Introduzione a Linux (ed un po' anche a Windows)

mobytrick

3 settembre 2014

Breve introduzione ai concetti di file, directory/cartella e link sia in ambiente Linux che Windows. I concetti sono poi stati inseriti nel contesto del file system e del sistema operativo.
Introduzione al concetto di comando, compresa la gestione. Spiegazione dei comandi (principali e non) relativi alla gestione dei file.

1 Il file

Possedere un PC (sigla che sta per **p**ersonal **c**omputer) una cosa abbastanza comune, al giorno d'oggi. Con un PC si possono fare tante cose, più di quelle che si possano immaginare. Si possono conservare foto, canzoni, ma anche conversare con altre persone oppure scambiare messaggi di posta elettronica. Questi sono alcuni esempi. L'elenco non ha alcuna pretesa di essere esaustivo.

Ad un utilizzatore poco interessato ai dettagli interni un PC appare come una specie di contenitore di file. Questo neologismo sta ad indicare una serie di dati correlati a cui è stato affibbiato un nome. Un file potrebbe contenere una foto, ma anche un video oppure un un brano musicale. Oppure un'applicazione se non, più banalmente, un testo. Riprendendo l'esempio della foto. Questa può essere pensata come un insieme di punti ed i dati descrivono le caratteristiche di ciascuno di essi (ad esempio il colore; ma non solo). Rispetto ai primi tempi in cui i file erano per lo più testuali oppure dei programmi (ma attualmente si preferisce il termine `"app"` contrazione di "*application*") oggi non esiste praticamente limite. Qualsiasi stimolo inerente la vista oppure l'udito può essere memorizzato in un file. Esiste quindi la necessità di riconoscere in maniera abbastanza precisa il contenuto di un file senza dover andarci dentro. Accanto al nome del file, che dovrebbe essere mnemonico, ha assunto grande importanza la cosiddetta "*estensione*". Nel mentre il nome identifica il file, l'estensione indica la natura del contenuto (dietro le quinte

c'è stato un tacito accordo sulle estensioni e sul significato da attribuire loro). Col tempo si sono venute a costituire delle convenzioni sulle quali poggia un'infrastruttura che magari ai più non è nota. Supponiamo di avere una foto. È abbastanza naturale che si voglia guardarla. Molto probabilmente l'estensione del file che la contiene è `.jpg`. Per la convenzione di cui sopra, ad ogni estensione è associata l'app appropriata. Così se voglio visualizzare una foto verrà invocato automaticamente il programma apposito. Che è differente da quello per visualizzare un file con estensione, ad esempio, `.pdf`. Il tutto poggia sulla fiducia. Se un file ha estensione `.jpg`, ci si aspetta che poi il contenuto sia conforme a determinate specifiche (cfr: l'abito non fa il monaco). Se così non fosse, il file sarebbe inutilizzabile. La convenzione è un'arma a doppio taglio. Può succedere che ci si procuri un file che può essere visualizzato solo da un'app che però non c'è nel PC che stiamo usando. Anche in questo caso, sino a che non ci si procura il programma apposito del file non ce ne facciamo nulla.

Ancora alcuni concetti che riguardano il nome dei file. Le regole cui attenersi sono alquanto differenti. Una caratteristica specifica di Linux è quella di essere "*case sensitive*". In altre parole, le lettere maiuscole e quelle minuscole sono differenti. L'esatto contrario di Windows. Altra caratteristica di Linux riguarda l'estensione. Non esiste però viene mimata. Pure in Linux è stata mutuata la tecnica Windows di associare all'estensione la relativa app. Però in Linux non si parla di estensione, bensì di "*ultimi caratteri del nome*". Ad esempio, il nome di un file che contiene una foto deve terminare con `".jpg"` ma giova ricordare che il collegamento tra ultima parte del nome ed app è molto più lasco che non in Windows. Un'altra grossa differenza riguarda il carattere spazio. Windows lo ammette e lo gestisce senza problemi. Anche in Linux è possibile usare lo spazio. Ma è difficile da gestire ed è fonte di malfunzionamenti, tanto da sconsigliare la sua presenza. Dovendo barcamenarsi tra i due sistemi operativi, per creare un nome di file valido per entrambi usare:

- lettere minuscole
- lettere maiuscole (con moderazione)
- cifre
- i caratteri `-` (trattino), `_` (sottolineato), `.` (punto)

Vale sempre la regola che in Windows maiuscole e minuscole sono uguali. Evitare la presenza dei caratteri speciali all'inizio del nome, così come la loro ripetizione. Bandite nel modo più rigoroso le vocali accentate ed in linea più

generale i segni diacritici che Linux non è in grado di gestire. Riguardo alla lunghezza del nome, Linux da sempre ammette nomi molto lunghi. Windows invece ha impiegato molti anni per abbandonare la restrittiva regola nota come "8+3": 8 caratteri per il nome e 3 per l'estensione. Ora non ci sono più differenze, ma va da sé che il nome di un file dovrebbe avere una lunghezza tale da far capire, anche col contributo dell'estensione, il suo contenuto senza per questo diventare prolisso.

2 La cartella/directory

I file presenti in un PC non sono messi dentro alla rinfusa. Se così fosse, un PC non sarebbe in grado di funzionare. Facciamo un paragone con una biblioteca. Il suo compito è quello di fornire libri. Questi non possono essere ammucchiati. Per essere fruibili, i libri vanno collocati fisicamente entro scaffalature ed inoltre quelli che trattano argomenti simili vanno messi assieme. In un PC le problematiche sono sorprendentemente simili. Esistono le analogie riportate nella tabella 1 a pag. 3.

libro	file
scaffale	cartella

Tabella 1: Analogie tra biblioteca e PC

Il termine `file` è ormai attestato nella lingua italiana come anglismo. Ai primordi qualcuno ebbe l'orrenda idea di usare il termine flusso. Per nostra fortuna l'idea non prese piede. Il termine "cartella" è normalmente usato in ambiente Windows. In ambito Linux si usa il termine *directory*. Anche in questo caso per nostra fortuna non hanno avuto fortuna i tentativi di traduzione (direttorio, direttrice).

In un PC la cartella/directory è un file di tipo particolare. Ad esempio, non è possibile visualizzare il suo contenuto. Che comunque è noto: una directory contiene l'elenco dei file presenti. In Linux se una directory viene popolata intensamente, la sua dimensione viene ampliata per poter contenere l'aumentata dimensione dell'elenco dei file. Ma se poi la popolazione viene diradata, la sua dimensione non viene ridotta.

3 Il sistema operativo

Senza energia elettrica un PC sarebbe un'inerte accozzaglia di circuiti elettronici. Ciò che rende un PC diverso da un oggetto inanimato è il cosiddetto

software. Ovvero, dietro la fornitura di energia elettrica, il PC prende delle decisioni. Che però sono state decise a priori da un essere umano. Prendiamo ad esempio un frigorifero collocato in un ambiente termicamente non estremo. Il cervello pensante del frigorifero è il termometro che misura la temperatura interna. Se questa è superiore al massimo (limite stabilito da un umano) il compressore si mette in moto e la abbassa. Quando questa scende al di sotto del minimo (pure questo un limite stabilito da un umano) il funzionamento si interrompe. La temperatura comincia ad innalzarsi e quando supera il limite superiore il funzionamento riprende. Avendo postulato che il frigorifero sia stato collocato in un ambiente non estremo, ci si aspetta un funzionamento intermittente pilotato da due temperature. Se invece è stato collocato in ambiente tropicale, il funzionamento potrebbe essere continuo (ad esempio, il frigo potrebbe non farcela ad abbassare la temperatura). Analogamente, collocandolo in ambiente polare, la temperatura ambientale potrebbe essere stabilmente al di sotto del minimo, precludendo così il funzionamento.

Una cosa analoga succede con i PC. Qui al posto del termometro e delle temperature c'è il sistema operativo: un insieme di applicazioni che cooperando tra di loro permettono il funzionamento. Se stabiliamo un parallelo tra un PC ed un'automobile, il motore sta all'auto come il sistema operativo sta al PC. Così come per le automobili esistono diverse ditte che costruiscono motori, così per i PC sono stati escogitati diversi sistemi operativi. Col tempo il loro numero si è drasticamente ridotto, tanto che attualmente esistono 2 modelli: Linux e Windows (citati in ordine *alfabetico*). È tradizione dare un nome al sistema operativo. Windows è stato così chiamato per enfatizzare la caratteristica che lo connota, vale a dire le finestre. Linux invece è stato coniato incrociando il nome di un preesistente sistema operativo (Unix, pure lui con la sua storia ma andremmo fuori tema) col nome dello studente finlandese (Linus) che lo ha creato.

