Comandi su file e directory

`ls` mostra il contenuto della directory corrente

`ls -l` mostra il contenuto della directory corrente, forma lunga: permessi, proprietario e gruppo del file, dimensioni

`ls -R` mostra il contenuto della directory corrente e di tutte le directori in essa contenute

`ls <nome directory>` mostra il contenuto della directory specificata

`ls <nome file>` mostra il file indicato

tutte queste forme possono essere combinate ad es.: **`ls -lR /home`** mostra il contenuto di `/home` e di tutte le sotto-directory, in forma lunga

`pwd` (print working directory) mostra il nome della directory corrente (quella in cui vengono eseguiti i comandi se non viene specificata una directory particolare)

`cd <nome directory>` entra nella directory specificata

`cd` (senza parametri) torna nella home directory

`mkdir <nome>` crea una nuova directory (nella directory corrente)

`rmdir <nome>` rimuove la directory specificata (soltanto se è vuota!)

`cp <nome file> <nuovo nome file>` copia un file

`cp <nome file> <nome directory>` copia un file in un'altra directory

attenzione questi due modi di usare **`cp`** sembrano identici, in realtà sono molto diversi. Nel primo caso il secondo parametro è il nome che vogliamo dare alla copia del file, nel secondo caso è il nome di una directory in cui vogliamo creare la copia del file (che avrà lo stesso nome dell'originale). Possiamo anche combinare i due utilizzi: **`cp <nome file> <nome directory>/<nome file>`**: in questo caso creiamo una copia in un'altra directory e con un'altro nome. La seconda forma di **`cp`** accetta anche più di un nome di file: **`cp <file 1> <file 2> <...> <directory>`**. In questo caso tutti i file specificati vengono copiati nella directory indicata, mantenendo il loro nome.

`mv <nome file> <nuovo nome>` rinomina un file

`mv <nome file> <nome directory>` sposta un file in un'altra directory

attenzione valgono le stesse cose dette per il comando **`cp`**

`rm <nome file>` cancella il file. **N.B.:** usando **`rm`** il file viene proprio *cancellato*, non spostato in un cestino o cose simili!

per i comandi `cp`, `mv` e `rm` si possono usare le opzioni **`-f`** (force) e/o **`-r`** (recursive): **`-r`** applica il comando anche a tutte le sottodirectory e ai file in esse contenuti, **`-f`** significa «fai proprio quello che ho detto, anche se è pericoloso»: ad esempio **`rm -fr /home/<tuo nome>`** cancella *tutto* il contenuto della tua home directory, senza chiederti nulla e senza possibilità di recuperare i file!

`file <nome file>` analizza il file e mostra informazioni sul suo contenuto

`du <directory>` calcola lo spazio occupato dai file contenuti nella directory indicata e in tutte le sue sottodirectory

Struttura del *filesystem*

`/`	la *radice* del filesystem
`/bin`	alcuni comandi fondamentali (ad es. **`ls`**, **`grep`**, ...)
`/boot`	file necessari all'avvio del sistema (*bootstrap*)
`/etc`	file di configurazione
`/home`	le home directory degli utenti
`/lib`	librerie
`/mnt` e/o `/media`	qui compaiono i filesystem contenuti su supporti esterni (es. dischetti, CD, ...)
`/root`	la home directory dell'amministratore
`/sbin`	alcuni comandi fondamentali per l'amministrazione del sistema
`/tmp`	spazio per i file temporanei: chiunque può creare file in questa directory, ma questi vengono automaticamente cancellati dal sistema dopo un certo tempo
`/usr`	la maggior parte dei comandi e dei programmi in generale, in particolare:
`/usr/bin`	comandi
`/usr/X11R6`	programmi «grafici»
`/usr/include`	la maggior parte degli header file (`*.h`)
`/usr/share/man`	i manuali
`/var`	«dati variabili»ad es. messaggi di posta elettronica non ancora letti, file in corso di stampa, ...

Scrivere i comandi

La *shell* di Linux mette a disposizione molte funzioni per semplificare la scrittura dei comandi

frecce orizzontali spostano il cursore sulla linea che si sta scrivendo

frecce verticali richiamano i comandi precedenti/successivi

ctrl-a vai all'inizio della riga

ctrl-e vai alla fine della riga

ctrl-k cancella dal cursore alla fine della riga

ctrl-u cancella dal cursore all'inizio della riga

ctrl-c annulla la riga (senza cancellarla)

tab cerca di completare la parola che stai scrivendo con un nome di comando o con un nome di file; se non è possibile completare in maniera univoca, premando di nuovo `tab` si ottiene la lista dei possibili comandi o nomi di file[1]

`history` mostra la lista di tutti i comandi eseguiti

`!<numero>` ripete il comando *n*-esimo (il numero del comando lo trovi usando `history`)

`!!` ripete l'ultimo comando

I caratteri `*` e `?` e altri caratteri speciali

In tutti i casi in cui un comando accetta come parametro uno o più nomi di file, possiamo usare i caratteri **`*`** e **`?`** per indicare molti file in modo più semplice e compatto: **`?`** significa «qualunque carattere», **`*`** significa «qualunque stringa di caratteri». Ad es. **`ls a*`** mostra la lista di tutti i file il cui nome inizia per 'a'.

