Laboratorio Linux/FOSS

Alberto Cammozzo
anno accademico 2005-2006

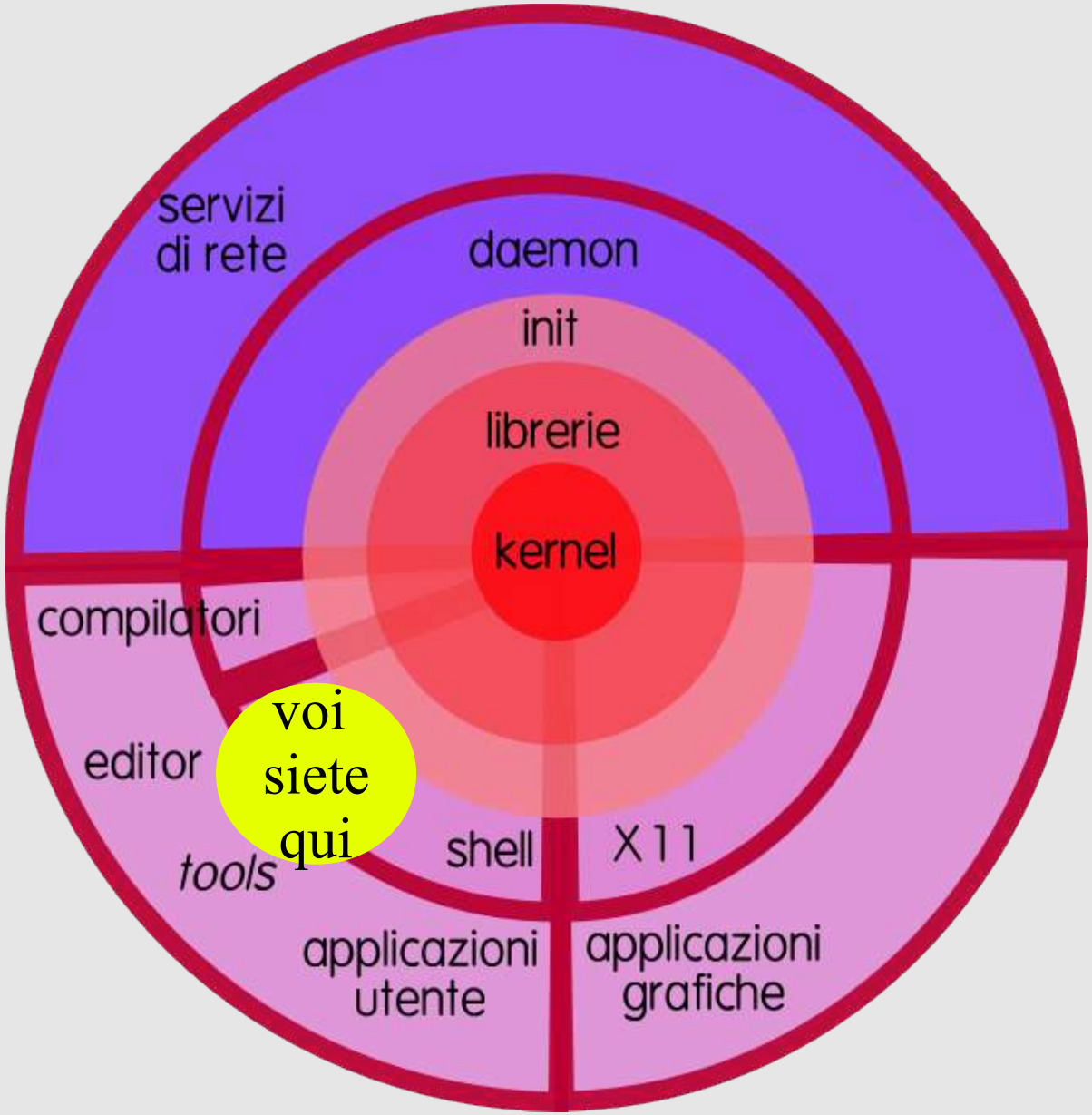
Parte Terza

GNU/Linux - 1

Parte del materiale di questo modulo è tratto dalle *slides* del Laboratorio di Sistemi Operativi di Renzo Davoli e Alberto Montresor all' Università di Bologna, http://www.cs.unibo.it/~montreso/so/lucidi-lso.shtml
rilasciate sotto Gnu Free documentation License (GNU FDL)

Indice

- la shell
- comandi relativi al *filesystem*, permessi
- comandi relativi ai *processi*
- editor, redirezione, *pipe*
- grep, sed e gli altri
- *regular expressions*
- *shell scripting* e applicazioni sui dati


servizi
di rete
daemon
init
librerie
kernel
compilatori
editor
voi
siete
qui
tools
shell
X11
applicazioni
utente
applicazioni
grafiche

