

Sistemi Operativi

Il Facoltà di Ingegneria - Cesena

a.a 2012/2013

docenti: Santi/Ricci

[modulo lab 1a]

INTEPRETE COMANDI

IN AMBIENTI UNIX

OUTLINE DEL MODULO

- Introduzione agli interpreti comandi
- Shell Linux
 - Processore comandi
 - Primi esempi di comandi
- File System Linux
 - Comandi di accesso/manipolazione al file system
- Accesso e Manipolazione file
- Redirezione Comandi
- Interprete e Esecuzione Comandi
- Ulteriori Comandi

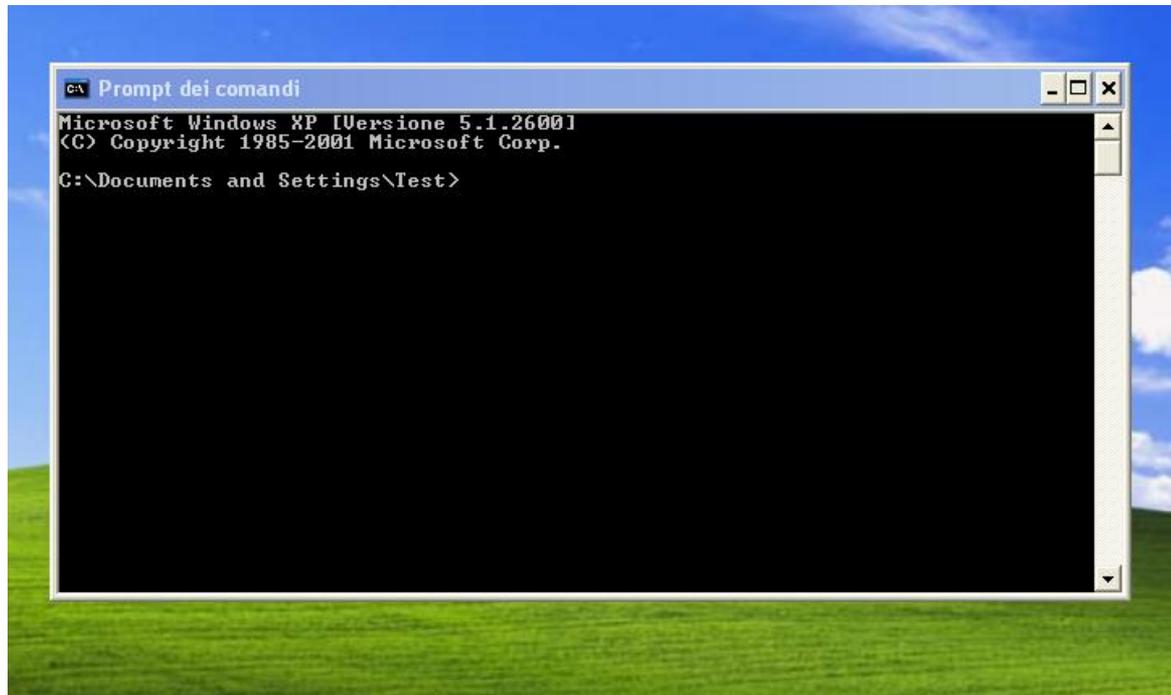
INTRODUZIONE AGLI INTERPRETI COMANDI

INTEPRETE COMANDI

- Programma che permette all'utente di interagire con il sistema mediante **comandi** impartiti in modalità testuale (non grafica), via *linea di comando*
 - nei sistemi operativi moderni *non è parte* del kernel del sistema operativo, è un'applicazione come le altre
 - i comandi possono essere implementati direttamente all'interno dell'interprete oppure richiamare l'esecuzione di programmi di sistema collocati in una specifica directory del file system
 - esempi:
 - `/usr/bin` in sistemi UNIX, `C:\WINDOWS\SYSTEM32` nei sistemi Windows
- Vari tipi di comandi
 - navigazione file system
 - interazione / manipolazione file
 - esecuzione programmi, visualizzandone l'output
 - controllo processi (programmi in esecuzione)
 - comandi per ispezionare/controllare lo stato del sistema
 - ...

SISTEMI WINDOWS

- L'interprete comandi è rappresentato dal programma **cmd.exe** in `C:\Windows\System32\`
 - da non confondere con `command.com` che rappresenta il vecchio interprete (MS)DOS eseguito su macchina virtuale IA16
 - eredita in realtà sintassi e funzionalità della maggior parte dei comandi del vecchio MSDOS



SISTEMI UNIX

- Nei sistemi UNIX esistono vari tipi di interpreti, chiamati **shell**
 - esempi
 - Bourne shell (**sh**)
 - prima shell sviluppata per UNIX (~1977)
 - /bin/sh
 - C-Shell (csh)
 - sviluppata da Bill Joy per BSD
 - /bin/csh
 - Bourne Again Shell (**bash**)
 - parte del progetto GNU, è un super set di Bourne shell
 - /bin/bash
 - Korn shell (**ksh**)
 - Bell labs
 - Z shell (**zsh**)
 - considerata la più completa, soprainsieme di tutte
 - Per una panoramica completa delle differenze:
 - <http://www.faqs.org/faqs/unix-faq/shell/shell-differences/>
 - Un utente può specificare quale shell utilizzare di default nelle proprie sessioni di lavoro

SCRIPTING

- L'interprete comandi può avere un linguaggio associato con cui è possibile scrivere **script**
 - eseguibili dall'interprete comandi
 - utili in particolare per automatizzare esecuzione di task di amministrazione

- Esempi

- sistemi Windows

- chiamati **batch** file, hanno estensione **.bat**

```
:: listmp3.bat  
  
@echo off  
dir %1\*.mp3 > %2
```

- sistemi UNIX

- uno **shell script** può avere qualsiasi estensione
 - tipicamente hanno estensione **.sh**
 - devono avere l'attributo di esecuzione settato e contenere come prima linea l'indicazione dell'interprete da usare

```
#!/bin/sh  
  
ls $1/*.mp3 > $2
```

SCHEMA DI UN PROCESSORE COMANDI

- Il comportamento astratto di un interprete comandi è descrivibile come un ciclo che accetta in ingresso (da terminale o file comandi) comandi e li esegue, fin quando ci sono comandi disponibili

```
do {
    <get command>
    <execute command>
} while (available command &&
        command != logout &&
        command != exit)
```

- Nei sistemi UNIX se il comando concerne un programma esterno, viene eseguito da una sotto-shell

SHELL UNIX

UNIX SHELL

- L'interprete comandi in UNIX prende il nome di **shell**
- **shell** come *guscio* che protegge l'utente dall'essere esposto ai dettagli del sistema
- Consente all'utente di usare il S.O. senza dover conoscere dettagli del sistema nel suo complesso

TIPI DI SHELL

- **interattive e non-interattive**

- una shell interattiva legge i comandi da un **terminale** (tty) in modo interattivo, visualizzando output
 - lo user può interagire con la shell
- una shell non-interattiva legge i comandi non da terminale, ma ad esempio da uno script file
 - è tuttavia possibile fare script che interagiscono con lo user...

- **login e non-login**

- una login shell è lanciata quando l'utente inizia la propria sessione di lavoro nel sistema
- non-login shell sono messe in esecuzione durante la sessione di lavoro, come sotto-shell

LOG IN E GESTIONE UTENTI UNIX

- Il **log in** (o log on) è la fase in cui un utente entra nel sistema, **autenticandosi**, dando inizio a una nuova sessione di lavoro
 - specificando il proprio user name e una password
- La fase di log in è necessaria in quanto UNIX è un sistema operativo **multi-utente**
 - necessità di avere opportuni meccanismi per realizzare politiche di protezione e sicurezza
- Gestione utenti in UNIX
 - gli utenti sono organizzati in gruppi.
 - ogni gruppo ha un nome simbolico (es: "admin") e un identificatore numerico univoco (**GID**, group ID).
 - ogni utente ha uno username simbolico, un identificatore numerico univoco (**UID**, user ID) e l'identificatore del gruppo a cui appartiene
 - le politiche di accesso alle risorse (tipicamente file) sono specificate a partire dallo UID e GID.
 - esiste un utente privilegiato rispetto agli altri, di nome **root** e UID 0, chiamato anche **super-user**
 - ha diritti di completo controllo e amministrazione sul sistema

LOGIN SHELL

- Nel caso in cui l'autenticazione vada a buon fine, viene eseguita una shell che prende il nome di **login shell**
- La login shell per prima cosa legge i propri file di inizializzazione (*profile files*) e configura l'ambiente come specificato
 - i file di configurazione sono di 2 tipi:
 - di sistema:
 - comuni a tutti gli utenti (es. bourne shell: **/etc/profile**)
 - relativi allo specifico utente:
 - **~/.bash_profile** (es. bash)
 - possono essere modificati dall'utente
- Poi visualizza il prompt in modo che l'utente possa inserire i comandi
 - sessione di lavoro interattiva
- La sessione termina quando l'utente impartisce il comando **logout** o **exit** oppure preme **CTRL-D**

FILE DI CONFIGURAZIONE

- L'interprete comandi può avere uno o più **file di configurazione** che vengono al login e in altri momenti della sessione di lavoro

