Sistemi Operativi, Secondo Modulo, Canale A–L e Teledidattica
Riassunto della lezione del 27/02/2019

Igor Melatti

Il filesystem

- 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 `geany nomefile &`; si aprirà un editor grafico standard, dove è possibile scrivere e salvare le modifiche. Per ora, ignorare il significato del carattere `&`.

  - 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`
    * (avete veramente provato ad eseguire `vi nomefile` e non sapete come uscire? digitate i caratteri `:q` seguiti da invio)

- 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`)

- 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)

- Il trucco usato è quello del *mounting*
  - una qualsiasi directory dell'albero gerarchico può diventare il punto di mount per un altro (nuovo) filesystem
    * 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
    * altrimenti detto: il comando `ls` x mostrerà il contenuto della directory root di F
    * leggere o modificare un file dentro il sottoalbero radicato in F avrà l'effetto di leggerlo o modificarlo nella corrispondente directory di F
    * più formalmente: 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)

- è meglio scegliere una directory x vuota, altrimenti il suo "vecchio" contenuto non sarà più visibile fino all'`umount`
  - però tranquilli: per l'appunto, 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
- 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 su una qualche directory, e ci si può risparmiare di dover ricopiare i vecchi dati della home: sono già lì
    - se si tratta semplicemente di un'altra distribuzione Linux non c' problema
    - se invece si passasse a Windows, usare il filesystem `home` sarebbe complicato (Windows non riconosce nativamente i tipi di filesystem di Linux, come l'`ext2` o `ext3`)
  - 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)

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

- Principali caratteristiche, dal punto di vista dell'utente, 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
  - dal punto di vista del progettista-programmatore di sistemi operativi, ovviamente il tipo definisce anche la metodologia con cui i dati vengono letti/scritti sul disco (rivedere l'esempio sul FAT fatto nel modulo 1)
  - 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)
  - 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)
  - 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
  - comando `mount`, è quello da usare per fare il *mounting* descritto sopra

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

  - esempi di tale tipo verranno visti nella prossima lezione

- 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 di entrambi questi 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)
  - come venga strutturata e memorizzata tale tabella dipende, ovviamente, dal tipo di filesystem; qui parliamo della sua organizzazione *logica*
- La struttura dati `inode` contiene diversi campi, tra i quali quelli indicati in Tabella 3
- 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`
  - user name, group name, dimensione e permessi sono visualizzati con l'opzione `-l`
    - per la dimensione, è quanto viene effettivamente occupato da quel file su disco, in bytes
    - questo vuol dire che, per le directory, si mostra la dimensione del file speciale corrispondente, che consiste in una lista di coppie (nomefile, numero di inode)