La differenza fondamentale tra Windows e Linux è la seguente. Il primo è stato creato da un gruppo di persone che si sono fatte pagare. Anzi, per tutelarsi da eventuali furti hanno preso tutte le misure di cui sono stati capaci. Si parla di sistema operativo "*proprietario*" in quanto proprietà della società che lo ha sviluppato, ne cura la manutenzione, l'aggiornamento e la commercializzazione. Si parla pure di sistema operativo "*chiuso*" in quanto i dettagli sono conosciuti solo ad un ristretto gruppo di persone.

Al contrario, Linux è il sistema operativo "*aperto*" per antonomasia. Tutto, ma proprio tutto, è pubblico, quindi accessibile. Nessuna zona opaca. I vari pezzi che lo compongono sono stati sviluppati da varie persone che per loro esplicita e precisa scelta ne hanno trattenuto per sé la sola proprietà intellettuale. Chiunque può proporre aggiunte e modifiche. Come pure esiste la possibilità che qualcuno proponga la propria versione di cose già esistenti.

Il giudizio è demandato alla rete, neutra per definizione, che ha dimostrato di possedere una capacità infallibile nel giudicare la validità delle soluzioni.

Due mondi antitetici. Dopo anni di battaglie e con la guerra a Linux intrapresa e persa da Microsoft ora c'è un modus vivendi. I due sistemi operativi possono essere installati sullo stesso PC lasciando la scelta all'accensione. Ma c'è di più. Da Linux è possibile intervenire tra i file di Windows con possibilità di lettura e scrittura (pratica però da utilizzare con cautela). Da parte di Windows è stata creata da terze parti un'app che permette di intervenire nella parte Linux, ma solo in lettura.

4 Il file system

Da quanto detto in precedenza, in un PC i file non sono collocati alla rinfusa, ma organizzati entro cartelle (o directory). Anzi, un file può essere collocato solo dentro una cartella. Questa, quindi, contiene file. Ma può contenere pure altre sotto cartelle. Che a loro volta possono contenere file ed altre cartelle. Si viene così a creare un'incastellatura -chiamata "*file system*"- fortemente gerarchizzata e di tipo piramidale.

In Linux la directory al vertice della piramide entro la quale si trovano alcuni file ed altre directory è nota con il nome di *root* (in it: radice) ed è indicata dal simbolo `"/"` (la sola barra). In ambiente Windows al vertice della piramide c'è il nome simbolico del disco sul quale risiede il sistema operativo. Nella stragrande maggioranza dei casi tale disco è quello contrassegnato dalla lettera `C` seguita dal carattere : (doppio punto). Qui si trova la cartella principale che oltre ad alcuni file contiene delle sotto cartelle. Che a loro volta possono contenere file e cartelle.

Di primo acchito le due incastellature sembrerebbero se non uguali abbastanza simili. Ma è una sensazione sbagliata. I 2 file system sono differenti e, a meno di accorgimenti, incompatibili tra di loro. A parte la possibilità, accennata in precedenza, di poter intervenire nel sistema operativo antagonista, per nostra fortuna esiste un ulteriore tipo di file system riconosciuto da entrambi. La cooperazione quindi, dopo anni di preclusione, è garantita. Lo scoglio maggiore da affrontare, ma non l'unico, riguarda la gestione dei file.

Ad un utilizzatore superficiale le complessità interne non interessano. Ma sino ad un determinato punto. In ambiente Windows è abbastanza normale ottenere l'elenco dei file presenti in una cartella, ordinandoli in maniera cronologica. Ecco, questo è il punto. La data di un file fa parte del file senza però essere parte dei dati che il file contiene. Un file, anzi, qualsiasi file è costituito da due parti. Una contiene i dati veri e propri e la dimensione varia

a seconda dei casi. L'altra invece ha una lunghezza fissa e contiene la parte "*amministrativa*": il nome, la dimensione, la data di creazione ed altre caratteristiche. Tale parte in ambiente Windows viene chiamata genericamente FAT (**F**ile **A**llocation **T**able) mentre in Unix assume il nome di "*i-node*"[1]. FAT ed i-node contengono in parte gli stessi dati, ma per la maggior parte racchiudono informazioni che sono peculiari del sistema operativo. Sono strutturalmente differenti. Rendere possibile la cooperazione significa che per i file deve essere prevista una struttura amministrativa in grado di contenere entrambe le informazioni senza che si intralcino a vicenda.

5 Il concetto di *user*

In un PC, a prescindere dal sistema operativo usato, ci sono i file del sistema operativo e quelli creati da chi sta utilizzando il PC. Per ragioni di sicurezza, queste due categorie di file devono essere tenute separate al meglio. Memorizzare i file in cartelle separate non è sufficiente. Per aumentare la sicurezza è stato introdotto il concetto di "*proprietà del file*". Qualsiasi file, ma il discorso vale pure per le cartelle, deve/può essere usato solo da chi ne è il proprietario. Supponiamo che esista una famiglia composta da una coppia di partner e che ci sia 1 solo PC. Ognuno dei due ha diritto ad avere i propri file e non deve interferire né con quelli del(la) partner né tanto meno con quelli del sistema operativo. La soluzione escogitata è la seguente: per poter usare il PC l'utilizzatore deve comunicare al sistema operativo chi è. Se è riconosciuto, si è ammessi ad usare il PC. In caso contrario l'accesso viene negato. Essere ammessi significa poter gestire i propri file, ad esempio modificarli, cancellarli, crearne di nuovi. In totale libertà e con la garanzia della privacy.

Quando si acquista un PC nella maggior parte dei casi il sistema operativo è già installato e si tratta di Windows. La prima installazione parziale, fatta a cura del venditore, prevede la creazione di un utente che è il proprietario dei file che costituiscono il sistema operativo. Tale utente viene indicato come "*Administrator*" (in it. Amministratore). Una volta acquistato il PC l'installazione va completata. Una delle prime incombenze da assolvere è associare ad Administrator una "*password*"[2]. Ovvero, per poter usare il PC non è sufficiente immettere una stringa di caratteri nota come utente (in

[1] ad essere rigorosi il nome del file non è contenuto nell'i-node. Il nome del file si trova elencato nella directory ove gli viene associato il cosiddetto *i-number* che individua l'i-node relativo.

[2] Limitatamente alla password, Windows fa un'eccezione alla sua filosofia che equipara le lettere maiuscole e quelle minuscole. Quindi anche in Windows la password è *case sensitive*

inglese "*user*"), perché troppo facile da individuare. Bisogna pure inserire un'ulteriore stringa, da mantenere segreta, la ben nota password. Costituisce uno dei cardini su cui poggia la sicurezza del PC.

Una delle prerogative dell'amministratore è quella di poter creare altri utenti. In pratica si tratta di tener traccia in un elenco delle persone a cui verrà consentito l'uso del PC. Ad ogni utente inserito nell'elenco viene associata una password che poi l'utente ha la facoltà/dovere di cambiare. Un'altra prerogativa è quella di poter ispezionare qualsiasi file/directory presente nel PC e ciò a prescindere dalla proprietà (ma chi amministra un calcolatore ha ben altro da fare che intrufolarsi indebitamente dove non gli compete).

In ambiente Linux le cose sono leggermente differenti, ma non di molto. Visto che il sistema operativo bisogna istallarselo da soli, la procedura prevede la creazione di un utente particolare -*root*- noto talvolta anche come superuser, cui bisogna associare la password. Le sue prerogative sono la fotocopia di quelle concesse ad Administrator. A lui solo è concessa la facoltà di creare nuovi utenti come pure il fatto di poter ispezionare il contenuto di qualsiasi file (sempre che ciò sia funzionale alla gestione della macchina).

Abbiamo visto che per usare un PC bisogna essere preventivamente registrati. In pratica chi materialmente si incarica di gestire il sistema operativo stabilisce chi può accedere al PC, assegnando alla persona il cosiddetto "*user name*" ed una password. Immettendo senza errori i due dati -spesso indicati collettivamente come credenziali- l'utente viene a trovarsi in una cartella/directory di cui è il proprietario e dove ha ampia facoltà di utilizzo del PC. Tale posizione è nota come `HOME` dell'utente. A dire il vero, il concetto appartiene al mondo Linux, (dove la `HOME` è addirittura codificata simbolicamente col carattere `~` [tilde]), ma è valido pure in ambito Windows.

