appunti.md

LEZIONE 1

INTRO

La Shell è un programma che prende un'input da tastiera e lo trasferisce all'OS. Solitamente preinstallata nelle distribuzioni Linux si ha BASH, *Bourne Again SHell*. Quando si usa una GUI si necessita di un programma, **l'emulatore di terminale** per interagire con la shell. Quando la shell è pronta a ricevere l'input si ha lo **shell prompt**:

```
[me@linuxbox ~]$
```

Se alla fine si ha il simbolo # si indica che si hanno i permessi di *superuser*:

```
[me@linuxbox ~]#
```

Nella shell si ha la *history* dei vecchi comandi (solitamente fino a 1000) utilizzando le frecce su e giù. Con le frecce destra e sinistra ci si sposta col cursore nel comando. Vediamo qualche comando base d'esempio:

- Per ora e data corrente:

  ```
  [me@linuxbox ~]$ date
  Thu Mar 8 15:09:41 EST 2018
  ```

- Per il calendario:

  ```
  [me@linuxbox ~]$ cal
  March 2018
  Su Mo Tu We Th Fr Sa
                1  2  3
   4  5  6  7  8  9  10
  11 12 13 14 15 16 17
  18 19 20 21 22 23 24
  25 26 27 28 29 30 31
  ```

- Per vedere l'uso di memoria sui dischi e sulle varie partizione:

  ```
  [me@linuxbox ~]$ df
  File system    1K-blocks       Used  Available Use% Mounted on
  dev               4014968          0   4014968   0% /dev
  run               4023532       9784   4013748   1% /run
  /dev/sda28      131449884  112330780  12412112  91% /
  tmpfs             4023532      36680   3986852   1% /dev/shm
  ```

- Per vedere l'uso della RAM:

  ```
  [me@linuxbox ~]$ free
                total        used        free      shared     buff/cache  available
  Mem:        8047068     2504188     1657424      357836     3885456     4839276
  Swap:             0           0           0
  ```

- Per terminare la sessione uso o Ctrl-d o:

  ```
  [me@linuxbox ~]$ exit
  ```

Inoltre si hanno delle sessioni di temrinale che girano sempre dietro la gui e sono accessibili con Ctrl-Alt + i tasti da F1 a F6; generalmente con Alt-F7 si torna alla GUI.

NAVIGAZIONE

Nei sistemi UNIX i file sono organizzati in una *hierical directory structure*, quindi in una struttura ad albero. La prima cartella è quindi chiamata *root directory* e contiene files e altre sottocartelle. I sistemi UNIX e UNIX-Like hanno un unico albero per tutti i device i quali vengono in caso montati in un determinato punto dell'albero a seconda del volere di chi gestisce il sistema.

Per vedere in che posizione dell'labero ci troviamo (ovvero in che cartella ci troviamo) si ha:

```
[me@linuxbox ~]$ pwd
/home/me
```

Inoltre all'avvio ci troveremo nella cartella */home*, diversa per ciascun account. Infatti dentro */home* si hanno le varie sottocartelle degli utenti (una per ogni utente). Per vedere cosa contiene una cartella, elencando files e directories uso:

```
[me@linuxbox ~]$ ls
Desktop Documents file.txt Music Pictures Public Templates Videos
```

Specificando il percorso possiamo vedere il contenuto di una determinata cartella

```
[me@linuxbox ~]$ ls Documents
document.txt other
```

per elencare anche file e directory nascoste uso:

```
[me@linuxbox ~]$ ls -a Documents
document.txt other .test .test.d
```

possiamo specificare più directories:

```
[me@linuxbox ~]$ ls ~ Documents
/home/me:
Desktop Documents file.txt Music Pictures Public Templates Videos
/home/me/Documents:
document.txt other
```

possiamo anche richiedere informazioni sui file:

```
[me@linuxbox ~]$ ls -l Documents
drwxr-xr-x 2 me me 4096 30 apr 11.37 other
-rw-r--r-- 2 me me 4096 30 apr 11.37 document.txt
```

con:

Field	Meaning
`-rw-r--r--`	Access rights to the file. The first character indicates the type of file. Among the different types, a leading dash means a regular file, while a “d” indicates a directory. The next three characters are the access rights for the file's owner, the next three are for members of the file's group, and the final three are for everyone else. Chapter 9 "Permissions" discusses the full meaning of this in more detail.
`1`	File's number of hard links. See the sections "Symbolic Links" and "Hard Links" later in this chapter.
`root`	The username of the file's owner.
`root`	The name of the group that owns the file.
`32059`	Size of the file in bytes.
`2007-04-03 11:05`	Date and time of the file's last modification.
`oo-cd-cover.odf`	Name of the file.

Il percorso di file o di una directory può essere specificato in due modi:

1. **con il PERCORSO ASSOLUTO**, ovvero indicando ogni cartella e sotto cartella fino a quella desiderata, per esempio

/home/Documents/. Si ricorda che il primo "/" indica la cartella di root

2. **con il PERCORSO RELATIVO**, che invece da partiren dalla cartella di root parte dalla cartella corrente. Si hanno anche "." per indicare la cartella corrente e ".." per indicare quella precedente, quindi:

```
[me@linuxbox ~]$ pwd
/home/me
[me@linuxbox ~]$ ls
Documents
[me@linuxbox ~]$ cd ./Documents
[me@linuxbox Documents]$ pwd
/home/me/Documents
[me@linuxbox Documents]$ cd ..
[me@linuxbox ~]$ pwd
/home/me
[me@linuxbox ~]$ cd Documents
[me@linuxbox Documents]$ pwd
/home/me/Documents
```

Shortcut	Result
`cd`	Changes the working directory to your home directory.
`cd -`	Changes the working directory to the previous working directory.
`cd ~user_name`	Changes the working directory to the home directory of *user_name*. For example, `cd ~bob` will change the directory to the home directory of user "bob."

