Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2020/2021

Corso di Laurea in Matematica

Linux, bash e comandi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Per cominciare
   - Informazioni iniziali
   - Comandi di base
2. Utenti, filesystem e file
   - Amministrazione degli utenti
   - Il filesystem
   - Comandi per la gestione di file e directory
   - Filesystem e comandi per la gestione
   - Comandi per cercare
   - File in Linux e comandi sui file
   - Comandi per ottenere informazioni dal sistema

Per cominciare

Informazioni iniziali

Terminale e shell

- Per dare comandi dalla riga di comando si può usare il **terminale**
- In questo modo viene utilizzata la **shell**, che funziona appunto su un terminale
- Quando si chiama il terminale viene subito avviata la **shell standard** (ad esempio la *Bourne again shell*, *Bash*)
- La *shell* è un programma che svolge la funzione di interfaccia tra sistema e utente: comprende un interprete della riga di comando, accoglie gli input utente tramite tastiera (cioè nella riga di comando) e li analizza, avvia i programmi e restituisce all'utente il risultato sotto forma di testo
- Ogni shell possiede un proprio linguaggio di programmazione che consente di scrivere *script shell*

Standard input, output ed error

- In Linux si usano i concetti di standard input, output ed error, cioè `stdin, stdout e stederr`
- Quando viene eseguito un comando vengono generati i tre *flussi* (o stream) standard, dove per *flusso* si intende un meccanismo usato per trasferire dati, in questo caso testo
- `stdin` è lo standard input stream e accetta testo come input
- `stdout` è lo standard output stream ed è il testo prodotto in output dal comando eseguito, di solito scritto sullo schermo
- `stderr` è lo standard error stream ed è visualizzato sullo schermo tramite messaggi di errore
- Come per i file, agli standard stream vengono associati degli identificatori: 0 per `stdin`, 1 per `stdout` e 2 per `stederr`

Utenti, filesystem e file

Comandi di base

Comando `man`

- `man [opzione] nomecomando` apre le pagine del manuale (man pages) nel terminale
- Le *man pages* di Linux sono suddivise in 10 tematiche:
  - (1) Comandi utente
  - (2) Collegamenti del sistema
  - (3) Funzioni del linguaggio di programmazione C
  - (4) Formati dei file
  - (5) File di configurazione
  - (6) Giochi
  - (7) Varie
  - (8) Comandi per l'amministrazione del sistema
  - (9) Funzioni del kernel
  - (10) Nuovi comandi
- Ad esempio, sia usando `man clear` che (restringendo la ricerca) usando `man 1 clear`, si apre la pagina del manuale riguardante il comando `clear`

Comandi `whatis` e `apropos`

- `whatis [opzioni] parolachiave` cerca le parole chiave nel manuale (o meglio nel database whatis)
- Se la parola cercata è presente nel manuale, `whatis` ne fornisce una breve descrizione nel terminale
- Sono visualizzate solo le corrispondenze con parole intere
- `apropos stringa` cerca una o più stringhe nel database whatis
- A differenza di whatis, sono visualizzate tutte le corrispondenze
- Ad esempio `apropos keyboard` visualizza le righe del database whatis contenenti la stringa keyboard

Comandi `history` e `clear`

- `history` mostra i comandi eseguiti
- Su Bash vengono memorizzati nella cronologia (*history*) gli ultimi comandi inseriti nella riga di comando (di solito 500)
- Consente di ricercare nella lista dei comandi precedenti con i tasti freccia ed eseguirli di nuovo confermando con il tasto di invio

- `clear` serve a rimuovere il contenuto dello schermo
- Si ottiene un terminale vuoto con aperta solo la finestra della riga di comando
- Gli input immessi precedentemente rimangono comunque memorizzati nello *scrollback buffer*

Comandi `help` e `info`

- `help` mostra una lista dei comandi shell integrati (comandi built-in)
- `help comandoshell` fornisce una descrizione del corrispettivo comando
- Molti comandi accettano anche l'opzione -h (o - -help) che fornisce una breve descrizione sull'utilizzo del comando e delle sue opzioni

- `info comando` fornisce informazioni estese sul comando
- Nella maggior parte dei casi si hanno le informazioni che si possono richiamare tramite man, ma con collegamenti che agevolano la navigazione nel manuale

Utenti, filesystem e file

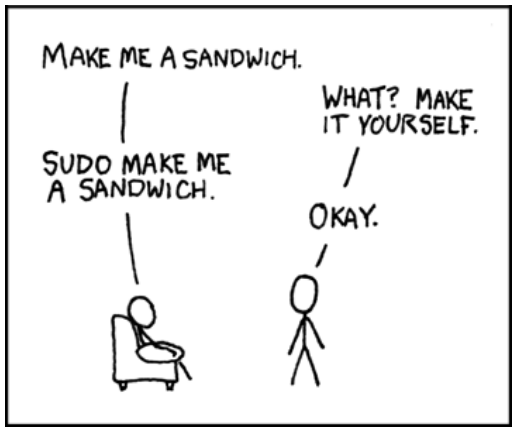
Amministrazione degli utenti

Utenti

- Durante l'installazione di un qualsiasi Linux, è necessario specificare (almeno) un **utente**
  - non tutti gli utenti possono fare login (per esempio, in molti sistemi Linux non può farlo l'utente `root`)
  - l'installazione di Linux crea (almeno una decina di) utenti, che servono solo per scopi interni del sistema operativo
- Ogni utente appartiene ad un **gruppo** (insieme di utenti)
  - viene tipicamente creato un gruppo con lo stesso nome dell'utente *principale* specificato in fase di installazione
- Esistono molti gruppi e uno stesso utente può appartenere a più gruppi: per sapere quali, si può usare il comando `groups [nomeutente]`
  - molti gruppi servono solo a scopi interni al sistema operativo

Utenti

- Nel caso di Ubuntu, l'utente creato a tempo di installazione è un **sudoer**, cioè un *utente con privilegi di amministratore di sistema*, e appartiene al gruppo predefinito `sudo`
  - quindi può eseguire comandi da *superutente* semplicemente preponendo il comando `sudo`, derivato da **s***uper* **u***ser* **do**: per esempio, può installare nuovi pacchetti
  - `sudo comando` è un comando particolare, che prende come argomento un altro comando, che può avere svariati argomenti



Utenti

- È possibile creare un altro utente usando il comando `adduser nuovoutente`
  - di default, questo crea un utente *non* sudoer
  - però il comando va dato da un utente sudoer, che deve anche aggiungere `sudo` all'inizio
  - infatti creare utenti rientra nelle prerogative di un amministratore di sistema

- **Esercizio** creare due nuovi utenti non-sudoer, con nome `utente1` ed `utente2`

Utenti

- È possibile aggiungere un utente già creato ad un gruppo, usando il comando `adduser utente gruppo`
  - dalla pagina di manuale: If called with two non-option arguments, adduser will add an existing user to an existing group.
  - di nuovo, va fatto dall'utente sudoer, con `sudo` davanti
- È possibile cambiare utente usando il comando `su -l nomeutente`

Utenti

- **Esercizio**:
  - Con l'utente sudoer, provare ad eseguire il comando `apt-get update` e il comando `sudo apt-get update`
  - Con l'utente `utente1`, provare ad eseguire `apt-get update` e il comando `sudo apt-get update`
- APT (Advanced Package Tool), sistema di gestione dei pacchetti predefinito in Ubuntu
- Il comando `apt-get update` aggiorna la lista dei pacchetti disponibili dai repository; andrebbe eseguito periodicamente per verificare che la propria lista di pacchetti sia aggiornata
- Il comando `apt-get install` installa il pacchetto specificato

Utenti

- È possibile cambiare o impostare la propria password o la password di un utente usando il comando `passwd`
  - `sudo passwd nuovoutente` consente di impostare la password dell'utente nuovoutente
  - `passwd` consente di cambiare la propria password
- È possibile cancellare un utente usando il comando `deluser utente`
  - di nuovo, va fatto dall'utente sudoer, con `sudo` davanti

Utenti, filesystem e file

Il filesystem

Filesystem in Linux

- Un **filesystem** è un'organizzazione di un'area di memoria (tipicamente di massa, come il disco), basata sul concetto di *file* e di *directory*
  - una directory serve a contenere al suo interno altre directory oppure file
  - induce naturalmente una struttura gerarchica, ad albero, dove ogni nodo è una directory o un file
  - solo le directory possono avere figli
  - i file *regolari* contengono sequenze di bit dell'area di memoria sulla quale c'è il filesystem e possono essere testi, dati, programmi sorgente, eseguibili
  - file *speciali* sono directory, device (dispositivi hardware collegati visti come file), pipe (file per lo scambio di dati sincrono tra due processi concorrenti, link (riferimento ad un altro file o directory)

