Sistemi Operativi, Secondo Modulo, Canale M–Z
Riassunto della lezione del 02/03/2017

Igor Melatti

Il filesystem

- 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 od 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
  - vedremo che esistono anche file *speciali* (non regolari)
- Linux ha un solo filesystem principale, che ha come radice la directory `/` (*root*)
  - tutti i file e le directory sono contenuti, direttamente od indirettamente, in tale directory
  - non esistono quindi i "volumi" di Windows
  - le foglie dell'albero possono essere o directory vuote o file
  - all'interno della stessa directory non ci possono essere due file, due directory, o 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`
- 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`

- C'è inoltre il concetto di *current working directory* (cwd), che vale per ogni processo
  - vale 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)
    - notare che `man cd` non funziona; vedremo in seguito il perché
  - 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

- A partire dalla cwd, si possono usare i *path relativi*
  - 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`

- Nel seguito, quando ad un comando dovrà essere dato come argomento un nome di un file o un nome di una directory, si intende che tale nome può essere un path relativo od assoluto

- Il comando `ls [-a] [-R] [nomedir]` mostra il contenuto della directory `nomedir` (o della cwd, se `nomedir` non è dato)
  - per ragioni storiche, i file con nomi che cominciano con il punto sono considerati "nascosti" (*hidden*)
  - `ls` li mostra solo se viene data l'opzione `-a`
  - con `-R` mostra tutto il sottoalbero con radice in `nomedir`
  - questo comando verrà ripreso in seguito

- Il comando `mkdir [-p] nomedir` crea la directory `nomedir` (vuota); il comando `touch nomefile` crea il file `nomefile` (vuoto)
  - se viene usata l'opzione `-p` e `nomedir` è un path con più di una directory, allora crea tutte le directory nel path
  - 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 `..`; per controllare il risultato usare il comando `ls`; controllare come cambia il path riportato nel prompt
  - per modificare un file, scrivendoci un testo dentro, dare il seguente comando `gedit nomefile &`; si aprirà un editor grafico standard, dove è possibile scrivere e salvare le modifiche. Per ora, ignorare il significato del carattere `&`.
- 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
- 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
  - filesystem virtuali, montati dal kernel per gestire le risorse
  - filesystem di rete
  - filesystem in memoria principale (RAM)
- Il trucco usato è quello del *mounting*
  - una qualsiasi directory dell'albero gerarchico può diventare il punto di mount per un (nuovo) filesystem
  - è meglio scegliere una directory vuota, altrimenti il suo "vecchio" contenuto non sarà più visibile fino all'`umount`
  - ad esempio, sulla directory `/proc`, durante il boot, viene montato un filesystem virtuale (non corrisponde ad alcun disco)

  - ad esempio, i filesystem di rete potrebbero essere montati su una directory `/nfs`, creata appositamente
  - ad esempio, un filesystem in memoria principale può essere montato su una directory all'interno della home di un utente
  - ad esempio, il disco principale, contenente l'installazione del sistema operativo, sarà montato su `/`
  - ad esempio, un disco secondario, senza l'installazione del sistema operativo (o con installato un altro sistema operativo!), può essere montato su `/windows`
  - 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: gli si dice di installarlo sulla partizione che era montata su `/`, gli si dice di montare la home così com'è, e ci si può risparmiare di dover ricopiare i vecchi dati della home: sono già lì
    * 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)

- Principali caratteristiche di un *tipo* di filesystem: dimensione massima di una partizione, dimensione massima di un file, lunghezza massima di un nome di file (e se supporta spazi, ma oramai lo fanno tutti), e se è journal o meno
  - 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
  - quando un disco, o una sua partizione, viene montata, occorre specificare il giusto filesystem (quello con cui è stato formattato)
  - i tipi più comuni di filesystem per Linux sono riportati in Tabella 1
  - 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)

Table 1: 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

Table 2: Le tipiche 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

  - è quindi sbagliato comprare un disco da 64 TB, formattarlo con ReiserFS e volerci fare una sola partizione

- Per sapere quali filesystem sono montati e dove: comando `mount` (senza argomenti), oppure anche `cat /etc/mtab`
  - 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, altrimenti scrive caratteri incomprensibili
- Per sapere quali filesystem vengono montati a tempo di boot (esclusi quelli gestiti dal kernel): `cat /etc/fstab`
- Lista delle directory più significative: Tabella 2
- Alcuni file importanti: `/etc/passwd` e `/etc/group`

