Modulo 1 - Sezione A
Introduzione all'uso di Linux

Laboratorio di Sistemi Operativi I
Anno Accademico 2007-2008

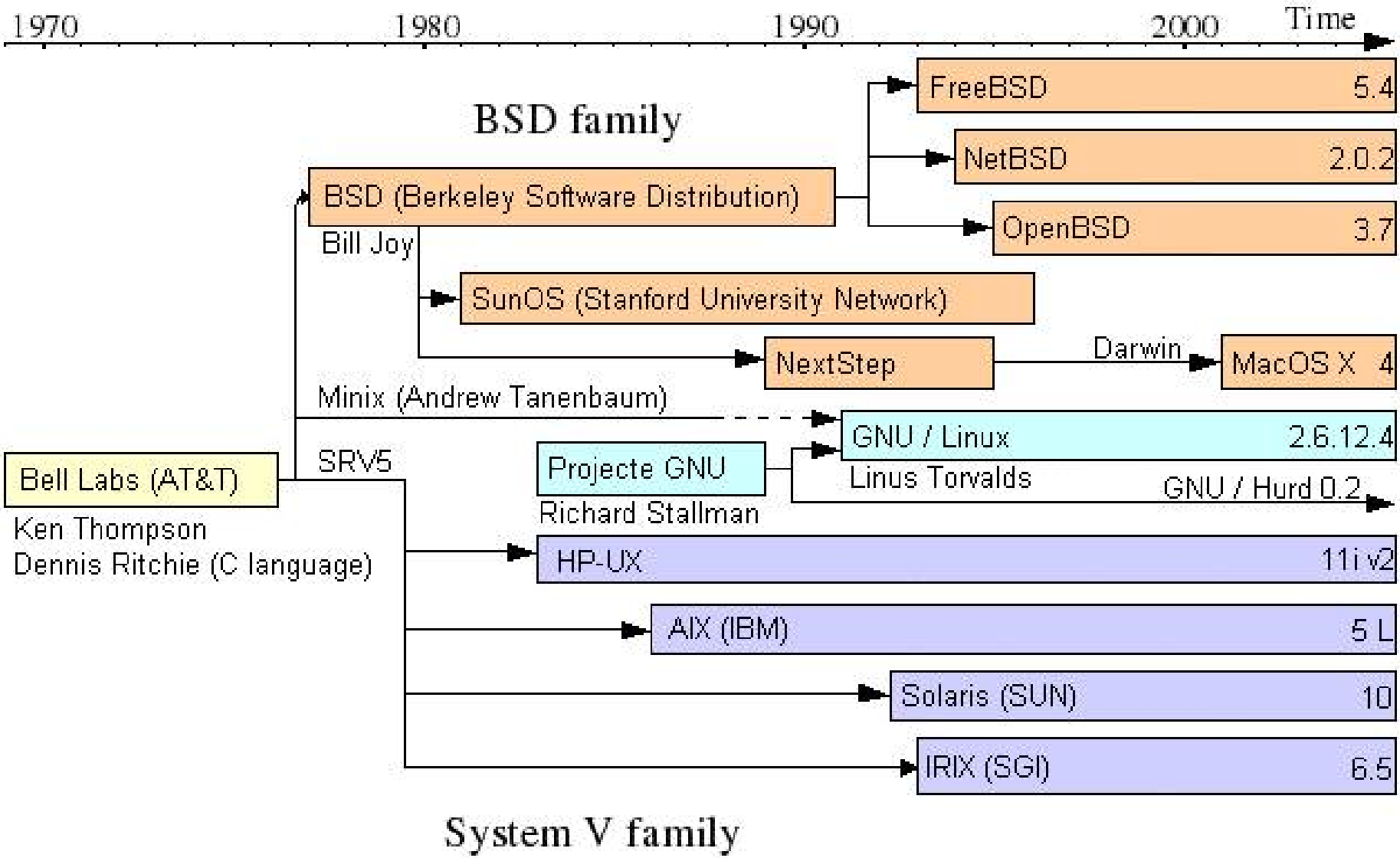
Francesco Pedullà
(Tecnologie Informatiche)

Massimo Verola
(Informatica)

Introduzione

- Linux è un sistema operativo “free” ispirato a Unix, creato originariamente da Linus Torvalds e cresciuto con il supporto di una moltitudine di sviluppatori in tutto il mondo
- Strettamente connesso alla suite GNU (compilatore, linker, debugger, etc.), il sistema può essere correttamente chiamato GNU/Linux
- Sviluppato sotto la *GNU General Public License* (GPL), il codice sorgente è gratuitamente e liberamente disponibile per chiunque
- Inizialmente sviluppato per microprocessori Intel 386, adesso è disponibile su tutte le architetture di calcolo più diffuse
- E' utilizzato in una molteplicità di sistemi: personal computers, supercomputers, sistemi *embedded* come router/firewall, telefoni cellulari e videoregistratori

Storia di UNIX

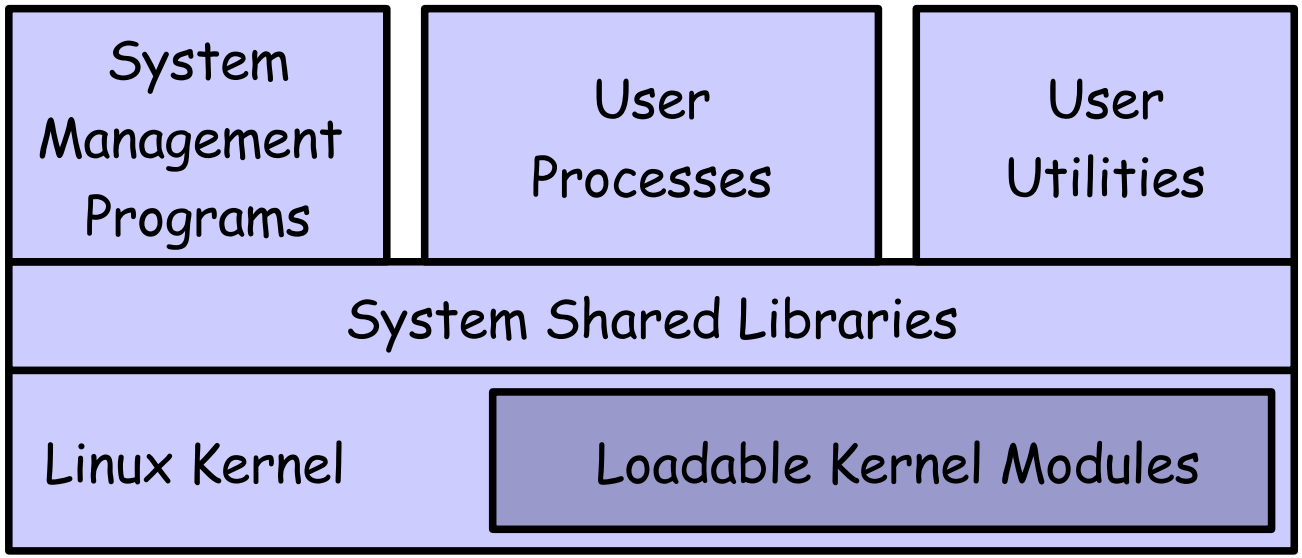


Caratteristiche di Unix/Linux

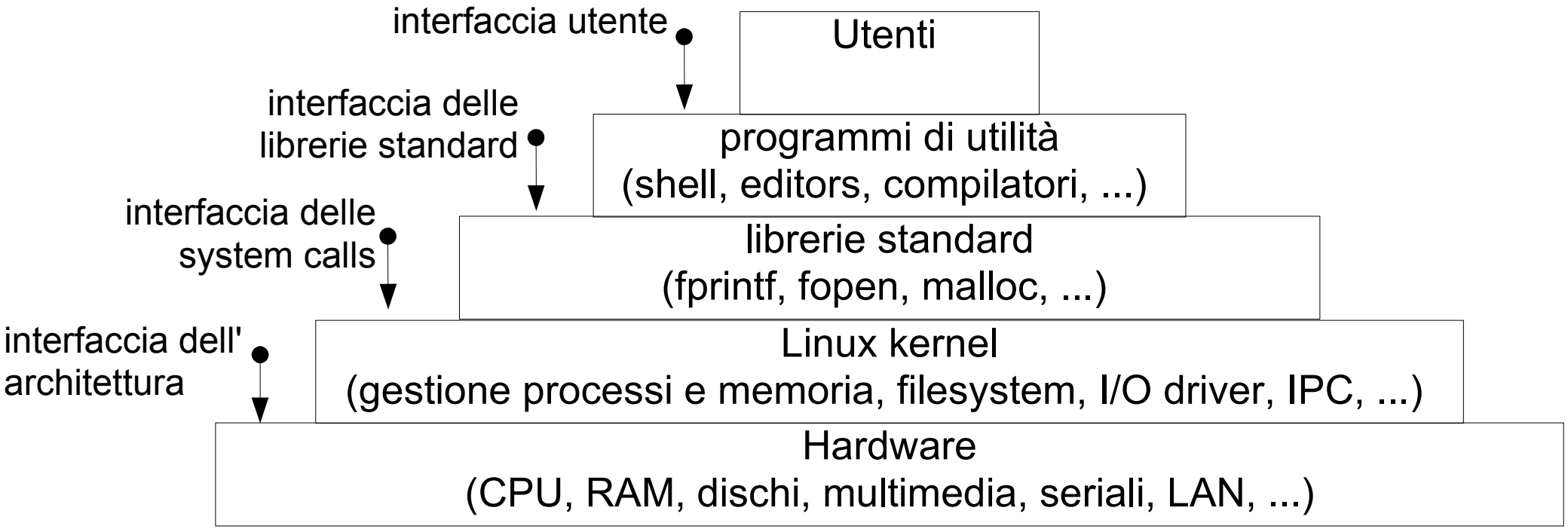
- Multiutente / multitasking
- Composizione di tool (filtri) a livello utente
- Progettato da programmatori per programmatori
- “Everything is a file”
- Kernel monolitico
- Modularità (e.g., file system)
- Scritto quasi integralmente in C (come Unix)

Principi generali di progettazione

- **Principali principi di progettazione:**
  - velocità, efficienza e standardizzazione
- **Rapporti con altri UNIX**
  - "compliant" con le specifiche POSIX
  - API fortemente basata su UNIX SVR4
  - molti tool e librerie derivanti da BSD
  - Integrazione con GNU



I layers di Linux



Il Kernel

- Il kernel supporta l'utilizzo di moduli dinamici (Loadable Kernel Module - LKM)
- i moduli sono sezioni del codice del kernel che possono essere compilati, caricati e scaricati in modo indipendente dal resto del kernel
- un modulo del kernel può implementare tipicamente un device driver, un file system, o un protocollo di networking
- l'interfaccia dei moduli permette a terze parti di scrivere o distribuire, in base ai propri termini, device driver o altro codice che non può essere distribuito sotto GPL

Login

- La procedura di login serve per:
  - autenticare l'utente
  - configurare un *environment* per l'utente
  - far partire la *shell*

```
Fedora Core release 3 (Heidelberg)
Kernel 2.6.9-1.667 on an i386

antarctic login: penguin
Password:
Last login: Thu Aug 18 17:13:26 on :0
[penguin@antarctic ~]$ echo $SHELL
/bin/bash
[penguin@antarctic ~]$
```

- Se lo *username* e la *password* sono corrette, l'utente ottiene il *prompt* (messaggio che il sistema è pronto) della shell

Logout

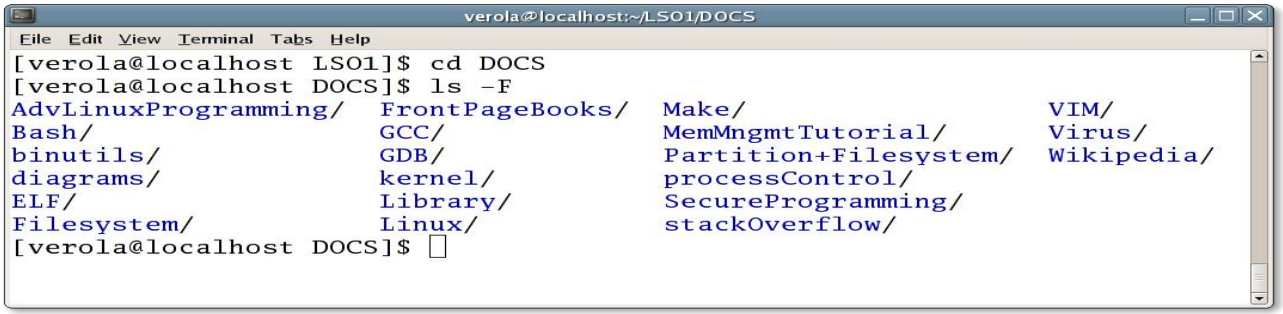
- La procedura di logout si usa per:
  - rilasciare le risorse allocate per l'utente
  - terminare i processi appartenenti all'utente
  - concludere la sessione di lavoro dell'utente

```
[penguin@antarctic ~]$ logout

Fedora Core release 3 (Heidelberg)
Kernel 2.6.9-1.667 on an i386

antarctic login:
```