- esempi

- sh

- /etc/profile (o ~/.profile)
 - » eseguito dopo ogni login

- Bash

- ~/.bash_profile (o bash_login)
 - » eseguito dopo ogni login
- ~/.bashrc
 - » eseguito all'avvio di ogni sub-shell
- ~/.bash_logout
 - » eseguito dopo ogni logout

```
# ~/.bash_profile example
### Variables used by bash itself#
Paths...export PATH=/usr/bin:/usr/sbin

# Control historyexport HISTFILESIZE=10export
HISTSIZE=10export HISTCONTROL=ignoreboth#
Setting session timeout
export TMOUT=3600# Promptexport PS1="[u@h]
\W [!]"### Variables that don't relate to
bash# Set variables for a warm fuzzy
environmentexport CVSROOT=~/.cvsrcexport
EDITOR=/usr/local/bin/emacsexport
PAGER=/usr/local/bin/less# end of
~/.bash_profile
```

- Utili per configurare la sessione di lavoro
 - es: settare variabili d'ambiente

AMBIENTE DI UNA SHELL 1/2

- Ogni istanza di esecuzione di una shell definisce un proprio **ambiente** dato da un insieme di **variabili**
 - usate per contenere informazioni utili per tutti i processi
 - relative all'utente e all'ambiente di esecuzione
 - recuperabili dai processi che vengono mandati in esecuzione dalla shell
- Le variabili d'ambiente sono caratterizzate da un nome simbolico e un valore di tipo stringa
 - es: nome **PATH**, valore **"/usr/bin"**
 - il nome è tipicamente in maiuscolo

AMBIENTE DI UNA SHELL 2/2

- Per definire una nuova variabile d'ambiente o per cambiarne il valore si utilizza l'operatore di assegnamento =
 - esempio: `$ MYVAR=pippo`
 - attenzione a non mettere spazi prima e dopo l'uguage
 - i numeri vengono visti come stringhe: `$ MYVAR1=123`

- L'accesso al contenuto della variabile si effettua utilizzando il prefisso \$
 - esempio: `$ echo $MYVAR`
 - `echo` è un comando che visualizza in output l'informazione passata come parametro

VARIABILI PRE-DEFINITE

- Ogni shell ha un insieme pre-definito di variabili
 - **HOME**
 - contiene il path della home directory dello user
 - **PATH**
 - contiene i path ove vengono cercati programmi eseguibili
 - **LOGNAME**
 - contiene il nome di login
 - **SHELL**
 - contiene il nome del file eseguibile dell'interprete comandi che si sta usando
 - **PS1**
 - contiene l'espressione che identifica il prompt dei comandi
 - <http://www.gnu.org/software/bash/manual/bashref.html#Printing-a-prompt>
 - **\$**
 - contiene l'identificatore numerico del processo corrente
 - **PWD**
 - contiene il percorso completo della directory corrente
 - **HOSTNAME**
 - nome dell'host corrente
 - ...

ESECUZIONE DEI COMANDI E SUB-SHELL

- In una sessione di lavoro mediante una shell interattiva (es. la login shell), la shell legge i comandi inseriti dall'utente via terminale (tty), li interpreta ed esegue
- I comandi si suddividono in *comandi interni* (**built-in**) e *comandi esterni*
 - i comandi interni sono comandi che la shell riconosce ed è in grado di eseguire direttamente
 - esempi: **echo**, **cd**
 - se un comando non è built-in, allora deve necessariamente riferirsi ad un programma disponibile nel file system
 - path corrente e tutti i path elencati nella variabile d'ambiente **PATH**
- I comandi esterni vengono eseguiti da una sotto-shell (**sub-shell**) creata dalla shell allo scopo
 - la sotto-shell viene eseguita *in un processo separato*, figlio del processo relativo alla shell
 - eredità l'ambiente di esecuzione della shell padre

ESEMPIO DI SHELL

```
sisop@sisopvm: ~ $
```

- `sisop` : nome utente
- `sisopvm` : hostname (id di rete)
- `~` : identificativo posizione filesystem (home)
- `$` : stiamo operando come utente “normale”

DESCRIZIONE DEI COMANDI

- La sintassi generale dei comandi di una shell è:

```
comando [-opzioni] [argomenti]
```

- Come separatore di programmi sulla stessa linea è possibile usare ;
- E' possibile consultare la documentazione relativamente ad un comando presente nel sistema usando
 - il comando **man** (man <NomeComando>)
 - es: \$ man ls
 - Digitando **--help** dopo il comando
 - es: \$ ls --help
- E' possibile cercare comandi per parole chiave con i comandi **apropos** e **info**
 - apropos <ParolaChiave> (es: \$ apropos file)
 - info <ParolaChiave>

NOTA

- Nelle slide che seguiranno si farà riferimento a dei sorgenti di test relativi a questo modulo
- Ogni qual volta troverete una dicitura del tipo “*directory di riferimento testX*” significa che si sta facendo riferimento alla cartella X contenuta nello zip del materiale aggiuntivo relativo a questo modulo

ESEMPIO DI COMANDO: ls

```
ls [opzioni] [directory]
```

- Il comando ls visualizza il contenuto di una directory
- La prima parola è il comando stesso (in questo caso ls).
- Dopo il comando ci sono i parametri;
 - quelli opzionali vengono racchiusi tra parentesi quadre.
- Le meta-variabili sono in corsivo
 - vanno sostituite con i parametri reali
- Le opzioni sono un caso speciale. Vengono racchiuse tra parentesi, e si possono usare in tutte le combinazioni possibili.

```
/home/sisop$ ls -la /home
drwxr-xr-x  3 sisop sisop 4096 2010-10-10 12:09 .
drwxr-xr-x 28 sisop sisop 4096 2010-10-10 12:08 ..
drwxrwxr-x 23 root  root  4096 2010-09-29 03:09 sisop
```

- Dove **-la** sono opzioni e **/home** è un argomento

OPZIONI COMANDO LS

- a** Include nell'elenco anche i file e directory il cui nome inizia per “.” file *nascosti*, per convenzione non mostrati normalmente.
- d** Elenca le proprietà delle directory specificate come parametri invece di elencare il loro contenuto.
- n** Indica proprietario e gruppo assegnato usando rispettivamente lo UID o GID numerici invece dei loro nomi.
- l** Produce un elenco esteso, una linea per ogni file, indicando da sinistra a destra: permessi, collegamenti proprietario etc.
- F** Aggiunge in coda a ciascuno dei nomi dei file elencati un carattere che ne rivela la natura:
- R** Elenca ricorsivamente anche il contenuto di sub-directory
- r** Inverte il senso di ordinamento dell'elenco.
- t** Ordina l'elenco per data e ora di ultima modifica

ESEMPI ls

```
$ ls -l
total 32
-rw-r--r--  1 sisop  staff    8 26 Set 19:44 ciao
-rw-r--r--  1 sisop  staff    6 26 Set 19:53 ciao2
-rw-r--r--  1 sisop  staff    6 26 Set 19:44 ciao3
-rw-r--r--  1 sisop  staff   59 21 Set 18:49 myc.c
```

```
$ ls -tl /*tempo estesa*/
total 32
-rw-r--r--  1 sisop      staff    6 26 Set 19:53 ciao2
-rw-r--r--  1 sisop      staff    6 26 Set 19:44 ciao3
-rw-r--r--  1 sisop      staff    8 26 Set 19:44 ciao
-rw-r--r--  1 sisop      staff   59 21 Set 18:49 myc.c
```

```
$ ls -tlr /*tempo estesa inversa*/
total 32
-rw-r--r--  1 sisop      staff   59 21 Set 18:49 myc.c
-rw-r--r--  1 sisop      staff    8 26 Set 19:44 ciao
-rw-r--r--  1 sisop      staff    6 26 Set 19:44 ciao3
-rw-r--r--  1 sisop      staff    6 26 Set 19:53 ciao2
```

META-CARATTERI

- Nella descrizione di nomi di file, la shell permette di specificare non solo nomi specifici, ma pattern o template, ovvero nomi con caratteri speciali (meta caratteri) che generalmente indicano un insieme di nomi
- I caratteri speciali sono:
 - * una qualsiasi stringa di zero o più caratteri
 - ? un qualunque carattere
 - [X,Y,..] un qualunque carattere incluso nella lista X, Y,..
 - [X-Y] un qualunque carattere da X ad Y
 - \ escape (indica alla shell di **NON** interpretare il carattere successivo a \ come speciale)
- # commento fino alla fine della linea
- Esempi (test1 come directory di riferimento):
 - ls *.h # listing di tutti i file con estensione h
 - echo * # stampa tutti i nomi della directory corrente
 - echo * # stampa asterisco
 - ls [b-d,?,a]*.[h,c]
(elenca tutti i file che iniziano con b,c,d, il terzo carattere è a, e hanno estensione .h o .c)

CARATTERI SPECIALI: ESEMPI

test2 come directory di riferimento

- I metacaratteri si possono combinare (es: `ls *0[8-9]-?`)
- In `[]` il segno `-` indica l'intervallo generato nel set di caratteri ASCII
- Il carattere di `\` annulla l'effetti del metacarattere che precede

```
/home/sisop/materiale-lab-1a$ ls -F
2011-1 2011-2 data1 data5 2011-3 2010-1 data-new data2
/home/sisop/materiale-lab-1a $ ls *0*
2011-1 2011-2 2011-3 2010-1
```

```
/home/sisop/materiale-lab-1a $ ls data*
data1 data5 data-new data2
```

```
/home/sisop/materiale-lab-1a $ ls 2011-[1-4]
2011-1 2011-2 2011-3
```