- **esercizio:** far scrivere a schermo il contenuto di tali file
- 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 dei 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)
- 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)
- **esercizio:** verificare che `utente3` non sia presente in `/etc/passwd`, crearlo, e poi controllare che ora sia presente
- **esercizio:** verificare che `utente3` non sia nel gruppo `sudo`, aggiungerlo al gruppo `sudo`, e poi controllare che ora sia presente

I file

- Ripasso: Linux usa gli i-node per memorizzare file su disco
- Quindi, ogni file del filesystem (directory comprese) è rappresentato da una struttura dati `inode` (Figura 1), ed è univocamente determinato da un `inode number`
  - non ci sono mai contemporaneamente 2 file con lo stesso inode number (a meno che non siano hard link, vedere più sotto)
  - 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 (vedere Figura 2)
- La struttura dati `inode` contiene diversi campi, tra i quali quelli indicati in Tabella 3

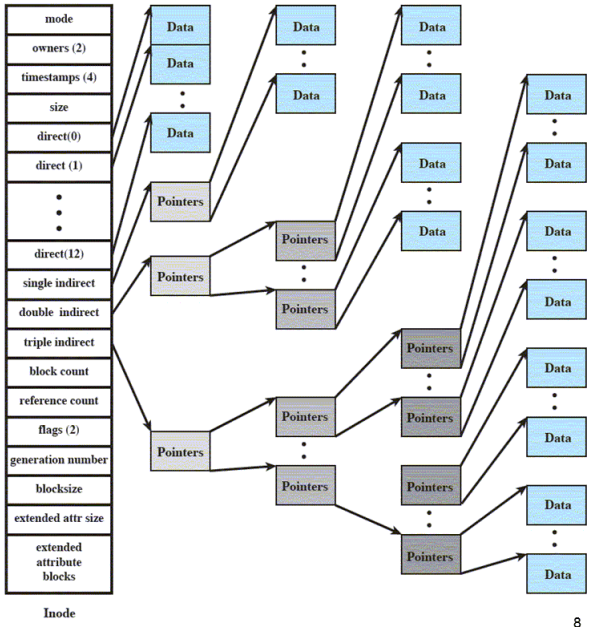

Figure 1: Inode tipico

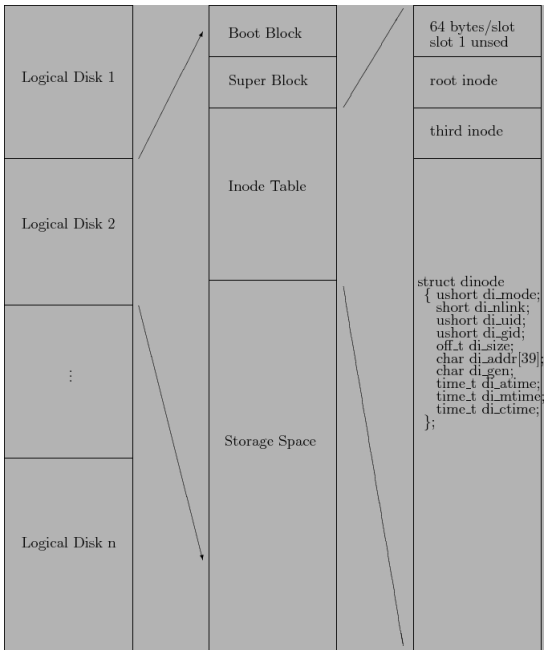

Figure 2: Inode table in un disco formattato con ext2

Table 3: I principali attributi della struttura `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 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

- Per vedere il valore di tali 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: mostrerà 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 di utente e gruppo, anziché il corrispettivo nome, usare l'opzione `-n`
  - user ID, group ID, dimensione e permessi sono visualizzati con l'opzione `-l`
  - per vedere i timestamp: sempre con l'opzione `-l`, ma in combinazione con `-c` (per ctime) o con `-u` (per atime) e senza niente (per mtime)
  - l'opzione `-l` mostra anche una riga "total" per ogni directory di cui mostra il contenuto: si tratta delle dimensione di quella directory in blocchi su disco (normalmente, 1 blocco = 4kB)
  - notare che queste ed altre informazioni non sono sul `man`: vanno cercate nelle informazioni estese, reperibili con il comando `info ls`
  - **esercizio:** creare un file, modificarlo, poi leggerlo e verificare che i valori per atime ed mtime cambino di conseguenza
- Il comando `ls` mostra un file per ogni riga; si può ottenere una visualizzazione più ampia con il comando `stat [-c format]` {`nomefile`}
  - usando opportunamente l'opzione `-c`, si può scegliere cosa far stampare, in maniera più certosina che con `ls`