In UNIX i nomi di file e directory sono *case sensitive* e non si usano le estensioni per determinare il contenuto di un file. Per comodità **NON SI USANO SPAZI NEI NOMI DI FILES E DIRECTORIES**

Come abbiamo visto si possono dare opzioni ai vari comandi con il simbolo "-" seguito da una lettera, oppure, in modalità estesa, con "--" seguito dal nome dell'opzione. Per *ls* si ha, per esempio:

Option	Long Option	Description
`-a`	`--all`	List all files, even those with names that begin with a period, which are normally not listed (that is, hidden).
`-A`	`--almost-all`	Like the `-a` option above except it does not list `.` (current directory) and `..` (parent directory).
`-d`	`--directory`	Ordinarily, if a directory is specified, `ls` will list the contents of the directory, not the directory itself. Use this option in conjunction with the `-l` option to see details about the directory rather than its contents.
`-F`	`--classify`	This option will append an indicator character to the end of each listed name. For example, a forward slash (/) if the name is a directory.
`-h`	`--human-readable`	In long format listings, display file sizes in human readable format rather than in bytes.
`-l`		Display results in long format.
`-r`	`--reverse`	Display the results in reverse order. Normally, `ls` displays its results in ascending alphabetical order.
`-S`		Sort results by file size.
`-t`		Sort by modification time.

Per determinare il tipo di file, ricordando che l'estensione non implica alcun tipo preciso di file, si ha il comando file:

```
[me@linuxbox ~]$ file picture.jpg
picture.jpg: JPEG image data, JFIF standard 1.01
```

In Linux *everything is a file* e questo comporta l'esistenza di migliaia di tipi di file.

Per visualizzare file di testo si ha *less*, che permette di spostarsi su e giù con le frecce per visualizzaree il contenuto (per uscire usare "q"):

```
[me@linuxbox ~]$ less /etc/passwd
```

vediamo una tabella con le shortcut principali:

Command	Action
Page Up or `b`	Scroll back one page
Page Down or space	Scroll forward one page
Up arrow	Scroll up one line
Down arrow	Scroll down one line
`G`	Move to the end of the text file
`1G` or `g`	Move to the beginning of the text file
/characters	Search forward to the next occurrence of *characters*
`n`	Search for the next occurrence of the previous search
`h`	Display help screen
`q`	Quit `less`

Torniamo sul directory tree, si hanno le seguenti direcory principali:

Directory	Comments
`/`	The root directory. Where everything begins.
`/bin`	Contains binaries (programs) that must be present for the system to boot and run.
`/boot`	Contains the Linux kernel, initial RAM disk image (for drivers needed at boot time), and the boot loader. Interesting files: • `/boot/grub/grub.conf` or `menu.lst`, which are used to configure the boot loader. • `/boot/vmlinuz` (or something similar), the Linux kernel

Directory	Comments
`/dev`	This is a special directory that contains *device nodes*. “Everything is a file” also applies to devices. Here is where the kernel maintains a list of all the devices it understands.
`/etc`	The `/etc` directory contains all of the system-wide configuration files. It also contains a collection of shell scripts that start each of the system services at boot time. Everything in this directory should be readable text. Interesting files: While everything in `/etc` is interesting, here are some all-time favorites: • `/etc/crontab`, a file that defines when automated jobs will run. • `/etc/fstab`, a table of storage devices and their associated mount points. • `/etc/passwd`, a list of the user accounts.
`/home`	In normal configurations, each user is given a directory in `/home`. Ordinary users can only write files in their home directories. This limitation protects the system from errant user activity.
`/lib`	Contains shared library files used by the core system programs. These are similar to dynamic link libraries (DLLs) in Windows.
`/lost+found`	Each formatted partition or device using a Linux file system, such as ext4, will have this directory. It is used in the case of a partial recovery from a file system corruption event. Unless something really bad has happened to our system, this directory will remain empty.
`/media`	On modern Linux systems the `/media` directory will contain the mount points for removable media such as USB drives, CD-ROMs, etc. that are mounted automatically at insertion.
`/mnt`	On older Linux systems, the `/mnt` directory contains mount points for removable devices that have been mounted manually.

