi resta solo l'accortezza di aggiungere l'estensione .gz al file. In caso si usi `bzip2` al posto di `gzip` il suffisso da appendere al nome del file è `.bz2` (**non** contraibile in .tbz2). Per quanto la terza opzione si sicuramente la più semplice, non ci consente di specificare alcuna opzione per il programma di compressione. Ecco quali sono le più utili per entrambi:

Opzione	Significato
-r	Comprime i file ricorsivamente (solo `gzip`)
-[1-9]	Imposta il livello di compressione dal minimo (-1) al massimo (-9)
-v	Produce un rapporto sul livello di compressione di ciascun file compresso

`gzip` non è utile solo in associazione ad un programma di archiviazione. È possibile ad esempio comprimere tutti i file presenti nella directory in cui ci si trova con il semplice:

```
$ gzip *
$
```

Se volessimo comprimere tutti i file (inclusi quelli nelle sottodirectory) al massimo e sapere quanto ha inciso il processo di compressione potremmo usare:

```
$ gzip -r9v *
corsoUnix.aux:        71.2% -- replaced with corsoUnix.aux.gz
corsoUnix.dvi:        60.9% -- replaced with corsoUnix.dvi.gz
corsoUnix.log:        73.5% -- replaced with corsoUnix.log.gz
corsoUnix.tex:        61.3% -- replaced with corsoUnix.tex.gz
corsoUnix.toc:        70.1% -- replaced with corsoUnix.toc.gz
shell/la_shell.aux:   66.6% -- replaced with shell/la_shell.aux.gz
shell/la_shell.tex:   54.7% -- replaced with shell/la_shell.tex.gz
$
```

Per estrarre un archivio compresso sono possibili le seguenti:

```
$ tar xzf works.tar.gz
$
```

oppure:

```
$ gzip -d works.tar.gz | tar xf -
$
```

oppure:

```
$ gunzip works.tar.gz | tar xf -
$
```

delle quali la prima è sicuramente la più semplice.

5.4 Dividere gli archivi con `split`

Un archivio (anche compresso) può avrere una dimensione scomoda da gestire (può ad esempio essere troppo grosso per essere salvato su un solo floppy). In questo caso `split`

Dividere gli archivi con `split`

ci viene in aiuto: divide i file secondo la dimensione da noi specificata. La sintassi di `split` è la seguente:

```
split [OPZIONI] [INPUT [PREFIX]]
```

Le opzioni più comuni sono:

Opzioni	Significato
-b *bytes*	Divide in parti di *bytes* byte
-l *lines*	Divide ogni *lines* linee

La dimensione dopo `-b` può essere indicata in blocchi da 512 byte *(b)*, da 1 kylobyte *(k)* o da 1 megabyte *(m)*, usando l'opportuna unità dopo il valore.

Decidiamo di dividere il file *big.tar* di 2 mega in due parti da un mega ciascuna:

```
$ ls -l big.tar
-rw-r--r--    1 tx0   tx0    2097152 Dec  4 23:09 big.tar
$
$ split -b1m big.tar big.tar.
$ ls -l big.tar*
-rw-r--r--    1 tx0   tx0    2097152 Dec  4 23:09 big.tar
-rw-r--r--    1 tx0   tx0    1048576 Dec  4 23:10 big.tar.aa
-rw-r--r--    1 tx0   tx0    1048576 Dec  4 23:10 big.tar.ab
$
```

Split ha creato due file (*big.tar.aa* e *big.tar.ab*) di un mega ciascuno, usando come sorgente big.tar e usando come prefisso dei nomi "big.tar.", ai quali ha poi aggiunto un suffisso progressivo come questi:

```
aa ab ac ad ... az ba bb bc ... bz ca cb ...
      vv vz za zb zc zd ... zv zz
```

Per riottenere il nostro file originale possiamo usare una serie di `cat`:

```
$ cat big.tar.aa > big.tar
$ cat big.tar.ab >> big.tar
$ ls -l big.tar
-rw-r--r--    1 tx0    tx0    2097152 Dec  4 23:09 big.tar
$
```

Attenzione: il primo `cat` crea un nuovo file (`>`) azzerando un eventuale file presente; il secondo `cat` (e eventuali successivi) usano un append (`>>`) per non riazzerare il file ma per accodare il contenuto di *big.tar.ab* a quello creato dal precedente.

5.5 Ai piedi dei file e oltre: `tail`, `sort`

`tail` consente di visualizzare le ultime 10 righe di un file istantaneamente. Il numero di righe è modificabile attraverso il parametro `-n` *lines*. La funzione più interessante di questo tool tuttavia è la possibilità di leggere all'infinito un file, mostrando ogni nuova riga di testo venga introdotta in coda. Questo sistema è particolarmente quando vi trovate nell'esigenza di consultare in tempo reale un file di log, nel quale un programma stà producendo output per informarvi dello stato dell'esecuzione o di quali operazioni stia compiendo. Si ottiene usando l'opzione `-f`. Ad esempio:

```
$ tail -n20 -f /tmp/logfile
$
```

vi mostrerà da subito le ultime 20 righe del file */tmp/logfile* e quindi attenderà all'infinito l'inserimento di nuove righe all'interno del file. Potete interrompere la lettura con un `Control-C`.

`sort` invece ordina le linee di un file secondo alcuni possibili criteri. Diciamo che nella nostra home sia presente un file con i numeri di telefono dei nostri amici per nome e numero. E diciamo che ne vogliamo una versione ordinata. Ad esempio:

```
$ cat tel_num
Tx0       02/7412309676
Shodan    02/753207213
Manhattan 02/987435213321
Bluca     02/0xa65db78c86
lidl      02/76453129982
Zeist     01337/34
$
$ sort tel_num
Bluca     02/0xa65db78c86
Manhattan 02/987435213321
Shodan    02/753207213
Tx0       02/7412309676
Zeist     01337/34
lidl      02/76453129982
$
```

Il sorting è avvenuto in ordine alfabetico ma con distinzione fra maiuscole e minuscole. Uhm, vediamo di ottenere un sorting *case insensitive* :

```
$ sort -f tel_num
Bluca     02/0xa65db78c86
lidl      02/76453129982
Manhattan 02/987435213321
Shodan    02/753207213
Tx0       02/7412309676
Zeist     01337/34
$
```

Et voilá. Questa volta lidl è al suo posto.

Le opzioni di `sort` sono molte e vanno al di là degli obiettivi di questo corso, quindi `man sort` e cercate da soli quello che vi serve! `:-)`

5.6 Ricerche su testo con grep

grep consente di eseguire ricerche sulla base di un *pattern di ricerca*. Il pattern segue i principi delle regexpr. La sintassi è la seguente:

```
grep [OPZIONI] PATTERN [FILE]
```

Il pattern è il solo elemento necessario. Le opzioni servono a modificare il funzionamento dei grep e soprattutto l'output generato dal comando. Il comando può funzionare sia su file su disco che su stream di output di altri comandi. Ad esempio se volessimo vedere tutte le directory presenti nella nostra home directory, potremmo usare il comando:

```
$ ls -l ~ | grep "^d"
drwxr-sr-x    2 tx0   tx0       4096 Sep 23 18:51 CorsoUnix
drwxr-sr-x    5 tx0   tx0       4096 Nov 28 01:51 LaTeX
drwx--S---    2 tx0   tx0       4096 Oct 31 00:16 Mail
drwxr-sr-x    9 tx0   tx0       4096 Oct  9 14:50 Perl
drwx--S---    2 tx0   tx0       4096 Apr 18  2000 mail
$
```

In questo caso grep ha eseguito una ricerca per il pattern ^d sull'output di ls, trovando le sole directory in quanto solo quelle generano una linea che inzia con una d.

Diciamo che vogliamo cercare tutti i file tar.gz che siano collocati in /tmp/. Una possibile soluzione è quella di utilizzare find /tmp -regex ".*targz". Questa possibilità ha però lo svantaggio di lavorare direttamente sul disco. Alternativamente è consigliabile un:

```
$ locate tar.gz | grep /tmp
/tmp/archive.tar.gz
/tmp/backup.tar.gz
/usr/local/tmp/old_backup.tar.gz
$
```

Raffinando ultreriormente il criterio di ricerca potremmo optare per:

```
$ locate tar.gz | grep "^/tmp"
/tmp/archive.tar.gz
/tmp/backup.tar.gz
$
```

il che vincola i match ai soli file contenuti nella directory `/tmp` o sottostanti.
`grep` non è come già detto utile solo su output di altri comandi ma anche nella ricerca all'interno di file. Diciamo di voler cercare tutte le occorrenze della parola *sql* dentro `/var/log`.[8]

```
$ grep -r sql /var/log/*
/var/log/mysql.err:mysqld started on  Sat Dec 9 19:26:48 CET 2000
/var/log/mysql.err:/usr/sbin/mysqld: ready for connections
/var/log/mysql.err:001209 20:12:56  /usr/sbin/mysqld: Normal shutdown
/var/log/mysql.err:001209 20:12:56  /usr/sbin/mysqld: Shutdown Complete
/var/log/mysql.err:mysqld ended on  Sat Dec 9 20:12:56 CET 2000
/var/log/mysql.err:mysqld started on  Mon Dec 11 14:54:46 CET 2000
/var/log/mysql.err:/usr/sbin/mysqld: ready for connections
$
```

`grep` ha operato una ricerca della stringa `sql` (che non richiedeva commento in quanto non contiene metacaratteri) su tutti i file posti nella directory `/var/log` e sue subdirectory. Ha trovato occorrenze in `/var/log/mysql.err` ed in `/var/log/syslog`. Sarebbe tuttavia più utile poter determinare in quali linee si siano verificati i match:

```
$ grep -rn sql /var/log/*
/var/log/mysql.err:605:mysqld started on  Sat Dec 9 19:26:48 CET 2000
/var/log/mysql.err:606:/usr/sbin/mysqld: ready for connections
/var/log/mysql.err:607:001209 20:12:56  /usr/sbin/mysqld: Normal shutdown
/var/log/mysql.err:609:001209 20:12:56  /usr/sbin/mysqld: Shutdown Complete
/var/log/mysql.err:611:mysqld ended on  Sat Dec 9 20:12:56 CET 2000
/var/log/mysql.err:612:mysqld started on  Mon Dec 11 14:54:46 CET 2000
/var/log/mysql.err:613:/usr/sbin/mysqld: ready for connections
$
```

8 Larga parte dell'output prodotto dal comando è stata cancellata per evitare che riempisse alcune pagine. In realtà è prevedibile che vengano generate diverse centinaia di righe di output da comandi di quasta natura

Come possiamo osservare, dopo ciascun nome di file `grep` ha introdotto il numero della linea alla quale la parola `sql` è stata trovata.

Vediamo una panoramica delle opzioni di `grep`:

Opzioni	Significato
-A *num*	Stampa le *num* linee seguenti ciascuna riga contenente un match
-B *num*	Stampa le *num* linee precedenti ciascuna riga contenente un match
-C *num*	Stampa *num* linee precedenti e seguenti ciascuna riga contenente un match
-c	Stampa il totale di match per ciascun file fornito anziché l'elenco dei match
-f *file*	Ottiene i pattern dal file *file*, uno per ogni linea
-h	Omette il nome dei file nell'output
-i	Cerca senza badare a maiuscole e minuscole
-n	Mostra anche il numero di ciascuna riga di ciascun file che contiene un match
-r	Percorre ricorsivamente le directory
-v	Inverte la ricerca

5.7 I processi e la loro gestione: `kill`, `top`, `ps` e `uptime`

Unix è un sistema operativo *multitasking*. Questo significa che più programmi possono *girare* contemporaneamente sullo stesso computer. Ciascuno di questi programmi è definito *processo*. Il sistema operativo si incarica di ripartire le risorse della macchina tra i processi, facendone funzionare uno alla volta per un breve periodo in modo che ciascuno di essi avanzi di pari passo nell'esecuzione.

Ciascun processo ha un codice di identificazione detto **PID** *(Process IDentification)*. Usando il PID di un processo è possibile inviare a questo processo delle informazioni sotto forma di *segnali*. Ciascun segnale viene interpretato dal processo secondo la sua programmazione, ma alcuni di essi sono standard e si possono riassumere così:

```
 1 HUP       2 INT      3 QUIT     4 ILL      5 TRAP     6 ABRT     7 BUS
 8 FPE       9 KILL    10 USR1    11 SEGV    12 USR2    13 PIPE    14 ALRM
15 TERM     16 STKFLT  17 CHLD    18 CONT    19 STOP    20 TSTP    21 TTIN
22 TTOU     23 URG     24 XCPU    25 XFSZ    26 VTALRM  27 PROF    28 WINCH
29 POLL     30 PWR     31 SYS
```

Attenzione: i segnali variano notevolmente da Unix a Unix; quelli qui riportati sono quelli di Linux. Prima di usarli consultate `man 7 signal` oppure usate `kill -l` per ottenere un output analogo a quello qui sopra riportato.

kill 5.7.1

Il comando utilizzato per inviare segnali ad un processo è `kill`:

```
$ kill [SEGNALE] PROCESSO [PROCESSO] [...]
```

Nato per uccidere, `kill` è in realtà un comando generico che invia un qualsiasi segnale (e non solo il KILL) ad un numero arbitrario di processi.

Commentiamo i tre segnali più utilizzati di frequente e che sono necessaria conoscenza anche del semplice utente:

Segnale	Numero	Significato
HUP	1	Informa il processo che la tty alla quale era collegato si è staccata.
KILL	9	Termina il processo istantaneamente
TERM	15	Chiede al processo di terminare secondo le sue procedure

Tutti questi segnali hanno come effetto la terminazione del processo. KILL tuttavia ha la particolarità di non essere interpretabile dal processo che lo esegue e di non poter essere ignorato. In pratica un `kill -11` (o `kill -TERM`) non comporta l'immediata cessazione del processo che può riservarsi tutto il tempo necessario a chiudere i file aperti, completare le sue procedure di terminazione e solo allora terminare. `kill -9` invece comporta l'immediata cessazione del processo senza appello.

`kill -HUP` ha invece la caratteristica di far ripartire il processo dopo la sua terminazione. Viene utilizzato da molti programmi come segnale per rileggere la configurazione dopo una modifica da parte dell'amministratore di sistema.

foreground o background? 5.7.2

I processi hanno anche un'altra caratteristica: possono girare in *foreground* oppure in *background*. Il primo caso si ha quando un processo occupa una tty per il tempo della sua esecuzione. Il secondo quando un processo viene lanciato senza vincolare alcuna tty.

Ad esempio, l'esecuzione del comando:

```
$ ls
corsoUnix.tex
la_shell.tex
piu_in_alto.tex
regexpr.tex
storia_Unix.tex
$
```

è avvenuta in foreground, occupando la tty alla quale ha restituito l'output per tutto il tempo di esecuzione. Nel caso di ls questo è semplicemente irrilevante, dato che il comando termina in una frazione di secondo. Ma un programma come una simulazione o un *demone* che provvede un particolare servizio, come un server http o ftp, girano decisamente meglio in background. Per lanciare un processo in background è sufficiente accodare una & *(e commerciale)* alla riga di comando. Ad esempio per chiamare il popolare browser netscape è possibile:

```
$ netscape &
[1] 687
$
```

La shell dalla quale abbiamo lanciato netscape ci ritorna subito il prompt e dopo qualche istante di caricamento appare la finestra di Netscape. Un metodo alternativo consiste nel lanciare un processo in foreground, interromperlo con un Control-Z e mandarlo in background con il comando bg:

```
$ netscape
^Z
[1]+  Stopped                 netscape
$ bg
$
```

La notazione ^Z indica la pressione contemporanea del tasto *Control* e del tasto *z*.

Le shell[9] offrono una gestione avanzata del *-ground* dei processi. Quando abbiamo lanciato *netscape* in background, la shell ci ha stampato un breve rapporto sul processo, includendo un numero fra parentesi quadre. Questo numero costituisce una più confortevole sistema rispetto al PID per gestire i processi. Precedendo questo numero con il simbolo % è possibile riferirsi a quel processo. Se ad esempio avessimo lanciato due processi in background come in:

```
$ netscape &
[1] 687
$ gimp &
[2] 691
$
```

sarebbe possibile portare in foreground netscape con il comando `fg %1` oppure terminarlo con il comando `kill %1`.[10]

top 5.7.3

Il comando più semplice per avere un'idea dei processi che stanno girando sul sistema è `top`. È un programma interattivo che mostra tutta la *process table*[11] e ci fornisce informazioni su ciascun processo ed un semplice sistema per inviare segnali ai processi. Vediamo una sessione di top e commentiamo le informazioni forniteci, avvalendoci di una numerazione delle linee per aiutarci:

9 Solo quelle più moderne, ossia più recenti del 1990

10 Questo è in realtà consentito dal fatto che il `kill` utilizzato non è il comando `/bin/kill` ma il comando kill *builtin* nella shell. Questo significa che la shell contiene delle funzionalità che le consentono di emulare `/bin/kill` ed offrire nuove possibilità come la gestione basata sul carattere %. Se non è chiaro il significato di builtin o di emulazione, non date peso a questa nota

11 Lista dei processi attivi sulla macchina

```
01:    3:43pm  up 49 min,  4 users,  load average: 0.00, 0.08, 0.11
02:  65 processes: 64 sleeping, 1 running, 0 zombie, 0 stopped
03:  CPU states: 10.3% user,  2.1% system,  0.0% nice, 87.5% idle
04:  Mem:   63552K av,  61016K used,   2536K free,  60736K shrd,   1616K buff
05:  Swap: 272144K av,   6384K used, 265760K free                 29260K cached
06:
07:    PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COMMAND
08:    596 tx0       16   0  1148 1148   684 R       0  5.3  1.8   0:00 top
09:    425 tx0        9   0  3984 3984  3092 S       0  3.5  6.2   1:43 deskguide_ap
10:    325 root       8   0  8556 3524   936 S       0  1.7  5.5   2:29 XF86_SVGA
11:    409 tx0        8   5  3588 3588  2804 S N     0  0.8  5.6   0:29 cpumemusage_
12:      1 root       0   0   464  464   404 S       0  0.0  0.7   0:05 init
13:      2 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kflushd
14:      3 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kupdate
15:      4 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kpiod
16:      5 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kswapd
17:     84 daemon     0   0   324  312   244 S       0  0.0  0.4   0:00 portmap
18:    153 root       0   0   596  596   480 S       0  0.0  0.9   0:00 syslogd
19:    155 root       0   0   412  408   284 S       0  0.0  0.6   0:00 klogd
20:    163 root       0   0   516  516   376 S       0  0.0  0.8   0:00 cardmgr
21:    172 root       0   0  1208  696   472 S       0  0.0  1.0   0:00 named
22:    182 root       0   0   500  500   424 S       0  0.0  0.7   0:00 rpc.statd
23:    188 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 lockd
24:    189 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 rpciod
```

La prima dice che sono le 15,43, la macchina è accesa da 49 minuti; ci sono 4 utenti sul sistema e i carichi del sistema sono 0.00, 0.08, 0.11.[12] La seconda riga fornisce una panoramica della process table: 65 processi sul sistema dei quali 64 non stanno operando nulla, uno solo è attivo, nessuno è uno zombie[13] e nessuno è stato bloccato.[14] La terza riga divide il tempo di calcolo assegnato ai processi di sistema o di utenti normali dal tempo non assegnato *(idle)*. La quarta e la quinta danno una summa della memoria e dello *swap*[15] fornendo nell'ordine i valori di: memoria totale disponibile, memoria utilizzata, memoria libera, memoria *condivisa da più processi* e memoria in buffer o in cache.

Dalla riga sette inizia l'elenco dei processi. Le colonne sono le seguenti:[16]

12 Rispettivamente carico minimo, medio e massimo

13 Un processo zombie è un processo che è morto senza che il sistema si riuscito a rimuoverlo dalla process table

14 Come dopo la pressione di ˆ Z

15 Memoria Virtuale: spazio su disco utilizzato *come* memoria RAM per sopperire a carenze di quest'ultima

16 Le colonne ignorate sono obsolete

Colonna	Significato
`PID`	Process ID: identificativo numerico univoco del processo
`USER`	Utente proprietario di quel processo
`PRI`	Priorità alla quale gira il processo
`NI`	*Nice* del processo (di più in seguito)
`SIZE`	Dimensione complessiva del processo
`RSS`	Dimensione del processo residente in memoria
`SHARE`	Dimensione della memoria condivisa dal processo
`STAT`	Stato del processo
`%CPU`	Percentuale di tempo CPU utilizzato dal processo
`%MEM`	Percentuale complessiva di memoria usata dal processo
`TIME`	Tempo complessivo di CPU usato dal processo dall'istante della partenza
`COMMAND`	Riga di comando con la quale è stato lanciato il processo

`top` fornisce una serie di comandi da utilizzare in maniera interattiva:

Comando	Significato
`k`	Invia un segnale ad un processo
`r`	Cambia il *nice* di un processo
`u`	Mostra solo un utente
`n`	Imposta in numero massimo di processi da mostrare
`s`	Imposta il tempo fra una misurazione e l'altra
`space`	Aggiorna i dati mostrati
`f/F`	Rimuove/Aggiunge colonne all'output
`o/O`	Cambia l'ordine con il quale le colonne sono mostrate
`S`	Attiva/Disattiva il *cumulative mode*
`i`	Include/Esclude i processi *idle*
`c`	Include/Esclude la *command line*
`l`	Include/Esclude le informazioni sul carico
`m`	Include/Esclude le informazioni sulla memoria
`t`	Include/Esclude le informazioni generali
`N`	Ordina per PID
`A`	Ordina per *età* del processo
`P`	Ordina per uso di CPU
`M`	Ordina per uso di memoria residente
`T`	Ordina per tempo (cumulativo)
`W`	Scrive un file di configurazione in `/.toprc`
`h`	Help
`q`	Esce

Abbiamo suddiviso i comandi fra quelli che *agiscono* sui processi, quelli che cambiano output a `top` e quelli che cambiano l'ordinamento nei processi nella lista.

Il comando sicuramente più utilizzato è `k` che permette di inviare un segnale ad un processo attraverso il suo PID, mostrato nella corrispondente colonna da `top`. Tuttavia di enorme utilità è anche `r`. Il suo scopo è variare il valore di *nice* di un processo. Questo valore determina il trattamento riservato al processo dal sistema all'atto della ripartizione delle risorse fra i processi. *nice* può variare da -20 a +19, dove -20 è la massima richiesta di priorità. In maniera concorde a quanto richiesto dal valore di *nice* richiesto, il sistema imposterà la priorità del processo (l'effettivo privilegio nell'accesso alle risorse della macchina) visibile nella colonna `PRI`. È implicito che il sistema consente di modificare il *nice* di un processo solo al proprietario di quel processo (e all'amministratore di sistema), così come solo il proprietario può terminare un processo.

Priorità e *nice* 5.7.4

La colonna `PRI` indica come già visto la priorità alla quale il processo stà girando. Questo valore viene impostato dal sistema e non è modificabile dall'utente.

La colonna `NI` indica invece il `nice` di un processo. Questo valore è modificabile dal proprietario del processo e dal sistemista ed indica la richiesta di risorse alla quale far girare il processo. Un utente può utilizzare solo nice positivi e può solo percorrere la scala verso nice minori (ossia più prossimi al 19). Quindi se un utente modifica il nice di un processo a 10 gli sarà consentito cambiarlo successivamente ad un valore compreso fra 11 e 19 e non inferiore a 10. Solo `root` avrà la possibilità di modificare in direzione opposta questo valore.

Esistono due comandi standard per specificare il livello di nice al quale far girare un processo. Il primo è `nice`, si utilizza in fase di avvio del comando con la sintassi:

```
$ nice -10 find ~ > ~/elenco_file &
$
```

In questo caso stiamo lanciando il comando con un nice pari a 10. Se invece volessimo ritoccare il nice di un processo `running` potremmo utilizzare il comando `renice` in questo modo:

```
$ ps -A | grep find
  488 pts/2    00:00:00 find
$ renice 19 488
488: old priority 10, new priority 19
$
```

Il comando ci informa circa l'avvenuto cambiamento, il precedente nice ed il nice attuale.

ps 5.7.5

L'alternativa a top per visualizzare i processi è ps. Rispetto al suo analogo interattivo, ps non offre una gestione dei segnali ai processi (permette solo di visionarli) e non aggiorna automaticamente l'elenco dei processi mano a mano che il tempo passa. Questo significa che per avere una stampa aggiornata dei processi occorre rilanciare il comando. Perché allora usare un comando apparentemente tanto scomodo?

Il motivo è che ps permette di conoscere in un batter d'occhio una mole incredibile di informazioni sui processi negata a top ed essendo un tool di riga di comando può passare i suoi dati in stream a grep o sort per particolari scopi.