Table 4: Permessi per un file in un sistema Linux

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 (altrimenti, per altre modifiche, bisognerebbe prima leggere); si può anche cancellare, ma occorrono opportuni diritti sulla directory (vedere Tabella 5)
`-wx`	3	Come il permesso 2
`r--`	4	Si può leggere
`r-x`	5	Si può leggere ed eseguire
`rw-`	6	Si può leggere e modificare a piacimento (ma occhio ancora alla cancellazione)
`rwx`	7	Si può fare tutto (ma occhio ancora alla cancellazione)

  - ad esempio, `-c %a` stampa solo l'access time del file dato (e solo quello)
  - ovviamente, funziona anche con i nomi di directory, ma dà solo informazioni sugli attributi dell'inode (quindi, non elenca il contenuto della directory)

- I permessi dei file: chi può far cosa
  - ogni file ha un utente ed un gruppo proprietari
  - solitamente, 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
  - il proprietario decide cosa permettere e cosa no agli altri utenti e agli altri gruppi, definendo i *permessi* di file e directory
- I permessi sono quelli di lettura, scrittura ed esecuzione
  - per un file, dovrebbe essere chiaro cosa significa, ma ci sono alcune cose non ovvie, quindi vedere Tabella 4
  - per una directory, la cosa è un po' più complicata, Tabella 5
  - altri permessi, o meglio attributi speciali: sticky bit (t), setuid bit (s), setgid bit (s)
    - lo sticky bit è ora inutile sui file (una volta, se applicato ad un file eseguibile, manteneva l'immagine del processo in memoria anche dopo che era terminato)

Table 5: Permessi per una directory in un sistema Linux

Permesso	**Ottale**	**Significato**
`---`	0	Non si può fare niente
`--x`	1	Si può settare come cwd; si può anche "attraversare", se già se ne conosce il contenuto (ad esempio, si può leggere un file o una directory al suo interno, se i permessi di questi ultimi contengono la lettura)
`-w-`	2	Non si può fare niente (per fare veramente modifiche, occorrono i permessi di esecuzione)
`-wx`	3	Come il permesso 7, ma non si può listare il contenuto (con o senza attributi)
`r--`	4	Si può solo listarne il contenuto, ma senza vedere gli attributi dei file/directories contenuti (l'unica cosa che si può sapere è 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)
`rwx`	7	Si può fare tutto: listare contenuto (attributi compresi), aggiungere file e directory, cancellare file contenuti in essa (anche senza avere il permesso di scrittura sul file! correggibile con lo sticky bit, vedere più avanti), cancellare directory contenute in essa (ma occorrono tutti i permessi su tali directory)

    * lo sticky bit, applicato su una directory, "corregge" il comportamento dei permessi write+execute (vedere Tabella 5): si possono cancellare files solo se si hanno i permessi di scrittura
    * 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 né ad u' né 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, sono necessari anche i permessi di scrittura su f per cancellare f.
    * il setuid bit si usa solo per i file eseguibili: quando vengono eseguiti, 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, viene eseguito con i privilegi di root, indipendentemente da chi lo ha eseguito
    * ad esempio, il mount/umount usato dall'interfaccia grafica per montare/smontare automaticamente CD e penne USB ha il setuid, altrimenti non funzionerebbe
    * lo stesso dicasi per il comando `passwd`, che permette ad un utente di modificare la propria password
    * il setgid è l'analogo ma con i gruppi (i privilegi sono quelli del gruppo che è proprietario del file eseguibile); può essere applicato anche ad una directory, e allora ogni file creato lì dentro ha il gruppo della directory, anziché quello primario di chi crea files
    * da un punto di vista di visualizzazione (provare a dare il comando `stat /tmp /usr/bin/passwd`), vengono visualizzati al posto del bit di esecuzione: il setuid nella terna utente (vedere più avanti), il setgid nella terna gruppo e lo sticky nella terna altro
    * se il permesso di esecuzione c'è, allora la "s" o la "t" saranno minuscoli, altrimenti saranno maiuscoli (ed inutili...)