FILE SYSTEM UNIX

COMANDI RELATIVI AL FILE SYSTEM

- Esiste tutta una serie di comandi con cui manipolare il **file system**
 - Il file system è quella parte del S.O. che fornisce i meccanismi di accesso e memorizzazione delle informazioni (programmi e dati) allocate in memoria di massa.
 - Il file system realizza i concetti astratti di:
 - **file**, come unità logica di memorizzazione
 - **directory** (direttorio), come insieme di file (e direttori)
 - **partizione**, come insieme di file associato ad un particolare dispositivo fisico (o porzione di esso)
 - Le caratteristiche di file, directory e partizione sono del tutto indipendenti dalla natura e dal tipo di dispositivo utilizzato

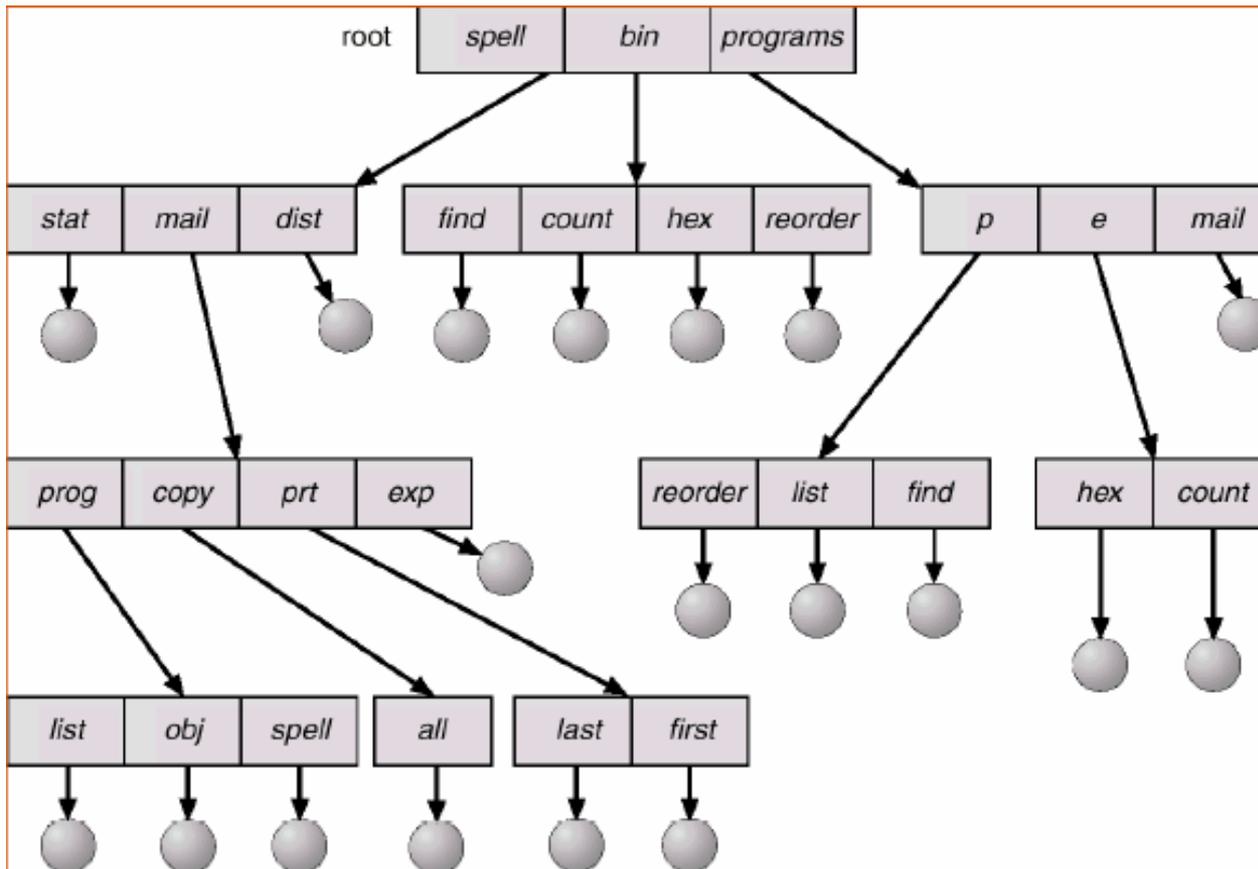
- Il file system di UNIX prende il nome di **UFS** (Unix File System)

FILE E DIRECTORY IN UNIX

- In UNIX l'astrazione di file viene utilizzata - oltre all'accezione ordinaria di contenitore di informazioni - per rappresentare in modo uniforme ogni risorsa con cui può interagire l'utente
- In particolare:
 - **dispositivi** (*device files*)
 - tutti i dispositivi di I/O (stampanti, disk driver, periferiche usb, terminali,...) sono mappati in file presenti nella directory **/dev**
 - in questo modo vi si interagisce in modo uniforme mediante operazioni quali open, read, write, close...
 - attributi speciali del descrittore di file
 - **directory** (*directory files*)
 - tengono traccia delle informazioni di una directory
 - attributo 'd'
 - file ".." = parent, "." corrente
 - **link** (*link files*)
 - rappresentano collegamenti a file locati in un'altra posizione del file system
 - attributo 'l'

DIRECTORY IN UFS

- UFS consente una organizzazione gerarchica delle directory a N livelli.
 - ogni direttorio può contenere file e altri direttori.

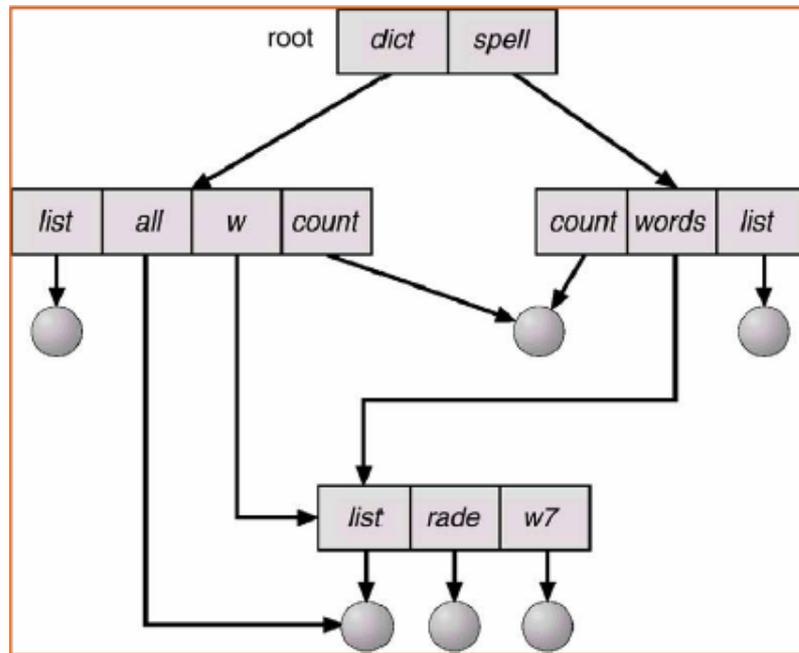


PATH NAME

- I file all'interno di strutture ad alberi sono riferiti mediante nomi simbolici detti **pathname**
- Un pathname **assoluto** riferisce il file specificando il percorso (insieme delle directory) a partire dalla radice e il nome del file
 - Ex: `/usr/asanti/docs/article.pdf`
 - percorso a partire dalla root del file system ("*/*")
- Un pathname **relativo** riferisce il file specificando il percorso (insieme delle directory) a partire dalla directory corrente e il nome del file
 - Ex: `docs/article.pdf`
 - supponendo che la directory corrente sia `/usr/asanti`

LINK FILE

- In realtà le directory in UNIX hanno una struttura a grafo, con nodi (directory) che possono condividere figli (file o directory).



- In questo modo è possibile avere la condivisione di file, riferiti con nomi (pathname) diversi
 - UNIX link file

DIRECTORY CORRENTE E **cd**

- Dopo la fase di login, le shell Linux, introducono l'utente in una zona del filesystem detta directory di "lavoro" o directory "corrente". Ad es:
`/home/sisop$;`
- Se non conoscete il vostro path assoluto potete digitare il comando **pwd (print working directory)** , che vi mostrerà il path assoluto della vostra directory corrente

```
sisop$ pwd
/home/sisop
```

- Possiamo cambiare la directory corrente usando il comando **cd [directory]**

```
sisop$ cd /home/sisop
/home$ ls
sisop/
```

IL COMANDO cd

- cd permette di usare percorsi sia assoluti sia relativi

```
/home$ cd /usr
/usr$ cd local/bin
/usr/local/bin$
```

- Ci sono due directory usate solamente nei percorsi relativi: “.” e “..”.
 - “.” si riferisce alla directory corrente
 - “..” è la directory madre: quella che contiene la directory corrente

```
/usr/local/bin$ cd ..
/usr/local$
```

- esistono in ogni directory. Anche la directory di root ha una directory madre ed è la directory di root stessa!