Ora si tratta di focalizzare 2 concetti:

- la posizione della `HOME` entro il file system
- il path (talvolta tradotto in italiano come percorso)

In Linux sappiamo che in cima al file system c'è `root`, la "*madre di tutte le directory*", simbolicamente rappresentata dalla barra (`"/"`). Entro questa directory si trova un'altra directory, chiamata `home` (minuscolo) ed è all'interno di quest'ultima che trovano posto le `HOME` (maiuscolo) degli utenti accreditati ad usare il PC. Supposto che sia stato creato lo user `pinco`, la sua `HOME` si trova in:

```
/home/pinco
```

La prima barra (a sinistra) indica la directory `root`. Segue `home` e quindi `pinco`, ovvero il percorso dalla base sino al posto convenuto. Quale separatore tra le varie componenti si usa il carattere `"/"` (barra). Il posto ove collocare la `HOME` degli utenti viene decisa da chi amministra il sistema. Ora va per la maggiore la collocazione appena indicata. Ai primordi era più usuale definire le `HOME` in

```
/usr/user
```

In Windows la situazione si discosta, ma non di molto. In cima al file system si indica il nome simbolico del disco sul quale risiedono i file. Nella stragrande maggioranza dei casi tale disco è individuato dalla lettera C seguita dal carattere `:` (doppio punto). Sino alla versione Windows Vista le `HOME` erano definite entro la cartella `"Documents and Settings"`. Quindi quella dell'utente `pinco` si trovava in

```
C:\Documents and Settings\pinco
```

A partire da Windows 7 le `HOME` sono definite in

```
C:\Users
```

ovvero MicroSoft ha deciso di seguire la falsariga di Linux sostituendo il lungo e non molto esplicativo `"Documents and Settings"` con il più agile ed autoesplicativo `"Users"`. Visivamente ci si accorge subito che in Windows quale elemento di separazione delle componenti il path viene usata la barra rovescia (`"\"`).

Una volta che un utente si è accreditato con successo, si trova nella propria `HOME` e da lì inizia ad operare. Se ad esempio crea un file contenente una foto può optare per chiamarlo `miafoto.jpg`. In realtà il nome *vero* del file è molto più lungo poiché comprende pure la specifica della `HOME`. Il nome completo risente pesantemente del sistema operativo, come evidenziato nella tabella 2 a pag. 8.

Linux	**/home/pinco/**`miafoto.jpg`
Windows	**C:\Users\pinco**`miafoto.jpg`

Tabella 2: Esempi di path

Anzi, se il file fosse stato creato in una sotto cartella, il nome completo avrebbe dovuto comprendere pure lei. La parte che precede il nome vero e

proprio, evidenziata in **grassetto**, è conosciuta col termine inglese di "*path*" (in it.: percorso). In questo caso particolare (partenza dal vertice della piramide), il path è detto *assoluto*. Nel normale utilizzo il path assoluto viene usato raramente (cfr: alla domanda "Dove abiti?" in genere si risponde col solo nome della via. Ma l'indirizzo completo comprende pure la città, la provincia e lo stato). Molto usato invece il path *relativo*, spiegato contestualmente al comando `cd` a pag. 12.

6 Comandi. Obbedisco!

Windows ha scelto un approccio cosiddetto "*user friendly*", ovvero amichevole, per l'utilizzo del proprio sistema operativo. Ai tempi del DOS, l'antenato di Windows, il PC era gestito ed usato digitando i comandi sulla tastiera. Con Windows la svolta epocale. Tutti i dettagli tecnici sono stati accuratamente occultati ed è stata posta la massima cura nel facilitare l'uso dei PC anche da parte di chi era digiuno di informatica. L'uso della tastiera fu quasi abolito perché la digitazione fu trasferita sul mouse.

Linux era nato con scopi ben differenti. L'ambiente ideale era quello universitario/ricerca ove è facile reperire persone con forte esperienza informatica. Ben presto ci si accorse che quel sistema operativo era molto più tosto delle apparenze. Si iniziò ad usarlo anche in ambienti di produzione, malgrado le difficoltà di utilizzo e l'approccio tutt'altro che amichevole. Tendenzialmente l'interazione uomo-macchina è rimasta immutata dai primordi. I comandi alla macchina devono essere scritti. Ci vollero anni per dotare Linux di un'interfaccia grafica *á la Windows*. Oggi si può dire che il distacco è stato colmato. I due modi operativi però coesistono perché non sono mutuamente esclusivi. I comandi altro non sono che programmi che svolgono le operazioni più comuni. Quelle che normalmente si compiono a suon di mouse e click. L'arsenale dei comandi è ingente ed è necessaria una conoscenza la più approfondita possibile. Cosa che richiede tempo perché si acquista con la pratica.

Paradossalmente è Windows che sta facendo una cauta retromarcia riabilitando l'era DOS. Non ostante gli sforzi profusi, certe operazioni non possono proprio essere effettuate per via esclusivamente grafica.

In Linux i comandi hanno tutti quanti la struttura mostrata nella figura 1 a pag. 10.

Il nome del comando è sempre molto corto e molte volte è astruso (critica meritata). Ciò detto, la posizione delle altre componenti non è fissa.

Figura 1: Anatomia di un comando

Le opzioni sono facoltative e rappresentano una variante del comportamento. Alle volte richiedono dei parametri supplementari. Per uno stesso comando si possono specificare più opzioni diverse.

La terza parte, genericamente indicata come `parametro`, talvolta manca del tutto, oppure può essere presente più volte. Dipende dalla sintassi del comando. Il più delle volte il parametro è il nome di un file oppure di una directory.

Ritorniamo sulle opzioni. Agli inizi erano poche. Successivamente sono aumentare a dismisura, tanto da far guadagnare a Linux il poco lusinghiero giudizio di essere diventato grasso. Ai primordi la caratteristica delle opzioni era quella di iniziare col carattere `-` (trattino) e la lunghezza era limitata ad 1 solo carattere. Visto che il loro numero è aumentato considerevolmente l'unicità del carattere è stata rivista. Sono state approntate nuove opzioni, dalla sintassi leggermente differente. Iniziano con 2 trattini e sono lunghe a piacere: ciò che si è perso in snellezza lo si è guadagnato in comprensione. È stato fatto un lavoro, parziale purtroppo, di standardizzazione. Alle vecchie opzioni (mantenute per garantire la "*backward compatibility*", ovvero la compatibilità col pregresso) sono state affiancate delle equivalenti, più lunghe.

Vediamo di mettere assieme i vari tasselli. Per ottenere l'elenco dei file e delle cartelle presenti nella directory in cui ci si trova, il comando è `ls`. Il comando nudo e crudo fornisce un elenco che è troppo spartano per essere realmente utile. Per ottenere delle informazioni circa la tipologia dei file si usa l'opzione `-F` (opzione breve, piuttosto criptica) oppure `--explain` (opzione lunga, autoesplicativa). Siccome il comando ha un sacco ed una sporta di opzioni, per avere una panoramica completa si usa il comando `ls --help` (opzione solo in formato lungo).

Abbiamo visto che l'uso di Linux si estrinseca nel sottomettergli dei comandi ed osservare le eventuali risposte. Il PC, dopo l'accensione, richiede l'immissione delle credenziali e poco dopo è pronto per il "*colloquio*". L'esatto momento è dato dalla comparsa del cosiddetto "*prompt*": una stringa di caratteri, ampiamente configurabile sia dall'amministratore che dall'utente stesso. Il consiglio è quello di contenerla entro limiti modesti.

Il comando digitato viene preso in carico dalla "*shell*", altrimenti nota come "*interprete dei comandi*". La shell esegue eventualmente dei compiti preliminari per poi passare il comando al sistema operativo. Sarà lui ad eseguire quanto richiesto.

I comandi di norma vengono immessi una alla volta. Ma esistono alcune varianti:

- `comando`$_1$`;` `comando`$_2$ più comandi vengono immessi in un'unica infornata, separandoli col carattere `;`. L'esecuzione è sequenziale. Dopo aver completato l'esecuzione un comando, viene eseguito quello successivo
- `comando`$_1$ `&&` `comando`$_2$ `comando`$_2$ viene eseguito solo se l'esecuzione del precedente (`comando`$_1$) è andata a buon fine (esecuzione condizionata)
- `comando&` l'esecuzione parte in "*background*" ma non c'è l'attesa della fine dell'esecuzione. Il prompt riappare subito e quindi il sistema è pronto per accettare un nuovo comando. L'esecuzione del comando continua per conto proprio. Fare attenzione a ciò che si sottopone in background. Se il comando emette dell'output questo appare sullo schermo, magari intralciando altre operazioni.