Facciamo qualche prova:

```
$ ps
  ID TTY          TIME CMD
  443 pts/0    00:00:00 bash
  451 pts/0    00:00:22 vim
  932 pts/0    00:00:00 bash
  933 pts/0    00:00:00 ps
$
```

ps ci da come output quattro informazioni ormai familiari: il PID del processo, la tty alla quale è collegato, il tempo di calcolo richiesto e la command line. Vediamo di avere qualche informazione di più. Vediamo le opzioni di ps e proviamo a farcene un'idea. Prima però un'ultima nota: le opzioni di ps sono le più bizzarre che vi possa capitare di incontrare, più ancora di quelle di tar. La stessa lettera ha un significato completamente diverso se preceduta da un meno (-) oppure no. Vediamole:

Opzione	Significato
-A	Tutti i processi
-a	Tutti i processi su una tty
T	Tutti i processi su questo terminale
a	Tutti i processi collegati a un terminale
r	Solo processi attualmente operanti *(running)*
x	Tutti i processi senza una tty
-H	Mostra i processi gerarchicamente
o	Formato definito dall'utente
u	Formato studiato per l'utente
c	Solo nome dei comandi e non command line

Vediamo un po' di esempi:

```
$ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:05 init
    2 ?        00:00:00 kflushd
    3 ?        00:00:00 kupdate
    4 ?        00:00:00 kpiod
    5 ?        00:00:03 kswapd
   84 ?        00:00:00 portmap
  152 ?        00:00:00 syslogd
  154 ?        00:00:00 klogd
  180 ?        00:00:00 named
  189 ?        00:00:00 rpc.statd
  196 ?        00:00:00 lockd
  197 ?        00:00:00 rpciod
  220 ?        00:00:00 inetd
  275 ?        00:00:00 sendmail
  283 ?        00:00:00 atd
  286 ?        00:00:00 cron
  326 tty4     00:00:00 getty
  327 tty5     00:00:00 getty
  328 tty6     00:00:00 getty
  435 pts/0    00:00:00 bash
  500 pts/0    00:00:01 vim
  501 pts/0    00:00:00 bash
  502 pts/0    00:00:00 ps
$
```

`ps` ha riportato l'elenco completo dei processi che girano sul sistema.[17] Cerchiamo di mettere in evidenza le dipendenze dei processi (ovvero quali processi siano *genitori* di quali processi); usiamo per questo l'opzione `-H`:

17 Molti processi sono stati tolti per esigenze di spazio e per non complicare troppo l'esempio

```
$ ps -AH
  PID TTY          TIME CMD
    1 ?        00:00:05 init
    2 ?        00:00:00   kflushd
    3 ?        00:00:00   kupdate
    4 ?        00:00:00   kpiod
    5 ?        00:00:03   kswapd
   84 ?        00:00:00   portmap
  152 ?        00:00:00   syslogd
  154 ?        00:00:00   klogd
  162 ?        00:00:00   cardmgr
  180 ?        00:00:00   named
  189 ?        00:00:00   rpc.statd
  196 ?        00:00:00   lockd
  197 ?        00:00:00     rpciod
  220 ?        00:00:00   inetd
  275 ?        00:00:00   sendmail
  283 ?        00:00:00   atd
  286 ?        00:00:00   cron
  326 tty4     00:00:00   getty
  327 tty5     00:00:00   getty
  328 tty6     00:00:00   getty
  435 pts/0    00:00:00   bash
  500 pts/0    00:00:03     vim
  504 pts/0    00:00:00   bash
  505 pts/0    00:00:00     ps
$
```

-H ci fornisce un output organizzato gerarchicamente, in cui i processi sono ravvicinati e spostati progressivamente verso destra per indicare padri e figli. Possiamo notare come tutti i processi dipendano dal processo init;[18] Più in basso notiamo come stiano girando due shell bash che hanno rispettivamente lanciato l'editor di testo vim e il comando ps.[19]

Altra cosa notevole è il ? che viene stampato nel campo della TTY per i processi che non sono collegati ad alcuna tty Proviamo ad estendere l'output con l'opzione o:

18 init è il processo principale che parte per primo all'avvio del sistema; ogni processo lanciato di seguito ne è un discendente. Per questo il PID di init è sempre 1

19 Proprio il ps che ha mostrato a video tutti i processi. Dato che un processo prima di fare qualsiasi cosa viene prima di tutto registrato nella process table, quando ps richiede il contenuto della process table per mostrarlo trova anche se stesso

```
$ ps -AH o user,pid,pcpu,rss,cmd
USER       PID %CPU  RSS CMD
root         1  0.0   64 init [2]
root         2  0.0    0   [kflushd]
root         3  0.0    0   [kupdate]
root         4  0.0    0   [kpiod]
root         5  0.0    0   [kswapd]
daemon      84  0.0    0   [portmap]
root       152  0.0  208   /sbin/syslogd
root       154  0.0    0   [klogd]
root       180  0.0  632   /usr/sbin/named
root       189  0.0    0   [rpc.statd]
root       196  0.0    0   [lockd]
root       197  0.0    0     [rpciod]
root       220  0.0    0   [inetd]
daemon     283  0.0   52   /usr/sbin/atd
root       286  0.0  160   /usr/sbin/cron
root       326  0.0    0   [getty]
root       327  0.0    0   [getty]
root       328  0.0    0   [getty]
tx0        435  0.0  548   -bash
tx0        500  0.1 2976     /usr/bin/vim -o corsoUnix.tex shell.tex
tx0        553  0.0  804         ps -AH o user,pid,pcpu,rss,cmd
tx0        542  0.1 1216   -bash
$
```

Abbiamo omesso il tempo impiegato dai processi e la tty alla quale i processi sono collegati, mentre abbiamo aggiunto la percentuale di cpu usata e il nome dell'utente proprietario del processo e la dimensione residente dei processi (RSS). Inoltre abbiamo richiesto la command line completa anziché il solo nome dei processi.

In conclusione il miglior consiglio che vi possiamo dare è consultare la man page di ps e fare un buon numero di sperimentazioni in materia.

uptime 5.7.6

uptime è un comando utile per conoscere la situazione alcuni dati generali sul funzionamento del computer da quando è stato acceso. Ad esempio:

```
$ uptime
 11:51:57 up 18 days, 19 min,  6 users,  load average: 0.56, 0.46, 0.37
$
```

Le informazioni che uptime ci restituisce sono intuitive: prima di tutto l'ora corrente (11:51:57), quindi il tempo di funzionamento (diciotto giornie 19 minuti), il numero di utenti collegati al sistema in quel momento (6), e il carico. Quest'ultimo è uno dei parametri con cui si valuta il lavoro al quale il computer è sottoposto. Il tre "load" indicati nell'ordinesono il massimo, il medio ed il minimo. Un punto di carico equivale a dire che la CPU ha lavoro a sufficienza per riempire il suo naturale ciclo di calcolo della durata di un secondo. Per dirla in altro modo, nell'arco di un secondo la CPU non ha tempo di eseguire un NOP, ossia un'istruzione vuota, che non fa nulla, che viene abitualmente "eseguita" nelle pause di elaborazione.

5.8 I Device: significato ed uso

I *device* sono le periferiche, la strumentazione, più in generale l'*hardware* con il quale UNIX può interagire. Ad esempio uno scanner A4 è un device, un mouse è un device, ma anche un disco fisso interno o una scheda di rete sono un device.

Il sistema con cui UNIX controlla tutti questi giocattoli è molto diverso rispetto a quello usa sotto altri Sistemi Operativi. UNIX è infatti fortemente improntato alla gestione dei file (come forse avrete già notato) e quindi è stato progettato per utilizzare anche altre periferiche, oltre agli hard disk. come fossero dei file.

Per far questo UNIX utilizza un tipo speciale di file che è stato poi definito esso stesso device. Un device file può essere di due tipi: *character device* o *block device* . I primi differiscono dai secondi in quanto NON utilizzano una cache per bufferizzare i dati. Ossia scrivono i dati sulla periferica *un carattere alla volta* , da cui il nome di character device. Al contrario i block device scrivono i dati usando un buffer ossia un "blocco di dati", da cui il loro nome.

Andiamo a dare un'occhiata dietro le quinte:

```
$ ls -l /dev/hda*
brw-rw----    1 root     disk       3,   0 Nov 30  2000 /dev/hda
brw-rw----    1 root     disk       3,   1 Nov 30  2000 /dev/hda1
brw-rw----    1 root     disk       3,  10 Nov 30  2000 /dev/hda10
brw-rw----    1 root     disk       3,  11 Nov 30  2000 /dev/hda11
brw-rw----    1 root     disk       3,  12 Nov 30  2000 /dev/hda12
brw-rw----    1 root     disk       3,  13 Nov 30  2000 /dev/hda13
brw-rw----    1 root     disk       3,  14 Nov 30  2000 /dev/hda14
brw-rw----    1 root     disk       3,  15 Nov 30  2000 /dev/hda15
brw-rw----    1 root     disk       3,  16 Nov 30  2000 /dev/hda16
brw-rw----    1 root     disk       3,  17 Nov 30  2000 /dev/hda17
brw-rw----    1 root     disk       3,  18 Nov 30  2000 /dev/hda18
brw-rw----    1 root     disk       3,  19 Nov 30  2000 /dev/hda19
brw-rw----    1 root     disk       3,   2 Nov 30  2000 /dev/hda2
brw-rw----    1 root     disk       3,  20 Nov 30  2000 /dev/hda20
brw-rw----    1 root     disk       3,   3 Nov 30  2000 /dev/hda3
brw-rw----    1 root     disk       3,   4 Nov 30  2000 /dev/hda4
brw-rw----    1 root     disk       3,   5 Nov 30  2000 /dev/hda5
brw-rw----    1 root     disk       3,   6 Nov 30  2000 /dev/hda6
brw-rw----    1 root     disk       3,   7 Nov 30  2000 /dev/hda7
brw-rw----    1 root     disk       3,   8 Nov 30  2000 /dev/hda8
brw-rw----    1 root     disk       3,   9 Nov 30  2000 /dev/hda9
$
```

Prima di tutto la directory `/dev/`. Esiste su tutti i sistemi UNIX, eppure è solo una convenzione. Un device[20] può essere creato in qualsiasi punto del filesystem. Converrete però che un directory che li raggruppi tutti sia più comoda da gestire. In effetti quello che distingue un device da un file "normale" è il modo con il quale è stato creato.

Quelli che stiamo vedendo sono i device che rappresentano il primo disco IDE su una macchina Linux. I dischi fissi IDE sono rappresentati da `hd`; la lettera `a` subito dopo dice che è il primo disco (ossia il master del primo canale), mentre il numero che può seguire indica la partizione.

Così, la seconda partizione (2) del secondo disco IDE (b) si indica per convenzione con il device `/dev/hdb2`.

I dischi fissi sono sempre gestiti da block device per questioni di performance anche se alcuni sistemi operativi (come Solaris) danno anche una versione character per operazioni a basso livello.[21] Infatti la prima lettera del listato non è `-` come per i file normali o `d` come per le directory, bensì `b` che sta per block device.

Già ma, se tutto è solo una convenzione, il nome del file, la directory dove si trova, come fa UNIX a sapere che quel device riguarda propri quella partizione di quel disco e

20 Da qui in avanti, quando parleremo di device intenderemo il file che lo rappresenta e non il device stesso.

21 . . . come stregoneria ed esorcismo di filesystem difettosi

non la rotella del mouse o gli occhialini 3D? La soluzione è nei due numeri che si trovano prima della data di accesso. Sono definiti *major number* e *minor number* .

Il primo indica il driver da utilizzare mentre il secondo specifica su cosa questo driver vada utilizzato. Notate infatti che per tutti i device sul primo controller IDE (`/dev/hda*` e `/dev/hdb*` il major number è sempre 3. Il minor number invece incrementa progressivamente di uno per ogni dispositivo o partizione che incontriamo.

Ad esempio le porte seriali, che il DOS chiama COM ports, sono:

```
$ ls -l /dev/ttyS*
crw-rw----    1 root     dialout    4,  64 Nov 30  2000 /dev/ttyS0
crw-rw----    1 root     dialout    4,  65 Nov 30  2000 /dev/ttyS1
crw-rw----    1 root     dialout    4,  66 Nov 30  2000 /dev/ttyS2
crw-rw----    1 root     dialout    4,  67 Nov 30  2000 /dev/ttyS3
$
```

Vedete che alcune cose cambiano. Ad esempio il major number è 4 per tutte le seriali, mentre per i dischi ide e' 3 o 22 etc. . . Questo specifica che, pure se per l'utente una porta seriale e un disco fisso sono sempre dei file su disco, in reatà il driver che li gestisce è molto differente.[22]

Ma soprattutto notate come i device delle seriali siano dei *character device* , come indica la lettera `c` all'inizio di ciascuna linea.

I parametri necessari a descrivere un device vengono specificati al momento della creazione. Per generare un nuovo device si usa un comando particolare: `mknod`. Ad esempio per generare il device `/home/tx0/discoide2` in tutto analogo a `/dev/hda2`:

```
$ ls -l /dev/hda2
brw-rw----    1 root     disk       3,   2 Nov 30  2000 /dev/hda2
$ mknod /home/tx0/discoide2 b 3 2
$
```

Comunque questa è una operazione normalmente svolta dall'amministratore di sistema. Se volete ulteriori informazioni sul comando `mknod` leggetene la manpage.

[22] Un driver è una porzioni di codice, di programma che serve a *pilotare* una periferica. Così un driver per un mouse saprà recuperare le informazioni che questo passa attraverso al porta seriale o la porta PS/2, mentre un driver per un plotter a penne saprà comunicare al plotter quando alzare o abbassare la penna e come spostarla sul foglio

6. Regular Expressions

di Tx0

Le Regular Expressions sono un sistema di regole rivolte alla creazione di pattern di ricerca utili a trovare occorrenze all'interno di un testo e a modificarle con altro testo. Sono una "lingua franca" in quanto la maggior parte dei tool e dei linguaggi di progammazione sotto UNIX utilizzano questo sistema di produrre ed applicare schemi per la ricerca e la modifica di stringhe o intere porzioni all'interno dei documenti testuali.

6.1 Perché le Regular Expression?

Partiamo dagli elementi più semplici. La parola *pattern* è traducibile in italiano con *schema*, anche se questa non rende completamente il senso del vocabolo inglese. Si avvicina comunque abbastanza da permetterci di comprendere cosa significhi in questo contesto. Uno schema di ricerca è una sequenza di catatteri singoli o combinazioni di più caratteri utile a descrivere la struttura di un insieme di parole (ma non solo, vedremo oltre) per consentirne l'individuazione all'interno di un testo ed a descrivere un eventuale criterio di sostituzione di questo insieme con altro testo.

È forse già più semplice capire cosa si intende per schema. Se non lo è la trattazione successiva fornirà un numero progressivamente maggiore di nozioni per comprendere il termine. Per tutto il capitolo è importante comunque ricordare che stiamo affrontando un insieme di regole logiche e quindi in parte astratte dall'esperienza quotidiana. Non per questo le Regular Expressions sono meno efficaci ed utili. Dovete solo pazientare prima di cominciare a comporre autonomamente espressioni, in quanto la teoria è un po' estesa. Se vi sentite scoraggiati ricordate che alla fine dello studio riuscirete a dominare il sistema di ricerca e sostituzione più potente mai creato. Tanto potente che, pur essendo parto della cultura UNIX, si è diffuso su tutte le piattaforme ed oggi anche i più comuni programmi utilizzano le Regular Expression.[1] E poi la magia ha il suo fascino, no?

[1] Programmi come HomeSite, supportano le Regular Expression

Due convenzioni, molti meno problemi 6.1.1

Il nome Regular Expression viene abitualmente contratto nel più conciso **RegExpr**. D'ora in avanti useremo questo termine che risulta anche più veloce da pronunciare.

La seconda, ben più rilevante, convenzione consiste nell'includere una Regular Expression fra una coppia di slash (es. `/regexp/`). Vedremo oltre come questa convenzione non sia solo fra esseri umani ma anche fra utente e macchina.

6.2 La più semplice Regular Expression

In una regexpr ciascun carattere ha un ruolo preciso. Esistono molti caratteri con ruoli (o significati) particolari, ma la prima cosa da imparare è che la maggior parte dei caratteri alfanumerici è qui quello che è in qualsiasi lingua del mondo: un carattere!

Tutto ciò significa che il carattere `a` rappresenta la lettera "a" e (così com'è) non ha altre interpretazioni. Quindi, volendo scrivere la regular expression che consente di cercare tutte le occorrenze della lettera "a" all'interno di un documento basterà comporre:

```
/a/
```

Immediata conseguenza è che per cercare le occorrenze della coppia di lettere "ab" in un testo si potrà usare la regexpr:

```
/ab/
```

A questo punto ne sappiamo già abbastanza per chiarire un dubbio forse già affiorato: il carattere di spazio ha un significato particolare? La risposta è: **ASSOLUTAMENTE NO**! Quindi se vogliamo cercare le occorrenze della frase 'Corso di UNIX' in un documento, utilizzeremo la regexpr:

```
/Corso di UNIX/
```

Attenzione però a non pensare da subito che tutto sia lecito con le regexpr! Proprio perchè i caratteri sono quello che sono (e nulla di più), il carattere 'C' **non è** il carattere 'c'. Quindi la regular expression:

```
/corso di UNIX/
```

è diversa da quella precedente e le due non troveranno mai la stessa sequenza di parole.[2]

[2] Non è del tutto vero, ma per ora facciamo finta che sia così, altrimenti si rischia un potente mal di testa

Riassumendo:

- I caratteri sono semplici caratteri fino a che non si danno indicazioni differenti nella regexpr (vedremo in seguito ed in dettaglio come fare questo).
- Inoltre lo spazio è un carattere come tutti gli altri.
- Le regexpr sono *case sensitive* (ossia distinguono rigorosamente fra minuscolo e maiuscolo – *case* in inglese).

6.3 I Quantificatori

Le regexpr forniscono la possibilità di specificare quante volte può riscontrarsi il testo specificato preservando la validità della ricerca. Esistono tre quantificatori fondamentali:

Simbolo	Significato
?	*zero o una volta*
+	*una o più volte*
*	*zero o più volte*

Vediamo subito un esempio di applicazione. Supponiamo di voler cercare tutte le occorrenze di **stringhe** (sequenze) di caratteri costitutite da un numero indefinito di lettere "a" Senza i quantificatori avremmo dovuto scrivere una serie di regexpr come: `/a/`, `/aa/`, `/aaa/`, `/aaaa/`, e così via fino a chè la pazienza non ci avesse abbandonato, per poi eseguire tutti questi confronti sul testo in sequenza. Grazie ai quantificatori possiamo scrivere la ben più concisa ed elegante:

```
/a+/
```

che significa letteralmente: *una stringa di testo composta da una o più lettere a*.

Il lettore attento[3] avrà notato come ci sia una spiccata somiglianza fra i quantificatori delle regexpr e i caratteri di espansione (*file globbing*) delle shell.

Accanto a questi tre quantificatori, esiste un sintassi più flessibile per quantificare gli elementi di una regexpr, basata sulle *parentesi graffe*. La forma generale è la seguente: `{min,max}`, dove `min` è il minimo numero di volte che quell'elemento deve essere reperito,

[3] Tutti i testi seri di informatica hanno un lettore attento. Perchè noi dovremmo essere da meno? (Preghiamo pertanto il gentile pubblico perchè ci faccia pervenire segnalazioni circa l'avvenuta lettura di questa nota.)

mentre `max` è prevedibilmente il massimo numero. Volendo quindi reperire una stringa di almeno tre "a", ma non più di cinque, si può usare questa regexpr:

`/a{3,5}/`

la quale corrisponde alle stringhe "aaa", "aaaa" e "aaaaa". I due elementi non sono obbligatori,[4] il che consente di creare definizioni *aperte* di limiti. Se si volesse cercare una stringa di *almeno* tre lettere "a", sarebbe sufficiente la semplice:

`/a{3,}/`

Notate come il secondo termine è stato omesso, rendendo non vincolante il numero di caratteri oltre il terzo.

Detto questo, possiamo notare come i tre caratteri di quantificazione (?, + e *) siano in realtà delle forme abbreviate per comodità di casi particolari di questa sintassi:

Simbolo	Sintassi esplicita
?	0,1
+	1,
*	0,

È infine importante notare come i quantificatori delle regexpr danno la misura di quante volte possa riscontrarsi l'elemento che li precede. Da soli **non hanno alcun significato**! Quindi la regexpr:

`/*/`

(volta probabilmente a riscontrare stringhe di testo di zero o più caratteri qualsiasi) semplicemente non ha alcun significato, in quanto l'asterisco non quantifica nulla. Come si risolve correttamente questo problema?

[4] A seconda del programma che state usando, il primo può essere obbligatorio oppure no; se lo è, specificando un valore pari a 0 si ottiene lo stesso risultato che omettendolo

6.4 Un carattere solitario

Fra i caratteri con significato particolare ne esiste uno particolarmente utile: il punto (".").
Questo carattere rappresenta un qualsiasi carattere; è in un certo senso l'astrazione stessa del concetto di carattere. Consente di specificare una posizione libera da vincoli di qualità ma obbligata nella quantità. Se volessimo individuare tutte le stringhe di testo composte dalla lettera "a" e da un qualsiasi altro carattere potremmo impiegare la semplice:

```
/a./
```

Tuttavia la vera versatilità del "." si ha in accoppiamento con i quantificatori. Tornando al quesito del punto precedente, come è possibile indicare una sequenza di lunghezza indefinita di caratteri qualsiasi? Semplice:

```
/.*/
```

Tanto basta per risolvere il problema.[5]

6.5 Caratteri di Classe

Dopo avere buttato così tanto Lego per terra, è ora di trovare una scatola adeguata dove riporlo. Disponendo i mattoncini in buon ordine è più facile capire quanti ce ne sono per tipo e come è possibile usarli. Non siamo impazziti e non abbiamo deciso di convertire il corso in un salone di edilizia danese. Abbiamo solo cercato di trovare un'efficace metafora delle **classi di caratteri**.

Una classe di caratteri è un insieme di caratteri (speciali o normali) racchiusa fra parentesi quadre. Da un punto di vista posizionale, essa occupa lo stesso spazio di un carattere. Da un punto di vista qualitativo essa rappresenta una via di mezzo fra un carattere esplicito (es. /a/) e un punto (/./). Facciamo subito un esempio.

5 Complimenti! Avete appena letto la vostra prima regexpr completamente priva di lettere o numeri! Se avvertite un senso di nausea o vertigine, potrete utilizzare il sacchetto di cartone che trovate sotto le vostre poltrone. La RegExprAir vi augura un buon proseguimento.

Ammettiamo di voler trovare tutte le occorrenze delle stringhe di testo "ab", "ac" e "ad". Le classi di caratteri ci forniscono un modo per condensare tre regexpr in una e al contempo escludere tutti i caratteri indesiderati. Infatti:

`/a./`

troverebbe anche stringhe come "az", "a5" o perfino "a " (a-spazio). Invece la più precisa:

`/a[bcd]/`

consente di descrivere inequivocabilmente le tre stringhe cercate.
Vediamo qualche uso creativo delle classi di caratteri. Abbiamo visto all'inizio che `/Corso di UNIX/` e `/corso di UNIX/` sono due regexpr completamente distinte. Come è possibile unirle? Basta applicare una semplice classe al primo carattere, in questa maniera:

`/[Cc]orso di UNIX/`

6.6 Infrangiamo (apparentemente) un po' di regole

- **Alle classi di caratteri si possono applicare i quantificatori.** Questa è solo un'infrazione parziale. Abbiamo stabilito che un quantificatore si applica al carattere che lo precede. Un'interpretazione pedante e ottusa di quasta regola potrebbe dedurre che in:

  `/[abc]*/`

  l'asterisco si applichi solo al carattere "`]`". Le regexpr sono in realtà più lungimiranti e applicano il quantificatore all'intera classe.

  L'infrazione è infatti solo apparente in quanto una classe di caratteri non è altro che un insieme di caratteri dal quale estrarre *un solo carattere*. Vista in questi termini, la regola si applica ancora come quando l'abbiamo definita.[6]

[6] Anche se questo presto finirà.

La classe mantiene la sua natura per tutta la ricerca. Per intenderci: se viene trovata una prima occorrenza del carattere "a", questo non significa che di lì in avanti /[abc]*/ diventa equivalente a /a*/. Il carattere successivo può essere uno qualsiasi dei tre inclusi. L'ultima regexpr data reperisce quindi tutte le seguenti stringhe: "aaaa", "ababcb", "abcabcabc", "bcabcaacb", "aaaaaaaab" e così via.

Attenzione: L'ordine con il quale vengono inclusi i caratteri non è rilevante. Questo significa che /[abc]*/ e /[cba]*/ sono due modi totalmente equivalenti di scrivere la stessa classe di caratteri. Come corollario si ha che /[caso]*/ non individua solo "caso" ma anche "sacco" e "caos".[7]

- **Le classi di caratteri possono essere negate.** Esiste cioè la possibilità di costruire l'inverso di una classe di caratteri descrivendo quella classe e facendola precedere con l'accento circonflesso (\^). In pratica è come dire che una classe di caratteri include tutti i caratteri possibili *tranne* quelli espressamente specificati.[8]

  Ipotizziamo di voler cercare tutte le sequenze di caratteri che non contengano la lettera "a". Il modo più semplice ed efficace di scrivere questo è:

  /[^a]+/

  che letteralmente significa *tutte le sequenze di almeno un carattere (abbiamo usato un più) che non includano la lettera "a"*.