La shell

- Una shell:
  - E' un programma che si frappone fra l'utente e il S.O.
  - Vi presenta il *prompt* per inserire comandi
  - E' programmabile: permette di definire *script*
    - programmi in formato testuale che raccolgono comandi
  - E' comunque un programma come tutti gli altri

Le shell

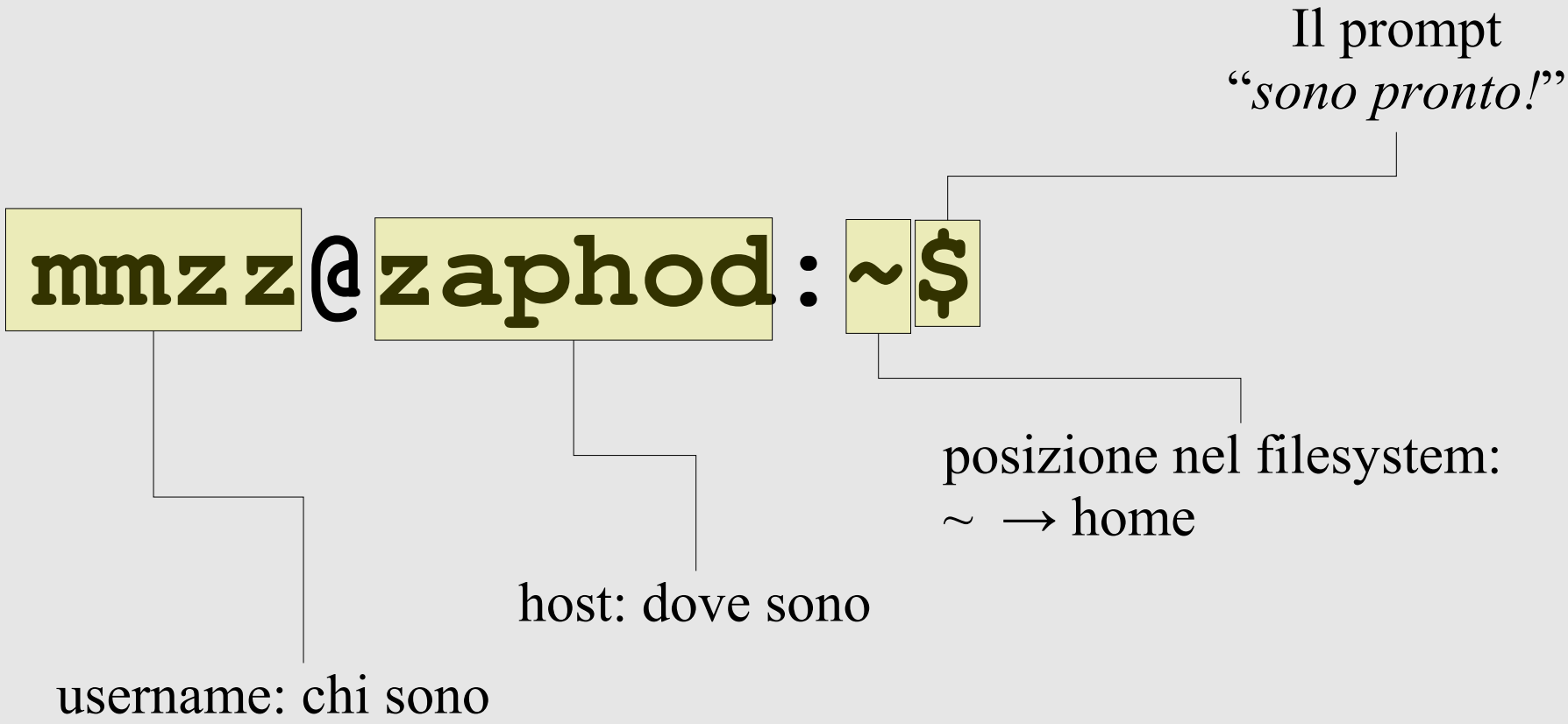
come spesso accade in Unix, c'è scelta:

- due “famiglie” di shell
  - bourne “$”
  - C-shell “>”
- bash: *bourne again* shell
  - default in linux
- tcsh: alternativa

per cambiare shell:

```
man chsh
```

Il prompt bash

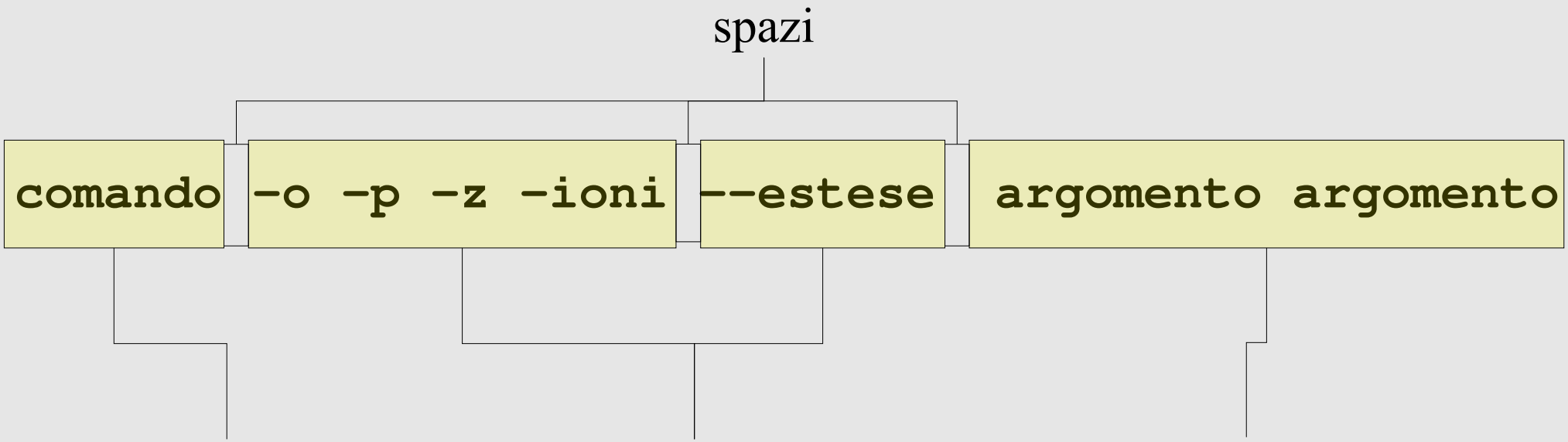


PS: può essere personalizzato.

`man bash`

`/PROMPTING`

Dare i comandi



il nome del **comando**

eventuali opzioni
modificano il comportamento del comando:
--help: aiuto
-v: verbose

eventuali argomenti
di solito file a cui applicare il comando.
A volte vanno *protetti* e messi tra virgolette.

RTFM

- `man [capitolo] comando`
- `man -k comando`
- capitoli del manuale (da `man man`):

1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within program libraries)
4 Special files (usually found in /dev)
5 File formats and conventions eg /etc/passwd
6 Games
7 Miscellaneous
8 System administration commands (usually only for root)
9 Kernel routines [Non standard]

Tasti di controllo nella shell

- **^S** sospende la visualizzazione
- **^Q** riattiva la visualizzazione
- **^C** cancella un'operazione, interrompe un programma
- **^D** end-of-line, fine input, fine sessione
- **^V** tratta il carattere di controllo seguente come se fosse un carattere normale
- **^Z** sospende l'esecuzione di un comando

BASH:

- **^A ^E**: inizio e fine riga
- **^K ^Y**: taglia e incolla
- TAB: *completion* completa il comando con gli argomenti

per personalizzare:

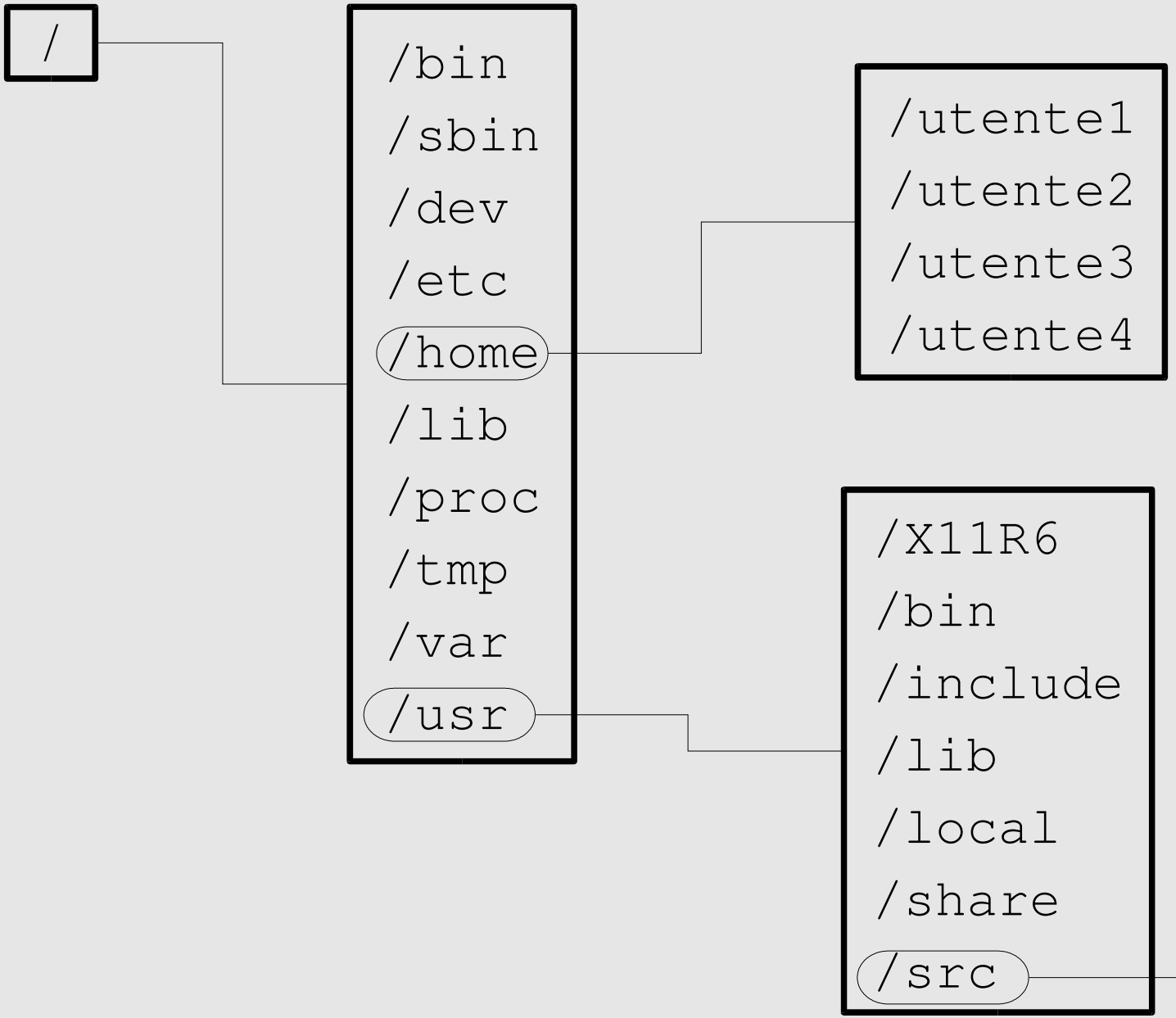
```
man bash
/READLINE
```

file system

- Il termine *file system* assume vari significati
  - è l'*insieme dei file e delle directory* che sono accessibili
  - è l'*organizzazione logica* utilizzata per gestire un insieme di file in memoria secondaria
  - anche: una *singola unità di memoria secondaria*

- I file system sono organizzati ad albero (quasi)
  - ***directory*** e ***sottodirectory*** costituiscono i nodi dell'albero
  - i ***file*** costituiscono le foglie
  - la directory ***root*** **/** costituisce la radice dell'albero

Gerarchia standard del file system



Percorso *relativo* a /usr/bin :
../src/linux

Percorso *assoluto*:
/usr/src/linux

Gerarchia standard

/bin, /usr/bin Comandi base per utenti "comuni"

/sbin, /usr/sbin Comandi per la gestione del sistema, non destinati ad utenti comuni

/dev Device file per accedere a periferiche o sistemi di memorizzazione

/etc File di configurazione del sistema

/home "Home directory" degli utenti: *casa,* $HOME, ~

/lib, /usr/lib Librerie condivise dai programmi e utili per il loro funzionamento

/tmp Directory dei file temporanei

/var Dati variabili, code di stampa

/usr Contiene gran parte dei programmi esistenti nel sistema

/proc file system *virtuale* senza reale allocazione su disco. Viene utilizzato per fornire informazioni di sistema relative in modo particolare al kernel.

navigare nel file system

`cd` ***`directory`*** Change working Directory ("go to" directory)

`pwd` Print Working Directory. *Dove sono?*

`mkdir` ***`directory`*** MaKe a directory

`rmdir` ***`directory`*** ReMove directory

`ls` ***`directory`*** LiSt directory content

- **`-a`** all files
- **`-l`** long listing
- **`-g`** group information

Attributi

- Per ottenere informazioni complete su un file:

```
$ ls -l
...
drwxrwxr-x   2 mmzz mmzz       37 2003-03-08 14:21 bin
...
-rw-r--r--  1 mmzz mmzz 1445315 2006-02-06 19:12 OS.sxi
```

