Lidl's – Manhattan's – Shodan's – Tx0's

Corso Base di UNIX

LOA HackLab MI

DISCLAIMER

QUESTO CORSO DI UNIX VIENE PENSATO, SCRITTO E STAMPATO AD USO E CONSUMO DI CHIUNQUE VOGLIA IMPARARE L'USO DEI SISTEMI OPERATIVI UNIX. GLI AUTORI SI RISERVANO IL DIRITTO DI MODIFICARE IL TESTO QUANDO E COME VOGLIONO PER AGGIORNARE OD ESPANDERE IL CONTENUTO. È CONSENTITA LA STAMPA LIBERA DI QUESTO MANUALE IL CUI UNICO SCOPO È QUELLO DI ESPANDERE LA CONOSCENZA DEI SISTEMI OPERATIVI UNIX. NON È CONSENTITA LA STAMPA A SCOPO DI LUCRO, LA RIVENDITA A QUALSIASI CIFRA DEL MANUALE IN FORMATO DIGITALE. NON È CONSENTITO INOLTRE IL PASSAGGIO A TERZI DI QUESTO TESTO QUALORA SPROVVISTO DEI RICONOSCIMENTI AGLI AUTORI ED AL LOA HACKLAB DI MILANO E DELLE INFORMAZIONI SULLA REPERIBILITA' DEL MANUALE AGGIORNATO IN RETE.

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.

Attenzione: Questo libro è attualmente in scrittura. La versione che state leggendo è temporanea e *altamente incompleta!* Non riportate come bug le mancanze di capitoli *(soprattutto quelli marcati esplicitamente come incompleti)*. Riportate invece errori di concetto, di scrittura, refusi di stampa e evidenti tracce di sostanze estranee nella mente degli autori.
Insomma questa versione è solo per ricordarvi quello che avete ascoltato durante il corso. Verso la fine di gennaio *dovrebbe*[1] essere pronta la versione completa e definitiva. Monitorate costantemente il sito per nuove versioni di questo libro. Il testo è disponibile in `pdf`, `PostScript` o direttamente nel mitico, magico, splendido LaTeX.

Web LOA: *http://loa.hacklab.it/*
Web Dispense Corso: *http://www.ecn.org/loa/shodan/*

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

Versione: 0.9
Ricompilato in data 22 febbraio 2002

[1] Prima di inveire ricordate che lo facciamo aggratiz

Le convenzioni adottate in questo libro

Troverete nel corso del libro alcuni segnali che vi aiuteranno a comprendere il testo e a fissare i contenuti nella memoria. Questi segnali si riferiscono a:

annotazioni esplicative

segnalazioni di comandi potenzialmente pericolosi

suggerimenti da adottare

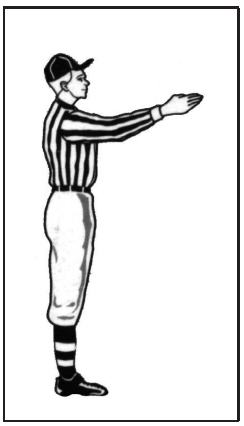
Incontrerete inoltre speciali riquadri che contengono codice, screenshot di videate di programmi come `vi` o output di comandi come `ls`. Eccovi un esempio:

```
$ ls -l
total 1428
-rw-r--r--    1 andrea   andrea        499 Oct 20 00:47 corsounix.aux
-rw-r--r--    1 andrea   andrea     261528 Oct 20 00:47 corsounix.dvi
-rw-r--r--    1 andrea   andrea      16634 Oct 20 00:47 corsounix.log
-rw-r--r--    1 andrea   andrea     231706 Oct 20 00:35 corsounix.pdf
-rw-r--r--    1 andrea   andrea     369184 Oct 20 00:40 corsounix.ps
-rw-r--r--    1 andrea   andrea       3617 Oct 20 00:45 corsounix.tex
-rw-r--r--    1 andrea   andrea       6653 Oct 20 00:47 corsounix.toc
-rw-r--r--    1 andrea   andrea     171868 Dec 16  2000 corsounix.txt
$
```

Capito??

Capitolo 1

Storia di UNIX



□ *Sembra incredibile, ma anche UNIX non esiste da sempre…*

Autore: *Tx0 <tx0autistici.org>*

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 versioni a venire. 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]

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 af-

[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!

[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 :-)

fatto 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:

1. Nuove chiamate al Kernel[4]

2. Nuova concezione dei processi di boot[5], configurazione dei servizi, gestione di account, dischi e
   dispositivi

3. Nuovi comandi

In dettaglio questo comporta:

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

2. 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.

3. 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.

[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

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

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 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.

E' un serio problema 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

[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

[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

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 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 da 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

[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

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

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: *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 della 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`.

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

[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

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:

- 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!

Capitolo 2

Sezione generica UNIX • Shodan

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.

2.1.1 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.

2.1.2 Unix è multitasking

Una traduzione letterale della parola *multitasking* potrebbe essere *multiobbiettivo*. Questo termine si applica ad un sistema operativo che e' 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.

2.1.3 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.

Capitolo 3

Entriamo nel sistema – INCOMPLETO – Shodan

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.

3.1.1 Consigli per una password sicura

La password più complicata è più difficile è 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 e' 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.

3.3 Comandi di Base

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

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

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
%
```

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.

3.3.2 Dove sono finito: *hostname, uname*

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[1].

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
%
```

Come si può notare questa informazione comunica solo il nome del host, non quello del *dominio*[2], 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?)[3]
- Nome del computer (come con *hostname*
- Versione del S/O o del kernel (a volte con data di creazione)

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

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

[3] 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

- 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[4] 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:

```
      % uname -a
Solaris: SunOS copkiller 5.5.1 Generic_103640-29 sun4u sparc SUNW,Ultra-1
      %
       % uname -a
e HP/UX 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[5]; le notizie qui iniziano ad essere diverse anche di posizione: con Solaris[6] 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[7] 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[8] 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.

[4] Con "x86" si intendono tutti i processori Intel e compatibili

[5] 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

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

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

[8] *Compilato*=tradotto da codice scritto dall'uomo a codice eseguibile dal computer

3.3.3 C'è nessuno?: *who, w*

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:

```
% 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[9], 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[10], nella seconda una 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

[9] 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.

[10] 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.

processi e relativi “figli” in quel terminale, la quantità di tempo macchina dei processi attivi e cosa sta facendo l’utente sulla data tty.

3.3.4 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 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 puo’ 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
%
```

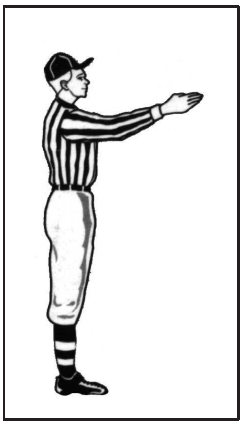
Si può notare che i file nascosti sono quelli il cui nome inizia con un “.”; la funzione dei file **.** e **..** sarà spiegata in seguito.

3.3.5 I permessi e la proprietà dei file: *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
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: `chmod 654 nomefile`

Capitolo 4

Puntiamo più in alto



□ *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*

Autore: *Tx0 <tx0autistici.org>*

4.1 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 `$SHELL`). 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*[a]
-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**).
-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		

-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*

[a] Il *pattern* utilizza i metacaratteri della shell per rappresentare il nome completo del 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.[1]

```
$ 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![2]

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

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

[2] A dispetto della combinazione, questi file saranno ben poco segreti! ;-)

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.

```
$ 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; `~/directory/dir2` ci consente il tentativo, ma per ciascun file o directory contenuto genera un errore; `~/directory/dir3` 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).

4.2 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.[3]

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

Il più comune programma per la creazione di archivi sotto Unix è `tar`.[4] 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*
-g	Comprime l'archivio usando `gzip`
-b	Comprime l'archivio usando `bzip2`
-Z	Comprime l'archivio usando `compress`

Facciamo alcuni esempi:

[4] 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

```
$ 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".[5] 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:

```
$ 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:

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

```
$ 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 `tar uf works.tar works/` 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 `tar xf works.tar`. 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.