- **I caratteri non si digitano solo ad uno ad uno.** Le regexpr ci forniscono un metodo comodo per definire sezioni di caratteri secondo il comune ordinamento alfabetico (o più precisamente secondo la tabella ASCII). È sufficiente scrivere i due estremi dell'intervallo divisi da un meno (-). Per cercare stringhe di testo formate solo da lettere minuscole si potrà quindi sintetizzare:

  /[a-z]*/

[8] Un modo alternativo per vedere una classe normale è quello di pensarla come *nessun carattere tranne quelli esplicitamente specificati*, che però ha significato giusto come contraltare di una classe negata.

mentre per indicare stringhe di testo con qualsiasi lettera e il carattere di spazio, sarà sufficiente:

`/[A-Za-z ]*/`

o una sua equivalente fra `/[a-z A-Z]*/` o `/[ a-zA-Z]*/` , per dirne alcune.

Attenzione agli errori di battitura! `/[a- zA-Z]*/` (lo spazio per errore è capitato dopo il primo meno) definisce quella classe formata da:

– Tutti i caratteri compresi fra la a minuscola e lo spazio

– La zeta minuscola

– Tutti i caratteri compresi fra la A maiuscola e la Z maiuscola
che non è probabilmente quello che doveva essere lo scopo della regexpr.

6.7 Viviamo in un mondo avaro, baby!

In gergo si usa dire che le regexpr sono *greed* (*avare*). Con questo si vuole intendere che, nel cercare una corrispondenza, una regexpr includerà il maggior numero di caratteri che soddisfa quella espressione (e non il minore come potrebbe essere spontaneo pensare). Per questo bisogna saper dominare la fame di una regexpr per evitare che questa porti via tutto. Questo è comunque semplice grazie alle classi di caratteri. Ammettiamo di voler cercare tutte le stringhe di testo che si concludono fra virgolette:

`/"[^"]+"/`

letteralmente stiamo richiedendo tutte le stringhe di testo che iniziano con le virgolette, continuano con uno o più caratteri diversi dalle virgolette e si chiudono con le virgolette. Quindi saranno trovate: ”stringa”, ”prova”, ”corso di UNIX” e ”Corso di UNIX”; il tutto virgolette incluse! Ma non saranno stringhe valide: ””, (le virgolette non contengono alcun carattere), o ””” in quanto le virgolette NON possono contenere altre virgolette.

103 103

Anzi, cerchiamo di essere più onesti con noi stessi; ammettiamo di avere la stringa di testo:[9]

"$_1$"$_2$"$_3$

Come si comporta la regexpr? Facciamo una completa analisi della logica usata per tentare il match. Mano a mano che procediamo nella simulazione della logica della regexpr, "taglieremo" per così dire le parti di testo che hanno fallito definitivamente il match.

Il primo tentativo avviene sul primo carattere di virgolette in posizione 1. Il match su quel carattere è positivo in quanto il primo carattere della stringa *deve* essere un carattere di virgolette. Quindi si procede al carattere successivo. La seconda virgoletta non è però accettabile in quanto il carattere successivo può essere qualsiasi tranne la virgoletta stessa.

Il motore che esegue la ricerca decide che questa via è definitivamente infruttuosa e ne tenta un'altra. Scartato il primo carattere, prova ripartendo dal secondo; la stringa rimanente risulta così essere:

"$_2$"$_3$ *secondo tentativo*

Questo è prevedibilmente positivo sul primo carattere (ossia quello in posizione 2), ma nuovamente il secondo carattere (in posizione 3) non soddisfa il match per lo stesso motivo per il quale il carattere in posizione 2 non lo soddisfaceva al tentativo precedente. Nuovamente si decide che questa strada non può portare ad alcun match e si intraprende un terzo tentativo. Quest'ultimo risulta ancora più breve.

"$_3$ *terzo tentativo*

La terza virgoletta è conforme al match ma dopo di essa non segue più alcun carattere. Questo determina il fallimento definitivo anche di questo tentativo e per conseguenza dell'intera regexpr sulla stringa.

Ammettiamo ora di voler sperimentare come si comporti la regexpr dopo una breve plastica. Diciamo che il punto di domanda diventa un asterisco, con il seguente risultato:

```
/"[^"]*"/
```

Riesaminiamo il processo di ricerca a partire da zero. La stringa è sempre costituita da tre virgolette in sequenza:

"$_1$"$_2$"$_3$

9 Abbiamo numerato i caratteri per una più rapida individuazione

103 103

Tentando con il primo carattere anche questa regexpr leggermente differente ha successo. È già a partire dal secondo carattere che si ha una notevole differenza di comportamento. Il secondo carattere può essere tanto un carattere qualsiasi purché differente da una virgoletta, quanto una virgoletta. Attenzione: la ridondanza è solo apparente. C'è infatti sostanziale differenza fra incontrare una virgoletta e un altro carattere. Una virgoletta chiuderebbe il match, mentre un altro carattere no, consentendo di cercare un ulteriore carattere diverso da una virgoletta.

In sostanza la regexpr al secondo carattere di virgolette (in questo caso) è già soddisfatta. Questo comporta che la porzione di testo riscontrata viene ritornata come esito della ricerca e la riga viene abbandonata. Infatti una regexpr non tenta mai autonomamente un secondo match su una riga che ne ha già prodotto uno.[10]

Se il carattere in posizione 2 fosse stato un altro (es. una lettera c) il match non si sarebbe concluso lì, ma sarebbe proseguito sino al terzo carattere, risultando nella stringa `"c"`. Non pensate mai tuttavia che se una stringa contiene un match più ricco (contenente più caratteri) le regexpr lo preferiscano ad uno più povero. Non è questo lo scopo del gioco. *Prima si ottiene un match, meglio è!* Questo significa che, dalla stringa:

"$_1$ab"$_2$abc"$_3$ "$_4$abcdefghi"$_5$

la precedente regexpr avrebbe quattro possibili luoghi dove riscontrare un match:

- In posizione 1, con "ab"
- In posizione 2, con "abc"
- In posizione 3, con " "
- In posizione 4, con "abcdefghi"

di cui sicuramente più corposo l'ultimo. Tuttavia il match riportato sarà `"ab"`, in quanto primo ad essere raggiunto. Questo non deve confondere con quanto detto all'inizio circa l'avarizia delle regexpr. Esse sono si portate a raggranellare più caratteri possibile, ma appena concluso un match la riga di testo viene abbandonata, senza ulteriori appelli.[11]

10 A meno che questo non venga esplicitamente richiesto. Si veda a proposito il paragrafo "Opzioni ed altre meraviglie"

11 Diciamo che sono delle avare idiote, anche se molto utili.

6.8 Tanto di cappello *(e scarpe)* : ^ e $

È possibile porre un ulteriore determinante vincolo sulla stringa di testo. L'accento circonflesso (^) posto all'inizio della regexpr consente di obbligare il match a trovarsi in inizio di riga. Similmente il simbolo di dollaro ($) consente, se posto in coda, di vincolare la regexpr alla fine della riga. Se volessimo tovare tutte le righe di commento contenute in uno script[12] sarebbe sufficiente usare la semplice:

```
/^#/
```

Attenzione a non confondere questo uso del circonflesso con la negazione delle classi di caratteri. Qui siamo al di fuori di qualsiasi parentesi quadra, e solo come primo carattere della regexpr!

L'uso combinato di ^ e $ consente una notevole flessibilità nel costringere la regexpr ad applicarsi ad un'intera riga. Ad esempio la regexpr:

```
/^[^a]+$/
```

trova tutte le righe che non contengono mai la lettera a. L'inesatta

```
/[^a]+/
```

avrebbe invece miseramente fallito, riportando ad esempio da "aaaaaaaaaabc" un match su "bc", cosa invece impossibile con l'uso dei vincoli di inizio e fine riga.

Un interessante corollario di questa situazione è che la regexpr:

```
/^$/
```

trova solo le righe completamente vuote.[13] Meno prevedibilmente la regexpr:

```
//
```

fallirà ancora più miserrimamente, ottenendo un match istantaneo, in prima posizione, su qualsiasi riga di testo, che essa contenga zero o dieci o cento o 1024 o 4 Terabyte di caratteri.[14] Questo è dovuto al fatto che una simile regexpr cerca una stringa nulla (o vuota, se il termine è più immediato ed universale), ed un simile tipo di stringa si può ottenere come sottostringa da qualsiasi stringa.

13 Niente spazi, tabulazioni, underscore o altre diavolerie di questo secolo

14 L'informazione sulla riga da 4 Terabyte è solo supposta, non avendo mai avuto nessuno degli autori disco a sufficienza per tentare di realizzare una simile stringa

Se una stringa contiene zero caratteri, essa stessa sara' una stringa nulla. Ma se una stringa di carattere ne contiene uno, come conseguenza questa contiene anche *ben due* sottostringhe nulle: una prima ed una dopo del carattere. Non è forse immediato per il lettore alle prime armi, ma è così che ragionano le regexpr ed anche voi, fra pochi mesi!

6.9 Commenti indiscreti

Domanda: come si fa ad includere un $ in una regexpr senza intendere la fine della riga? Domanda: è possibile includere un "-" dentro una classe di caratteri senza che questo sia interpretato come un separatore di un intervallo di caratteri? Domanda: si può includere una parentesi quadra aperta "[" senza che questa venga interpretata come un inizio di classe di caratteri?

Si! Per raggiungere questo scopo è sufficiente commentare i caratteri con il carattere speciale di backslash (\). Quindi `\$`, `\-` e `\[` raggiungono lo scopo. Per ovvia conseguenza, per ottenere un backslash sarà sufficiente commentarlo con un backslash. Ad esempio, se volessimo una regexpr in grado di riconoscere sia filename UNIX che Windows, avremmo bisogno di cercare stringhe separate sia da slash (UNIX) che da backslash (Windows), con la regexpr:

```
/[\/\\]/
```

Lo slash necessita di commento in quanto sarebbe altrimenti interpretato come fine della regexpr /[/ che è inderogabilmente un *non senso*. In qualsiasi *contesto* esso richiede un backslash di commento, come in:

```
/\/usr\/local\/bin/
```

che cerca le occorrenze della stringa "/usr/local/bin". Se gli slash non fossero commentati, il secondo chiuderebbe la definizione della regexpr e dal terzo carattere in avanti (la "u" di "usr") sarebbe tutto considerato un enorme errore sintattico.

6.10 Meglio poter scegliere

Come già visto per i caratteri, è possibile definire classi alternative di stringhe di testo arbitrario da utilizzare come alternativa in un match. La sintassi è piuttosto semplice. Una *pipe* ("|") separa gli elementi. Ad esempio:

```
/[Cc]orso di UNIX/
```

avrebbe anche potuto scriversi:

```
/(Corso|corso) di UNIX/
```

o anche:

```
/(Corso di UNIX|corso di UNIX)/
```

o in ultimo:

```
/(C|c)orso di UNIX/
```

per quanto l'ultima risulti subito come una forzatura che si riduce ad un caso particolare delle classi di caratteri, delle quali perde la brevità ed è meno flessibile. Le parentesi tonde servono a dare il senso di quale parte della stringa includere in ciascuna alternativa. Senza parentesi tonde ad esempio la prima alternanza avrebbe dovuto interpretarsi come *scegli fra "Corso" e "corso di UNIX"*. Dove tuttavia l'assenza di parentesi tonde non comprometta l'univocità dell'espressione, queste si possono omettere. Esempio:

```
s/uno|due|tre/zero/
```

L'utilità delle parentesi tonde però non si limita al raggruppamento di scelte multiple. Ha anche la proprietà di segnare la posizione di un elemento all'interno del pattern della regexpr per poter poi, tramite il numero di riferimento di quella posizione, recuperare tutto il testo che la porzione di regexpr li inclusa ha corrisposto. La spiegazione di questo concetto risulta sempre più intricata della sua comprensione tramite l'applicazione pratica. Prima di darci ad essa però introduciamo ancora qualcosa per poterne poi fare l'uso migliore.

6.11 Sostituzione con le regexpr

Le regexpr non sono solo un potente sistema di ricerca del testo, ma anche (e forse per alcuni *soprattutto*) un potente sistema di sostituzione del testo. Mantenendo valido tutto quello sin qui esposte circa la ricerca del testo, vediamo ora piccole aggiunte alla teoria che consentono di arrivare alla sostituzione.

Anzi tutto, l'operatore che consente le sostituzioni è il seguente:

```
s/regexpr/testo in sostituzione/
```

dove la `s` sta per *substitution*.[15] `regexpr` è la definizione di regexpr usata per la ricerca di stringhe nel testo mentre `testo in sostituzione` è il testo da applicare al posto del testo che ha soddisfatto la regexpr.

Per consolidare quanto esposto sin qui, proviamo un po' di esercizi di ricerca e sostituzione. Attenzione solo ad un dettaglio importante: di seguito verranno usate le parentesi tonde senza commenti. Molti tool comuni come `vi` richiedono un commento per ogni parentesi tonda o carattere speciale (di controllo), quindi:

```
/(Sole|Luna)/
```

è in realtà corretta solo come:

```
/\(Sole\|Luna\)/
```

Non ci insultate: non l'abbiamo stabilito noi!

Eliminare i commenti di uno script 6.11.1

Cancellare le righe di commento è tanto semplice quanto usare:

```
s/^#.*//
```

la quale individua tutte le righe di commento e le sostituisce con una riga completamente vuota

[15] Abbiamo sin qui omesso un dettaglio formale circa la forma che esegue le ricerche; la scrittura `/regexpr/` è in realtà una forma concisa di `m /regexpr/` dove la `m` sta per *match*, ossia occorrenza, riscontro.

Sostituzione multipla 6.11.2

Volendo sostituire tutti i *Sistemi Aperitivi* per PC e Apple con un vero Sistema Operativo, sarà sufficiente:

```
s/Windows|MacOS/UNIX/
```

Sostituzione multipla con uso delle posizioni 6.11.3

Con una semplice:

```
s/(Windows|MacOS) e' meglio/\1 e' peggio/
```

tutte le stringhe “Windows è meglio” e “MacOS è meglio” saranno cambiate rispettivamente in “Windows è peggio” e “MacOS è peggio”. Questo esempio introduce l'uso delle posizioni delle parentesi tonde. La scelta multipla `Windows|MacOS` è posta fra parentesi. Questa coppia di parentesi è la prima (oltre che l'unica) nell'espressione. Ad essa si associa quindi la posizione `\1`. Utilizzando questo carattere speciale (uno commentato) si intende includere il testo che è risultato conforme ai criteri specificati dentro quella coppia di tonde. Nei due casi questo testo è “Windows” e “MacOS” rispettivamente.

Un'altro esempio può risultare vantaggioso:

```
s/Meglio (prima|presto) che (dopo|tardi)/Meglio \2 che \1/
```

Prevedibilmente tutte le stringhe “Meglio prima che dopo” sono cambiate in “Meglio dopo che prima”, mentre tutte le stringhe “Meglio presto che tardi” divengono “Meglio tardi che presto”.[16]

6.12 Opzioni e altre meraviglie

In coda ad una ricerca od una sostituzione si possono applicare opzioni che modifichino il funzionamento dell'expression in modo radicale. Ne tracciamo un profilo ridotto:[17]

16 Il solito lettore attento avrà però anche intuito che “Meglio prima che tardi” diviene “Meglio tardi che prima” mentre “Meglio presto che dopo” diviene “Meglio dopo che presto”; siamo tuttavia alla ricerca del motivo socioculturale che potrebbe spingere un essere raziocinante a esprimersi in maniera tanto abominevole

17 Esistono altre opzioni, ma la loro applicazione è troppo complessa e specialistica perché possa rientrare tra gli scopi del nostro corso

Opzione	Mnemonic	Comportamento
/i	**I**gnore case	No dà attenzione al fatto che le lettere siano maiuscole o minuscole
/s	**S**tart new line	Il "." può rappresentare anche il carattere CR/LF
/m	**M**ultiple lines	La regexpr considere più linee come una linea unica
/g	**G**lobal search	Riuscito il primo tentativo non abbandona la stringa, ma tenta nuovi match

Vediamo qualche applicazione. Torniamo alla precedente `/[Cc]orso di UNIX/`. Una scrittura sostituibile, per quanto **non di uguali esiti**, potrebbe essere:

```
/Corso di UNIX/i
```

L'applicazione di `/i` cerca tanto la stringa "Corso di UNIX" quanto "corso di UNIX". Il motivo per cui non è sostitutiva è dovuto al fatto che essa rappresenta anche versioni più esotiche come "CorSo dI UnIx". Decisamente non quello che volevamo!

Altra situazione:

```
s/abc/def/
```

applicata alla stringa "abc abc abc", la regexpr la trasforma in "def abc abc". Se volessimo eseguire la sostituzione anche sui possibili match successivi potremmo aggiungere l'opzione /g come in:

```
s/abc/def/g
```

la quale porterebbe alla stringa finale "def def def".

Sulle altre due opzioni non si danno esempi, risultando molto semplici e di rada applicazione.

7. Editor di testo

di Tx0

Per editing di testo si intende l'insieme delle procedure e dell'uso di programmi atto alla creazione, modifica ed elaborazione di documenti di testo, siano essi racconti, programmi, file di configurazione, email o quant'altro.

7.1 vi

Il più noto, diffuso ed universale editor di testo sotto UNIX è indubbiamente `vi`. Nel rispetto della tradizione UNIX `vi` **NON** è assolutamente l'unico editor di testo disponibile. Tuttavia su qualsiasi sistema UNIX è sicuramente possibile trovarne una versione installata e funzionante.

`vi` è un editor improntato allo schermo (il che significa che si può vedere contemporaneamente tutta la porzione del file che il vostro schermo è in grado di mostrare).[1] Sorprendentemente `vi` non ha alcun menù e non utilizza il mouse. Infatti `vi` è nato all'epoca dei terminali seriali, quando la grafica non esisteva, i mouse erano solo attaccati alle fotocopiatrici[2] e la velocità non era certo il punto forte degli utenti.

Ancor più sorprendentemente però `vi` ha mantenuto intatto il suo fascino e la sua versatilità. Gli autori stessi lo usano per qualsiasi tipo di editing, dalla creazione di file di configurazione, alla stesura di testi, alla creazione di interi siti web, alla compilazione di questo libro.

1 A differenza degli *editor orientati alla linea* che mostrano solo una linea di testo alla volta. Può sembrare assurdo ma esiste anche questo tipo di programmi ed è stato usato per lungo tempo

2 Questa NON è una battuta

Una personalità schizofrenica 7.1.1

vi lavora in due possibili modalità: *command mode* e *insert mode*.[3] L'insert mode è la modalità che qualsiasi utente di un editor di testo si aspetta; in questa modalità il testo viene inserito. Il command mode invece è la modalità nella quale si danno comandi a vi perché esegua modifiche al testo, cancelli porzioni di testo, esegua *taglia e incolla* sul testo e così via.

Eseguiamo una prima sessione di prova di vi:

```
$ vi /tmp/prova
```

lo schermo cambia e diventa qualcosa di simile[4] a questo:

```
~
~
~
~
~
~
/tmp/prova: new file: line 1
```

Le tilde (˜) indicano le linee vuote (nel nostro caso tutte in quanto abbiamo iniziato l'editing di un nuovo file). L'ultima riga, detta *status line*, è il luogo dove vi presenta informazioni all'utente. Non sempre è visibile e a volte viene invece utilizzata dall'utente stesso per impartire i comandi che iniziano con un prompt (:).

3 Da qui in avanti abbreviati *cm* e *im*

4 Esistono differenti versioni di vi; per questo usiamo forme come "qualcosa di simile"

Inserire del testo 7.1.2

`vi` è attualmente in *cm*. È insomma in attesa dell'inserimento di un comando. Il comando `i` passa dal *cm* al *im*. Dopo la pressione del solo tasto `i` è ora possibile digitare il testo. Quando si è terminato di inserire il testo si può uscire dal *im* premendo il tasto `[Esc]`.

```
Corso di UNIX
Loa HackLab MI
L.S.O.A. Deposito Bulk
~
~
~
```

Muoversi attraverso il testo 7.1.3

Subito dopo la pressione del tasto `[Esc]` `vi` torna in *cm*. Il cursore è posizionato sotto all'ultima lettera digitata. Proviamo a muoverci lungo il testo. Tutte le versioni più moderne di `vi` accettano le frecce come sistema di spostamento. Le versioni più vecchie invece usano solo i tasti `h j k l`.

Tasto	Direzione
`h`	Sinistra
`j`	Basso
`k`	Alto
`l`	Destra
`w`	Una parola a destra
`W`	Una parola (spazi e `Tab` inclusi) a destra
`e`	Alla fine di una parola a destra
`E`	Alla fine di una parola (spazi e `Tab` inclusi) a destra
`b`	Una parola a sinistra
`B`	Una parola (spazi e `Tab` inclusi) a sinistra
0 *(zero)*	Sposta il cursore all'inizio della linea
`$`	Sposta il cursore alla fine della linea

Tuttavia è difficile che troviate una versione di `vi` tanto vecchia. Attenzione ad una cosa: i comandi di spostamento (sia le frecce che le lettere corrispondenti) si possono usare tipicamente solo in *cm*.[5] Per spostarci quindi sulla "L" di "Loa" dovremo quindi premere 20 volte la "h" e una volta la "k". Oppure 20 $\leftarrow$ e 1 $\uparrow$.

[5] Solo i `vi` più recenti hanno introdotto un minimo di relax nel movimento consentendolo anche in *im*

Salvare ed uscire 7.1.4

vi è un editor particolarmente ricco di comandi. Sono tutti composti da una o due lettere, a volte preceduti da un numero che ne indica il raggio di azione. Sulle prime risultano sicuramente complessi e imperscrutabili, ma un po' di utilizzo li renderà molto familiari.

Per semplificare le cose[6] vediamo prima di tutti i comandi per salvare il file e lasciare l'editor. Questi comandi sono mostrati sulla status line e si attivano facendoli precedere dai due punti (:) in *cm*, e sono tutti terminati dalla pressione del tasto [Enter]:[7]

Comando	Funzionamento
:q	Esce dall'editor
:w	Salva il file
:wq	Salva il file ed esce dall'editor
:q!	Esce dall'editor senza salvare le modifiche

In *cm*[8] diamo la sequenza :q[Enter]

```
Corso di UNIX
Loa HackLab MI
L.S.O.A. Deposito Bulk
~
~
~
:q[Enter]
```

L'uscita dall'editor ci riporta al prompt della shell. Rientriamo: vi /tmp/prova.

