



Lezione 1 - Unix: Definizioni e Storia



Def: Sistema operativo

Collezione di programmi per il coordinamento
delle attività e delle componenti funzionali
di un calcolatore

- ◇ Le attività possono venir generate a seguito dell'interazione operatore/elaboratore





Collezione di programmi

- Kernel o Nucleo del S.O.
- Driver
- Gestore degli accessi
- Gestore dei protocolli di rete
- Programmi di utilità
- Compilatori
- Librerie
- ...



Attività funzionali

- elaborazione numerica
- stampe e salvataggi
- scrittura di applicativi
- interazione con applicativi
- attività ludica
- ...

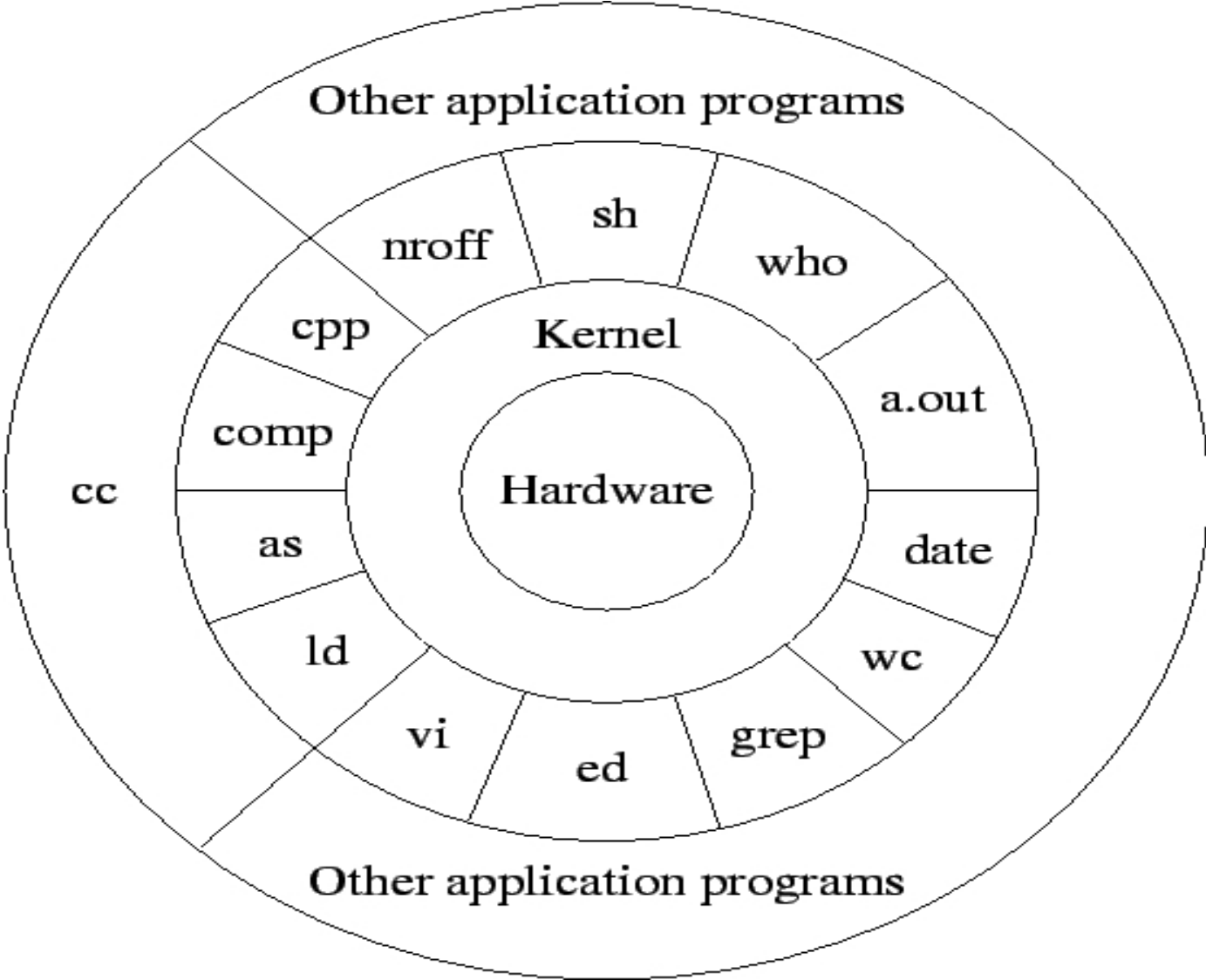




Coordinamento

- Gestione del tempo di processore
- Condivisione d'accesso
- Mutua esclusione
- Serializzazione
- Controllo permessi
- ...