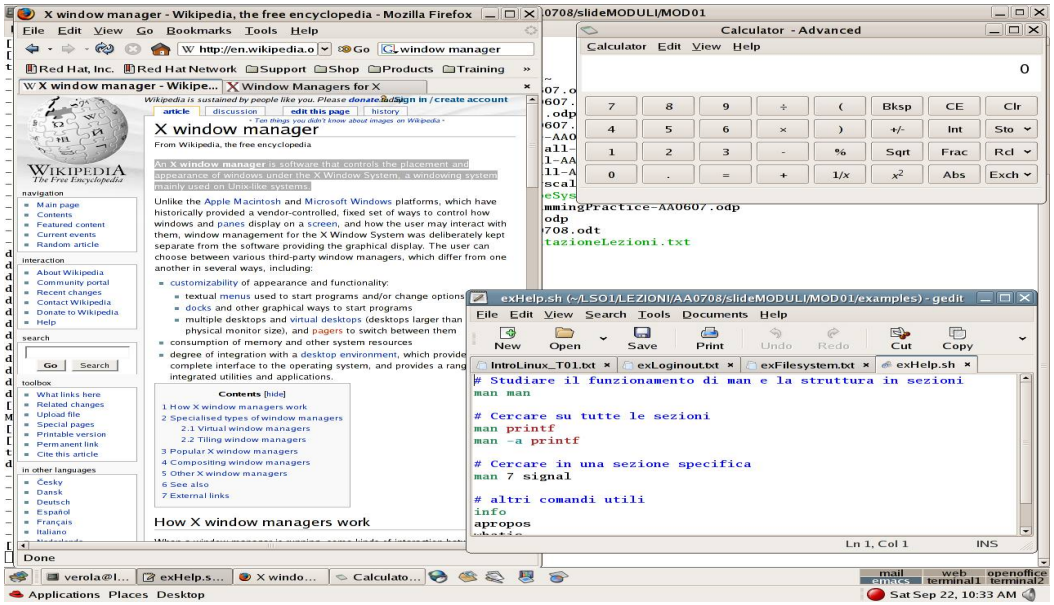
- In alternativa al comando **`logout`** si possono usare anche **`exit`** o **`Ctrl-d`**

Interfaccia con l'utente

- Due modi diversi di interazione con l'utente:
  - **interfaccia a comandi (gestita dalla shell)**



  - **interfaccia grafica (gestita dal server X Windows)**



Interfaccia a comandi

- l'utente interagisce con il sistema inviando ad esso dei comandi
- ogni comando è costituito da una sequenza di caratteri alfanumerici terminata dal tasto **<INVIO>** (**<Enter>** o **<Return>** o ↵)
- i comandi seguono una sintassi ben definita, che è analizzata da un *interprete dei comandi* (in Linux è la Shell ed anche l'utility di SO invocata dalla Shell per conto dell'utente)
- l'interprete dei comandi verifica la correttezza della stringa sulla linea di comando ed esegue le azioni richieste, eventualmente generando dei risultati o messaggi in output sul video

```
[penguin@antarctic ~]$ du -sh /home/penguin
507M    /home/penguin
[penguin@antarctic ~]$ di -sh /home/penguin
bash: di: command not found
[penguin@antarctic ~]$ du -sh (/home/verola
bash: syntax error near unexpected token `('
```

Interfaccia grafica

- si basa sull'utilizzo di icone e pannelli grafici per l'inserimento dai dati di input o la selezione di argomenti da menù predefiniti
- risponde all'esigenza di maggiore facilità d'uso e apprendimento del SO, richiede un terminale grafico e un mouse, oltre alla consueta tastiera
- l'interfaccia grafica in ambiente Linux è basata su **X Window System** (**X11** o anche solo **X**), che è un'applicazione client/server dedicata a fornire le primitive per realizzare GUI:
  - **X server:** gestisce mouse e tastiera, disegna all'interno delle finestre grafiche riservate agli *X Client* (processi che comunicano con l'X Server) i dati di output che tali client hanno inviato al server
  - **X client:** rimane in attesa dei dati di input provenienti dal server per elaborarli e invia al server i risultati dell'elaborazione
- le GUI sotto X Window System sono denotate come **X window manager,** cioe' quei software che controllano il posizionamento, le funzionalita' e lo stile grafico delle finestre
- l'utente puo' scegliere tra vari X window manager, i quali possono differire per vari fattori tra cui livello di personalizzazione ed integrazione con l'ambiente del desktop

Comandi principali

I comandi base e le utilities principali di Linux (derivati da Unix) sono:

- Creazione directory e file e spostamenti nel filesystem: **`ls cd pwd mkdir rmdir tree touch cp rm find`**
- Visualizzazione e editing di file: **`more ed vi emacs ex`**
- Elaborazione di testo: **`echo cat grep sort uniq sed awk tail tee head cut tr split printf`**
- Confronto tra file: **`comm cmp diff patch`**
- Programmi vari di utilità: **`test xargs find`**
- Amministrazione di sistema: **`chmod chown ps su who`**
- Comunicazione via rete: **`mail telnet ftp finger ssh`**
- Lancio di shell: **`sh bash csh ksh tcsh`**

Esercizi

- Dopo aver ottenuto un'utenza sul PC e una password, effettuare login, cambiare password e fare logout nei diversi modi:
  - chiudendo l'intera sessione grafica
  - chiudendo la terminal window
- Aprire una nuova terminal window, familiarizzare con le funzioni del window manager (*minimize, maximize, resize, move, close*), se non note!
- Analizzare il meccanismo dell'interfaccia testuale:
  - il prompt
  - l'inserimento di un comando
  - l'output del comando
- Lanciare il comando **`man man`** e spiegarne il significato

Filesystem - I

- Il filesystem è un modulo del SO dedicato all'archiviazione delle informazioni sulle memorie di massa
- Esistono diversi tipi di file system (ntfs, fat, ext2, ext3, jfs, ...)
- Diversi file system possono essere utilizzati contemporanemente
- Le funzioni principali sono:
  - semplificare gli accessi alle memorie di massa: per leggere e scrivere dati non si devono specificare gli indirizzi fisici mediante *head, sector, cylinder,* ma basta riferirsi a nomi di file e directory
  - offrire all'utente una modalità strutturata ed efficiente per organizzare i propri dati
  - gestire i vari dischi collegati al sistema, assegnando o rilasciando aree di memoria secondaria in base alle richieste dei programmi che utilizzano il filesystem

Filesystem - II

- Altre funzioni importanti del filesystem sono:
  - Implementare degli schemi di protezione dei dati (diritti di accesso a file e directory) e politiche di limitazione dell'uso delle spazio disco (quotas)
  - Supportare l'accesso diretto a dispositivi periferici (**/dev**)
  - Permettere l'accesso ad informazioni sul sistema (**/proc**)
  - Permettere l'accesso a dati remoti (samba, NFS)
  - Permettere l'accesso a dati di altri sistemi operativi (NTFS, FAT)

Linux file

- *"On a UNIX/Linux system, everything is a file; if something is not a file, it is a process"*
- Infatti, oltre ai file "regolari" (programmi, testo, immagini, ...), ci sono i file "speciali" che possono rappresentare dispositivi di I/O, canali di comunicazione quali pipe e socket, aree di memoria del kernel, ...
- Un *file regolare* è una sequenza di byte (non esiste una struttura di record)
- Il SO non interpreta la struttura e il contenuto di un file; ciò è delegato alle varie applicazioni
- Una *directory* è un file contenente una lista di nomi di altri file e/o di altre directory (dette subdirectory) contenuti in essa
- Il file system di Linux è organizzato gerarchicamente ad albero:
  - la directory root (indicata con "*/*") costituisce la radice dell'albero
  - le directory costituiscono i nodi dell'albero
  - i file costituiscono le foglie

Tipi di file (I)

- Un file Linux può essere dei seguenti tipi (il primo carattere è quello utilizzato dal comando `ls` per indicare il tipo di file):
  - ***file regolare:*** dati in formato testo o binario, eseguibili, programmi sorgente, immagini, ...
  - **d** ***directory:*** file che contiene una lista di altri file
  - **l** ***link simbolico:*** file che *punta* ad un altro file
  - **c** ***device a caratteri:*** dispositivo di I/O organizzato a singoli byte
  - **b** ***device a blocchi:*** dispositivo di I/O organizzato a singoli byte
  - **p** ***named pipe:*** meccanismo per l'IPC all'interno del kernel
  - **s** ***socket:*** canale per le comunicazioni su rete

Tipi di file (II)

- L'opzione **-l** del comando **ls** mostra il tipo di file, utilizzando il primo carattere di ciascuna linea di output:

```
[penguin@antarctic ~]$ ls -l
total 160
drwxr-xr-x   2 penguin birds 4096 Aug 13 23:07 Desktop
drwxrwxr-x   9 penguin birds 4096 Aug 17 23:11 LSO1
-rwxrwxr-x   1 penguin birds 6240 Aug 31 09:35 main
-rw-rw-r--   1 penguin birds  337 Aug 31 09:35 main.c
lrwxrwxrwx   1 penguin birds    6 Aug 31 00:04 prog.c -> main.c
prw-rw-r--   1 penguin birds    0 Aug 31 17:40 myfifo
```

Nomi di file (I)

- Un nome di file può essere di qualsiasi lunghezza e può utilizzare qualsiasi carattere stampabile, eccetto il *forward slash* '/'
- I nomi dei file devono essere unici all'interno di una stessa directory
- Un nome di file che inizia per '.' denota un file/directory nascosto/a (*hidden*) e non viene mostrato dal comando **`ls`** (a meno di non usare l'opzione **`-a`**)
- Un nome di file non dovrebbe mai iniziare con *hyphen* '-', in quanto molti comandi considerano una stringa che inizia per '-' come indicazione di un'opzione
- Un nome di file/directory non ha una struttura definita: eventuali estensioni (.c, .h, .txt, ...) hanno solo un valore convenzionale e non intrinseco

Nomi di file (II)

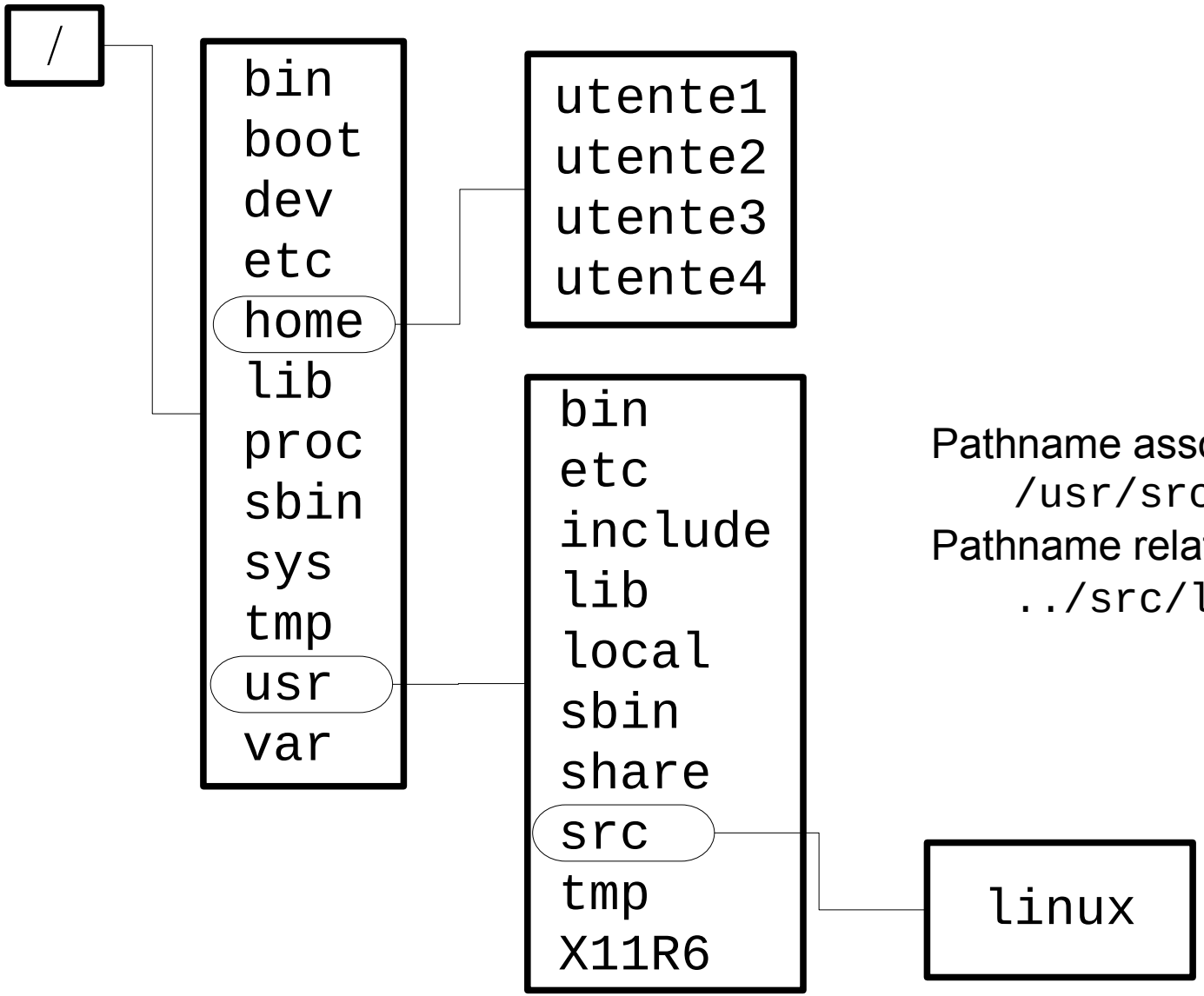
- Se un nome di file contiene caratteri speciali quali **&, *, \, $, ?, <spazio>**, e deve essere usato su linea di comando della shell, bisogna usare degli accorgimenti mediante l'utilizzo del caratteri di escape '\' e il quoting ' ' per evitare che la shell interpreti ed espanda i caratteri speciali

```
[penguin@antarctic DUMMY]$ ls -l
total 24
-rw-rw-r--  1 verola verola 337 Sep  1 11:14 filename with spaces
-rw-rw-r--  1 verola verola   4 Sep  1 11:18 fileWith$Dollar
-rw-rw-r--  1 verola verola  38 Sep  1 11:17 fileWithWildcard*.c
[penguin@antarctic DUMMY]$ file filename with spaces
filename: ERROR: cannot open `filename' (No such file or directory)
with:     ERROR: cannot open `with' (No such file or directory)
spaces:   ERROR: cannot open `spaces' (No such file or directory)
[penguin@antarctic DUMMY]$ file 'filename with spaces'
filename with spaces: ASCII C program text
[verola@localhost DUMMY]$ file fileWith$Dollar
fileWith: ERROR: cannot open `fileWith' (No such file or directory)
[verola@localhost DUMMY]$ file fileWith\$Dollar
fileWith$Dollar: ASCII text
[verola@localhost DUMMY]$
```