Cancellare, copiare, modificare e sostituire 7.1.5

Tutti i comandi di vi seguono comunque questre tre semplici regole:

- I comandi sono differenti in maiuscolo o minuscolo (es I e i non sono la stessa cosa)
- I comandi dati non vengono mostrati a schermo, a meno che non siano comandi di prompt
- Nessun comando richiede la pressione di Enter, a meno che non siano comandi di prompt

6 ...ed evitare un forte senso di claustrofobia
7 Non scrivete i sette caratteri [E n t e r] !!
8 Se non siete sicuri di essere in *cm* premete un'altra volta [Esc]; al massimo l'editor vi risponderà con un *beep*

Cancellare, copiare, modificare e sostituire

Non tentate di imparare `vi` memorizzando tutti i comandi in una volta. La pratica e il collegamento mnemonico fra una lettera ed il comando associato faranno il resto. Nella tabella che segue sono segnati distintamente i comandi che inseriscono del testo nel *buffer temporaneo* da quelli che non lo alterano.[9]

Comando	In Buffer	Funzionamento
n`dd`	•	Cancella *n* o una riga
n`dw`	•	Cancella *n* o una parola
n`x`	•	Cancella *n* o un carattere a partire da quello sul quale è posto il cursore continuando verso destra
n`yy`	•	Copia *n* o una riga nel buffer
n`yw`	•	Copia *n* o una parola nel buffer
n`cw`		Modifica *n* o una parola
n`r`		Modifica *n* o un carattere con un carattere
n`s`		Modifica *n* o un carattere con una stringa arbitraria di caratteri
`p`		Inserisce il contenuto del buffer temporaneo
`J`		Unisce due linee

Vogliamo unire la seconda e la terza linea del file. Posizioniamoci in un qualsiasi punto della seconda linea e premiamo `J`. Il risultatò sarà il seguente:

```
Corso di UNIX
Loa HackLab MI L.S.O.A. Deposito Bulk
~
~
~
~
```

`vi` ha riportato l'intera terza linea sulla seconda, interponendo uno spazio fra l'ultimo carattere della seconda ed il primo della terza (se non già presente). Posizioniamoci ora sullo spazio fra "MI" e "L.S.O.A." e premiamo `i[Enter][Esc]`. Ossia:

`i`	→	Entra in *insert mode* e inserisci...
`[Enter]`	→	...un *a capo*...
`[Esc]`	→	...e torna in *command mode*

9 È normale che non sappiate cosa sia un buffer temporaneo, non sentitevi penalizzati e continuate a leggere

Posizioniamoci ora sulla parola `HackLab` per modificarla in `Hack Lab`. Quando siamo sulla “H” digitiamo `cw`. `vi` modifica l’ultimo carattere della parola in un dollaro ($) per indicarci fin dove la nostra modifica sostituirà il testo sottostante. La situazione è la seguente:

```
Corso di UNIX
Loa HackLa$ MI
L.S.O.A. Deposito Bulk
~
~
~
```

Iniziamo a digitare la modifica: “Hack Lab”. `vi` cambia le lettere fino a quella marcata dal dollaro, quindi inizia ad inserire i caratteri prima della parola “MI” fino a che l’utente non ha terminato l’inserimento.

```
Corso di UNIX
Loa Hack Lab MI
L.S.O.A. Deposito Bulk
~
~
~
```

Premiamo `[Esc]` per tornare in *cm*. Decidiamo di voler spostare la terza linea prima della seconda. Ci posizioniamo in un punto qualsiasi della terza linea e diamo il comando `dd`. La linea viene cancellata dallo schermo.

```
Corso di UNIX
Loa Hack Lab MI
~
~
~
```

La cancellazione di testo preserva una copia in uno speciale *buffer temporaneo* che può essere successivamente reinserito nel testo e non viene modificato fino alla successiva operazione di cancellazione o di copia. Ci posizioniamo in un punto qualsiasi della prima riga e premiamo il tasto `p`. La situazione sarà la seguente:

```
Corso di UNIX
L.S.O.A. Deposito Bulk
Loa Hack Lab MI
~
~
~
```

Rivediamo cosa è successo. La precedente operazione di cancellazione (`dd` sulla terza linea) ha cancellato l'intera linea, salvandone una copia nel buffer temporaneo. La pressione del tasto `p` sulla prima linea ha reimmesso nel testo il contenuto del buffer.[10]

Il comando `p` immette il buffer temporaneo a partire dalla posizione in cui si trova. Questo però deve intendersi nel senso che: se il buffer contiene una o più parole, queste verranno introdotte nella linea alla quale ci si trova a partire dal carattere sotto il cursore; se il buffer contiene una o più linee queste verranno immesse a partire dalla linea sulla quale ci si trova, proseguendo in giù. Ad esempio cancellando le due parole "Corso di" con un comando `2dw`, spostandosi sulla terza riga, primo carattere, e premendo `p`, l'effetto sarà il seguente:

```
UNIX
L.S.O.A. Deposito Bulk
LCorso di oa Hack Lab MI
~
~
~
```

Le due parole sono state inserite SULLA terza riga, a partire dal primo carattere e continuando verso destra.[11].

Ma insomma: cos'è questo buffer temporaneo? La parola *buffer* indica una zona di memoria nella quale vengono depositati dei dati per un successivo utilizzo. L'aggettivo temporaneo è dovuto al fatto che il contenuto di questo particolare buffer viene *automaticamente* modificato da tutte le operazioni di taglio, copia e cancellazione, durando quindi l'intervallo di tempo da una di queste operazioni alla successiva.

10 Il buffer temporaneo non si svuota dopo un `p`; è possibile immettere nel testo infinite volte il contenuto del buffer

11 L'effetto non è quello desiderato? Tra poco saprete come fare!

Anche gli Utenti UNIX possono sbagliare 7.1.6

Le ultime modifiche fatte non ci piacciono. Vogliamo tornare indietro. Abbiamo bisogno di un comando di *undo* che annulli gli ultimi cambiamenti al file. Avrete già indovinato che il comando in questione è `u`. Usiamolo una volta:

```
L.S.O.A. Deposito Bulk
Loa Hack Lab MI
~
~
~
```

Proviamo una seconda volta: cosa vi aspettereste di vedere?

Dirvi cosa vedrete non è semplice in quanto differenti versioni di `vi` si comportano qui in maniera discordante. Comunque un `vi` *storicamente attento* riscriverà "`LCorso di oa Hack Lab MI`" sulla terza linea. `vi` infatti ha un solo livello di undo. Questo comporta una necessaria attenzione ad ogni operazione di inserimento e cancellazione del testo, in quanto un eventuale ripensamento andrà manifestato subito dopo l'operazione.

Come fare allora per tornare alla situazione iniziale? Nessun problema. Posizioniamoci sul primo carattere della prima linea (sopra la "U" di "UNIX") e premiamo `P`. Non è un errore: *p maiuscola*. Al contrario di `p`, `P` inserisce il testo a partire dalla posizione corrente e procede *all'indietro*. Il buffer temporaneo contiene ancora "Corso di ". Le due parole sono posizionate prima della "U" e la prima linea torna ad essere "Corso di UNIX".

◻ Diversi modi per entrare in *insert mode* 7.1.7

`i` non è il solo modo che `vi` offre per passare dal *cm* al *im*. Vediamo l'insieme di possibilità:

Comando	Inizio[12]	Direzione[13]	Funzionamento
`i`	←	→	Entra in *im* dopo il carattere in cui si trova
`I`	⟵	→	Entra in *im* all'inizio della linea
`a`	→	→	Entra in *im* prima del carattere su cui si trova
`A`	⟶	→	Entra in *im* alla fine della riga
n`o`	↓	↓	Entra in *im* aggiungendo una linea dopo quella attuale
n`O`	↑	↓	Entra in *im* aggiungendo una linea prima di quella attuale
n`s`	→	→	Sostituisce il carattere su cui si trova
n`S`		→	Sostituisce l'intera linea sulla quale ci si trova
`R`	→	→	Sovrascrive i caratteri esistenti

Abbiamo già familiarizzato con il comando `i` quindi impariamo a capire come interpretare la tabella a partire da questo. `i` entra in *im* partendo dalla posizione immediatamente a sinistra di quella alla quale ci troviamo (←) e continua l'inserimento verso destra (→).[14]

`I` invece inizia l'inserimento all'inizio della linea (⟵) e prosegue da lì verso destra (→).

Ben diversi sono `o` ed `O`. Il primo inserisce una nuova linea subito dopo l'attuale (↓), posiziona il cursore sul primo carattere di questa linea e inizia l'inserimento verso destra. La direzione è indicata verso il basso (↓) per rimarcare il fatto che questo comando *inserisce un'intera nuova linea* e non solo nuovi caratteri sull'attuale.

`O` per contro inserisce una nuova linea *prima* dell'attuale (↑) e continua l'inserimento da lì verso destra e verso il basso (↓).

Tutti i comandi di inserimento preceduti da *n* consentono di stabilire a quanti caratteri o righe il comando faccia riferimento. Ad esempio: `s` sostituisce un carattere sul quale ci si trova con un numero arbitrario di caratteri. La sequenza `ssostituto[Esc]` modifica il carattere sul quale si trova il cursore con la stringa di testo "sostituto". Tuttavia la sequenza `4ssostituto[Esc]` modifica i 4 caratteri a partire da quello sotto il cursore proseguendo verso destra con la stringa "sostituto".

Ultimo comando, `R` entra in quello che più correttamente dovrebbe definirsi *replace mode*: funziona come `i` per quanto riguarda posizione e direzione di inserimento. Tuttavia il testo immesso *sovrascrive* quello preesistente anziché inserirsi prima di esso.

12 La posizione dalla quale parte l'inserimento, rispetto a quella attuale
13 Come prosegue l'inserimento dopo il primo carattere
14 Ovviamente non ci sarà mai un comando che inserisce il testo proseguendo verso sinistra, sarebbe contrario alla scrittura occidentale

Caratteri speciali e comandi di *scrolling* 7.1.8

Rivediamo tutti insieme i caratteri che hanno un particolare significato per `vi`.

Carattere	Significato
`.`	Ripete l'ultimo comando non di prompt eseguito
n`~`	Cambia maiuscolo/minuscolo per i successivi uno o *n* caratteri
`$`	Sposta il cursore alla fine della linea
`0` *(zero)*	Sposta il cursore all'inizio della linea
`^`	Sposta il cursore sul primo carattere non di spazio della linea
n `|`	Sposta il cursore al carattere *n* della linea

`vi` offre un nutrito numero di comandi per spostarsi lungo il file *(scrolling)*.

Comando	Scrolling
`Control-F`	Avanti di una schermata
`Control-B`	Indietro di una schermata
`Control-D`	Avanti di mezza schermata *(in alcuni anche PgDown)*
`Control-U`	Indietro di mezza schermata *(in alcuni anche PgUp)*
`z[Enter]`	Posiziona la linea corrente all'inizio dello schermo
`z.`	Posiziona la linea corrente nel mezzo dello schermo
`z-`	Posiziona la linea corrente alla fine dello schermo
`H`	Muove il cursore sulla prima linea
`L`	Muove il cursore sull'ultima linea
`M`	Muove il cursore sulla linea centrale
`:`*n*	Esempio: `:57[Enter]` sposta il cursore alla linea 57
n`G`	Sposta il cursore alla linea *n* *(Senza n sposta il cursore all'ultima linea)*

Non diamo esempi e spiegazioni di questi controlli e comandi ritenendoli sufficientemente semplici da essere compresi da subito; piuttosto più utile risulta una certa pratica.

Cut'n'paste, baby! 7.1.9

Abbiamo già visto come i comandi `dd`, `dw` e `x` cancellino del testo ponendolo nel buffer temporaneo dal quale è possibile recuperarlo con `p`. Allo stesso modo `yw` e `yy` copiano nel buffer del testo senza cancellarlo.

Tuttavia un solo buffer può essere troppo poco per un *Vero Utente UNIX*.[15] Ed infatti `vi` ci offre la possibilità di usare più di un buffer per memorizzare le nostre operazioni. Sono

[15] Era un complimento, coraggio!

disponibili 26 buffer definiti *named buffer*[16] in quanto associati alle 26 lettere dell'alfabeto. Il trucco per introdurre testo in uno di questi buffer sta nel precedere i comandi di cancellazione e copia con la coppia `"l` dove la lettera elle indica una qualsiasi lettera dell'alfabeto.[17]

Per memorizzare nel buffer *a* la prima linea del nostro file, posizioniamoci su di essa e digitiamo la sequenza `"ayy`. Semplice no?[18] Spostiamoci quindi sull'ultima linea (ad esempio con `G`), e diamo un semplice `"ap`. Risultato?

```
Corso di UNIX
L.S.O.A. Deposito Bulk
Loa Hack Lab MI
Corso di UNIX
~
~
~
```

Il contenuto dei buffer è completamente indipendente; per conseguenza qualsiasi operazione fatta sul buffer temporaneo[19] non tocca i named buffer e un semplice `dd` non cambia il contenuto del buffer `"a`.

Marcare la propria posizione 7.1.10

Al crescere del file, lo spostamento per linee o per schermate può comunque non bastare. Conviene allora utilizzare il meccanismo di *position marking* offerto da `vi`. Vediamo rapidamente i comandi di marking:

Comando	Funzionamento
`m`*x*	Marca la posizione attuale con la lettera *x*
`` `x ``	Muove il cursore al mark point associato alla lettera *x*
` `` `	Torna all'ultimo mark point
`'x`	Muove il cursore al primo carattere della linea che contiene il mark point *x*
`''`	Muove il cursore al primo carattere della linea che contiene l'ultimo mark point

Se ci troviamo in un punto qualsiasi del testo e premiamo `ma` verrà impostato un mark point su quella posizione chiamato "a". Spostiamoci altrove nel file; premiamo quindi `` `a ``: il cursore torna a posizionarsi sul carattere marcato in precedenza. Se quindi premiamo `'a` il cursore viene spostato all'inizio della linea che contiene il mark point "a".

16 ...per quanto un nome di una sola lettera possa non sembrare un nome!
17 Inclusa la lettera elle, certo!
18 Almeno, nel 1975 sembrava semplice...
19 A questo punto potremmo chiamarlo *il buffer anonimo*

Ricerche e sostituzioni con le regexpr 7.1.11

`vi` integra pieno supporto per le regular expression. Ricerche sul testo e sostituzioni vengono tutte eseguite attraverso pattern regexpr.[20]

Per eseguire una ricerca è sufficiente usare la sintassi da prompt:

```
Corso di UNIX
L.S.O.A. Deposito Bulk
Loa Hack Lab MI
~
~
~
/Loa[Enter]
```

Il cursore dopo la pressione del tasto `[Enter]` viene posizionato sul primo carattere del match alla prima occorrenza dello stesso. Per eseguire la stessa ricerca sarà sufficiente usare `n`. `N` al contrario inverte l'ordine di ricerca.

Per eseguire una ricerca dal basso verso l'alto, utilizzate `?` anziché `/`, come in:

```
Corso di UNIX
L.S.O.A. Deposito Bulk
Loa Hack Lab MI
~
~
~
?UNIX[Enter]
```

Le sostituzioni avvengono con la nota sintassi `s/pattern/testo in sostituzione/`. Prima della `s` è possibile introdurre dei delimitatori che indichino all'editor le linee sulle quali eseguire la sostituzione. Se i delimitatori sono omessi la sostituzione avviene solo sulla linea corrente. La sintassi dei delimitatori è `:`*inizio* , *fine* `s/pattern/testo/`. I delimitatori possibili sono:

Delimitatore	Significato
n	La linea *n*
.	La linea corrente
%	Tutto il file *(si usa da solo)*
$	L'ultima linea del file
+*n*	*n* linee dopo la linea corrente

[20] Per una conoscenza delle regular expression si rimanda al capitolo ad esse dedicato

Ricerche e sostituzioni con le regexpr

Diciamo di voler sostituire in tutto il file “Corso di” con “Incontri su”:

```
Corso di UNIX
Loa Hack Lab Milano
L.S.O.A. Deposito Bulk Milano
~
~
~
:%s/Corso di/Incontri su/g[Enter]
```

Notate come il carattere % sia da solo. In pratica % è una scorciatoia alla scrittura `1,$` , ossia dalla prima all'ultima linea del file. Se invece il cursore fosse alla linea 2 e volessimo modificare da lì alla fine del file “Milano” in “MI”, potremmo usare:

```
Incontri su UNIX
Loa Hack Lab Milano
L.S.O.A. Deposito Bulk Milano
~
~
~
:.,$s/Milano/MI/g[Enter]
```

ossia *modifica dalla linea attuale (.) alla linea finale (\$) la stringa “Milano” con la stringa “MI”*.

```
Incontri su UNIX
Loa Hack Lab MI
L.S.O.A. Deposito Bulk MI
~
~
~
```

Ricordiamo che del significato delle regexpr non si dà spiegazione rimandando al capitolo ad esse dedicato per una comprensione della sintassi e degli elementi. Solo scopo di questo paragrafo è mostrare l'uso delle regexpr all'interno di `vi`.

Vediamo alcuni trucchi per *Rendere Il Mondo Un Posto Migliore* usando `vi`. È possibile eseguire delle sostituzioni che tengano conto del contesto della riga. Ad esempio ammettiamo di voler eseguire la modifica della parola "MI" in "Milano" solo se la riga contiene la parola "Bulk". Esiste allo scopo una comodissima sintassi:

```
Corso di UNIX
Loa Hack Lab MI
L.S.O.A. Deposito Bulk MI
~
~
~
:/Bulk/s/MI/Milano/[Enter]
```

Il significato dell'espressione può essere parafrasato *cerca la parola "Bulk": se la ricerca è positiva modifica la parola "MI" con "Milano"*. Tutto chiaro?

Attenzione all'uso di stringhe non vincolate come pattern di ricerca! La regexpr `:%s/sistema/metodo/` ad esempio modificherà anche "sistematico" in "metodotico", che non è di certo quello che vogliamo. Utilizziamo allora i marcatori di inizio e fine di parola (`\<` e `\>`), come in `:\%s/\<sistema\>/metodo/`. Questa non cambia "sistematico" in "metodotico" perché dopo "sistema" non c'è uno spazio o un `Tab` ma un carattere ("t").

Se non siete certi dell'effetto che la sostituzione avrà sul testo e preferite controllare ogni match prima di cambiarlo, basta aggiungere l'opzione `c` alla fine della regexpr: `:%s/casa/cassa/gc`. In questo modo `vi` chiederà conferma, attendento `y[Enter]` per una risposta positiva o solo `[Enter]` per una negativa.

Personalizzare `vi` 7.1.12

`vi` consente un'alta personalizzazione dell'ambiente di lavoro attraverso opzioni, mappature e abbreviazioni. Tutta la configurazione di `vi` può essere cambiata dentro una sessione o attraverso il file di configurazione `~/.exrc`.[21] Inoltre `vi` legge dopo questo un altro eventuale `.exrc` che si trovi nella directory corrente. Questo consente di posizionare in `~/.exrc` i parametri di configurazione che volete sempre a vostra disposizione, mentre i parametri specifici ad un progetto si trovano nel `.exrc` collocato nella directory che ospita solo quel progetto.

Per impostare il valore di un'opzione si usa il comando `:set [no]opzione[=valore]`. Le opzioni si dividono fra quelle a due possibili valori (che possono essere `opzione` o

[21] Perché `~/.exrc`? Dovete sapere che dietro `vi` si cela un altro editor di testo chiamato `ex`. UNIX usa chiamare i file di configurazione `*rc` dove rc stà per *run command* in quanto il file contiene una serie di comandi da eseguire per configurare il programma. Quindi in questo caso è il `ex` *run command* file ossia .exrc — `vi` viene talvolta definito anche il `visual mode` di `ex` in quanto lavora come `ex` ma mostra una porzione del file alla volta

noopzione), e quelle che accettano parametri numerici nella forma opzione=valore (le quali non hanno un corrispettivo noopzione=... non avendo senso).

Vediamo un breve elenco delle opzioni più comuni.

Opzione	no	...=valore	Comportamento
autoindent	•		Allinea le nuove linee con le precedenti automaticamente
errorbells	•		Emette un *beep* in caso di errore
exrc	•		Abilita la lettura dei file .exrc in altre directory che non siano la home dell'utente
ignorecase	•		Ignora maiuscolo/minuscolo nelle ricerche e sostituzioni
list	•		Mostra ^I per i le tabulazioni e $ in fine di linea
magic	•		., [] e * hanno significato speciale nelle ricerche
mesg	•		Permette la comparsa dei messaggi di sistema durante l'editing di un file
number	•		Mostra il numero delle linee
readonly	•		Fà fallire i salvataggi a meno che non siano forzati con un punto esclamativo (es. :w!)
report		•	Imposta il numero minimo di linee che devono essere modificate da un comando perché il sistema avverta l'utente con un messaggio
scroll		•	Imposta la porzione di testo scrollata espressa in mezze schermate
showmatch	•		In *insert mode*, quando viene chiusa una parentesi tonda o graffa, vi sposta per un istante il cursore sulla corrispondente parentesi di apertura (molto comodo per i programmatori)
showmode	•		Mostra il *tipo* di *insert mode* fra *Insert, Append, Replace*
tabstop		•	Imposta il numero di caratteri da mostrare per ogni tabulazione
wrapscan	•		Quando viene toccato il fondo del file durante una ricerca/sostituzione riparte dall'inizio del file
wrapmargin		•	Imposta la dimensione in caratteri del margine destro. Quando questa dimensione viene superata vi inserisce automaticamente un *a capo* e inizia una nuova riga

Per la scrittura di testi è consigliabile una configurazione come la seguente:

```
:set tabstop=8
:set noautoindent
:set ignorecase
:set nomesg
:set report=1
:set noshowmatch
:set showmode
:set wrapscan
:set noexrc
:set wrapmargin=5
```

mentre per la programmazione o la scrittura di file di configurazione può essere più confortevole:

```
:set tabstop=4
:set autoindent
:set noignorecase
:set nomesg
:set report=1
:set showmatch
:set showmode
:set wrapscan
:set exrc
:set wrapmargin=0
```

Notate in particolare la differenza del `wrapmargin` a zero (che disabilita l'inserimento dell'a capo, cosa indesiderabile in programmazione), la riduzione delle tabulazioni (`tabstop=4`) e l'autoidentazione delle linee (`autoindent`). Utile anche `showmatch` e consentita la lettura di altri .exrc con `exrc`. A totale discrezione dell'utente e dei suoi colleghi di lavoro o familiari la scelta `errorbells/noerrorbells`.

Un'altro modo in cui `vi` ci viene incontro è nel risparmiarci di digitare lunghe frasi ricorrenti con il comando `:ab abbreviazione frase estesa`. Può essere annullato con `:unab abbreviazione`. In pratica quando in *im* digitiamo per intero l'abbreviazione `vi` si preoccupa di sostituirla con il testo esteso corrispondente. Facciamo un esempio: `:ab loa Loa HackLab MI`. D'ora in avanti ogni volta che scriveremo "loa" `vi` introdurrà nel testo "Loa HackLab MI". Per annullare questa abbreviazione basta dare `:unab loa`.

Analogo ma più esteso è il comando `:map[!] x sequenza`. Collega la pressione del tasto `x` con la sequenza di comandi `sequenza`. Il punto esclamativo opzionale di seguito al `:map` assegna la mappatura al *insert mode* anziché al *command mode*. Ad esempio vogliamo mappare al tasto q il salvataggio e l'uscita dal file: `:map q :wq^M`. La notazione `^M` indica il carattere di a capo.[22] Commentiamo il comando.

`:map`	→	Attiva un mapping in command mode...
`q`	→	...sul tasto q...
`:wq`	→	...corrispondente al salvataggio e all'uscita dall'editor...
`^M`	→	...*ed eseguilo!!!*

Inserire un `[Enter]` *(^M)* nella sequenza è necessario in quanto `vi` esegue le mappature *fedelmente* senza nulla aggiungere! In questo caso senza l'`[Enter]` avrebbe attivato la command line, ci avrebbe scritto dentro `:wq` ed avrebbe atteso la pressione dell'`[Enter]` da parte dell'utente. Può sembrare eccessivamente cervellotico come meccanismo ma consente in realtà giochetti molto divertenti. Ad esempio la mappatura `:map q G3k4dd` cosa fà?

`G`	→	Raggiunge la fine del file
`3k`	→	Risale di 3 righe
`4dd`	→	Cancella 4 righe

ossia cancella le ultime 4 righe del file in qualsiasi punto del file ci si trovi in quel momento. Tuttavia il cursore resta in fondo al file; meglio allora: `:map q mzG3k4dd‘z`.

`mz`	→	Imposta il mark point `z` sul carattere corrente
`G`	→	Raggiunge la fine del file
`3k`	→	Risale di 3 righe
`4dd`	→	Cancella 4 righe
`‘x`	→	Ritorna al mark point `z`

Diciamo infine che vogliamo eseguire una sostituzione su tutto il file e quindi salvarlo ed uscire dall'editor. Utilizziamo ad esempio: `:map q :%s/loa/Loa HackLab MI/g^M:wq^M`

`:%s/loa/Loa HackLab MI/g^M`	→	Esegue la sostituzione
`:wq^M`	→	Salva ed esce

[22] **Non è composta dai due caratteri ^ e M** bensì si ottiene con la pressione di `Control-V Control-M` in sequenza. `Control-V` è uno speciale modo di inserimento che *commenta* il carattere successivo a `vi`. Se provate a premere `Control-M` vi accorgerte che ottiene lo stesso effetto della pressione del tasto [Enter]. Questo perché **È** il tasto `[Enter]`. Usando `Control-V` prima, `Control-M` viene inserito come carattere nel testo e non interpretato come un `[Enter]` destinato a `vi`

A questo punto dovrebbe essere chiaro come l'inserimento esplicito del tasto `[Enter]` sia un vantaggio che ci permette di accorpare più comandi distinti in un unico mapping!

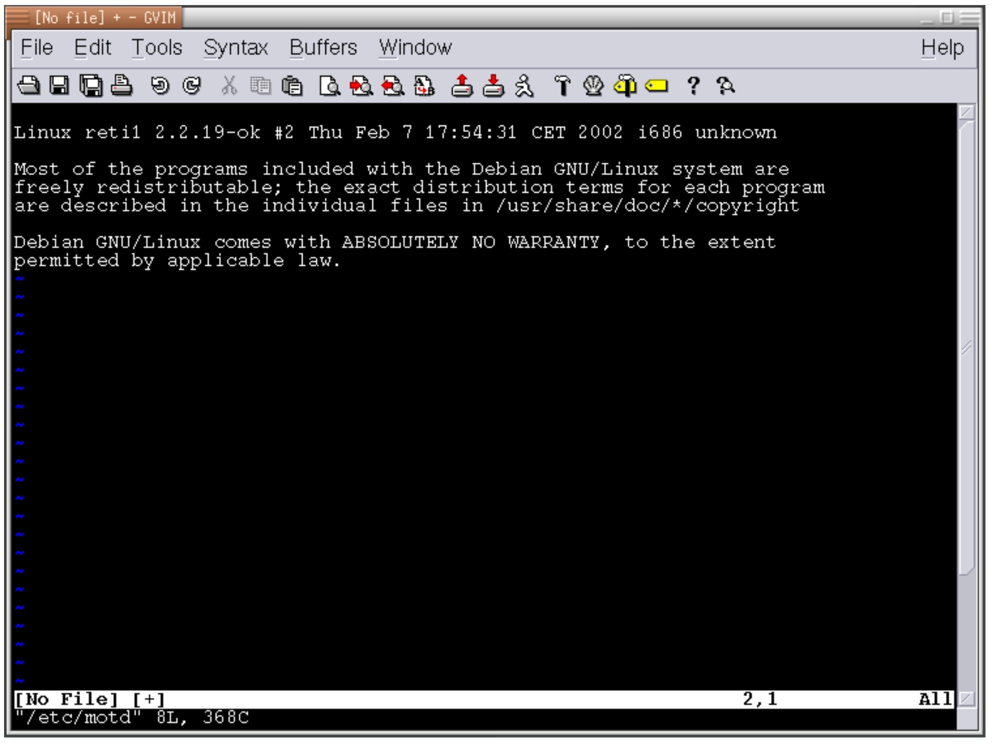


Figure 7.1 Una sessione di gvim

`vi`-derivati 7.1.13

Ormai `vi` è in realtà una famiglia di editor di testo più che un singolo programma. Dall'originale sono derivati molti editor di testo che garantiscono compatibilità con il programma originale e aggiungono funzioni nuove.