4.3 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[6], 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:

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

```
$ 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
$
```

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.

4.4 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` 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.

4.5 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
P@sky   02/987435213321
Bluca   02/0xa65db78c86
pbm     02/76453129982
Zeist   01337/34
$
$ sort tel_num
Bluca   02/0xa65db78c86
P@sky   02/987435213321
Shodan  02/753207213
Tx0     02/7412309676
Zeist   01337/34
pbm     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
P@sky   02/987435213321
pbm     02/76453129982
Shodan  02/753207213
Tx0     02/7412309676
Zeist   01337/34
$
```

Et voilá. Questa volta *pbm* 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! ;-)

4.6 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 ".*tar\.gz"`. 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`.[7]

```
$ 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
$
```

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`:

[7] 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

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

4.7 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.

4.7.1 `kill`

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.

4.7.2 foreground o background?

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[8] 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`.[9]

[8] Solo quelle più moderne, ossia più recenti del 1990

[9] 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

4.7.3 `top`

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*[10] 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:

```
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.[11] 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[12] e nessuno è stato bloccato.[13] 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*[14] fornendo nell'ordine i valori di:

[10] Lista dei processi attivi sulla macchina
[11] Rispettivamente carico minimo, medio e massimo
[12] Un processo zombie è un processo che è morto senza che il sistema si riuscito a rimuoverlo dalla process table
[13] Come dopo la pressione di ˆZ
[14] Memoria Virtuale: spazio su disco utilizzato *come* memoria RAM per sopperire a carenze di quest'ultima

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:[15]

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:

[15] Le colonne ignorate sono obsolete

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.

4.7.4 Priorità e *nice*

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.

4.7.5 ps

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.[16] Cerchiamo di mettere in evidenza le dipendenze dei processi (ovvero quali processi siano *genitori* di quali processi); usiamo per questo l'opzione `-H`:

[16] 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`;[17] Più in basso notiamo come stiano girando due shell `bash` che hanno rispettivamente lanciato l'editor di testo `vim` e il comando `ps`.[18]

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`:

[17] `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

[18] 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

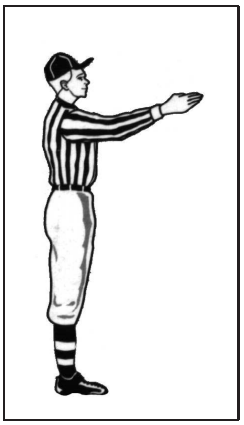
```
$ ps -AH o user,pid,pcpu,rss,cmd
USER       PID %CPU  RSS CMD
root         1  0.0   64 init [2]
root         2  0.0    0   [kflushd]
root         3  0.0    0   [kupdate]
root         4  0.0    0   [kpiod]
root         5  0.0    0   [kswapd]
daemon      84  0.0    0   [portmap]
root       152  0.0  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.

Capitolo 5

La Shell • INCOMPLETO • Vedi Sezioni



□ *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 perché 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*

Autore: *Tx0 <tx0autistici.org> e Shodan <shodan@autistici.org>*

5.1 Differenti shell

La shell è un programma a tutti gli effetti quindi 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.

5.1.1 Elementi di una shell

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

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

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.

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*.

5.2 `sh` e derivate

5.2.1 Il Prompt

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:

`Pagina conclusa artificialmente ...`

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@defian:/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

$
```

5.2.2 Variabili notevoli

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.

5.2.3 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].

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

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.

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

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:

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

5.2.5 Alias

Gli *alias* consentono di dare nuovi nomi ai comandi più complessi o più lunghi da scrivere. La sintassi è:

[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

```
$ alias ll='ls -la'
$ ll
total 8088
drwsrwxrwx    7 andrea   andrea       4096 Dec 15 01:44 .
drwsr-xr-x   10 andrea   andrea       4096 Dec  9 01:50 ..
-rw-r--r--    1 andrea   andrea        523 Dec 15 01:45 corsounix.aux
-rw-r--r--    1 andrea   andrea      30916 Dec 15 01:45 corsounix.log
-rw-r--r--    1 andrea   andrea       1543 Dec 10 11:26 corsounix.tex
-rw-r--r--    1 andrea   andrea       9537 Dec 15 01:45 corsounix.toc
$ alias ls="ls -s"
$ ll
total 8088
   4 drwsrwxrwx    7 andrea   andrea       4096 Dec 15 01:44 .
   4 drwsr-xr-x   10 andrea   andrea       4096 Dec  9 01:50 ..
   4 -rw-r--r--    1 andrea   andrea        523 Dec 15 01:45 corsounix.aux
  32 -rw-r--r--    1 andrea   andrea      30916 Dec 15 01:45 corsounix.log
   4 -rw-r--r--    1 andrea   andrea       1543 Dec 10 11:26 corsounix.tex
  12 -rw-r--r--    1 andrea   andrea       9537 Dec 15 01:45 corsounix.toc
$ ls
total 7656
   4 corsounix.aux
  32 corsounix.log
   4 corsounix.tex
  12 corsounix.toc
$
```

Da questo semplice esempio possiamo già notare le due caratteristiche più interessanti degli alias:

1. **Gli alias vengono interpretati ad ogni chiamata.** Dopo la definizione, `ll` viene espanso in `ls -la`. Dopo la successiva definizione dell'alias `ls`, `ll` viene reinterpretato come `ls -las`, senza che `ll` stesso sia stato ridefinito.

2. **Gli alias mascherano i comandi di sistema.** Dopo la definizione dell'alias `ls`, il comando di sistema `/bin/ls` non viene più chiamato direttamente ma sempre attravarso l'alias. Per evitare l'interpretazione dell'alias si può eliminare l'alias con il comando `unalias ls`, ma questo richiederebbe la successiva ridefinizione dell'alias, oppure la chiamata del comando con il suo percorso assoluto, `/bin/ls`, oppure con un carattere di commento, `\ls`.

Il comando `alias` senza argomenti mostra l'elenco degli alias attivi nella shell:

```
$ alias
alias l='ls -l'
alias la='ls -A'
alias ll='ls -la'
alias ls='ls --color=auto '
$
```

5.2.6 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`.

Pagina conclusa artificialmente ...

5.3 csh e derivate • INCOMPLETO • Shodan

5.3.1 Il Prompt

DA SCRIVERE!!!

5.3.2 Variabili notevoli

DA SCRIVERE!!!

5.3.3 Input e Output

DA SCRIVERE!!!

5.3.4 Controllo dei processi

DA SCRIVERE!!!

5.3.5 Sintassi di programmazione

DA SCRIVERE!!!

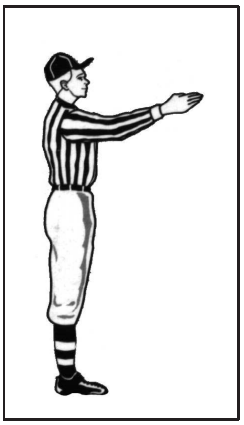
5.3.6 I file di configurazione

DA SCRIVERE!!!

INCOMPLETO!!!!!!

Capitolo 6

Regular Expressions



□ *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.*

Autore: *Tx0 <tx0autistici.org>*

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?

6.1.1 Due convenzioni, molti meno problemi

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. `/regexpr/`). 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:

[1] Programmi come HomeSite, supportano le Regular Expression

```
/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]

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

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

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, 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,}

[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.)

[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

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:

```
/*/ <- Errata!
```

(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?

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**.

[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.

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.

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]

  Attenzione: 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

[6] Anche se questo presto finirà.

`/[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]*/
  ```

  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

[7] Non temete: *non è il caos.* Esiste un modo per fare verifiche su sequenze di caratteri in ordine determinato. Stiamo per arrivarci.

[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.

- 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 *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.