Pathname

- Per identificare un file o una directory si utilizza un nome di percorso (*pathname*) composto da nomi di directory separati dal carattere '**/**'; l'ultimo nome è quello della directory o del file in questione
- Il pathname può essere:
  - *assoluto*: inizia per '/' e quindi si riferisce alla root del filesystem
  - *relativo*: inizia con un nome di directory o è composto dal solo nome del file da indirizzare e la ricerca inizia a partire dalla *directory attiva o di lavoro (working directory)* del processo in esecuzione
- Il simbolo '.' indica la directory attiva, mentre '..' indica la directory che contiene la directory attiva (*parent directory*)

Esempio di filesystem Linux



Pathname assoluto:
`/usr/src/linux`
Pathname relativo a `/usr/bin` :
`../src/linux`

Directory principali sotto Linux

`/bin`	Comandi base (*utilities*) per gli utenti
`/boot`	File immagine del kernel
`/dev`	Device file per accedere a periferiche o sistemi di memorizzazione
`/etc`	File di configurazione del sistema
`/home`	"Home directory" degli utenti
`/lib`	Librerie condivise dai programmi e utili per il loro funzionamento
`/proc`	File system virtuale senza reale allocazione su disco. Viene utilizzato per fornire informazioni di sistema, in particolare relative al kernel
`/sbin`	Comandi per la gestione del sistema, destinati al *system administrator*
`/sys`	File relativi ad informazioni sull'HW di sistema
`/tmp`	Directory dei file temporanei
`/usr`	Directory contenente gran parte dei programmi esistenti nel sistema
`/var`	Dati variabili, code di stampa

Comandi relativi al filesystem

- **`pwd`**
  - Print Working Directory
- **`cd` *`directory`***
  - Change working Directory (“go to” directory)
- **`mkdir` *`directory`***
  - MaKe a directory
- **`rmdir` *`directory`***
  - ReMove directory
- **`ls` *`directory`***
  - LiSt directory content
    - **`-a`** all files
    - **`-l`** long listing
    - **`-g`** group information
- **`tree` *`directory`***
  - display directory tree

Gestione dei file

- **rm**
  - ReMove (delete) files
- **cp**
  - CoPy files
- **mv**
  - MoVe (or rename) file
- **ln**
  - LiNk creation (symbolic or not)
- **more, less**
  - page through a text file
- **df [options] [directory]**
  - mostra lo spazio libero nei dischi

```
> df -Tm
```

- **du [options] [directory]**

```
> du
> du directory
> du -s directory
> du -k directory
```

Esercizi

1. Cambiare directory di lavoro, verificare che il comando abbia avuto effetto e poi tornare alla propria home directory.
2. Verificare i file presenti nella directory di lavoro e in **`/etc`**.
3. Trovare le opzioni del comando **`ls`** per:
   a) visualizzare i *file nascosti*
   b) output ricorsivo (lista contenuto tutte subdirectory)
   c) listare i file in ordine alfabetico inverso
   d) stampare l'inode dei file
4. Creare nella propria home directory una nuova directory **`test`**.
5. Trovare opzione di **`cp`** per copiare ricorsivamente un albero di directory.
6. Trovare opzione di **`rm`** che richiede la conferma della cancellazione del file.
7. Rimuovere un file che inizia per '-' (as es. **`-file`**).
8. Trovare opzione di mkdir che permette di creare la directory e le eventuali parent directory, se non esistenti.

Il filesystem nella realtà

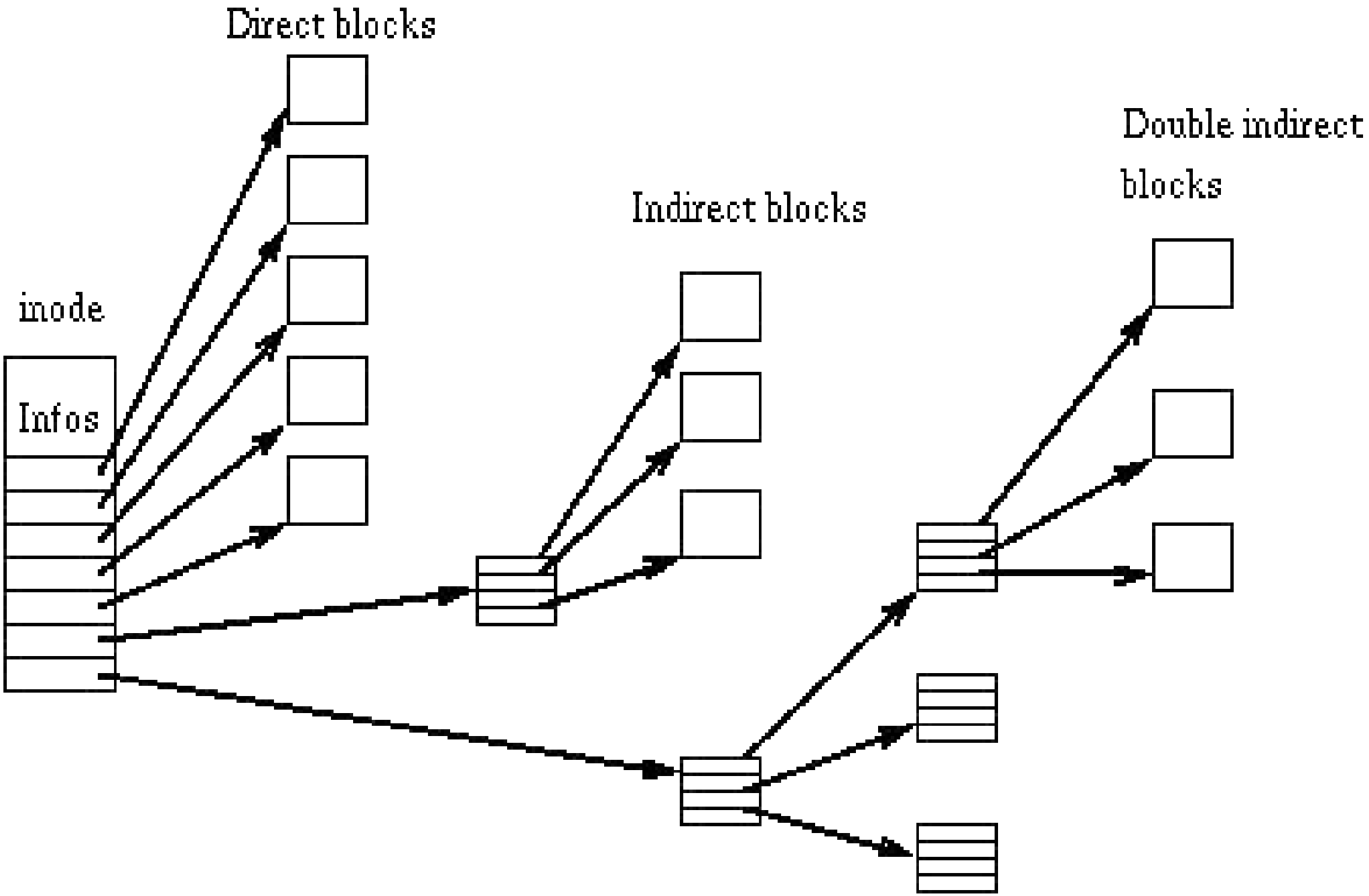
- Anche se per l'utente i file e le directory sono organizzate secondo una struttura ordinata ad albero, il computer in realtà non conosce il significato di tale rappresentazione
- Il disco è diviso in partizioni ed ogni partizione ha un suo filesystem; l'insieme dei filesystem forma la struttura ad albero con cui l'utente interagisce
- I nomi dei file sono utili per l'utente, ma il SO identifica i file mediante i loro *inode number*
- Un *inode number* è un identificatore univoco associato ad una struttura di dati (*inode*) contenente informazioni sulle proprietà del file e sulla locazione fisica dei dati che costituiscono il file stesso
- Ogni partizione ha il suo insieme di *inode:* lo spazio per essi viene generalmente creato durante il processo di installazione del sistema

L'*inode*

- Ad ogni file creato viene assegnato un *inode* con le seguenti informazioni:
  - lunghezza del file in bytes
  - ID del dispositivo contenente il file
  - user ID del proprietario del file e group ID del file
  - *inode number*
  - *file mode,* che determina il tipo e i permessi di accesso e esecuzione del file
  - data di ultima modifica (*mtime*), di ultimo accesso (*atime*) e di ultima modifica dell'*inode* (*ctime*)
  - *link number*, che indica il numero di *hard-link* collegati all'*inode*

inode e blocchi su disco

Esempio di struttura di *inode* (**`ext2`** filesystem)



Attributi dei file

- Per ottenere informazioni complete su un file:

```
% ls -lis prova.txt
72670 8 -rwxr--r-- 1 penguin birds 213 Oct 2  00:12
   prova.txt
```

- Descrizione dei campi:
  - Indice dell'*inode* del file
  - Numero di blocchi utilizzati dal file (1 blocco = 1024 bytes)
  - Tipo e permessi del file
  - Il conteggio di hard-link
  - Username e groupname del possessore del file
  - Dimensione in byte
  - Data di ultima modifica
  - Nome del file
- Si noti che questi sono gran parte dei dati contenuti in un *inode*

Concetto di owner e 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 newUserId file(s)`**
  - normalmente non disponibile in un sistema in cui vengono gestite system quotas

Gestione dei gruppi

- **Come ottenere la lista dei gruppi a cui appartenete**

  `groups [`*`username`*`]`

  invocata senza argomenti, elenca i gruppi a cui appartenete;

  indicando **`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(s)>`**

Permessi dei file (I)

Ogni file è associato a 9 flag chiamati “Permission bits”

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

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

Permessi dei file (II)

- **In realtà:**
  - la gestione dei permessi è leggermente più complessa di quanto presentato
- **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?**
  - Se l'user ID effettivo corrisponde a quello del possessore del file, si applicano le User permission
  - Altrimenti, se il group ID effettivo corrisponde a quello del file, si applicano le Group permission
  - Altrimenti, si applicano le Others permission

Come cambiare i permessi (I)

Relativo: `chmod [ugoa][+-=][rwxXstugo] file(s)`

- 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

Come cambiare i permessi (II)

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

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

Esempi:

- **`chmod 755 public_html`**

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

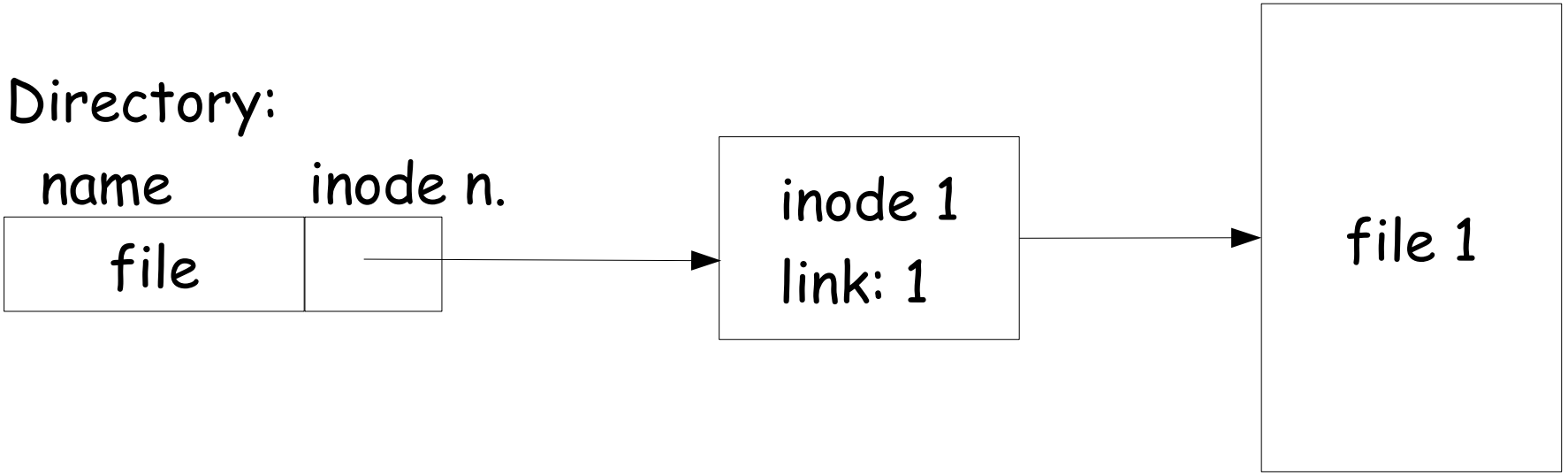
- **`chmod 644 .procmailrc`**

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

Link

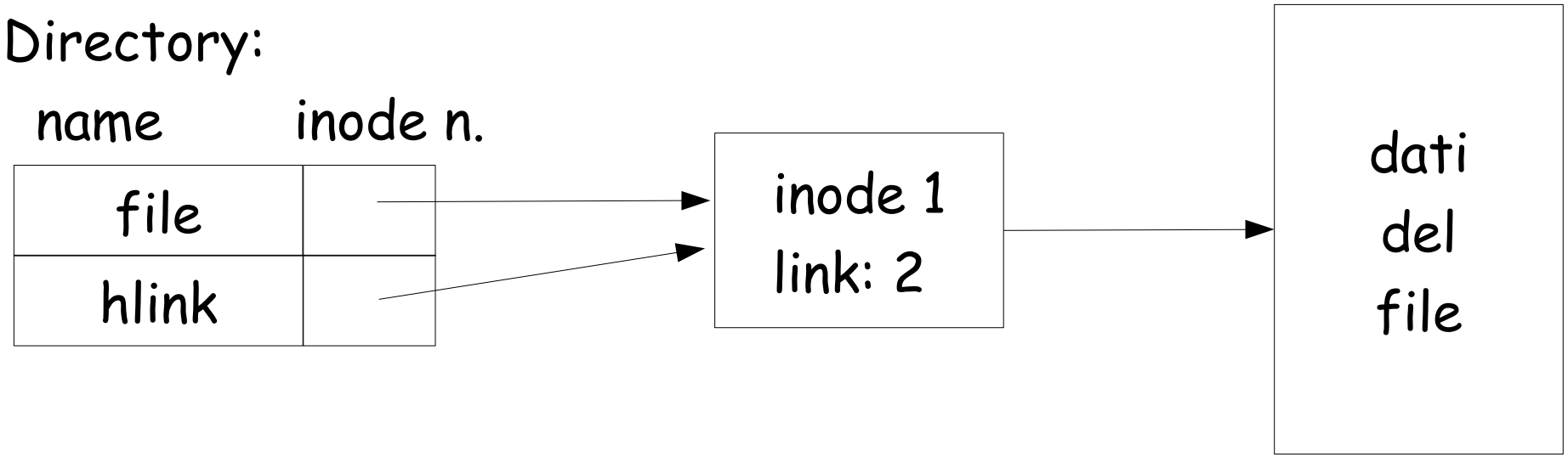
- **`ln` *`file hlink`***
  - E' un hard-link, cioè crea una entry (nella directory corrente) chiamata *hlink* con lo stesso *inode number* di *file*
  - Il *link number* dell'inode di *file* viene incrementato di 1
- **`ln -s` *`file slink`***
  - E' un link simbolico, cioè crea un file speciale (nella directory corrente) chiamato *slink* che "punta" alla entry nella directory di nome *file*
  - Il link number dell'inode di *file* non viene incrementato
- Se cancello ***`file`***:
  - ***hard-link***: il *link number* dell'inode viene decrementato, ma i dati del file non vengono rimossi dal disco, fintanto che il *link number* non diventa uguale a 0.
  - ***link simbolico***: il link diviene "*stale*", ovvero punta ad un file inesistente.

Esempio - Situazione iniziale



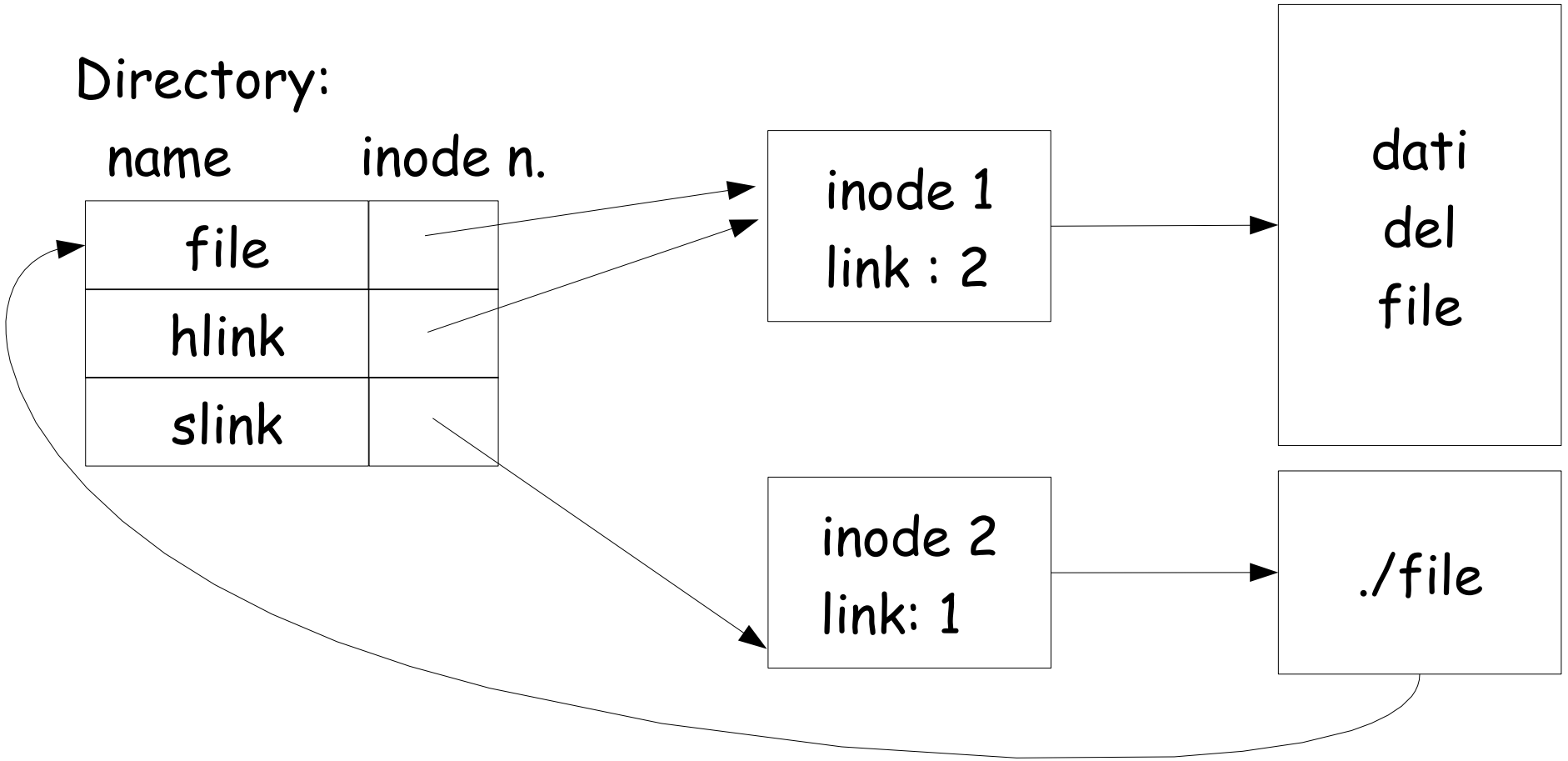
Esiste un file di nome **`file`** con *inode number*=1 e *link number*=1
(cioè una sola entry nella directory si riferisce all'*inode* del file)

Esempio - Creazione di un hard-link



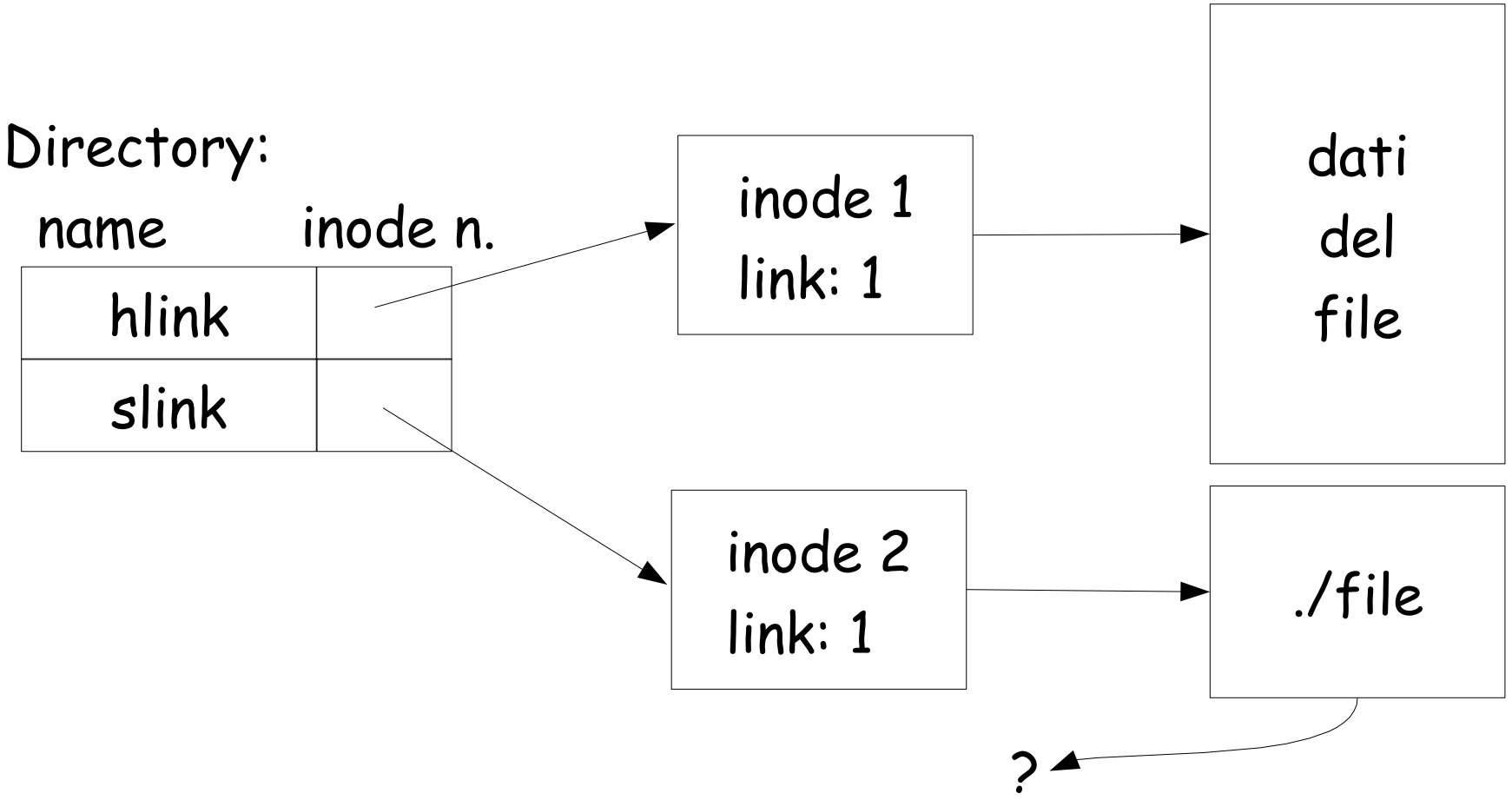
Viene creato un (secondo) *hard link* di nome **`hlink`** che si riferisce all'*inode* 1, quindi il *link number* dell'inode diventa 2 (cioè 2 entry nella directory si riferiscono all'*inode* del file)

Esempio - Creazione di un link simbolico



Viene creato un *link simbolico* di nome **`slink`** che **punta** al file con *inode* 1,
quindi **`slink`** si riferisce a un nuovo inode 2 con *link number*=1;
il contenuto del file **`slink`** è il pathname del file puntato

Esempio - Rimozione del file originale



Se viene rimossa la entry **`file`** dalla directory, il link simbolico **`slink`** pur rimanendo un file esistente, diventa *stale*

Esercizi

1. Utilizzare ls per visualizzare gli attribuiti del file, inode incluso.
2. Qual e' la rappresentazione ottale della maschere di permessi -rwxr----- ? E la rappresentazione simbolica della maschere di permessi ottale 755 ?
3. Rendere una directory non eseguibile, ma leggibile, e fare cd nella dir. Cosa si ottiene? E con ls dir?
4. Rendere una directory non leggibile, ma eseguibile, e fare cd nella dir. Cosa si ottiene? E con ls dir?
5. Come si puo' listare i file in ordine cronologico? Cosa fanno le opzione -u e -c di ls?
6. Analizzare le info fornite dal comando stat.
7. Creare un link simbolico symlink1 che punta a un link simbolico symlink2 che a sua volta punta a symlink1 (dipendenza circolare). Cosa succede se si tenta di leggere il contenuto di uno dei 2 file?
8. Creare una sottodirectory bin all'interno della propria home directory in cui mettere file eseguibili e script. Fare un hard-link al file del comando compress ed un link simbolico al file del comando date.

Attributi dei processi

- **`pid`:** identificatore del processo
- **`ppid`:** parent pid (identificatore del processo padre)
- **`nice number`:** priorità statica del processo; può essere cambiata (diminuita) con il comando **`nice`**
- **`TTY`:** terminal device associato al processo
- ***real, effective user id***
  real, effective group id
  Identificatori dell'owner e del group owner del processo
- altro: memoria utilizzata, cpu utilizzata, etc.