I COMANDI mkdir e rmdir

- per creare nuove directory:

```
mkdir dir1 [dir2 ... dirN]
```

```
$ mkdir new0 new1 new2  
$ ls  
new0 new1 new2
```

- per rimuovere directory:

```
rmdir dir1 [dir2 ... dirN]
```

```
$ rmdir new1 new2  
$ ls  
new0
```

COPIARE FILES: IL COMANDO cp

- I principali comandi per manipolare i file in Linux sono cp, mv e rm. Rispettivamente stanno per copia, sposta e rimuovi.

```
cp [-i] origine destinazione
```

```
cp [-i] file1 file2 ... fileN dir_dest
```

- **cp** permette di copiare il file origine sul file destinatario oppure uno o più file in una determinata *directory_di_destinazione*
 - Tramite l'opzione **i** (*interactive*) viene chiesta conferma prima di sovrascrivere file esistenti

```
/home/sisop$ ls -F
old.c
/home/sisop$ cp old.c new.c
/home/sisop$ ls -F
old.c  new.c
```

```
/home/sisop$ ls -F
old.c  new.c  dir_dest/
/home/sisop$ cp old.c new.c dir_dest
/home/sisop$ cd dir_di_dest
/home/sisop/dir_di_dest$ ls -F
old.c  new.c
```

RIMUOVERE e SPOSTARE FILES: rm e mv

Il comando rm elimina file: qualsiasi file che date come parametro a rm viene cancellato (directory di riferimento test3)

```
rm [-i] file1 file2 . . . fileN
```

```
/home/sisop$ ls -F
```

```
yoda pippo dir/
```

```
/home/sisop$ rm yoda pippo pluto
```

```
rm: pluto: No such file or directory
```

```
/home/sisop$ ls -F
```

```
dir/
```

Il comando mv sposta file: copia il file ed elimina il file originale

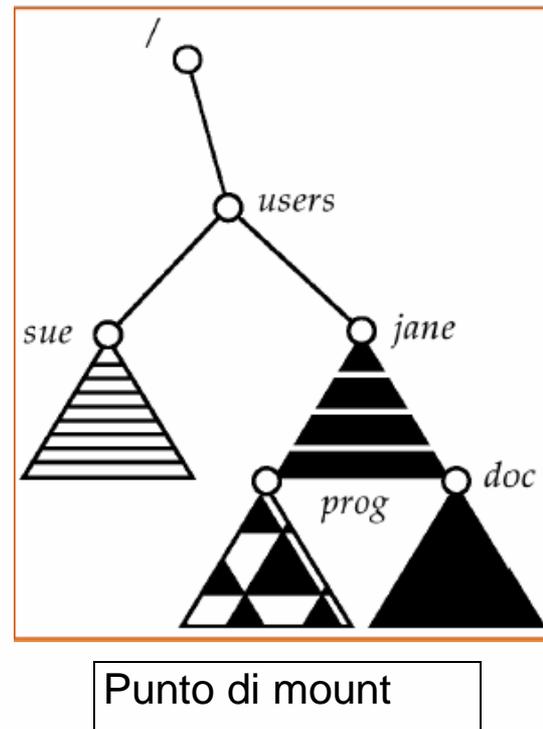
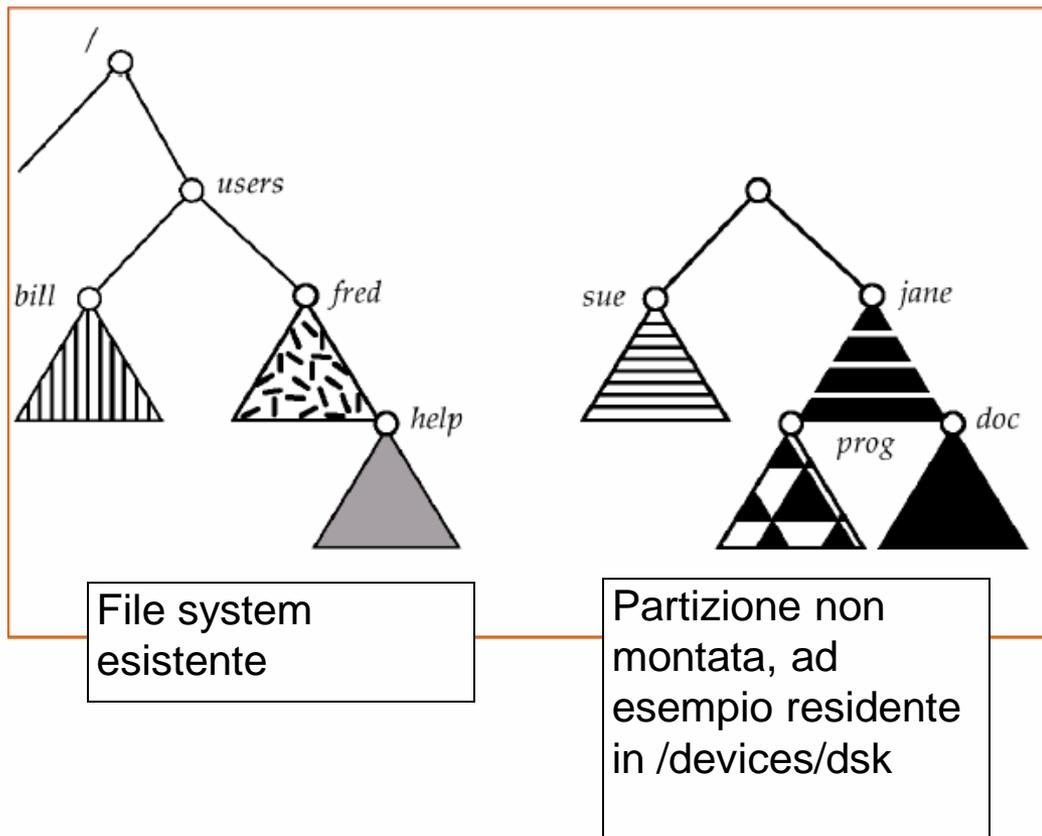
```
mv [-i] old-name new-name
```

```
mv [-i] file1 file2 . . . fileN new-dir
```

MOUNTING DI UN FILE SYSTEM

- In UNIX - e nei sistemi operativi moderni - un file system deve essere *montato* (**mounted**) prima di poter essere disponibile ai processi del sistema
 - analogamente al fatto che un file debba essere aperto prima di essere usato
- Un file system non ancora montato viene montato in un cosiddetto punto di mount (**mount point**)
 - tipicamente una directory vuota
- Un file system montato può essere poi successivamente smontato, con una operazione di unmounting

MOUNTING DI UN FILE SYSTEM



MOUNTING NEI SISTEMI UNIX

- Per fare il mounting di un file system si utilizza il comando **mount**, specificando device / partizione da montare e punto di mounting

- tra le opzioni anche il tipo di file system

- Es: mount di un floppy (device **/dev/fd0** come file system floppy con punto di mount **/mnt/floppy**) e copia di file

```
$ mount -t ext2 /dev/fd0 /mnt/floppy
```

```
$ cp ~/src/* /mnt/floppy/*
```

- Per l'unmounting esiste il comando **umount**

```
$ umount /mnt/floppy
```

- Durante il processo di boot i file system elencati nel file **/etc/fstab** sono automaticamente montati

- il file contiene una lista di linee di testo in cui si specifica il device da montare, il punto di mounting, il tipo di file system e varie opzioni

```
$ cat /etc/fstab
```

```
/dev/fd0    /mnt/floppy  auto      rw,user,noauto  0 0
```

```
/dev/hdc    /mnt/cdrom   iso9660   ro,user,noauto  0 0
```

ROOT FILE SYSTEM IN UNIX: /

- File system montato al boot, contiene utility e file fondamentali, organizzati nelle seguenti directory standard:
 - **bin**
 - contiene i comandi e utility essenziali per amministratori e utenti, necessari prima ancora che sia montato qualsiasi altro file system
 - Esempi di utility: cat, cp, ls, mkdir, sh,...
 - **boot**
 - file necessari per il boot
 - **dev**
 - contiene i dispositivi del sistema rappresentati da opportuni file
 - **etc**
 - file di configurazione del sistema. Esempi
 - fstab file: informazioni sui file system presenti
 - profile file: file inizializzazione shell sh
 - **lib**
 - librerie condivise essenziali
 - **usr**
 - directory condivisa fra tutti gli utenti, con informazioni di sola lettura

ROOT FILE SYSTEM IN UNIX: /

- (directories - segue):
 - **var**
 - Contiene file dal contenuto dinamico, come informazioni circa le sessioni aperte, cache,...
 - /var/accounts: informazioni di log sugli utenti
 - /var/cache: informazioni cached delle applicazioni
 - ...
 - **sbin**
 - contiene utility di sistema, utilizzabili esclusivamente dall'amministratore
 - **procs**
 - file system che contiene in forma di file informazioni sui processi in esecuzione
 - **media**
 - punto di mount per i media removibili
 - **mnt**
 - punto di mount per file-systems temporanei opt
 - add-ons e packages per applicazioni
 - **tmp**
 - contiene file temporanei creati dai programmi e dal sistema

/usr

- Contiene dati condivisi fra gli utenti, di sola lettura
- Directories
 - **/usr/bin**
 - contiene la maggior parte dei comandi utente
 - **/usr/include**
 - header file inclusi dai programmi C
 - **/usr/local**
 - utilizzata dagli amministratori per installare programmi in locale
 - **/usr/sbin**
 - utility non essenziali per amministratori
 - **/usr/share**
 - contiene file dati di programmi o package

/dev

- Dischi: **/dev/hd??**
 - /dev/hda, /dev/hdb, /dev/hdc
 - partizioni: /dev/hda1, /dev/hda2, ...
- Floppy: **/dev/fd?**
 - /dev/fd0 (floppy disk a:), /dev/fd1, ...
- CDROM: **/dev/cdrom**
- Dischi SCSI/SATA: **/dev/sd??**
 - /dev/sda, /dev/sdb, /dev/sdc
 - partizioni: /dev/sda1, /dev/sda2, ...
- Porte seriali: **/dev/ttyS?**
 - /dev/ttyS0 (COM1), /dev/ttyS1, ...
- Parallele: **/dev/par?**
 - /dev/par0, ...
- Stampante: **/dev/lp?**
 - /dev/lp1, ...