Filesystem in Linux

- Linux ha un solo filesystem principale, che ha come radice la directory / (cioè la directory *root*)
  - tutti i file e le directory sono contenuti, direttamente o indirettamente, in tale directory
  - le foglie dell'albero possono essere directory vuote oppure file
  - all'interno della stessa directory non ci possono essere due file, due directory oppure un file e una directory con lo stesso nome
  - cambiare le maiuscole/minuscole è sufficiente a distinguere tra due files o directory: `nomeFile` è diverso da `nomefile`

Filesystem in Linux

- Ogni file o directory è raggiungibile dalla directory radice attraverso un *path assoluto*
  - una sequenza di directory separate da slash e avente slash come primo carattere
  - (quindi, il carattere slash non può essere usato per dare un nome ad una directory o ad un file)
  - esempio `/home/utente1/dir1/dir3/dir7/file.png`
  - come parziale eccezione, è un path assoluto anche quello che comincia con una tilde $\sim$
  - infatti, come vedremo, la tilde è una scorciatoia per la directory home dell'utente corrente x: `/home/`x

Directory principali

- Alcune delle principali directory e loro utilizzo su Ubuntu
  - `/bin` Contiene i programmi basilari per la gestione del sistema, cioè buona parte dei comandi base utilizzabili dalla riga di comando da qualsiasi utente senza dover utilizzare i privilegi dell'amministratore
  - `/boot` Contiene le immagini del kernel e i file indispensabili al bootstrap del sistema
  - `/dev` È la directory che individua sotto forma di file le periferiche hardware
  - `/etc` Contiene i file di configurazione del sistema. Ad esempio `/etc/apt` file di configurazione dei repository
  - `/home` Contiene tutte le directory personali degli utenti del sistema
  - `/lib` Contiene tutte le librerie condivise del sistema

Directory principali

- Alcune delle principali directory e loro utilizzo su Ubuntu
  - `/root` Contiene la directory home dell'amministratore del sistema ed è esplorabile solo utilizzando i privilegi da super utente. Il contenuto è analogo a quello delle singole home-utente descritte nel capitolo della directory /home
  - `/tmp` Contiene file temporanei
  - `/usr` È la directory che contiene la maggior parte dei programmi installati sul sistema
    - `/usr/bin` file eseguibili delle applicazioni accessibili a tutti gli utenti, cioè i programmi normalmente avviabili dal menù delle applicazioni.
    - `/usr/sbin` file eseguibili delle applicazioni di sistema accessibili solo all'amministratore
    - `/usr/share` file di vario genere (configurazione, documenti di testo, ecc..) indipendenti dall'architettura del sistema (i386, amd64). Ad esempio, le cartelle backgrounds, icons e themes contengono sfondi, icone e temi del desktop

Directory

- Concetto di *current working directory* (cwd)
  - vale per ogni processo, quindi anche per le shell, che la mostrano nel prompt
  - per sapere qual è la cwd, usare il comando `pwd`
  - per cambiare la cwd, usare il comando `cd [path]` (se non si specifica il `path`, la nuova directory sarà la home)
  - all'interno di `path` può essere usato
    - sia `..` (directory *parent*, che contiene quella attuale; se fatto sulla root, ritorna la stessa root),
    - oppure anche `.` (la directory stessa)
    - il path `/home/utente1/dir1/dir3/dir7/file.png` può essere equivalentemente scritto, ad esempio, `/home/./utente1/dir1/./dir3/dir7/file.png` oppure `/home/utente1/../utente1/dir1/dir3/dir7/file.png` oppure `/home/utente1/dir1/../dir1/dir3/dir7/file.png`
- **Esercizio:** posizionarsi nella directory `/lib` e controllare come cambia il path nel prompt

Directory

- A partire dalla cwd, si possono usare i *path relativi*
  - sono quelli *non* assoluti; pertanto, non cominciano con uno slash (né con la tilde)
  - per esempio: se la cwd è la home dell'utente `utente1`, allora lo stesso file di sopra è raggiungibile con il path relativo `dir1/dir3/dir7/file.png`, o anche `./dir1/dir3/dir7/file.png`, o anche `../utente1/dir1/dir3/dir7/file.png`
  - a seguito di un `cd dir1/dir3` (o equivalentemente, `cd /home/utente1/dir1/dir3/`), lo stesso file di sopra è raggiungibile con il path relativo `dir7/file.png`, o anche `./dir7/file.png`, o anche `../dir3/dir7/file.png`
- La differenza tra un path assoluto ed uno relativo sta nel fatto che il path assoluto è valido qualsiasi sia la cwd, mentre il path relativo può non essere valido quando si cambia la cwd

Utenti, filesystem e file

Comandi per la gestione di file e directory

Comando `ls`

- Il comando `ls [-la] [-R] [nomedir]` mostra il contenuto della directory `nomedir` (o della cwd, se `nomedir` non è dato)
  - `ls -l` mostra informazioni relative al contenuto della directory
  - `ls -a` mostra i file considerati *nascosti* (*hidden*), cioè i file con nomi che cominciano con il punto
  - `ls -R` mostra tutto il sottoalbero con radice in `nomedir`
  - `ls -s` mostra la dimensione dei file

Comando `mkdir`

- Il comando `mkdir [-p] nomedir` crea la directory `nomedir` (vuota)
  - Se si usa l'opzione `-p` e se `nomedir` è un path con più di una directory, allora crea tutte le directory nel path (se non esistono)
  - ad esempio, `mkdir -p dir11/dir13/dir15`, supponendo che `dir11` esista già, creerà la directory `dir13` dentro `dir11`, e poi `dir15` dentro `dir13`
  - **Esercizio** provare a vedere cosa succede, nella stessa situazione, senza l'opzione `-p`

- **Esercizio**
  - creare l'intero albero di directory dato sopra (ovvero, `/home/utente1/dir1/dir3/dir7/`), posizionarsi dentro `dir1` e poi in `dir7` sia usando che non usando la directory parent `..`
  - controllare il risultato usare il comando `ls`;
  - controllare come cambia il path riportato nel prompt

Comando `touch`

- Il comando `touch nomefile` crea il file `nomefile` (vuoto)
- Per modificare un file, scrivendoci un testo dentro, dare il seguente comando `geany nomefile &`; si aprirà un editor grafico standard, dove è possibile scrivere e salvare le modifiche (il carattere `&` serve per esecuzione in background)
- Alternativamente, si possono usare editor che si interfacciano direttamente con il terminale (ma non sempre sono installati): `nano nomefile` oppure `pico nomefile` (attenzione: *non usare* il carattere `&`). Più complicato: `vi nomefile`
  - se avete provato ad eseguire `vi nomefile` e non sapete come uscire, digitate i caratteri `:q` seguiti da invio)

Altri file

- Per creare file non di testo, occorre usare opportuni altri programmi (a seconda di cosa serve)
  - ad esempio, i vari applicativi di LibreOffice possono essere usati per creare file contenenti documenti formattati (DOC, DOCX, ODT, etc), fogli di calcolo (XLS, XLSX, ODS, etc)
  - basta usare il comando `libreoffice nomefile`, che però non è installato sul disco virtuale fornito (ma per installarlo basta scrivere `sudo apt-get install libreoffice`)
  - attenzione: `nomefile` deve già esistere; altrimenti si può eseguire `libreoffice` senza argomenti e poi usare l'interfaccia grafica per creare un nuovo documento del tipo desiderato

Comando `tree`

- Si può visualizzare un intero albero di directory con il comando `tree [-a] [-L maxdepth] [-d] [-x] [nomedir]`
  - potrebbe non essere installato: `sudo apt-get install tree`
  - in generale: se si dà un comando sbagliato, e l'output sembra non finire mai, provare a premere CTRL+c
  - usare l'opzione `-L` per limitare la profondità dell'albero mostrato
  - attenzione: l'output contiene anche caratteri ASCII non-standard, per visualizzare la struttura dell'albero. Si tratta di caratteri UTF-8 a 3 bytes per carattere (vedere `http://www.fileformat.info/info/unicode/utf8.htm`)

Utenti, filesystem e file

Filesystem e comandi per la gestione

Filesystem di Linux e altri filesystem

- Il filesystem di Linux è unico, ma può contenere elementi eterogenei
  - il disco, ovviamente
  - potrebbe esserci però più di un disco, ad esempio uno tradizionale e uno a stato solido
  - potrebbe esserci un disco solo, ma partizionato; in questo caso, ogni partizione può trovarsi in punti diversi del file system
  - filesystem virtuali, montati dal kernel per gestire le risorse
  - filesystem di rete
  - filesystem in memoria principale (RAM)

