

Sistemi Operativi A
A.A. 2004/2005
Breve introduzione alla shell di Unix

Ing. Paolo Torroni
tel. 93767

Linux - 1

argomenti

- introduzione
- shell
- file system
- file
- protezione
- compilatore gcc
- Hello World
- processi
- redirectione e piping
- programmazione shell

Linux - 2

Il sistema operativo Unix (Linux)

introduzione

Linux - 3

Cenni storici

- 1969: primo sviluppato nei Bell Laboratories dell' AT&T. L'obiettivo era quello di realizzare un ambiente di calcolo multiprogrammato e portabile per macchine di medie dimensioni.
- 1970: prima versione di UNIX, interamente sviluppata nel linguaggio assembler del calcolatore PDP-7; questa versione era multiprogrammata e monoutente.
- Anni successivi al 1970: nuove versioni, arricchite con altre caratteristiche e funzionalità. In particolare venne introdotto un supporto alla multiutenza.

Linux - 4

Unix e il C

- 1973: Unix viene realizzato nel linguaggio di programmazione C, recando numerosi vantaggi:
- Elevata portabilità: capacità di fornire lo stesso ambiente di esecuzione e di sviluppo su architetture diverse.
- Leggibilità: la struttura del sistema è facilmente comprensibile e quindi apprezzabile da parte della comunità scientifica e accademica.

Linux - 5

Versioni di Unix

- Negli anni 80 la grande popolarità di Unix ha determinato anche il proliferare di svariate versioni. In particolare, le due *famiglie* di sistemi Unix maggiormente diffuse sono:
- Unix System V (AT&T Unix System Laboratories)
- Unix Berkeley Software Distributions, o BSD (University of California at Berkeley).

Linux - 6

POSIX

- 1988: POSIX (Portable Operating Systems Interface) è lo standard definito dall'IEEE. Definisce le caratteristiche relative alle modalità di utilizzo del sistema operativo.
- 1990: POSIX viene anche riconosciuto dall'*International Standards Organization* (ISO).
- Negli anni seguenti, le versioni successive di Unix SystemV e BSD (versione 4.3), si uniformano a POSIX.

Linux - 7

Linux

- Sviluppo 'a più mani' (internet). Prima versione: 0.02 (1991). Vers. 1.0: 1994
- S.O. multi-utente, multi-processo e multi-thread con le caratteristiche di Unix
- Portabilità: Intel x86, SUN sparc, motorola 68k, PowerPC, Alpha, MIPS.
- È disponibile gratuitamente
- Distribuzioni (assemblaggi di kernel linux + componenti): Slackware, Debian, Red Hat, ...

Linux - 8

www.linux.org

- lista delle FAQ (Frequently Asked Questions: problematiche comuni)
- lista degli HOWTO (risoluzione di problemi particolari, per argomenti)
- Linux Documentation Project (manuali: installazione, amministrazione, ...)
- Applicazioni

Linux - 9

Linux / Unix: la shell

utenti e gruppi, shell, comandi

Linux - 10

Utenti e gruppi

- Sistema multiutente → problemi di privacy (possibili interferenze): necessità di proteggere / nascondere informazione
- Concetto di gruppo (es. staff, users, root, ...): possibilità di lavorare sugli stessi documenti
- Ogni utente appartiene a un gruppo ma può far parte anche di altri a seconda delle esigenze e configurazioni

Linux - 11

Accesso a Linux: *login*

- Per iniziare una sessione bisogna essere in possesso di una *combinazione*:
 - username (es. x135462, d1l28493, ...)
 - password (es. dfh@2#q, **a890, aPP&x., ...)
- nota: maiuscole / minuscole sono caratteri diversi!! (la password **a890 è diversa da **A890)
- Accesso al sistema: login: x135462
 Password: *****

Linux - 12

shell...

- Una volta superata la fase di login, l'utente è collegato al sistema Linux. Di norma è presente una finestra di *shell*
- La shell è un programma che consente di far interagire l'utente col sistema. Resta in attesa di comandi (da digitare con la tastiera), che li manda in esecuzione una volta ricevuto l'<ENTER>

Linux - 13

ciclo della shell di login di Unix

```
<login>
do
{
  <ricevi comando dal file di input>
  <interpreta il comando>
  <esegui comando>
}
while (!EOF);
<logout>
```

Linux - 14

...shell

- interfaccia di alto livello tra utente e S.O.
- processore comandi evoluto: interpreta e mette in esecuzione comandi da:
 - standard input
 - file comandi
- linguaggio comandi con elevato potere espressivo

Linux - 15

Varie shell di Unix

- esistono diverse Shell in Unix:
 - Bourne Shell (standard)
 - C Shell
 - Korn Shell
 - Tc Shell
 - etc
- L'implementazione della Bourne shell sotto Linux si chiama *bash* (Bourne-Again).