Monitoraggio dei processi

Il comando **ps** riporta lo stato dei processi attivi nel sistema (vedi **man ps** per il significato delle varie colonne)

```
$ ps
PID TTY TIME CMD
648 pts/2 00:00:00 bash

$ ps alx
F   UID   PID  PPID PRI  NI    VSZ   RSS  WCHAN STAT TTY      TIME COMMAND
0     0     1     0  15   0    500   244 1207b7 S    ?        0:05 init
0     0     2     1  15   0      0     0 124a05 SW   ?        0:00 [keventd]
0     0     3     1  34  19      0     0 11d0be SWN  ?        0:00 [ksoftirqd_CPU0]
0     0     4     1  25   0      0     0 135409 SW   ?        0:00 [kswapd]
0     0     5     1  25   0      0     0 140f23 SW   ?        0:00 [bdflush]
0     0     6     1  15   0      0     0 1207b7 SW   ?        0:00 [kupdated]
0     0     7     1  25   0      0     0 15115f SW   ?        0:00 [kinoded]
0     0     9     1  19   0      0     0 23469f SW   ?        0:00 [mdrecoveryd]
0     0    12     1  15   0      0     0 1207b7 SW   ?        0:00 [kreiserfsd]
0     0   150     1   0 -20      0     0 107713 SW<  ?        0:00 [lvm-mpd]
```

Priorità e terminazione dei processi

- Comando **nice**
  - esegue un comando con una priorità statica diversa da quella di default (nel range da -20=massima priorità a 19=minima priorità, 0=default, per dettagli vedi **info nice**)

```
nice -n 19 command
```

- Comando **renice**
  - cambia la priorità di un processo mentre è in esecuzione

```
renice [+-]value -p pid
```

- Comando **kill**
  - manda un segnale a un processo; alcuni segnali possono provocare la terminazione del processo

```
kill -9 pid
```

Lancio e controllo dei processi

- **Processi in foreground**
  - Processi che "controllano" il terminale da cui sono stati lanciati
  - In ogni istante, un solo processo è in foreground
- **Processi in background**
  - Vengono eseguiti senza "controllare" il terminale a cui sono “attaccati”
- **Job control**
  - Permette di portare i processi da background a foreground e viceversa

- **`&`** Lancia un processo direttamente in background
  Esempio: **`long_cmd &`**
- **`Ctrl+z`** Ferma (stop) il processo in foreground
- **`jobs`** Lista i processi in background
- **`%n`** Si riferisce al processo n-esimo in background
  Esempio: **`kill %1`**
- **`fg`** Porta un processo da background a foreground
  Esempio: **`fg %1`**
- **`bg`** Fa ripartire in background i processi fermati

Una lista più completa di tasti di controllo `Ctrl+<x>` e caratteri speciali con il loro significato/azione verrà presentata nel modulo sulla Shell.

Esercizi

1. Esaminare e provare le varie opzioni di **`ps`**.
2. Utilizzare l'utility per un ps periodico: **`top`**.
3. Fare test di lancio processi in foreground e background.
4. Verificare l'effetto dei vari interrupt **`Ctrl+c, Crtl+\, Crtl+z`** su processi in foreground e in background.
5. Trovare l'opzione di **`kill`** per listare i segnali supportati.
6. Trovare il modo di lanciare un processo (ad es. **`sleep 30`**) in foreground, sospenderlo, listarlo nella lista dei job, riattivarlo in background, sospenderlo di nuovo, riattivarlo in foreground.
7. Trovare il modo di verificare l'esistenza di un processo di cui e' noto il PID, utilizzando **`kill`**, ma senza terminarlo.
8. Abbassare la priorita' di un processo alla minima possibile.

Modulo 1 - Sezione B: La shell

Laboratorio di Sistemi Operativi I
Anno Accademico 2006-2007

Francesco Pedullà
(Tecnologie Informatiche)

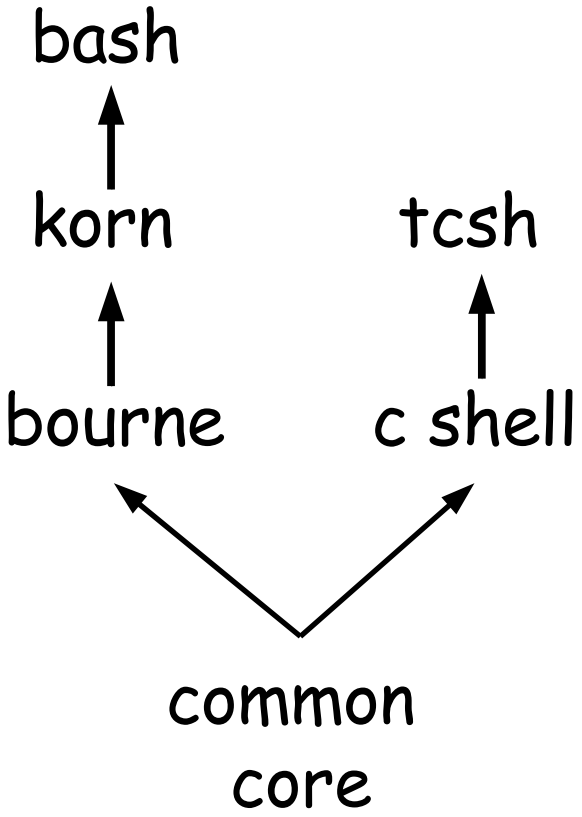
Massimo Verola
(Informatica)

La shell di Linux

- E' un programma che si interpone fra l'utente e il SO (deve il suo nome al fatto che nasconde dietro la sua interfaccia i dettagli del SO sottostante)
- Ha la funzione di interprete della linea comandi ed esecutore di comandi su richiesta dell'utente
- Presenta all'utente il *prompt* (una stringa personalizzabile, per convenzione terminata dal carattere **%** o **$** o **>**) dopo il quale l'utente può digitare i comandi da sottomettere al sistema
- E' programmabile: permette di scrivere delle procedure in un linguaggio interpretato detto *script*
- Per analogia, le interfacce grafiche disponibili per Linux (quali KDE, GNOME) sono a volte identificate come *shell grafiche* o *visuali*

Tipi di shell

- **Esistono diversi tipi di shell**
  - Bourne shell (*sh*)
  - Korn shell (*ksh*)
  - C shell (*csh*, *tcsh*)
  - Bash (Bourne-again shell) (*bash*)
- **Quale shell scegliere?**
  - Questione di gusti ... (ma anche secondo le funzioni e sintassi offerte)
  - Bash è la più diffusa in ambiente Linux
- **Studio della shell**
  - Caratteristiche comuni (common core)
  - Analisi della bash



Caratteristiche delle Shell

- **Sommario:**
  - Comandi interni
  - Metacaratteri
  - Variabili
    - Locali
    - Di ambiente
  - Redirezione dell'I/O
  - Pipes
  - Wildcards
  - Sequenze
    - Condizionali
    - Non condizionali
  - Raggruppamento di comandi
  - Esecuzione in background
  - Command substitution
  - Quoting
  - Utili comandi esterni

Meccanismo di funzionamento

- Quando l'utente digita una linea di comando e la sottomette alla shell mediante il tasto **<RETURN>**, la shell estrae innanzitutto la prima parola
- Assumendo che sia il nome di una programma, lo ricerca nel filesystem (a meno che non sia un comando interno) e lo manda in esecuzione
- Quindi si sospende fino alla terminazione del programma, dopo la quale assume di nuovo il controllo e visualizza il prompt, in attesa del prossimo comando
- Da notare che la shell è un ordinario programma che viene eseguito nello spazio utente, che legge/scrive da/su terminale e che ha la capacità di lanciare altri programmi

Selezionare una shell

- Quando vi viene fornito un account UNIX, una shell di default viene selezionata per voi
- Per vedere che shell state utilizzando:

```
% echo $SHELL # display the content of variable
  SHELL
```

- Per cambiare shell:

```
% chsh [<username>] # asks for the full pathname of
  the new shell
```

```
[penguin@artarctic ~]$ echo $SHELL
/bin/bash
[penguin@artarctic ~]$ chsh
Changing shell for penguin.
Password:
New shell [/bin/bash]: /bin/tcsh
Shell changed.
[penguin@artarctic ~]$
```

Subshells

- **Quando aprite un terminale, viene eseguita una shell**
- **Viene creata una child shell (o subshell)**
  - Quando viene eseguito uno script
  - Quando viene eseguito un processo in background
  - Nel caso di comandi raggruppati, come **`(date; ls; pwd)`**
- **Caratteristiche delle subshell:**
  - Hanno la propria directory corrente:

```
% (cd / ; pwd)
/
% pwd
/home/penguin
```

  - Due distinte aree di variabili vengono gestite differentemente (vedi oltre)

Metacaratteri

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

Comandi

- rappresentano una richiesta di esecuzione
- sintassi generale: **`comando`** ***`opzioni argomenti`***
- gli argomenti indicano l'oggetto su cui eseguire il comando
- le opzioni generalmente:
  - iniziano con - (seguito da una lettera) o -- (seguito da una parola)
  - modificano l'azione del comando
- **Esempi di comandi:**
  - comandi interni della shell
  - script interpretati
  - programmi eseguibili

Tasti di controllo nella shell

`Ctrl-s` sospende la visualizzazione

`Ctrl-q` riattiva la visualizzazione

`Ctrl-c` interrompe il processo

`Ctrl-z` ferma il processo

`Ctrl-d` end-of-file

Comandi interni ed esterni

- **Interni, o *built-in***

  Il comando viene riconosciuto ed eseguito nel contesto (*processo*) della shell corrente.

  Esempi:

```
echo   # displays all of its arguments to standard
  output
cd     # change working directory
```

- **Esterni**

  Il comando corrisponde ad un file eseguibile che viene cercato nel filesystem, caricato in memoria ed eseguito come nuovo processo.

  Esempio: il comando `ls` si trova in `/bin/ls`

Variabili

- **Ogni shell gestisce le variabili in due modi diversi:**
  - *Variabili locali*: non ereditate dalle subshell della shell di partenza
    - Utilizzate per elaborazioni *locali* all'interno di uno script
  - *Variabili di ambiente*: ereditate dalle subshell create dalla shell
    - Utilizzate per comunicazioni da *parent* a *child* shell
- **In ogni caso le variabili contengono solo dati di tipo stringa**
- **Ogni shell ha alcune variabili di ambiente inizializzate da file di startup o dalla shell stessa**
  - **`$HOME`, `$PATH`, `$MAIL`, `$USER`, `$SHELL`, `$TERM`**, etc.
  - Per visualizzare l'elenco completo, usate il comando **`env`**

Significato di alcune variabili d'ambiente predefinite

DISPLAY	used by the X Window system to identify the display server
HISTSIZE	size of the shell history file in number of lines
HOME	path to your home directory
HOSTNAME	local host name
LANG	preferred language
LD_LIBRARY_PATH	paths to search for libraries
LOGNAME	login name
MAIL	location of your incoming mail folder
MANPATH	paths to search for man pages
PATH	search paths for commands
PPID	process ID of the shell's parent. This variable is readonly.
PS1	primary prompt
PS2	secondary prompt
PWD	present working directory
SHELL	current shell
TERM	terminal type
UID	user ID

Utilizzo delle variabili

- **Per accedere al contenuto di una variabile:**
  - Utilizzate il metacarattere **`$`**
  - **`$name`** è la versione abbreviata di **`${name}`**, da usare solo quando non vi siano ambiguità nell'interpretazione
  - se si vuole estrarre una sottostringa: **`${name:start:len}`**
- **Per assegnare un valore ad una variabile:**
  - Sintassi diversa a seconda della shell
  - Nel caso di bash:

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

  - Variabili dichiarate in questo modo sono locali
  - Per trasformare una variabile locale in una d'ambiente, usate il comando **`export`**

```
% export nome
```

Ancora sulle sottostringhe

- E' possibile eliminare la più lunga o la più breve sottostringa iniziale:

```
% myvar="foodforthought"
% echo ${myvar##*fo}
rthought
% echo ${myvar#*fo}
odforthought
```

- E' possibile eliminare la più lunga o la più breve sottostringa finale:

```
% myvar="foodforthought"
% echo ${myvar%%fo*}

% echo ${myvar%fo*}
food
```

Esercizi

- Prevedere l'output del sistema in questa sequenza di comandi:

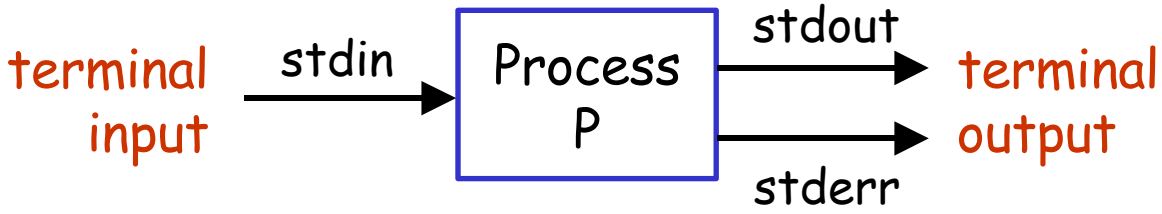
```
% firstname="Penguin"
% lastname="Blackwhite"
% echo $firstname $lastname
% export firstname
% bash
% echo $firstname $lastname
% firstname="Zebra"
% exit
% echo $firstname $lastname
```

- Sia **`myvar='prova1'`** e si voglia creare la nuova variabile **`myvar2`** con valore **`'abcprova1def'`**: quale sintassi usare per l'assegnazione?

- Sia **`myvar='unoduetre'`** e si voglia creare tre variabili contenenti rispettivamente **`'uno' 'due' 'tre'`**: come estrarle da **`myvar`**?