Comando `mount`

- Il comando `mount` permette di *aggiungere* (o *montare*) un filesystem (di un dispositivo esterno o di un'altra partizione del disco rigido) agganciandolo a una directory - stratagemma del **mounting**
- Serve a rendere accessibili ai programmi e agli utenti del sistema, file e directory contenuti nel *nuovo* filesystem
- Una qualsiasi directory dell'albero gerarchico può diventare il punto di **mount** per il *nuovo* filesystem

Comando `mount`

- Più formalmente:
  - una directory x del filesystem di Linux si dice *punto di mount* per un altro (nuovo) filesystem F se e solo se la directory root di F diventa accessibile a partire da x
  - così, il comando `ls` x mostrerà il contenuto della directory root di F
  - leggere o modificare un file dentro il sottoalbero radicato in x avrà l'effetto di leggerlo o modificarlo nella corrispondente directory di F
  - quindi, un path assoluto p dentro il filesystem F diventa il path assoluto x/p dentro Linux (supponendo che x sia anch'esso un path assoluto)

Comando `mount`

- È meglio scegliere una directory x vuota, altrimenti il suo *vecchio* contenuto non sarà più visibile fino all'`umount`
  - **ma** il contenuto di x sul disco non viene cancellato, semplicemente non è più accessibile (né in lettura né in scrittura) tramite `ls` ed altri comandi per il filesystem
  - più precisamente: se x si trovava, prima del mount, nel filesystem $G \neq F$, allora i contenuti di G relativi alla directory di x vengono nascosti (ma non cancellati), e si mostrano quelli di F a partire dalla sua root

Comando `mount`

- **Ad esempio**:
  - Sulla directory `/proc`, durante il boot, viene montato un filesystem virtuale (non corrisponde ad alcun disco)
  - I filesystem di rete potrebbero essere montati su una directory `/nfs`, creata appositamente
  - Un filesystem in memoria principale può essere montato su una directory all'interno della home di un utente
  - Il disco principale, contenente l'installazione del sistema operativo, sarà montato su `/`
  - Un disco secondario, senza l'installazione del sistema operativo (o con installato un altro sistema operativo!), può essere montato su `/windows`

Filesystem, directory e dischi

- Anche se c'è un solo disco, è possibile *partizionarlo*: una parte si prende il sistema operativo (e viene montato su `/`) e una parte si prende la directory home (e viene montato su `/home`)
  - utile se poi si vuole installare un altro sistema operativo da capo: si installa sulla partizione che era montata su `/`, si monta la home così com'è su una qualche directory, e ci si risparmia di dover copiare i vecchi dati della home: sono già lì
    - se si tratta semplicemente di un'altra distribuzione Linux non c'è problema
    - se invece si passasse a Windows, usare il filesystem `home` sarebbe complicato (Windows non riconosce nativamente i tipi di filesystem di Linux, come `ext2` o `ext3`)

Filesystem, directory e dischi

- **Attenzione** partizionare un disco implica cancellare i dati precedenti
- Per essere più precisi: partizionare ulteriormente una partizione di un disco cancella i dati presenti in quella partizione
- Programmi per partizionare dischi: `gparted` (grafico), `parted` ed altri (testuali)
- Nei sistemi Linux ci sono sempre almeno due partizioni: una montata su `/` e una di tipo particolare usata per lo *swap* (memoria virtuale)

Tipi di filesystem

- Principali caratteristiche del **tipo** di filesystem
  - Dal punto di vista dell'**utente**: dimensione massima di una partizione, dimensione massima di un file, lunghezza massima di un nome di file, se è journal o meno
  - *journal* vuol dire che ogni modifica viene scritta su un file speciale, e applicata su disco in un secondo tempo (meglio come prestazioni e come resistenza ad alcuni tipi di fault)
  - Dal punto di vista del **progettista-programmatore** di sistemi operativi, il tipo definisce anche la metodologia con cui i dati vengono letti/scritti sul disco

Tipi di filesystem

- I tipi di filesystem sono relativamente pochi: ci sono quelli di Windows (NTFS, MSDOS, FAT32, FAT64) e svariati per Linux
- Per ognuno di questi tipi, ci sono le caratteristiche dette sopra
- Ogni disco, o meglio ogni partizione, va inizialmente *formattata* con uno di questi tipi: ovvero, occorre dichiarare con quale tpo di filesystem si vuole usare quella partizione
- quando un disco, o una sua partizione, viene montata, occorre specificare il giusto filesystem (quello con cui è stato formattato)

Tipi di filesystem

- I principali tipi di filesystem di un sistema Linux

Nome	Journal	Partiz (TB)	File (TB)	Nome file (bytes)
Ext2	No	32	2	255
Ext3	Sì	32	2	255
Ext4	Sì	1000	16	255
ReiserFS	Sì	16	8	4032

Comando `cat`

- Per sapere quali filesystem sono montati e dove: comando `cat /etc/mtab`, oppure anche `mount` (senza argomenti)
- Il comando `cat [nomefile]`: scrive a schermo il contenuto di `nomefile`
  - senza argomenti, resta in attesa: se si scrive qualcosa e poi si preme invio, ripete quanto scritto, finché non si preme CTRL+d, che è il carattere EOF (*end-of-file*)
  - funziona bene solo se il file è di testo (e se usa la codifica riconosciuta da `cat`), altrimenti scrive caratteri incomprensibili
  - comando `mount`, è quello da usare per fare il *mounting* descritto prima
- Per sapere quali filesystem vengono montati a tempo di boot (esclusi quelli gestiti dal kernel): `cat /etc/fstab`

- Tabella delle directory di primo livello di un sistema Linux

Directory	**Spiegazione**	**Montata**
/boot	Kernel e file di boot	NO
/bin	Binari (programmi eseguibili) di base	NO
/dev	Devices (periferiche) hardware e virtuali	boot
/etc	File di configurazione di sistema	NO
/proc	Dati e statistiche dei processi e parametri del kernel	boot
/sys	Informazioni e statistiche di device di sistema	boot
/media	Mountpoint per device di I/O (es: CD, DVD, USB pen)	quando necessario
/mnt	(come /media)	quando necessario
/sbin	Binari di sistema	NO
/var	File variabili (log file, code di stampa, mail ...)	NO
/tmp	File temporanei	NO
/lib	Librerie	NO

File di info per utenti e gruppi

- Alcuni file importanti: `/etc/passwd` e `/etc/group`
  - il primo file elenca tutti gli utenti, il secondo tutti gruppi
  - entrambi i file sono un esempio della filosofia Linux: si usano file di testo (con codifica ASCII a 8 bit) con una struttura definita e conosciuta dai programmi che devono interagire con quei file
  - ad esempio, `adduser` conosce la struttura di entrambi i file
  - questi due file, come molti altri, sono organizzati a *righe*, dove per *riga* si intende una sequenza di caratteri terminati dall'andata a capo LF (*line feed*, carattere 0x0A ASCII)

File di info per utenti e gruppi

- **Esercizio** far scrivere a schermo il contenuto di tali file
  - Le righe che iniziano con il carattere # sono da intendersi come commenti, e vengono ignorate dai programmi che leggono/scrivono tali file
  - Ogni riga è formata da campi separati dal caratteri speciale : (che, quindi, non può essere usato per definire un nome utente)
  - per `/etc/passwd`, i campi sono i seguenti: `username:password:uid:gid:gecos:homedir:shell`
  - per `/etc/group`, i campi sono i seguenti: `groupname:password:groupID:lista_utenti` (dove la lista degli utenti è separata da virgole, e quindi neanche le virgole possono comparire in un nome utente)
  - **Esercizio** verificare che `utente3` non sia presente in `/etc/passwd`, crearlo, e poi controllare che sia presente
  - **Esercizio** verificare che `utente3` non sia nel gruppo `sudo`, aggiungerlo al gruppo `sudo` e poi controllare che sia presente

Utenti, filesystem e file

Comandi per cercare

Il comando `find`

- Comando `find [opzioni] [directory] [condizione] [azione]`: serve per la ricerca di file
  - La ricerca avviene nella directory specificata e nelle sue sottodirectory, oppure nella cwd se la directory non è specificata
  - Criteri di ricerca tipici sono: il nome del file (-name nomefile [suffisso]), il nome utente (-user nomeutente), le dimensioni del file (-size), l'accesso indicato in giorni (-atime [+-]n) o la modifica indicata in giorni (-mtime [+-]n)
  - Esempio: `find /tmp -name "*.odt" -mtime -3 -size +20k`