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

[9] Abbiamo numerato i caratteri per una più rapida individuazione

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:

$"_1"_2"_3$

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

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

- 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]

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

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

[12] Ah! Dimenticavamo: i commenti negli script di shell si indicano con un # come primo carattere

[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

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.

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.

[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.

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!

6.11.1 Eliminare i commenti di uno script

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

6.11.2 Sostituzione multipla

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

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

6.11.3 Sostituzione multipla con uso delle posizioni

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:

```
/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]

Opzione	Mnemonic	Comportamento
/i	**I**gnore case minuscole	No dà attenzione al fatto che le lettere siano maiuscole o
/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 nuovi match	Riuscito il primo tentativo non abbandona la stringa, ma tenta

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
```

[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

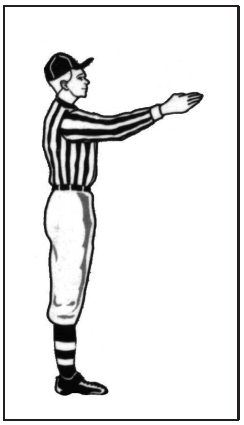
[17] Esistono altre opzioni, ma la loro applicazione è troppo complessa e specialistica perché possa rientrare tra gli scopi del nostro corso

la quale porterebbe alla stringa finale “def def def”.

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

Capitolo 7

Editor di testo – INCOMPLETO – vedi sezioni



□ *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.*

Autore: *Tx0 <tx0@autistici.org> e Shodan <shodan@autistici.org>*

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.

7.1.1 Una personalità schizofrenica

`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
```

[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

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

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

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 (`:`).

7.1.2 Inserire del testo

`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
~
~
~
```

7.1.3 Muoversi attraverso il testo

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 ← e 1 ↑.

7.1.4 Salvare ed uscire

`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`.

7.1.5 Cancellare, copiare, modificare e sostituire

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)

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

- 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

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:

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

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

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!

7.1.6 Anche gli Utenti UNIX possono sbagliare

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".

7.1.7 Diversi modi per entrare in *insert mode*

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

Comando	Inizio[a]	Direzione[b]	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

[a] La posizione dalla quale parte l'inserimento, rispetto a quella attuale
[b] Come prosegue l'inserimento dopo il primo carattere

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 (→).[12]

`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.

7.1.8 Caratteri speciali e comandi di *scrolling*

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)*.

[12] Ovviamente non ci sarà mai un comando che inserisce il testo proseguendo verso sinistra, sarebbe contrario alla scrittura occidentale

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.

7.1.9 *Cut'n'paste, baby!*

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*.[13] Ed infatti `vi` ci offre la possibilità di usare più di un buffer per memorizzare le nostre operazioni. Sono disponibili 26 buffer definiti *named buffer*[14] 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.[15]

Per memorizzare nel buffer *a* la prima linea del nostro file, posizioniamoci su di essa e digitiamo la sequenza `ayy`. Semplice no?[16] Spostiamoci quindi sull'ultima linea (ad esempio con `G`), e diamo un semplice `ap`. Risultato?

[13] Era un complimento, coraggio!
[14] ...per quanto un nome di una sola lettera possa non sembrare un nome!
[15] Inclusa la lettera elle, certo!
[16] Almeno, nel 1975 sembrava semplice...

```
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[17] non tocca i named buffer e un semplice `dd` non cambia il contenuto del buffer `a`.

7.1.10 Marcare la propria posizione

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".

7.1.11 Ricerche e sostituzioni con le regexpr

`vi` integra pieno supporto per le regular expression. Ricerche sul testo e sostituzioni vengono tutte eseguite attraverso pattern regexpr.[18]

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

[17] A questo punto potremmo chiamarlo *il buffer anonimo*

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

```
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

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.

7.1.12 Personalizzare `vi`

`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`.[19] 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.

[19] 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

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 `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.[20] 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

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!

[20] **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`

7.1.13 `vi`–derivati

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.

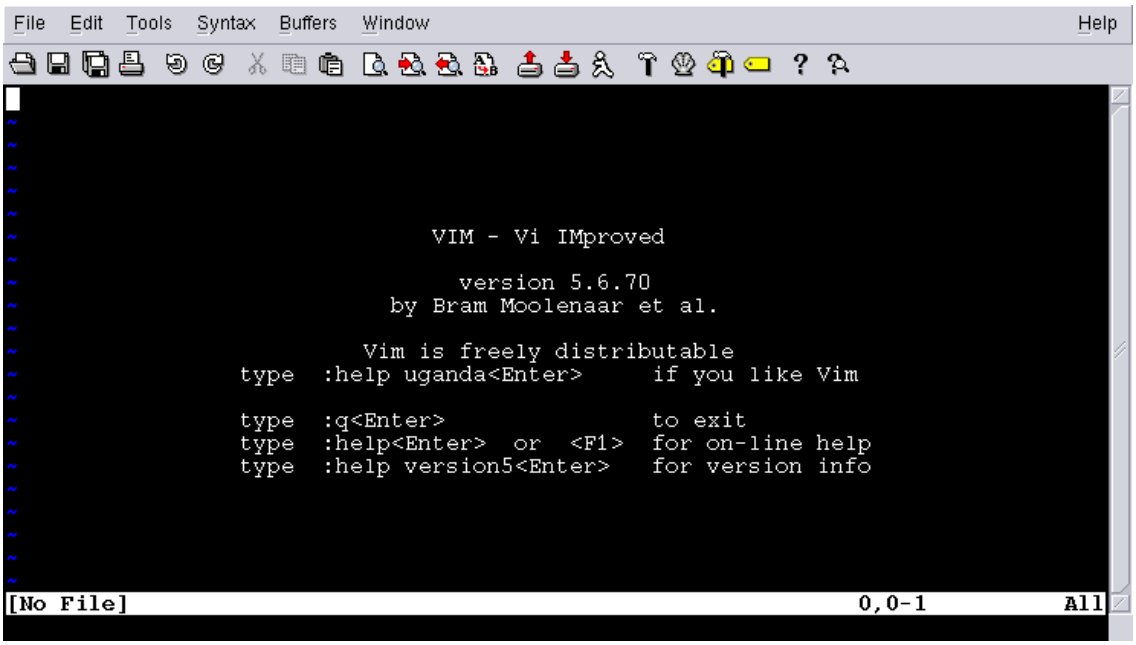
Per citare alcuni esempi:

`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`.

`Pagina conclusa artificialmente ...`



Vim con supporto grafico

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`).

7.2.1 Primo approccio con `sed`

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[21] 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:

```
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`:

[21] Attenzione: per uno script non si intende il nome di un file che contiene lo script, ma lo script stesso! Di più in seguito

```
$ 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.

7.2.2 Un primo script

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;[22] 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

[22] 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

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`[23] in quanto non modifica il file sorgente ma mostra il risultato a video.

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:

1. `myfile` *viene azzerato*
2. `sed` *viene chiamato ad operare su un file completamente vuoto*
3. *risultato:* `myfile` *è grande zero caratteri!*

7.2.3 Come viene applicato uno script?

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:

[23] E non a `ed`

```
$ cat myfile
Sistema Operativo
SO
$
$ sed '
> s/Sistema Operativo/SO/
> s/SO/Sistema Operativo/
> ' myfile
Sistema Operativo
Sistema Operativo
$
```

Esaminiamo riga per riga[24] 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*[25] 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.[26] 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.

[24] ...del file di testo

[25] La riga è stata modificata dall'istruzione prima proprio in "SO"

[26] Volendo fare un parallelo per comprendere la differenza potremmo dire che è come se con `vi` *si utilizzasse* prima di ogni regexpr un `%`

7.2.4 Comandi

`sed` definisce un set di comandi per l'editing sul testo analoghi ma leggermente differenti rispetto a quelli di `vi`. Vediamoli riassunti:

Pagina conclusa artificialmente ...