Redirezione dell'input/output e pipes

- **Ogni processo è associato a tre *stream* (flussi di byte)**
  - Standard input (*stdin*) (rediretto con il simbolo < oppure <0)
  - Standard output (*stdout*) (rediretto con il simbolo > oppure >1)
  - Standard error (*stderr*) (rediretto con il simbolo >2)
- **Redirezione dell'I/O e pipe permettono:**
  - "Disconnettere" questi stream dalle loro sorgenti/destinazioni abituali
  - "Connetterli" ad altri sorgenti/destinazioni



Redirezione dell'input/output (I)

- **Usi della *redirection***
  - Salvare l'output di un processo su un file (output redirection)
  - Usare il contenuto di un file come input di un processo
- **Esempi:**
  - Salva l'output di `ls` in `list.txt`

    ```
    ls > list.txt
    ```
  - Aggiunge (*append*) l'output di `ls` a `list.txt`

    ```
    ls >> list.txt
    ```
  - Spedisce il contenuto di `list.txt` a `penguin@antarctic.org`

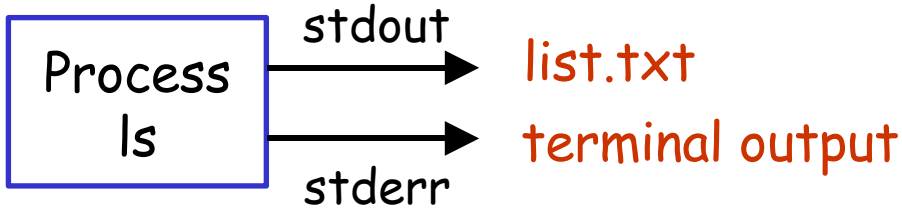
    ```
    mail penguin@antarctic.org < list.txt
    ```
  - Redireziona `stdout` e `stderr` del comando `rm` al file `/dev/null`

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

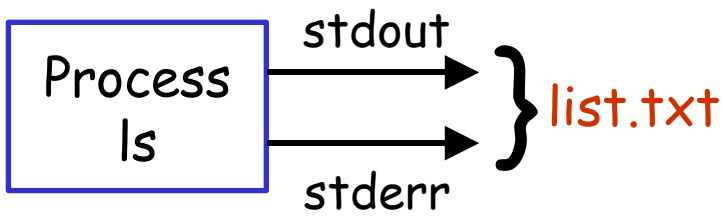
Redirezione dell'input/output (II)

```
mail penguin@antarctic.org < list.txt
```


```
ls > list.txt
```




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



Redirezione dell'input/output (III)

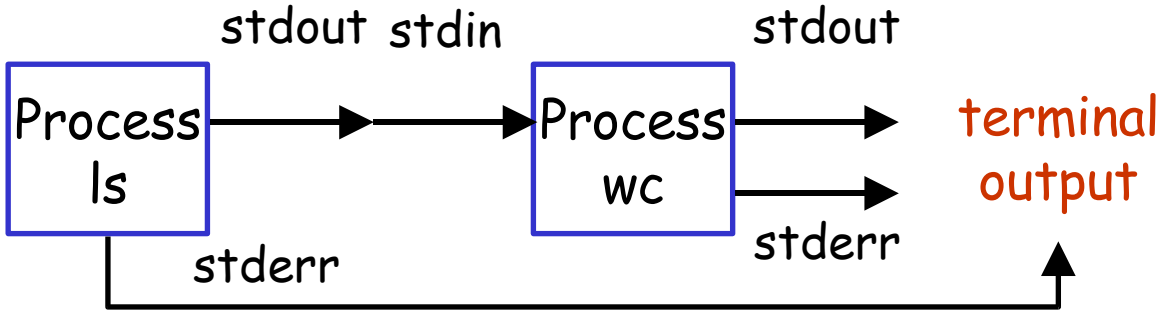
Altri esempi:

```
[penguin@artarctic ~]$ cat 1> myfile
dati del file
<Ctrl+d>
[penguin@artarctic ~]$ cat myfile
dati del file
[penguin@artarctic ~]$ cat myfile myfill > myfile2 2> myfilerr
[penguin@artarctic ~]$ cat myfile2
dati del file
[penguin@artarctic ~]$ cat myfilerr
cat: myfill: No such file or directory
[penguin@artarctic ~]$ cat myfile myfill >& myfile3
[penguin@artarctic ~]$ cat myfile3
dati del file
cat: myfill: No such file or directory
[penguin@artarctic ~]$
```

Pipes

- **Pipe, o catena di montaggio:**
  - La shell vi permette di usare lo standard output di un processo come standard input di un altro processo

```
% ls
a b c d f1 f2
% ls | wc -w
6
```



- **Esempio:**
  - `who | cut -c1-8 | sort -u | pr -l1 -8 -w78 -t`

`who` lista gli utenti che hanno fatto login

`cut -c1-8` mostra le colonne da 1 a 8 dell'output di `who`

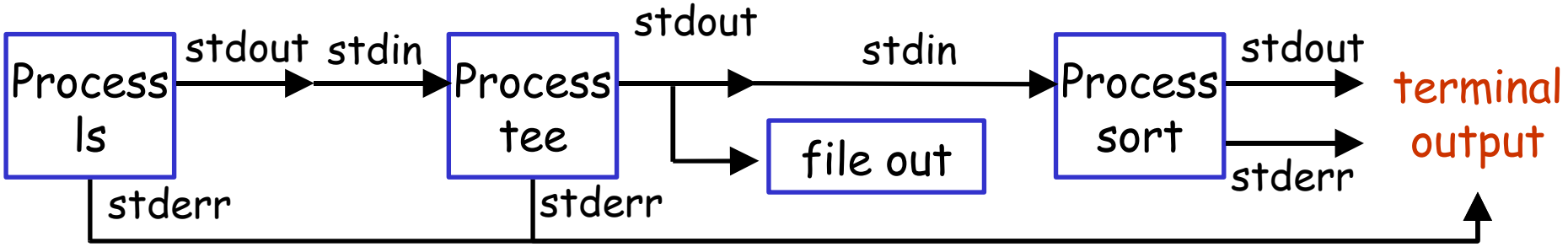
`sort -u` ordina in nomi ed elimina righe multiple con stesso nome

`pr -l1 -8 -w78 -t` mostra le informazioni disposte su 8 colonne, un nome per colonna, su linee di 78 caratteri

Duplicazione dell'output

- **Comando `tee`**
  - copia il contenuto dello standard input sia sul file specificato sia sullo standard output (il nome **`tee`** deriva dai "giunti a T" usati dagli idraulici!)
  - utile quando si vuole mandare l'output in una pipe ma allo stesso tempo salvarlo in un file

```
[penguin@artarctic]$ ls -t -1 *.txt | tee out | sort
due.txt
tre.txt
uno.txt
[penguin@artarctic]$ cat out
tre.txt
due.txt
uno.txt
```



Wildcards per filename expansion

- **Utilizzate per specificare filename pattern**
  - La shell sostituisce la stringa contenente *wildcards* 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:
    - `*.c`
    - `prova00?.c`
    - `prova[0-9][!3][v,V].txt`

Esercizi

- Spiegare in dettaglio come la shell tratta il seguente comando:
  `mail penguin@antarctic.org < list.txt`
- Scrivere un comando che emuli il comando **`ls *file*`** senza usare il comando **`ls`**
- Il comando **`grep PATTERN [FILE ...]`** cerca nei file indicati l'espressione regolare **`PATTERN`** e stampa solo le linee che la contengono; esiste anche l'opzione **`-r (o -R)`** che permette di cercare **`PATTERN`** ricorsivamente nei file della directory indicata e in tutte le subdirectory ricorsivamente; scrivere un comando che emuli **`grep -r PATTERN DIR`**, utilizzando grep (ed altri comandi) ma senza ricorrere all'opzione **`-r`**

Sequenze (condizionali e non)

- **Sequenze non condizionali**
  - Il metacarattere **;** viene utilizzato per eseguire due comandi in sequenza
  - Esempio:

```
% date ; pwd ; ls
```

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

```
% gcc prog.c && a.out
% gcc prog.c || echo Compilazione fallita
```

Raggruppamento di comandi

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

```
[penguin@artarctic]$ date ; ls ; pwd > out.txt
Mon Sep  4 23:45:34 CEST 2006
Desktop  DUMMY  main.c   myfile2   out
[penguin@artarctic]$ cat out.txt
/home/penguin
[penguin@artarctic]$ (date ; ls ; pwd) > out.txt
[penguin@artarctic]$ cat out.txt
Mon Sep  4 23:45:34 CEST 2006
Desktop
DUMMY
main.c
myfile2
out
/home/penguin
```

Esecuzione 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 richiedonoi input dall'utente
  - Esempi:

```
% find / -name passwd -print &
20123
% /etc/passwd
% find / -name passwd -print >& results.txt &
20124
```

PID del processo find

output del comando find lanciato in background

Command substitution

- **Gli apici ` ` (accento grave) sono utilizzati per fare *command substitution***
  - Il comando racchiuso fra apici viene eseguito, e il suo standard output viene sostituito al posto della stringa del comando
  - Esempi:

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

- **Sintassi alternativa con `$(...)`:**

```
% today=$(date)
```

Quoting

- **Esiste la possibilità di disabilitare wildcard / command substitution / variable substitution (cioè l'interpretazione della shell di questi metacaratteri)**
  - Single quotes `''` inibiscono wildcard, command substitution, variable substitution
  - Esempio:

```
% echo 3 * 4 = 12
% echo '3*4 = 12'
```

  - Double quotes `" "` inibiscono wildcard e basta
  - Esempio:

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

Help in linea

- Per fare riferimento alla documentazione on-line
  - `man` ***`command`***
  - `info` ***`command`***
  - `apropos` ***`keyword`***
- Per informazioni sull'utilizzo di `man` ed `info`:
  - `man man`
  - `info`

Leggere la documentazione

- **Nella documentazione e nei libri si trovano varie sintassi per la documentazione. Ad esempio:**

  `command` ***`[opt-arg] mandatory-arg {rep-arg}*`***

  - ***`opt-arg`*** è opzionale
  - ***`mandatory-arg`*** è obbligatorio
  - ***`rep-arg`*** può essere ripetuto **n** volte, con **n** >= 0, oppure si utilizza … per indicare argomento ripetibile
  - le opzioni (con **-**) possono raggruppate, e spesso non sono racchiuse fra parentesi quadre
  - corsivo per argomenti da sostituire

- **Esempi:**

  `uniq –c –`*`number`*` [ `*`inputfile`*` [ `*`outputfile`*` ] ]`

  `mv [OPTION]... SOURCE... DIRECTORY`

Esercizi

- Qual é l'effetto del comando **`sort file > file`**, dove **`file`** è il nome di un file? E quello del comando **`sort file >> file ?`**
- Fare alcuni esperimenti per scoprire qual é l'effetto del comando **`tr str1 str2`** se le stringhe **`str1`** e **`str2`** hanno lunghezze diverse. Scrivere un comando per sostituire tutti i caratteri alfanumerici nell'input con un carattere **`<Tab>`**, in modo che non compaiano più **`<Tab>`** consecutivi.
- Il comando **`date`** fornisce data e ora su standard output. Studiarne la sintassi per estrarre soltanto l'ora.
- Estrarre da **`/etc/passwd`** il primo campo della prima riga

Esercizi

- Mediante **`ls`**, listare SOLO le subdirectory contenute in una directory assegnata.
- Trovare il modo per rimuovere un file conoscendo SOLO il suo *i-node*.
- Selezionare un editor e impadronirsi dei comandi base.
- Scrivere un file testo ASCII contenente una lista di comandi in sequenza per contare il numero di linee dei file il cui nome finisce per '**`.txt`**' presenti in una directory, il cui nome deve essere prelevato dalla variabile d'ambiente **`MYDIR`**.
  L'output deve andare sia su file che a video e deve contenere:
  - il pathname della working directory
  - il pathname della directory selezionata
  - il nome dei file '**`.txt`**' trovati e il loro numero
  - il numero di linee totale di tutti i file '**`.txt`**' trovati
  - il numero di tutti i file regolari nella directory
  - il numero di tutte le subdirectory nella directory
- Studiare i fondamenti di **`sed`** sulle slide e risolvere il quesito:
  mediante sed sostituire nel file di input tutte le cifre da 0 a 9 con -

Appendice 1
-o-o-
Breve rassegna di comandi (esterni) in ambiente Linux

Comandi: `cat` - `tac` - `rev`

- **Comando `cat` (concatenate):**
  - Stampa i file su stdout. Utilizzato con redirection, serve a concatenare file:

```
cat file.1 file.2 file.3 > file.123
```

- **Comando `tac` (inverso di cat):**
  - Stampa le linee dei file in ordine inverso, partendo dall'ultima linea
- **Comando `rev` (reverse):**
  - Stampa (invertite) le linee dei file, partendo dalla prima linea

Comando: `find` (I)

- **Comando: `find` *`pathlist expression`***
  - L'utility **`find`** analizza ricorsivamente i path contenuti in ***`pathlist`*** e applica ***`expression`*** ad ogni file
- **Sintassi di expression:**
  - ***`-name pattern`***

    True se il nome del file fa matching con pattern, che può includere i metacaratteri di shell: * [ ] ?
  - ***`-perm permission`***

    True se permission corrisponde ai permessi del file
  - ***`-print`***

    stampa il pathname del file e ritorna true

Comando: `find` (II)

- **Sintassi di expression:**
  - **`-ls`**

    Stampa gli attributi del file e ritorna true
  - **`-user` *`username`*`, -uid` *`userId`***

    True se il possessore del file è ***`username`*** / ***`userId`***
  - **`-group` *`groupname`*`, -gid` *`groupId`***

    True se il gruppo del file è ***`groupname`*** / ***`groupId`***
  - **`-atime | -mtime | -ctime -`*`count`***

    True se il file è stato “acceduto | modificato | modificato oppure cambiati gli attributi” negli ultimi ***`count`*** giorni

Comando: `find` (III)

- **Sintassi di expression:**
  - **`-type b | c | d | p | f | l | s`**

    True se il file è di tipo a blocchi | a caratteri | una directory | un named pipe | un file regolare | un link | un socket
  - **`-exec` *`command`***

    True se l'exit status di ***`command`*** è 0.
    `command` deve essere terminato da un "escaped ;" (\;).
    Se si specifica il simbolo {} come argomento di ***`command`*** , esso viene sostituito con il pathname del file corrente
  - **`-not, !, -a, -and, -o, -or`**

    Operatori logici (not, and, or)

Comando `find`: esempi

- `find . -name "*.c" -print`
  - Stampa il nome dei file sorgente C nella directory corrente e in tutte le sottodirectory
- `find / -mtime -14 -ls`
  - Elenca tutti i file che sono stati modificati negli ultimi 14 giorni
- `find . -name "*.bak" -ls -exec rm {} \;`
  - Cancella tutti i file che hanno estensione .bak

Comando: `xargs`

- **Comando: `xargs` *`command`***
  - **`xargs`** legge una lista di argomenti dallo standard input, delimitati da blank oppure da newline
  - esegue ***`command`*** passandogli la suddetta lista di argomenti
- **Esempio: concatena tutti i file `*.c`**

```
% find . -name "*.c" -print
./a.c
./b.c
% find . -name "*.c" -print | xargs cat > prova
```

Esempi: `find` e `xargs`

- `vi $(find . -name "*.java" | xargs grep -l "Alfa")`
  - Utilizza vi per visualizzare tutti i file nella directory corrente e nelle sottodirectory che hanno estensione java e contengono la parola "Alfa".
- `find ~ \( -name "*~" -o "#*#" \) -print | \`
  `xargs --no-run-if-empty rm -vf`
  - Rimuove tutti i file di backup o temporanei di emacs dalla home directory (ricorsivamente)
- `find . -type d -not -perm ug=w | xargs chmod ug+w`
  - Aggiunge il diritto di scrittura a tutti le directory nella directory corrente e nel suo sottoalbero

Esercizi

- E' possibile eseguire la funzione del primo esempio del lucido precedente senza usare il comando **`xargs`**?
- E' possibile eseguire la funzione del secondo esempio del lucido precedente senza usare il comando **`find`**?
- E' possibile eseguire la funzione del terzo script del lucido precedente senza usare il comando **`find`**?