Il comando grep

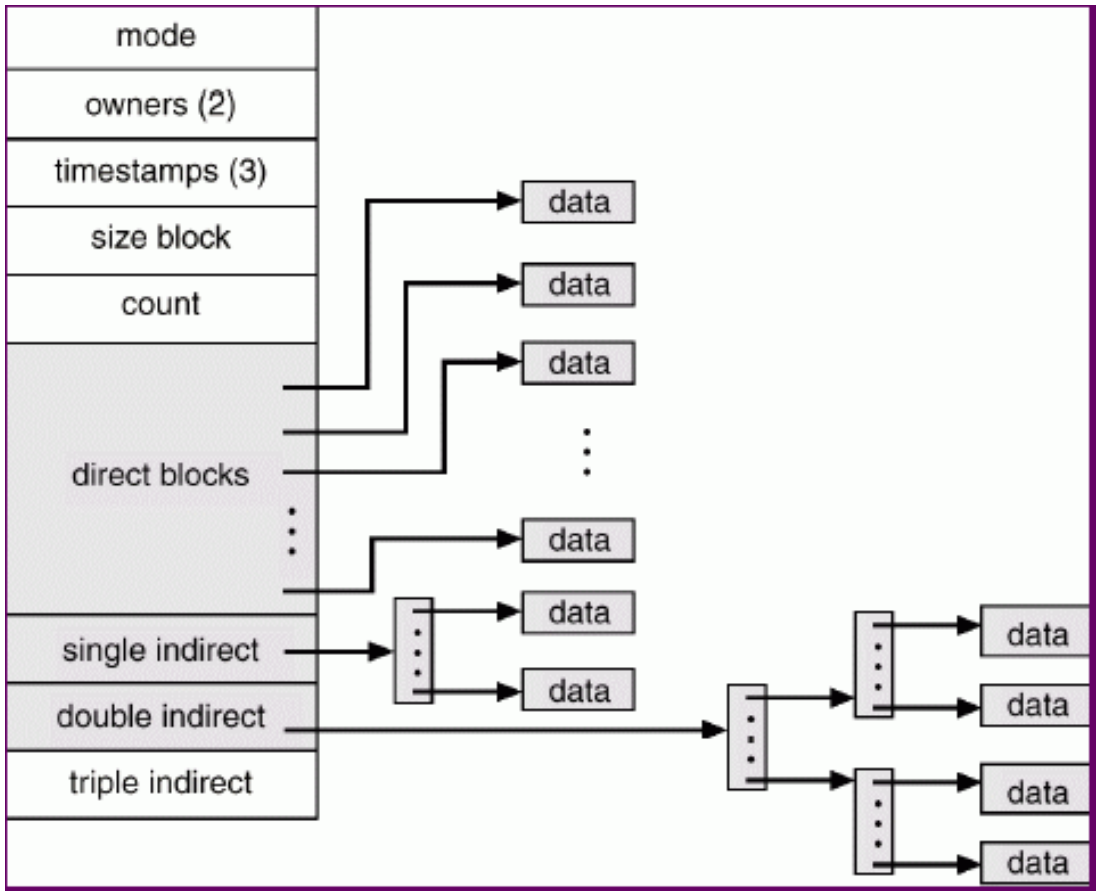
- Comando `grep [opzioni] stringa [file]`: serve per cercare la stringa specificata in un file di testo
- grep - acronimo di global regular expression print
- La stringa è una sequenza di uno o più caratteri (lettera singola, parola o frase) e può includere spazi, segni di interpunzione e caratteri di controllo
- Solitamente grep viene utilizzato su tutti i file nella cartella corrente
- `-i` ignora le differenze tra lettere maiuscole e minuscole
- `-n` in ogni linea del risultato viene mostrato il numero di riga del file
- `-r` consente una ricerca ricorsiva nelle sottocartelle

Utenti, filesystem e file

File in Linux e comandi sui file

- Linux usa gli **i-node** per memorizzare file su disco
- Un inode (o i-node, abbreviazione di *index node*) è una struttura dati sul filesystem che contiene gli attributi base di un *filesystem object*, cioè file, directory o altri oggetti
- Le informazioni includono:
  - la dimensione del file e la sua locazione fisica (se risiede su un dispositivo a blocchi, come ad esempio un hard disk)
  - il proprietario e il gruppo di appartenenza
  - le informazioni temporali di modifica (mtime), ultimo accesso (atime) e di cambio di stato (ctime)
  - il numero di collegamenti fisici che referenziano l'inode
  - i permessi d'accesso
  - un puntatore allo spazio del disco che contiene i file veri e propri

- Quindi, ogni file del filesystem (directory comprese) è rappresentato da una struttura dati `inode` ed è univocamente determinato da un `inode number`



- Non ci sono mai contemporaneamente 2 file con lo stesso inode number
- Gli inode number *liberati* dalla cancellazione di un file verranno riusati alla prima occasione
- Esiste ovviamente una tabella di tutti gli inode, che si trova solitamente all'inizio del disco

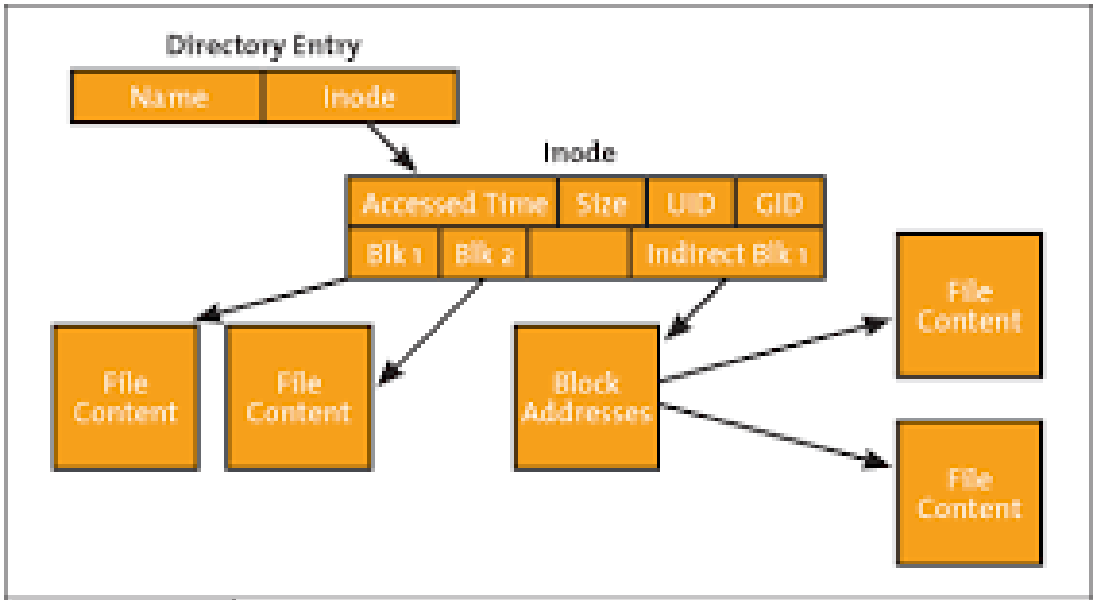


FIGURE 1 RELATIONSHIP BETWEEN THE DIRECTORY ENTRY, AN INODE, AND BLOCKS OF AN ALLOCATED FILE

- Tabella dei principali attributi della struttura dati `inode`

Attributo	**Spiegazione**
Type	Tipo di file (regular, block, fifo ...)
User ID	Id dell'utente proprietario del file
Group ID	Id del gruppo a cui è associato il file
Mode	Permessi (read, write, exec) di accesso per il proprietario, il gruppo e tutti gli altri
Size	Dimensione in byte del file
Timestamps	ctime (inode changing time: cambiamento di un attributo), mtime (content modification time: solo scrittura), atime (content access time: solo lettura)
Link Count	Numero di hard links
Data Pointers	Puntatore alla lista dei blocchi che compongono il file; se si tratta di una directory, il contenuto su disco è costituito di una tabella con 2 colonne: nome del file/directory e suo inode number

Il comando `ls`

- Per vedere il valore degli attributi, occorre usare in modo più esteso il comando `ls`:
  `ls [-a] [-c] [-u] [-R] [-l] [-i] [-n] [-S] [-h] [-1] [nomedir1 ... nomedir`*n*`] [nomefile1 ... nomefile`*n*`]`
- Si possono passare indifferentemente directory e files: si otterrà il contenuto delle directory indicate e i files indicati
- Per vedere l'inode number dei file e delle directory, usare l'opzione `-i`
- Per vedere l'ID (numerico) di utente e gruppo, anziché il corrispettivo nome, usare l'opzione `-n`
- Per vedere i timestamp, si usa l'opzione `-l` in combinazione con `-c` (per *ctime*) o con `-u` (per *atime*) e senza niente (per *mtime*)