Directory	Comments
/opt	The /opt directory is used to install “optional” software. This is mainly used to hold commercial software products that might be installed on the system.
/proc	The /proc directory is special. It's not a real file system in the sense of files stored on the hard drive. Rather, it is a virtual file system maintained by the Linux kernel. The “files” it contains are peepholes into the kernel itself. The files are readable and will give us a picture of how the kernel sees the computer.
/root	This is the home directory for the root account.
/sbin	This directory contains “system” binaries. These are programs that perform vital system tasks that are generally reserved for the superuser.
/tmp	The /tmp directory is intended for the storage of temporary, transient files created by various programs. Some configurations cause this directory to be emptied each time the system is rebooted.
/usr	The /usr directory tree is likely the largest one on a Linux system. It contains all the programs and support files used by regular users.
/usr/bin	/usr/bin contains the executable programs installed by the Linux distribution. It is not uncommon for this directory to hold thousands of programs.
/usr/lib	The shared libraries for the programs in /usr/bin.
/usr/local	The /usr/local tree is where programs that are not included with the distribution but are intended for system-wide use are installed. Programs compiled from source code are normally installed in /usr/local/bin. On a newly installed Linux system, this tree exists, but it will be empty until the system administrator puts something in it.
/usr/sbin	Contains more system administration programs.
/usr/share	/usr/share contains all the shared data used by programs in /usr/bin. This includes things such as default configuration files, icons, screen backgrounds, sound files, etc.

Directory	Comments
/usr/share/doc	Most packages installed on the system will include some kind of documentation. In /usr/share/doc, we will find documentation files organized by package.
/var	With the exception of /tmp and /home, the directories we have looked at so far remain relatively static, that is, their contents don't change. The /var directory tree is where data that is likely to change is stored. Various databases, spool files, user mail, etc. are located here.
/var/log	/var/log contains *log files*, records of various system activity. These are important and should be monitored from time to time. The most useful ones are /var/log/messages and /var/log/syslog. Note that for security reasons on some systems only the superuser may view log files.

Vediamo un dettaglio:

```
[me@linuxbox ~]$ ls -l
lrwxrwxrwx 1 root root 11 2007-08-11 07:34 libc.so.6 -> libc-2.6.so
```

quella "l" all'inizio significa che abbiamo di fronte un link simbolico (*symlink*), con il link chiamato *libc.so.6* che punta alla libreria condivisa *libc.2.6.so*.

MANIPOLAZIONE DEI FILE

Prima di tutto introduciamo una feature della shell essenziale: le **wildcards**, ovvero dei caratteri speciali che si usano per specificare l'uso di più file in vase ad una certa richiesta:

Wildcard	Meaning
*	Matches any characters
?	Matches any single character
[*characters*]	Matches any character that is a member of the set *characters*
[!*characters*]	Matches any character that is not a member of the set *characters*
[[:*class*:]]	Matches any character that is a member of the specified *class*

con (si usano anche le solite regole delle regex come [a-z] per indicare un carattere da a e z (minuscoli), [a-z]*, per indicare n caratteri tra a e z (minuscoli) etc...):

Character Class	Meaning
[:alnum:]	Matches any alphanumeric character
[:alpha:]	Matches any alphabetic character
[:digit:]	Matches any numeral
[:lower:]	Matches any lowercase letter
[:upper:]	Matches any uppercase letter

Pattern	Matches
*	All files
g*	Any file beginning with “g”
b*.txt	Any file beginning with “b” followed by any characters and ending with “.txt”
Data???	Any file beginning with “Data” followed by exactly three characters
[abc]*	Any file beginning with either an “a”, a “b”, or a “c”
BACKUP.[0-9][0-9][0-9]	Any file beginning with “BACKUP.” followed by exactly three numerals
[[:upper:]]*	Any file beginning with an uppercase letter
[![:digit:]]*	Any file not beginning with a numeral
*[[:lower:]123]	Any file ending with a lowercase letter or the numerals “1”, “2”, or “3”

Per creare una directory usiamo *mkdir* seguito dal nome della cartella, o di più cartelle:

```
[me@linuxbox ~]$ ls
dir1 dir2
[me@linuxbox ~]$ mkdir dir3 dir4
[me@linuxbox ~]$ ls
dir1 dir2 dir3 dir4
```

Per copiare si ha il comando *cp*. Si può copiare il file/direcory in un altro/a con (da source a dest):

```
[me@linuxbox ~]$ cp source dest
```

si possono copiare più files mettendoli in sequenza (o usando una wildcard). Si hanno le seguenti opzioni:

Option	Long Option	Meaning
`-a`	`--archive`	Copy the files and directories and all of their attributes, including ownerships and permissions. Normally, copies take on the default attributes of the user performing the copy. We'll take a look at file permissions in Chapter 9 "Permissions."
`-i`	`--interactive`	Before overwriting an existing file, prompt the user for confirmation. **If this option is not specified, `cp` will silently (meaning there will be no warning) overwrite files.**
`-r`	`--recursive`	Recursively copy directories and their contents. This option (or the `-a` option) is required when copying directories.
`-u`	`--update`	When copying files from one directory to another, only copy files that either don't exist or are newer than the existing corresponding files, in the destination directory. This is useful when copying large numbers of files as it skips files that don't need to be copied.
`-v`	`--verbose`	Display informative messages as the copy is performed.

Vediamo qualche esempio:

Command	Results
`cp file1 file2`	Copy *file1* to *file2*. **If *file2* exists, it is overwritten with the contents of *file1*.** If *file2* does not exist, it is created.
`cp -i file1 file2`	Same as previous command, except that if *file2* exists, the user is prompted before it is overwritten.
`cp file1 file2 dir1`	Copy *file1* and *file2* into directory *dir1*. The directory *dir1* must already exist.
`cp dir1/* dir2`	Using a wildcard, copy all the files in *dir1* into *dir2*. The directory *dir2* must already exist.
`cp -r dir1 dir2`	Copy the contents of directory *dir1* to directory *dir2*. If directory *dir2* does not exist, it is created and, after the copy, will contain the same contents as directory *dir1*. If directory *dir2* does exist, then directory *dir1* (and its contents) will be copied into *dir2*.