Per citare alcuni esempi:

- `nvi`
  Editor inteso a sostituire la versione BSD di `vi` restando compatibile fino nei bug.

- `elvis`
  Include un editor esadecimale e syntax highlighting per alcuni linguaggi.

- `vim`
  Acronimo di **V**i **IM**proved, aggiunge un numero enorme di opzioni in più, consente la gestione di più file su finestre distinte, è fornito di un linguaggio di definizione della

sintassi dei linguaggi che consente di scrivere nuovi modelli per il sintax highlighting dei linguaggi, comprende anche un'interfaccia grafica. A parere degli autori è il più comodo e flessibile dei tre. Infatti questo libro è stato scritto esclusivamente usando `vim`.

7.2 sed

`sed` è l'acronimo di *stream editor*. Infatti `sed` lavora tanto su file quanto sull'output di pipe line di comandi con lo stesso criterio: *il testo è uno stream (flusso) di dati*. Le parti in gioco sono fondamentalmente due: *il testo* e un *sed script*, ossia un elenco di comandi che `sed` deve eseguire sullo stream fornito.

Lavorando secondo il principio dell'applicazione di una serie di comandi predefiniti su uno stream di testo, `sed` si può ascrivere di buon conto alla categoria degli editor non interattivi. Per quanto possa in un primo momento sembrare poco attrattivo elaborare del testo *alla cieca*, rapidamente noterete come `sed` sia un potentissimo tool in grado di eseguire 27 differenti cambiamenti ed elaborazioni su 96 file in un minuto neanche. Niente male! Pensate questo modo di lavorare come un analogo del *multifile global search and replace* che molti editor offrono anche in ambiente Windows, con la differenza che:

- è piccolo *(meno di 50 kilobyte)*
- consente non solo di operare ricerche e sostituzioni su più file contemporaneamente ma anche di effettuare modifiche come cancellare 23 righe, salvare 400 righe in una altro file e così via
- può essere affiancato a strutture logiche della shell dalla quale viene eseguito consentendo così una comoda collocazione, lo spostamento o archiviazione e compressione (per dirne alcune) dei file una volta elaborati

Abbiamo visto nella precedente sezione su `vi` come esista una differenza fra editor *visuali* (ossia orientati all'elaborazione interattiva con possibilità di vedere tutto il testo a video) ed editor *di linea* (orientati all'elaborazione del testo una linea alla volta, senza visualizzazione diretta del testo).

`sed` discende direttamente da `ed`, l'editor di testo *orientato alla linea* standard sui sistemi UNIX. `ed` lavora seguendo questa logica:

- ogni comando si riferisce solo alla *linea corrente*, se non altrimenti specificato
- l'elaborazione è effettuata in due diverse modalità: *insert mode* (per l'inserimento del testo) e *command mode* (per l'esecuzione di comandi).

- i comandi di elaborazione comprendono le regexpr e un set di parole chiave per eseguire la cancellazione, copia e riposizionamento delle linee
- `ed` consente tanto l'editing interattivo quanto l'editing automatizzato attraverso script
- qualsiasi modifica viene automaticamente scritta sul file che si stà elaborando

Gli ultimi due punti sono la ragione principale della creazione di `sed`: avere a disposizione un programma più facilmente impiegabile come filtro che consenta successivi tentativi di elaborazione. `sed` infatti **non** salva il risultato dell'elaborazione come nuova versione del file, ma lo stampa a video (più precisamente lo invia a `stdout`).

Primo approccio con `sed` 7.2.1

Vediamo la sintassi di `sed`.

```
sed [opzioni] script filename
```

oppure

```
sed -f scriptfile filename
```

oppure

```
sed -e 'istruzione1' [ -e 'istruzione2' ] filename
```

Nella prima versione `sed` riceve uno script come primo argomento[23] e il nome di un file da editare come secondo argomento. Nella seconda forma ottiene il nome di un file che contiene uno script come valore all'opzione `-f` e il nome del file da editare come seconda opzione. Ancora, `sed` viene impiegato all'interno di uno *shell wrapper*, ossia di uno script di shell che contiene tanto la riga di comando che esegue `sed` quanto lo script che `sed` esegue sul file.

Vediamo subito qualche applicazione. Diciamo che abbiamo il file `languages` con il seguente contenuto:

[23] Attenzione: per uno script non si intende il nome di un file che contiene lo script, ma lo script stesso! Di più in seguito

```
Perl
Tcl
FORTRAN
```

Applichiamo qualche primo semplice comando con sed per vedere come questo cambi il contenuto del file (sempre senza nulla scrivere nel file originale, ricordate).

```
$ sed -e 's/Perl/Practical Extraction and Report Language/'\
> languages
C
Practical Extraction and Report Language
Tcl
FORTRAN
$
```

sed esegue la regexpr specificata nell'opzione -e e stampa il contenuto del file modificato. Notate come alla seconda linea "Perl" sia stato modificato in "Practical Extraction and Report Language". È possibile dare più comandi di editing per linea usando più di una opzione -e:

```
$ sed -e 's/Tcl/Tool Command Language/'\
> -e 's/FORTRAN/Formula Translator/' languages
C
Perl
Tool Command Language
Formula Translator
$
```

In questo caso sono state eseguite due modifiche con una sola esecuzione di sed. Un'ultima possibilità è quella di utilizzare l'inserimento multilinea di sh e shell derivate (bash, zsh, ksh, pdksh):

```
$ sed '
> s/C/C is C/
> s/Perl/Practical Extraction and Report Language/
> s/Tcl/Tool Command Language/
> s/FORTRAN/Formula Translator/
> ' languages
C is C
Practical Extraction and Report Language
Tool Command Language
Formula Translator
$
```

Dopo aver scritto la prima linea (`sed '[Enter]`) la shell sa di essere dentro al contesto definito dagli apostrofi quindi propone un prompt differente (può variare a seconda delle impostazioni della vostra configurazione) consentendo di inserire nuove linee come se fossero dentro gli apostrofi. Continuiamo ad inserire i comandi uno alla volta e solo dopo l'ultimo comando chiudiamo gli apostrofi e aggiungiamo il nome del file da editare.

Un primo script 7.2.2

Notate come nell'esempio precedente abbiamo usato la prima sintassi proposta per `sed` ossia abbiamo specificato lo script come primo argomento del comando e il file come secondo;[24] torna tutto?

Perché è importante rilevare che abbiamo usato la prima sintassi? Perché questo implica che quello che abbiamo composto è a tutti gli effetti uno script per `sed`. A prima vista sembra forse un'osservazione ridondante, ma ha in realtà qualcosa da osservare. Uno script `sed` è composto da più comandi disposti uno per linea e separati (quindi) da un carattere di *a capo*. Utilizzare la tecnica dell'inline code o scrivere il codice dentro un file e quindi passarlo a `sed` con l'opzione `-f` è sintatticamente la stessa identica cosa.

La vera differenza fra le due tecniche stà nel fatto che il codice inline deve essere inserito ogni volta a mano, il che non è molto comodo se si sta sperimentando uno script procedendo per successivi perfezionamenti; mentre la scrittura dello script dentro un file consente invece di aggiungere, modificare o riumuovere i comandi ad uno ad uno senza dovere reintrodurre ogni volta l'intero script. Inoltre la filosofia dei perfezionamenti successivi ben si adatta a `sed`[25] in quanto non modifica il file sorgente ma mostra il risultato a video.

24 Questo tipo di definizione dello script viene chiamato *inline code* o anche *here document*; le due definizioni indicano che lo script (che nulla vieta di vedere come un *documento*) viene inserito *inline* ossia *nella linea* di comando – altrimenti pensabile come un documento posto *here* (*qui*) nella linea

25 E non a `ed`

E quindi? Come si fà a salvare i risultati di una sessione di sed se vengono solo stampati a video? Risposta: si usa la redirezione della shell!

```
$ sed -e 's/war/love/' infile > outfile
$
```

L'output di sed è stato salvato in outfile. **Non tentate MAI di fare una cosa simile:**

```
$ sed -e 's/programmed/suicide/' myfile > myfile
```

Se stavate cercando di salvare direttamente le modifiche effettuate da sed nel file, sappiate che avete fallito miseramente! Questo per il semplice motivo che l'operatore > *prima ancora che la shell chiami il comando* azzera il file al quale l'output andrà rediretto. Quindi:

- `myfile` viene azzerato
- `sed` viene chiamato ad operare su un file *completamente vuoto*
- risultato: `myfile` è grande zero caratteri!

Come viene applicato uno script? 7.2.3

È il momento di ragionare su come venga applicato uno script ad un file. Concetto fondamentale è che per ogni riga del file viene applicato l'intero script. Questo significa che è importante porre attenzione all'ordine con il quale vengono immesse le istruzioni nello script. Prendiamo in esame il file:

```
$ cat myfile
Sistema Operativo
SO
$
$ sed '
> s/Sistema Operativo/SO/
> s/SO/Sistema Operativo/
> ' myfile
Sistema Operativo
Sistema Operativo
$
```

Esaminiamo riga per riga[26] cosa sia successo. Alla prima riga abbiamo "Sistema Operativo". `sed` applica prima l'istruzione `s/Sistema Operativo/SO/` ottenendo un match positivo e modificandola quindi in "SO". Quindi applica alla stessa riga l'istruzione `s/SO/Sistema Operativo/` *ottenendo nuovamente un match positivo*[27] e modificandola quindi in "Sistema Operativo".

Lo script è terminato, quindi si può procedere alla riga successiva. Questa contiene "SO". `sed` applica la prima istruzione e fallisce il match. Quindi applica la seconda istruzione trovando un match e modificando la riga in "Sistema Operativo".

Potreste erroneamente aver pensato che l'output definitivo sarebbe stato:

```
SO
Sistema Operativo
```

credendo che `sed` tralasci le successive istruzioni appena una ha avuto effetto positivo su una riga di testo. Non è questa la logica con la quale opera `sed`. *A ciascuna riga di input viene applicato l'intero script.*

L'uso dei delimitatori fatto in `vi` ha qui un significato leggermente diverso. Qualsiasi regexpr che non sia vincolata da limitatori si applica tacitamente a tutto il file.[28] Questo non toglie la possibilità di utilizzare limitatori con `sed`. Essi fanno riferimento al file esattamente come in `vi`. Inoltre vedremo in questo contesto alcune possibilità di delimitare il campo d'azione di una regexpr validi anche in `vi` ma presentati solo qui in quanto ritenuti più utili in un editor orientato allo streaming.

26 ...del file di testo

27 La riga è stata modificata dall'istruzione prima proprio in "SO"

28 Volendo fare un parallelo per comprendere la differenza potremmo dire che è come se con `vi` *si utilizzasse* prima di ogni regexpr un `%`

Comandi 7.2.4

`sed` definisce un set di comandi per l'editing sul testo analoghi ma leggermente differenti rispetto a quelli di `vi`. Vediamoli riassunti:

Comando	Comportamento
`[`*addr1*`]a` *testo*	Appende *testo* dopo il carattere corrente o dopo *addr*
`[`*addr1*`]a` *testo*	Appende *testo* dopo il carattere corrente o dopo *addr*
`[`*addr1*`,[`*addr2*`]]c` *testo*	Modifica il *pattern space* (o le linee dell'intervallo) con *testo*
`[`*addr1*`,[`*addr2*`]]d`	Cancella il *pattern space* (o le linee nell'intervallo)
`[`*addr1*`,[`*addr2*`]]g`	Copia il contenuto dell'*hold space* (un equivalente del buffer temporaneo di `vi`) nel *pattern space*
`[`*addr1*`,[`*addr2*`]]h`	Copia il *patternspace* nell'*hold space*
`[`*addr1*`]i` *testo*	Inserisce *testo* dopo il carattere corrente o dopo *addr*
`[`*addr1*`,[`*addr2*`]]p`	Stampa il *pattern space* (o le linee nell'intervallo)
`[`*addr1*`,[`*addr2*`]]w` *file*	Accoda il contenuto del *pattern space* (o dell'intervallo specificato) dentro `file`
`[`*addr1*`,[`*addr2*`]]x`	Scambia il contenuto del *pattern space* con il contenuto dell'*hold space*
`[`*addr1*`,[`*addr2*`]]y/`*abc*`/`*xyz*`/`	Scambia i caratteri *abc* con il corrispettivo per posizione fra *xyz*

continua. . .

Comando	Comportamento
[*addr1*,[*addr2*]]s/*regexpr*/*sostituzione*/[*flags*]	Sostituisce nel *pattern space* (o sulle linee specificate nell'intervallo) le occorrenze di *regexpr* con *sostituzione*. Il comportamento dell'istruzione può essere modificato usando uno o più flags fra:

n	Sostituisce l'*n*esima corrispondenza anziché la prima
g	Sostituisce tutte le occorrenze nel *pattern space*
p	Mostra le linee modificate (se il *pattern space* offre più di un match verrà mostrato una volta per ogni match
w *file*	Scrive il *pattern space* dentro *file* se è stato modificato

Delimitatori in `sed` 7.2.5

In `sed` restano validi come delimitatori:

- i numeri delle linee
- `$`
- `.`
- `/regexpr/`

L'ultimo tipo di delimitatore consente di porre un vincolo sulle linee che offrono un match alla regexpr specificata. Ad esempio usando una espressione come `/Nel mezzo/,/era smarrita/` identifica i primi tre versi della Divina Commedia. Il delimitatore `%` non ha senso in quanto ciascuna pattern viene già applicato a qualsiasi riga di testo contenuta nel file.

7.3 `awk`

awk[29] è un vero e proprio linguaggio di programmazione. Lo scopo del linguaggio è quello di analizzare e processare file di testo e di sperimentare gli algoritmi durante la fase di prototipizzazione.

Noi non saremo particolarmente rigorosi ed esaustivi nella presentazione del linguaggio, dato che i tre inventori hanno scritto addirittura un intero libro.[30] È quindi evidente che è al di sopra del nostro testo spingersi così avanti. Ci limiteremo alla sintassi elementare e ai suoi usi più immediati, giusto per stimolarvi l'acquolina.

La sintassi di awk può essere una delle due seguenti:

```
awk  [-W  option] [-F value] [-v var=value] [--]
   'program text' [file ...]

awk [-W option] [-F value] [-v var=value]
   [-f program-file] [--] [file ...]
```

Le opzioni sono:

Opzione	Significato
-F sep	Imposta il separatore di campo a *sep*
-f file	Il testo del programma è letto dal file *file* . Sono ammesse più opzioni -f
-v var=valore	Assegna *valore* alla variabile *var*
–	Indica la fine delle opzioni

Un esempio elementare: recuperare la shell dal passwd 7.3.1

Diciamo che vogliamo sapere quale shell utilizzi un utente del sistema. I passaggi logici da fare sono:

- Trovare l'utente nel file /etc/passwd (grep)
- Estrarre il settimo campo (shell) e stamparlo a video

Abbiamo così modo di sperimentarci con le pipe che abbiamo già visto in precedenza. Cominciamo a recuperare il nostro utente:

```
$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
$
```

29 Il nome deriva da Aho, Kernighan e Weinberger, i tre creatori del linguaggio

30 Aho, Kernighan e Weinberger, The AWK Programming Language, Addison-Wesley Publishing, 1988

Bene. Ora estraiamo il settimo campo. Dobbiamo prima di tutto specificare ad `awk` quale separatore di campo usare. Useremo "`-f :`". Quindi il corpo del programma. Anticipando quanto vedremo nella sezione successiva, diciamo che il "programma" da utilizzare è `print $7`.

Quindi:

```
$ grep root /etc/passwd | awk -F : '{ print $7}'
/bin/bash
$
```

Et voilá. La shell di root è `/bin/bash`. La sintassi `$n` indica l'*n* -esimo campo delimitato dal separatore specificato con `-f`. Notate che abbiamo dovuto includere il corpo del programma tra apostrofi per evitare che la shell sottostante tentasse una espansione di `$7` pensando che fosse una sua variabile d'ambiente.

La numerazione dei campi parte da 1 e prosegue.

Descrizione del linguaggio 7.3.2

Un programma `awk` è un elenco di statement *pattern* {*action*} . Un pattern può essere:

- `BEGIN`
- `END`
- *expression*
- *expression* , *expression*

Il *pattern* O l'*action* possono essere omessi. Se il pattern viene omesso è come se fosse risultato positivo al match. Se ad essere omessa è l'action si dà per assunta `{print}`.

Elenchiamo rapidamente i costrutti:

```
if ( expr ) statement

if ( expr ) statement else statement

while ( expr ) statement

do statement while ( expr )

for ( opt_expr ; opt_expr ; opt_expr ) statement

for ( var in array ) statement

continue

break
```

Il significato è in tutto e per tutto analogo a quello già affrontato con le shell. Per questo non spiegeremo da capo ciascuno di essi. È solo opportuno prestare attenzioni alle differenze che possiamo rilevare. Ad esempio con `awk` il costrutto `if` non viene terminato da `fi`. Anzi non viene terminato proprio.

Come è prevedibile aspettarsi, `awk` supporta le regular expression. È sufficiente specificare la regexpr con la sintassi abituale: `/regexpr/` . Tramite il costrutto `expr ~ /regexpr/` è possibile applicare una regular expression sul `expr`. Il costrutto risulta `1` se la regexpr ha un match positivo.

In fila per uno: gli array 7.3.3

`awk` offre supporto per array monodimensionali. La scrittura `array["indice"]` indica l'elemento contenuto nell'array `array` alla posizione `"indice"`. Notate come l'organizzazione dei dati non sia posizionale, bensì associativa. Ossia posso usare tanto `"3"` quanto `"mercoledi"` per descrivere una posizione in un array.

Il costrutto `expr in array` viene ritenuto vero se esiste `array["expr"]`.

Esiste una forma di `for` che itera sugli elementi di un array:

```
for ( var in array ) statement
```

Dentro statement, var conterrà un elemento diverso di array per ciascuna iterazione.

Funzioni 7.3.4

awk offre un certo numero di funzioni sue interne e provvede all'utente la possibilità di crearne di proprie.

Vediamo prima alcune delle funzioni *builtin* di awk:

Funzione	Scopo
index(string, substring)	Se *substring* è contenuta in *string* , index ritorna l'indice di carattere al quale è contenuta.
length(string)	Ritorna la lunghezza di *string*
tolower(string) toupper(string)	Cambia capitalizzazione alle lettere di *string* , scrivendole minuscole (*tolower*) o maiuscole (*toupper*)
split(string,A,regexp)	La stringa *string* viene divisa dalla regular expression *regexpr* e gli spezzoni creati vengono caricati nell'array *A* .
sub(regexp,sub,var) gsub(regexp,sub,var)	Applica la regular expression *regexp* e sostituisce il primo match con *sub* all'interno della variabile *var* . Se la chiamata è a gsub, awk non si ferma al primo match ma continua fino alla fine (equivalente all'opzione /g che abbiamo visto nel capitolo delle regexpr).

awk presenta anche altre builtin function che potete trovare documentate nella man page del comando. Invece, la sintassi per dichiarare una subroutine nuova è:

```
function name( args ) { statements }
```

Il corpo della funzione può contenere uno o più return statement nella forma:

```
return opt_expr
```

dove opt_expr è una espressione opzionale da ritornare.

Le chiamate alle funzioni possono essere ricorsive e nidificate. Gli argomenti vengono passati per valore (*expression*) o per reference (*array*).

Facciamo un esempio. La funzione printdown stampa una parola in verticale:

```
function printdown(n, i)
{
  n = length($0)
  for( i = 1 ; i <= n ; i++ ) print substr($0, i, 1)
}

{ printdown(n, i) }
```

Proviamo ad eseguire il codice:

```
$ awk -f prog
UNIX base
U
N
I
X

b
a
s
e
$
```

Per terminare l'esecuzione del programma abbiamo premuto `Control-d`. `$0` è la riga corrente. Dopo aver lanciato il programma, abbiamo scritto `UNIX base` a mano e premuto `[INVIO]`. Appena `awk` ha ricevuto una nuova linea di input ha subito iniziato il processing ed ha applicato "il programma" (ossia la linea `{printdown(n, i)}`) su quell'input.

8. Il mondo là fuori

di Little-John

Un computer isolato non ha molto senso, e le informazioni, la conoscenza, devono essere scambiate, condivise con tutti, liberamente. Come? Semplice: telnet e ssh per usare i sistemi remoti, ncftp e gftp per scambiare file via ftp, e poi l'email e internet. SHARE, SHAAARE, SHAAAAAAAAAAAAAAARE.

8.1 Collegarsi ad un sistema remoto con telnet e ssh

telnet 8.1.1

Il comando `telnet` normalmente viene utilizzato per collegarsi ad un host remoto che offra questo servizio attraverso un apposito demone e presso cui abbiamo un account. La sua utilizzazione classica è:

```
% telnet -l nomeutente host
```

La porta di default del demone di `telnet` è la 23, ma se ce ne fosse la necessità è possibile specificarne una diversa sulla riga di comando:

```
% telnet -l nomeutente host porta
```

Si può anche omettere buona parte degli argomenti e usare il `telnet` in maniera più diretta:

```
% telnet sherwood
Trying 192.168.1.4...
Connected to sherwood.loa.taz.
Escape character is '^]'.
Debian GNU/Linux 2.2 sherwood.loa.taz
sherwood login:
```

e inserendo la coppia utente/password otteniamo una shell.

Inoltre `telnet` può essere utilizzato per interagire anche con altri demoni, ad esempio con un demone smtp (che è il server di posta), o anche per collegarsi ad un server http o irc, per lo più per motivi di diagnostica e in questo caso possiamo anche evitare di specificare il nomeutente. Ad esempio una connessione diretta al un server di posta locale si fa così:

```
% telnet 127.0.0.1 25
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
help 220 sherwood.loa.taz ESMTP
Exim 3.12 #1 Tue, 23 Oct 2001 21:14:41 +0200 214
-Commands supported: 214-
    HELO EHLO MAIL RCPT DATA AUTH
214 NOOP QUIT RSET HELP
```

ssh 8.1.2

`ssh`, alla stessa stregua di `telnet`, serve per loggarsi e interagire con un sistema. La differenza sostanziale tra `telnet` e `ssh` risiede nel fatto che quest'ultimo è più sicuro: le connessioni sono infatti crittate con un algoritmo a scambio di chiavi. La sintassi di `ssh` è:

```
% ssh nomeutente@host -p porta
```

Lo switch -p è nella maggior parte dei casi superfluo, e serve solo se la configurazione del demone `ssh` sull'host remoto non è quella standard. Se ci si connette con un modem o in generale con una canale lento, conviene specificare anche l'opzione -C per abilitare la compressione dei dati:

```
% ssh -C -v nomeutente@host
```

Se utilizzate una connessione veloce, o siete in una lan, lo switch -C non farà altro che rallentare la comunicazione; con -v richiediamo al programma di darci più informazioni sulla connessione (*verbose*).

Ecco l'output di ssh utilizzando il -v:

```
% ssh -v little@sherwood.loa.taz
SSH Version OpenSSH-1.2.3, protocol version 1.5.
Compiled with SSL.
debug: Reading configuration data /etc/ssh/ssh_config
debug: Applying options for *
debug: ssh_connect: getuid 1000 geteuid 1000 anon 1
debug: Connecting to sherwood.loa.taz [192.168.1.1] port 22.
debug: Connection established.
debug: Remote protocol version 1.5, remote software version OpenSSH-
1.2.3
debug: Waiting for server public key.
debug: Received server public key (768 bits) and host key (1024 bits).
debug: Host 'littlejohn.loa.taz' is known and matches the host key.
debug: Encryption type: 3des debug: Sent encrypted session key.
debug: Installing crc compensation attack detector.
debug: Received encrypted confirmation.
debug: RSA
authentication using agent refused.
debug: Doing password authentication.
little@sherwood.loa.taz's password:
```

Se utilizzate ssh molto spesso verso determinati host, pu farvi piacere automatizzare il meccanismo di autenticazione in modo tale da non inserire ogni volta la password. Saranno necessari pochi passi. Per prima cosa occorre creare una coppia di chiavi (privata/pubblica), utilizzando il comando ssh-keygen:

```
% ssh-keygen
Generating RSA keys: ......[....]
Key generation complete.
Enter file in which to save the key (/home/little/.ssh/identity):
                                            /home/little/.ssh/identity
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/little/.ssh/identity.
Your public key has been saved in /home/little/.ssh/identity.pub.
The key fingerprint is:
1024 12:34:56:78:90:ab:cd:ef:12:34:56:78:90:ab:cd:ef little@robinhood
%
```

Gli unici due input che diamo al programma sono il nome del file utilizzato per salvare le chiavi e la passphrase. L'operazione successiva consiste nel copiare il file con la chiave pubblica (`/home/little/.ssh/identity.pub`) sull'host remoto, appendendolo al file `authorized_keys` (create il file se non esiste) nella directory *.ssh* del vostro utente. Ai prossimi login il client e il server ssh (`sshd`) si scambieranno le informazioni necessarie per la connessione, in maniera del tutto trasparente. L'effetto lo stesso che si ottiene con il programma rlogin, oramai caduto in disuso per motivi di sicurezza.