Chi paventa l'idea di dover trasformarsi in una specie di novello scrivano, sappia che non è proprio così. Linux mantiene la storia dei comandi pregressi per cui è semplice richiamarne uno già digitato. Ma c'è di più. Oltre al richiamo semplice, è possibile il richiamo cui viene aggiunta una modifica. Le scorciatoie non vengono qui spiegate, perché si andrebbe fuori tema.

7 Operativamente

Vengono qui indicati alcuni comandi Linux. Grosso modo all'inizio si trovano quelli più usati (scelta personale). Di tanto in tanto sono presenti degli approfondimenti in modo da poter giostrarsi in maniera più proficua.

mkdir il comando crea una nuova directory nella cartella in cui ci si trova. Quindi `mkdir` esige un parametro, ossia il nome della nuova directory. Contestualmente alla creazione della directory, questa viene popolata con 2 file speciali la cui gestione è demandata al sistema operativo:

- `.` un file il cui nome è formato da 1 solo punto[3]. Rappresenta simbolicamente l'attuale directory

[3]I file il cui nome inizia con il carattere `.` sono noti con la locuzione inglese di *dot files*

- **..** un file il cui nome è formato da due punti. Rappresenta simbolicamente la directory madre

Un piccolo, ma utile, approfondimento. Il comando

```
mkdir uno/due
```

potrebbe non funzionare. L'intento probabilmente è quello di creare la cartella `due` all'interno della cartella `uno`. Il comando funziona a patto che quest'ultima sia esistente. Se non lo fosse ci sono due possibilità:

- creare la cartella `uno`; quindi spostarsi in questa e creare la cartella `due` (metodo laborioso)
- usare il comando `mkdir -p uno/due`. L'opzione `-p` [per **p**arents] istruisce il comando a creare le directory intermedie nel caso non esistano (metodo Linux doc, ove la concisione è un pregio)

cd è il comando per cambiare directory. Esige un parametro: il nome della directory verso cui ci si deve dirigere. Però vedremo che esiste un eccezione abbastanza importante. Apparentemente il comando appare banale. Ma non è così. Creare delle cartelle presuppone poi che ci si debba andare dentro e successivamente uscirne fuori. Questi movimenti hanno un vincolo: sono ammessi solo quelli "*verticali*". Supponiamo di avere la struttura di cartelle mostrata nella figura 2 a pag. 12.

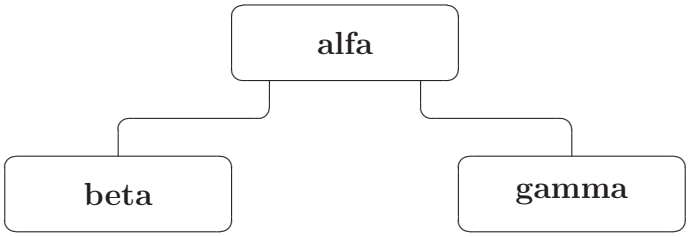

Figura 2: Cartelle

Ci si trova in `alfa`. Da qui ci si può spostare indifferentemente in `beta` oppure in `gamma` (spostamento verticale). Si decide di spostarsi in `beta`. Poiché non ci sono altre cartelle sottostanti verso cui dirigersi, l'unico movimento possibile è ritornare in `alfa` (spostamento verticale). Da `beta` lo spostamento diretto in `gamma` non è possibile. Ci si deve spostare in `alfa` (spostamento verticale) e da qui in `gamma` (idem). All'inizio tutto ciò può apparire complicato, ma basta prendere confidenza e non ci si pensa sopra. Nella figura 3 a pag. 13 sono indicati i percorsi possibili e quelli non ammessi.
Come detto poc'anzi il comando `cd` riserva delle particolarità. Vediamole:

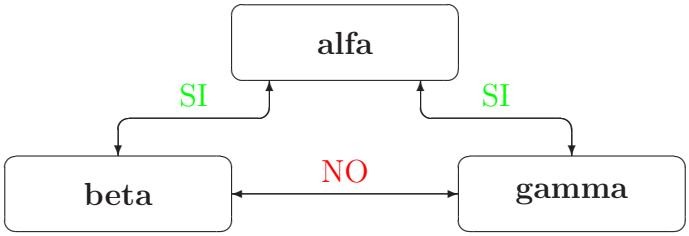

Figura 3: Percorsi `cd`

- `cd` senza opzioni né destinazione riporta alla `HOME`
- `cd ..` permette di risalire alla directory madre (si ritorna al livello immediatamente superiore di annidamento). Quindi, con riferimento allo schema precedente, per lo spostamento dalla cartella `beta` alla `gamma` il comando è `cd ../gamma`
- `cd -` permette di ritornare alla posizione precedente. Un esempio per chiarire il funzionamento. Sempre facendo riferimento allo schema nella figura 2 a pag. 12, ci si trova nella cartella `beta`. Per spostarsi in `gamma` si usa il comando `cd ../gamma`, come spiegato in precedenza. Essendo ora in questa cartella col comando `cd -` si ritorna da dove si era partiti, ovvero nella cartella `beta`. Se qui si dà il comando `cd -`, si ritorna al punto di partenza, ovvero in `gamma`. Dovendo lavorare alternativamente in due directory, il comando risulta molto comodo

pwd è un comando molto utile. `pwd` sta per **p**rint **w**orking **d**irectory. In altre parole il comando stampa la directory (nome completo di path) in cui ci si trova. Utile dopo che si è andati un po' a zonzo tra le varie directory e ci si è persi. Non ci sono opzioni.

clear pulisce lo schermo. Di tanto in tanto un reset dello schermo non guasta

rmdir è l'opposto di `mkdir`. Serve a rimuovere una directory. Quindi `rmdir` esige un parametro, ossia la directory da eliminare. Che però deve essere vuota. Attenzione: i due dot files `.` e `..` non sono affatto eliminabili. Quindi per vuota si intende quella directory popolata solo con quei soli due file

ls un comando usatissimo. Fornisce l'elenco dei file presenti in una cartella. L'ordine è alfabetico. Ma, con uso sapiente delle opzioni, fornisce pure informazioni dettagliate su un gruppo di file (gruppo che può essere costituito anche da 1 solo file). Una breve panoramica delle possibilità:

- `ls` fornisce l'elenco dei file e directory presenti nella directory in cui ci si trova. L'elenco è costituito dai soli nomi (senza distinzione tra le varie tipologie) e sono esclusi i dot file. Se si indica una directory, allora l'elenco si riferisce a quella directory. Il path può essere sia relativo alla directory presente che assoluto

- `ls -l` fornisce informazioni molto dettagliate riguardo i file della directory presente. Si può indicare una cartella, con path relativo oppure assoluto. In tal caso le informazioni riguardano i file presenti riguardano quella directory. Si può indicare 1 solo file, oppure un gruppo di file usando il carattere `*` (asterisco). Nulla vieta di combinare directory e nomi di file (path relativo).
  Il carattere asterisco è una specie di jolly. Tecnicamente sta per *qualsiasi carattere presente da zero ad infinite volte*. Indicare un file tramite ad es. `"alfa*"` significa indicare collettivamente ma in forma concisa tutti i file che iniziano con la stringa `"alfa"`. Analogamente indicare un file tramite ad es. `"*.pdf"` significa indicare collettivamente tutti i file che finiscono la stringa `".pdf"`. Ed infine `"foto*.jpg"` è un metodo conciso per indicare tutti i file che iniziano con `"foto"` e terminano con `".jpg"`. In Windows esiste un altro carattere jolly, il `%` (percento). Ha una funzione abbastanza simile all'asterisco: sta per 1 solo carattere. In Linux non esiste, ma solo perché vengono messe a disposizione le *regular expressions*, ben più potenti e di cui fa parte l'asterisco.