Per spostare un file in un altro uso *mv*:

```
[me@linuxbox ~]$ mv source dist
```

Si hanno le seguenti opzioni:

Option	Long Option	Meaning
`-i`	`--interactive`	Before overwriting an existing file, prompt the

Vediamo qualche esempio:

		user for confirmation. **If this option is not specified, `mv` will silently overwrite files.**
`-u`	`--update`	When moving files from one directory to another, only move files that either don't exist, or are newer than the existing corresponding files in the destination directory.
`-v`	`--verbose`	Display informative messages as the move is performed.

Command	Results
`mv file1 file2`	Move *file1* to *file2*. **If *file2* exists, it is overwritten with the contents of *file1*.** If *file2* does not exist, it is created. **In either case, *file1* ceases to exist.**
`mv -i file1 file2`	Same as the previous command, except that if *file2* exists, the user is prompted before it is overwritten.
`mv file1 file2 dir1`	Move *file1* and *file2* into directory *dir1*. The dierctory *dir1* must already exist.
`mv dir1 dir2`	If directory *dir2* does not exist, create directory *dir2* and move the contents of directory *dir1* into *dir2* and delete directory *dir1*. If directory *dir2* does exist, move directory *dir1* (and its contents) into directory *dir2*.

Per eliminare file e direcories (con -r) uso *rm*:

```
[me@linuxbox ~]$ ls
file.txt file2.txt
[me@linuxbox ~]$ rm file2.txt
[me@linuxbox ~]$ ls
file.txt
```

Si hanno le seguenti opzioni:

Option	Long Option	Meaning
`-i`	`--interactive`	Before deleting an existing file, prompt the user for confirmation. **If this option is not specified, `rm` will silently delete files.**
`-r`	`--recursive`	Recursively delete directories. This means that if a directory being deleted has subdirectories, delete them too. To delete a directory, this option must be specified.
`-f`	`--force`	Ignore nonexistent files and do not prompt. This overrides the `--interactive` option.
`-v`	`--verbose`	Display informative messages as the deletion is performed.

Vediamo qualche esempio:

Command	Results
`rm file1`	Delete *file1* silently.
`rm -i file1`	Same as the previous command, except that the user is prompted for confirmation before the deletion is performed.
`rm -r file1 dir1`	Delete *file1* and *dir1* and its contents.
`rm -rf file1 dir1`	Same as the previous command, except that if either *file1* or *dir1* do not exist, `rm` will continue silently.

per creare link, sia hard links che symlinks, uso *ln*. Per un link normale:

```
[me@linuxbox ~] ln file link
```

Per un link simbolico:

```
[me@linuxbox ~] ln -s file link
```

COMANDI PER LA PRODUTTIVITÀ

I comandi possono essere built-in della shell, programmi, alias o funzioni della shell. Con *type* possono essere identificati:

```
[me@linuxbox ~]$ type type
type is a shell builtin
[me@linuxbox ~]$ type ls
ls is aliased to `ls --color=tty'
[me@linuxbox ~]$ type cp
cp is /bin/cp
[me@linuxbox ~]$ type emacs
emacs is /usr/bin/emacs
```

Per determinare dove si trova un eseguibile (solo un eseguibile non un alias o altro) uso *which*:

```
[me@linuxbox ~] which ls
/bin/ls
```

Ci sono vari modi per ottenere per ottenere la documentazione di un comando.

Bash ha il comando built-in *help*:

```
[me@linuxbox ~]$ help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

        Change the current directory to DIR.
    value of the HOME shell variable.

    ...
    Options:
       -L force symbolic links to be followed: resolve symbolic
       links in DIR after processing instances of `..'
       ...
```

Molti eseguibili hanno *--help*:

```
[me@linuxbox ~]$ mkdir --help
Usage: mkdir [OPTION] DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.
   -Z, --context=CONTEXT (SELinux) set security context to CONTEXT
   ...
```

Quasi tutti hanno la *man page*, richiamabile con *man*:

```
[me@linuxbox ~]$ man ls
```

Una man page è divisa in 8 sezioni:

1. User commands
2. Programming interfaces for kernel system calls
3. Programming interfaces to the C library
4. Special files such as device nodes and drivers
5. File formats
6. Games and amusements such as screen savers

7. Miscellaneous
8. System administration commands

richiamabili, per esempio per il *file formats* di passwd si ha:

```
[me@linuxbox ~]$ man 5 passwd
```

Con *apropos* si cerca una parola chiave nei vari man, ottenendo i comandi più consoni, tra parenetsi è indicata la sezione:

```
[me@linuxbox ~]$ apropos partiton
addpart (8)   - simple wrapper around the "add partition"...
all-swaps (7) - event signalling that all swap partitions...
cfdisk (8)    - display or manipulate disk partition table
cgdisk (8)    - Curses-based GUID partition table (GPT)...
delpart (8)   - simple wrapper around the "del partition"...
fdisk (8)     - manipulate disk partition table
fixparts (8)  - MBR partition table repair utility
...
```