Il comando `ls`

- Per vedere user name, group name, dimensione e permessi usare l'opzione `-l`
  - **per i file** la dimensione è quanto effettivamente occupato da quel file su disco, in bytes
  - **per le directory** viene mostrata la dimensione del file speciale corrispondente, che consiste in una lista di coppie (nomefile, numero di inode), quindi non conta quanto siano grandi i file contenuti, ma *quanti* sono (e quanto sono lunghi i loro nomi)
  - solitamente, le directory hanno una dimensione minima di 4kB, anche se sono vuote

Il comando `ls`

- L'opzione `-l` mostra anche una riga iniziale *total* per ogni directory di cui mostra il contenuto: si tratta delle dimensione dei file contenuti in quella directory, come numero di *blocchi* su disco (normalmente, 1 blocco ha dimensione di un 1kB)
  - si tratta solo di una comodità per l'utente, non dei blocchi con cui il kernel comunica con il disco
  - per vedere l'output in Mega, cioè 10^6, si può scrivere `ls --block-size=M`, per vederlo in MegaByte, cioè 2^20, si può scrivere `ls --block-size=MB`, mentre con `ls --block-size=k` e `ls --block-size=kB` si hanno le dimensioni rispetto a $1000 = 10^3$ e a $1024 = 2^10$ (kbyte) rispettivamente
  - N.B. le dimensioni sono tutte approssimate per eccesso

Il comando `ls`

- Notare che queste ed altre informazioni non sono sul `man`: vanno cercate nelle informazioni estese, reperibili con il comando `info ls`

- **Esercizio** verificare le dimensioni con `ls --block-size=M`, `ls --block-size=MB`, `ls --block-size=k` e `ls --block-size=kB` posizionandosi in diverse directory

- **Esercizio** creare un file, modificarlo, poi leggerlo e verificare che i valori per atime ed mtime cambino di conseguenza

I permessi dei file

- I permessi dei file servono a specificare chi può far cosa
  - **ogni file** ha associato un **utente** ed un **gruppo proprietari**
  - inizialmente, il proprietario è chi crea il file, ed il gruppo è il gruppo primario (ovvero, quello specificato per primo in `/etc/passwd`) di quell'utente
  - *inizialmente* perché si può usare il comando `chown` per cambiare il proprietario (vedere più sotto)
  - il proprietario decide cosa permettere e cosa no agli altri utenti e agli altri gruppi, definendo i **permessi** di file e directory

I permessi dei file

- **File**: i permessi sono quelli di lettura, scrittura ed esecuzione

Permesso	Ottale	Significato
- - -	0	Non si può fare niente (è però possibile vedere gli attributi, se i permessi sulla directory lo consentono)
- - x	1	Non si può fare niente (non si può eseguire, perché bisognerebbe prima leggere)
- w -	2	Si può scrivere, ma solo da riga di comando, sovrascrivendo completamente il file o appendendo dati alla fine (per altre modifiche, bisognerebbe prima leggere); si può anche cancellare, ma occorrono opportuni diritti sulla directory
- w x	3	Come il permesso 2
r - -	4	Si può leggere
r - x	5	Si può leggere ed eseguire
r w -	6	Si può leggere e modificare a piacimento (ma attenzione alla cancellazione)
r w x	7	Si può fare tutto (ma attenzione alla cancellazione)

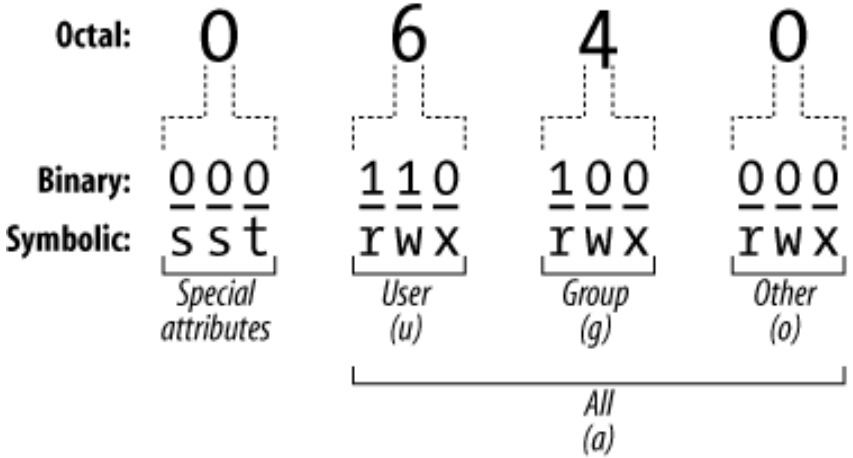
I permessi dei file

- **Directory**: la cosa è un po' più complicata

Permesso	Ottale	Significato
- - -	0	Non si può fare niente
- - x	1	Si può settare come cwd (ma solo se c'è il permesso per *tutte* le directory nel path); si può anche *attraversare* (cioè vedere un file o una directory al suo interno, se c'è permesso in lettura)
-w -	2	Non si può fare niente (per fare veramente modifiche, occorrono i permessi di esecuzione)
- w x	3	Come il permesso 7, ma non si può listare il contenuto (con o senza attributi)
r - -	4	Si può solo listarne il contenuto, senza vedere gli attributi dii file/directory contenuti (si può sapere se si tratta di file o di directory); non può essere *attraversata*
r - x	5	Si può leggere (attributi compresi), settare come cwd ed attraversare; non è possibile cancellare o aggiungere file/directory
rw -	6	Come il permesso 4 (write senza execute è inutile)
r w x	7	Si può fare tutto: listare contenuto, aggiungere file e directory, cancellare file contenuti (anche senza permesso di scrittura sul file!) e directory contenute (ma occorrono tutti i permessi)

I permessi dei file

- Per ogni file/directory, sono specificati 3 insiemi di permessi come quelli definiti sopra
  - il primo da sinistra è per l'**utente**: si applica se proprietario e utente utilizzatore coincidono
  - il secondo per il **gruppo**: si applica se l'utente utilizzatore appartiene al gruppo del file
  - il terzo per **tutti gli altri utenti**: si applica nei rimanenti casi
  - vengono mostrati da `ls -l` e `stat`
  - ci sono poi i permessi (o attributi) speciali



I permessi dei file

- Permessi speciali: **setuid bit** (s), **setgid bit** (s), **sticky bit** (t)
  - I permessi speciali consentono di impostare funzioni avanzate sui file o sulle directory
  - Sono rappresentati dai bit:
    - **setuid** - Set User Identification
    - **setgid** - Set Group ID
    - **sticky** - conosciuto come Sticky bit
  - I permessi speciali vanno usati con molta prudenza
  - Vengono visualizzati al posto del bit di esecuzione: il setuid nella terna utente, il setgid nella terna gruppo e lo sticky nella terna altro
  - Vengono impostati usando la lettera **s** o la **t**: se è minuscola il permesso c'è, altrimenti è maiuscola (ed *inutile*)

I permessi dei file

- Il **setuid bit** si usa solo per i file eseguibili
- Impostando questa modalità su un file si fa in modo che il file eseguito da qualunque utente del sistema, venga eseguito con gli stessi privilegi dell'utente proprietario
- Quindi quando il file viene eseguito, i privilegi con cui opera il corrispondente processo non sono quelli dell'utente che esegue il file, bensì quelli dell'utente proprietario del file
- Quindi, se il proprietario è root, il file viene eseguito con i privilegi di root, indipendentemente da chi lo ha eseguito
- Ad esempio, il comando `passwd` ha il setuid, che permette ad un utente di modificare la propria password

I permessi dei file

- Il **setgid bit** è l'analogo del *setuid bit*, ma vale per i gruppi
- I privilegi sono quelli del gruppo che è proprietario del file eseguibile
- Il **setgid bit** può essere applicato anche ad una directory e allora ogni file nella directory ha il gruppo della directory, anziché quello primario di chi crea files
- **Esercizio** provare a dare il comando `stat /tmp /usr/bin/passwd` e controllare cosa viene visualizzato
- **Esercizio** capire come mai `adduser` va usato solo da un utente amministratore, mentre `groups` no

I permessi dei file

- Lo **sticky bit**, applicato su una directory, *corregge* il comportamento dei permessi write+execute (vedere Tabella directory): si possono cancellare file solo se si hanno i permessi di scrittura su quei file
- Per essere più precisi, lo sticky bit ha effetto nel seguente caso: se una directory d appartiene all'utente u e un utente $u' \neq u$ cerca di cancellare un file f in d che non appartiene nè ad u' nè al gruppo cui appartiene u', allora, senza sticky bit su d, sarà sufficiente avere i diritti di scrittura su d (nel gruppo *other*) per cancellare f, anche se non si hanno i permessi di scrittura su f (sempre su *other*). Con lo sticky bit, per cancellare f sono necessari anche i permessi di scrittura su f.