- `ls -t` elenca i file in base alla data di ultima modifica (in testa i più giovani). Aggiungere `-r` (per reverse) per cambiare il verso dell'ordinamento

- `ls -S` elenca i file in base alla dimensione (in testa i più piccolini). Anche in questo caso aggiungere `-r` per cambiare il verso dell'ordinamento

cat serve per stampare il contenuto di un file. È necessario fornire un file, in una qualsiasi delle forme possibili. `cat` da solo non risulta particolarmente adatto con file piuttosto lunghi. L'output infatti viene scritto a video con continuità e quando non c'è più spazio per le nuove righe, questo viene creato facendo uscire dallo schermo le più vecchie (`"`*scrolling*`"`). Se il file che si intende ispezionare contiene molte righe bianche adiacenti, è possibile compattare l'output stampando 1 sola riga bianca. Basta usare l'apposita opzione `-s` (per squeeze [= comprimi])

less serve per stampare un file a videate. Rispetto a cat che funziona in maniera spiccia, `less` tiene conto del numero delle righe a disposizione. Una

volta riempita una schermata l'output si ferma, consentendo una comoda visione del contenuto. Per procedere alla videata successiva basta battere la barra spaziatrice. `less` però è stato progettato con lungimiranza. Se si sta visualizzando una videata che non è la prima, è possibile rivedere qualsiasi videata precedente (premendo il tasto **b** [per back] si ritorna indietro di 1 videata). `less` ha un numero sorprendentemente alto di comandi interni (attenzione a non confondere le opzioni del comando con i comandi interni). Per una panoramica di questi: `less -H`

rm il comando rimuove un file. Tecnicamente, i file etichettati come directory contengono l'elenco dei file presenti. `rm` agisce su tale elenco, togliendo il file che deve essere fornito obbligatoriamente. Non si tratta di una eliminazione vera e propria però, a differenza di Windows, non è possibile ripristinare un file cancellato per sbaglio. Alcune opzioni particolarmente importanti:

- `-i` prima di procedere all'eliminazione chiede una conferma esplicita: **y** (per yes). Qualsiasi tasto diverso da **y** è considerato come no
- `-r` eliminazione ricorsiva. Il tal caso al comando viene passata una directory. Il comando elimina tutti i file ed alla fine cancella la directory. Usare congiuntamente le opzioni `-i` e `-r` nel rimuovere una directory con molti file può risultare frustrante
- `-v` conferma l'avvenuta cancellazione

shred il comando rimuove un file. Con `shred`, introdotto solo in tempi molto recenti, il file viene sovrascritto in maniera opportuna alcune volte in modo da ottenere la cancellazione del contenuto. Visto che il file viene riscritto più volte, il comando, se applicato a file piuttosto corposi, richiede molto tempo per portare a termine l'operazione.
Se un file **deve** essere distrutto con certezza, si usano le opzioni

- `-n 35` se l'opzione non è usata il file viene sovrascritto solo 3 volte (troppo poche). Al posto di 35 -numero di tutta sicurezza- si può usare un altro valore
- `-f` forza l'eliminazione del file (`shred` provvede alla sola sovrascrittura)
- `-z` l'ultima sovrascrittura consiste nell'azzeramento (per mascherare l'operazione)

cp copia un file. Richiede quindi 2 file. Il primo è il file da copiare. Il secondo può essere:

- un nome di file. Il contenuto del primo file viene copiato nel secondo che se esiste viene sovrascritto. Attenzione a non modificare l'estensione
- un nome di directory. Il file viene copiato nella nuova destinazione mantenendo il suo nome

Alcune opzioni importanti:

- -i la copiatura è potenzialmente pericolosa. Con questa opzione prima di distruggere un file viene richiesta una conferma esplicita: y (per yes). Qualsiasi altro tasto vale per no
- -v conferma l'avvenuta copia
- -r richiede non due file, ma due directory. Copia ricorsivamente i file presenti nella prima nella cartella in quella di destinazione
- -p ai file copiati viene imposta come data ed ora quelle presenti in macchina al momento dell'operazione. Ciò potrebbe non essere gradito. Con quest'opzione le caratteristiche amministrative del file, quindi le date ma non solo quelle, vengono mantenute

mv cambia il nome del file. Ma è un comando un po' ibrido perchè assomma in sé 2 funzioni. Esige il file originale ed il nuovo nome. Se esiste già un file con questo nome, viene sovrascritto (*rename*). Se il secondo nome è una directory, allora il file viene semplicemente spostato mantenendo il nome (*move*). Alcune opzioni particolarmente importanti:

- -i l'operazione è potenzialmente pericolosa. Con questa opzione prima di distruggere un file viene richiesta una conferma esplicita: **y** (per yes). Qualsiasi altra risposta vale per no
- -v conferma l'avvenuta operazione

date fornisce sia la data che l'ora desumendole dall'orologio del PC. Il comando può impostare una nuova data/ora purché si disponga dei privilegi di superuser. Tramite i formati, forniti in linea, si possono estrarre le varie componenti (ad es. il nome abbreviato del mese, oppure i secondi) e comporle secondo le proprie esigenze. L'elenco dei formati è smisurato. La tabella 3 a pag. 17 fornisce un estratto di pura sopravvivenza
2 esempi per chiarire l'uso

- `date +%Y` fornisce l'anno (4 cifre)

`%Y`	anno (4 cifre)	`%m`	mese (01..12)	`%d`	giorno (2 cifre)
`%H`	ora (00..23)	`%M`	minuto (00..59)	`%S`	secondo (00..59)
`%A`	nome giorno	`%b`	nome mese (3 caratteri)	`%T`	= `%H:%M:%S`

Tabella 3: Alcune opzioni del comando `date`

- `date + "%A %d %b %Y %T"` stampa nome giorno, numero giorno, nome mese (3 caratteri), anno (4 cifre), ora (con le componenti separate dal carattere `:`)

In Windows spesso si usa la locuzione "*file di testo*". I file di questo tipo sono quelli creati attraverso il block-notes. Se ad esempio inserisco in un file vuoto la stringa aperto tramite un programma di video scrittura la dimensione del file è di gran lunga maggiore. In Linux la locuzione non ha alcun significato. I file sono file, e basta. Va da sé che se ispeziono un file di testo riesco a leggere il contenuto senza difficoltà. Ma nulla vieta di sbirciare dentro un'app, sapendo a cosa si può andare incontro.

head serve per visualizzare le prime 10 righe di un file. Il comando quindi richiede la presenza di un file che può essere espresso nelle forme già viste (uso dell'asterisco; presenza del path). Se si desidera variare il numero di righe allora il comando è

```
head -n num file
```

ove `num` è il numero desiderato (l'opzione `-n` è vecchio stampo; l'equivalente opzione lunga è `--lines=)`

tail serve per visualizzare le ultime 10 righe di un file. Al pari di `head`, anche `tail` richiede la presenza di un file, esprimibile in tutte le forme già viste. Se si desidera modificare il numero di righe, si ricorre ad un'opzione, analoga a quella vista per il comando `head`

```
tail -n num file
```

Questo comando prevede anche un'altra possibilità. Quella di visualizzare il contenuto di un file a partire da un numero di linea fornito dall'utilizzatore (con le linee numerate a partire da 1). Porre quindi attenzione alla sintassi, come mostrato dalla tabella 4 a pag. 18

In Linux è pratica comune costituire delle catene di comandi (note col termine di "*pipe*"). Il meccanismo prevede che l'output di un comando venga

`tail -n num file`	stampa le ultime `num` righe
`tail -n +num file`	stampa a partire dalla riga `num`

Tabella 4: I due usi del comando `tail`

passato come input al comando seguente. Per indicare che due comandi sono collegati in una pipe si usa il simbolo `"|"` (la barra verticale, senza gli apici). Quindi:

$$\texttt{comando}_1 \texttt{ | } \texttt{comando}_2$$

Ovviamente la pipe può collegare tra loro un numero arbitrario di comandi.

Il meccanismo delle pipe accoppiato ai comandi `head` e `tail` permette di visualizzare una porzione intermedia di un file. Supponiamo di voler visualizzare 8 righe del file `alfa`, saltando però le prime 5. La frase "saltando le prime 5 righe" viene riformulata nell'equivalente "a partire dalla riga 6". Ora siamo a posto e la pipe può essere scritta

```
tail -n +6 alfa | head -n 8
```

che si interpreta nel seguente modo: `tail -n +6 alfa` fornisce come output il contenuto del file `alfa` a partire dalla riga 6 (= salto delle prime 5 righe). Tale output viene passato come input al comando seguente `head -n 8` che legge ciò che proviene dal comando precedente e stampa a video le prime 8 righe.

tac serve per stampare un file in ordine inverso. Ovvero l'ultima riga viene stampata per prima, la penultima come seconda e così via. Basta poco per accorgersi che `tac` è `cat` (già visto) scritto alla rovescia

rev serve per stampare un file in maniera speculare. Le righe mantengono la posizione ma ciascuna viene stampata al contrario: l'ultimo carattere viene stampato per primo, il penultimo per secondo e così via. Anche in questo caso con una pipe è possibile stampare un file alla rovescia (l'ultimo carattere del file per primo, il penultimo per secondo e così via). La pipe è:

```
tac file | rev
```