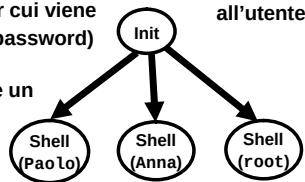
Linux - 16

Lo Shell di Unix: Accesso al Sistema

- un utente può attivare più shell, anche diverse: tcsh, csh, bash, ...

- una shell in particolare è chiamata shell di login (quella per cui viene chiesta inizialmente la password)

- la shell di login fornisce un accesso al sistema a ciascun utente:



Linux - 17

uscita da una shell

- per uscire dal ciclo di una shell di login si può:
 - usare il comando `logout`, oppure
 - digitare `CTRL+D` (carattere di end-of-file)

- una volta effettuato il `logout`, per riprendere a usare linux bisogna inserire nuovamente username e password (login)

- per uscire da una shell anche non di login esiste il comando `exit`.

Linux - 18

Comandi della shell di Unix

standard input, output, error; tipi di comandi

Linux - 19

Lo Shell di Unix: Comandi

- ogni comando richiede al nucleo l'esecuzione di una particolare azione

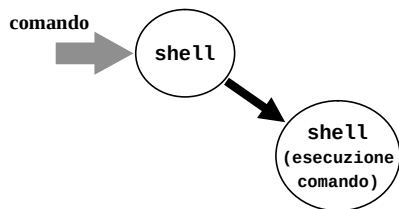
- i comandi esistono nel file system come file binari, generalmente eseguibili da tutti gli utenti (direttorio `/bin`)

- possibilità di realizzare nuovi comandi: programmazione in shell

Linux - 20

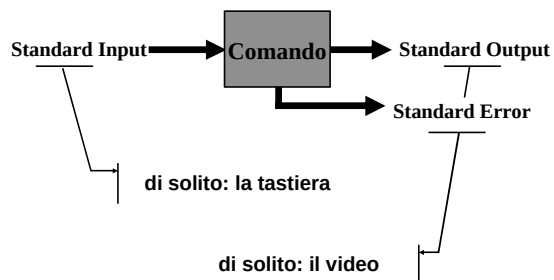
esecuzione di comandi

- per ogni comando da eseguire lo shell crea uno shell figlio, dedicato all'esecuzione del comando:



Linux - 21

input / output di un comando



Linux - 22

alcuni tipi di comandi

- interazione con il file system:
 - gestione di file e direttori
- gestione del sistema:
 - informazioni sulle risorse
 - modifica di dati di sistema

Linux - 23

esempi di comandi

```
ptorroni@lab3-linux:~$ date
Wed Apr 27 21:48:24 CEST 2005

ptorroni@lab3-linux:~$ who      (connected users info)
root      pst/3      Apr  9 14:02
root      pst/4      Apr 22 17:11 (:0.0)
paolo     pst/12     Apr 27 12:21 (deis32...)

ptorroni@lab3-linux:~$ who am i
paolo     pst/12     Apr 27 12:21 (deis32...
```

Linux - 24

File System

struttura logica del file system: tipi di file,
persorsi assoluti e relativi, comando cd

Linux - 25

file

- logicamente, un file è una sequenza di bit, a cui viene dato un nome
- in pratica, il file è una astrazione molto potente che consente di trattare allo stesso modo entità fisicamente diverse come: file di testo, dischi rigidi, stampanti, direttori, soft link, la tastiera, il video, etc.

Linux - 26

tipi di file

- regolari: archivi di dati, testi, comandi, programmi sorgente, eseguibili, ...
- directory: file gestiti direttamente solo dal S.O., che contengono riferimenti ad altri file
- speciali: dispositivi hardware, memoria centrale, hard disk, ...
- FIFO (pipe): file per la comunicazione tra processi
- soft link: riferimenti (puntatori) ad altri file o direttori

Linux - 27

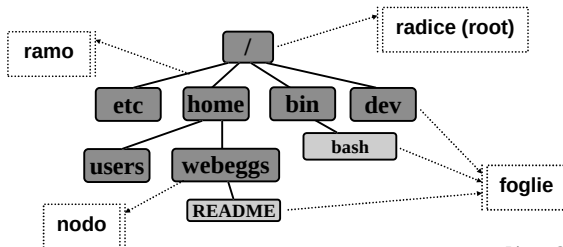
file: nomi

- È possibile nominare un file con una qualsiasi sequenza di caratteri (max. 255), a eccezione di '.' e '..' (sono nomi che hanno un significato particolare)
- È sconsigliabile utilizzare per il nome di file dei caratteri speciali, ad es. metacaratteri e segni di punteggiatura
- ad ogni file possono essere associati uno o più nomi simbolici (link) ma ad ogni file è associato uno ed un solo descrittore (i-node) identificato da un intero (i-number)