Comando	Comportamento
[*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*]]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
[*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*

7.2.5 Delimitatori in sed

In sed restano validi come delimitatori:

- i numeri delle linee
- $
- .
- /regexpr/

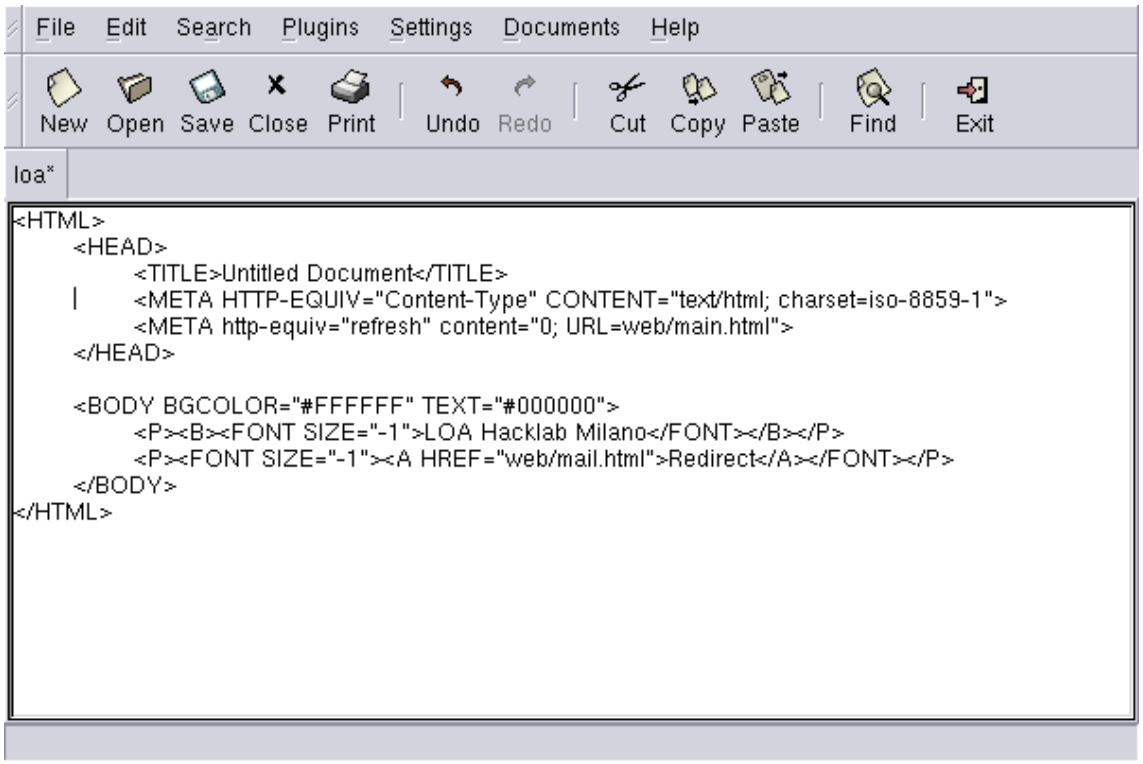
L'ultimo tipo di delimitatore consente di porre un vincolo sulle linee che offrono un match alla regexpr specificata. Ad esempio usando una espressione come /Nel mezzo/,/era smarrita/ identifica i primi tre versi della Divina Commedia. Il delimitatore % non ha senso in quanto ciascuna pattern viene già applicato a qualsiasi riga di testo contenuta nel file.

7.3 awk – INCOMPLETO – Shodan

DA SCRIVERE!!!

7.4 Altri editor

Altri editor sono nati con il passare del tempo, soprattutto in concomitanza di nuovi desktop come Gnome. Un esempio possono essere gedit.

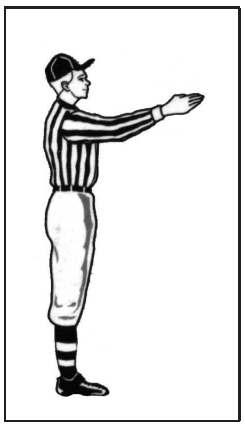


Una sessione con Gedit

INCOMPLETO!!!!!!

Capitolo 8

Il mondo là fuori



□ *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, SHAAAAAAAAAAAAARE.*

Autore: *Little-John <lidl@autistici.org>*

8.1 Collegarsi ad un sistema remoto con `telnet` e `ssh`

8.1.1 telnet

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
```

8.1.2 ssh

`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, può 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 é lo stesso che si ottiene con il programma rlogin, oramai caduto in disuso per motivi di sicurezza.

L'algoritmo é stato pensato in modo da essere sicuro e da non mandare dati relativi alla chiave privata in rete (nella manpage di `ssh` c'é una descrizione sommaria del metodo utilizzato), la vostra unica attenzione é nei confronti di chi accede fisicamente alla macchina... purtroppo `ssh` non é così evoluto da riconoscere chi sta usando il vostro computer.

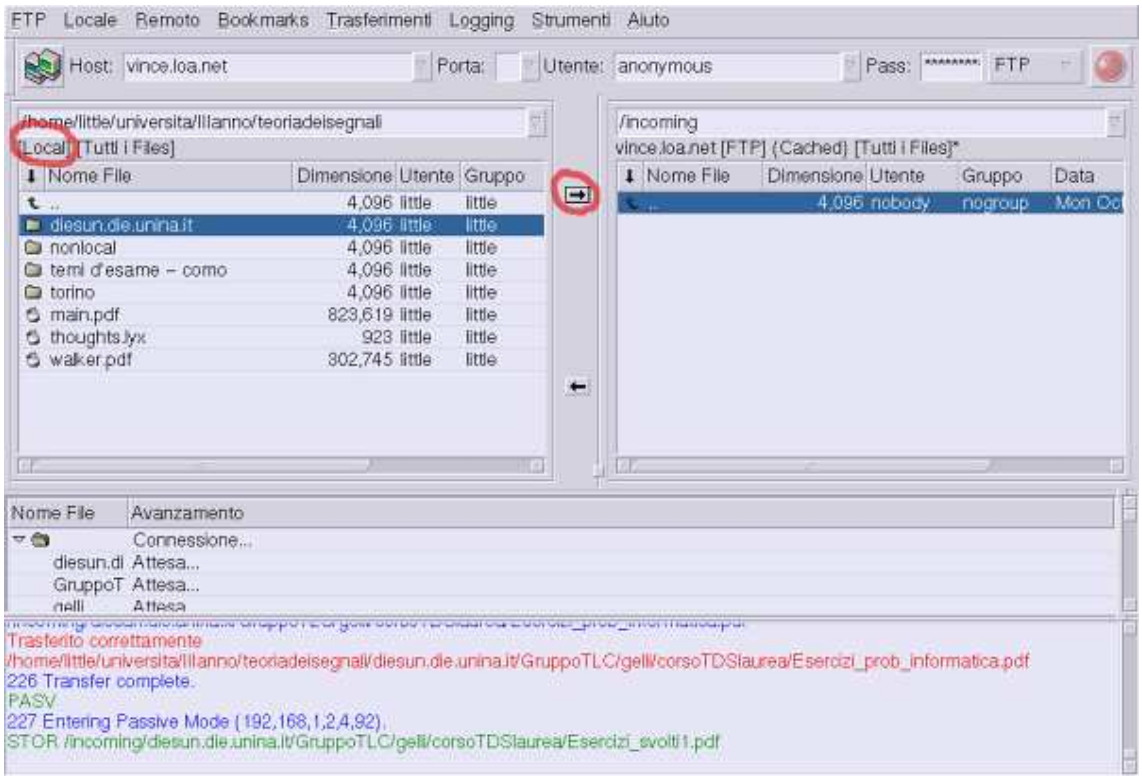
8.2 Spostare file da un host con `gftp` e `ncftp`

8.2.1 `gftp`

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.



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.

8.2.2 `ncftp`

`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 potenzialità della posta elettronica servono principalmente tre programmi:

- `fetchmail`
- `mutt`
- `procmail`

Ma analizziamoli uno per uno.

8.3.1 fetchmail

fetchmail è un ottimo tool scritto da Eric S. Raymond per poter scaricare i messaggi di posta elettronica da un server remoto. Configurarlo é 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 risolverà 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 é 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 é 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 può anche esser lanciato da riga di comando, ad esempio:

```
fetchmail -p pop3 pop.loa.taz -u littlejohn
```

che scaricherà 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 é 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 più 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`.

8.3.2 `mutt`

mutt, come dice la manpage, é un "piccolo ma molto potente Mail User Agent", ovvero client di posta. Conta approssimativamente 200 variabili di configurazione, e tra le sue feature più interessanti ricordiamo il thread sorting, ovvero la capacità 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) é corredato da una serie di file di configurazione (muttrc) utilizzabili e molto istruttivi che in un primo tempo vi possono rendere la vita più 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 farà subito una bella impressione, ma le sue potenzialità e la sua flessibilità non tarderanno ad impressionarvi. Nel prosieguo di questa sezione farò riferimento al pacchetto `mutt` distribuito con GNU/Linux Debian, ma si tratta comunque di concetti generali e di ampia applicabilità.

Supponendo di aver ricevuto un paio di messaggi di posta, lanciamo mutt, e la sua schermata di default sarà 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 più 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.

Vediamo più 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 é 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 é immediata, per sincronizzare la casella basterà premere $, oppure alla chiusura del programma ci verrà richiesto di decidere se cancellare i messaggi o no. Nel momento in cui decidiamo di scrivere una email (premendo m), `mutt` ci domanderà 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` é 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 é 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'é 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 dilungherò sulle altre opzioni, per quelle c'é 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 più facile da ricordare e, in generale, più breve da digitare. Il formato di un alias é 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 sarà automaticamente completato.

Buona lettura della vostra casella.

8.3.3 `procmail`

`procmail` é 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, più semplicemente quest'altra:

```
|/usr/bin/procmail
```

Il file .forward deve essere leggibile da tutti "world readable", o `procmail` *non funzionerà.*

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 è il sever di posta

Se siamo fortunati, `procmail` é già 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` é l'MDA di default, ti propongo una pausa di riflessione ;-), per cui vi basterà 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 é la directory che conterrà la posta, DEFAULT é il file che contiene l'INBOX, LOGFILE é il file che sarà utilizzato per tracciare il comportamento di procmail. Fatto ciò possiamo definire i filtri per la posta. Un filtro d'esempio può 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 sarà 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 é 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

Le flags più 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 sarà case sensitive

Ci sono altre flags, che riguardano i casi più 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 sarà più che sufficiente.

conditions

Le condizioni partono con un *, e sono processate dall'egrep interno (che é totalmente compatibile con la sintassi di egrep, con l'unica differenza che quello di `procmail` é case insensitive per default). Si tratta di regular expressions vere e proprie (vedi il capitolo sulle regular expressions), quindi non c'é molto da dire. La comodità é che potete specificare più regular expressions (una per riga), per tenere il procmailrc più compatto e leggibile. Ad esempio:

```
:0:
* ^From.*iOLnews@libero.it.$
* ^From.*mail.lucky.it.$
spam
```

con evidente significato dei simboli.

action

Solitamente basta scrivere la mailbox in cui vogliamo stipare l'email. Ma `procmail` offre di più:

action	Significato
!	forwarda le mail interessate dalla condizione agli indirizzi che seguono
|	permette di processare l'email con un programma esterno

mailstat

Una volta che abbiamo scaricato tutta la posta, `procmail` l'avrà 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 ~ %
```

così é più facile capire cosa é successo durante il donwload delle mail.

Giochi di prestigio con procmail

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 é 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'utilità di tutto ciò ;-).

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 é un programma che vi permette di fare tante belle cose, tra cui inviare delle mail da riga di comando. Non mi soffermerò su formail, perchè esula dagli scopi di questa versione del capitolo. Al di là 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 é nella manpage procmailrc), ed é 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

8.4.1 lynx

`lynx` é tra i più anziani browser testuali, con circa dieci anni di sviluppo alle spalle. La navigazione é un po' ostica (finquando non prendete la mano ;-) e poco agevole lì 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 é 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

8.4.2 links

`links` é un altro client testuale, che rispetto a lynx offre il supporto per i frames, il download in background e una formattazione delle tabelle più decente, giusto per citare le differenze più 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 più comuni del browser (apri l'url, scarica il file, reload...).

```
 File   View   Link   Downloads   Setup   Help
+-----------------------------+
| Go to URL                 g |
| Go back                  <- |
| History                   > |
| Reload              Ctrl-R  |
| Bookmarks                 s |
| Add bookmark              a |
| New window                  |
|-----------------------------|
| Save as                     |
| Save URL as                 |
| Save formatted document     |
|-----------------------------|
| Kill background connections |
| Flush all caches            |
| Resource info               |
| Memory info                 |
|-----------------------------|
| OS shell                    |
|-----------------------------|
| Exit                      q |
+-----------------------------+
```

Naturalmente i programmatori non ci costringono ad usare solo i menu, ma le funzioni più 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.

8.4.3 `mozilla`

`mozilla` é il browser grafico per antonomasia, derivato dai sorgenti di Netscape 5, in costante aggiornamento ed evoluzione. `mozilla` é basato sul motore di rendering delle pagine html chiamato Gecko, che non ha concorrenti in fatto di performance e qualità di rendering, e su cui

sono basati anche altri programmi opensource. Uno dei punti contro `mozilla` é 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 é molto simile a quella di Netscape, con i menu oramai di rito in un programma grafico motivo per cui non mi soffermerò su di essi, lasciandovi la curiosità di navigarli.

L'argomento di cui mi voglio occupare é quello della gestion della privacy con mozilla. Se andate sul menu Edit e scegliete Preferencies, vi comparirà una schermata in cui tra le diverse voci di personalizzazione, c'é "Privacy and Security". Vediamo una per una le voci, cercando di capire il loro significato.

Cookies

Un cookie é un piccolo file che un server web può lasciare e leggere sul vostro computer, e può 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 pubblicità selvaggia in rete, gli scopi sono altri (sapere quali banner avete già 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 é la migliore, visto che così non accetterete nessun cookie. Alcuni siti, specie quelli con delle sezioni ad accesso con password, hanno necessità dei cookie, ma in questo modo non riuscirete a visitarli.

La seconda é un passo intermedio tra l'accettazione totale dei cookie (cioè la terza opzione) e il rifiuto dei cookie illeggittimi, visto che il browser accetterrà i cookie solo dal sito che state effettivamente visitando e non da altri server. Non é di grande utilità, 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

Più che impostazioni di sicurezza, si tratta di personalizzare il comportamento del browser nei confronti delle immagini sui siti. L'impostazione più interessante é quella sul loop delle immagini. Nella maggior parte dei casi le immagini gif animate servono per rendere i banner pubblicitari più visibili, facendoli ad esempio lampeggiare nella pagina con una frequenza elevata. Per evitare che ciò accada, semplicemente scegliete nel riquadro in basso, "Never".

Forms

Quando riempite dei form in rete, `mozilla` può salvare quanto avete scritto. Si tratta di un'arma a doppio taglio, perché 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

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 più idonea, a seconda che si tratti di un computer più o meno accessibile ad altri utenti. Una forma di sicurezza in più, consiste nella crittazione delle informazioni sul disco, che ottenete selezionando il box "Use encryption when storing sensitive data".

Master Passwords

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 chiederà una password per accedere alle informazioni sul disco. A seconda dell'impostazione che scegliete vi sarà richiesta la master password con più o meno frequenza.

SSL

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, é 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 é 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 é un problema purtroppo non molto avvertito dai navigatori, ed é un vero peccato, visto che lo sniffing (ascolto passivo delle informazioni in transito) é una delle forme di abuso più utilizzate e più temibili perchè non si può evitare se non con la crittografia.