Il primo comando stampa le righe in ordine inverso (l'ultima per prima) e passa l'output a `rev` che a sua volta stampa ciascuna riga stampando l'ultimo

carattere per primo, etc

touch serve per creare oppure per "*aggiornare*" un file. In Linux, ma pure in Windows, ciascun file è caratterizzato dalla presenza di 3 date: quella di creazione, quella di ultimo accesso e quella di ultima modifica. Delle 3 l'ultima è indubbiamente quella più importante. Il comando `touch` richiede la presenza di un file. Se questo esiste, viene aggiornata la data di ultima modifica usando l'ora e la data desumendole dal PC. Se il file non esiste, viene creato. È possibile un'ulteriore variante:

$$\texttt{touch -r file}_1\ \texttt{file}_2$$

Le date di $\texttt{file}_1$ vengono impostate a $\texttt{file}_2$.

wc è una sigla che sta per **w**ord **c**ount (*Honi soit qui mal y pense*). Spesso di un file interessano conoscere alcune caratteristiche che si potrebbero ottenere laboriosamente per altra via. `wc` fornisce la dimensione di un file in caratteri, il numero delle righe ed il numero delle parole. Rimane da definire l'entità chiamata "*parola*": una stringa di caratteri delimitata da caratteri spazio, tabulatori oppure dal carattere di fine riga. Al comando deve essere fornito un file singolo oppure un gruppo di file. In quest'ultimo caso, vengono stampati non solo i contatori per ciascun file, ma viene fornita pure la somma (sempre per ciascun contatore). Queste le opzioni:

- `-l` fornisce il solo contatore delle righe
- `-c` fornisce il solo contatore dei caratteri (ma la dimensione del file si ottiene anche per altra via)
- `-w` fornisce il solo contatore delle parole

diff confronto tra file (benché si operi in ambiente Linux, è sconsigliabile usare il comando con file "non di testo").
Talvolta interessa sapere se due file sono uguali o meno. `diff` confronta due file sulla base delle righe e se non stampa nulla significa che i file sono identici. Altrimenti stampa le differenze con l'output fortemente influenzato dalla pletora di opzioni. Il problema è stato sorprendentemente molto sentito tanto da aver indotto molti a cimentarsi. I comandi per il confronto si contano a dozzine e molti evidenziano le differenze in maniera grafica, a colori. Un'altra caratteristica che riguarda buona parte dei comandi è quella di fornire le differenze in un formato tale che, se applicate ad uno dei due file, lo rende uguale all'altro. Attenzione a come si indicano i file. Nella tabella 5 a pag. 20 sono indicate tutte le combinazioni possibili.
Conviene usare 3 opzioni:

`diff` $\texttt{file}_1$ $\texttt{file}_2$	confronta i due file
`diff dir file`	confronta `file` con `dir/file`
`diff file dir`	idem
`diff` $\texttt{dir}_1$ $\texttt{dir}_2$	confronta ricorsivamente i file nelle due directory

Tabella 5: Il comando `diff`

- `-B` ignora le differenze dovute solo ed esclusivamente ad un differente numero di righe bianche
- `-y` stampa entrambe le righe differenti
- `--suppress-common-lines` autoesplicativo

od è una sigla che sta per **o**ctal **d**ump. Il file viene stampato carattere per carattere. Quindi il comando risulta molto utile per ispezionare file testuali che contengono caratteri non stampabili. È indispensabile per ispezionare file non di testo. L'output è influenzato dal formato

- `-a` i caratteri vengono stampati 1 ad 1. Se non stampabili sono sostituiti dal carattere `.` (punto). Lo spazio e le "*funzioni ASCII*" vengono stampati in maniera simbolica (`sp` per lo spazio; `nl` per fine riga e via discorrendo)
- `-b` il valore dei singoli caratteri viene stampato in formato ottale

Due utili opzioni:

- `-N num` limita l'ispezione al numero di caratteri indicato
- `-j num` inizia l'ispezione a partire dal carattere successivo a quello indicato

In Linux, ma anche in Windows, si parla di "*ridirezione*" dell'output. Nella maggior parte dei casi l'output di un comando viene inviato sullo schermo del PC. Ma alle volte c'è la necessità di salvare l'output di un comando in un file. Da qui la locuzione di ridirezione dell'output: dallo schermo ad un file. Per indicare la ridirezione si usa il carattere `">"` (senza gli apici doppi). Porre attenzione al file su cui dirottare l'output. Se il file non esiste, non ci sono problemi. Viene creato. Se invece esiste, il comportamento non è univoco. L'amministratore del sistema potrebbe aver arrangiato le cose in modo da

bloccare la scrittura a salvaguardia dei file presenti; ma questa è solo una possibilità, non un obbligo.

Accanto alla ridirezione esiste l'"*accodamento*". Ovvero si dirige l'output verso un file. Se esiste la scrittura avviene in coda. Se invece il file non esiste, viene creato. Si usa il carattere `">>"`.

La ridirezione esiste pure per l'input, anche se è molto meno usata. In particolari situazioni, anziché fornire i dati da tastiera, si forza il comando a leggerli da un file. Si usa il carattere `"<"`.
Nella tabella 6 a pag. 21 sono riportate in maniera sinottica le possibilità:

`comando > file`	output su file; comportamento incerto
`comando >| file`	output su file; se file esiste, viene prima rimosso
`comando >> file`	output su file; accodamento, con eventuale creazione
`comando < file`	input da file

Tabella 6: Le ridirezioni

Nulla vieta di usare entrambe le ridirezioni (l'ordine è ininfluente):

`comando < file`$_{in}$ `> file`$_{out}$

tr esegue delle trasformazioni sui caratteri di un file. Il comando è stato congegnato per ricevere i dati in ingresso dalla tastiera mentre l'output va sullo schermo. Cosa che obbliga ad appoggiarsi alla ridirezione. È un comando poliedrico di cui è difficile riassumere unitariamente lo scopo. Qui di seguito vengono indicati alcuni usi abbastanza tipici:

- `tr "[:upper:]" "[:lower:]" < file`$_{in}$`> file`$_{out}$ trasforma le lettere maiuscole in minuscole. Invertendo di posto `upper` e `lower` si ottiene l'effetto opposto
- `tr -d "†" < file`$_{in}$ `> file`$_{out}$ tutte le occorrenze del carattere indicato vengono rimosse
- `tr -s "†" < file`$_{in}$ `> file`$_{out}$ le occorrenze consecutive del carattere indicato vengono collassate ad 1 sola
- `tr "†" "‡" < file`$_{in}$ `> file`$_{out}$ sostituisci le occorrenze del carattere "†" col carattere "‡"

paste unisce due file riga a riga: la prima riga dell'output risulta formata dalla concatenazione delle prime righe dei due file. E con le righe successive si procede con lo stesso metodo. Conviene controllare preventivamente che i due file abbiano lo stesso numero di righe. Tra le due entità che costituiscono le linee dell'output viene posto per default il tabulatore orizzontale. Ma con l'opzione `-d†` il tabulatore viene sostituito dal carattere indicato. Opzione e carattere sono contigui.

split spezza un file in tanti file di dimensioni minori. Intuitivamente le difficoltà di gestione di un file sono proporzionali alla sua dimensione. Il nome dei file generati è formato da una parte fissa (prefisso) seguito da un progressivo numerico. `split` spezza il file originale in tanti pezzi determinando sia il prefisso sia la dimensione. Il nome del file da spezzare ed il prefisso devono essere obbligatoriamente specificati nell'ordine indicato. Le opzioni determinano la dimensione dei pezzi

- `-b num` ciascun pezzo ha la dimensione di `num` caratteri
- `-l num` ciascun pezzo è formato da `num` righe

`split` e `cat` sono speculari. Qui vengono riportati i comandi per frammentare un file (originale) e poi ricrearlo a partire dalle "*briciole*" ciascuna delle quali lunga 1 Kbyte (ma l'ultima può avere una dimensione inferiore) ed il cui nome inizia con la stringa `segmento`

```
split -b 1024 originale segmento
cat segmento* >| originale
```

cut estrae informazioni dalle righe di un file. Attenzione a non confondere questo comando con `cat`. Il comando parte dal presupposto che ciascuna riga del file (il cui nome deve essere fornito) sia composta da informazioni delimitate da un ben determinato carattere. La riga quindi risulta essere una successione di "*campi*" delimitati. I caratteri compresi tra l'inizio della riga e la prima occorrenza del carattere costituiscono il primo campo. Quelli compresi tra la prima e la seconda occorrenza costituiscono il secondo campo e così via.
Queste le opzioni, tenendo conto che in un caso -che verrà fatto notare- un'opzione sarà obbligatoria

- `-d†` specifica il carattere di delimitazione (per default è il tabulatore orizzontale). Opzione e carattere sono contigui

- `-f num` opzione **obbligatoria**. Indica quale campo visualizzare (la numerazione parte da 1). Il campo può essere uno solo oppure un gruppo di campi contigui (sintassi: `-f primo-ultimo`). Per motivi didattici è stato lasciato uno spazio tra l'opzione ed il numero. Nella realtà tale spazio va tolto