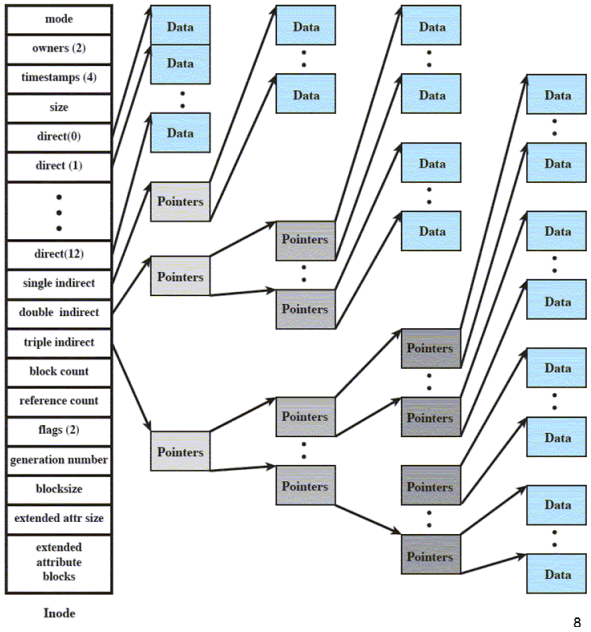

Figure 1: Inode tipico

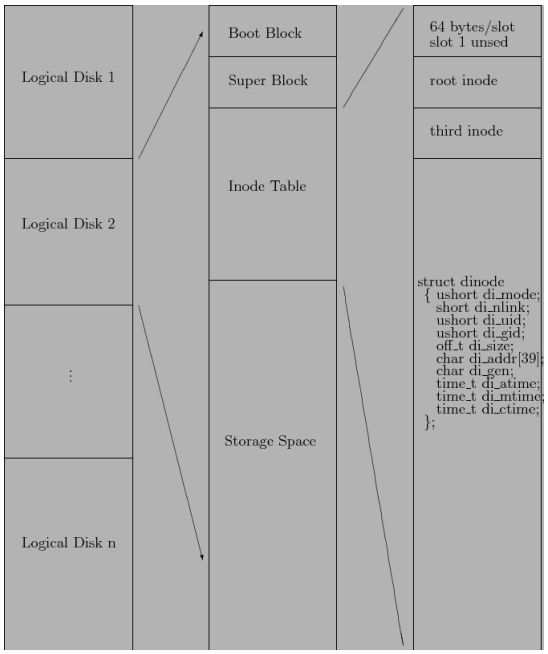

Figure 2: Inode table in un disco formattato con ext2

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

    - quindi, ai fini della dimensione di una directory, non ha importanza quanto siano grandi i file ivi contenuti, ma ha importanza *quanti* siano (e quanto siano lunghi i loro nomi)
    - solitamente, le directory hanno una dimensione minima di 4kB, anche se sono vuote
- per vedere l'ID (numerico) di utente e gruppo, anziché il corrispettivo nome, usare l'opzione `-n`
- 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 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
    - non si tratta dei blocchi con cui il kernel comunica con il disco, ma solo di una comodità per l'utente
    - ad esempio: per vedere l'output assumendo una dimensione dei blocchi di 100kB, basta scrivere `BLOCKSIZE=100kB ls -l`
        - sì, vedremo nelle prossime lezioni che è possibile preporre degli assegnamenti ai comandi
    - precisazione: "total" guarda solo la directory attuale, non tutto il sottoalbero
    - per ogni sottodirectory, ne considera solo la dimensione, come definita sopra
    - il risultato è anche approssimato per eccesso alla dimensione dei blocchi per il trasfimento dati su disco *per ciascun file*
        - per vedere qual è la dimensione dei blocchi, `stat -f --format=%s nomefile` con `nomefile` un file qualsiasi nella directory (leggere quanto detto sul comando `stat` più sotto)

· vedere anche poco più avanti

  - 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 [-f] [-c format]` {`nomefile`}
  - usando opportunamente l'opzione `-c`, si può scegliere cosa far stampare, in maniera più certosina che con `ls`
  - 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 come fa `ls`)
  - l'opzione `-f`, anziché fornire informazioni sul file, fornisce informazioni sul file system nel quale il file è salvato
  - notare che c'è un po' di confusione con le size dei blocchi: da un lato c'è l'`I/O Block`, che è la dimensione usata quando si leggono/scrivono file (`stat -f --format=%s nomefile`); dall'altra c'è la dimensione su disco di ogni blocco, che può essere diversa (`stat -f --format=%B nomefile`) e coincide con la dimensione di un *settore* di disco; a complicare le cose, moltiplicando dimensione del blocco su disco e numero di blocchi si può ottenere una size *maggiore* di quella effettiva del file (perché si alloca sempre almeno la dimensione di trasferimento, quindi c'è frammentazione interna)

- I permessi dei file: chi può far cosa
  - ogni file ha 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 esiste `chown` (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 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)

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)

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 (ma solo se il permesso c'è per *tutte* le directory nel path); 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 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)
    - 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 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 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 comando `passwd` ha il setuid, 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

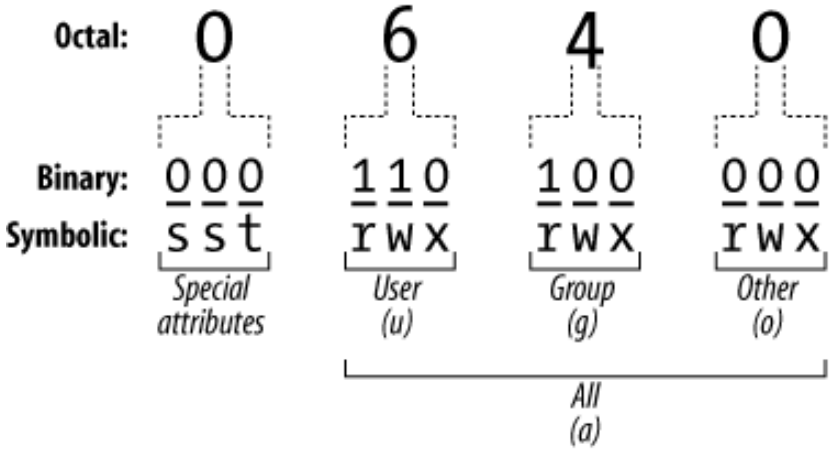

Figure 3: I permessi in Linux

- 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 a visualizzarlo con `cat` o con `geany`
  - **esercizio:** il comando `chmod` modifica il ctime del file, verificarlo
  - una precisazione sul ctime: dovrebbe essere modificato ogni volta che cambiano gli attributi. C'è però un'eccezione: se si accede in lettura al file si modificano gli attributi, perché viene modificato il timestamp di accesso; tuttavia, il ctime, in questo caso, non viene modificato

- 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
  - solo le 3 terne, niente attributi specili
  - definisce (in negativo) i permessi per i nuovi file e directory (solo sulle terne; gli attributi speciali iniziali sono sempre tutti a 0)
  - il permesso per una directory appena creata sarà il risultato, in ottale, dell'operazione bit-a-bit 777 AND NOT(umask)
  - il permesso per un file appena creato sarà il risultato, in ottale, dell'operazione bit-a-bit 666 AND NOT(umask)
  - ha effetto anche su `chmod`
  - **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`
  - 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...
- **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`...