Esiste anche *whatis* che mostra una linea di descrizione:

```
[me@linuxbox ~]$ whatis ls
ls        (1) - list directory contents
```

GNU Project mette ancge a disposizionje *info* che mostra le informazioni in un reader come quello di *less*. Si hanno le seguenti opzioni:

Command	Action
?	Display command help
PgUp or Backspace	Display previous page
PgDn or Space	Display next page
n	Next - Display the next node
p	Previous - Display the previous node
u	Up - Display the parent node of the currently displayed node, usually a menu
Enter	Follow the hyperlink at the cursor location

Si possono mettere più comandi in sequenza con ";":

```
[me@linuxbox ~]$ cd /usr; ls; cd -
bin games include lib local sbin
/home/me
[me@linuxbox ~]$
```

Per creare alias si ha:

```
[me@linuxbox ~]$ alias foo='cd /usr; ls; cd -'
```

che può essere aggiunto al *.bashrc* o al file di config della propria shell (*.zshrc* etc...).

Per rimuovere un alias uso:

```
[me@linuxbox ~]$ unalias foo
[me@linuxbox ~]$ type foo
bash: type: foo: not found
```

Con *alias* elenco tutti i vari aliases:

```
[me@linuxbox ~] alias
alias ls='ls --color=tty'
...
```

REDIREZIONAMENTO

Esistono due tipi di output:

1. il risultato dell'esecuzione di un programma
2. un messaggio di stato o di errore indicante come procede l'esecuzione

Dato che *everything is a file* programmi come *ls* direzionano il risultato su un file chiamato *standard output (stdout) e gli errori su uno chiamato *standard error (stderr)*. Questi file non vengono salvati su disco ma vengono reindirizzatti allo schermo. Molti programmi hanno anche uno *standard input (stdin)* collegato alla tastiera. **Tutte ciò può, ovvero l'I/O, essere reindirizzato.**

Per reindirizzare lo stdout verso un file uso l'operatore ">" seguito dal nome del file:

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

abbiamo quindi un file di testo, abbastanza grande, col risultato di *ls*:

```
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me   me   167878 2018-02-01 15:07 ls-output.txt
```

Se però facciamo:

```
[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt
ls: cannot access /bin/usr: No such file or directory
```

non avremo l'output sul file in quanto *ls* non direziona gli errori su *stdout* ma su *stderr*. Nonostante non ci sia nulla da scrivere però il file viene creato, vuoto, in quanto ">" riscrive sempre il file dall'inizio. Quindi si può usare questo trucco per creare file vuoti:

```
[me@linuxbox ~]$ > empty_file.txt
```

Se invece vogliamo fare l*append* invece di riscrivere usiamo l'operatore ">>":

```
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
```

quindi se facciamo:

```
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me   me   503634 2018-02-01 15:07 ls-output.txt
```

scriviamo tre volte l'output sul file.

Se vogliamo reindirizzare tutto l'output, quindi anche lo *stderr*, possiamo dare:

```
[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt 2>&1
```

dove si reindirizza anche il *file descriptor 2*, ovvero lo *stderr*, al *file descriptor 1*, ovvero lo *stdout*, con *2>&1*.

Le versioni recenti di BASH consentono di vare lo stesso anche con:

```
[me@linuxbox ~]$ ls -l /bin/usr &> ls-output.txt
```

dove la notazione "&>" consente di reindirizzare sia sia *stdout* che *stderr*. Si ha una notazione simile per l'*append* con ">>":

```
[me@linuxbox ~]$ ls -l /bin/usr &>> ls-output.txt
```

Spesso si ha la filosofia del *silence il golden* se non vogliamo l'output di un comando. Spesso si applica nel caso di errori e messaggi di stato. Per far ciò si direziona su un file speciale chiamato */dev/null* che è un *system device* che accetta l'input e non ci fa nulla:

```
[me@linuxbox ~]$ ls -l /bin/usr 2> /dev/null
```

Vediamo ora come concatenare files. *cat* legge il contenuto di un file e lo redireziona a *stdout*. Quindi, per visualizzare semplicemente il contenuto di un file ho:

```
[me@linuxbox ~]$ cat ls-output.txt
```

Si può anche concatenare una serie di file usando cat:

```
[me@linuxbox ~]$ cat file1.txt file2.txt file3.txt > file_tot.txt
```

Ovviamente si possono usare le wildcards:

```
[me@linuxbox ~]$ cat movie.mpeg.0* > movie.mpeg
```

Senza un file *cat* aspetta un *stdin* e poi lo riporta come output, apettando poi un altro input. Con *cat* si possono creare anche file (scrivendo il contenuto subito dopo l'immissione del comando):

```
[me@linuxbox ~]$ cat lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

Con ">" possiamo anche redirezionare lo *stdin* dalla tastiera al file di input.