DEVICE FILE SPECIALI

- NULL device: **/dev/null**
 - funge da ‘buco nero’, utile per raccogliere output indesiderato
- RANDOM device: **/dev/random**
 - generatore di numeri casuali
- ZERO device file: **/dev/zero**
 - funge da generatore di byte uguali a 0
 - da usare in combinazione con il comando **dd**

ATTRIBUTI DI UN FILE IN UFS

- Per un file esistono tre tipi di utilizzatori:
 - il proprietario: owner / user (**U**)
 - il gruppo del proprietario, group (**G**)
 - tutti gli altri utenti: others (**O**)
- Ogni file ha permessi associati ad esso, che comunicano al sistema chi può accedere a quel file o modificarlo o, in caso di un programma, eseguirlo.
- Per ogni utilizzatore è possibile specificare tre modi di accesso al file:
 - lettura (r)
 - scrittura (w)
 - esecuzione (x)
- Ognuno di questi permessi può essere impostato separatamente per il proprietario, il gruppo e tutti gli altri utenti.

PERMESSI E REGOLE DI ACCESSO AI FILE

- Per ogni file nel file system sono mantenute informazioni relative a-UID e GID del proprietario e un insieme di 12 bit che specificano le regole di accesso al file:

12	11	10		9	8	7	6	5	4		3	2	1	
0	0	0		1	1	1		1	0	0		1	0	0
SUID	SGID	sticky		R	W	X		R	W	X		R	W	X
				User				Group				Others		

- I bit dal 1 al 9 (meno significativi) contengono le tre triplette di permessi per proprietario, gruppo e altri (sono memorizzate in formato ottale, tre triplette sono tre cifre da 0 a 7)
 - Es: 744 indica tutti permessi per il proprietario e solo lettura per gruppo e altri (esempio sopra)

SUID e SGID

- Il bit **SUID** (Set-User-ID): identificatore di utente effettivo.
 - si applica solo a file eseguibili
 - se vale 1 l'utente che sta eseguendo il programma assume temporaneamente (per tutta l'esecuzione del programma) l'ID del proprietario (e i relativi diritti..)
 - Ad esempio: il comando `/bin/passwd` permette di cambiare la password di un utente. Il proprietario del comando è root. Il comando modifica il file di sistema `/etc/passwd`: per fare ciò necessita di diritti di superuser. Ha dunque il SUID settato: chiunque lo esegue può accedere e modificare (in modo controlalto) il file `/etc/passwd`, vestendo i panni di superutente.

- Il bit **SGID** è come SUID, ma **a livello di gruppo**

COMANDI GESTIONE FILE SYSTEM (1/2)

- Comandi per la creazione / gestione di direttori:
 - **mkdir** <nomedir> creazione nuova directory
 - **rmdir** <nomedir> cancellazione di una directory
 - **cd** <nomedir> cambio della directory corrente
 - **pwd** visualizza in standard output il direttorio corrente
 - **ls** <nomedir> visualizzazione contenuto di una directory
- Trattamento file
 - **ln** <oldName> <newName> creazione link
 - **cp** <fileSorgente> <fileDestinazione> copia
 - **mv** <oldName> <newName> rinomina / spostamento
 - **cat** <nomefile> concatenamento / visualizzazione
 - **touch** <nomefile> creazione di un file vuoto
- Ottenere informazioni sul disco
 - **du** <nomedir>
 - visualizzazione statistiche sullo spazio occupato su disco

COMANDI GESTIONE FILE SYSTEM (2/2)

- Per cambiare permessi relativi ai file
 - **chmod** [u g o] [+ -] [rwx] <nomeFile> per cambiare permessi / regole di accesso
 - **chown** <nomeutente> <nomeFile> per specificare lo user a cui appartiene un file
 - **chgroup** <nomegruppo> <nomeFile> per specificare il gruppo di appartenenza di un file

ESEMPI CHMOD

- `chmod o-x nomefile`
il parametro `o-x` si legge “others meno execute”, cioè toglie a others il permesso di esecuzione
- `chmod u-r nomefile`
“user meno read”, cioè toglie allo user il permesso di lettura
- `chmod ugo+x nomefile`
aggiunge a tutti il permesso di esecuzione

ESEMPIO PRATICO CHMOD

- Directory di riferimento test4
- ```
$ ls -la
```

```
-rw-rw-rw- 1 sisop sisop 0 2011-10-03 23:56 accessible_by_all
```

```
----- 1 sisop sisop 24 2011-10-03 23:54 not_accessible
```

```
-rw-r--r-- 1 sisop sisop 0 2011-10-03 23:55 sisop_owner
```
- ```
$ cat accessible_by_all
```

```
This is readable by all
```
- ```
$ chmod ugo-rw accessible_by_all
```
- ```
$ cat accessible_by_all
```

```
- cat: not_accessible: Permission denied
```

ACCESSO E MANIPOLAZIONE FILE

COMANDI ACCESSO CONTENUTO FILE

- Altri comandi di sistema molto utili sono quelli che manipolano il contenuto dei file, considerati come insieme di linee, fatte da parole.
- Le parole sono sequenze di caratteri separate da spazi.
 - **cat** <file1> <file2> <fileN> concatena e stampa in stdout il contenuto dei file forniti in input
 - **more** <nomeFile> visualizza il contenuto una pagina per volta
 - **less** <nomeFile> visualizza il contenuto una pagina per volta
 - **sort** <nomeFile1> <nomeFile2> ordina alfabeticamente tutte le righe dei file forniti in input e stampa il risultato in stdout
 - **diff** <file1> <file2> mostra le righe diverse fra due file
 - **find** <directory> -name <nomeFile> -print cerca nomeFile nella directory
 - **grep** <testo> <files> cerca un determinato testo nei files specificati
 - **wc** [-lwc] <nomeFile> conta le linee (opzione l) o parole (opzione w) o i caratteri (opzione c) dallo standard input o da file
 - **uniq** elimina le eventuali linee ripetute

IL COMANDO find

```
$ find directory -name targetfile
```

- Il comando ricerca il file **targetfile** all'interno della directory fornita in input (ed eventuali sottocartelle).
 - **targetfile** può includere anche caratteri wildcard
 - Esempio:

```
$ find /home -name *.txt
```

Find: altri esempi

- Ricerca di files per tipo (-type f per file, -type d per directory), o per permessi (-perm o=r per tutti i file e le directories che possono essere lette da others), per grandezza (-size) etc.

```
- find . -type f  
cerca nella dir corrente . (e figlie) i file (f)
```

- il parametro -exec permette di eseguire comandi sui files trovati

```
- $ find . -name "*.txt" -exec wc -l '{} ' ';' 
```

conta il numero di linee in ogni file txt della dir corrente (.) e figlie

- '{}' è sostituito dal nome dei file trovati
- ';' termina la parte -exec

```
- $ find . -type f -name "*.bak" -exec rm {} \;
```

cerca e rimuove i file con estensione .bak eseguendo il comando rm tante volte quante sono i file trovati

IL COMANDO grep

- grep (General Regular Expression Print):

```
$ grep options pattern files
```

- Ricerca all'interno di **files** linee di testo che fanno match con **pattern**

- \$ grep hello *.txt

- Cerca tutte le linee di testo che contengono "hello "all'interno della directory corrente

- Altre utili opzioni

- -c (stampa il numero di linee che fanno match), -i (case insensitive), -v (stampa le linee che NON fanno match), -n (aggiunge il numero di linea a cui il match è stato trovato)

- \$ grep -vi hello *.txt

- Ricerca all'interno dei file .txt del direttorio corrente tutte le linee che **non** contengono *nessuna forma* della parola hello (e.g. Hello, HELLO, or hELIO)

REGULAR EXPRESSIONS

- I pattern specificati in grep sono un tipo particolare di pattern, noti come espressioni regolari
 - Come le espressioni aritmetiche, le espressioni regolari sono costruite tramite sub-espressioni semplici combinate tramite opportuni operatori
- L'espressione più semplice è una espressione regolare che fa match con un singolo carattere
 - Tutte le lettere e i numeri sono espressioni regolari che fanno match con loro stessi. Caratteri speciali possono essere referenziati in forma testuale utilizzando il carattere di escape backslash \
 - ^ e \$ sono anchors che fanno match rispettivamente con l'inizio e la fine di una linea di testo
 - abc\$ fa match con linee che contengono abc alla fine
 - Una lista di caratteri racchiusa tra [] fa match con ogni carattere in quella lista. Se il primo carattere è preceduto da ^, nessun carattere nella lista
 - Un rangedi caratteri racchiuso tra [] ([0-9]) fa match con ogni carattere compreso nel range. Se il primo carattere è preceduto da ^, nessun carattere nel range
 - . Fa match con qualunque carattere