Linux - 28

direttori (directory)

- Il file system è organizzato come un albero rovesciato, composto da file



Linux - 29

gerarchie di direttori

- all'atto del login, l'utente può cominciare a operare all'interno di uno specifico direttorio (la sua home). In seguito è possibile cambiare direttorio.
- È possibile visualizzare il percorso completo attraverso il comando pwd (print working directory)
- Essendo i file organizzati in gerarchie di direttori, il sistema operativo mette a disposizione dei comandi per muoversi all'interno di essi

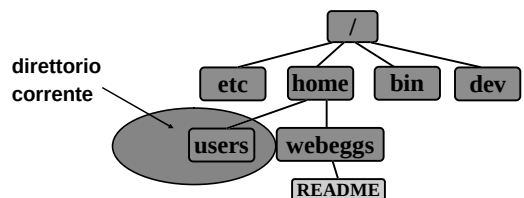
Linux - 30

nomi relativi / nomi assoluti

- ogni utente può specificare un file attraverso:
 - nome relativo: è riferito alla posizione dell'utente nel file system (direttorio corrente)
 - nome assoluto: è riferito alla radice della gerarchia (/)
- nomi particolari
 - . è il direttorio corrente (visualizzato da pwd)
 - .. è il direttorio 'padre'

Linux - 31

nomi relativi / assoluti: esempio



nome assoluto: /home/webeggs/README

nome relativo: ../webeggs/README

Linux - 32

file

concetto di file, comando ls, metacaratteri

Linux - 33

file

■ file = insieme (possibilmente vuoto) di byte organizzati in sequenza e identificato da un nome

■ creazione di un file vuoto: '`> nome_file`'

■ esempio:

```
ptorroni@lab3-linux:~$ > f1.txt
```

Linux - 34

gestione dei file: comando ls

■ consente di visualizzare nomi di file

■ varie opzioni: es. `ls -l` per avere più informazioni (non solo il nome del file)

■ possibilità di usare metacaratteri (*wildcard*): es. se esistono i file `f1`, `f2`, `f3`, `f4`, ci si può riferire ad essi scrivendo: `f*`, o più precisamente `f[1-4]`

Linux - 35

formato dei comandi

■ tipicamente: *nome* *-opzioni* *argomenti*

■ esempio: `ls -l temp.txt`

■ convenzione nel dare la sintassi dei comandi:

- se un'opzione / argomento può essere omesso, si mette tra quadre: `[opzione]`
- se due opzioni / argomenti sono mutuamente esclusivi, vengono separati da `|`: `arg1 | arg2`
- quando un arg. può essere ripetuto n volte, si aggiungono dei puntini: `arg...`

Linux - 36

opzioni del comando ls...

■ *sintassi (semplice):* `ls [-opzioni...] [file...]`

■ opzioni:

- **l** (long format): per ogni file una linea che contiene i diritti, il numero di link, il proprietario del file, il gruppo del proprietario, l'occupazione di disco (blocchi), la data e l'ora dell'ultima modifica o dell'ultimo accesso, e il nome
- **t** (time): la lista è ordinata per data dell'ultima modifica

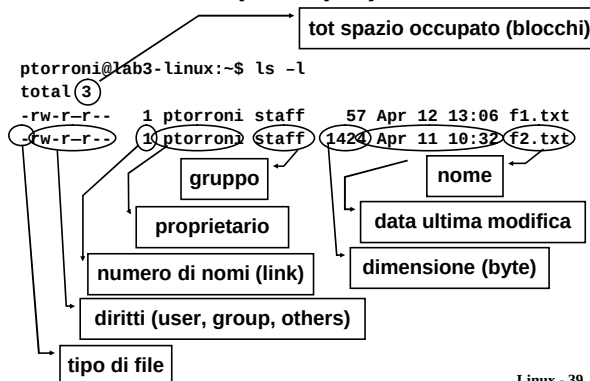
Linux - 37

...opzioni del comando ls

- **u**: la lista è ordinata per data dell'ultimo accesso
- **r** (reverse order): inverte l'ordine
- **a** (all files): fornisce una lista completa (normalmente i file che cominciano con il punto non vengono visualizzati)
- **F** (classify): aggiunge al termine del nome del file un carattere che ne indica il tipo (eseguibile: *, direttorio: /, link simbolico: @, FIFO: |, socket: =, niente per file regolari)

Linux - 38

ls -l (esempio)



Linux - 39

comandi, opzioni ??