Unix - storia

Nasce negli anni '70 alla Bell Labs (Thompson, Ritchie)
come sistema proprietario in contrapposizione
alla politica di
MULTICS

(Multiplexed Information and Computing Service)

Gira inizialmente di PDP-7 (assembler)
Poi su PDP-11 viene riscritto in C (1973) da
Dennis Ritchie



Unix - storia (2)

In seguito ad un accordo del 1953 AT&T non poteva commercializzare prodotti informatici



Unix viene offerto gratuitamente (no support)



Viene portato su altre architetture



Unix - storia (3)

AT&T nel 1983 estende Unix



System V

Alla Berkeley Unix viene portato su VAX



Berkeley System Distribution



Unix - inquadramento

Sistema operativo:

- multitasking grazie a:
 - time sharing
 - contest switching
- multiuser con:
 - superuser
 - utenti
 - gruppi





Unix - inquadramento (2)

- portabile
 - scritto in linguaggio C
 - implementa una serie di funzioni standard
- con funzionalità di rete
 - client/server
 - peer2peer
 - 3-Tier





Progetto POSIX

Portable Operating System Standard for Computer Environment

- Standard IEEE
- Detta i requisiti di compatibilita con lo standard Unix
- si suddivide in:
 - POSIX.1 (*IEEE 1003.1*) ⇒ Kernel
 - POSIX.2 (*IEEE 1003.2*) ⇒ Interfaccia, utilities
 - POSIX.3 (*IEEE 1003.3*) ⇒ Tools di valutazione



Linux & Open Source

progetto GNU (GNU is NOT UNIX)

FSF: 1985 - RICHARD STALLMAN (MIT)

progetto Linux

LINUX ON MINIX: 1991 - LINUS TORVALD

progetto Minix

MINIX: PROF. TANENBAUM

progetto GPL (GNU Public License)

GNU/LINUX + DEBIAN



Il kernel

Gestisce:

- accesso al filesystem
- scheduling
- accesso ai protocolli di rete

Fornisce:

- API per l'accesso ai dispositivi HW
- virtualizzazione della memoria
- strumenti di mutua esclusione





Il filesystem Unix

Il *filesystem* è il complesso sistema che i sistemi operativi moderni utilizzano per organizzare le entità di memorizzazione permanente

Poiché in Unix ogni cosa è un file il *filesystem* è fondamentale

La struttura logica del FS Unix è un albero

La struttura fisica del FS Unix è fatta di *blocchi* e *inode*

Il filesystem Unix inizia da *root* (/)





File ordinari

Rientrano in questa categoria l'80% dei file di uso comune. Possono avere un nome lungo fino a 255 caratteri, che può iniziare e contenere qualunque lettera o numero, degli spazi, i segni di interpunzione.

```
[oratio:~] andrea% .,asdf. sdfjk,234_ad:
```

è un nome di file valido





File ordinari (2)

Bisogna stare attenti solo al fatto che:

- il primo carattere non sia un .
- il primo carattere non sia un -
- il nome non contenga il ; \$
- il nome non contenga |, &





File ordinari (3)

Si possono creare con:

- touch *nomefile*
- attraverso ridirezione dell'output
- con un editor (VI, Emacs)

Si riconoscono listando la directory con `ls -l`

```
drwxr-xr-x  42 andrea  staff           1428 Nov  4 21:19 ex-Progetti-oratio
drwxr-xr-x  29 andrea  staff             986 Nov  4 21:19 ex-home-oratio
-rw-r--r--   1 root    staff    150266148 Sep 26 18:10 ex-root-oratio.tgz
-rw-r--r--   1 andrea  staff     37326 Oct 17 09:29 httpd.conf
```





Directory

Hanno la stessa caratteristica dei file, ma il loro nome finisce sempre con *slash* (/).

Ovviamente possono contenere file o directory

Si riconoscono listando la directory con `ls -l`

```
drwxr-xr-x  42 andrea  staff           1428 Nov  4 21:19 ex-Progetti-oratio
drwxr-xr-x  29 andrea  staff            986 Nov  4 21:19 ex-home-oratio
-rw-r--r--   1 root    staff    150266148 Sep 26 18:10 ex-root-oratio.tgz
-rw-r--r--   1 andrea  staff     37326 Oct 17 09:29 httpd.conf
```





File nascosti?

Tutti i file o le directory che iniziano con il punto non sono visibili con il comando `ls` a meno di non usare il parametro `-a`

```
[oratio:~] andrea% ls -la
```

```
drwxr-xr-x  42 andrea  staff      1428 Nov  4 21:19 .
drwxr-xr-x  29 andrea  staff       986 Nov  4 21:19 ..
-rw-r--r--   1 root    staff    150266148 Sep 26 18:10 .ex-root-oratio.tgz
-rw-r--r--   1 andrea  staff     37326 Oct 17 09:29 .httpd.conf
-rw-r--r--   1 andrea  staff     37326 Oct 17 09:29 .bashrc
```





File speciali - mount point

Non abbiamo degli identificatori standard come in DOS per i dispositivi di memorizzazione

Possiamo far corrispondere i dispositivi di memorizzazione con delle directory

e.g.

Floppy disk 1 (A:) \Rightarrow /mnt/floppy

Partizione primaria del primo HD (C:) \Rightarrow /

Partizione secondaria del primo HD (D:) \Rightarrow /home





File speciali - mount point (2)

Alcuni mount point sono previsti dal sistema:

- scritti nel file `/etc/fstab`
- verificabili attraverso il comando `mount`

Avendo opportuni privilegi se ne possono montare di propri

- dischetti \Rightarrow `/dev/fdn`
- partizioni \Rightarrow `/dev/hdcn`
- immagini ISO \Rightarrow `/dev/loopn`



File speciali - altri

Altri tipi di file:

- socket
- pipe
- link
- devices





Le directory importanti

Se ci si fa un giretto fuori dalla propria home si scopre un mondo vastissimo

Applications	Encode	automount	resources
Applications (Mac OS 9)	Encode.pm	bin	rr
Cartella Sistema	Library	cores	sbin
Desktop	Network	dev	sw
Desktop (Mac OS 9)	Previous Systems	etc	temporaneo
Desktop DB	System	mach	tmp
Desktop DF	Temporary Items	mach.sym	usr
Desktop Folder	TheVolumeSettingsFolder	mach_kernel	var
Developer	Trash	opt	
Documenti	Users	private	
Elimina al Riavvio	Volumes	res	