Il comando `chmod`

- Comando `chmod mode[, mode...] filename`: serve a cambiare i permessi
- Due modi di settare il *mode*: ottale o con lettere (simbolico)
  - **ottale**: si danno 4 numeri
  - Il primo numero indica setuid (4), setgid (2) e sticky (1), gli altri numeri vanno da 0 a 7, come nelle Tabelle, e sono per utente, gruppo ed altri
  - Si possono anche dare soltanto 3 numeri, e si intende che i bit speciali sono tutti a 0 (caso più comune)
  - **Esercizio** creare un file e settarne i permessi a `rws r-S -w-` e poi a `rwx r-- -wT` usando la modalità ottale

Il comando `chmod`

- Comando `chmod mode[, mode...]  filename`
  - **lettere**: qui se ne possono specificare molti, separati da virgole
  - il formato di ogni modo simbolico è :
    `[ugo][+-=][perms...]`
    dove `perms` è zero, una o più lettere nell'insieme {rwxst},
    oppure una lettera nell'insieme {ugo}
  - **Esercizio** creare un file e settarne i permessi a `rws r-S -w-` e poi a `rwx r-- -wT` usando la modalità simbolica

Il comando `chmod`: esempi ed esercizi

- **Esempio: Attivare suid**
  - Usando la modalità simbolica: `chmod u+s file.txt`
  - Usando la modalità ottale: `chmod 4755 file.txt`
  - Visualizzando si ha:
    ```
    -rwsr-xr-x 1 utente root 500 2020-10-03 18:46
    file.txt
    ```
- **Esempio: Disattivare suid**
  - Usando la modalità simbolica: `chmod u-s file.txt`
  - Usando la modalità ottale: `chmod 0755 file.txt`
  - Visualizzando si ha:
    ```
    -rwSr-xr-x 1 utente root 500 2020-10-03 18:46
    file.txt
    ```
- **Esercizio** togliere il permesso di lettura ad un file, e poi provare a visualizzarlo con `cat` o con `geany`
- **Esercizio** verificare che `chmod` modifica il ctime del file

Il comando `umask`

- Comando `umask [mode]`: è un comando della bash (non si trova in man, ma si può fare help umask)
  - Definisce quali permessi **negare** al momento della creazione di nuovi file e directory
  - Senza argomenti mostra l'umask corrente, altrimenti setta la maschera dei file al `mode` specificato
  - Si possono specificare solo le 3 terne, non i permessi speciali (che inizialmente sono tutti a 0)
  - I permessi predefiniti sono **666** per i file e **777** per le directory
  - Settando umask, alla creazione i permessi per un file saranno il risultato dell'operazione bit-a-bit *666 AND NOT(umask)*, mentre per una directory saranno *777 AND NOT(umask)*
  - `umask` ha effetto anche su `chmod`
  - **Esercizio:** modificare l'umask in modo che i permessi siano 664, sia che venga creata una directory che un file. Modificare poi in modo che il permesso per una nuova directory sia 775 e per un file sia 664.

I comandi `chown` e `chgrp`

- Comandi `chown [-R] proprietario {file}` e `chgrp [-R] gruppo {file}`: servono a cambiare proprietario o gruppo
  - Possono essere usati solo da root, quindi ci vuole `sudo` (altrimenti chiunque potrebbe creare un file con *contenuti compromettenti* e darlo ad utente ignaro)
  - Se si passano delle directory e c'è l'opzione `-R`, si cambiano ricorsivamente tutti i file e le directory in esse contenute
  - **Esercizio** riprendendo l'esempio dell'albero di directory `/home/utente1/dir1/dir3/dir7/`, creare un file (vuoto) `file`*i* dentro ciascuna directory `dir`*i*, e cambiare i proprietari in questo modo: la directory `dir3` diventa di `utente3`, tutti i file dentro `dir3` diventano di `utente2`, `dir7` diventa di `utente2` e `file7` diventa di `utente3`. Dopodiché, provare a cambiare i permessi di `file3` e `dir7`

Esercizi

- **Esercizio**: provare a fare un esempio funzionante di sticky bit. A tal proposito, creare da root (usando `sudo`) una directory avente tutti i diritti per tutti i gruppi e crearci dentro un file, togliendo a quest'ultimo i permessi per il gruppo *other*; infine, provare a cancellarlo. Fare lo stesso dopo aver aggiunto lo sticky bit alla directory
- **Esercizio**: provare a fare un esempio funzionante di setuid bit. A tal proposito, copiate `/bin/cat` in una directory, e abilitategli il setuid bit. Create un file da root (usando `sudo`), e dategli il solo permesso di lettura per il solo gruppo "owner". Provare a vedere il contenuto di questo file usando il `cat` di sistema: fallirà. Ora provate a farlo usando `./cat`...

Il comando `touch`

- Qualche dettaglio in più su `touch [-a] [-m] [-t timestamp] {file}`
  - Il suo *vero* uso è quello di cambiare i timestamps negli attributi dei file dati
  - Li cambia normalmente tutti e 3, a meno che non si sia data l'opzione `-a` (solo atime) o `-m` (solo mtime)
  - Come *effetto collaterale*, se un file dato non esiste lo crea
  - Può essere usato anche su una directory (ma solo se esiste: come mai?)
  - Con `-t timestamp` setta i timestamp del file al timestamp dato, anziché al timestamp del tempo attuale
  - **Esercizio** immaginare almeno due casi in cui `touch` non riesce a creare un file

Il comando cp

- Comando `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`: permette di copiare file e directory
  - 2 modalità basilari:
    - con 2 argomenti, `filedestinazione` può essere una directory od un file (in questo caso, ovviamente, il sorgente dev'essere a sua volta un file...)
    - con 2 argomenti, la destinazione può non esistere, e verrà creata (un file se la sorgente è un file, una directory se la sorgente è una directory)
    - con più di 2 argomenti, `filedestinazione` dev'essere una directory (esistente)
    - se si vuole copiare un file che si trova in un'altra directory nella cwd (mantenendone il nome), il secondo argomento sarà `.`

Il comando cp

- Comando `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`
  - Tra i `filesorgenti` ci possono essere file e/o directory
  - Ovviamente, se la destinazione è una directory esistente, i sorgenti vengono copiati *dentro* la directory destinazione (anche se il sorgente è una directory)
  - Se ci sono directory tra i sorgenti, allora occorre dare l'opzione `-r`, altrimenti quelle directory non verranno copiate (verranno copiati solo i file)
  - Se la copia avviene su file esistenti, verranno sovrascritti; con l'opzione `-i`, prima di sovrascrivere viene chiesta conferma

Il comando cp

- Comando `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`
  - Con l'opzione `-u`, la sovrascrittura avviene solo se l'mtime del sorgente è più recente di quello della destinazione (o quello di destinazione non esiste)
    - **Esercizio** Vedere il comando `rsync`, più completo
  - I permessi del file sorgente potrebbero non venire preservati: sono soggetti alla dura legge dell'`umask`
  - Per forzare a mantenere i permessi, c'è l'opzione `-a` (che serve anche per altri motivi)

Il comando cp

- Comando `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`
  - **Esercizio** verificare il funzionamento di ciascuna delle opzioni mostrate. Per esempio, per l'opzione `-i`, provare a copiare un file sorgente in un file destinazione esistente e vedere che effettivamente viene chiesta conferma. Per l'opzione `-a`, è sufficiente creare un file ed aggiungere qualche permesso, poi fare la copia con e senza `-a`
  - **Esercizio** verificare il funzionamento del comando `cp` con diverse tipologie di argomenti: i) 2 file (tutti e 4 i casi tra esistenti e non); ii) 2 directory (tutti e 4 i casi tra esistenti e non); iii) 3 file (l'ultimo sia esistente che non); iv) 3 directory (idem); v) 2 file e 2 directory...