- esiste un manuale on-line (man), che si può consultare ogni volta che si hanno dubbi su un comando Linux. Indica:
 - formato del comando (input)
 - risultato atteso (output)
 - descrizione delle opzioni
 - possibili restrizioni
 - file di sistema interessati dal comando
 - comandi correlati
 - eventuali difetti (bugs)
- per uscire dal manuale, digitare 'q' (quit)

Linux - 40

uso dei metacaratteri

i metacaratteri servono a specificare un pattern per identificare un insieme di nomi di file già esistenti. La shell provvede a sostituirli metacaratteri con i nomi di file.

- * sta per qualsiasi sequenza (anche vuota) di caratteri: es. *.java, file.*
- ? sta per esattamente un carattere: es. file.do?
- [] specificano una lista o un intervallo di caratteri (es. [a-c], [A-Za-z]: file.do[ct])
- gli apici possono essere usati per indicare che eventuali metacaratteri non vanno espansi

Linux - 41

uso dei metacaratteri (esempio)

```
ptorroni@lab3-linux:~$ ls
Xrootenv.0  f12.txt    hw         temp       vi.txt
f1.txt      f2.txt     p.c        temp.c

ptorroni@lab3-linux:~$ ls f?.txt
f1.txt f2.txt

ptorroni@lab3-linux:~$ ls f*.txt
f1.txt    f12.txt    f2.txt

ptorroni@lab3-linux:~$ ls [f-t]*.*
f1.txt    f12.txt    f2.txt    p.c        temp.c

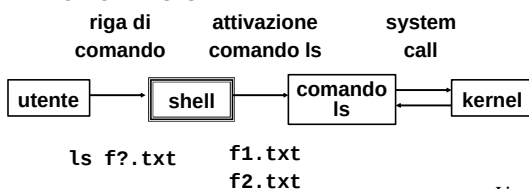
ptorroni@lab3-linux:~$ ls '*.c'
```

Linux - 42

funzione della shell

- la shell provvede a interpretare i comandi, a espandere i nomi dei file, e ad attivare i comandi stessi

■ Es: `ls f?.txt`



Linux - 43

il comando passwd

- È possibile cambiare la propria password di utente, mediante il comando `passwd`
- Verrà prima chiesta la vecchia password (per motivi di sicurezza)
- Se ci si dimentica della password, bisogna chiedere all'amministratore di sistema (utente `root`)

Linux - 44

vi

come creare e modificare un file di testo

Linux - 45

editor di testo: vi

- editor standard per Unix, è presente in tutte le distribuzioni base. Funziona con qualsiasi terminale a caratteri (per usarlo è sufficiente avere, oltre ai tasti carattere, un tasto <ESC> e <INVIO>/<ENTER>)
- prevede diverse modalità operative: comando, testo (inserimento/sostituzione), editor di linea.
- <ESC> viene utilizzato per passare alla modalità comando. Per passare dalla modalità comando alla modalità editor di linea, bisogna digitare il carattere due punti (:)

Linux - 46

vi: creazione di un file

- vi <nome_file> apre in ambiente vi il file indicato come argomento. Se il file non esiste lo crea. vi parte in modalità comando.
- es: vi f1.txt
- si può passare dalla modalità comando alla modalità testo, digitando i (insert), oppure a (append), oppure R (replace)

Linux - 47

vi: uscita dall'editor...

- per uscire da vi bisogna passare alla modalità riga di comando (<ESC> seguito da due punti, :). A questo punto:
- :q! (quit!) esce scartando le modifiche fatte dall'ultimo salvataggio (il punto esclamativo non è necessario se non si sono fatte modifiche)
- nota: i comandi di vi sono *case-sensitive* !

Linux - 48

... vi: uscita dall'editor / salvataggio di un file

- :wq (write and quit) o anche :x (exit) salva il testo con le eventuali modifiche ed esce
- :w (write) salva il testo che si sta modificando
- :w <file> (write to file) salva il contenuto dell'editor in un nuovo file. Se il file esiste già e si intende sovrascriverne il contenuto bisogna aggiungere un punto esclamativo: w! <file>

Linux - 49

vi: comandi...