Certificates

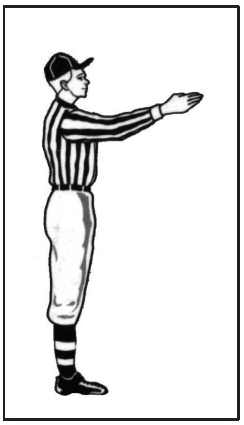
Un certificato non é altro che un altro strumento di crittografia, per esser certi che le informazioni che un sito ci sta passando sono autentiche, cioè che non ci sia stato "poisoning" (avvelenamento) di quanto riceviamo dal server http da parte di terzi. Quando un sito ci offre un certificato, `mozilla` ci mostra le caratteristiche del certificato, e se non ci fidiamo possiamo scegliere di non accettarlo.

8.4.4 galeon

`galeon` é un "hack" di mozilla, che mira ad essere un software di navigazione più leggero e che faccia solo quello (il suo motto é "The web, only the web"...). Come dire, é "solo" un web browser. Le sue caratteristiche sono la leggerezza e la qualità di rendering, visto che `galeon` é 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` é 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 soffermerò sulla trattazione del pacchetto. Un consiglio, installatelo e usatelo.

Capitolo 9

X Windows System – Manhattan



□ *X Window System*[a] *è lo standard grafico di riferimento per i sistemi UNIX.*
X non può essere definito semplicemente come GUI o Graphical User Interface, ma è un esempio di vero e proprio sistema grafico completo. Esso non fornisce una singola interfaccia all'utente, ma un intero set di GUI da cui sceglierne una da utilizzare di volta in volta. Non importa, poi, quale applicazione grafica deciderete di usare all'interno di X, essa funzionerà[b] *egregiamente anche se non espressamente creata per l'ambiente grafico utilizzato al momento.*

Autore: *Manhattan <manhattan@ecn.org>*

[a] Informalmente conosciuto come X
[b] Il 99% delle volte :-)

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, l'MIT non poteva sapere quale tipo di hardware si sarebbe reso disponibile e 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, facevano giusto al caso, così 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 uno stesso sistema o in dialogo attraverso 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[1]: server (*X server* viene definito come l'apparato che visualizzerà la grafica richiesta mentre il **programma** che si occupa di fornire i dati

[1] Pensavate fosse così semplice?? ... In effetto lo è! :-)

al server viene definito *X client*. Spesso anche i professionisti hanno problemi a capire questa tipologia di modello; voi, al momento, accettatela e dite di sì con la testa.

È molto più importante capire che ciò che potrete vedere visualizzato sullo schermo sta girando altrove sulla vostra macchina o 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.

9.2.1 X Window Protocol

Il protocollo X Window è una sorta di 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 (in questo caso le regole permettono di disegnare 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. Anche una macchina Windows o Macintosh potrà implementare tale protocollo e dunque visualizzare la grafica di 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 X11R6.4, cioè versione 11, revisione 6.4.

La versione di X Window Protocol in uso oggi è datata Settembre 1987. Poiche' dopo oltre 10 anni molti sistemi e applicazioni sfruttano questo semplice protocollo, che si adatta in questa sua stesura anche alle piattaforme e all'hardware piu' obsoleti, possiamo stimare che non ci saranno nuove versioni di questo protocollo. Modificarne le strutture comporterebbe solo incompatibilità[2].

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).

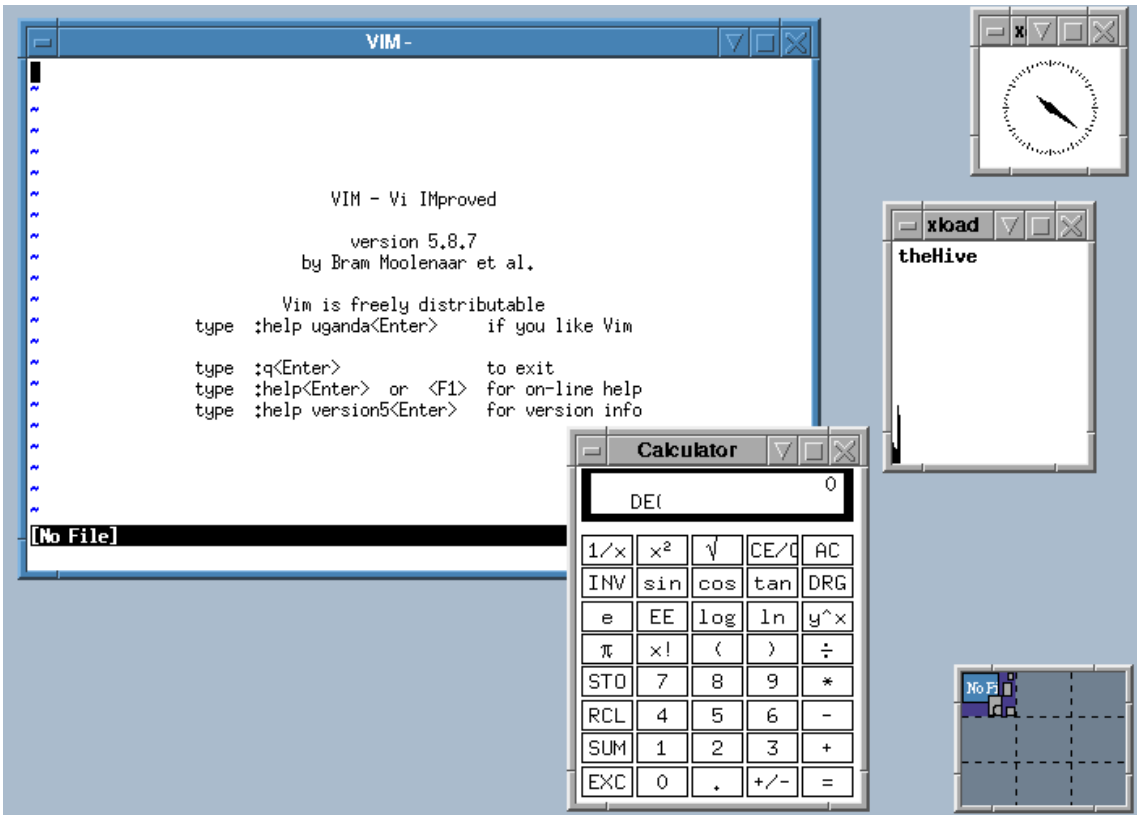
Gli svantaggi di un simile approccio consistono nella mancanza di una standardizzazione del cosiddetto *look-and-feel*[3] delle applicazioni, nella mancanza di regole definite per l'attribuizione di

[2] La verità è che X Window Protocol è perfetto

[3] Anche se non mi sembra uno svantaggio

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, seppur rallentati, i moderni programmi possono girare su piattaforme vetuste o poco potenti, senza doversi curare della possibilità di creare nuove regole di visualizzazione.



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, cioè 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.

Questo particolare fa sì che sia dunque possibile operare, ad esempio, su interi gruppi di finestre contemporaneamente.