Le directory importanti

Lo standard POSIX prevede la presenza delle seguenti directory di sistema

boot	bin	dev
etc	home	lib
opt	sbin	usr
var	tmp	
usr/local		var/log



Il contenuto di etc

Contiene:

- file di configurazione (rete, sicurezza, servizi)
- script e directory di runlevel
- file di password e gruppi
- file e directory di crontab
- file di default





Il contenuto di bin

Contiene i comandi di base del sistema:

- cp, mv, ls, rm,...
- echo, cat, ...
- kill, stat,...
- le shell



Il contenuto di sbin

Contiene i comandi specifici di pseudo-amministrazione:

- utility per la rete (setup, probe)
- utility per i dischi
- utility per il logging
- utility per il boot





Il contenuto di opt

Directory di nuova fattura:

- tende a sostituire `usr/local`
- ci vanno i programmi che nella loro folder hanno una gestione non conforme allo standard per eseguibili, configurazioni e manuali
- Java, ...



Il contenuto di usr

Directory storica di Unix, contiene:

- il sistema X11
- le utility evolute per la configurazione di:
 - rete
 - stampanti
 - dischi
 - librerie
 - utenti
 - sicurezza





Il contenuto di `usr/local`

Directory repository per i programmi locali

All'inizio Unix fu pensato con filesystem distribuito/remoto

Questa directory era il punto di montaggio dei dispositivi locali alla macchina

Ha in sè tutta la struttura di `/` (root)



Il contenuto di dev

Directory dei *dispositivi virtuali*:

- dischi e partizioni
- porte seriali, parallele, ausiliari, ...
- frame-buffer, video, ram, ...
- terminali seriali e virtuali
- standard input, output e error
- zero, null e random





Il contenuto di mnt

Directory che contiene i mount-point dei dispositivi removibili:

- floppy
- cdrom
- secure digital & flash card
- partizioni DOS/Windows :-)





non ci perdiamo... (pwd)

Il *filesystem* Unix è molto vasto, ma con `pwd` possiamo sempre conoscere il path attuale

Comando:

```
[Computer] andrea% pwd  
/System/Library/StartupItems/Portmap/Resources
```

Attenzione ai link simbolici poiché vengono risolti !!





...path della propria homedir

Viene deciso dall'amministratore durante la creazione dell'utenza insieme con la shell di default

```
root:*:0:0:System Administrator:/var/root:/bin/tcsh
```

Il default è /home/<nomeutente>

Si può raggiungere attraverso cd, ~ o \$HOME





Il comando mkdir

Come in DOS server per creare una DIRECTORY

```
[Computer] andrea% mkdir Resources/Tezzuia  
/System/Library/StartupItems/Portmap/Resources
```

Attenzione a guardar bene dove siamo!!





Il comando `rm -r`

Server per cancellare *ricorsivamente* il contenuto di una DIRECTORY

Comando:

```
[Computer] andrea% rm -r Portmap  
/System/Library/StartupItems/Portmap/Resources
```

Attenzione al PATH, possiamo cancellare TUTTO!!





Il comando `ls -l`

Server per listare la DIRECTORY mostrandi tutti gli attributi

Non visualizza i file . (punto)

Per visualizzare i file speciali ci mettiamo il parametro `-a`

Comando:

```
[Computer] andrea% ls -l Portmap  
/System/Library/StartupItems/Portmap/Resources
```





Il comando `ls -l` (2)

Server per listare la DIRECTORY mostrandi tutti gli attributi

```
total 1116
-rw-r--r-- 1 andrea staff    33783 Nov  4 09:14 back.jpg
-rw-r--r-- 1 andrea staff   36334 Nov  4 09:14 back2.jpg
-rw-r--r-- 1 andrea staff   10933 Nov  4 09:14 bg.jpg
-rw-r--r-- 1 andrea staff   42173 Nov  4 09:14 d12.jpg
-rw-r--r-- 1 andrea staff   11161 Nov  4 10:49 img4.gif
-rw-r--r-- 1 andrea staff  116144 Nov  4 10:54 img4.jpg
-rw-r--r-- 1 andrea staff    5691 Nov 11 22:03 slide.out
-rw-r--r-- 1 andrea staff  663403 Nov 11 22:03 slide.pdf
-rw-r--r-- 1 andrea wheel   29397 Nov 11 22:05 slide.tex
-rw-r--r-- 1 andrea wheel   29167 Nov 11 22:03 slide.tex
```



Il comando `ln`

Crea dei link ai file

Due tipi di link:

- hard link - (ogni file è un hard link)
- soft link - (puntatori, il kernel sostituisce la destinazione)

Opzioni:

`-s` : link simbolici





Il comando touch

Cambia la data di accesso e modifica di un file

```
[Computer] andrea% touch Tezzuia.file  
[Computer] andrea% ls -l Tezzuia.file  
-rw-r--r-- 1 andrea staff 0 Nov 11 22:22 Tezzuia.file
```

Riproviamo dopo 2 minuti !!