ESEMPI grep

- `$ grep 1133 hello.txt`
 - visualizza tutte le linee con espressioni contenenti 1133
- `$ grep aa[c-n]aa hello.txt`
 - trova le linee contenenti aaXaa, dove X in [c-n]
- `$ grep 200[^5-9] hello.txt`
 - 200X dove X non è in [5-9]
- `$ grep 200[^a-z,^5-9] hello.txt`
 - 200X dove X non è in [5-9] né in [a-z]
- `$ grep ^..[l-z] hello.txt`
 - trova le linee in hello.txt che iniziano con una sequenza di esattamente 3 caratteri
 - i primi due arbitrari
 - l'ultimo lettera minuscola in [l-z]

IL COMANDO egrep

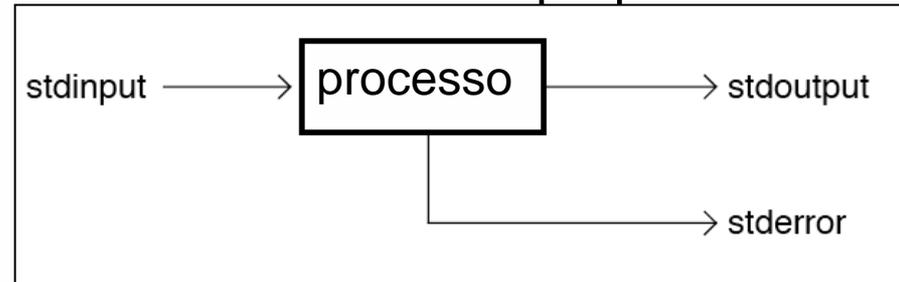
- egrep (extended grep), variant of grep, for more sophisticated reg ex
 - Here two regular expressions may be joined by the operator `|' (or)
 - the resulting regular expression matches any string matching either subexpression.
 - `$ egrep 'sisop|SistOp' file.txt`
 - `$ egrep -i '^#include[]+' program.c`
 - Brackets '(' and ')' may be used for grouping regular expressions. In addition, a regular expression may be followed by one of several repetition operators:
 - ? means the preceding item is optional (matched at most once).
 - * means the preceding item will be matched zero or more times.
 - + means the preceding item will be matched one or more times.
 - {N} means the preceding item is matched exactly N times.
 - {N,} means the preceding item is matched N or more times.
 - {N,M} means the preceding item is matched at least N times, but not more than M
 - Complex pattern example: `'(^[0-9]{1,5}[a-zA-Z]+$)|none'`
 - it would match any line that either: begins with a number up to five digits long, followed by a sequence of one or more letters or spaces, or contains the word none

REDIREZIONE COMANDI

STANDARD INPUT, OUTPUT ED ERROR

- Ad ogni programma in esecuzione sono associati tre canali (stream) da cui il programma può ricevere o inviare dati durante la propria esecuzione:

- STANDARD INPUT (**stdin**)
 - canale da cui riceve dati di input
- STANDARD OUTPUT (**stdout**)
 - canale verso cui invia dati di output
- STANDARD ERROR (**stderr**)
 - canale verso cui invia dati relativi ad errori



- I tre canali in UNIX sono sempre gestiti come file
 - stdin è per convenzione il file descriptor 0
 - stdout è per convenzione il file descriptor 1
 - stderr è per convenzione il file descriptor 2
- Lanciando un programma da shell, di default stdin, stdout, e stderr sono associati al terminale della shell

REDIREZIONE

- Ogni comando può essere rediretto su un file diverso senza cambiare il comando stesso, mediante degli operatori di redirezione:

- REDIREZIONE dell'INPUT:

```
<comando> < <inputFile>
```

– Esempio: `$ sort < data.txt`

- REDIREZIONE dell'OUTPUT: Esempio: `ls > dir.txt`

```
<comando> > <outputFile>
```

– Esempio: `$ echo hello > hello.txt`

- REDIREZIONE dell'OUTPUT IN APPEND:

```
<comando> >> <outputFile>
```

– Esempio: `$ echo hello >> hello.txt`

- La shell provvede ad aprire per ogni comando / programma lanciato **stdout**, **stdin**, **stderr**

– la redirezione viene eseguita prima del comando

ESEMPI REDIREZIONE

```
$ echo hello > hello.txt
```

```
$ echo hello2 > hello2.txt
```

```
$ echo hello3 > hello3.txt
```

```
$ ls
```

```
hello.txt  hello2.txt  hello3.txt
```

```
$ ls > elenco
```

```
$ ls
```

```
elenco hello.txt  hello2.txt  hello3.txt
```

```
$ cat elenco
```

```
...
```

```
$ ls
```

```
mail archivio
```

caso1)

```
$ cat mail > archivio /*rimane solo l'ultima lettera*/
```

caso2)

```
$ cat mail >> archivio /* viene messa in coda */
```

PIPELINE

- I comandi da shell possono essere composti, collegati tra loro in modo l'uscita di un comando divenga l'input di un comando successivo
 - si ottengono delle **pipeline** di comandi
`<comando A> | <comando B>`
- Ciò avviene grazie al meccanismo di comunicazione delle **pipe** (simbolo |):
 - l'output del comando A viene mandato in input al comando B
- I comandi sono eseguiti **in parallelo**, concorrentemente
 - il comando B consuma l'input man mano che viene prodotto come output dal comando A

ESEMPI PIPING

- Primo esempio (`lpr` invia file da stampare alla stampante)

```
$ ls > elenco
$ lpr < elenco
```

equivale a:

```
$ ls | lpr
```

- Altri esempi di piping:

```
grep pattern1 *.txt | grep -v pattern2
```

```
cat tesi.txt | lpr
```

```
sort elenco | cat -n | lpr
```

```
who | wc -l
```

```
ls -R | more
```

```
rev < file1 | rev | sort | more
```

- Nota

- una forma di piping esiste anche in DOS, tuttavia è 'finta', nel senso che i comandi non sono eseguiti in parallelo e si usano *file temporanei* come meccanismo per realizzare condivisione di informazioni.

FILTRI

- Sono denominati **filtri** quei comandi che prelevano le informazioni dallo standard input e lo inviano in standard output dopo aver operato un certo processo di trasformazione / selezione
 - si compongono via pipe
- Filtri utili che operano sulle informazioni in stdin a livello di *linee* di testo:
 - **tail -n<N>**
 - filtra in standard output le ultime N linee dello standard input
 - Esempio: `ls | tail -n 10`
 - **head -n<N>**
 - filtra in standard output le prime N linee dello standard input
 - Esempio: `ls *.h | head -n 5`
 - **more**
 - filtra in standard output lo standard input, pagina per pagina
 - Esempio: `ls | more`

FILTRI

- (continua)
 - **sort**
 - filtra in standard output le linee dello standard input in ordine
 - Esempio: `ls | sort | more`
 - **grep <string>**
 - filtra in standard output le linee dello standard input in cui compare la cerca la stringa specificata
 - Esempio: `ls -al | grep "rwxrwxrwx"`
 - **rev**
 - filtra in standard output le linee dello standard input in ordine inverso
 - Esempio: `ls | rev | rev`
 - **tee <File>**
 - copia standard input in standard output salvando le informazioni anche su file

ESEMPI FILTRI

```
$ ls | grep hel
```

```
hello1.txt
```

```
hello2.txt
```

```
hello3.txt
```

```
$ ls
```

```
elenco errors hello1.txt hello2.txt hello3.txt
```

```
$ cat elenco | tee elenco2
```

```
elenco
```

```
errors
```

```
hello1.txt
```

```
hello2.txt
```

```
hello3.txt
```

```
$ ls
```

```
elenco errors hello1.txt hello2.txt hello3.txt elenco2
```

```
$
```

INTERPRETE E ESECUZIONE COMANDI

ASPETTI AVANZATI: ESECUZIONE LISTE DI COMANDI

- Una lista di comandi è una sequenza di uno o più comandi o pipeline di comandi separati da un operatore incluso in {;, &&, ||} e opzionalmente che termina con un operatore incluso in {;, & o \n }
- Esecuzione **sequenziale**
 - comandi separati da ; vengono eseguiti sequenzialmente
- Esecuzione **in background**
 - se un comando termina con &, la shell esegue il comando in modo asincrono in una sotto-shell
 - standard input: /dev/null
 - standard output / error: ereditato dalla shell
- Esecuzione in **and** e in **or**
 - nel caso di comando1 && comando2, comando2 viene eseguito solo se comando1 termina con exit status pari a zero (no errori)
 - nel caso di comando1 || comando2, comando2 viene eseguito solo se comando1 termina con exit status diverso da zero (presenza di errori)
 - operatori associativi a sinistra

ASPETTI AVANZATI: PARSING DEI COMANDI

- Dato un nuovo comando da eseguire, prima della pura esecuzione la shell esegue una fase di parsing per gestire possibili redirezioni e per sostituire i metacaratteri
 - cerca i caratteri speciali >,<,>>,| per preparare le redirezioni / piping ingresso / uscita per i comandi che fungono da filtri
 - quindi cerca gli altri metacaratteri, operando delle sostituzioni, secondo il seguente ordine:
 - (1) **sostituzione dei comandi:**
 - il comandi contenuti fra backquote ` (ALT+96) sono eseguiti e ne viene prodotto il risultato in sostituzione della stringa in backquote:

```
$ echo `pwd` # stampa il direttorio corrente
```
 - (2) **sostituzione delle variabili e dei parametri**
 - I nomi delle variabili \$<NomeVariabile> sono espansi nei valori corrispondenti
 - (3) **sostituzione dei nomi di file**
 - I metacaratteri *,?,[] sono espansi nei nomi di file secondo un meccanismo di pattern matching