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, eventualmente, 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*[4]

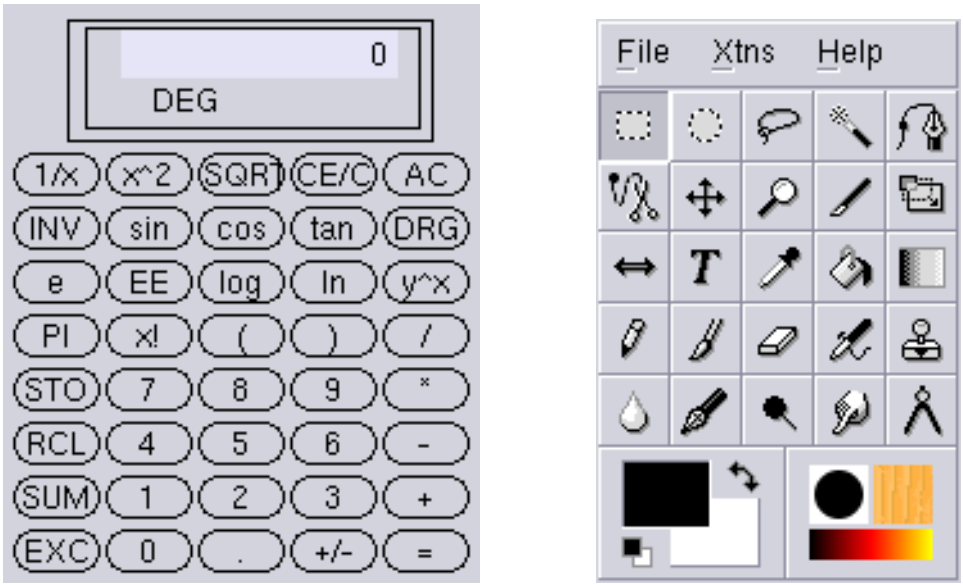
In termini di X Window System, widget sono tutti i particolari che compongono una finestra, come la scrollbar, i menù, tutti i vari tipi di bottoni e via dicendo. Ognuno è libero di creare e utilizzare un insieme di widget a picere, anche se, di fatto, esistono da molto tempo set di widget ben rodati che permettono di creare con facilità ogni tipo di interfaccia.

I widget più conosciuti sono: Athena widget, scaturito 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.

[4]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)



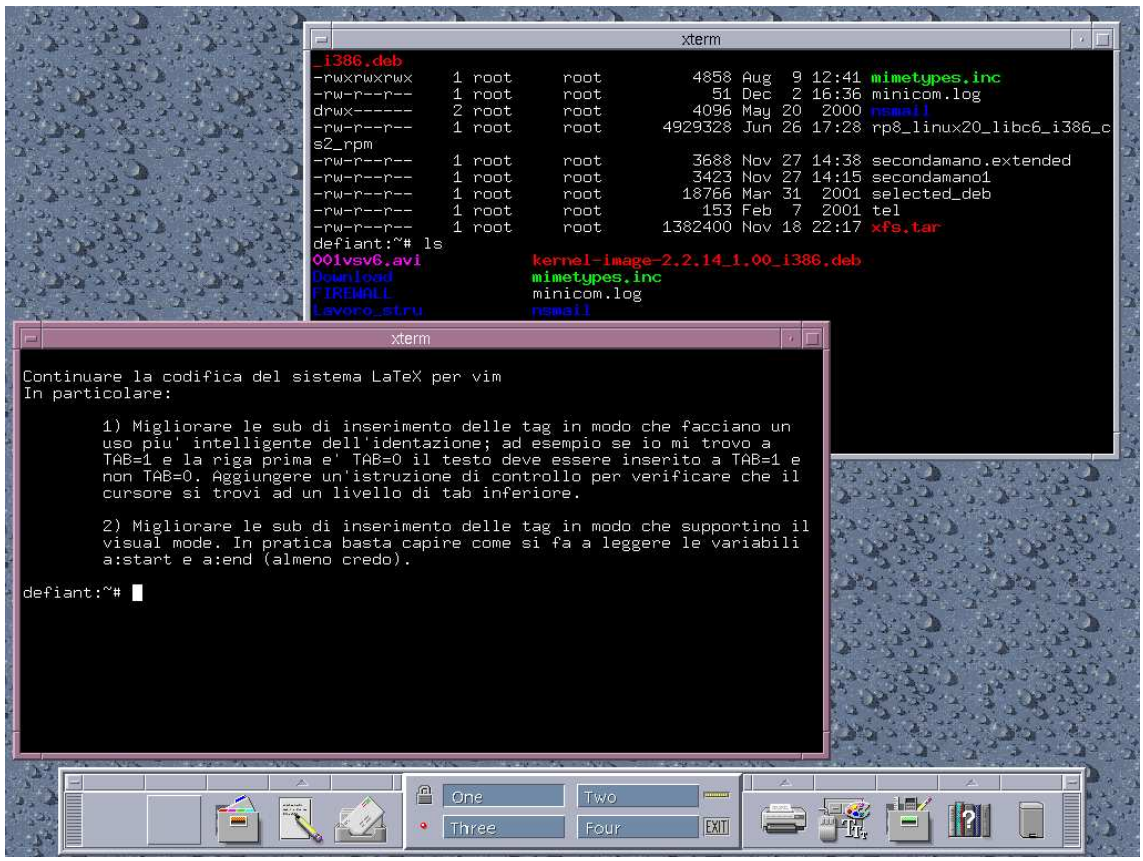
Finestre realizzate con due toolkit differenti: `xcalc`*, a sinistra, usa il Toolkit Athena, The GIMP, a destra, usa il Toolkit Gtk*

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.

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?



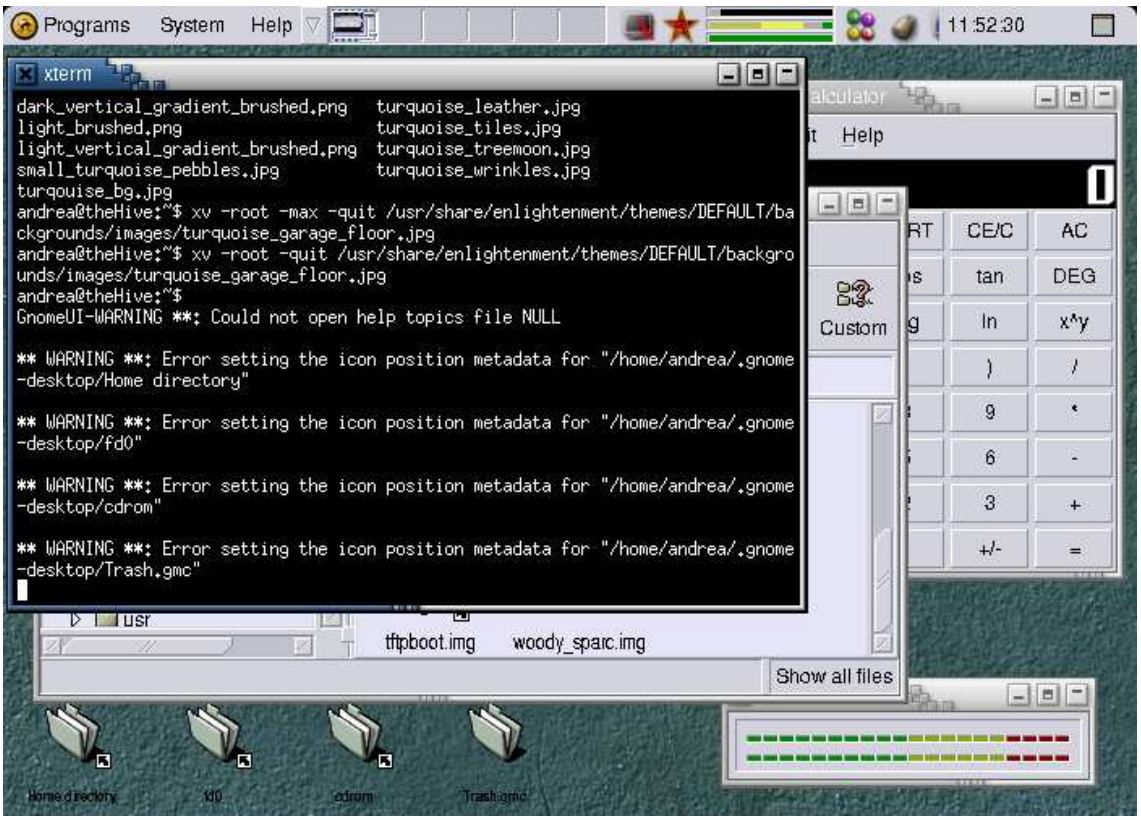
CDE con il Motif Window Manager

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 utilizzare o emulare i tre tasti del vostro mouse 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[5].