- ESC u (undo): annulla l'ultima modifica /
- ESC U ripristina il contenuto della linea corrente
- ESC o inserisce una linea
- ESC x cancella un carattere
- ESC D cancella una riga a partire dal cursore
- ESC y copia nel buffer la riga corrente
- ESC p incolla (paste) nella posizione corrente il contenuto del buffer
- ESC / ricerca una stringa (poi n: next, per andare avanti nella ricerca)

Linux - 50

... vi: comandi

- vi prevede una sintassi particolare per ripetere n volte un certo comando. es.:
 - <ESC> 4d cancella 4 linee
 - <ESC> 5x cancella 5 caratteri
 - ...
- <ESC> :! consente di eseguire un comando a livello di sistema operativo
 - es: <ESC> :! ls -l

Linux - 51

perché vi ?

- può essere difficile all'inizio per chi è abituato a editor grafici; del resto, vi è l'unico editor sicuramente presente in tutti i sistemi Unix, per cui vale la pena spendere un po' di tempo per impraticarsene
- in alternativa a vi, esiste un altro editor, emacs, che fa uso anche dei tasti CTRL, ALT. Ovviamente ciascuno può utilizzare lo strumento che meglio crede

Linux - 52

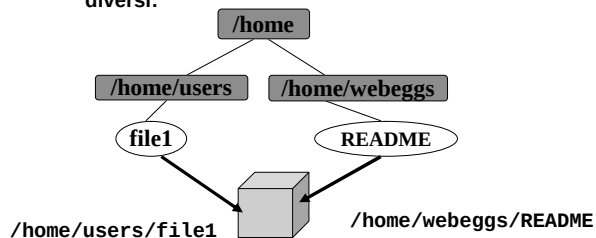
file con più nomi

il linking

Linux - 53

linking

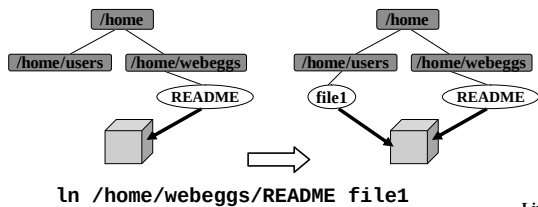
- lo stesso file può essere individuato da nomi diversi:



Linux - 54

comando ln

- il comando `ln` (link) serve a eseguire il linking di un file a un nuovo nome simbolico, serve cioè ad assegnare a un file esistente un nuovo nome in aggiunta a quello che già possiede.



Linux - 55

numero di link

- il numero di link di un file è quindi il numero di nomi assoluti diversi che esso ha all'interno del file system.

- il comando `ls -l` consente di conoscerlo. Es:

```
$ ls -l
total 2
-rw-r--r-- 1 anna staff 1424 Apr 11 10:32 f2.txt
$ ln f2.txt ../f1.txt
$ ls -l
total 2
-rw-r--r-- 2 anna staff 1424 Apr 11 10:32 f2.txt
```

Linux - 56

gestione dei link

- I file `.` e `..` sono in realtà dei link: il primo al direttorio corrente, il secondo al direttorio padre (nel quale il direttorio corrente risiede).
- Come vedremo, in un file system oltre che creare file è possibile eliminare file. In questo caso, conoscere il numero di link di un file è importante, perché eliminando un file il S.O. rischierebbe di lasciare nel sistema un nome di file, senza file...
- Unix gestisce questa operazione trasparentemente: cancellare un file significa effettivamente eliminarlo dal sistema solo se il relativo numero di link è 1.

Linux - 57

protezione

proprietà, accessi, bit di protezione

Linux - 58

proprietà di file

- Come abbiamo visto, a ciascun utente viene assegnato uno username e una password, e gli utenti sono classificati in gruppi. Es:
Username: anna [User-id: 1530]
Group: staff [Group-id: 22]
- ad ogni file è associato lo username ed il gruppo dell'utente proprietario (inizialmente, chi lo crea)
- In Unix è possibile cambiare la proprietà di un file (assegnandola a un altro utente / gruppo): comandi `chown`, `chgrp`

Linux - 59

accesso ai file

- esistono tre modalità di accesso ai file: lettura, scrittura, esecuzione
- il proprietario può concedere o negare agli altri utenti il permesso di accedere ai propri file
- esiste un utente privilegiato (`root`) che ha accesso incondizionato ad ogni file del sistema

Linux - 60

bit di protezione

- Ad ogni file sono associati 12 bit di protezione:

suid	sgid	sticky	r w x	r w x	r w x
			U	G	O

Linux - 61

Bit di Protezione: lettura, scrittura, esecuzione

suid	sgid	sticky	r w x	r w x	r w x
			U	G	O

9 bit di lettura (read), scrittura (write),
esecuzione(execute) per:

- utente proprietario (User)
- utenti del gruppo (Group)
- tutti gli altri utenti (Others)

Linux - 62

bit di protezione: lettura, scrittura, esecuzione

Ad esempio, il file:

	U	G	O
pippo	1 1 1	0 0 1	0 0 0
	r w x	- - x	- - -

- è leggibile, scrivibile, eseguibile per il proprietario
- è solo eseguibile per gli utenti dello stesso gruppo
- nessun tipo di modalità per gli altri

- formato ottale: 111 => 7; 010 => 2; ... -rwx--x--- => 0710

Linux - 63

bit di protezione per file eseguibili

suid	sgid	sticky	r w x	r w x	r w x
------	------	--------	-------	-------	-------

3 bit di permessi per file eseguibili:

- Set-User-ID (SUID)
- Set-Group-ID (SGID)
- Save-Text-Image (Sticky)

Linux - 64

SUID, SGID, Sticky

File eseguibili:

- al processo che esegue un file eseguibile è associato dinamicamente uno User-ID (e Group-ID) → User-ID effettivo
- Default: User-ID (e Group-ID) dell'utente che lancia il processo → User-ID reale

Linux - 65

SUID e SGID

- Set-User-ID (SUID): associa al processo che esegue il file lo User-Id del proprietario del file
- Set-Group-ID (SGID): associa al processo che esegue il file il Group-Id del proprietario del file



Chi lancia il processo assume temporaneamente l'identità del proprietario

Linux - 66

SUID, SGID , e file /etc/passwd

- in Unix tutte le informazioni relative alla amministrazione del sistema sono rappresentate da file (di root)

Esempio: utenti, gruppi e password sono registrati nel file /etc/passwd:

```
root:Mz5DJvSXy:0:1:Operator:::/bin/csh
paola:eLQZs:290:30:Paola Neri:/home/paola:/bin/csh
```

Linux - 67

SUID, SGID, e il file /etc/passwd

/etc/passwd

- necessità di concedere l'accesso in scrittura ad ogni utente solo per le modifiche relative al proprio username



attraverso il comando /bin/passwd (di root)

Linux - 68

SUID, SGID, e il file /etc/passwd

il comando `/bin/passwd` modifica il file `/etc/passwd`

`/bin/passwd`

1	1	111	101	101
SUID	SGID	U	G	O

⇒ chiunque lo esegue può accedere e modificare (in modo controllato) il file `/etc/passwd`, vestendo i panni del superutente (root)

Linux - 69

Bit di Protezione: Sticky bit

- **Save-Text-Image (Sticky):** l'immagine del processo viene mantenuta in memoria centrale (se possibile) anche dopo che ha finito il proprio compito (il processo è terminato)

Esempio:
i comandi utilizzati frequentemente

Linux - 70

modifica dei bit di protezione

- l'amministratore (root) può modificare la proprietà di un file attraverso i comandi `chown` / `chgrp`

- è possibile cambiare i permessi dei propri file attraverso il comando `chmod`:
 - `chmod mode <nomefile>`

mode: `[ugoa] [[+=][rwxXstugoa...]]... ] [, ...]`
oppure: formato ottale dei bit di protezione

Linux - 71

esempio chmod

```
$ ls -l file1.c
-rw-rw-r-- 1 anna staff ... file1.c

$ chmod 0666 file1.c
$ ls -l file1.c
-rw-rw-rw- 1 anna staff ... file1.c

$ chmod a-w,u=rw file1.c
$ ls -l file1.c
-rw-r--r-- 1 anna staff ... file1.c
```

Linux - 72

comandi per la gestione del file system

cd, rm, cp, mv, mkdir, rmdir

Linux - 73

muoversi all'interno del file system

- Unix consente di 'navigare' la gerarchia di direttori costituita dal file system.
- Abbiamo già visto il comando pwd, che consente di visualizzare il direttorio in cui 'ci si trova'. È possibile 'spostarsi' da un direttorio a un altro attraverso il comando cd
- Es: cd ..

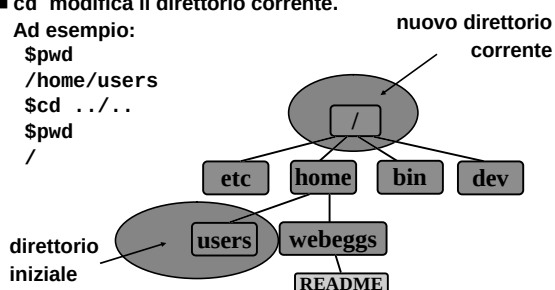
Linux - 74

il comando cd...