Spazi e caratteri speciali

- `find ~ \( -name "*~" -o "#*#" \) -print0 | \`
  `xargs --no-run-if-empty --null rm -vf`
- **Attenzione agli spazi e ai caratteri speciali:**
  - **`xargs`** separa i nomi dei file tramite spazi o new line
  - il file **`relazione 1.txt`** viene trattato come due file
  - l'opzione **`-print0`** di **`find`** stampa stringhe null-terminated
  - l'opzione **`--null`** di **`xargs`** prende stringhe null-terminated
  - questa versione gestisce correttamente qualunque tipo di carattere speciale (come ")

Esempio

```
#!/bin/bash

# Move (verbose) all files in current directory

# to directory specified on command line.

if [ -z "$1" ];  then

   echo "Usage: `basename $0` directory-to-copy-to"

   exit 65

fi

ls . | xargs -i -t mv {} $1

 # This is the exact equivalent of mv * $1

 # unless any of the filenames has "whitespace"
   characters.

exit 0
```

Comandi per la gestione del testo

- **Comando: `head [-n] file`**
  - Lista le prime *n* linee di un file (10 default)
- **Comando: `tail [-n] file`**
  - Lista le ultime *n* linee di un file (10 default)
- **Comando: `cut`**
  - Un tool per estrarre campi dai file. Nonostante esistano altri tool (più sofisticati), **`cut`** è utile per la sua semplicità.
  - Due opzioni particolarmente importanti:
    - **`-d delimiter`**: specifica il carattere di delimitazione (tab default)
    - **`-f fields`**: specifica quali campi stampare

Comandi per la gestione del testo

- **Comandi: `expand, unexpand`**
  - L'utility **expand** converte i tab in spazi. L'utility **unexpand** converte gli spazi in *tab.*
- **Comando: `uniq`**
  - Questa utility rimuove linee duplicate (consecutive) dallo standard input. Viene usato spesso nei pipe con sort.
- **Comando: `sort`**
  - Ordina lo standard input linea per linea. E' in grado di eseguire ordinamenti lessicografici sulle linee, o di gestire l'ordinamento dei vari campi.
  - Ad esempio: l'opzione **-g** ordina in modo numerico secondo il primo campo dell'input

Esempio

```
% du -s /home/*
10000 /home/penguin
500 /home/black
2345 /home/white
26758 /home/gray
% du -s /home/* | sort -gr
26758 /home/gray
10000 /home/penguin
2345 /home/white
500 /home/black
```

Esempio (cont.)

```
% du -s /home/* | sort -gr | head -2

26758 /home/white

10000 /home/penguin

% du -s /home/* | sort -gr | head -2 | cut -f2

/home/white

/home/penguin

% for homedir in $(du -s /home/* | sort -gr | \
  head -2 | cut -f2); do echo $(basename $homedir) ; \
  done

white

penguin
```

Comandi per la gestione del testo

- **Comando: `wc` (word count)**
  - Conta linee, parole, caratteri
  - **`-l | -w | -c`** conta solo le linee | le parole | i caratteri
  - Certi comandi includono le funzionalità di **`wc`** come opzioni:

    `... | grep foo | wc -l` è equivalente a

    `... | grep --count foo`

- **Comando: `tr`**
  - Utility per la conversione di caratteri, seguendo un insieme di regole definite dall'utente:
  - Esempio: **`tr A-Z a-z < filename`**
    - Stampa filename trasformando tutti i caratteri in minuscoli
  - Esempio: **`tr -d 0-9 < filename`**
    - Stampa filename eliminando tutte i caratteri numerici

Esempio: filenames-to-lowercase

```
#! /bin/bash
# Changes every filename in working directory
# to all lowercase.
for filename in * ; do
  fname=`basename $filename`
  n=`echo $fname | tr A-Z a-z`
    # Change name to lowercase.
  if [ "$fname" != "$n" ] ; then
    # Rename only files not lowercase.
    mv "$fname" "$n"
fi
done
exit 0
```

Esempio: dos2unix

```
!/bin/bash

# dos2unix.sh: DOS to UNIX text file converter.

E_WRONGARGS=65

if [ -z "$1" -a -z "$2"]; then

  echo "Usage: `basename $0` file-source file-dest"

  exit $E_WRONGARGS

fi

CR='\015'  # Lines in DOS text files end in a CR-
  LF.

tr -d $CR < $1 > $2 # Delete CR and write to $2

exit 0
```

Comandi per il confronto di file

- **Comando:** `cmp` ***`file1 file2`***
  - Ritorna true (exit status 0) se due file sono uguali, ritorna false (exit status != 0) altrimenti
  - Stampa la prima linea con differenze
  - Con l'opzione `-s` non stampa nulla (utile per script)
- **Comando:** `diff` ***`file1 file2`***
  - Ritorna true (exit status 0) se due file sono uguali, ritorna false (exit status != 0) altrimenti
  - Stampa un elenco di differenze tra i due file (linee aggiunte, linee modificate, linee cancellate)
- **Comando:** `diff` ***`dir1 dir2`***
  - Confronta due directory e mostra le differenze (file presenti in una sola delle due)

Comandi per la gestione di file

- **Comando: `locate, slocate, updatedb`**
  - I comandi **`locate`** e **`slocate`** (secure version di **`locate`**) cercano file utilizzando un database apposito. Il database riflette il contenuto del file system, ma va aggiornato con **`updatedb`** (solo root)
- **Comando: `file` *`file`***
  - Identifica il tipo di file a partire dal suo *magic number* (quando disponibile). L'elenco dei magic number si trova in **`/usr/share/magic`** (o altre posizioni, consultate info file)
  - Esempio:

```
% file prova.sh

prova.sh: Bourne-Again shell script text executable
```

Comandi per la gestione dei file

- **Comando: `basename` *`file`***
  - rimuove l'informazione del path da un pathname
- **Comando: `dirname` *`file`***
  - rimuove l'informazione del nome di file da un pathname
- **Nota: sono funzioni di stringa, non agiscono su un file effettivo**
- **Esempi:**

```
% basename /home/penguin/index.html
index.html
% dirname /home/penguin/index.html
/home/penguin
echo "Usage: `basename $0` arg1 arg2 ... argn"
```

Archiviazione e compressione

- **Compressione:**
  - Comandi: **`compress`**, **`uncompress`**
  - Comandi: **`gzip`**, **`gunzip`**
  - Comandi: **`bzip2`**, **`bunzip2`**
  - Comprimono e decomprimono file. **`compress`** è ormai in disuso (originario dei primi sistemi Unix); **`bzip2`** è meno diffuso (almeno per ora), ma è in alcuni casi più efficiente di **`gzip`**
- **Archivazione: tar**
  - Archivazione: **`tar zcvf`** ***`archive-name {file}*`***
    - **`z`**=comprimi, **`c`**=crea, **`v`**=verbose, **`f`**=su file
  - Estrazione: **`tar zxvf`** ***`archive-name`***
    - **`z`**=espandi, **`x`**=estrai, **`v`**=verbose, **`f`**=da file

Comandi per la gestione del tempo

- **Comando: `touch file`**
  - utility per modificare il tempo di accesso/modifica di un file, portandolo ad un tempo specificato (**`touch -d`**)
  - può essere utilizzata anche per creare un nuovo file
  - creare file vuoti può essere utile per memorizzare solo la data
- **Comando: `date`**
  - Stampa informazioni sulla data corrente, in vari formati
- **Comando: `time command`**
  - esegue il comando *command* e stampa una serie di statistiche sul tempo impiegato
  - Esempio: **`time find / -name "*.bak" -print`**

Appendice 2
-o-o-
L'editor *vi* e cenni su *sed*

Cos'è VI

- **Vi** è l'editor *storico* di UNIX ed è disponibile su tutte le piattaforme (AIX, HP/UX, Solaris) oltreché su Linux e su Windows
- E' piccolo: chiede poche risorse al sistema
- E' potente: ha un insieme di comandi completo e flessibile
- E' un editor di testo per cui produce solo file di testo puri a differenza, per esempio, di OpenOffice
- Esistono versioni *migliorate*:
  - Vim: aggiunge, tra l'altro, multi level undo, multi windows e buffers, syntax highlighting, command line editing, filename completion, on-line help, visual selection
  - Gvim: maggiore integrazione con l'interfaccia grafica (e.g., paste)
- **Emacs** è un altro editor tradizionale in ambiente UNIX/Linux, ancora più potente ma più complesso

Cosa studiare

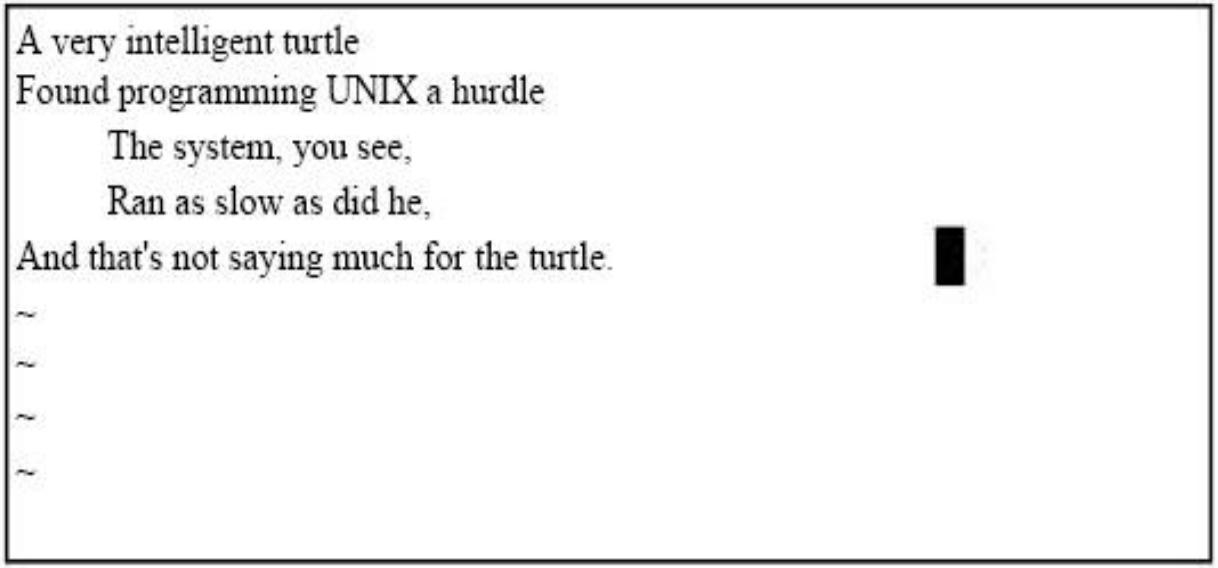
- Per approfondimenti potete usare:
  - Vimbook-OPL (reperibile all'URL http://truth.sk/vim/vimbook-OPL.pdf)
  - Vim-HOWTO (scaricabile da http://www.tldp.org)
  - Vimdoc (scaricabile da http://vimdoc.sf.net)
- Esistono molti altri tutorial disponibili su Internet che potete utilizzare
- Seguiremo soprattutto il primo testo (i primi tre capitoli)
- Il corso comprende solo questa lezione introduttiva che illustra i concetti fondamentali nell'uso di **`vim`**
- Una lezione non basta per usare bene **`vi`**: dovrete leggere uno dei documenti
- Ai fini dell'esame bisogna padroneggiare bene un editor di testo (vi o emacs)

Aprire un file

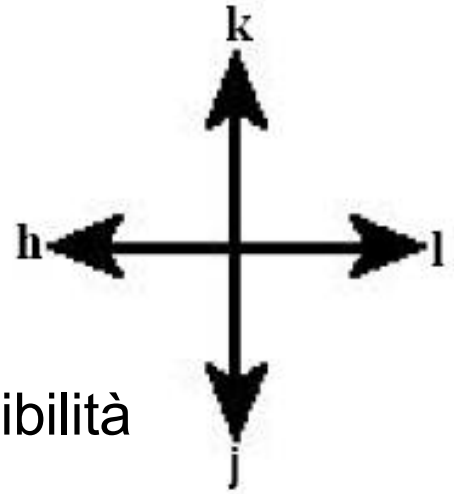
- Dalla linea comandi, nella directory in cui si vuole creare il file:
  - **`vim file.txt`**
- Il carattere ~ indica righe che non appartengono al file
- **`vim`** parte in modo *normale*
  - L'ultima riga mostra alcune informazioni utili
  - I caratteri digitati in modo normale non sono inseriti nel documento ma interpretati come comandi