Il comando chmod

Server per cambiare gli attributi di un file o directory

```
0400    read by owner
0200    write by owner
0100    execute (or search for directories) by owner
0070    read, write, execute/search by group
0007    read, write, execute/search by others
```

```
r = read , w = write , x = execute
```

```
u = user , g = group , o = other , a = all
```





Il comando chown

Server per cambiare i proprietari di un file o directory

Comando:

```
[Computer] andrea% chown [-R] <user>.<group> <file>
```

e.g.

```
[Computer] andrea% chown -R nobody.nobody file
```



Il comando cp -ra

Guardare il man!!



Il comando mv

Guardare il man!!



Il comando mount

Guardare il man!!





Il comando `wc`

Riporta in standard output il numero di *parole*,
byte, *linee* che compongono il file

Opzioni:

- l : numero di linee
- c : numero di caratteri
- w : numero di parole



Il comando file

Restituisce il tipo di file

Il nome dei file può contenere qualsiasi carattere (anche il punto), non esiste il concetto di *estensione* per identificare il tipo di file.

Il comando file confronta il contenuto del file con un database (*magic.mime*) e restituisce il tipo di dati contenuti nel file.





Il comando find

Serve per ricercare un file all'interno del filesystem

Realizza la ricerca discendendo ricorsivamente le directory a partire da quella indicata come parametro.

Individua in questa passeggiata i file che soddisfano l'espressione immessa come parametro

L'espressione è formata da *PRIMARIES* ed *OPERANDI*





Il comando find (2)

PRIMARIES importanti:

- -delete : cancella tutti i file trovati
- -empty : ci dice se il file attuale vuoto
- -iname <pattern> : pattern non case-sensitive
- -links <n> : ritorna i file che hanno n links
- -ls : riporta l'output in formato del comando ls -l
- -nogroup : riporta i file che non appartengono a gruppi conosciuti
- -nouser : riporta i file che non appartengono a utenti conosciuti
- -perm <nnnn> : riporta i file che hanno il bit dei permessi <nnnn>





Il comando cat

Non serve solo a visualizzare, ma anche a *concatenare* i file

e.g.

```
bash-2.05a$ cat ilfile sefile > dufile
```

Concatena il contenuto di *ilfile* e di *sefile* e scrive il risultato in *dufile*

e.g.2

```
bash-2.05a$ cat ilfile sefile >> dufile
```

Concatena il contenuto di *ilfile* e di *sefile* e appende il risultato alla fine di *dufile*



Il comando cat (2)

Le opzioni più interessanti sono:

- -n: numera le linee
- -b: non numera le linee vuote
- -s: toglie le linee vuote consecutive
- -v: visualizza i caratteri non stampabili
- -t: visualizza i caratteri TAB come CTRL-I





Comandi `more` e `less`

Vengono utilizzati quando l'output supera l'ampiezza della zona di visualizzazione

Possono essere utilizzati in *pipe all'output...*

```
bash-2.05a$ cat ilfile dufile | less
```

... o con parametro il nome di un file

```
bash-2.05a$ more ilfile
```



Comandi head e tail

Vengono utilizzati per selezionare parti del file

head visualizza le prime 10 righe del file
(*parametro*) o dell'output (*pipe*)

tail visualizza le ultime 10 righe del file
(*parametro*) o dell'output (*pipe*)



Comandi head e tail (2)

Opzioni di head:

- -n: decide il numero di linee da visualizzare



Comandi head e tail (3)

Opzioni importanti di tail:

- -f: non esce dalla visualizzazione aspettando ulteriori linee
- -n: visualizza le ultime n righe
- -n +: visualizza a partire dall'n-esima riga
- -b: visualizza gli ultimi n blocchi da 512 bytes
- -c: visualizza gli ultimi n bytes
- -r: utilizza l'ordine inverso di visualizzazione





Il multitasking - time sharing

Metodo di ripartizione del tempo di elaborazione in cui un componente detto *scheduler* si occupa di attivare i processi in base alla priorità delle *code di attesa* in cui essi sono inseriti

Si differenzia dal *multitasking cooperativo* per il fatto che non sono i processi ad avere il controllo sul tempo di esecuzione a loro disposizione, ma esiste un controllore centralizzato per tutto il sistema





Concetto di processo

Un'entità di elaborazione unitaria, generato da un *programma* (eventualmente compilato), ovvero da un metodo simbolico con cui vengono espressi gli algoritmi che un processore deve eseguire per elaborare dei risultati a partire da alcuni dati iniziali.

In Unix ogni processo è di proprietà di un singolo utente (colui che lo ha *lanciato*), ha una priorità, una sua area di memoria assegnata e può eventualmente scambiare messaggi e dati con altri processi





Concetto di thread

Unità logica di elaborazione all'interno di un processo (e.g. stampa, calcolo, accesso alla rete, etc..)

Ogni thread condivide con gli altri (dello stesso processo) l'area di memoria assegnata e non è quindi necessario avviare procedure di scambio messaggi con memoria condivisa fra processi

Viene evitato il *contest switching*





Processi di sistema

Esistono particolari tipi di processi che sono attivati dal sistema all'avvio e vivono per tutto il tempo in cui è attivo il sistema detti *demoni*

Gestiscono:

- servizi di rete
- azioni automatizzate e temporizzate
- controlli sullo stato del sistema
- accessi locali





Processi utente

Fanno parte di questa classe tutti i processi che derivano dall'invocazione di comandi dalla shell e tutti quelli attivati da particolari condizioni (cron e post-login/pre-logout)

Ogni utente può:

- sospendere
- uccidere :-(
● osservare
- riavviare





Il comando ps

Un processo è identificato dal suo nome e dal suo *Process-ID* (PID)