Tipo e permessi del file

Il conteggio di link fisici

Lo username, groupname possessori del file

Dimensione in byte. Opzione -h: *human readable*

Data di ultima modifica

Nome del file

Owner-gruppo

- Ogni file è associato a:
  - un *utente* proprietario del file
  - un *gruppo* (i.e., insieme di utenti) con speciali diritti sul file
- Come identificare utenti e gruppi:
  - *user id* (valore intero, Unix internals); *username* (stringa)
  - *group id* (valore intero, Unix internals); *groupname* (stringa)
- Come associare un utente ad un file:
  - quando create un file, viene associato al vostro user id
  - potete modificare il proprietario tramite

```
chown userId nomefile
```

Chi sono io?

```
id
who am I
```

gruppi

- Come ottenere la lista dei gruppi a cui appartenete

  `groups` elenca i gruppi a cui appartienete

  `groups` ***`username`*** ritorna i gruppi associati a `username`

- Come associare un gruppo ad un file
  - quando create un file, viene associato al vostro gruppo corrente
  - il vostro gruppo corrente iniziale è scelto dall'amministratore
  - potete cambiare il vostro gruppo corrente (aprendo una nuova shell) tramite **`newgrp`** ***`groupname`***
  - Potete modificare il gruppo associato ad un file tramite il comando

    `chgrp` ***`groupname file file ...`***

Permessi

- Ogni file è associato a 9 flag chiamati *Permission*

User			Group			Others		
R	W	X	R	W	X	R	W	X

  - *Read*:
    - file: possibilità di leggere il contenuto
    - directory: leggere l'elenco dei file contenuti in una directory
  - *Write*:
    - file: possibilità di modificare il contenuto
    - directory: possibilità di aggiungere, rimuovere, rinominare file
  - *Execute*:
    - file: possibilità di eseguire il file (se ha senso)
    - directory: possibilità di fare **cd** nella directory o accedervi

Permessi dei file

- Quando un **processo** è in esecuzione, possiede:
  - Un user ID / group ID *reale* (usato per accounting)
  - Un user *ID* / group *ID* ***effettivo*** (usato per accesso)

- Quali permission vengono utilizzati sui **file**?
  - Se l'*user ID effettivo* del processo corrisponde a quello del *possessore* del file, si applicano le **User** permission
  - Altrimenti, se il *group ID effettivo* del processo corrisponde a quello del file, si applicano le **Group** permission
  - Altrimenti, si applicano le **Others** permission

setuid

- se il file ha il bit *setuid* attivo (rw**s**rwxrwx)
- e se è un file eseguibile
  - all'esecuzione il *processo* prenderà lo uid del proprietario del file
- analogamente per *setgid* (rwxrw**s**rwx)
  - all'esecuzione il *processo* prenderà lo uid del gruppo del file
- se usato male è un colossale pericolo di sicurezza
- utile in casi particolari

```
$ ls -l `which su`
-rwsr-xr-x  1 root root 23416 2005-05-18 08:33 /bin/su
```

cambiare i permessi

- Relativo: **`chmod [ugo][+-][rwxX] file`**
  - Esempi:

    `chmod u+x script.sh`

    Aggiunge il diritto di esecuzione per il proprietario per il file script.sh

    `chmod -R ug+rwX src/*`

    Aggiunge il diritto di scrittura, lettura per il proprietario e il gruppo per i file e contenuti in src/ , *ricorsivamente*. Inoltre aggiunge il diritto di esecuzione per le directory

    `chmod -R o-rwx $HOME`

    Toglie tutti i diritti a tutti gli utenti che non sono il proprietario e non appartengono al gruppo, ricorsivamente
  - Nota:

    Consultate **`info chmod`** per maggiori dettagli

cambiare i permessi

- Assoluto: **`chmod octal-number file`**`(s)`

User			Group			Others		
R 4	W 2	X 1	R 4	W 2	X 1	R 4	W 2	X 1

`chmod 755 file`

Assegna diritti di scrittura, lettura e esecuzione all'utente, diritti di lettura e esecuzione al gruppo e agli utenti

`chmod 644 file`

Assegna diritti di scrittura, lettura all'utente, diritti di lettura al gruppo e agli altri

gestione dei file

rm

ReMove (delete) files

cp

CoPy files

mv

MoVe (or rename) file

ln

LiNk creation (symbolic or not)

more, less

scorri un file a pagine

df [options] [directory]

mostra lo spazio libero nei dischi

```
$ df -Tm
```

du [options] [directory]

```
$ du

$ du directory

$ du -s directory

$ du -h directory
```

link

ln *file hlink*

- E' un hard-link
- Crea una entry nella directory corrente chiamata *hlink* con lo stesso inode number di *file*
- Il link number dell'inode di *file* viene incrementato di 1

ln -s *file slink*

- E' un link simbolico
- Crea un file speciale nella directory corrente chiamato *slink* che "punta" alla directory entry *file*
- Il link number dell'inode di *file* non viene incrementato