Allo stesso le parentesi graffe possono essere usate per comporre liste di nomi (di file) simili:
`cp myprogram.{c,h} /tmp/test/` è equivalente a **`cp myprogram.c myprogram.h /tmp/test/`**.

[1] premendo due volte `tab` all'inizio di una riga vuota si ottiene la lista di tutti i comandi disponibili!

Altri comandi utili

man <*nome comando*> mostra il manuale del comando indicato. Durante la visualizzazione del manuale sono disponibili molti comandi di navigazione:

- **q** esce dal manuale
- **h** mostra il manuale del manuale, cioè i comandi di navigazione
- **<*frecce*>** sposta avanti e indietro il testo
- **p** torna in cima
- **G** va' in fondo
- **/<*parola*>** cerca (verso il basso) la parola indicata
- **?<*parola*>** cerca (verso l'alto) la parola indicata
- **n** ripete l'ultima ricerca effettuata con **/** o **?**
- **N** ripete l'ultima ricerca effettuata, nella direzione opposta

man -k <*parola*> cerca tutti i manuali nel cui titolo è presente la parola cercata

<*comando*> | grep <*parola*> passa tutto l'output del comando a **grep**, che mostra soltanto le righe in cui è presente la parola cercata (utile ad es. per *filtrare* l'output di **man -k** o **ls -R**)

<*comando*> | less passa tutto l'output del comando a **less**, che permette di scorrere comodamente il testo. **less** usa gli stessi comandi di navigazione di **man** (anzi, `man` usa `less` per mostrare i manuali)

clear cancella lo schermo del terminale (lo stesso effetto si ottiene premendo ctrl-L)

grep

grep può filtrare l'output di un altro comando, oppure può andare a leggerlo direttamente da uno o più file. Il più semplica comando che possiamo usare per passare dei dati a **grep** è **cat <*file*>** che legge il file indicato e lo riproduce in output (attenzione: se usate `cat` su un file binario possono succede cose strane al vostro teminale)

cat <*file*> | grep <*parola*> oppure

grep <*file*> <*parola*> mostra tutte le righe del file in cui è presente la parola indicata

grep -i effettua la ricerca senza tener conto di maiuscole/minuscole

grep -color evidenzia nell'output la parola cercata (attenzione, potrebbe non funzionare su alcuni terminali!)

grep -c non mostra nessun testo; conta solamente il numero di righe in cui è presente la parola cercata

grep -v seleziona le righe che *non* contengono la parola indicata

grep -B <*numero*> , **grep -A <*numero*>**, **grep -C <*numero*>** mostra (anche) il numero indicato di righe prima (rispettivamente dopo, attorno) ciascuna riga selezionata

esempio grep -C 2 -i -color grep schede.tex
cerca tutte le righe in cui compare il termine `grep`, scritto con qualunque combinazione di maiuscole e minuscole, nel file `schede.tex` (questo file!), e mostra anche le due righe precedenti e successive a ogni riga trovata, evidenziando la parola cercata

Cercare un file

`locate <nome>` cerca tutti i file che contengono la parola indicata

Attenzione:

1. non tutte le distribuzioni di Linux installano **`locate`**; la maggior parte degli altri Unix *non* lo installa
2. `locate` è molto veloce perché non effettua la ricerca nel filesystem, ma in un database che viene aggiornato una volta al giorno; quindi i file che sono stati aggiunti dopo l'ultimo aggiornamento del database non possono essere trovati
3. in genere il database *non* contiene un indice completo di tutto il filesystem; in particolare non vengono indicizzati i file in `/tmp`
4. spesso `locate` trova un numero enorme di file (ad es. sul computer su cui sto scrivendo **`locate png`** trova 15416 file); bisogna allora cambiare termine di ricerca, oppure filtrare l'output di `locate` con `grep`, oppure usare altri comandi di ricerca (vedi sotto)

`find` permette di fare ricerche molto più mirate, ma ha una sintassi più complessa: **`find <base> <espressione>`** dove `<base>` è la directory da cui iniziare la ricerca e `<espressione>` è la descrizione di quello che vogliamo cercare. Vediamo alcuni esempi:

`find . -name 'a*'` cerca a partire dalla directory corrente tutti i file il cui nome inizia per 'a' (ricorda il significato di `*`; nota l'uso degli apici attorno a `a*`: sono necessari tutte le volte che usiamo `*` oppure `?`)

`find . -iname 'a*'` come sopra, ma senza tener conto della differenza tra maiuscole e minuscole

`find /tmp -mtime +5` cerca in `/tmp` i file che sono stati modificati più di 5 giorni fa

`find /tmp -mtime 5` cerca in `/tmp` i file che sono stati modificati 5 giorni fa

`find /tmp -mtime -5` cerca in `/tmp` i file che sono stati modificati meno di 5 giorni fa

`find /tmp -mtime 0 -and -name 'gpg*'` cerca in `/tmp` i file il cui nome inizia con `gpg` e che sono stati modificati nelle ultime 24 ore