Attraverso il comando ps è possibile visualizzare lo stato dei processi

Opzioni:

- -a: mostra tutti i processi
- -c: mostra solo il comando e non il path
- -u: mostra informazioni sull'utente proprietario del processo
- -x: non mostra le informazioni sulle linee di terminale



Il comando top

Mostra le statistiche sull'uso del sistema

Nelle statistiche è previsto lo stato di ogni singolo processo

I processi sono ordinati per quantità di tempo di CPU occupato

Schiacciando la M il sistema ordina la visualizzazione per quantità di memoria occupata





I segnali

È possibile inviare dei *segnali* ad un processo
Attraverso la tastiera:

- CTRL-C : KILL
- CTRL-Z : STOP

Attraverso il comando `kill`:

- -9 o -KILL
- -2 o -INT
- -1 o -HUP





Foreground & Background

Sono le due modalità di esecuzione in cui si può trovare un processo

I processi che sono interattivi non possono essere mandati in bg

È possibile cambiare la modalità di esecuzione con il comando bg dopo aver inviato un segnale di sospensione ad un processo (CTRL-Z) e ritornare alla modalità interattiva attraverso fg



Il comando jobs

Il comando permette di visualizzare quali siano i processi in esecuzione bg o sospesi



Il comando fg



68/136



Il comando bg



69/136



Il comando &





Il comando `kill`

Invia un segnale ad un processo attraverso il suo
PID

Prima ricerco il PID del processo a cui voglio
inviare il segnale (`ps`), poi:

```
kill -9 <PID>
```





Il comando `killall`

Invia un segnale ai processi attraverso il loro
NOME

Nel nome del processo non va incluso il PATH:

```
killall -9 <Process-NAME>
```





Il filesystem /proc

Tutto è un file, quindi anche l'ambiente e le opzioni dei processi devono poter essere acceduti come file

Il filesystem speciale /proc contiene una directory per ogni processo (identificata dal PID): in ogni directory:

- cmdline
- cwd
- environ





Il filesystem /proc (2)

- exe
- fd/
- maps
- mem
- mounts
- root
- stat e statm
- status





Il filesystem /proc (3)

In Linux il filesystem proc contiene anche alcuni file ed alcune directory speciali che permettono di controllare a *run-time* il funzionamento del kernel.

Per esempio:

- /proc/cpuinfo
- /proc/sys/net/ipv4/ip_forward
- /proc/sys/net/kernel/modprobe

Le opzioni possono anche essere manipolate con `sysctl`



Moduli

Anche nei kernel monolitici come quello di Unix e di Linux alcuni componenti possono essere inseriti e rimossi durante il funzionamento.

In questo modo è possibile risparmiare memoria dedicata al kernel senza doverlo ricompilare al sorgere di una nuova esigenza.

Per motivi legati alla gestione di memoria, però, l'uso di moduli riduce leggermente le prestazioni del sistema.



Moduli (2)

Alcuni comandi per gestire i moduli

- insmod
- modprobe
- rmmod
- modinfo

Nei kernel recenti è stata aggiunta la possibilità di gestire automaticamente i moduli attraverso il file `/etc/modules.conf` oppure l'utility `hotplug`.



La Shell

- assomiglia al prompt del DOS :-)
- stabilisce un'interfaccia fra Utente e S.O.
 - interpreta i comandi immessi da tastiera
 - restituisce il valore di ritorno delle applicazioni
- fornisce un linguaggio di programmazione





Diverse esigenze \Rightarrow diverse shell

- Bourne Shell (Steve Bourne - MIT)
- Korn Shell (David Korn - Bell Labs)
- C-Shell (sea-shell)
- a-Shell (ascetic shell)
- tcshell
 - (C-shell free)
- Zshell
 - (Korn shell free)





Regole di base

- Qualsiasi SHELL di Unix è in grado di distinguere le maiuscole dalle minuscole
- I comandi si terminano con un ENTER
- I comandi possono essere più lunghi di una riga
- I parametri dei comandi in Unix sono preceduti dal meno (-)
- I nomi delle variabili d'ambiente sono MAIUSCOLI
- Per ottenere il valore delle variabili d'ambiente si usa il prefisso \$
- Negli script di shell se si vuole proseguire oltre una riga il comando è buona norma mettere un backslash alla fine di ogni riga (\)



Comandi

Comando DOS	Comando Unix
ATTRIB (set)	chmod
ATTRIB (get)	ls -l
DIR	ls
COPY	cp
MOVE	mv
XCOPY	cp -r
MORE	less more
CHKDSK (stat)	df



Comandi (2)

DELTREE	rm -r
DEL	rm
FIND	grep
SORT	sort
DATE (get)	date
DATE (set)	date -s
TIME (get)	date
TIME (set)	date -s
CLS	clear reset
TYPE	cat
VER	uname -a



Comandi (3)

MD	mkdir
CD (set)	cd
CD (get)	pwd
RD	rm -ry
Comandi di Help	man & apropos



grep

Il comando `grep` stampa le linee che soddisfano il *pattern*

Comando:

```
grep chino
```

inserire delle parole e vedere se si riesce a farle stampare
dare CTRL-C per terminare



grep (2)

Opzioni importanti:

- `-i`: non fa differenza fra maiuscole e minuscole
- `-r`: ricerca ricorsivamente in [sotto]directory
- `-l`: da solo il nome dell'entità in cui si è trovato il pattern
- `-v`: inverte il match (NOT pattern)
- `-c`: conta le linee che corrispondono il pattern





Esempio: seleziono le directory

Voglio selezionare tutti i file che hanno nel loro nome il carattere “/”

metto in *pipe* (*concateno*)
l'*output* di `ls`
con
l'*input* di `grep`

Comando:

```
bash-2.05a$ ls -F | grep /
```





Input e output

Unix ha a disposizione 4 operatori standard:

- | (pipe o concatenamento)
- > (output verso file)
- < (input da file)
- >> (output in aggiunta ad un file)





Esempio - Redirezione verso un file

Alla fine dell'esercizio dobbiamo avere la lista delle subdirectory in un file

Comando:

```
bash-2.05a$ ls -F | grep / > ilfile
```

Come possiamo sapere se il file contiene quello che volevamo?