Inserire testo nel file

- Per passare in modo *inserimento* digitare **`i`**
- Scrivere il testo desiderato
- Si può cancellare il testo con il tasto **BACKSPACE**
- Per inserire una nuova linea digitare **RETURN**
- Inserito il testo, uscire dal modo *inserimento* premendo **ESC**



Muoversi nel testo



- Per muovere il cursore abbiamo due possibilità:
  - Usare **`h,k,j,l`** per muoversi come in figura
  - Usare le frecce
- Per muovere il testo di una pagina abbiamo due possibilità
  - Usare **`Ctrl-u`** e **`Ctrl-d`** per salire o scendere
  - Usare i tasti **`PgUp`** e **`PgDn`**
- Per muovere il cursore lungo una linea di testo abbiamo due possibilità:
  - Usare **`0`** o **`$`** per andare al'inizio o alla fine della linea
  - Usare i tasti **`Home`** o **`End`**
- L'uso dei tasti speciali (frecce, etc.) è permesso sia in modo normale che in modo inserimento

Modificare il testo

- Per cancellare un carattere abbiamo due possibilità:
  - Usare **`x`**
  - Usare **BACKSPACE**
- Per cancellare una linea (tutta o in parte):
  - Tutta la linea (ovunque sia il cursore): **`dd`**
  - Il resto della linea (a partire dal cursore): **`D`**
- Per inserire nuovo testo:
  - Nella posizione del cursore: **`i`**
  - Dalla riga successiva: **`o`**
  - Dalla riga precedente: **`O`**
  - Dal carattere successivo: **`a`**
  - Alla fine della linea: **`A`**
- Altri comandi utili
  - Undo: **`u`**
  - Redo: **`Ctrl-r`**

Come terminare

- Bisogna andare in modo *comando* digitando il carattere **:**
- Per uscire dall'editor:
  - Se si vuole anche salvare il file, usare il comando **ZZ** (senza entrare in modo comando) oppure **:wq**
  - Se si vuole soltanto uscire, usare il comando :**q**
  - Se si vuole uscire senza salvare il file, usare il comando :**q!**
  - Se si vuole solo salvare il file, usare il comando :**w**
- Per uscire dal modo comando e tornare al modo normale usare **ESC**

Riassunto dei modi e dei comandi principali

Modo	Per entrare	A cosa serve
Normale	ESC	Muovere il cursore
Inserimento	i	Inserire il testo
Comando	:	Salvare/chiudere il file

Modo	Comando	Funzione
Normale	h,k,j,l	Spostamento del cursore (un carattere)
	Ctrl-u, Ctrl-d	Cambio pagina
	0, $	Spostamento del cursore (inizio o fine linea)
	X	Cancellazione di un carattere
	dd, D	Cancellazione della linea (intera o in parte)
	u, Ctrl-r	Undo, repeat
Inserimento	i,o,O,a,A	Inserimento nel testo
	BACKSPACE	Cancellazione di un carattere
	PgUp, PgDn	Cambio pagina
	Home, End	Spostamento del cursore (inizio o fine linea)
	Frecce	Spostamento del cursore (un carattere)
Comando	ZZ	Esci e salva
	q	Esci
	q!	Esci senza salvare
	w	Salva

Esercizio

- Creare due file, **usage.sh** e **max.sh**, contenenti rispettivamente:

```
#!/bin/bash
usage () {
   if [ -n "$1" ] ; then
      echo "Usage: " \
        "$1 [options] args"
   fi
}

# Prints the usage
usage $(basename $0)
```

```
#!/bin/bash
max () {
   if [ $1 -gt $2 ] ; then
      return $1
   else
      return $2
   fi
}

max 12 14
echo $?
```

Ripetizione di comandi e help

- E' possibile ripetere automaticamente il comando, premettendo allo specifico comando un numero:
  - **`10x`** cancella 10 caratteri
  - **`5dd`** cancella 5 linee
  - **`3$`** muove il cursore alla fine della terza linea successiva
- Per accedere all'help online, andare in modo comando e digitare **`help`** oppure premere **`F1`**
- E' disponibile anche un tutor con il comando (dalla shell Linux!) **`vimtutor`**

Altri comandi di movimento

- Sono disponibili comandi di movimento più sofisticati:
  - Per avanzare (o indietreggiare) di una parola, usare il comando **`w`** (o **`b`**): **`5w`** sposta il cursore 5 parole più avanti
  - Per cercare un carattere sulla linea corrente, usare il comando **`f`** (o **`F`** verso sinistra): **`3fy`** ricerca la terza "y" nella linea del cursore
  - Per andare alla linea numero **`n`** (inizio prima parola) usare il comando n**`G`** (**`G`** senza argomento va alla fine del file)
- Per conoscere la posizione del cursore nel file: **`Ctrl-g`**
- Per vedere i numeri di linea del testo usare il comando **`:set number`** (per eliminarli **`:set nonumber`**)

I comandi d, c, y, p

- Il comando **d** (delete) può essere seguito da un comando di movimento e cancella il testo dal cursore fino al testo trovato:
  - **dw** cancella fino alla fine della parola
  - **d3w** cancella le tre parole successive
  - **d$** cancella fino alla fine della linea (come i comandi **dd** e **D**)
- Anche il comando **c** (change) può essere seguito da un comando di movimento
  - **cw** cambia la parola successiva
  - **c$** cambia il testo fino alla fine della linea (come i comandi **cc** e **C**)
- Lo stesso vale per **y** (yank)
  - **yw** copia la parola seguente nel buffer
  - **yy** copia tutta la linea
- Il comando **p** (put) inserisce il contenuto del buffer a partire dalla posizione del cursore

Altri comandi di modifica

- Il comando **J** (join) unisce la linea corrente e quella successiva:
  - **3J** unisce tre linee
- Il comando r (replace) sostituisce il carattere dove è posizionato il cursore
  - **rx** sostituisce il carattere con **x**
  - **3rx** sostituisce tre caratteri con **x**
- Il comando **R** sostituisce il testo finché non si esce dal modo di inserimento
- Il comando ~ scambia minuscole con maiuscole (e viceversa)
  - **5~** scambia maiuscole con minuscole nei 5 caratteri successivi
- Il comando **.** ripete l'ultimo comando di modifica

Ricerca

- Per cercare una stringa usare il comando **/**
  - **/abc** ricerca abc a partire dal cursore
- Alcuni caratteri nella stringa di ricerca hanno un significato speciale:
  - **$** indica la fine della linea
  - **^** indice l'inizio della linea
  - **.** indica un qualunque carattere
  - Altri caratteri speciali: ***[]\?~/%**
  - Se la stringa cercata contiene un carattere speciale, questo va preceduto da \
- Il comando **?** esegue la ricerca all'indietro
- Il comando **n** ripete la ricerca

Le macro - I

- Supponiamo di voler modificare un file come segue:

```
stdio.h
fcntl.h
unistd.h
stdlib.h
main()
{
...
```

```
#include "stdio.h"
#include "fcntl.h"
#include "unistd.h"
#include "stdlib.h"
main()
{
...
```

Le macro - II

- Registriamo una macro che modifica come richiesto una singola riga, e poi la ripetiamo per ogni riga successiva
  - **qa** inizia la registrazione nel registro **a** (i registri vanno da **a** a **z**)
  - **^** va all'inizio della linea
  - **i#include"<ESC>** inserisce il testo prima del nome del file
  - **$** va alla fine della linea
  - **a"<ESC>** aggiunge il carattere " alla fine della linea
  - **j** va alla linea successiva
  - **q** finisce la registrazione
- La macro può essere invocata con **@a** (con **3@a** l'intera modifica è fatta!)

Esercizi

- Aprire un file di testo con **`vi`** ed eseguire le seguenti operazioni:
  - inserire una nuova linea di testo in fondo al file;
  - copiare le ultime 4 linee del file all'inizio del file;
  - sostituire tutte le occorrenze della stringa *are* con il carattere - ;
  - salvare le modifiche.
- Aprire un file con **`vi`** e ripetere l'esempio della macro dopo avere inserito il testo su cui intervenire

sed (I)

- Il nome del comando **`sed`** sta per **S**tream **ED**itor e permette di editare un testo, per esempio letto da stdin in una pipeline oppure da un file.
- La sua sintassi è la seguente: **`sed actions files`**
- **`actions`** è un'espressione composta dagli indirizzi di linea da un'azione, generalmente indicata da un carattere, da svolgere sugli indirizzi specificati (per la corretta sintassi, che infatti risulta molto simile a quella dei comandi accettati da **`vi`**, si veda **`info sed`**).
- Le **`actions`** possono manipolare il testo in vario modo:
  `i` = inserisci testo, **`s`** = trova e sostituisci, **`y`** = trasforma caratteri,
  `d` = cancella, **`p`** = stampa, **`q`** = esci da sed, ...
- Se **`actions`** è una stringa vuota (' '), **`sed`** stampa sullo standard output le linee in input, lasciandole inalterate.

sed (II)

- Gli indirizzi di linea si specificano come numeri o espressioni regolari:
  '4' quarta linea
  '4,8' dalla quarta all'ottava linea (incluse)
  '4,$' dalla quarta all'ultima linea
  '1~2' a partire dalla prima linea, ogni 2 linee (cioe' le linee dispari)
  '/sh/' tutte le righe che contengono la stringa *'sh'*
  '4,/sh/' dalla quarta alla prima linea che contiene la stringa *'sh'* (inclusa)
- Se viene specificato un singolo indirizzo, l'azione viene svolta sulle linee che fanno il *match* con quella specifica di indirizzo; se viene specificato un intervallo, l'azione viene svolta su tutte le linee incluse nell'intervallo (estremi inclusi)
- Se non viene specificato un indirizzo o un intervallo di indirizzi di linea su cui eseguire l'azione, quest'ultima viene applicata a tutte le linee in input.
- Se vi è più di un'azione, esse possono essere specificate sulla riga di comando precedendo ognuna con l'opzione **-e**, oppure possono essere lette da un file esterno specificato sulla linea di comando con l'opzione **-f**

Esempi d'uso di sed

- **`> sed '4,$d' /etc/passwd`**
  stampa a video soltanto le prime 3 righe del file **`/etc/passwd`**:
  d è il comando di cancellazione che elimina dall'output tutte le righe a partire dalla quarta (si noti l'uso del quoting per impedire che la shell interpreti il metacarattere $).

- **`> sed 3q /etc/passwd`**
  stesso effetto del precedente comando: in questo caso **sed** esce dopo aver elaborato la terza riga (**3q**).

- **`> sed /sh/y/:0/_%/ /etc/passwd`**
  sostituisce in tutte le righe che contengono la stringa **sh** il carattere **:** con il carattere **_** ed il carattere **0** con il carattere **%**.

- **`> sed '/sh/!y/:0/_%/' /etc/passwd`**
  sostituisce in tutte le righe che non contengono la stringa **sh** il carattere **:** con il carattere **_** ed il carattere **0** con il carattere **%** (si noti l'uso del quoting per impedire che la shell interpreti il metacarattere **!**).

Sostituzione del testo con sed

- Il formato dell'azione di sostituzione in **sed** è il seguente
  s/expr/new/flags
  dove:
  expr è l'espressione da cercare,
  new è la stringa da sostituire al posto di **expr**,
  flags può assumere una delle seguenti forme:
  - un numero da 0 a 9 che specifica quale occorrenza di expr deve essere sostituita (di default è la prima),
  - **g**: (global) ogni occorrenza di **expr** viene sostituita,
  - **p**: (print) la linea corrente viene stampata sullo standard output nel caso vi sia stata una sostituzione,
  - **w *file***: (write) la linea corrente viene accodata nel file nel caso vi sia stata una sostituzione.

Esempi di sostituzioni con sed

- **`sed '/^root/,/^bin/s/:.............:/::/w disabled.txt' /etc/passwd`**
  Nelle righe del file **`/etc/passwd`** comprese fra quella che inizia con **`root`** e quella che inizia con **`bin,`** sostituisce la password criptata (lunga 13 caratteri) con la stringa vuota; tali righe sono poi accodate nel file *disabled.txt*.

- **`cat /etc/passwd | sed 's?/bin/.*sh$?/usr/local&?'`**
  Cerca tutte le righe in input in cui compare la stringa corrispondente all'espressione regolare **`/bin/.*sh$`** (ad esempio **`/bin/bash`**) e sostituisce quest'ultima con la stringa corrispondente a **`/usr/local/bin/.*sh$`** (ad esempio **`/usr/local/bin/bash`**).
  Si noti che, siccome il carattere separatore di sed compare nella stringa da cercare, si è usato il carattere **`?`** come separatore.
  Inoltre il carattere **`&`** viene rimpiazzato automaticamente da **`sed`** con la stringa cercata (corrispondente a **`/bin/.*sh$`**).

Esercizi con sed e vi

- Ripetere gli esempi di sostituzione e ricerca con **`sed`** aprendo con **`vi`** il file di input e utilizzando i comandi di **`vi`**
- Utilizzando l'help di **`vi`**, verificare quali sono le opzioni dei comandi di sostituzione e ricerca
- Utilizzando **`sed`** sostituire nel file di input tutte le cifre da 0 a 9 con -