Il comando `mv`

- Comando: `mv [-i] [-u] [-f] {filesorgenti} filedestinazione`
  - Come `cp`, ma serve a *spostare* anziché *copiare*: quindi, i sorgenti risulteranno *cancellati* dopo l'esecuzione del comando, ed esisteranno solo nella destinazione
  - Con 2 argomenti omologhi (2 file o 2 directory) effettua in pratica una ridenominazione
  - Le opzioni `-i` e `-u` hanno lo stesso significato di `cp`; `-f` è il contrario di `-i` (ed è l'opzione di default)
  - `-r` e `-a` non servono: è come se fossero sempre abilitate
  - **Esercizio** rifare entrambi gli esercizi visti per `cp`

Il comando `rm`

- Comando `rm [-f] [-i] [-r] {file}`: per cancellare e basta
  - Cancella definitivamente i file e le directory indicati (non completamente vero nel caso di hard link, vedere più avanti)
  - Le opzioni `-f` e `-i` sono come quelle della `mv` (ma stavolta il default è `-i`); l'opzione `-r` è come quella della `cp`
  - **Esercizio** creare una directory, poi crearci dentro un file, togliere il permesso di scrittura a quest'ultimo e cancellarlo. Lo lascia fare? Perché? Provare a fare la stessa dentro la directory predefinita `/tmp`
  - **Esercizio** creare un file, guardare il suo inode, cancellarlo e poi creare subito dopo un altro file: qual è l'inode del nuovo file (supponendo che nessun altro utente stia usando il computer)?

Il comando dd

- Comando `dd [opzioni]`
  - Copia file in blocchi in modo elaborato
  - Può eseguire *conversioni* durante l'operazione di copia
  - Utilizza un modo diverso per dare le opzioni, cioè tramite assegnamenti del tipo `variabile=valore`
  - Le variabili più importanti sono:
    - `if` file di input (se non dato, legge da tastiera - `stdin`)
    - `of` file di output (se non dato, scrive su schermo - `stdout`))
    - `bs` dimensione di un singolo blocco in lettura/scrittura
    - `count` numero di blocchi da copiare
    - `convert`, in questo caso, il valore specifica una conversione, per esempio di codifica (da minuscolo a maiuscolo e viceversa, più svariate altre cose; vedere il man)

Il comando dd

- Comando `dd [opzioni]`
  - Si usa soprattutto nei casi in cui la copia tramite cp non funzionerebbe, oltre che per le conversioni, ma ha svariati usi
    - `/dev/zero` è un file *speciale*, o meglio *character special file*: se si prova a farsi stampare questo file, vengono fuori un numero infinito di zeri
    - non si trova su un disco, ma è connesso direttamente al kernel, che risponde appunto con un numero infinito di zeri quando si cerca di leggerlo
    - grazie a `dd` e a sue opportune opzioni, si può creare un file con un certo numero di zeri in modo semplice e veloce
    - altro file speciale simile è `/dev/urandom`, che risponde con un numero infinito di numeri
    - ad esempio `dd if=/dev/zero of=test_file.zeri bs=1M count=10` crea un file da 10MB, tutto fatto di zeri pseudo-casuali

Il comando dd

- Comando `dd [opzioni]`
  - **Esercizio** provare ad aprire il file creato con `dd` come detto sopra usando ad esempio `geany`
    - contiene quello che ci si aspetta? se no, perché?
    - perché uno dovrebbe fare una cosa del genere, apparentemente stupida?
    - ad esempio, per cancellare completamente dati da un supporto di memoria oppure per preparare un supporto di memoria (o meglio ancora un file) ad essere *formattato*
    - altro possibile uso: copiare solo una parte di un file, grazie a `skip`=*n*, che salta *n* blocchi (quanti bytes ci sono in un blocco è dato dalla variabile `bs`)
    - analogamente, `seek`=*m* permette di non modificare i primi *m* blocchi del file destinazione (nel caso il file destinazione non sia vuoto, o abbia almeno quel numero di blocchi)

Il comando dd

- Comando `dd [opzioni]`
  - **Esercizio** creare un file di testo usando `geany`, con il seguente contenuto:
    ```
    ciao1
    addio2
    via3
    ehila4
    dove vai5
    ```
  - copiarlo in un altro file in modo che quest'ultimo contenga solo i bytes che vanno dal decimo al ventesimo
  - usare sia 1 che 10 come valori per `bs`
  - rifare nuovamente la copiatura sullo stesso file destinazione, ma questa volta fare in modo che il contenuto del file destinazione sia duplicato (ovvero, contenga per 2 volte consecutive i bytes che vanno dal decimo al ventesimo)

Il comando `ln`

- Comando `ln [-s] sorgente [destinazione]`
  - Crea un **soft link** se c'è l'opzione `-s`, un **hard link** altrimenti
  - In pratica corrisponde a copiare un file, senza copiare l'intero file (che potrebbe essere grande)
  - Inoltre, successive modifiche al *contenuto* del sorgente si rifletteranno sulla destinazione e viceversa
  - Se c'è un solo argomento, allora la sorgente dev'essere un file in un'altra directory e il link destinazione avrà lo stesso nome di questo file

Il comando `ln`

- Comando `ln [-s] sorgente [destinazione]`
  - **soft link**: viene creato un nuovo file (destinazione), il contenuto del quale coincide con `sorgente` (lo si può intuire guardando la dimensione del file con `ls -l` o con `stat`); da notare che spesso questo contenuto è direttamente negli attributi del file
  - Se si cancella il file sorgente, il link diventa *morto* e provare a visualizzare il file porta ad un errore
  - **hard link**: aumenta il link count della sorgente, e crea una destinazione con lo stesso link count
  - Se si cancella la sorgente, il link count decresce, ma la destinazione continua a mantenere il contenuto del file
  - Se si fa un hard link di un hard link, il link count cresce ancora (per tutti i file coinvolti)

Il comando `ln`

- Comando `ln [-s] sorgente [destinazione]`
  - Cancellare un file non vuol dire più automaticamente rimuovere il suo contenuto dal disco: potrebbero esserci hard links...
  - Per vedere a cosa *punta* un hard link, occorre visualizzare l'inode number
  - Per vedere a cosa punta un soft link, basta `ls -l`
  - Con i soft link c'è un *puntatore* ed un *puntato* e cancellare il puntatore non ha effetti sul puntato, mentre cancellare il puntato ha effetti sul puntatore
  - Con gli hard link è come se fossero tutti puntatori al file su disco (o nell'area di memoria collegata al rispettivo filesystem)
  - non si possono fare hard links a directory (tranne quelli predefiniti `..` e `.`), mentre si possono fare i soft links