Nel frattempo mettiamo in un altro file tutto quello che sta nella home tranne le sottodirectory





Soluzione

Utilizziamo il comando `cat` per visualizzare *ilfile*

Comando:

```
bash-2.05a$ cat ilfile
```

Utilizziamo il parametro `-v` di `grep` per invertire la selezione

Comando:

```
bash-2.05a$ ls -F | grep -v / > sefile
```



Prompt della shell

Quando una shell attende ed esegue i comandi in maniera interattiva.

La disponibilità a ricevere comandi viene evidenziata da un messaggio di invito o *prompt*.

Prompt composto da simboli e informazioni utili all'utente per tenere d'occhio il contesto in cui sta operando.



Esempio prompt

```
[luigi@lg600 luigi]$ pwd
/home/luigi
[luigi@lg600 luigi]$ cd /tmp
[luigi@lg600 /tmp]$ echo $PS1
[\u@\h \W]\$
[luigi@lg600 /tmp]$ whoami
luigi
[luigi@lg600 /tmp]$ hostname
lg600.gangitano.it
[luigi@lg600 /tmp]$ pwd
/tmp
```



Storico dei comandi

Bash mantiene uno storico dei comandi utilizzati dall'utente.

Lo storico NON si interrompe alla fine di una sessione o con un reboot (é su file).

Freccette (sù,giù) per sfogliarlo

CTRL-R+[qualche carattere] per cercare all'interno dello storico

CTRL-R cerca ancora



Altri tasti utili

Andare all'inizio della riga - CTRL-A

Andare alla fine della riga - CTRL-E

CANC - CTRL-D

(spesso CANC é mappato male)



Alias

Bash permette la definizione di nuovi comandi in forma di alias di altri comandi (o serie di comandi).

alias - elenca gli alias esistenti

alias nome="comando con tutti i parametri"

alias dir=ls -color





Serie di comandi

É possibile dare più comandi da eseguire in serie separando gli elementi della lista con dei ;

Esempio

```
$ mv pippo.ps pluto.ps;mv pluto.ps pippo.ps
```

Equivale a:

```
$ mv pippo.ps pluto.ps
```

```
$ mv pluto.ps pippo.ps
```



Ambiente

Ogni programma in funzione nel sistema ha un proprio **ambiente** definito in base a delle **variabili di ambiente**.





Variabili d'Ambiente

Le **variabili d'ambiente** sono un mezzo elementare e pratico di configurazione del sistema: i programmi, a seconda dei loro compiti e del loro contesto, leggono alcune variabili, e in base al contenuto si comportano.

Esempio:
la variabile PATH



PATH

La variabile PATH indica tutte le directory dove la shell deve andare a guardare per trovare dei file eseguibili (i programmi per capirci)

É una variabile a tutti gli effetti:

```
$ echo $PATH  
/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr  
/X11R6/bin:/usr/games:/home/luigi/bin:/usr/sbin  
:/usr/X11R6/bin:/usr/games
```





PATH e la directory corrente

La directory corrente normalmente **NON** viene inclusa nella variabile PATH

Non é molto sicuro, e non dovrebbe mai essere “prima” di quelle canoniche.

Si può aggiungere in fondo a PATH, o ovviare così:

```
./nome_file_eseguibile
```



Ambiente (2)

L'ambiente consegnato a ogni programma che viene messo in esecuzione, controllato dalla shell che può assegnare ambienti diversi a programmi diversi.

La shell pu creare, modificare e leggere queste variabili, e utile per la realizzare file script.





Gestione Variabili in BASH

Variabili di shell

Una variabile definita quando contiene un valore, anche se vuoto.

L'assegnamento di un valore si ottiene con una dichiarazione del tipo:

```
nome_di_variabile=[valore]
```

Esempio:

```
[luigi@lg600 luigi]$ pluto=cane_di_topolino  
[luigi@lg600 luigi]$ echo $pluto  
cane_di_topolino
```





Esportazione Variabili

Le variabili di shell hanno validità limitata all'ambito della shell stessa.

I comandi interni alla shell stessa sono al corrente di queste variazioni mentre i programmi che vengono avviati non ne risentono.

Per i programmi esterni le variabili devono essere **esportate**. L'esportazione delle variabili si ottiene con il comando interno `export`.





Esportazione Variabili (2)

Esempi

```
$ PIPPO="ciao"  
$ export PIPPO
```

Crea la variabile PIPPO e quindi la esporta.

```
$ export PIPPO="ciao"
```

In un colpo solo.