- cd modifica il direttorio corrente.

Ad esempio:

```
$pwd  
/home/users  
$cd ../../  
$pwd  
/
```



Linux - 75

...il comando cd

la sintassi è:

cd [<nuovo direttorio>]

- il direttorio destinazione si può esprimere con il nome relativo oppure assoluto
- se l'argomento non viene specificato, il nuovo direttorio è la home directory dell'utente
- per spostarsi all'interno di un determinato direttorio bisogna avere per tale direttorio i diritti di esecuzione

Linux - 76

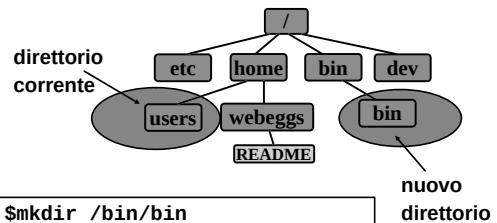
modifica del file system: direttori

- creazione di un direttorio: `mkdir <nomedir>`
- eliminazione di un direttorio : `rmdir <nomedir>`
- per creare un direttorio è necessario avere i diritti di scrittura nel direttorio all'interno del quale lo si vuole inserire
- per eliminare un direttorio è necessario avere i diritti di scrittura di tale direttorio

Linux - 77

esempio di mkdir

- `mkdir` crea un nuovo direttorio:



```
$mkdir /bin/bin  
$mkdir ../../bin/bin
```

Linux - 78

soft link (link simbolici)

- È possibile utilizzare il comando `ln` anche per creare un link a direttorio, mediante un link *simbolico*
- La sintassi è `ln -s <dir_origine> <nuovo_dir>`
- Il soft link è semplicemente un file contenente un puntatore a file (non viene tenuto il conto del numero di soft link)
- Il soft link esiste indipendentemente dall'oggetto a cui fa riferimento (file o direttorio)

Linux - 79

esplorazione ricorsiva

- Il parametro `-R` del comando `ls` specifica l'applicazione del comando stesso a tutti i direttori di un sottoalbero: `ls -R`
- I soft link non vengono inclusi nella ricorsione (non sono direttori): viene visualizzato il nome del soft link, e anche se si tratta di un link a direttorio il contenuto di questo direttorio non viene esplorato

Linux - 80

lettura di file di testo

- è necessario avere i diritti di lettura per visualizzare il contenuto di un file di testo
- `cat [<nomefile>...]`: visualizza l'intero file
- `more [<nomefile>...]`: visualizza per videate
- altri comandi:
 - `grep <stringa> [<nomefile>...]` (ricerca di una stringa in un file),
 - `wc [-lwc] [<nomefile>...]` (conto di righe / parole / caratteri)

Linux - 81

esempi di lettura file di testo

```
ptorroni@lab3-linux:~$ cat quasimodo
ciascuno sta solo sul cuor della terra
trafitto da un raggio di sole:
ed è subito sera
```

```
ptorroni@lab3-linux:~$ wc quasimodo
  3  17  87  quasimodo
```

```
ptorroni@lab3-linux:~$ grep sera quasimodo
ed è subito sera
```

Linux - 82

cancellazione, copia e spostamento di file

- è necessario avere i diritti di scrittura sul file per modificarlo (es. `vi`) o eliminarlo
- eliminazione di un file:
`rm <nomefile>`
- copia di un file (e diritti):
`cp <nomefile> <nuovofile>`
- spostamento di un file (e diritti):
`mv <nomefile> <nuovofile>`

Linux - 83

esempi

```
ptorroni@lab3-linux:~$ cp quasimodo sera
ptorroni@lab3-linux:~$ ls
quasimodo      sera
```

```
ptorroni@lab3-linux:~$ mv quasimodo poesia
ptorroni@lab3-linux:~$ ls
poesia        sera
```

```
ptorroni@lab3-linux:~$ rm poesia
ptorroni@lab3-linux:~$ ls
sera
```

Linux - 84

esempi modifica file (diritti)

```
ptorroni@lab3-linux:~$ ls -l quasimodo
-rw-r--r-- 1 ptorroni staff ... quasimodo
ptorroni@lab3-linux:~$ chmod 0400 quasimodo
ptorroni@lab3-linux:~$ mv quasimodo subito
ptorroni@lab3-linux:~$ rm subito
ptorroni@lab3-linux:~$ ls
rm: remove 'subito', overriding mode 0400? n
ptorroni@lab3-linux:~$ vi subito
.....
<ESC>:x
'readonly' option is set (use ! to override)
```