CONTROLLO DELLE ESPANSIONI

- Sono messi a disposizione degli operatori per il controllo delle espansioni:
 - **QUOTE:** la parte contenuto fra quote ' non subisce nessuna delle tre espansioni (sostituzioni)
- **DOUBLE QUOTE:** la parte contenuta fra double-quote " subisce solo le espansioni 1) e 2), non la 3)

Esempio:

```
$ echo '`pwd`' # stampa `pwd`
```

Esempio:

```
$ echo "`pwd`" # stampa la
                    directory corrente
$ echo "*" # stampa *
```

PASSATE MULTIPLE

- Ogni fase comporta **una** passata da parte della shell

- Esempio (1)

```
$ prog='*'
```

```
$ $prog
```

```
pippo.dat : execute permission denied
```

(la shell esegue le fasi 1,2 (sostituzione di prog con *), 3 (sostituzione di * con il file pippo.dat della directory corrente) e quindi prova ad eseguire pippo.dat che però non ha i diritti di esecuzione)

- Esempio (2):

```
$ cmd=`who`
```

```
$ echo $cmd
```

```
aricci console Jan 31 16:58 aricci tty1 Jan 31 22:45
```

(la shell esegue le fasi 1,2 (sostituzione di cmd con l'esecuzione risultante di who), 3, ed esegue quindi echo dell'utente corrente)

UNA SOLA ESPANSIONE PER TIPO

- Da notare che la shell esegue una sola espansione per ogni tipo: per ottenere espansioni multiple per una determinata fase occorre forzare l'ulteriore sostituzione mediante comando **eval**.

- Esempio

```
$ name1=pippo
$ name2='$name1'
$ echo $name2          # stampa $name1
$ eval echo $name2    #stampa pippo
```

- L'eval in pratica esegue il comando passato come argomento, applicando le sostituzioni:

```
$ cmd='ls | more'
$ $cmd
'ls | more': command not found
$ eval $cmd
a.out
copy1.c
copy2.c
```

CONTROLLO ESECUZIONE COMANDI

- È possibile interrompere l'esecuzione dei comandi prima della loro fine, oppure è possibile sospendere un comando, riavviandolo in seguito dal punto in cui lo si era lasciato.
- Per questo esistono speciali comandi di shell:
 - **jobs**: elenca informazioni sui job attivi o sospesi al momento. Talvolta indica anche quelli che sono stati appena terminati.
 - **ctrl-c**: Termina un programma in primo piano, (foreground); è il generico carattere di interrupt. non funziona con tutti i programmi (es: vi).
 - **ctrl-z**: sospende un programma, anche se alcuni programmi la ignorano. Una volta che è stato sospeso, il job può essere avviato in background o ucciso

ULTERIORI COMANDI

COMANDI DI CONTROLLO DEI PROCESSI

- Alcuni comandi per il controllo dei processi in esecuzione:
 - **ps**
 - elenca i processi correnti
 - **top**
 - monitora e visualizza l'elenco dei processi e thread in esecuzione
 - **kill** <sign>
 - termina o invia un segnale ad un processo corrente
 - kill <PID>
 - termina il processo dall'identificatore specificato
 - kill -s <SIGNO> <PID>
 - invia il segnale specificato al processo
 - **sleep** <NumSec>
 - sospende il processo per il numero di secondi specificati
 - **time** <Comando>
 - esegue comando (programma) cronometrandone l'esecuzione

ALTRI COMANDI UTILI

- Altri comandi frequentemente usati sono:
 - **date**
 - data e ora attuale
 - **who**
 - mostra gli utenti attualmente collegati
 - **whoami**
 - mostra le informazioni complete circa l'utente corrente
 - **man** <NomeComando>
 - recupera la documentazione relativa ad un comando
 - **apropos** <ParolaChiave>
 - trova i comandi concernenti la parola chiave

tar

- tar (tape archiver)
 - tar backs up entire directories and files onto a tape device or (more commonly) into a single disk file known as an archive.
 - An archive is a file that contains other files plus information about them, such as their filename, owner, timestamps, and access permissions.
 - tar does not perform any compression by default.
- To create a disk file tar archive, use

```
$ tar -cvf archivename filenames
```

- where archivename will usually have a .tar extension.
 - Here the c option means create, v means verbose (output filenames as they are archived), and f means file.
- To list the contents of a tar archive, use

```
$ tar -tvf archivename
```
- To restore files from a tar archive, use

```
$ tar -xvf archivename
```

gzip, compress

- compress and gzip are utilities for compressing and decompressing individual files (which may be or may not be archive files).
- To compress files, use:

```
$ gzip filename  
or  
$ compress filename
```

- In each case, filename will be deleted and replaced by a compressed file called filename.Z or filename.gz.
- To reverse the compression process, use:

```
$ gzip -d filename  
or  
$ compress -d filename
```

od

Il comando `od` effettua un dump di un file su `stdout` in differenti formati, incluso l'ottale, il decimale, virgola mobile, esadecimale e formato carattere.

od [opzioni] file

```
$ cat hello.txt
hello world
$ od -c hello.txt
0000000 h e l l o   w   o   r   l   d  \n
0000014
$ od -x hello.txt
0000000 6865 6c6c 6f20 776f 726c 640a
0000014
```

Il comando **dd**

- Il comando `dd` (*disk-dump*) è l'equivalente di `cat` specializzato per device file
 - Trasferisce un certo numero di *blocchi* di byte da un device file ad un altro
 - Sintassi:

```
dd if=<device-from> of=<device-to> bs=<block-size> count=<nblocks>
```

- Utile per vari scopi:
 - Creazione di un floppy con una certa immagine:
 - `dd if=boot.img of=/dev/fd0`
 - Generazione di un file con 1024 bytes a zero
 - `dd if=/dev/zero of=test.bin bs=1024 count=1`
 - Copia totale del disco di boot in altro disco
 - `dd if=/dev/hda of=/dev/hdd`
 - Azzeramento contenuto floppy
 - `dd if=/dev/zero of=/dev/fd0 bs=1024 count=1440`

COMANDO awk

```
awk [opzioni] file
```

“A program that you can use to select particular records in a file and perform operations upon them”

- AWK apre e chiude i files di input, legge il contenuto una riga alla volta e applica a ciascuna riga le regole (che fanno match) definite dal programmatore.

- Supponiamo di voler estrarre da un elenco di nomi e numeri di telefono tutte le righe che contengono il nome MARIO. Possiamo fare così:

```
awk '/MARIO/ { print }' elenco.txt /*oppure*/  
cat elenco.txt | awk '/MARIO/ { print }'
```

ESEMPI BASE awk

Immaginiamo di operare nella seguente directory

```
$ ls -l
total 24
-rw-r--r--  1  sisop   staff   8 26 Set 19:44 ciao
-rw-r--r--  1  sisop   staff   6 26 Set 19:44 ciao3
-rw-r--r--  1  sisop   staff  59 21 Set 18:49 myc.c
```

eseguendo un pipe con awk '{print \$1}' viene stampata la prima colonna

```
$ ls -l | awk '{print $1}'
total
-rw-r--r--
-rw-r--r--
-rw-r--r--
```

E con awk '{print \$9}' la nona colonna (se esiste)

```
$ ls -l | awk '{print $9}'
Ciao
Ciao3
myc.c
```

Esempio di pattern matching: se \$5=="6" allora stampa l'intera linea \$0

```
$ ls -l | awk '$5 == "6" {print $0}'
-rw-r--r--  1  sisop   staff   6 26 Set 19:44 ciao3
```

Acquisire diritti di Super User (sudo)

sisop@ubuntu:

- `sisop` : nome utente
- `ubuntu` : hostname (id di rete)
- `~` : identificativo posizione filesystem ((home)
- `$` : stiamo operando come utente "normale", privilegi limitati

- Spesso è necessario acquisire i diritti di super user per modificare impostazioni, eseguire comandi di sistema. Esempio:

```
sisop@ubuntu:~$ date 10011200
date: impossibile impostare la data: Funzione non permessa
```

- L'ora di sistema può essere cambiata **solo dall'utente root** (l'amministratore del computer), ed è per questo che si ricorre al comando `sudo` (Super User DO)

```
utente@ubuntu:~$ sudo date 10011200
Password:
dom ott 1 12:00:00 CEST 2006
```

- `sudo`, anteposto ad un qualsiasi comando, consente di eseguire temporaneamente comandi con i privilegi di root
 - l'effetto dura qualche minuto

sudo e su

- Se bisogna lavorare frequentemente con comandi richiedenti privilegi da super user, abbiamo 3 possibilità:

- Anteporre sudo ad ogni comando, ed eventualmente ridigitare la password

- Puo' essere tedioso

- ```
sisop@ubuntu:~$ sudo -s
root@ubuntu:~#
```

- In questo caso i diritti di su rimangono acquisiti
- `root` : nome utente
- `#` : stiamo operando come su

- Abilitare il super user:

```
sisop@ubuntu:~$ sudo passwd root
Password: /*inserite la vostra password utente*/
Enter new UNIX password: /*inserite la nuova password di root*/
Retype new UNIX password: 7*ripetete la nuova password di root*7
```