- Se cancello *file*:
  - hard-link: il link number dell'inode viene decrementato
  - link simbolico: il link diviene "stale", ovvero punta ad un file non esistente

processi

`ps` elenco dei processi

`ps ux` tutti i *miei* processi

`ps faux` tutti i processi *del sistema*

`kill` *segnale pid* : uccidi il processo

riassegnazione priorità:

`nice` *segnale pid*

`renice` *segnale pid*

`^Z` sospendi processo interattivo

`bg` manda il processo sospeso in *background*

`fg` manda il processo sospeso in interattivo (foreground)

`jobs` elenco processi in background

`comando &` manda il processo direttamente in bg

caratteri speciali

`>, >>, <`	redirezione I/O
`|`	*pipe*
`*, ?, […]`	*wildcards*
`` `comando` ``	*command substitution*
`;`	esecuzione sequenziale
`||, &&`	esecuzione condizionale
`(…)`	raggruppamento comandi
`&`	esecuzione in background
`"", ''`	*quoting*
`#`	commento
`$variabile`	espansione di variabile
`\`	carattere di *escape*
`<<`	“here documents”

processi in background

Se un comando è seguito dal metacarattere &:

- Viene creata una subshell
- il comando viene eseguito in background, in concorrenza con la shell che state utilizzando
- non prende il controllo della tastiera
- utile per eseguire attività lunghe che non necessitano di input dall'utente

```
$ find . -name indirizzi –print &

$ find $HOME -name indirizzi –print &> results.txt &
```

comandi

esterni

file eseguibile che viene cercato, caricato in memoria ed eseguito

`which` *comando*

`whereis` *comando*

interni

riconosciuti ed eseguiti dalla shell

`source`

`alias`

`cd`

`exit`

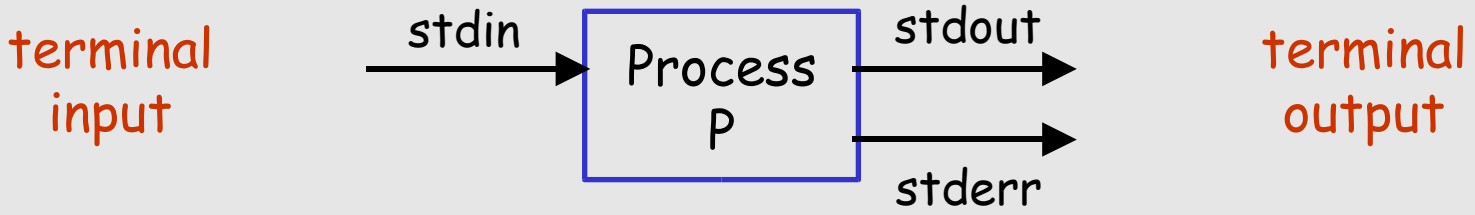
`history`

`kill`

`jobs`

redirezione

- Ogni processo è associato a tre "stream" o flussi
  - Standard output (*stdout*)
  - Standard input (*stdin*)
  - Standard error (*stderr*): errori
- Redirezione dell'I/O e pipe permettono:
  - "Slegare" questi stream dalle loro sorgenti/destinazioni abituali
  - "Legarli" ad altri sorgenti / destinazioni



redirezione

OUTPUT

Salva l'output di ls in list.txt

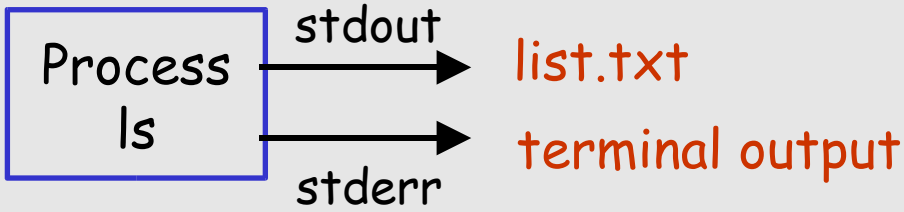
```
ls > list.txt
```

Aggiunge l'output di ls a list.txt

```
ls >> list.txt
```

Redireziona stdout e stderr del comando rm al file /dev/null

```
rm file >& /dev/null
```