L'algoritmo stato pensato in modo da essere sicuro e da non mandare dati relativi alla chiave privata in rete (nella manpage di `ssh` c' una descrizione sommaria del metodo utilizzato), la vostra unica attenzione nei confronti di chi accede fisicamente alla macchina... purtroppo `ssh` non cos evoluto da riconoscere chi sta usando il vostro computer.

8.2 Spostare file da un host con `gftp` e `ncftp`

`gftp` 8.2.1

Il `gftp` è un client ftp grafico, molto simile a wsftp o ad altri client grafici in ambiente windows. È un client molto user-friendly, che supporta features avanzate come il drag'n'drop. Per collegarsi all'host vanno riempiti i campi in alto e premere invio quando il cursore è ancora in uno di essi, oppure direttamente utilizzando il menu "Remoto", e scegliere successivamente "Apri URL".

In ogni menu a fianco alle voci che lo compongono ci sono gli shortcut. Usali e sarai più veloce.

Per trasferire file sull'host bisogna prima selezionarli nella finestra alla nostra sinistra (quella in cui c'è scritto "Local") e per trasferirli si clicca sulla freccettina (vedi disegno), mentre per scaricarli basta fare l'operazione opposta, ovvero selezionare i file nella finestra di destra, e cliccare sull'altra freccettina.

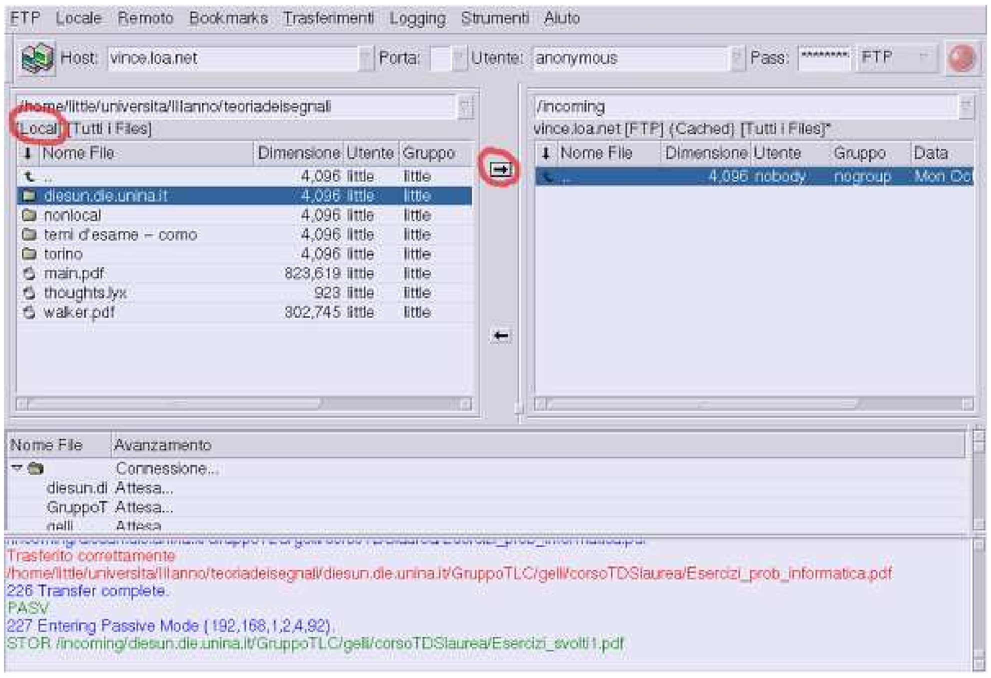


Figure 8.1 Una schermata di `gftp` in azione

Durante i trasferimenti nella parte inferiore c'è un pannello che ci terrà sempre informati su quanto accade, mostrando di fatto i log della connessione. Per configurare al meglio `gftp` occorre editare i valori della finestra delle opzioni raggiungibile dal menu "Ftp". Dal numero delle opzioni presenti si può ben capire qual è il livello del programma e la sua versatilità, e i meno esperti non si devono troppo preoccupare di comprendere il significato di ogni singola opzione, dato che nella maggior parte dei casi le opzioni di default sono già ottimali.

ncftp 8.2.2

ncftp a differenza di gftp è un client di tipo testuale, ma non per questo meno avanzato. Infatti tra le features di ncftp ci sono il completamento automatico dei nomi dei file e delle directory usando il tasto TAB (proprio come nella shell), la cancellazione ricorsiva delle dei comandi, una interazione completa con l'ambiente locale e molto altro. Per accedere ad un server ftp in maniera anonima, cioè ad un server che ospita contenuti pubblici, è sufficiente invocare ncftp in questo modo:

```
% ncftp sherwood.loa.taz
NcFTP 3.0.0 beta 21 (October 04, 1999) by Mike Gleason (ncftp@ncftp.com).
Connecting to sherwood.loa.taz...
ProFTPD 1.2.0pre10 Server (Debian) [sherwood.loa.taz]
Logging in...
Welcome, archive user anonymous@sherwood.loa.taz !

The local time is: Mon Oct 22 13:00:32 2001

This is an experimental FTP server.  If have any unusual problems,
please report them via e-mail to <root@sherwood.loa.taz>.

Anonymous access granted, restrictions apply.
Logged in to sherwood.loa.taz.
ncftp / >
```

quindi specificando solamente l'indirizzo ip, il client utilizzerà di default l'utente anonimo. Per ottenere la lista dei comandi basterà digitare help al prompt di ncftp:

```
ncftp / > help showall
Commands may be abbreviated.  'help showall' shows hidden and unsupported
commands.  'help <command>' gives a brief description of <command>.

!           bye        exit       lmkdir     mkdir      put        set
?           cat        get        lookup     mls        pwd        show
ascii       cd         help       lpage      more       quit       site
bgget       chmod      hosts      lpwd       mput       quote      symlink
bgput       close      jobs       lrename    open       rename     type
bgstart     debug      lcd        lrm        page       rglob      umask
binary      delete     lchmod     lrmdir     pdir       rhelp      version
bookmark    dir        less       ls         pls        rm
bookmarks   echo       lls        mget       prefs      rmdir
```

molti dei comandi listati sono quelli supportati da un qualunque client ftp, ma ce ne sono altri che rendono questo programma davvero unico. I comandi che cominciano con bg (cioè *bgget*, *bgput* e *bgstart*) permettono di lanciare delle operazioni di upload o download in background. Se ad esempio volessimo scaricare un grosso file, ma nel frattempo avere la possibilità di navigare il contenuto delle altre directory ci basterebbe istruire `ncftp` in questo modo:

```
ncftp / > bgget bigfile.tar.bz2
  + Spooled: get bigfile.tar.bz2
ncftp / > bgstart
Background process started.
Watch the ''/home/little/.ncftp/batchlog'' file to see how it is progressing.
ncftp / >
```

e lo stesso vale per le operazioni di upload, utilizzando *bgput* al posto di *bgget*.

I comandi *bgget* e *bgput* non fanno partire rispettivamente le operazioni di download e upload, ma semplicemente le mettono in spool. Per dare inizio al processing dello spooling di `ncftp` bisognerà invocare il comando *bgstart*.

L'interazione offerta da `ncftp` con l'ambiente locale è davvero alta, come si vede dai comandi che cominciano per "l":

```
ncftp / > l
    lcd      less     lmkdir   lpage    lrename  lrmdir
    lchmod   lls      lookup   lpwd     lrm      ls
```

Per ottenere la lista dei comandi si può sfruttare il completamento automatico: digita le prime lettere di ciò che ti interessa e premi il tasto TAB.

Tranne *less*, *lookup* ed *ls*, gli altri comandi ci permettono di operare in maniera completa sul filesystem locale con comandi molto intuitivi.

Per poter configurare `ncftp` in maniera opportuna possiamo settare una serie di variabili visualizzabili con *show*:

```
ncftp / > show
anon-password                   littlejohn@sherwood.loa.taz
auto-resume                     no
autosave-bookmark-changes       no
confirm-close                   yes
connect-timeout                 20
control-timeout                 135
logsize                         10240
pager                           more
passive                         on
progress-meter                  2 (statbar)
save-passwords                  ask
show-status-in-xterm-titlebar   no
so-bufsize                      0 (use system default)
xfer-timeout                    3600
ncftp / >
```

Ad esempio per disattivare il *passive mode*, digiteremo:

```
ncftp / > set passive off
ncftp / >
```

per constatare l'efficacia di quanto fatto basterà ridigitare *show*:

```
ncftp / > show
...
passive                         off
...
ncftp / >
```

infatti la riga passive indica ora off.

Un'altra feature di `ncftp` è la possibilità di utilizzare dei "bookmark". Quando chiudiamo una connessione ftp, ci sarà chiesto di salvare un bookmark per il server corrente (se ciò non fosse stato fatto in passato):

```
ncftp / > bye

You have not saved a bookmark for this site.

Would you like to save a bookmark to:
ftp://192.168.1.2

Save? (yes/no) yes
Enter a name for this bookmark: sherwood
Bookmark ‘‘sherwood’’ saved.
```

Quindi per una nuova connessione con questo server digiteremo:

```
% ncftp sherwood
```

> Tutti i bookmark sono nel file /.ncftp/bookmarks, mentre la configurazione si trova nel file prefs della stessa directory

Se siamo in possesso di un account ftp presso un host, ci basterà specificare il nome dell'utente prima dell'host usando lo switch -u:

```
% ncftp -u littlejohn sherwood.loa.taz
```

8.3 Tutto sulla mail

Per poter utilizzare in maniera efficace tutte le potenzialit della posta elettronica servono principalmente tre programmi:

- fetchmail
- mutt
- procmail

Ma analizziamoli uno per uno.

fetchmail 8.3.1

fetchmail un ottimo tool scritto da Eric S. Raymond per poter scaricare i messaggi di posta elettronica da un server remoto. Configurarlo molto semplice, infatti a differenza di molti altri tools che incontrerete in *nix, potrete quasi scrivere il suo file di configurazione quasi in inglese. Per chi usasse un modem nella maggior parte dei casi risolver i suoi problemi scrivendo qualcosa di simile:

```
poll pop.loa.taz proto pop3 user littlejohn
 with password "password", is lidl here
```

ipotizzando che io abbia un indirizzo email littlejohn@loa.taz, e che la stia scaricando sulla macchina locale dall'utente lidl.

Che dire, strabiliante. Ma analizziamo questo formato:

keyword	Significato
poll serverpop	il comando che lanciamo al server di posta (nell'esempio il server di posta pop.loa.taz)
proto protocollo	il protocollo che stiamo utilizzando per comunicare con il server (nell'esempio, pop3)
user nomeutente	il nome dell'utente che abbiamo sul server, solitamente il nome che precede @ nell'indirizzo di posta
with password ''password''	la password di accesso alla casella di posta
is utente here	l'utente locale a cui recapitare la posta

fetchmail pu anche esser lanciato da riga di comando, ad esempio:

```
$ fetchmail -p pop3 pop.loa.taz -u littlejohn
```

che scaricher direttamente i messaggi nella mailbox dell'utente che l'ha lanciato.

Tra le opzioni "importanti" di fetchmail ci sono:

Opzione	Significato
`-k , --keep`	non cancella i messaggi dal server di posta remoto
`-p , --proto <proto>`	il protocollo di comunicazione, si veda la manpage per la lista
`--ssl`	per il supporto ssl, in modo da crittare la comunicazione per una maggiore privacy. L'unico problema è che l'ssl deve essere supportata anche da parte del server
`-d , --daemon <secondi>`	per lanciare `fetchmail` ogni tot ¡secondi¿ in modo da ricevere le email nel pi breve tempo possibile
`-L , --logfile <file>`	per salvare l'output di `fetchmail` in un file

Una volta che i messaggi sono scaricati in locale, dobbiamo poterli leggere. Per questo saltiamo alla prossima sezione, `mutt`.

◻ mutt 8.3.2

mutt, come dice la manpage, un "piccolo ma molto potente Mail User Agent", ovvero client di posta. Conta approssimativamente 200 variabili di configurazione, e tra le sue feature pi interessanti ricordiamo il thread sorting, ovvero la capacit di organizzare le email per thread in maniera "logica", oltre al supporto per delle regular expression della famiglia "POSIX Extended", per intenderci le stesse di egrep e awk (vedi il capitolo sulle regular expressions), e l'integrazione completa con i software di crittazione delle email come PGP o GnuPG. Il pacchetto debian di `mutt` (informazioni che trovate ampiamente anche sul sito web www.mutt.org) corredato da una serie di file di configurazione (muttrc) utilizzabili e molto istruttivi che in un primo tempo vi possono rendere la vita pi facile. In ogni caso vi consiglio di leggere il manuale successivamente per poter davvero avere un client che vi soddisfi in ogni sua parte (dall'editor al pager, ai programmi di visualizzazione degli allegati). A chi fosse abituato a client grafici, `mutt` non far subito una bella impressione, ma le sue potenzialit e la sua flessibilit non tarderanno ad impressionarvi. Nel prosieguo di questa sezione far riferimento al pacchetto `mutt` distribuito con GNU/Linux Debian, ma si tratta comunque di concetti generali e di ampia applicabilit.

Supponendo di aver ricevuto un paio di messaggi di posta, lanciamo mutt, e la sua schermata di default sar simile alla seguente

```
q:Esci  d:Canc  u:DeCanc  s:Salva  m:Mail  r:Rispondi  g:Gruppo  ?:Aiuto
------------------------------------------------------------------------
-> 1     Oct 29 Albert Einstein (  32) ecco la formula che cercavi
   2     Oct 29 Nicole Kidman   (1780) usciamo stasera?
```

Abbiamo quindi nella prima riga alcuni keystrokes per i comandi pi importanti, e successivamente le due email, con il numero della email, la data di arrivo, il nome del mittente, la dimensione del messaggio e il subject.

154 154

Vediamo pi da vicino i comandi presenti nella barra di mutt:

Keystroke	Significato
q	esci dal programma
d	marca il messaggio corrente come cancellato
u	se il messaggio corrente marcato come cancellato, lo demarca
s	salva il messaggio corrente in una mailbox da specificare successivamente
m	scrivi una email
r	rispondi alla email corrente
g	rispondi alla email corrente, inviando copia della risposta anche a tutti i destinatari
?	help per tutti i keystrokes

Quando marchiamo i messaggi ad esempio come cancellati, la cancellazione non immediata, per sincronizzare la casella baster premere $, oppure alla chiusura del programma ci verr richiesto di decidere se cancellare i messaggi o no. Nel momento in cui decidiamo di scrivere una email (premendo m), `mutt` ci domander a chi mandare il messaggio (To:) e il motivo della email (Subject:), aprendo poi l'editor di default. Una volta scritta l'email, ed esser usciti dall'editor avendo salvato il file, `mutt` ci fa vedere una schermata riassuntiva simile a questa:

```
y:Spedisci q:Abbandona t:To c:CC s:Subj a:Allega un file d:Descr ?:Aiuto
------------------------------------------------------------------------
    From: lidl <littlejohn@sherwood.taz>
      To: Nicole Kidman <nicole.kidman@sherwood.taz>
      Cc:
     Bcc:
 Subject: ceniamo da me
Reply-To:
     Fcc:
     Mix: <no chain defined>
     PGP: Normale
```

Ancora una volta `mutt` completo, fornendoci i keystrokes utili alla gestione del messaggio in questa fase (vedi la prima riga):

keystroke	Significato
y	invia la mail
q	non invia la mail e vi chiede se la volete postporre
t	edita il campo To:
c	edita il campo Cc:
s	edita il subject
a	allega file alla mail

d descrive gli allegati
? help per tutti i keystrokes

Potete aggiungere allegati in due modi, dopo aver premuto il tasto “a”, potete inserire il percorso completo del file oppure premere “?”, e navigare le vostre directory, premendo invio quando siete sul file da allegare.

Ora che abbiamo queste nozioni di base per la gestione del programma, cerchiamo di personalizzarlo un po’, editando il muttrc.

La sintassi del muttrc molto semplice, in generale basta indicare set seguito da una variabile e dal valore in questo modo:

```
set variabile = ‘‘valore’’
```

Delle impostazioni comodo possono essere quelle che seguono:

```
set beep_new                    # quando c’ un nuovo messaggio fa ‘‘beep’’
set editor="/usr/bin/emacs"     # non so voi, ma io uso emacs
set edit_headers                # let me edit the message header when composing
set force_name=yes
set pager_index_lines=7
set postponed=~/Mail/postponed  # qui finiscono i messaggi posposti
set sort=threads                # ordina le email per thread, molto comodo se
                                # seguite delle mailing-list
```

Potete anche personalizzare i colori, io ad esempio uso:

```
color bold cyan default
color error brightblue brightyellow
color hdrdefault brightblue default
color quoted magenta default
color quoted1 green default
color quoted2 brightgreen default
color signature red default
color indicator brightred blue
```

ma potete sbizzarrirvi come vi pare. Naturalmente questi sono solo pochi parametri, ma non mi dilungher sulle altre opzioni, per quelle c’ il manuale di `mutt` che trovate sul sito o nella directory della documentazione sul vostro sistema (per chi avesse Debian, /usr/share/doc/mutt/manual).

Infine ci sono gli alias, che ci permettono di associare ad un indirizzo di posta un nome pi facile da ricordare e, in generale, pi breve da digitare. Il formato di un alias il seguente

```
alias nomebreve Nome Cognome <indirizzo@email.qui>
```

e quindi concretamente:

```
alias miadonna Nicole Kidman <nicole.kidman@loa.taz>
```

Gli alias vanno posizionati *alla fine* del .muttrc, uno per riga. Nel momento in cui scrivete una email, quando `mutt` vi chiede il "To:", potete scrivere nomebreve (nell'esempio, miadonna) e l'indirizzo sar automaticamente completato.

Buona lettura della vostra casella.

procmail 8.3.3

`procmail` un programma che ci aiuta nella gestione delle email, permettendoci di decidere il destino di ognuna di esse nel momento in cui le scarichiamo. Per abilitare procmail, creiamo un file .forward nella nostra home e a seconda dell'MTA che usiamo (per esempio `sendmail`) scrivete una riga come questa:

```
"|IFS=' '&&p=/usr/bin/procmail&&test -f $p&&exec $p -Yf-||exit 75 #YOUR_USERNAME"
```

o se usate exim, pi semplicemente quest'altra:

```
|/usr/bin/procmail
```

> Il file .forward deve essere leggibile da tutti "world readable", o `procmail` non funzioner.

Per sapere che server di posta possediamo, usiamo il comando:

```
% grep smtp /etc/inetd.conf
% smtp      stream  tcp    nowait  mail   /usr/sbin/exim exim -bs
                                          --------------
```

la parte contrassegnata il sever di posta

Se siamo fortunati, procmail gi installato come "delivery agent" di default sul vostro sistema (chiedete all'amministratore di sistema, se l'amministratore sei tu stesso e non sai se procmail l'MDA di default, ti propongo una pausa di riflessione ;-), per cui vi baster creare il file .procmailrc, che contiene la configurazione di procmail.

Una volta che avete installato procmail in un modo o in un altro, dobbiamo configurarlo. Come prima cosa, ci vogliono delle impostazioni di carattere generale:

```
PATH=/usr/bin:/usr/local/bin:/bin
MAILDIR=$HOME/Mail
DEFAULT=/var/spool/mail/little
LOGFILE=$MAILDIR/procmail.log
```

dove MAILDIR la directory che conterr la posta, DEFAULT il file che contiene l'INBOX, LOGFILE il file che sar utilizzato per tracciare il comportamento di procmail. Fatto ci possiamo definire i filtri per la posta. Un filtro d'esempio pu essere il seguente:

```
:0:
* ^From.*business.news@libero.it.$
spam

:0:
* ^X-Mailing.*kernel.org$
kernel
```

Si tratta di due filtri, il primo server per inviare nella cartella spam i messaggi inviati dall'indirizzo business.news@libero.it (l'operazione sar totalmente efficace se linkiamo la cartella spam al "buco nero" del pc, /dev/null), mentre il secondo invece serve a direzionare i messaggi delle mailinglist dello sviluppo del kernel nella cartella kernel.

È evidente che la prima riga (:0:) indica l'inizio della nuova "ricetta" (l'autore di procmail usa la parola recipies), seguita dalla condizione e dalla cartella in cui vengono direzionate le email. La sintassi completa di una ricetta questa (presa direttamente dalla man page di procmailrc):

```
:0 [flags] [ : [locallockfile] ]
<zero or more conditions (one per line)>
<exactly one action line>
```

Andiamo con ordine.

flags 8.3.4

Le flags pi importanti sono:

flag	Significato
H	Applica la condizione (regular expression) all'header (default)
B	Applica la condizione al corpo del messaggio
D	La regular expression sar case sensitive

Ci sono altre flags, che riguardano i casi pi complessi che qui non tratto, ma che trovate in maniera abbastanza completa nella manpage di procmailrc. Se le vostre esigenze sono quelle di un utente casalingo che riceve normalmente la posta dalla nonna e dall'amica di banco, e volete separare i due ambiti (giustamente ;-), potete anche non specificare nessuna flag, quella di default sugli header sar pi che sufficiente.

conditions 8.3.5

Le condizioni partono con un *, e sono processate dall'egrep interno (che totalmente compatibile con la sintassi di egrep, con l'unica differenza che quello di `procmail` case insensitive per default). Si tratta di regular expressions vere e proprie (vedi il capitolo sulle regular expressions), quindi non c' molto da dire. La comodit che potete specificare pi regular expressions (una per riga), per tenere il procmailrc pi compatto e leggibile. Ad esempio:

```
:0:
* ^From.*iOLnews@libero.it.$
* ^From.*mail.lucky.it.$
spam
```

con evidente significato dei simboli.

action 8.3.6

Solitamente basta scrivere la mailbox in cui vogliamo stipare l'email. Ma `procmail` offre di pi:

action	Significato
!	forwarda le mail interessate dalla condizione agli indirizzi che seguono
|	permette di processare l'email con un programma esterno

mailstat 8.3.7

Una volta che abbiamo scaricato tutta la posta, `procmail` l'avr smistata a dovere. Ma come facciamo a sapere esattemente quanti messaggi sono stati direzionati nelle diverse caselle? `procmail` scrive tutto nel suo file di log ed esiste un programma, mailstat, che interpreta questo file in questo modo:

```
% mailstat Mail/procmail.log

  Total  Number Folder
  -----  ------ ------
 553206      18 /var/spool/mail/little
 181385       9 spam
  -----  ------
 734591      27
18:45 - little@littlejohn ~ %
```

cos è più facile capire cosa successo durante il donwload delle mail.

Giochi di prestigio con procmail 8.3.8

In questa sezione analizziamo assieme alcuni degli esempi che si trovano nella manpage procmailex. Cominciamo con questo semplice esempio:

```
:0
* ^From.*peter
* ^Subject:.*compilers
{
     :0 c
     ! william@somewhere.edu

     :0
     petcompil
}
```

la prima riga (:0) comincia la recipe, seguita da due conditions. Al posto di action, troviamo una parentesi graffa che comincia un blocco che specifica cosa fare delle email con due diverse actions. La prima action (:0 c) serve per creare una copia (proprio una carbon copy) della mail e a forwardarla (!) a william, la seconda invece semplicemente manda la mail nella mailbox petcompil.

Altro esempio:

```
:0 hwc:
* !^FROM_MAILER
| gzip >>headc.gz
```

(:0 hwc:), dice di inviare l'header della mail al programma specificato due righe dopo con il | (gzip) (h), aspettando che il programma specificato completi la sua operazione (w) e facendo una carbon copy della mail (c). Come condition, vogliamo tutte le email che *non* provengono dal postmaster (notate il ! iniziale). L'action è appunto un pipe a gzip. A che serve questa ricetta? Semplicemente crea un archivio (headc) con tutti gli header delle mail che vi arrivano (a questo punto potremmo discutere lungamente sull'utilit di tutto ci `:-)`.