Il comando va utilizzato in maniera parsimoniosa in quanto richiede cospicue risorse computazionali. `awk` (comando che qui non viene spiegato) costituisce una valida alternativa specialmente se c'è da estrarre un solo campo.

sort ordina le linee di un file. Il comando richiede che sia specificato almeno un file. Se sono specificati più file, alla fine si ottiene sempre 1 solo file (questa fusione di più file in uno solo è nota col termine di "*merge*"). Senza alcuna opzione le righe vengono ordinate in maniera ascendente, sono considerate nella loro interezza e trattate come stringhe di caratteri. Se il file contiene quantità numeriche bisogna intervenire con un'opzione apposita altrimenti l'ordinamento risulta fasullo. Si pensi ad un file composto da due righe che contengano rispettivamente 5 e 16. L'ordinamento antepone 16 a 5 perché il primo carattere di una riga è 1 mentre il primo carattere dell'altra riga è 5 ed 1 è inferiore a 5. Talvolta è necessario effettuare l'ordinamento basandosi non sull'intera riga ma su una porzione (detta "*chiave*"). Per individuare con precisione la chiave si indica il carattere che la delimita (e che non fa parte della chiave stessa). La riga risulta quindi formata da una serie di campi delimitati. La numerazione inizia da 1. Alcune opzioni:

- `-r` rovescia l'ordinamento (per default è ascendente)
- `-n` ordinamento numerico (per default è alfabetico)
- `-u` in caso di righe uguali ripetute multiple volte, stampa la riga 1 sola volta
- `-t`† il carattere indicato funge da delimitatore delle chiavi (per default è a scelta tra spazio e tabulatore orizzontale). Opzione e carattere sono contigui
- `--key=inizio[-fine]` indica il campo che contiene la chiave (che può estendersi per più di un campo)
- `-M` ordinamento "*mensile*", in base al quale `Jan < .. < Dec`

ln crea un "*link*" (vedi poco più avanti). Ma qui verrà fatta una piccola forzatura. Verrà spiegato il comando `ln -s` che crea un link simbolico. Non

confondere questo comando con `ls`.
Talvolta è necessario oppure desiderabile poter riferirsi ad un file usando un nome diverso, oppure fare in modo che un'app sia usabile più facilmente sollevando l'utilizzatore dal tedio di dover battere sulla tastiera il path completo. Il comando richiede quindi 2 parametri: il nome originale del file e l'alias. In Linux quest'ultimo è noto col termine di "*link*". Tecnicamente si tratta di un file particolare, diverso sia dai file normali che dalle directory. Il concetto di link è presente pure in Windows, ove assume il nome di "*shortcut*" (in it.: collegamento) ed i file hanno estensione ".lnk". Attenzione a non creare i cosiddetti "*dangling links*", ovvero i link mozzi. Se il file originale viene cancellato, questi link rappresentano una situazione potenzialmente pericolosa.

In Linux le caratteristiche amministrative di qualsiasi file e directory sono collocate nell'i-node. Tra queste assumono un particolare interesse i cosiddetti permessi associati al file. Qualsiasi utente afferisce obbligatoriamente ad un gruppo -group in inglese- (che tecnicamente è una stringa di caratteri che si trova in un file del sistema operativo). I file e le directory dell'utente appartengono, per proprietà transitiva, pure al gruppo di cui l'utente fa parte. Dal punto di vista del file, gli utenti possono essere classificati in 3 tipologie differenti, note collettivamente come "*UGO*"

- `U` (per user) è il proprietario
- `G` (per group) sono gli utenti afferenti al medesimo gruppo (gli "*amici*")
- `O` (per other) sono gli utenti che afferiscono agli altri gruppi

Per ciascuna di queste tre categorie sono previste le seguenti possibilità:

- `r` per read: lettura
- `w` per write: scrittura/modifica/cancellazione
- `x` per execute: esecuzione. Per le directory il permesso di esecuzione va interpretato come diritto di accesso. Ma è più corretto usare il termine "*attraversamento*". Una volta entrati in una directory, si può eventualmente proseguire il viaggio entrando in una sotto directory (permessi permettendo)

Nell'i-node esiste una "*maschera*" di 9 posizioni che ha la struttura mostrata nella figura 4 a pag. 25.

user			group			other		
r	**w**	**x**	**r**	**w**	**x**	**r**	**w**	**x**

Figura 4: Maschera permessi

In ciascuna posizione può esserci o uno 0 (zero) che significa possibilità negata oppure un 1 (uno) che significa possibilità concessa. Le 9 cifre binarie -perché vengono usate le cifre 0 e 1- vengono raggruppate in 3 gruppi di 3 e convertite in ottale per facilitare la gestione. La maschera tipica per un file che non sia un'applicazione è 600 (esiste una maschera di default). Tenendo a mente che la cifra 6 trasformata in binario è 110, si ha il prospetto della figura 5 a pag. 25.

user			group			other		
r	**w**	**x**	**r**	**w**	**x**	**r**	**w**	**x**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
6			**0**			**0**		

Figura 5: Maschera permessi

Ovvero: la maschera 600 consente al proprietario di leggere e modificare il file. Ma la frase può essere interpretata anche alla rovescia: concedere al solo proprietario la possibilità di leggere e modificare un file significa imporre una maschera dei permessi (interpretata in base ottale) pari a 600. C'è una particolarità che non va sottovalutata. Tra i 3 meta utenti noti come UGO vige la relazione prospetta nella figura 6 a pag. 26.
Ovvero il proprietario è al contempo anche "amico" ed "other". Gli "amici" sono pure "other". Se per caso ho fatto baruffa con gli amici e per ripicca vorrei togliere solo che a loro il permesso di lettura dei miei file, non ho la possibilità tecnica di farlo. Morale: in Linux dagli amici non ci si può difendere. A meno di non ricorrere alle cosiddette "*ACL*" (**A**ccess **C**ontrol **L**ist) qui non spiegate.

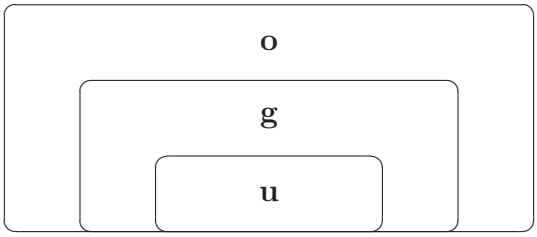

Figura 6: Relazioni tra UGO

chmod permette il cambio dei permessi di una file/directory. Detto preventivamente che per conoscere la maschera dei permessi impostati si usa il comando `ls -l`, `chmod` opera in due modalità

- si imposta ex novo la maschera (modalità *diretta*). Con carta e penna si stabiliscono i sì (1) ed i no (0) trasformando le cifre binarie in un numero ottale di 3 cifre. Quindi

  ```
  chmod maschera file
  ```

- si indicano le differenze da apportare (modalità *differenziale*) secondo lo schema di fig. 7 a pag. 26.

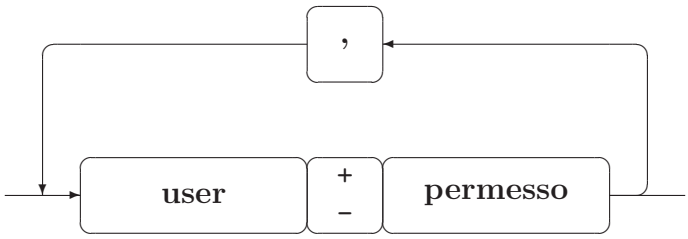

Figura 7: Comando `chattr`: modifica differenziale

user va scelto tra:

- `u` (per user): il proprietario
- `g` (per group): gli utenti afferenti allo stesso gruppo
- `o` (per other): gli utenti afferenti agli altri gruppi
- `a` (per all): notazione concisa per indicare user `+` group `+` other

Il carattere `+` aggiunge il permesso all'attuale maschera. Il carattere `-` lo toglie. Esempio:

```
chmod a+r,o-w file
```

accorda a tutti il premesso in lettura e toglie ad other quello di esecuzione

chattr per cambiare gli attributi di un file. Oltre alle caratteristiche contenute nell'i-node, al file possono essere agganciati degli "*attributi*", ovvero delle caratteristiche di tipo particolare. Il loro numero è insospettabilmente alto. Malgrado il consiglio sia quello di andarci cauti, 2 sono particolarmente utili ed interessanti (tutti gli attributi sono individuati tramite una sola lettera, per lo più mnemonica):

- **i** per "*immutable*". Se attivato, il file non può essere modificato
- **u** per "*undeletable*". Se attivato, il file non può essere cancellato

Per conoscere gli attributi di un file esiste l'apposito comando `lsattr`. Per gestire gli attributi si usa lo schema 7 a pag. 27.