INPUT

Spedisce il contenuto di list.txt a

```
mail pippo@host.net < list.txt
```

cat, less, more

output a video

```
cat filename
cat > prova
   prima riga
   seconda riga
   ^D
cat < prova
cat file1 file2 file3 > fileN
  cat file4 >> fileN
```

output paginato

```
less filename
more filename
```

ripristino video

```
reset
```

grep

- estrae le *righe* che contengono determinate stringhe
- **grep *stringa* file**

```
grep prima prova

grep riga prova
```

- **grep *pattern* file**

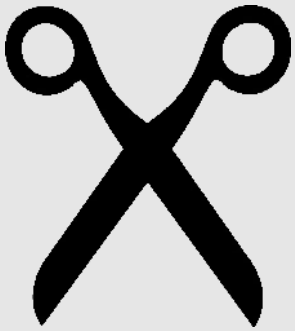
```
egrep ri.a prova

egrep r*a$ prova
```

- grep **-ivc**

pattern

- `.` singolo carattere
- `*` stringa
- `$` fine riga
- `^` inizio riga

cut



taglia determinate *colonne* dal file

per campi (opzione -f)

cut -f *colonne* **-d** *delimitatore file*

```
cut -f1 -d\  prova
```

per colonne di caratteri (opzione -c)

- **cut -c** *#col1-#col2 file*

```
cut -c 2-3,6  prova
```

altri comandi utili

`echo`	output stringhe	`echo *`
`wc`	conta righe, parole, lettere	`wc -l prova`
`sort`	ordinamento	`sort prova`
`uniq`	elimina righe doppie	
`tail`	coda di un file	`tail -2 prova`
`head`	testa di un file	`head -2 prova`
`tac`	cat rovescio	
`seq`	genera sequenza di numeri	`seq 1 10`
`split`	fa un file a pezzi	`split -l3 prova`

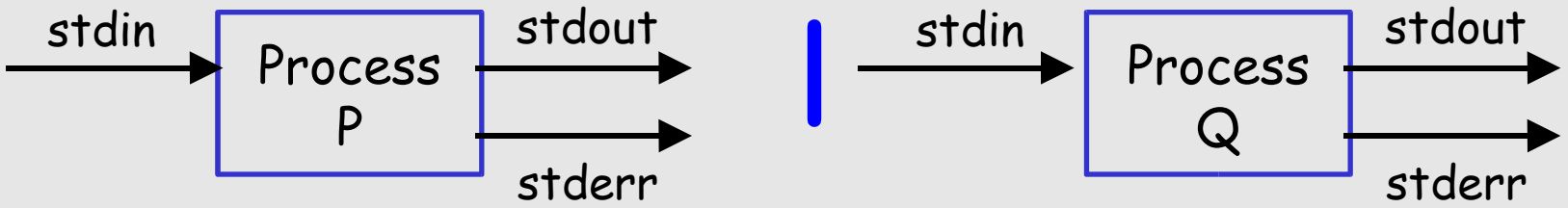
pipe

- *Pipe*, ovvero tubo o catena di montaggio:
  - *standard output* di un processo come *standard input* di un altro processo

1) `$ ls -l /etc > /tmp/conta`

2) `$ wc -l /tmp/conta`

```
$ ls -l /etc | wc -l
```



tee

tee -ia *file*

- Copia il contenuto dello standard input sul file specificato e sullo standard output; **-a** esegue l'append
- il nome **tee** deriva dai "giunti a T" usati dagli idraulici

```
$ who | tee who.capture | sort
```

processi

`sleep 10` attende 10 secondi

`sleep 10 &` direttamente in background

`sleep 10`

- `^Z` sospendi
- `bg` metti in *background*
- `jobs` elenca i lavori
- `fg %1` porta in *foreground* il primo job

`nohup echo ciao && sleep 10 &`

archivi

- comprimere

```
gzip file          gunzip file
bzip2 file         bunzip2 file
```

- archiviare

```
tar cvf archivio file file dir ...
tar xvf archivio
tar zcvf file file      tar zxvf archivio
tar jcvf  file file     tar jxvf archivio
```

- `zcat`, `zless`, `zgrep`, ...

navigazione e download

`wget [-c] url` scarica l'URL in un file

`curl url` in *stdout* la pagina http

`lynx url` browser testuali

`w3m url`

wildcards

- Il *jolly:* utilizzato per specificare file pattern
  - la stringa contenente wildcards viene sostituita con l'elenco dei file che soddisfano la condizione
  - Caratteri speciali:
    - `*` matching di qualsiasi stringa
    - `?` matching di qualsiasi carattere singolo
    - `[...]` matching di qualsiasi carattere inserito nelle parentesi
  - Esempi:

```
echo *

echo ??

echo .??*
```

command substitution

- Il comando racchiuso fra apici *rovesci*[1] viene eseguito, e il suo *standard output* viene *sostituito* al posto del comando
  - Esempi:

```
echo Data odierna: `date`
echo Utenti collegati: `who | wc -l`
tar zcvf src-`date`.tgz src/
```