Ultimo esempio, sul reply automatico.

```
:0 h c
* !^FROM_DAEMON
* !^X-Loop: your@own.mail.address
| (formail -r -I"Precedence: junk" \
      -A"X-Loop: your@own.mail.address" ; \
      echo "Mail received.") | $SENDMAIL -t
```

Niente di strano fino al pipe (vegono esclusi i messaggi del postmaster e i propri), ed inviato l'header al programma del pipe. formail un programma che vi permette di fare tante belle cose, tra cui inviare delle mail da riga di comando. Non mi soffermer su formail, perch esula dagli scopi di questa versione del capitolo. Al di l del comando utilizzato, voglio sottolineare l'uso del doppio pipe, e della variabile SENDMAIL. SENDMAIL fa parte della famiglia di variabili interne di `procmail` (la lista completa nella manpage procmailrc), ed settata pari a /usr/sbin/sendmail, ma potete scegliere un qualsiasi altro valore, ad esempio:

```
SENDMAIL = /usr/sbin/exim
```

posizionando la riga in testa al file .procmailrc .

8.4 Navigare in rete

lynx 8.4.1

`lynx` tra i pi anziani browser testuali, con circa dieci anni di sviluppo alle spalle. La navigazione un po' ostica (finquando non prendete la mano ;-) e poco agevole l dove ci sono troppi frames. In ogni caso si tratta di un programma squisitamente comodo quando si tratta di visitare pagine che contengono immagini "pesanti", o per navigare molto velocemente con il nostro 486 (personalmente lo uso anche sul PIII). La riga di comando abbastanza intuitiva:

```
# lynx nomesito.xxx
```

e saremo in tempo breve sulla pagina. I keystrokes di default di `lynx` sono

keystroke	Significato
q	esce dal programma
F1 o H	help
g	vai al sito
G	vai su una pagina web partendo dall'url corrente
l	visualizza i link nella pagina corrente

→	segui il link evidenziato
←	torna indietro
↑	evidenzia link precedente
↓	evidenzia il link successivo
d	download il link selezionato
a	ci fa scegliere tra il salvataggio del documento corrente o salvare il link in un bookmark (d/l)
backspace	visualizza la history
spazio	scrolla il documento corrente di una pagina avanti
b	scrolla il documento corrente di una pagina indietro

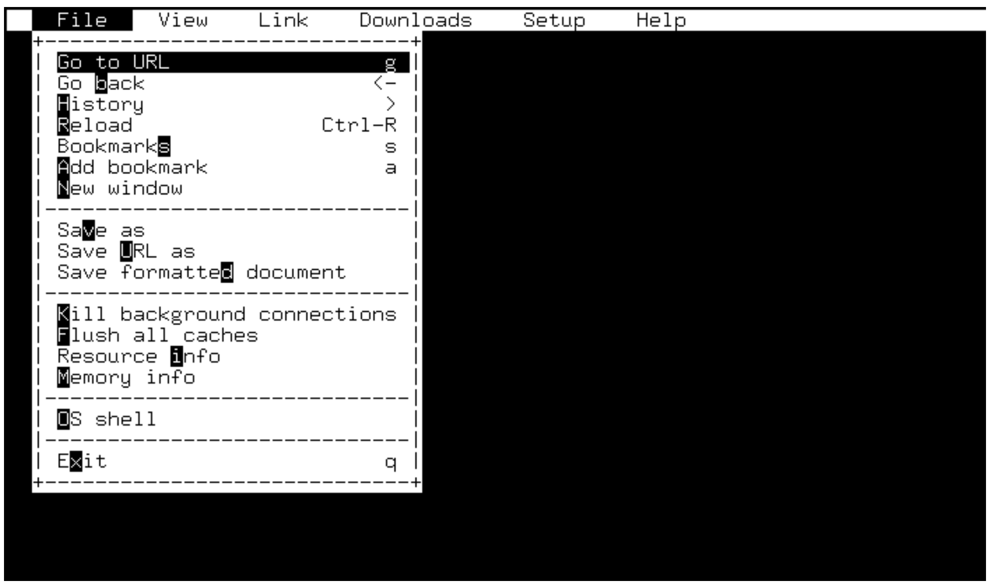


Figure 8.2 Una schermata di `links` in azione

◻ links 8.4.2

`links` un altro client testuale, che rispetto a lynx offre il supporto per i frames, il download in background e una formattazione delle tabelle pi decente, giusto per citare le differenze pi marcate. Inoltre si differenzia da lynx per la presenza di un menu (a cui si accede premendo F10) che ci permette di scegliere tra le funzioni pi comuni del browser (apri l'url, scarica il file, reload...).

Naturalmente i programmatori non ci costringono ad usare solo i menu, ma le funzioni pi comuni sono collegate ad uno shortcut (un keystroke):

Keystroke	Significato
g	apri url
backspace	vai alla pagina precedente
ctrl-R	ricarica la pagina
/	cerca una stringa nella pagina
backslash	visualizza il codice HTML della pagina
D	scarica il file
→	segue il link corrente

In ogni caso per avere la lista completa degli shortcut, ci tocca selezionare l'ultimo menu, help, e successivamente keys.

◻ mozilla 8.4.3

mozilla il browser grafico per antonomasia, derivato dai sorgenti di Netscape 5, in costante aggiornamento ed evoluzione. mozilla basato sul motore di rendering delle pagine html chiamato Gecko, che non ha concorrenti in fatto di performance e qualit di rendering, e su cui sono basati anche altri programmi opensource. Uno dei punti contro mozilla la sua pesantezza. Se volete usare stabilmente questo programma assicuratevi di avere almeno un computer non inferiore al p200mmx per quanto riguarda la potenza di calcolo, e 64mb di ram, per avere delle prestazioni sufficienti (sul mio p166mmx, con 128 mb ram ci vuole quasi un minuto per lanciarlo e si rischia la crisi di nervi se si ha fretta). Dopo questa breve introduzione, vediamo come usare mozilla.

L'interfaccia molto simile a quella di Netscape, con i menu oramai di rito in un programma grafico motivo per cui non mi soffermer su di essi, lasciandovi la curiosit di navigarli.

Figure 8.3 La barra strumenti di mozilla

L'argomento di cui mi voglio occupare quello della gestion della privacy con mozilla. Se andate sul menu Edit e scegliete Preferencies, vi comparir una schermata in cui tra le diverse voci di personalizzazione, c' "Privacy and Security". Vediamo una per una le voci, cercando di capire il loro significato.

◻ Cookies 8.4.4

Un cookie un piccolo file che un server web pu lasciare e leggere sul vostro computer, e pu contenere qualsiasi cosa (nel senso che il browser non effettua un controllo su cosa il server web vi invia). Che tipo di informazioni vengono salvate di solito? Nella maggior parte dei casi si tratta di contatori per raccogliere delle informazioni statistiche (quante volte ad esempio visitate un sito), ma con l'avvento della pubblicit selvaggia in rete, gli scopi sono altri (sapere quali banner avete gi visto, in modo da non riproperveli, o peggio collezionare una serie di informazioni sui siti che visitate per poi elaborare un profilo delle vostre abitudini e preferenze). Personalmente vedo i cookie come "un'intrusione" della mia privacy e per fortuna esistono gli strumenti per poter evitare questo genere di intrusione. Le scelte a nostra disposizione sono 3:

- Disable cookies
- Accept cookies for the originating web site only
- Enable all cookies

La prima opzione la migliore, visto che cos non accetterete nessun cookie. Alcuni siti, specie quelli con delle sezioni ad accesso con password, hanno necessit dei cookie, ma in questo modo non riuscirete a visitarli.

La seconda un passo intermedio tra l'accettazione totale dei cookie (cio la terza opzione) e il rifiuto dei cookie illeggittimi, visto che il browser accetterr i cookie solo dal sito che state effettivamente visitando e non da altri server. Non di grande utilit, ma meglio di niente. Potete anche scegliere di essere avvisati ogni volta che un server cerca di creare un cookie per poi scegliere se accettare o meno.

Per avere un'idea generale della privacy in rete, cliccate sul tasto More Information in questa pagina

Images 8.4.5

Pi che impostazioni di sicurezza, si tratta di personalizzare il comportamento del browser nei confronti delle immagini sui siti. L'impostazione pi interessante quella sul loop delle immagini. Nella maggior parte dei casi le immagini gif animate servono per rendere i banner pubblicitari pi visibili, facendoli ad esempio lampeggiare nella pagina con una frequenza elevata. Per evitare che ci accada, semplicemente scegliete nel riquadro in basso, "Never".

Forms 8.4.6

Quando riempite dei form in rete, `mozilla` pu salvare quanto avete scritto. Si tratta di un'arma a doppio taglio, perch se da un lato vi permette, ad esempio, di iscrivervi ad un servizio on-line senza riempire lunghi moduli, dall'altro si tratta di impostazioni salvate sul computer che state utilizzando in quel momento, facilmente prelevabili a chi abbia accesso alla vostra home.

Web Passwords 8.4.7

Si tratta delle password inserite nei siti in cui sia richiesto un login, ad esempio le caselle di posta online. Come per i form, scegliete l'impostazione pi idonea, a seconda che si tratti di un computer pi o meno accessibile ad altri utenti. Una forma di sicurezza in pi, consiste nella crittazione delle informazioni sul disco, che ottenete selezionando il box "Use encryption when storing sensitive data".

Master Passwords 8.4.8

Nel momento in cui decidete di salvare le informazioni personali sul computer, potete cercare di preservarle utilizzando la Master Password. Se abilitate questa feature, `mozilla` vi chieder una password per accedere alle informazioni sul disco. A seconda dell'impostazione che scegliete vi sar richiesta la master password con pi o meno frequenza.

SSL 8.4.9

L'ssl permette di crittare i dati tra voi e il sito che state visitando. Sarebbe d'obbligo nei siti in cui sono presenti servizi come webmail e simili, in modo da non permettere che nessuno possa leggere le vostre email mentre le scaricate dal sito. Come consiglio generela, bene abilitare il supporto alle diverse versioni di SSL, e chiedete a `mozilla` di avvisarvi se:

- ci sono siti con una crittazione troppo debole (Loading a page that uses low-grade encryption)
- lasciate la parte di sito in cui la comunicazione crittata (Leaving a page that supports encryption)
- inviate dei form su un canale non crittato (Sending form data form an unencrypted page to an uncrypted page)
- visitate pagine che sono in parte crittate e in parte non crittate (Viewing a page with an encrypted/unencrypted mix)

Quello della transitazione dei dati in chiaro un problema purtroppo non molto avvertito dai navigatori, ed un vero peccato, visto che lo sniffing (ascolto passivo delle informazioni in transito) una delle forme di abuso pi utilizzate e pi temibili perch non si pu evitare se non con la crittografia.

Certificates 8.4.10

Un certificato non altro che un altro strumento di crittografia, per esser certi che le informazioni che un sito ci sta passando sono autentiche, cio che non ci sia stato "poisoning" (avvelenamento) di quanto riceviamo dal server http da parte di terzi. Quando un sito ci offre un certificato, `mozilla` ci mostra le caratteristiche del certificato, e se non ci fidiamo possiamo scegliere di non accettarlo.

galeon 8.4.11

`galeon` un "hack" di mozilla, che mira ad essere un software di navigazione pi leggero e che faccia solo quello (il suo motto "The web, only the web"...). Come dire, "solo" un web browser. Le sue caratteristiche sono la leggerezza e la qualit di rendering, visto che `galeon` basato sul motore di rendering gecko (esatto, lo stesso di mozilla). Mentre `mozilla` offre anche un client di posta, un editor html e un address book, `galeon` un semplice browser (e forse l'unica cosa di cui gli utenti hanno davvero bisogno). È molto semplice da usare, avendo un'interfaccia simile a quella di ogni browser grafico, per cui non mi soffermer sulla trattazione del pacchetto. Un consiglio, installatelo e usatelo.

9. X Windows System

di Manhattan

X Window System[1] è lo standard grafico di riferimento per i sistemi UNIX.
X non può essere definito semplicemente GUI o Graphical User Interface, ma è l'esempio di un vero e proprio sistema grafico completo. Esso non fornisce una singola interfaccia all'utente, ma un intero set di GUI da cui scegliere quella da utilizzare di volta in volta. Non importa, poi, quale applicazione grafica deciderete di visualizzare all'interno di X, essa funzionerà[2] egregiamente anche se non espressamente creata per l'ambiente grafico caricato al momento.

9.1 Un po' di storia ...

All'inizio degli anni '80 l'MIT, il Massachuset Institute of Technology decise di proseguire l'esperienza della Stanford University nel campo degli ambienti grafici, trasformando il *W Window System* in X Window System. Contemporaneamente, dal 1983, il *Project Athena* del MIT si occupava di creare un ambiente distribuito di computazione all'interno del Campus.

Le idee scaturite da questi progetti si fusero nell' *X Consortium*, che vide la sua nascita nel 1988.

Contrariamente al sistema grafico sviluppato dalla Apple (nel 1984 usciva il primo Mac), X non portava con sè la qualità grafica tipica dell'ambiente Macintosh, il cui team di programmatori era composto in gran parte da artisti e designer, tuttavia ebbe il pregio di essere concepito come **indipendente** dalla piattaforma hardware su cui avrebbe potuto essere implementato.

Semplicemente, non potendo sapere quale tipo di hardware si sarebbe reso disponibile, diventò impensabile creare un ambiente grafico legato ad una singola architettura.

Come prevedibile, le prime versioni di X furuno supportate da una ristretta cerchia di sistemi e piattaforme, ma il concetto di *terminale grafico* aveva ormai preso piede: invece di caricare una CPU del peso di un intero sistema grafico, perché non lasciare il compito della gestione dell'intero apparato ad un grosso sistema distribuito che avesse le capacità di rendere disponibili ad ogni macchina le applicazioni grafiche richieste?

I server UNIX, multitasking e multiuser, vennero così impostati in modo da permettere ad un semplice terminale di visualizzare, senza eccessivo dispendio di risorse interne, qualsiasi strumento grafico fosse disponibile sul server di riferimento.

9.2 Il modello Client/Server di X Window

È ormai noto a tutti il modello di gestione dei compiti, o *task*, all'interno di un sistema: esso viene definito *modello client/server*.

La nomenclatua indica come server la macchina dotata di maggiori risorse rispetto alla macchina client, la quale delega e richiede alcuni servizi al server. Lo stesso concetto vale per le applicazioni e i programmi, siano essi all'interno di un sistema o in distribuiti all'interno di una rete.

Un esempio classico è rappresentato dal binomio *Web Browser/Web Site*. Il browser, ad esempio Netscape Navigator, richiede i dati relativi ad una pagina web al Web Server del sito da visitare e aspetta da esso l'arrivo della pagina. Appena soddisfatta la richiesta, il browser non dovrà far altro che visualizzare il contenuto della pagina all'utente.

X Window ribalta i termini del concetto Client/Server[3]: server (*X server*) viene definito come l'apparato che visualizzerà la grafica richiesta mentre il **programma** che si occupa di fornire i dati al server viene definito *X client*. Spesso anche i professionisti hanno problemi a capire questa tipologia di modello; voi, al momento, accettatela e fate sì con la testa.

È molto più importante capire che ciò che vedrete visualizzato sullo schermo forse stà girando su una macchina remota. Il protocollo X Window è definito infatti come *network transparent*, poichè si comporta in modo uguale a prescindere dalla modalità di invio e trasporto dei dati. Significa che l'X server e i client possono condividere lo stesso sistema[4] oppure essere distribuiti su piu' macchine che parlino tra di loro l'X Window Protocol.

X Window Protocol 9.2.1

Il protocollo X Window è il linguaggio con cui il client invia i dati che dovranno essere visualizzati dal server. È un protocollo come lo sono l'HTTP o l'SMTP e consiste in un set di regole e procedure che permettono a due sistemi di comunicare tra di loro (in questo caso le regole permettono di disegnare oggetti su uno schermo).

Ogni macchina che sia in grado di interpretare il protocollo X Window e sia capace di disegnare sullo schermo immagini e finestre in modo corretto sarà dunque un X server.

3 Pensavate fosse così semplice?? . . . In effetto lo è!
4 Il vostro computer portatile

Anche una macchina Windows o Macintosh potrà implementare tale protocollo e dunque visualizzare la grafica secondo gli standard X.

9.3 L'evoluzione di X negli anni

Come UNIX, come ogni software o sistema che abbia più di 15 anni di storia, anche X Window si è evoluto ed è migliorato.

Attualmente la versione corrente di X Window è X11R6.6, cioè versione 11, revisione 6.6[5].

La versione di X Window in precedenza usata era datata Settembre 1987. In questi 13 anni numerosissime applicazioni, sistemi e ambienti grafici si sono adattati allo standard X e si puo' tranquillamente stimare che le nuove revisioni non porteranno modifiche incisive ad X, permettendo in tale maniera di conservare la compatibilità tra un appplicazione creata nel 1989 con un sistema del 2002[6].

9.4 Meccanismi, non regole

Se il protocollo su cui si basa X e la sua gestione client/server cambia cosi' raramente, come fare ad aggiungere nuove funzioni al passo con i tempi?

La domanda in sè è banale, ma ci permette di introdurre alcuni aspetti positivi e negativi della concezione grafica di X. "Meccanismi, non regole" sta a sigificare che, contrariamente alle GUI tradizionali di Windows & Macintosh, X non impone nessun tipo di policy riguardo l'aspetto, la forma, la disposizione e le funzioni di nessuna applicazione, ma fornisce esclusivamente un metodo per visualizzarle (e per questo motivo il suo protocollo è trasparente).

Gli svantaggi di un simile approccio consistono nella mancanza di una standardizzazione del cosiddetto *look-and-feel*[7] delle applicazioni, nella mancanza di regole definite per l'attribuizione di talune funzioni o per il raggiungimento di particolari scopi e, in ultima analisi, in un aspetto meno uniforme dell'intera interfaccia grafica.

I vantaggi, però, sono quelli derivanti dalla possibilita' di modificare e di evolvere le varie applicazioni senza dover modificare il substrato che permette il loro uso: così,

5 Tale versione e' stata rilasciata il 25 Aprile 2001
6 Windows e il suo ambiente grafico non esistevano nemmeno nel 1989, e sono d'altra parte incompatibili anche con le loro stesse versioni!
7 Anche se non mi sembra uno svantaggio

seppur rallentati, i moderni programmi possono girare su piattaforme vetuste o poco potenti, senza doversi curare della necessità di creare nuove regole di visualizzazione.

In ultima analisi, X Window System consente di ottenere risultati impensabili, in termini di portabilita e flessibilità delle applicazioni, rispetto a qualsiasi altro ambiente grafico.

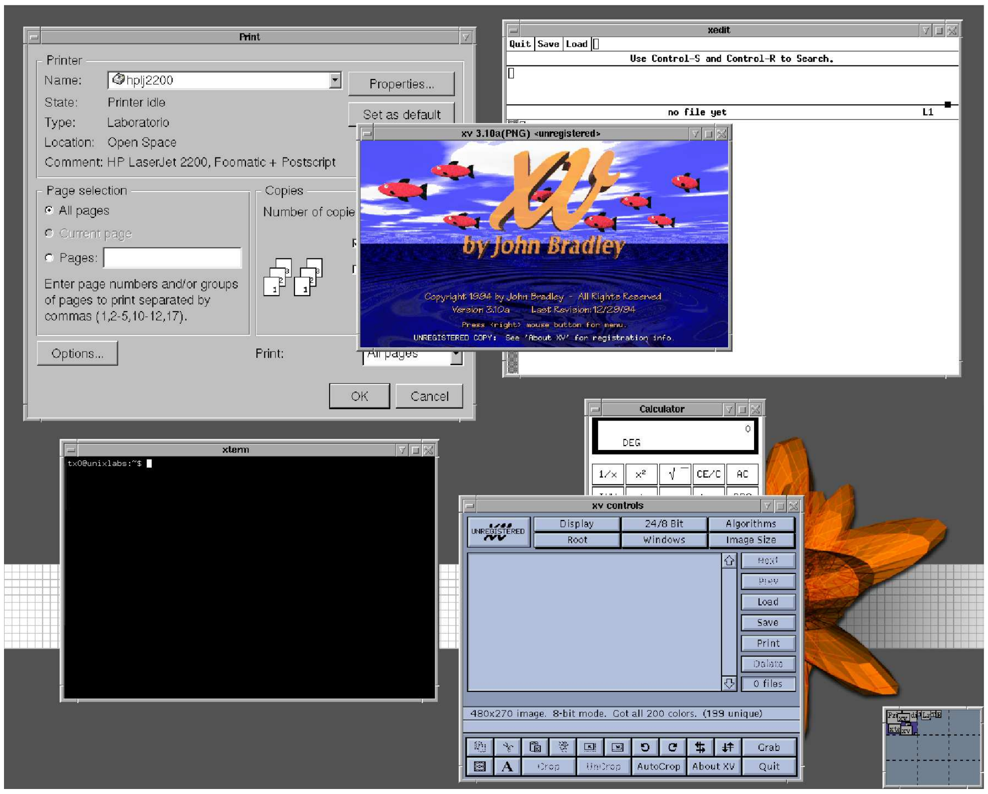


Figure 9.1 Desktop spartano basato su FVWM 2.2

9.5 Gerarchie, Widgets e Toolkit

Rispettando l'approccio gerarchico tipico di UNIX, che viene ad esempio utilizzato nella gestione delle directory all'interno del filesystem, anche in ambiente X la finestra pricipale, il background, è definita come *root window*. Ogni altra finestra viene creata e dipende in linea gerarchica dalla propria *parent window*, la finestra d'origine. La parent window primaria è dunque la root window.

Questo particolare fa sì che sia dunque possibile operare, ad esempio, su interi gruppi di finestre contemporaneamente, basandosi sulle proprietà di una parent window.

Ma come vengono create, in linea di massima, tali finestre? Pur non essendoci degli standard univoci, com'è possibile disegnare a schermo un oggetto o una finestra?

Il primo passo consiste nella scelta di una serie di *widget* e di *toolkit*, poi, necessariamente, nella scelta di un *window manager*.

Widget stà ad indicare, nel linguaggio comune, ogni oggetto che sia possibile utilizzare, in via ipotetica, in sostituzione di un oggetto attuale: qualcuno potrebbe tradurre widget con il termine "*un coso*", "*un ambaradan*"[8]

In termini di X Window System, widget sono tutti i particolari che compongono una finestra, come la scrollbar, i menù, tutti i vari tipi di bottoni e via dicendo. Ognuno è libero di creare e utilizzare un insieme di widget a picere, anche se, di fatto, esistono da molto tempo set di widget ben rodati che permettono di creare con facilità ogni tipo di interfaccia.

I widget più conosciuti fanno parte di famiglie specifiche: Athena widget, scaturita dallo stesso Project Athena, Athena 3D, con un look piu' moderno dello spartano predecessore, Motif, molto simile esteticamente a Windows 3.x (e come Windows, utilizabile solo a seguito del pagamento della licenza d'uso. . .) e Lesstif, un clone free di Motif.

Utilizzando uno qualunque dei set sopraelencati è possibile creare e personalizzare qualsiasi tipo si interfaccia, evitando di preoccuparsi della compatibilità e della portabilità delle applicazioni risultanti, che potranno essere visualizzate sulla stragrande maggioranza degli UNIX.

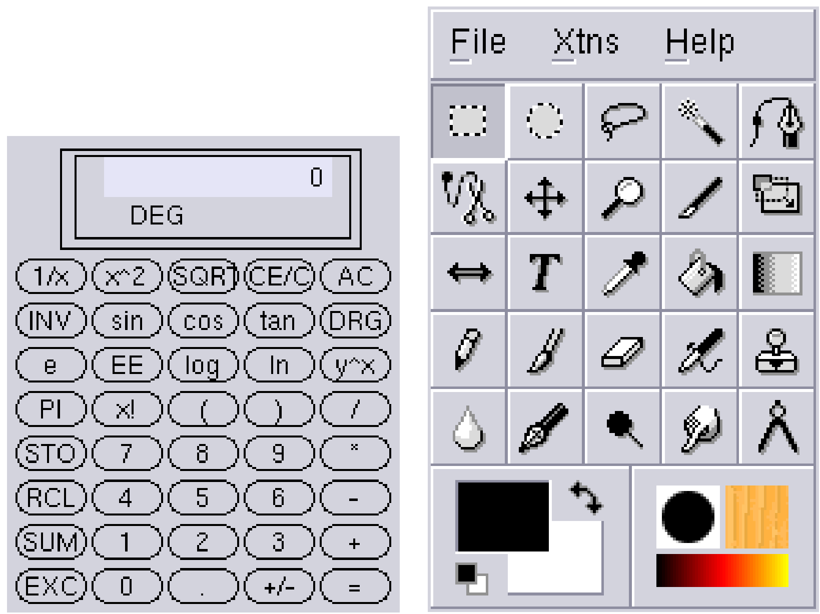
I widget sono d'altra parte sempre accompagnati da una serie di toolkit, ovvero da una serie di *attrezzi* che vengono utilizzati per la gestione complessiva dei widget. Il più comune toolkit è X Toolkit, lo trovate con lo stesso X Window System. Motif, inoltre, aggiunge ai suoi widget un toolkit, come molti altri hanno fatto.