```
[me@linuxbox ~]$ cat < lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

Parliamo ora delle *pipelines*, che si indica con "|" (il simpbolo di pipe appunto) e permette di indirizzare lo *stdout* del primo comando nello *stdin* del secondo. Per esempio reindirizziamo l'output di *ls* in *less*:

```
[me@linuxbox ~]$ ls -l /usr/bin | less
```

Solitamente le pipe si usano per "filtrare". Un esempio è usare il comando *sort*, che appunto mette in ordine quando ricevuto , per esempio l'output di *ls* su due cartelle, è restituisce un un'unica lista ordinata:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort
```

Abbiamo poi il comando *uniq* che accetta una lista ordinata o un file e rimuove i dublicati. Viene quindi usato spesso con *sort*:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq
```

Possiamo anche vedere solo la lista dei duplicati con l'opzione "-d":

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq -d
```

Abbiamo poi il comando *wc (word count)*, che stampa, in sequenza, il numero di righe, di parle e di byte:

```
[me@linuxbox ~]$ wc ls-output.txt
 7902 64566 503634 ls-output.txt
```

Si possono anche stampare determinate informazioni col le varie opzioni, tipo "-l" per avere solo il numero di righe. È spesso usato in pipe, per esempio per vedere quante righe ordinate non ripetute ho da due *ls*:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | wc -l
2728
```

Vediamo ora il comando di filtro più importante: *grep*. Questo comando si usa in pipe e cerca un "pattern", rappresentato da stringhe o regex, per esempio:

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
```

Si hanno delle opzioni interesanti, come "-i" che rende la ricerca *case-insensitive* e "-v" che restituisce le righe che **non** rispettano il pattern.

Abbiamo poi *head* e *tail*, che stampano le prime e le ultime 10 righe di un testo. Il numero può essere cambiato con l'opzione "-n":

```
[me@linuxbox ~]$ head -n 1 ls-output.txt
total 4
-rwxr-xr-x 1 root root      32432 2007-12-05 08:58 primo
[me@linuxbox ~]$ tail -n 1 ls-output.txt
total 4
-rwxr-xr-x 1 root root      32432 2007-12-05 08:58 ultimo
```

Ovviamente si possono usare in pipe. Con *tail* e l'opzione "-f" si possono evedere i file in realtime, come per esempio i log che si aggiornano continuamente (si richiedono spesso i permessi di root).

Per direzionare una parte intermediaria della pipe su un file, permettendo però l'uso dell'output per la pipe successiva si ha *tee*:

```
[me@linuxbox ~]$ ls /usr/bin | tee ls.txt | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
```

Un comando particolare è *echo*, che stampa l'argomento (anche una wildcard con il *pathname expansion*) su *stdout*. Le possibilità variano dal testo, al tree delle directory, ai conti aritmetici etc...:

```
[me@linuxbox ~]$ echo this is a test
this is a test
[me@linuxbox ~]$ echo *
Desktop Documents ls-output.txt Music Pictures Public Templates Videos
[me@linuxbox ~]$ echo D*
Desktop Documents
[me@linuxbox ~]$ echo *s
Documents Pictures Templates Videos
[me@linuxbox ~]$ echo [[:upper:]]*
Desktop Documents Music Pictures Public Templates Videos
[me@linuxbox ~]$ echo /usr/*/share
/usr/kerberos/share /usr/local/share
[me@linuxbox ~]$ echo ~
/home/me
[me@linuxbox ~]$ echo ~foo
```

```
/home/foo
[me@linuxbox ~]$ echo $((2 + 2))
4
```

Per le operazioni si ha la sintassi *$((expression))* e si hanno le seguenti operazioni:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division (but remember, since expansion supports only integer arithmetic, results are integers).
%	Modulo, which simply means “remainder.”
**	Exponentiation

```
[me@linuxbox ~]$ echo $(($((5**2)) * 3))
75
[me@linuxbox ~]$ echo Five divided by two equals $((5/2))
Five divided by two equals 2
[me@linuxbox ~]$ echo with $((5%2)) left over.
with 1 left over.
```

Si hanno le *brace expansion*:

```
[me@linuxbox ~]$ echo Front-{A,B,C}-Back
Front-A-Back Front-B-Back Front-C-Back
[me@linuxbox ~]$ echo Number_{1..5}
Number_1 Number_2 Number_3 Number_4 Number_5
[me@linuxbox ~]$ echo {01..15}
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
[me@linuxbox ~]$ echo {001..15}
001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
[me@linuxbox ~]$ echo {Z..A}
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
[me@linuxbox ~]$ echo a{A{1,2},B{3,4}}b
aA1b aA2b aB3b aB4b
```

le brace si possono usare anche con altri comandi, tipo *mkdir*:

```
[me@linuxbox Photos]$ mkdir {2007..2009}-{01..12}
[me@linuxbox Photos]$ ls
2007-01 2007-07 2008-01 2008-07 2009-01 2009-07
2007-02 2007-08 2008-02 2008-08 2009-02 2009-08
2007-03 2007-09 2008-03 2008-09 2009-03 2009-09
2007-04 2007-10 2008-04 2008-10 2009-04 2009-10
2007-05 2007-11 2008-05 2008-11 2009-05 2009-11
2007-06 2007-12 2008-06 2008-12 2009-06 2009-12
```

Alcune informazioni vengono nominate in *variabili di sistema*, per esempio *USER* che contiene il nostro username. Si richiamano con "$" davanti:

```
[me@linuxbox ~]$ echo $USER
me
```

e si listano tutte con:

```
[me@linuxbox ~]$ printenv
```

Si può usare anche un comando come *expansion*:

```
[me@linuxbox ~]$ echo $(ls)
```

```
Desktop Documents ls-output.txt Music Pictures Public Templates
Videos
[me@linuxbox ~]$ ls -l $(which cp)
-rwxr-xr-x 1 root root 71516 2007-12-05 08:58 /bin/cp
```

Si possono ovviamente anche usare intere pipelines:

```
[me@linuxbox ~]$ file $(ls -d /usr/bin/* | grep zip)
/usr/bin/bunzip2: symbolic link to `bzip2'
...
```

Si possono usare le "quote doppie", che permettono di valutare considerando gli sapzi, e le 'quote singole', che sopprimono l'espansione:

```
[me@linuxbox ~]$ ls -l "two words.txt"
-rw-rw-r-- 1 me me 18 2016-02-20 13:03 two words.txt
[me@linuxbox ~]$ mv "two words.txt" two_words.txt
[me@linuxbox ~]$ echo "$USER $((2+2)) $(cal)"
me 4 February 2019
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
[me@linuxbox ~]$ echo this is a   test
this is a test
[me@linuxbox ~]$ echo "this is a   test"
this is a   test

[me@linuxbox ~]$ echo text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
text /home/me/ls-output.txt a b foo 4 me
[me@linuxbox ~]$ echo "text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER"
text ~/*.txt {a,b} foo 4 me
[me@linuxbox ~]$ echo 'text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER'
text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
```

Si hanno i soliti caratteri di escape, come *$*:

```
[me@linuxbox ~]$ echo "The balance for user $USER is: \$5.00"
The balance for user me is: $5.00
```

e si usano per "!", "&", etc...Per uil carattere spazio si usa "\ ", per il backslash "\". Inoltre se ne hanno alcune particlari, per newline e tab per esempio:

Escape Sequence	Meaning
\a	Bell (an alert that causes the computer to beep)
\b	Backspace
\n	Newline. On Unix-like systems, this produces a linefeed.
\r	Carriage return
\t	Tab

APPROFONDIMENTO DI BASH

BASH usa una libreria, la *readline*, per implementare l'editing della command line. Per il movimento si hanno le seguenti shortcut:

`Ctrl-a`	Move cursor to the beginning of the line.
`Ctrl-e`	Move cursor to the end of the line.
`Ctrl-f`	Move cursor forward one character; same as the right arrow key.
`Ctrl-b`	Move cursor backward one character; same as the left arrow key.
`Alt-f`	Move cursor forward one word.
`Alt-b`	Move cursor backward one word.
`Ctrl-l`	Clear the screen and move the cursor to the top-left corner. The `clear` command does the same thing.

per l'editing:

`Ctrl-d`	Delete the character at the cursor location.
`Ctrl-t`	Transpose (exchange) the character at the cursor location with the one preceding it.
`Alt-t`	Transpose the word at the cursor location with the one preceding it.
`Alt-l`	Convert the characters from the cursor location to the end of the word to lowercase.
`Alt-u`	Convert the characters from the cursor location to the end of the word to uppercase.

si hanno poi:

`Ctrl-k`	Kill text from the cursor location to the end of line.
`Ctrl-u`	Kill text from the cursor location to the beginning of the line.
`Alt-d`	Kill text from the cursor location to the end of the current word.
`Alt-Backspace`	Kill text from the cursor location to the beginning of the current word. If the cursor is at the beginning of a word, kill the previous word.
`Ctrl-y`	Yank text from the kill-ring and insert it at the cursor location.

Per l'*autocomplete* si usa TAB e anche:

`Alt-?`	Display a list of possible completions. *On most systems you can also do this by pressing the* `Tab` *key a second time, which is much easier.*
`Alt-*`	Insert all possible completions. This is useful when you want to use more than one possible match.

La history (solitamente con gli ultimi 1000 comandi) è visibile con:

```
[me@linuxbox ~]$ history
```

e il comando `!` ti restituisce l'n-simo comando.

Inoltre:

`Ctrl-p`	Move to the previous history entry. This is the same action as the up arrow.
`Ctrl-n`	Move to the next history entry. This is the same action as the down arrow.
`Alt-<`	Move to the beginning (top) of the history list.
`Alt->`	Move to the end (bottom) of the history list, i.e., the current command line.
`Ctrl-r`	Reverse incremental search. This searches incrementally from the current command line up the history list.
`Alt-p`	Reverse search, nonincremental. With this key, type in the search string and press enter before the search is performed.
`Alt-n`	Forward search, nonincremental.
`Ctrl-o`	Execute the current item in the history list and advance to the next

e oltre al numero si hanno altre espansioni:

`!!`	Repeat the last command. It is probably easier to press up arrow and enter.
`!`*`number`*	Repeat history list item *number*.
`!string`	Repeat last history list item starting with string.
`!?string`	Repeat last history list item containing string.

PERMESSI

In linux un utente può essere *owner* di un file e l'utente di root è il proprietario anche dei file di sistema. Gli user possono appartenere ad un gruppo che permette a più utenti di avere accesso a determinati file. Con *id* si possono vedere le informazioni riguardo un user, che cambiano in abse alla distro:

```
[me@linuxbox ~]$ id
uid=1000(me) gid=1000(me)
groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(v
ideo),46(plugdev),108(lpadmin),114(admin),1000(me)
```

Torniamo all'output di *ls -l*:

```
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 me me 0 2016-03-06 14:52 foo.txt
```

Le prime 10 righe sono gli attributi del file e il primo carattere ne indica il tipo:

`-`	A regular file.
`d`	A directory.
`l`	A symbolic link. Notice that with symbolic links, the remaining file attributes are always “rwxrwxrwx” and are dummy values. The real file attributes are those of the file the symbolic link points to.

`c`	A *character special file*. This file type refers to a device that handles data as a stream of bytes, such as a terminal or `/dev/null`.
`b`	A *block special file*. This file type refers to a device that handles data in blocks, such as a hard drive or DVD drive.

mentre gli altri 9 indicano la *file mode* (i permessi di lettura,"r", scrittura, "w" e esecuzione "x"), i primi 3 per l'owner, i secondi per il

gruppo dell'owner e gli ultimi per tutti:

Owner	Group	World
rwx	rwx	rwx

r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated, however this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is determined by directory attributes.	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a program and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered, e.g., cd *directory*.

-rwx------	A regular file that is readable, writable, and executable by the file's owner. No one else has any access.
-rw-------	A regular file that is readable and writable by the file's owner. No one else has any access.
-rw-r--r--	A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world-readable.
-rwxr-xr-x	A regular file that is readable, writable, and executable by the file's owner. The file may be read and executed by everybody else.
-rw-rw----	A regular file that is readable and writable by the file's owner and members of the file's group owner only.
lrwxrwxrwx	A symbolic link. All symbolic links have “dummy” permissions. The real permissions are kept with the actual file pointed to by the symbolic link.
drwxrwx---	A directory. The owner and the members of the owner group may enter the directory and create, rename and remove files within the directory.
drwxr-x---	A directory. The owner may enter the directory and create, rename, and delete files within the directory. Members of the owner group may enter the directory but cannot create, delete, or rename files.

Per cambiare *file mode* uso il comando *chmod*, che può essere effettuato solo dal file owner o con i privilegi di superuser. *chmod* supporta due modi per specificare i cambiamenti, la rappresentazione in base 8 o quella simbolica. Vediamo quella in base 8:

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Si usa quindi una sequenza di tre numeri, tutti e tre secondo la tabella sopra, per owner, group e world. per esempio, per dare permessi di lettura e scrittura solo a owner si usa 600:

```
[me@linuxbox ~]$ chmod 600 foo.txt
```

Si ha anche una notazione simbolica per indicare a chi cambiare i permessi:

Symbol	Meaning
u	Short for "user" but means the file or directory owner.
g	Group owner.
o	Short for "others" but means world.
a	Short for "all." This is the combination of "u", "g", and "o".

e per i permessi stessi

Notation	Meaning
u+x	Add execute permission for the owner.
u-x	Remove execute permission from the owner.
+x	Add execute permission for the owner, group, and world. This is equivalent to a+x.
o-rw	Remove the read and write permissions from anyone besides the owner and group owner.
go=rw	Set the group owner and anyone besides the owner to have read and write permission. If either the group owner or the world previously had execute permission, it is removed.
u+x,go=rx	Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.

Si ha poi il comando *umask* che rimette i permessi di default. (*parte da sistemare*). Di default si ha 0002

Ci sono dei modi per cambiare user, come sloggare e riloggare o usare *sudo* o *su*. *su* ti permette di cambiare utente mentrev *sudo* permette ad un admin di sistemare un file, */etc/sudoers* dove è indicato come gli utenti possono cambiare user.

Per passare ad utente superuser si può usare:

```
[me@linuxbox ~]$ su -
Password:
[root@linuxbox ~]#
[root@linuxbox ~]# exit
[me@linuxbox ~]$
```

ma si può anche solo eseguire un comando:

```
[me@linuxbox ~]$ su -c 'ls -l /root/*'
Password:
-rw------- 1 root root 754 2007-08-11 03:19 /root/anaconda-ks.cfg
/root/Mail:
total 0
[me@linuxbox ~]$
```

Una cosa simile la permette *sudo*, che permette di eseguire qualcosa come se si fosse un altro utente. Per esempio:

```
[me@linuxbox ~]$ sudo backup_script
Password:
System Backup Starting...
```

Per cambiare owner e gruppo si ha *chown* che può essere eseguito con privilegi di superuser. Si ha la seguente sintassi: *chown [owner] [:[group]] file...*. vediamo qualche argomento d'esempio:

`bob`	Changes the ownership of the file from its current owner to user `bob`.
`bob:users`	Changes the ownership of the file from its current owner to user `bob` and changes the file group owner to group `users`.
`:admins`	Changes the group owner to the group `admins`. The file owner is unchanged.
`bob:`	Changes the file owner from the current owner to user `bob` and changes the group owner to the login group of user `bob`.

quindi tipo: ```shell [janet@linuxbox ~]$ sudo chown tony: ~tony/myfile.txt ```
Per cambiare la password di un user si ha *passwd [user]*, se l'user non viene indicato si sta usando l'user corrente:

```
[me@linuxbox ~]$ passwd
(current) UNIX password:
New UNIX password:
```

e si effettuano dei controlli sulla qualità della password:

```
[me@linuxbox ~]$ passwd
(current) UNIX password:
New UNIX password:
BAD PASSWORD: is too similar to the old one
New UNIX password:
BAD PASSWORD: it is WAY too short
New UNIX password:
BAD PASSWORD: it is based on a dictionary word
```

Si hanno anche altri comandi come *useradd, adduser, groupadd*.

Processi

pagina 134