Espansione/Sostituzione

Come già si intuiva da alcuni comandi che abbiamo visto la shell, prima di lanciare i programmi e/o comandi fa delle modifiche alla stringa che noi gli passiamo, meccanismo noto come **espansione**





Passi dell'espansione

1. parentesi graffe;
2. tilde ;
3. parametri e variabili;
4. comandi;
5. aritmetica (da sinistra a destra);
6. suddivisione delle parole;
7. percorso o pathname.





Espansione della tilde

Se una parola inizia con il simbolo tilde (`~`) interpreta quello che segue, fino alla prima barra obliqua (`/`), come un nome-utente.

È sostituisce questa prima parte con il nome della directory personale dell'utente stesso.

`~` da sola sottointende l'utente attuale

Esempi

```
$ cd ~
```

```
$ cd ~tizio/public_html
```





Espansione di parametri e variabili

Già visto
sostituzione del parametro o della variabile con il suo contenuto.

Esempio

```
[luigi@lg600 luigi]$ pippo=PIPP0  
[luigi@lg600 luigi]$ echo $pippo  
PIPP0
```





Sostituzione comandi

La sostituzione dei comandi consente di utilizzare quanto emesso attraverso lo standard output da un comando. Ci sono due forme possibili:

`$(comando)`

‘ comando ‘

Apici “al contrario” `altgr+apice`

Per ora Standard Output = testo riportato come risultato dal comando





Espansione di percorso

I simboli *, ? e [vengono interpretati ed espansi in serie di stringhe (che indicano percorsi)

Vediamo solo il più importante: *

Corrisponde ad una qualsiasi stringa
quindi se siamo in una directory con questi file:

```
paperino paperone paperoga minnie topolino
```

```
rm -f pape* -> rm -f paperino paperone paperoga
```





Caratteri di protezione

A volte magari abbiamo bisogno di scrivere effettivamente caratteri come:

\$ * { [... }

Esistono 3 modalità:

1. Escape
2. Apici singoli
3. Apici doppi





Escape

La barra obliqua inversa (`\`) rappresenta il carattere di escape. Serve per preservare il significato letterale del carattere successivo, cioè evitare che venga interpretato diversamente da quello che veramente. Esempio

```
ls -l \ $fff
```

3B cerca di leggere il file che si chiama proprio `$fff`





Escape (2)

Usato da solo (quindi con uno spazio dopo) serve per continuare una linea (nonostante si vada a capo) Esempio

```
$ ls -l \  
supercalifragilischespiralidoso
```

viene interpretato come un unico comando su una linea





Apici Singoli

Racchiudendo dei caratteri tra apici semplici (') si mantiene il valore letterale di questi caratteri. Un apice singolo non pu essere contenuto in una stringa del genere. Esempio

```
$ echo '$0 $1 \ ... restano "inalterati".'
```

```
$0 $1 \ ... restano "inalterati".
```





Apici Doppi

Racchiudendo una serie di caratteri tra una coppia di apici doppi si mantiene il valore letterale di questi caratteri, a eccezione di \$, ' e \. Esempio

```
$ echo "Il parametro \$0 contiene: \"$0\""  
Il parametro $0 contiene: "-bash"
```





Espressioni regolari

A (modern) RE is one or more non-empty branches, separated by ' | '.
It matches anything that matches one of the branches.

A branch is one or more pieces, concatenated. It matches a match for the first, followed by a match for the second, etc.

A piece is an atom possibly followed by a single '*', '+', '?', or bound.
An atom followed by '*' matches a sequence of 0 or more matches of the atom. An atom followed by '+' matches a sequence of 1 or more matches of the atom. An atom followed by '?' matches a sequence of 0 or 1 matches of the atom.





Espressioni regolari (2)

A bound is ' {' followed by an unsigned decimal integer, possibly followed by ',' possibly followed by another unsigned decimal integer, always followed by ' }'. The integers must lie between 0 and RE_DUP_MAX (255) inclusive, and if there are two of them, the first may not exceed the second. An atom followed by a bound containing one integer *i* and no comma matches a sequence of exactly *i* matches of the atom. An atom followed by a bound containing one integer *i* and a comma matches a sequence of *i* or more matches of the atom. An atom followed by a bound containing two integers *i* and *j* matches a sequence of *i* through *j* (inclusive) matches of the atom.





Espressioni regolari di Bash

- * ⇒ Corrisponde ad ogni stringa, NULL STRING inclusa.
- ? ⇒ Corrisponde ad ogni singolo carattere.
- [...] ⇒ Corrisponde con uno dei caratteri racchiusi fra parentesi.
- Una coppia di caratteri separati da una tilde denota un intervallo

Apice e punto esclamativo invertono la selezione, se messi come primo carattere dopo [

i caratteri] e — possono essere ricercati mettendoli al primo posto dopo la [





Le classi POSIX

esistono anche delle classi POSIX
(alnum,alpha,digit,space,upper,lower)
che possono venir utilizzate con la sintassi
[[:classe:]]

Spiegazione:

- alnum : caratteri alfanumerici
- alpha : solo i caratteri alfabetici
- digit : solo le cifre
- space : i caratteri di spazio e TAB
- upper : i caratteri MAIUSCOLI





Le opzioni nelle espressioni regolari

Per scegliere fra diversi *token* opzionali si usano le parentesi tonde per delimitare l'insieme e il segno di *pipe* | per separare le opzioni

e.g.

```
ls -Fa | egrep '(a|b)'
```





Le ripetizioni nelle espressioni regolari

Per identificare *token* di lunghezza predefinita (fino a 255 ripetizioni) o intervalli di ripetizione (da n a m volte) si utilizzano le parentesi graffe con un numero, con dei numeri separati da virgole o con due numeri separati dal meno

e.g.

```
ls | egrep 's{3}|+g{3}'
```





Comandi - sed

Comando utile ma molto complesso, ne vediamo solo un uso: sostituzione di stringhe Riga per riga di un file (o dello standard output) sostituisce una stringa con un'altra.