---

[8] Esistono, casualmente, due oggetti che effettivamente si chiamano Widget. Il primo è un oggetto molto, molto importante, ovvero cio' che viene inserito nelle lattine di Guinness Stout e che rilascia, all'apertura della lattina stessa, l'azoto che aromatizza la mia birra preferita. Il secondo oggetto chiamato realmente widget è quel piccolo affarino in cui sono inserite le lamette del vostro rasoio ("Think Unix", by John Lasser, QUE Publishing, 2000, pag. 204)

Uno dei nuovi arrivati è GTK, il toolkit che accompagna il fantastico programma di manipolazione delle immagini *The GIMP*.

GTK va oltre il semplice toolkit e vi permette di creare anche temi personalizzati, oltre a quelli che imitano, ad esempio, Motif, MacOS oppure BeOS.



`xcalc` `gimp`

Figure 9.2 *Finestre realizzate con due toolkit differenti:* `xcalc`*, a sinistra, usa il Toolkit Athena, The GIMP, a destra, usa il Toolkit Gtk*

9.6 Window Manager e Desktop Environment

Ora che avete compreso come le applicazioni, o meglio, le loro interfacce, vengono create e gestite complessivamente nell'ambiente X, facciamo un passo più in là. Forse il concetto che introdurrò destabilizzerà drasticamente alcuni dei vostri punti fermi ma sappiate che le applicazioni grafiche non rappresentano l'intero panorama di ciò che può essere rappresentato e visulizzato graficamente. Sarò più chiaro: come pensate che possano essere gestite le finestre all'interno del vostro schermo? Quello che comunemente viene chiamato, soprattutto in ambiente Windows, Desktop è una finestra? Com'è possibile aprire, spostare o chiudere una finestra?

Widget e toolkit, l' X server e l'intero sistema X Window, da soli, non possono fornirvi un ambiente di lavoro così completo. La gestione della posizione delle finestre all'interno del vostro schermo, le impostazioni relative al desktop (no, non è una finestra), la barra degli strumenti, le icone e via dicendo sono affidate ad un programma (o ad una serie di programmi) chiamati Window Manager, gestori di finestre, appunto.

Un sistema grafico completo è composto, in sintesi, da un X server, da un window manager, da una serie di programmi dotati di interfacce grafiche.

Se le interfacce dei programmi variano da applicazione ad applicazione, i window manager variano a seconda della vostra personale scelta. Potrete anche sbizzarrirvi assegnando ad ogni utente un window manager diverso e appezzerete sicuramente alcune peculiarità assenti in ambiente Windows Mac, come la possibilità di sfruttare e utilizzare uno o più *virtual desktop*.

Un virtual desktop vi permette di dividere e mantenere piu' schermi aperti comtemporaneamente, visualizzandoli uno per volta e permettendovi di spostarvi da uno all'altro semplicemente spostando il mouse *fuori* dallo schermo fisico verso un altro virtual desktop[9].

I window manager sono decine e non vale la pena di analizzarli uno per uno. Tutti quanti sono estremamente personalizzabili e tra di loro le diversità maggiori riguardano il consumo di risorse, in particolare la memoria RAM. Si possono citare, per esempio, i classici e standard *twm*, Tab Window Manager, *mwm*, il window manager di Motif, *fvwm*, Fantasmagoric[11] Virtual Window Manager, basato su twm, leggerissimo e configurabile più che mai. Ancora, *window maker*, simile al window manager di NextStep/OpenStep e *enlightenment*, molto potente, forse solo un po' troppo pesante...

[9] La suddivisione dello schermo fisico in piu' schermi virtuali o virtual desktop non va confusa con la possibilità di visualizzare più finestre sovrapposte contemporaneamente. Dovete pensare alla possibilità di poter disporre, virtualmente, di 2, 4, 9 schermi differenti, ognuno con le sue icone, il suo background[10] e le sue finestre aperte!

[10] Ogni background sara' diverso ovviamente !

[11] In realtà il significato esatto della *f* di fvwm si è perso nel più profondo mistero...

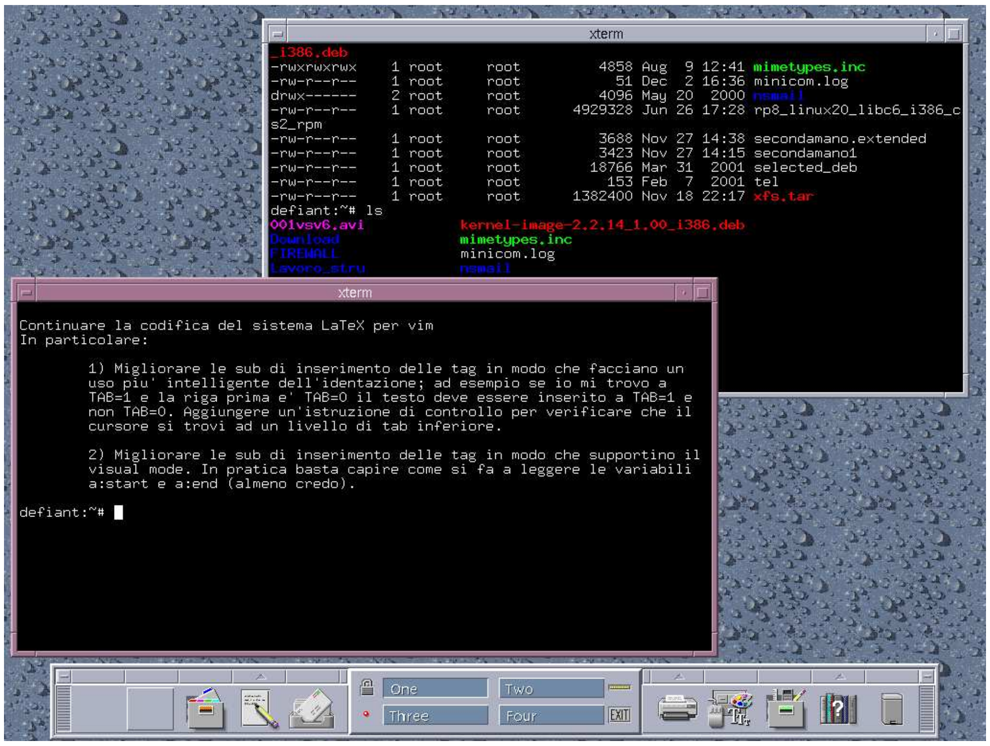


Figure 9.3 CDE con il Motif Window Manager

Tab's Window Manager

Motif Window Manager

Figure 9.4 Esempi di due differenti window manager

Non pensiate però che gli strumenti forniti da UNIX per la creazione di un ambiente grafico **veramente** completo siano tutti qui!!

Non vorreste disporre di un ulteriore applicazione che vi permetta di integrare in modo pressoché perfetto programmi, interfacce, window manager, immagini di backgroud e i vostri temi preferiti?

Potrete allora utilizzare uno dei tanti *Desktop Environment*, ormai molto popolari. In ambiente Windows o Mac vi sembreranno soluzioni scontate, questo solo perché vi

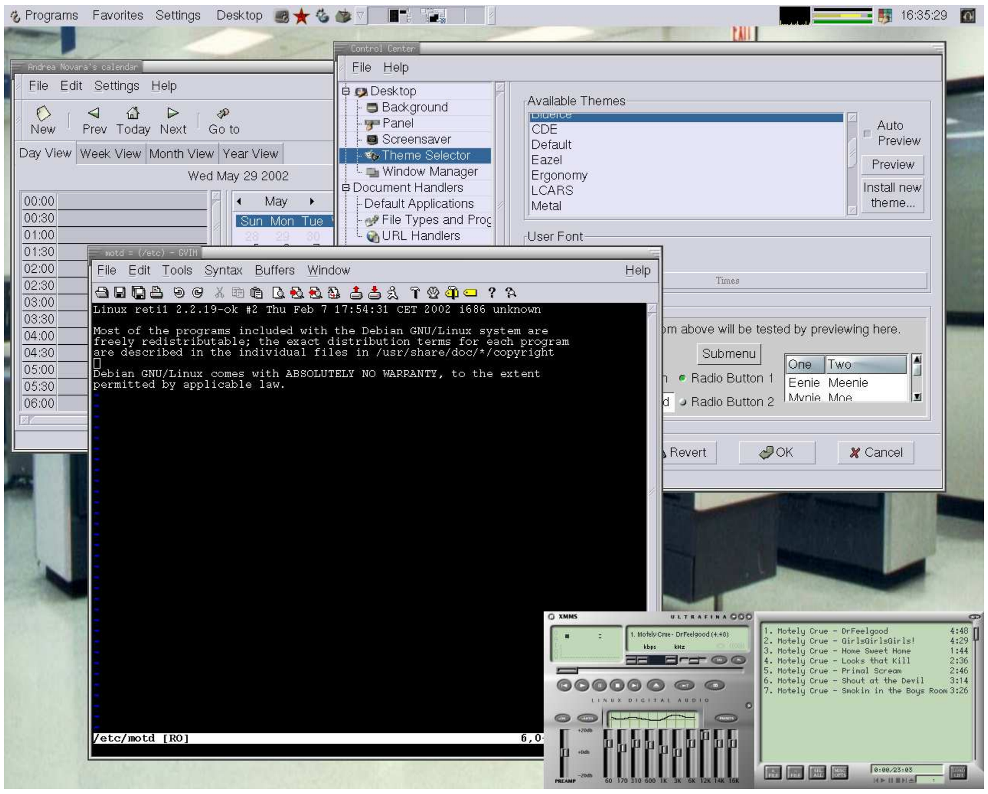


Figure 9.5 Un desktop basato su Gnome e Sawfish

vengono fornite forzatamente, senza possibilità di farne a meno. Invece il sistema X Window può tranquillamente lavorare egregiamente anche senza il loro ausilio[12].

In effetti molti utenti si accontentano di un ambiente grafico senza troppi orpelli e fronzoli, sgombro da icone e applet pressoché inutili e parco di memoria RAM. Io sono uno di questi utenti. Chi invece non fosse preoccupato dal dispendio di risorse[13] e volesse immergersi in un'esperienza molto simile a quella che si ha accedendo ad un Mac, potrà scegliere tra *CDE*, *GNOME* o *KDE* e configurare a proprio piacimento il suo desktop.

12 Potreste anche fare a meno del window manager, a dire il vero, ma perdereste la praticità di finestre che si spostano e si ridimensionano a vostro piacere

13 32 Mb di RAM sono più che sufficienti

10. Post Scripta Manent...

di Tx0

Come si stampa sotto UNIX – Quali sono i comandi per stampare, verificare la "coda" di stampa, cancellare una stampa in attesa.

10.1 Formati di stampa

Esistono numerosi formati con i quali è possibile specificare i contenuti e la forma di un documento. Il più semplice di questi è sicuramente il formato ASCII, ossia il testo piano, nudo e crudo senza l'aggiunta di alcuna formattazione.

Esiste poi il formato *PostScript*, creato da Adobe, è un file di testo contenente i comandi di impaginazione e posizionamento dei contenuti, che possono poi essere interpretati da una stampante abilitata o essere filtrati per una stampante non abilitata al riconoscimento.

Il PostScript ha un cugino più recente che e' il *Portable Document Format* (o pdf per gli amici...) che consente una migliore consultazione a video con possibilità di indicizzazione e rimando ipertestuale dei contenuti. È quindi più adatto alla distribuzione di testi per consulazione diretta senza però perdere i vantaggi in fase di stampa che il PostScript garantisce.

Esistono i formati grafici pittorici (e qui l'elenco è infinito...), solo per citarne alcuni abbiamo:

- GIF
- JPEG
- PNG
- XPM

Non è quindi sufficiente pensare *¡¡Devo stampare un file¿¿*; bisogna porsi sempre la domanda *¡¡A quale tipo di formati appartiene il file che devo stampare?¿¿*. In realtà poi vedremo come i filtri di stampa sono in grado di rispondere a questa domanda per conto nostro e ci consentono di pensare solo *¡¡Devo stampare un file¿¿*.

10.2 Capolino dietro le quinte

La stampa sotto UNIX non è un sistema banale. Tuttavia come vedremo questa complessità è ampiamente giustificata dalla flessibilità che garantisce e in realtà è ha diretto contatto solo dell'Amministratore di Sistema, mentre l'Utente vede solo l'interfaccia più esterna.

Il meccanismo di stampa funziona così:

File dell'utente da stampare
↓
Comando di stampa (`lpr` o `lp`)
↓
Demone di stampa (`lpd`)
↓
Filtri di stampa (*magicfilter*)
↓ ↓
Stampante Locale — Demone Remoto
↓
Filtri di stampa remoti
↓
Stampante Remota

Figure 10.1 Il meccanismo di stampa sotto UNIX

I comandi di stampa variano da un flavour UNIX all'altro. Il comando System V è `lp` mentre quello BSD è `lpr`. Entrambi sono contrazioni di *Line PRinter*. Il comando riceve il file e lo passa al demone. Questo demone (ossia fornitore di servizio che svolazza in background nel computer) ricava dalla configurazione di sistema o dai parametri passati dal comando di stampa su quale stampante il file vada riprodotto. In base a questa decisione il demone sceglie anche il filtro opportuno da applicare[1]. Processa il file attraverso il filtro e lo recupera filtrato[2].

Con in mano il file pronto per la stampa, il demone decide se inviare il file su una porta parallela, ad un altro server o magari alla stampante via rete, se questa ha una interfaccia di rete. Nel caso in cui la stampante sia collegata ad un server remoto, il demone locale conttatta il demone remoto, il quale ottiene il file e riparte localmente con tutto il procedimento di riconoscimento del formato e della stampante, e quindi con il filtraggio del file.

> È molto raro che venga configurato un filtro in locale quando poi la stampante è attaccata ad un server remoto. Se l'Amministratore del server remoto decidesse di sostituire la stampante e modificare i filtri senza avvertire, ci si troverebbe in una situazione di incomunicabilità e di impossibilità alla stampa.

Perché tanto odio? In realtà di odio ce n'è proprio poco. Il meccanismo è lineare pur nella sua articolazione. Vediamo i passaggi.

Un utente ha un file da stampare (`articolo.ps`). L'estensione ci dice che è un file *PostScript*. Diciamo che il nostro fulgido Amministratore di Sistema ha configurato un filtro di stampa che riconosce automaticamente il file e lo stampa interpretandolo con *ghostscript*. Il file arriva alla stampante interpretato e la stampante lo inizia a riprodurre su carta.

10.3 Cioè per stampare?

Il comando di stampa varia da flavour a flavour. Linux adotta il sistema BSD (anche per le distribuzioni System V). Quindi il comando di stampa e' `lpr`. La sintassi più elementare è la seguente:

---

1 I filtri dipendono non solo dal formato del file ma anche dalla stampante. Se, poniamo, ho una stampante laser che interpreta il PostScript e una a getto di inchiostro che non lo fa, il filtro per la prima passa alla stampante i file PostScript senza filtrarli, in modo da alleggerire il sistema, mentre il secondo deve usare *ghostscript*, ossia un programma, per interpretare il file in favore di una stampante che non è progettata per svolgere questo compito

2 Ok, abbiamo imbrogliato. Dallo schema sembra che sia il filtro a mandare in stampa l'output, ma questo non è possibile dato che solo il demone sa se la stampante è locale, e nel caso a quale porta è collegata, o se è remota e nel caso a quale server inoltrare la richiesta

```
$ lpr mybook.ps
$
```

UNIX come al solito quando tutto funziona non ci dice nulla, quindi dobbiamo presumere che il nostro job di stampa sia in coda. Per verificarlo basta usare il comando `lpq` che stà ovviamente per *line printer queue*.

> Ogni job di stampa ha un suo numero identificativo. Questo numero viene scelto secondo criteri differenti fra diversi UNIX. Ad esempio sotto Linux è un numero progressivo, Solaris è più "*fantasioso*" nella scelta di questo numero. In ogni caso `lpq` risolverà ogni dubbio.

Vediamo come funziona `lpq`:

```
$ lpr mybook.ps
$ lpq
Rank   Owner      Job  Files                          Total Size
1st    tx0        2    mybook.ps                      11824 bytes
$
```

In questo caso abbiamo un solo job di stampa, di proprietà dell'utente tx0, il job di stampa è numerato con un 2, proviene dal file `mybook.ps` grande 11824 bytes.

10.4 Ok, ok, e mettiamo che sbagli?

Abbiamo visto alla sezione precedente come si possa recuperare l'identificativo numerico di un job di stampa usando il comando `lpq`. Saputo questo possiamo rimuovere il nostro job di stampa dalla coda prima che sia stampato nel caso ci accorgessimo di avere stampato il file sbagliato o che altro... il comando da usare per questo è `lprm`[3].

Il funzionamento è semplice:

[3] ossia? indovinato vero? *line printer remove*

Ok, ok, e mettiamo che sbagli?


```
$ lprm 2
dfA002defiant dequeued
cfA002defiant dequeued
$
```

Il comando ci informa in questo modo che il nostro job di stampa è stato cancellato. I due file riportati *sono* il nostro job di stampa[4].

[4] per ogni job di stampa UNIX crea un file di contenuto che poi è il documento che avete mandato in stampa, dei due file è quello che inizia per **df** ossia *document file*, e un file di controllo, che inizia per **cf**, *control file* appunto, che contiene le informazioni relative all'utente che ha stampato il job e alla priorità e così via...

11. Conclusioni, ringraziamenti, riconoscimenti, speranze, anatemi ed altri sollazzi di fine libro

di tutti

Prima di tutto e di tutti, ringrazio shodan :-) di non aver scritto la parte sulle reti, sennò non la scrivevo io. I miei più (a)cari saluti a Tx0, sempre pieno di risorse, sempre pieno di consigli... peccato che usi `vim`, ma è perdonabile :-)

Non posso non citare poi jaromil, C1cc10, e BOmboclat, con cui portiamo avanti quella distro magnifica comunemente chiamata dynebolic.

Saluti obbligati al LoA HackLab Milano :-)

Last but not least vorrei anche ringraziare il latex, per tutte le cose belle che mi fa fare, ed il LaTeX perché è un bel linguaggio.

Many thanks to: zsh, ls, more – less, emacs, piCo, nano, mIcro, femto, fsf, gnu, linux, As, ar, at, ld, mv, rm, -rf, /, Outs, \, awk, biew, fOr, ftp, kill, killall, slay, neTstaT, hexdump, strace, ltrace, gdb, gdm, gpm, gpl, gpg, pgp, dasm, perl, fravia, tex, tetex, tetEtex, latex, lacheck, wipe, srm, jobs, gpic, gprof, gproc, make, menu, config, bc, bg, dc, cc, cp, ci, co, cd, dD, du, do, df . . . and many many more.

— all your books are belong to us

Little John

Vorrei ringraziare, in quest'ordine:

Tx0, per la pazienza con cui ha sopportato due anni di lungaggini e ritardi!

Shah, per il supporto morale instancabile e per la tastiera con cui scrivo :-)

Donald Knuth per aver scritto TeX e Laslie Lamport per averlo condensato in LaTeX.

Manhattan

C11 Conclusioni, ringraziamenti, riconoscimenti, speranze, anatemi ed altri sollazzi di fine libro

Ringrazio chi ha gentilmente offerto le macchine su cui lavorare a questo testo e che mi hanno permesso di imparare tutto ciò che so sul magico mondo di Unix. Ringrazio inoltre il Signore nostro Iddio, per aver fornito così spesso un comodo bersaglio per tutti gli improperi e insulti pronunciati e per aver fatto da capro espiatorio quando le cose andavano male. amen.

Shodan

Spero che questo libro possa essere uno strumento utile a chi si accosta a UNIX per la prima volta senza conoscerne alcunché. UNIX non è un sistema operativo come gli altri. È invece una vera e propria lingua. Un modo di ragionare alla soluzione di problemi di varia natura radicalmente differente rispetto ai paradigmi grafici che l'industria del software ci sta imponendo. È uno stimolo continuo a pensare ed a conoscere sempre di più. A inventare forme alternative di procedere verso lo stesso obiettivo. È un sistema operativo hacker, creato da hacker (anche se magari non si facevano chiamare così) per essere usato da tutti, ma con grande gioia soprattutto dagli hacker. `:-)`

Spero che la scelta di scriverlo in TEX non sia di intralcio per alcuno, ma anzi consenta una futura espansione da parte di chiunque.

Un primo ringraziamento lo devo alla mia famiglia che, pur non vedendomi mai, mi dà ancora un letto nel quale dormire.

Poi a Linus per aver creato quel kernel che ci sconvolse la vita, a Richard per il suo modo utopistico ma anche pragmatico di vedere l'informatica e ovviamente per tutto l'ottimo software che FSF ci ha dato, a Donald per questo sistema di scrittura di testi che ti trascina nella più bieca aberrazione quando qualcosa non funziona, a Dennis e Ken per l'invenzione del sistema operativo più bello, a Brian e di nuovo a Dennis per il C e il loro libro, e a Larry perché ha reso tutto più facile.

Un ringraziamento ancora a tutte le compagne ed i compagni del Deposito Bulk di Milano, per l'affetto e la simpatia, per le lezioni di vita, per le serate e la birra. Per tutte le lotte fatte insieme. Ma soprattutto per quanto mi hanno fatto cambiare e crescere. Un ringraziamento particolare a Martina per essersi messa in testa di scrivere una *Magna Premessa* a questo libro, anche se non ha ancora scritto nulla.

Un ringraziamento speciale al LOA HackLab di Milano e a tutti gli acari e le acare che ne fanno parte, per avermi rubato così tanto tempo, rendendomi vivo.

A Shodan amico di sempre ed estenuante coautore di questo testo; a bomboclat (sì, proprio 'bo' di dynebolic) e a c1cc10 (sì, proprio 'c1' di dynebolic(1)) per il sostegno e le notti davanti al computer; a blicero per un elenco di cose per il quale forse pubblicherò un testo a parte; a shah e manhattan per l'amicizia e il couscous, ed a echomirage (*photoshrek*) per il debugging angosciante; a lidl, +mala e megabug per i pensieri rasenti il delirio, le idee, il reversing, tutto il resto; a davidone per le lezioni di TEX e di cinese; a odo perché non c'è una cosa che non sappia (`http://www.odoogle.org/`); a liw perché non c'è paura; a Zeist perché anche spararle grosse è un'arte `:-)` , a pinq "non può essere tutto rose e chiodi"; a giusman per l'incoraggiamento e le discussioni; a Belgarath per le lezioni di Perl sprecate e per averci dato la possibilità di giocare a Quake dentro il Bulk; a putro, per lo stile naturalmente; a giucas, perché finalmente il lunedì ha un senso; a manu, germin e zurg che ha a che spartire con questo libro qualcosa che non sa.

E infine al giro hackmeeting. A mag-one, fin dai giorni di hackit'99, instancabile bevitore di birra; a Elettrico e ginox, fieri responsabili della succursale torinese :-) ; a qu3st ed asbesto per radio cybernet, i computer di legno e mille altre cose; a gipoco, provvidenziale detendore di un PC con fornetto a resistenza scalda brioche; a Cojote, IlNonSubire e Mille (mezz'euro) per la loro assurda curva di apprendimento informatico; a jaromil per muse e la scimmia dello streaming, maledetto fricchettone; a lobo, per diskofrigido, Mufhd0 e le innumerevoli minacce di morte del primogenito.

A loro e a tutti quelli di cui mi sono sicuramente dimenticato. . . grazie!
19 a tutti!

Tx0

GNU Free Documentation License

— Version 1.1, March 2000 —

0. *PREAMBLE*

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. *APPLICABILITY AND DEFINITIONS*

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five). C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified

Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. *COMBINING DOCUMENTS*

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. *COLLECTIONS OF DOCUMENTS*

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. *AGGREGATION WITH INDEPENDENT WORKS*

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. *TRANSLATION*

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. *TERMINATION*

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. *FUTURE REVISIONS OF THIS LICENSE*

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Per una traduzione non ufficiale in italiano consulta

http://www.softwarelibero.it/gnudoc/fdl.it.html

Nota sul CD-ROM allegato

Il mini-cd che alleghiamo a questo testo contiene il testo del libro che avete in mano in formato digitale, sia PostScript, più adatto alla stampa, che PDF, più adatto alla consultazione on-line.

Per espandere ulteriormente le vostre conoscenze su UNIX e su Linux più in dettaglio, abbiamo incluso nel CD anche una scelta dei titoli del *Linux Documentation Project* , un progetto sviluppato in rete che si prefigge lo scopo di scrivere un set completo di manuali per Linux.

Trovate inoltre anche il sorgente completo, i file di immagine e tutto il materiale che noi abbiamo utilizzato per produrre questo Corso. Insieme ai sorgenti c'è anche la documentazione su TEX e su ConTEXt che siamo riusciti a reperire in rete.

Ricordate comunque di controllare sul sito del Corso l'uscita di una nuova versione. Noi vi incoraggiamo a spedirci qualsiasi materiale testuale o anche solo concettuale voi riteniate possa essere vantaggioso includere nel testo e vi ringraziamo in anticipo per tutti gli errori che vorrete segnalarci per migliorare questo testo open source.