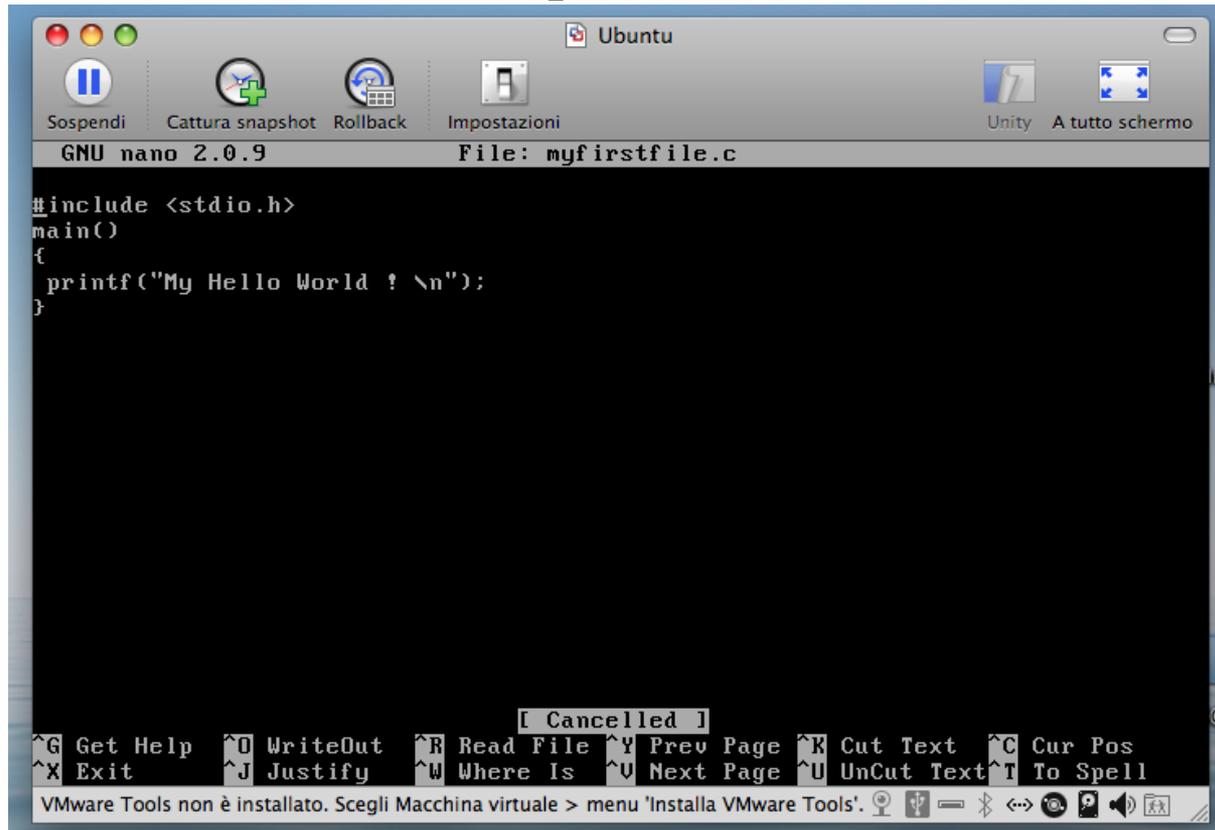
**e d'ora in avanti:**

```
sisop@ubuntu:~$ su
Password: inserite la password di root
root@ubuntu:home/sisop#
```

# EDITOR DI TESTO

- Uso di semplici programmi utili a editare file
- Pico, nano, emacs, vi, **gedit**, kate

```
$ nano myfirstfile.c
```



```
GNU nano 2.0.9 File: myfirstfile.c

#include <stdio.h>
main()
{
printf("My Hello World ! \n");
}

[Cancelled]
^G Get Help ^O WriteOut ^R Read File ^V Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^U Next Page ^U UnCut Text ^T To Spell
VMware Tools non è installato. Scegli Macchina virtuale > menu 'Installa VMware Tools'.
```

`ctrl^X` salva ed esce

# GCC

- GNU Compiler Collection
- <http://gcc.gnu.org/>
- Include strumenti e librerie per C, C++, Objective-C, Fortran, Java e Ada

- Esempi d'uso:

```
/* compila il file myfirstfile.c */
$ gcc myfirstfile.c
```

```
/* compila il file myfirstfile.c in verbose mode */
$ gcc -v myfirstfile.c
```

```
/* modo verbose, warnings, optimisation */
$ gcc -v -W -O test1.c test2.c
```

- Verificare con “ls” i file compilati

# GCC USO

Senza specificare il file eseguibile gcc crea un file `a.out`, che possiamo lanciare con:

```
$./a.out
```

Volendo creare un file eseguibile, usiamo l'opzione `-o` (minuscolo) e lo indichiamo:

```
$ gcc -o pippo.out pippo.c
```

GCC cerca file "inclusi" secondo la seguente politica :

- nella directory contenente il source
- nella directory contenente gli headers C standard (dentro all'installazione di gcc)
- nelle directory specificate in seguito all'opzione `-I`

Ad esempio supponendo il file `example.c` con un `#include "test.h"`, il comando:

```
gcc -IMyLibrary example.c
```

(dove `MyLibrary` punta in qualche directory contenente librerie create da noi) fa sì che gcc cerchi `test.h` nella directory di `example.c`, nella directory standard e in `MyLibrary`

# OPZIONI GCC

- o** (name) The name for the compiled output. The default name is '!RunImage'.
- v** (verbose) Give details of what gcc is doing. Use this to help track down problems.
- Wall** (all warnings) This makes the compiler print many helpful warnings which may indicate problems with the code. It's a good idea to use this option.
- c** Just compile and assemble the source files, don't do linking. This makes an o file from each c file passed on the command line. You can then link these files by calling gcc again with any mix of o and c files.
- O** (optimize) This attempts to make the compiled program run faster, but compiling will take longer. Use this for the final version of a program.
- O2**, -**O3** (optimize more) Compilation will take longer, and the compiled program may be slightly faster.
- Idirectory** Look for #include files in this directory.
- llibrary** Use the specified library file.

# CONFRONTO COMANDI MS-DOS UNIX

| Command's Purpose                   | MS-DOS                        | Linux                                    | Basic Linux Example                                                         |
|-------------------------------------|-------------------------------|------------------------------------------|-----------------------------------------------------------------------------|
| Copies files                        | copy                          | cp                                       | <code>cp thisfile.txt /home/thisdirectory</code>                            |
| Moves files                         | move                          | mv                                       | <code>mv thisfile.txt /home/thisdirectory</code>                            |
| Lists files                         | dir                           | ls                                       | <code>ls</code>                                                             |
| Clears screen                       | cls                           | clear                                    | <code>clear</code>                                                          |
| Closes prompt window                | exit                          | exit                                     | <code>exit</code>                                                           |
| Displays or sets date               | date                          | date                                     | <code>date</code>                                                           |
| Deletes files                       | del                           | rm                                       | <code>rm thisfile.txt</code>                                                |
| "Echoes" output on the screen       | echo                          | echo                                     | <code>echo this message</code>                                              |
| Edits files with simple text editor | edit                          | pico <a href="#">[a]</a>                 | <code>pico thisfile.txt</code>                                              |
| Compares the contents of files      | fc                            | diff                                     | <code>diff file1 file2</code>                                               |
| Finds a string of text in a file    | find                          | grep                                     | <code>grep this word or phrase thisfile.txt</code>                          |
| Formats a floppy                    | format a: (if floppy's in A:) | mke2fs (or mformat <a href="#">[b]</a> ) | <code>/sbin/mke2fs /dev/fd0 (/dev/fd0 is the Linux equivalent of A:)</code> |
| Displays command help               | <code>command /?</code>       | man <a href="#">[c]</a>                  | <code>man command</code>                                                    |
| Creates a directory                 | mkdir                         | mkdir                                    | <code>mkdir directory</code>                                                |
| Screens through a file              | more                          | less <a href="#">[d]</a>                 | <code>less thisfile.txt</code>                                              |

# CONFRONTO COMANDI MS-DOS UNIX

|                                                                    |                                 |                                 |                                                                      |
|--------------------------------------------------------------------|---------------------------------|---------------------------------|----------------------------------------------------------------------|
| Renames a file                                                     | <code>ren</code>                | <code>mv</code>                 | <code>mv <i>thisfile.txt thatfile.txt</i></code> <a href="#">[e]</a> |
| Shows your location in the file system                             | <code>chdir</code>              | <code>pwd</code>                | <code>pwd</code>                                                     |
| Changes directories with a specified path ( <i>absolute path</i> ) | <code>cd <i>pathname</i></code> | <code>cd <i>pathname</i></code> | <code>cd <i>/directory/directory</i></code>                          |
| Changes directories with a <i>relative path</i>                    | <code>cd ..</code>              | <code>cd ..</code>              | <code>cd ..</code>                                                   |
| Displays the time                                                  | <code>time</code>               | <code>date</code>               | <code>date</code>                                                    |
| Shows amount of RAM and use                                        | <code>mem</code>                | <code>free</code>               | <code>procinfo</code>                                                |

- Ogni comando supporta una ricca serie di opzioni
- Per capirne l'uso eseguire  
`man <comando>`
- **es:** `man ls`

# OUTLINE DEL MODULO

- Introduzione agli interpreti comandi
- Shell Linux
  - Processore comandi
  - Primi esempi di comandi
- File System Linux
  - Comandi di accesso/manipolazione al file system
- Accesso e Manipolazione file
- Redirezione Comandi
- Interprete e Esecuzione Comandi
- Ulteriori Comandi