Esempio

abbiamo il file prova.html e vogliamo cambiare alcuni link senza editare il file:

```
$ cat prova.html | \  
    sed 's/<a href="immagini/<a href="images/' > \  
    prova-modificato.html
```





Un editor universale: vi

Per poter creare degli script è necessario avere un editor di testo. Linux dispone di molti editor, ciascuno con le sue peculiarità, ma uno solo lo troverete in tutti i sistemi Unix, e molto spesso è l'unico che avrete a disposizione.

Benvenuti nel magico mondo di *vi*.



Esecuzione

Il comando base per accedere a *vi* è, ovviamente, **vi**, ma nelle distribuzioni più recenti è presente una versione *enhanced*, con molte funzionalità in più, che prende il nome di **vim** (*Vi IMproved*).





Descrizione dell'ambiente

Eseguendo il comando **vi** vi troverete davanti all'ambiente di editing, che appare inizialmente piuttosto scarno. L'area di lavoro è completamente dedicata al testo da elaborare, ad esclusione dell'ultima riga in basso che mantiene, di volta in volta, indicazioni sullo stato in cui si trova l'editor.





Ambienti

vi infatti è stato progettato in modo da distinguere anche concettualmente l'immissione del testo dall'interpretazione di comandi che operano sul testo inserito.

In qualsiasi momento, per poter inserire un comando è sufficiente tornare alla *modalità comandi* premendo il tasto **Esc**.

Inoltre il tasto **Esc** consente di annullare un comando non completato. Premerlo più volte di seguito non ha effetti collaterali.





Apertura di un file

L'esecuzione del comando **vi** senza argomenti, crea un nuovo buffer vuoto.
Se invece vogliamo editare un file esistente possiamo invocare **vi** seguito dal nome del file che vogliamo editare.

```
vi nomefile
```

In alternativa, una volta avviato **vi**, è possibile aprire un file con il comando **:e**

```
:e nomefile
```





Inserimento del testo

Premendo il tasto **i** oppure **Ins** si passa alla modalità di inserimento del testo, in cui è possibile digitare del testo normalmente.

Solitamente questa modalità è identificata dalla presenza della stringa – *INSERT* – nella barra di stato.





Inserimento del testo (2)

```
Testo inserito dopo aver premuti il tasto [i]
```

```
~
```

```
~
```

```
~
```

```
~
```

```
-- INSERT --
```

È da notare che il carattere `~` (tilde) identifica le linee vuote.





Modifica del testo

La pressione del tasto **r** attiva la modalità modifica del testo in cui il testo inserito sovrascrive il testo esistente.

L'attivazione di tale modalità è identificata dalla presenza della stringa – *REPLACE* – nella barra di stato.

Lo stesso risultato si ottiene premendo due volte il tasto **Ins**.





Cancellazione del testo

Per cancellare del testo è possibile utilizzare il comando **x** che cancella il carattere identificato dal cursore, oppure il comando **dd** preceduto da un numero *n* il quale, appunto, cancella *n* righe a partire da quella occupata dal cursore.

```
Testo inserito nelle slide precedenti  
continua sulla seconda riga  
e sulla terza  
~
```

L'esecuzione del comando **2 dd** quando il cursore si trova sulla prima riga cancellerà le prime due righe e lascerà la terza.





Copia ed incolla

Il comando **yy**, eventualmente preceduto da un numero, copia in un buffer n righe a partire da quella in cui si trova il cursore.

La funzione “Taglia” in realtà l’abbiamo già vista, dato che viene svolta dal comando **dd** che non si limita a cancellare una riga ma la copia in un buffer.

Per incollare si usano i comandi **p** oppure **P**. Il primo incolla a partire dalla riga seguente a quella in cui si trova il cursore, il secondo a partire da quella in cui si trova il cursore.





Ricerche nel testo

Per cercare una stringa all'interno di un file di testo si usa il comando `/`. Tra l'altro questo comando svolge la stessa funzione in molte situazioni (all'interno di `man`, `less`, `more`, ecc.).

Dopo la pressione del tasto `/` è possibile digitare la stringa da cercare che apparirà nella barra di stato.

Per ripetere l'ultima ricerca effettuata si usa il tasto `n`.





Salvataggio e chiusura

Per salvare il testo editato, si usa il comando **:w**, mentre per uscire dal programma si usa il comando **:q**.

I due comandi possono esseri combinati (**:wq**) per salvare ed uscire nello stesso momento.

Inoltre e' possibile forzare l'uscita senza salvare (con conseguente perdita delle modifiche), con il comando **:q!**.



Riepilogo (1)

- **vi** *nomefile*, apre il file in vi
- **Esc** torna alla modalità comandi
- **:e** (*edit*) apre un file
- **i** o **Ins** inserisce del testo
- **r** o due volte *Ins* sovrascrive il testo
- **[n] dd** taglia *n* righe
- **[n] yy** copia *n* righe
- **p** incolla dopo la riga corrente
- **P** incolla a partire dalla riga corrente





Riepilogo (2)

- / cerca all'interno del testo
- n ripete l'ultima ricerca
- :w salva
- :q chiude
- :q! chiude senza salvare