Ximian Gnome e Sawfish

I window manager sono decine e non vale la pena di analizzarli uno per uno. Tutti quanti sono ampiamente personalizzabili e tra di loro le diversità maggiori riguardano il consumo di memoria RAM. Si possono citare, per esempio, i classici e standard *twm*, Tab Window Manager, *mwm*, il window manager di Motif, *fvwm*, Fantasmagoric[6] 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...

[5] 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 e le sue finestre aperte!

[6] In realtà il significato esatto della *f* di fvwm si è perso nel più profondo mistero...

Tab Window Manager (`twm`)

Motif Window Manager (`mwm`)

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 vengono fornite forzatamente, senza possibilità di farne a meno. Invece il sistema X Window può tranquillamente lavorare egregiamente anche senza il loro ausilio[7].

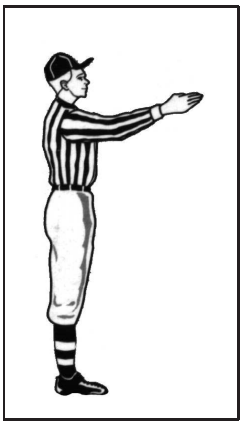
In effetti molti utenti si accontentano di un ambiente grafico senza troppi orpelli e fronzoli, sgombro da icone e applet pressoché inutili e parco di memoria RAM. Io sono uno di questi utenti. Chi invece non fosse preoccupato dal dispendio di risorse[8] e volesse immergersi in un'esperienza molto simile a quella che si ha accedendo ad un Mac, potrà scegliere tra *CDE*, *GNOME* o *KDE* e configurare a proprio piacimento il suo desktop.

[7] 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

[8] 32 Mb di RAM sono più che sufficienti

Capitolo 10

Post Scripta Manent...



□ *Come si stampa sotto UNIX – Quali sono i comandi per stampare, verificare la "coda" di stampa, cancellare una stampa in attesa.*

Autore: *Tx0 <tx0autistici.org>*

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:

Pagina conclusa artificialmente ...

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

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.

[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

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:

```
$ 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:

```
$ 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].

[3] ossia? indovinato vero? *line printer remove*

[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...

Capitolo 11

Conclusioni, ringraziamenti, riconoscimenti, speranze, anatemi e altri sollazzi di fine libro

Indice

1 Storia di UNIX **1**

2 Sezione generica UNIX • Shodan **9**
2.1 Cos'è un sistema operativo 9
2.1.1 Il Kernel 9
2.1.2 Unix è multitasking 9
2.1.3 Unix è multiutente 10

3 Entriamo nel sistema – INCOMPLETO – Shodan **11**
3.1 Cos'è un account 11
3.1.1 Consigli per una password sicura 11
3.2 La procedura di login e logout 12
3.3 Comandi di Base 12
3.3.1 Oggi è un bel giorno da ricordare: *date, cal* 12
3.3.2 Dove sono finito: *hostname, uname* 14
3.3.3 C'è nessuno?: *who, w* 16
3.3.4 I file: *ls* 17
3.3.5 I permessi e la proprietà dei file: *chmod, chown* 18

4 Puntiamo più in alto **19**
4.1 Trovare file con `which`, `find` e `locate` 20
4.2 Creare archivi con `tar` 24
4.3 Comprimere file con `gzip` e `bzip2` 27
4.4 Dividere gli archivi con `split` 29
4.5 Ai piedi dei file e oltre: `tail, sort` 31
4.6 Ricerche su testo con `grep` 32
4.7 I processi e la loro gestione: `kill`, `top`, `ps` e `uptime` 34
4.7.1 `kill` 35
4.7.2 foreground o background? 35
4.7.3 `top` 37
4.7.4 Priorità e *nice* 40

4.7.5 `ps` 41

5 La Shell • INCOMPLETO • Vedi Sezioni 45
5.1 Differenti shell 46
5.1.1 Elementi di una shell 46
5.2 `sh` e derivate 47
5.2.1 Il Prompt 47
5.2.2 Variabili notevoli 49
5.2.3 Controllo dei processi 50
5.2.4 Input e Output, Redirezione, Cilindri, Conigli Bianchi... 51
5.2.5 Alias 52
5.2.6 Sintassi di programmazione 54
5.3 `csh` e derivate • INCOMPLETO • Shodan 57
5.3.1 Il Prompt 57
5.3.2 Variabili notevoli 57
5.3.3 Input e Output 57
5.3.4 Controllo dei processi 57
5.3.5 Sintassi di programmazione 58
5.3.6 I file di configurazione 58

6 Regular Expressions 59
6.1 Perché le Regular Expression? 60
6.1.1 Due convenzioni, molti meno problemi 60
6.2 La più semplice Regular Expression 60
6.3 I Quantificatori 61
6.4 Un carattere solitario 63
6.5 Caratteri di Classe 63
6.6 Infrangiamo (apparentemente) un po' di regole 64
6.7 Viviamo in un mondo avaro, baby! 66
6.8 Tanto di cappello *(e scarpe)*: ˆ e $ 68
6.9 Commenti indiscreti 69
6.10 Meglio poter scegliere 69
6.11 Sostituzione con le regexpr 70
6.11.1 Eliminare i commenti di uno script 71
6.11.2 Sostituzione multipla 71
6.11.3 Sostituzione multipla con uso delle posizioni 71
6.12 Opzioni e altre meraviglie 72

7 Editor di testo – INCOMPLETO – vedi sezioni 75
7.1 `vi` 76
7.1.1 Una personalità schizofrenica 76
7.1.2 Inserire del testo 77
7.1.3 Muoversi attraverso il testo 77

7.1.4 Salvare ed uscire . . . 78
7.1.5 Cancellare, copiare, modificare e sostituire . . . 78
7.1.6 Anche gli Utenti UNIX possono sbagliare . . . 81
7.1.7 Diversi modi per entrare in *insert mode* . . . 82
7.1.8 Caratteri speciali e comandi di *scrolling* . . . 83
7.1.9 *Cut'n'paste, baby!* . . . 84
7.1.10 Marcare la propria posizione . . . 85
7.1.11 Ricerche e sostituzioni con le regexpr . . . 85
7.1.12 Personalizzare `vi` . . . 88
7.1.13 `vi`–derivati . . . 91
7.2 `sed` . . . 93
7.2.1 Primo approccio con `sed` . . . 94
7.2.2 Un primo script . . . 96
7.2.3 Come viene applicato uno script? . . . 97
7.2.4 Comandi . . . 98
7.2.5 Delimitatori in `sed` . . . 100
7.3 `awk` – INCOMPLETO – Shodan . . . 100
7.4 Altri editor . . . 100

8 Il mondo là fuori 103
8.1 Collegarsi ad un sistema remoto con `telnet` e `ssh` . . . 104
8.1.1 telnet . . . 104
8.1.2 ssh . . . 105
8.2 Spostare file da un host con `gftp` e `ncftp` . . . 107
8.2.1 `gftp` . . . 107
8.2.2 `ncftp` . . . 108
8.3 Tutto sulla mail . . . 112
8.3.1 `fetchmail` . . . 113
8.3.2 `mutt` . . . 114
8.3.3 `procmail` . . . 117
8.4 Navigare in rete . . . 121
8.4.1 lynx . . . 121
8.4.2 links . . . 122
8.4.3 `mozilla` . . . 123
8.4.4 galeon . . . 126

9 X Windows System – Manhattan 127
9.1 Un po' di storia . . . 128
9.2 Il modello Client/Server di X Window . . . 128
9.2.1 X Window Protocol . . . 129
9.3 L'evoluzione di X negli anni . . . 129
9.4 Meccanismi, non regole . . . 129
9.5 Gerarchie, Widgets e Toolkit . . . 130

9.6 Window Manager e Desktop Environment 132

10 Post Scripta Manent... 137
10.1 Formati di stampa 138
10.2 Capolino dietro le quinte 138
10.3 Cioè per stampare? 140
10.4 Ok, ok, e mettiamo che sbagli? 140

11 Conclusioni, ringraziamenti, riconoscimenti, speranze, anatemi e altri sollazzi di fine libro 143