- 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` (Figure 3)
  - **esercizio:** capire come mai `adduser` va usato solo da un utente amministratore, mentre `groups` no

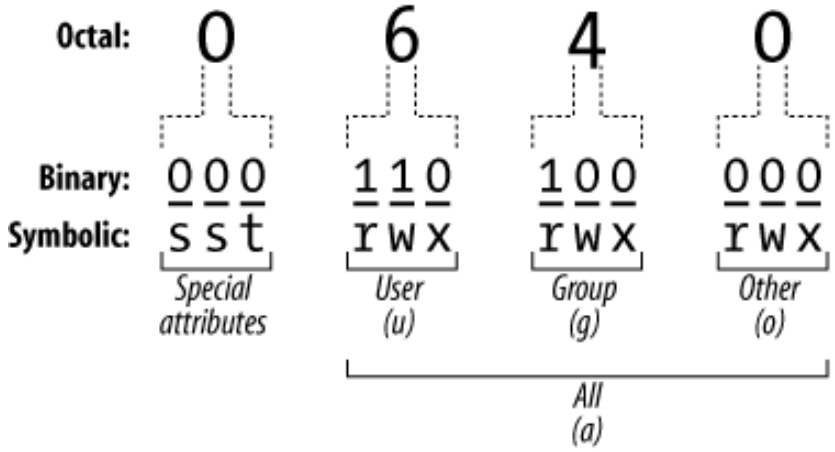

Figure 3: I permessi in Linux

- Comando `chmod mode[, mode...]  filename`
  - svariati modi di settare mode: ottale o con le lettere (simbolico)
  - ottale: 4 numeri tra 0 e 7, come nelle Tabelle 4 e 5; il primo numero indica setuid (4), setgid (2) e sticky (1), gli altri sono per utente, gruppo ed altri come detto sopra
  - 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
  - con le lettere: qui se ne possono specificare molti, separati da virgole
  - il formato di ogni modo simbolico è `[ugoa][+-=][perms...]`, dove `perms` è zero, una o più lettere nell'insieme {rxwXst}, oppure una lettera nell'insieme {ugo}. Capire dal manuale come funziona (attenzione alla `a`, che tiene conto del masking, di cui si tratterà più avanti)
  - **esercizio:** creare un file e settarne i permessi a `rws r-S -w-` e poi a `rwx r-- -wT` usando la modalità simbolica
  - **esercizio:** togliere il permesso di lettura ad un file, e poi provare ad parirlo con `gedit`
  - **esercizio:** il comando `chmod` modifica il ctime del file, verificarlo
- Comando `umask [mode]`: attenzione, è un comando della bash, non si trova in `man`; occorre invece fare `man bash` e cercare `umask`
  - setta la maschera dei file a `mode` se quest'ultimo è dato, altrimenti ritorna l'umask corrente

  - definisce (in negativo) i permessi per i nuovi file e directory: il permesso per un file o una directory appena creata sarà, in ottale, dell'operazione bit-a-bit 7777 AND NOT(umask)
  - con una piccola eccezione: quando si crea un file, il permesso di esecuzione non viene mai settato, su nessuna terna; c'è invece per le directory (a meno che non lo "blocchi" umask)
  - ha effetto anche su `chmod`, quando quest'ultimo viene usato in modo simbolico con controllore `a`
  - **esercizio:** modificare l'umask in modo che, sia che venga creata una directory che un file, i permessi siano 664. Dopodiché, modificare nuovamente in modo che il permesso per una nuova directory sia 775 e per un file sia 664.

- Comandi `chown [-R] proprietario {file}` e `chgrp [-R] gruppo {file}`
  - possono essere usati solo da root, quindi ci vuole `sudo`
    - altrimenti, ad esempio, uno potrebbe creare un file con contenuti compromettenti, e poi darlo ad un altro ignaro utente ...
  - 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`.

- Script per "giocare" con i permessi: `create_dirs_and_files.script`
  - creare una nuova directory e copiarci (o spostarci) dentro `create_dirs_and_files.script`
  - eseguire con il seguente comando: `bash create_dirs_and_files.script`
  - dopodiché, eseguire `ls -lR` (nota a margine: le opzioni si possono sempre mettere a fattor comune scrivendo una sola volta il dash e poi tutte le opzioni che si vogliono; attenzione, funziona solo per le opzioni con un solo dash, quelle con 2 dash vanno sempre staccate)
  - come si può vedere, ora ci sono 8 file, uno per ogni permesso, e 8 directory, una per ogni permesso
  - all'interno di ciascuna directory, si ripete lo schema: 8 file e 8 directory, ciascuna di queste ultime con dentro 8 file

– **esercizio:** testare i vari permessi, come sono riportati nelle tabelle della lezione 2. Ad esempio, provare a leggere un file solo eseguibile, o ad appendere testo ad un file solo leggibile, o ad attraversare una directory senza permesso di esecuzione...