+|-	carattere

Tabella 7: Comando `chattr`

Dove ovviamente `+` serve per attivare e `-` per disattivare l'attributo individuato da carattere (1 sola lettera). Perciò

```
chattr +i file
```

il file non può essere modificato, mentre

```
chattr -u file
```

rende il file cancellabile.

comm evidenzia le parti in comune (in inglese: "*commonalities*") tra due file. Il comando esige la presenza di due nomi di file che devono essere ordinati in maniera coerente. `comm` confronta il contenuto dei due file e stampa a video i risultati. Questi formano 3 colonne: quella a sinistra contiene le righe presenti solo nel 1° file; quella a destra contiene le righe presenti solo nel 2° file; nella centrale si trovano le righe che sono presenti in entrambi i file. Il comando è particolarmente utile quanto si debbano eseguire dei lavori di spunta su due elenchi. Interessanti le opzioni:

- `-1` sopprime la prima colonna (dati solo nel 1° file)
- `-2` sopprime la colonna centrale (dati in comune nei due file)
- `-3` sopprime la colonna di destra (dati solo nel 2° file)

Un'aggiunta riguardante le ridirezioni (già trattate a pag. 20). In Linux 3 file hanno una gestione particolare e talvolta si può far loro riferimento in maniera numerica secondo la tabella 8 a pag. 28.

numero	file	significato
0	input	la tastiera
1	output	lo schermo
2	errori	segnalazioni d'errore

Tabella 8: Approfondimento sulla ridirezione

Si nota immediatamente la presenza di due tipi di output: quello "*normale*" e le segnalazioni d'errore. Per impostazione generale, output ed errori sono agganciati e finiscono entrambi sullo schermo del PC. Ma è possibile modificare l'impostazione e far sì che le segnalazioni siano disgiunte. Si ottiene un output "*pulito*" e gli errori, non più inframmezzati con l'output, risultano meglio evidenziati.

`comando > file`$_{out}$ `2> file`$_{err}$

Il primo simbolo di ridirezione `>` dirige l'output "*normale*" del comando verso `file`$_{out}$ mentre il secondo simbolo di ridirezione `2>` ridirige le segnalazioni d'errore verso un file differente. Il numero che appare davanti al simbolo di ridirezione sta ad indicare a quale file ci si sta riferendo. Il simbolo `>` altro non è che una semplificazione di `1>`. L'ordine delle due ridirezioni non ha alcuna importanza.

Infine esiste l'ultima possibilità: dirottare output ed errori verso un apposito file. Il comando:

`comando > file`$_{out}$

non risolve il problema perché la ridirezione riguarda solo l'output, ma non le segnalazioni d'errore che continuerebbero ad essere stampate a video. Anziché ripetere due volte il nome del file di output, si ricorre ad una simbologia più concisa:

`comando 1> file`$_{out}$ `2>&1`

Nella prima ridirezione (`1>`) l'1 (= output) è stato posto a scopo puramente didattico ma è pleonastico. `2>&1` significa dirottare le segnalazioni d'errore verso lo stesso file dove è stato ridiretto l'output. L'ordine delle due ridirezioni è **importante**.

find per la ricerca di un file. Spesso esiste la necessità di trovare un file. A ciò provvede il comando in questione. Comando con una caterva di opzioni *interne*. Qui verranno illustrate quelle più comunemente usate. Il paradigma d'uso è illustrato nella tabella 9 a pag. 29.

find	dove	cosa	operazioni

Tabella 9: Sintassi comando `find`

Detto che "dove" è obbligatorio, mentre "cosa" ed "operazioni" sono opzionali, il comando si conclude **obbligatoriamente** con i due caratteri `\;`.

Il comando è ricco di opzioni *interne* con le quali si creano delle espressioni. Rispetto agli altri comandi, ove le opzioni possono essere indicate abbastanza liberamente, qui ci sono dei vincoli ben precisi. Le espressioni relative all'ipotetica domanda "dove" non possono essere mischiate a quelle relative all'ipotetica domanda "cosa".

Nel campo "*dove*" si indica la directory da cui iniziare la scansione. Per indicare quella corrente si usa il carattere `.` (punto). Altrimenti si ricorre all'accoppiata path `+` directory. A meno di controindicazioni, la ricerca si estende a tutte le sotto cartelle. Operando in un PC altamente popolato di file non è garantita una risposta rapida. Ciò è tanto più vero per `root`, che può vagare per l'intero PC. Per limitare la "*profondità*" della ricerca si usano, anche disgiuntamente

```
-mindepth num
-maxdepth num
```

`-maxpath` indica di quanti livelli si può scendere. Se si indica 1 la ricerca è limitata alla directory indicata nel comando. `-minpath` invece fa iniziare la ricerca quando si raggiunge il livello di profondità indicato.

Più complesse le espressioni riguardanti "*cosa*" ricercare. Questa parte del comando è opzionale. Se non si indica nulla `find` raccatta tutto ciò che trova: tutti i file, tutte le directory, tutti i link. Non molto utile. Pur non essendo

obbligatorio, il campo andrebbe specificato. Fondamentalmemnte si possono ricercare entità che abbiano determinate caratteristiche quali: tipologia, nome, data di modifica, dimensione e molto altro ancora.

`-type` †

† può essere **f** per file, oppure **d** per directory oppure **l** per link. Se si usa quest'espressione, la ricerca viene limitata alla sola tipologia indicata, altrimenti non si fa distinzione.

`-name nome`

nome è quello del file/cartella/link (definiti d'ora in avanti come "*entità*") che si ricerca. Può essere noto totalmente oppure parzialmente. In quest'ultimo caso si usa il carattere asterisco (già visto) ed il nome va racchiuso tra apici doppi. Proprio perché la conoscenza non è certa, anziché `-name` conviene usare `-iname` che non fa distinzione tra maiuscole e minuscole.

`-size num`

ciò che si ricerca occupa esattamente `num` byte. Ma con `+num` la ricerca è limitata a ciò che supera il limite. Per contrasto, `-num` limita la ricerca ad entità meno invasive.

`-mtime num`

Il tempo delle entità è calcolato nella seguente maniera. Viene calcolato il quoziente intero della divisione per 86400 (numero dei secondi in un giorno) della differenza tra il tempo attuale (quella della ricerca) e quello memorizzato nell'i-node. Con `-mtime 0` vengono individuate le entità modificate sino a 24 ore prima della data di ricerca, in altre parole: 0 [zero] giorni *cronologici*. Ma se la ricerca viene ripetuta dopo un ragionevole lasco di tempo i risultati potrebbero essere differenti. Se si indica solo `num` la ricerca è esatta: il numero di giorni cronologici specificato. Con `+num` si ricercano le entità più "*anziane*"; con -num quelle più "*giovani*".
Con `atime` ci si basa sul tempo di ultimo accesso mentre con `ctime` ci si dovrebbe basare sulla data di creazione (ma la documentazione al riguardo è imprecisa). Come già detto, `find` è ricchissimo di espressioni. Ci si può sbizzarrire a ricercare entità non basandosi sul giorno, bensì sui minuti oppure su entità che siano più vecchie/giovani di altre. Ovvero si possono ricercare file che siano stati creati dopo un file indicato. A mo' di esempio:

```
find . -maxdepth 0 -type f -iname "alfa*" -mtime +1 ...
```

(NB: il comando non è completo). Si ricerca nella directory corrente (`.`), e solo in quella (`-maxdepth 0`), i file (`-type f`) il cui nome inizia con alfa (`-iname alfa*`), senza sottilizzare sul caso delle lettere (perché è stato usato `-iname`) e che siano stati modificati (`-mtime`) almeno due (`+1`) giorni prima.

Ora viene la parte operativa ("*operazioni*") del comando. Questa parte del comando è opzionale. Se non si indica nulla `find` si limita a stampare il nome, comprensivo di path, e solo quello, delle entità trovate. Ma ciò può andar bene solo se ci si limita ad una ricerca pura. Alle entità trovate si possono applicare i comandi visti. Per farlo, l'espressione da usare è `-exec` mentre l'entità viene riferita simbolicamente tramite `{}`. Un esempio per chiarire come stanno le cose. Si desidera applicare a tutte le sottocartelle dell'attuale directory la maschera 750, così da permettere l'attraversamento agli "*amici*". Il comando è:

```
find . -type d -exec chmod 750 {} \;
```