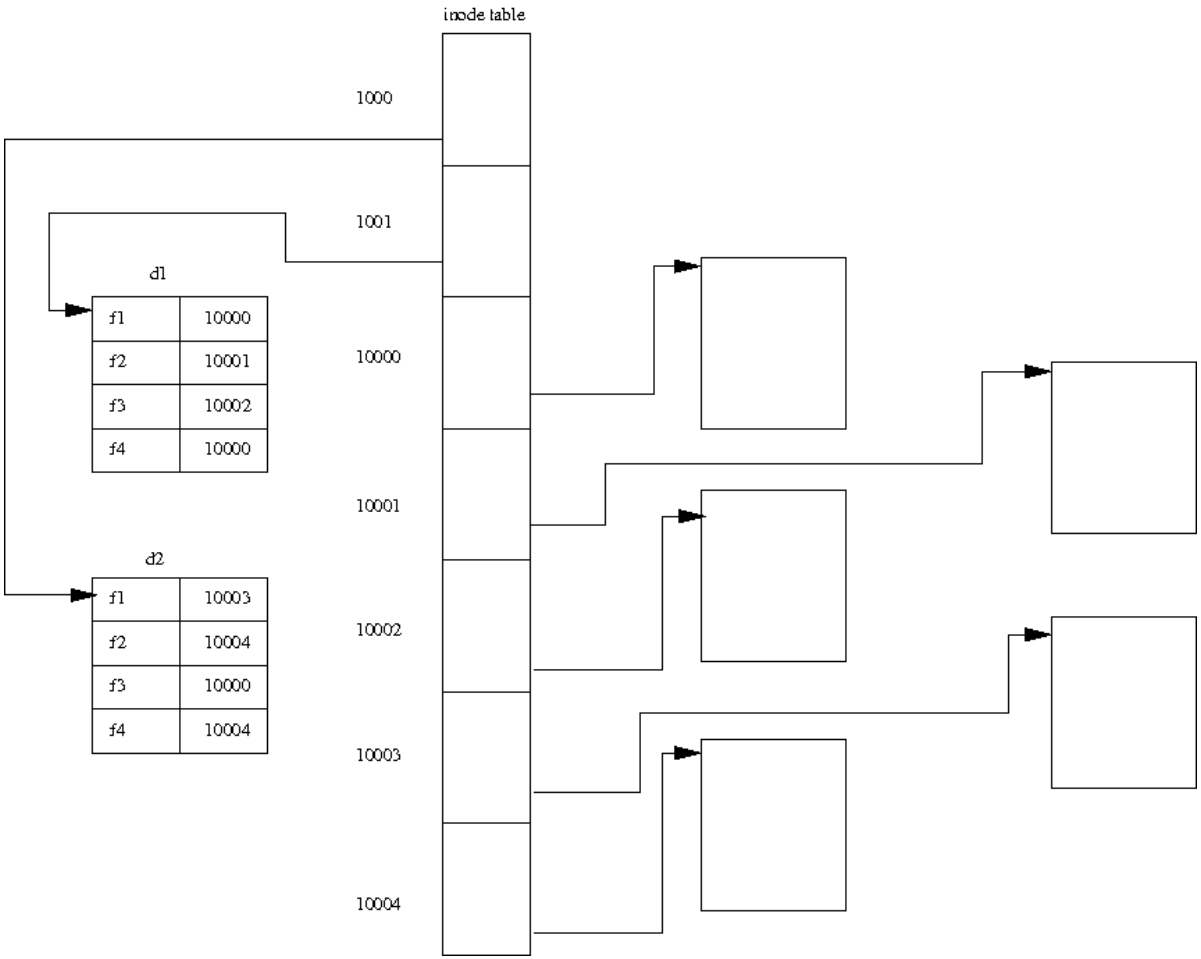
Il comando `ln`

- Comando `ln [-s] sorgente [destinazione]`
  - Si immagini, da una directory, di dare i seguenti comandi:

```
mkdir d1 d2
dd if=/dev/urandom of=d1/f1 bs=1M count=10
dd if=/dev/urandom of=d1/f2 bs=1M count=10
dd if=/dev/urandom of=d1/f3 bs=1M count=10
dd if=/dev/urandom of=d2/f1 bs=1M count=10
dd if=/dev/urandom of=d2/f2 bs=1M count=10
ln d1/f1 d1/f4
ln d1/f1 d2/f3
#sarebbe stato lo stesso ln d1/f4 d2/f3
ln d2/f2 d2/f4
```

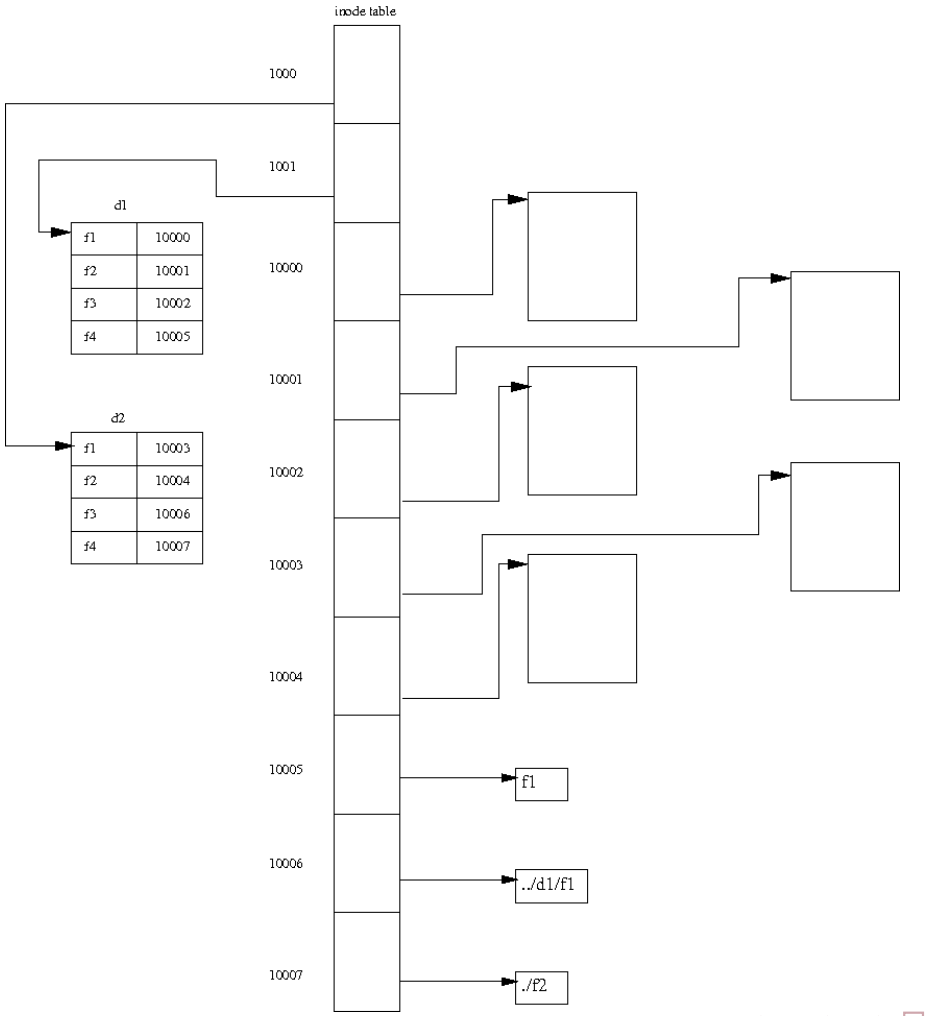
Il comando `ln`

- Il risultato è quello mostrato in Figura



Il comando `ln`

- Sostituendo ogni `ln` con `ln -s` il risultato è il seguente



Il comando `ln`

- Comando `ln [-s] sorgente [destinazione]`
  - Da notare che con `ln -s` è diverso fare il link a `d1/f4` o a `d1/f1`, infatti cambia il contenuto del file
  - **Esercizio**
    - verificare con esempi tutte le cose dette sopra
    - provare anche a fare l'hard link di un soft link e viceversa

Utenti, filesystem e file

Comandi per ottenere informazioni dal sistema

Il comando du

- Comando `du [opzioni] [--exclude=PATTERN] [file...]` visualizza lo spazio occupato sul disco da file e directory specificati come argomento
  - Se non ci sono argomenti, dà la dimensione della cwd
  - Per le directory, considera tutti i file nel sottoalbero
  - `-s` o `-a`: mostra solo il totale o tutti i files (di default, fa una cosa intermedia, mostrando tutte le eventuali sottodirectory)
  - `-c`, utile solo se c'è più di 1 argomento: fa la somma tra le dimensioni di tutti gli argomenti
  - `-h` (*human readable*), anziché stampare in bytes lo mostra arrotondato al kB, al MB, al GB
  - `--exclude=PATTERN`, toglie dal conto i file il cui nome soddisfa il pattern `PATTERN`
  - **Esercizio** verificare le opzioni sopra riportate su alcune directory standard di Linux (ad esempio `/etc`)

Il comando df

- Comando `df [opzioni] [file...]`: mostra la dimensione e l'attuale uso dei filesystem
  - Senza argomenti, li mostra tutti
  - Altrimenti, mostra solo quello che contiene `file`
  - `-h`: aggiunge alle dimensioni il suffisso (human readable): M per megabyte, G per Gigabyte, ecc.
  - `-H`: come h ma con le unità SI, potenze di 10 invece che base 2 (ad esempio 1000 invece di 1024)
  - `-l`: solo filesystem locali (ad es.: niente filesystem di rete)
  - `-i`: mostra quanti inode sono ancora disponibili prima che si riempia la inode table
- **Esercizio** vedere le statistiche di uso di tutti i filesystem, e poi solo del filesystem contenente i seguenti files: `/etc/passwd` e `/proc`

I comandi `free`, `top` e `uname`

- Comando `free [opzioni]`: mostra informazioni sulla memoria del sistema
- Utile se si vuole conoscere la memoria disponibile sul sistema, la memoria in uso e quella libera
  - `-b`: mostra la quantità di memoria in byte
  - `-k`: mostra la quantità di memoria in KB (default)
  - `-t`: mostra una riga contenente i totali
- Comando `top`: mostra informazioni sul sistema , processi in esecuzione e risorse del sistema, utilizzo di CPU, RAM spazio swap, ecc. (si esce con q)
- Comando `uname [opzioni]`: mostra informazioni sul sistema
  - `-a`: mostra tutte le informazioni del sistema
  - `-m`: mostra il tipo di macchina
  - `-s`: mostra il nome del kernel
  - `-t`: mostra il nome del sistema operativo