[1] Sulla tastiera italiana: `alt-gr '` (apice normale)

quoting

- per inibire wildcard / command substitution / variable substitution:
  - Single quotes ' inibiscono: *wildcard, command substitution, variable substitution*

```
echo 3 * 4 = 12

echo '3*4 = 12'
```

  - Double quotes " inibiscono *wildcard* e basta

```
echo " my name is $name - date is `date` "

echo ' my name is $name - date is `date` '
```

sequenze

- non condizionali:

```
date ; pwd ; ls
```

- condizionali: && e ||
  || viene utilizzato per eseguire due comandi in sequenza, solo se il primo ha un exit code = **1** (*failure*)
  && viene utilizzato per eseguire due comandi in sequenza, solo se il primo ha un exit code = **0** (*success*)

```
who | grep mmzz &> /dev/null && echo mmzz
```

```
wget -q
http://www.stat.unipd.it/~mmzz/Papers/inesistente.
pdf || echo download fallito
```

raggruppamenti

- E' possibile raggruppare comandi racchiudendoli dentro delle parentesi
  - Vengono eseguiti in una subshell
  - Condividono gli stessi stdin, stdout e stderr

```
date ; ls ; pwd > out.txt
(date ; ls ; pwd) > out.txt
```

variabili

- *locali*: non ereditate da una shell alle subshell create da essa
  - Utilizzate all'interno di uno script
- *di ambiente (environment)*: ereditate da una shell alle subshell create da essa
  - Utilizzate per comunicazioni fra *parent* e *child* shell
- contengono dati di tipo stringa
- variabili di ambiente predefinite
  - **`$HOME`**, **`$PATH`**, **`$MAIL`**, **`$USER`**, **`$SHELL`**, **`$TERM`**, etc.
  - Per visualizzare l'elenco completo, usate il comando **`env`**

uso delle variabili

- Per accedere al contenuto di una variabile:
  - **`$name`** è la versione abbreviata di **`${name}`** (a volte è necessario)
- Per assegnare un valore ad una variabile:
  - Sintassi diversa a seconda della shell
  - Nel caso di bash

```
nome=valore          # problem with spaces
nome="valore"        # no problem with spaces
```

  - Variabili dichiarate in questo modo sono locali

uso delle variabili

- Uso di variabili locali e d'ambiente:

```
local="produzione locale"
per_esportazione="made in ASID"
echo $local $per_esportazione
export $per_esportazione
bash
echo $local $per_esportazione
exit
echo $local $per_esportazione
```

cicli

```
for i in lista
do
   fai qualcosa con $i
done
```

```
for i in `seq -w 1 10`
do
   ping -c1 pc0$i &> /dev/null && echo pc0$i
done
```

condizioni e altri cicli

```
if [ "`comando 2>/dev/null`" != "" ]
then
            echo -n " done "
fi
```

```
if list; then list; [ elif list; then
list; ] ... [ else list; ] fi
```

```
while list; do list; done
until list; do list; done
```

e molti altri... `man bash`

shell script

- raggruppamento strutturato di comandi shell che forma un programma.

```
#!/bin/bash
# (c) Autore, mail@, Licenza
# scopo del programma, commenti
variabile=istanza
comandi ...
exit valore
```

editor, come uscirne vivi

- vi
  - senza salvare: `ESC:q!`
  - salvando: `ESC:wq`
- emacs
  - senza salvare: `^X^C`
  - salvando: `^X^S ^X^C`

editor: vi

^	inizio riga
$	fine riga
i	inserisci → edit mode
o	*open* nuova riga → edit mode
n **g**	vai alla riga *n*, con `G` vai a fine file
r	rimpiazza 1 carattere → edit mode
R	rimpiazza continuato → edit mode
j	*join* congiunge riga con successiva
[n] **dd**	cancella *n* righe
dw	cancella parola
x	cancella carattere
p	incolla righe o parole cancellate

editor: emacs

- piu' che un editor un *ambiente*

ESC-□ , ^-□ : command mode

^g reset comando

^a inizio riga

^e fine riga

^d cancella carattere

^k taglia riga

^y incolla riga tagliata

^s ricerca stringa in avanti **^r** ricerca indietro

^_ undo

^SPAZIO set **mark**

^w taglia fino a **mark**

^h b descrive tutti i comandi