Linux - 85

Creazione ed esecuzione di comandi in Linux

compilatore C

file comandi (scripting)

Linux - 86

Compilatore C: gcc

- File sorgente: deve avere come suffisso .c (Es: HelloWorld.c)
- compilazione di HelloWorld.c :
- gcc HelloWorld.c
- viene prodotto un eseguibile, che per default è chiamato a.out

Linux - 87

gcc

- senza opzioni, gcc preprocessa, compila, assembla e linka producendo a.out
 - gcc HelloWorld.c # produce a.out
- Se voglio produrre un eseguibile che si chiami in un modo diverso da a.out, specifico l'opzione -o (output)
 - gcc -o hw HelloWorld.c # produce hw
- Se voglio arrestarmi dopo la produzione del modulo oggetto, specifico -c (output per default: .o)
 - gcc -c HelloWorld.c # produce HelloWorld.o

Linux - 88

Esercizio: Hello World!

- utilizzare un editor per scrivere un programma C dal titolo `HelloWorld.c`, implementando il famoso programma di test.
- compilare `HelloWorld.c`, producendo l'eseguibile `hw`. Nota: per eseguirlo, includere il path!! (es. `./hw`)

Linux - 89

comando ps

- Un processo utente in genere viene attivato a partire da un comando. Di esso prende il nome. Ad es., dopo aver mandato in esecuzione il comando `hw`, verrà visualizzato un processo dal nome `hw`. Tramite `ps` si può vedere la lista dei processi attivi

```
ptorroni@lab3-linux:~$ ps
  PID  TTY  STAT  TIME  COMMAND
  4837  p2  S    0:00  -bash
  6945  p2  S    0:00  sleep 5s
  6948  p2  R    0:00  ps
```

Linux - 90

Esercizio: uso di ps

- Esercizio:
 - inserire in `HelloWorld.c` un ritardo di qualche secondo (ad es. aggiungere le righe `sleep(20);` `printf("Finito!\n");`)
 - ricompilare
 - aprire un'altra finestra di shell
 - mandare in esecuzione `hw`, e dall'altra finestra verificare l'esistenza di un processo nuovo chiamato `hw...`

Linux - 91

Esecuzione in background

- I processi possono essere attivi in foreground o in background
- È possibile attivare un processo in background, tramite la seguente sintassi: `<nome_comando> &`
- È possibile chiamare in foreground un processo attualmente in background, tramite il comando `fg`
- Il `CTRL+Z` interrompe momentaneamente un processo; restituisce tra parentesi quadre il numero del "job" relativo al processo interrotto
- Dato un processo attualmente in foreground, è possibile:
 - interromperlo, tramite `CTRL-Z`
 - riprenderlo, in background o foreground (`bg / fg <job#>`)

Linux - 92

terminazione forzata di un processo

- È possibile 'terminare forzatamente' un processo tramite il comando `kill`
- `kill -9 <PID>` provoca l'invio di un segnale `SIGKILL` al processo identificato dal PID
 - Esempio: `kill -9 6944`
- per conoscere il PID di un determinato processo, si può utilizzare il comando `ps`

Linux - 93

monitor dei processi: top

```
4:20pm up 3 days, 6:48, 1 user, load average: 0.00, 0.00, 0.00
84 processes: 83 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 0.7% user, 0.3% system, 0.0% nice, 99.0% idle
Mem: 127840K av, 122660K used, 5100K free, 37088K shrd, 10024K buff
Swap: 130404K av, 16K used, 130388K free, 100040K cached
```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	LIB	%CPU	%MEM	TIME	COMMAND	
28704	pterroni	12	0	748	748	568	R		0	0.9	0.5	0:02	tcp
28208	root	2	0	1084	1084	732	S		0	0.1	0.8	0:00	sshd
1	root	0	0	432	432	356	S		0	0.0	0.3	0:01	init
2	root	0	0	0	0	0	SW		0	0.0	0.0	0:00	kflushd
3	root	-12	-12	0	0	0	SW		0	0.0	0.0	0:00	kswapd
4	root	0	0	0	0	0	SW		0	0.0	0.0	0:00	ad_thread
5	root	0	0	0	0	0	SW		0	0.0	0.0	0:00	ad_thread
495	root	0	0	4660	4660	1612	S		0	0.0	3.6	0:41	X
621	root	0	0	320	320	264	S		0	0.0	0.2	0:00	minigetty
28748	pterroni	8	0	280	280	236	S		0	0.0	0.2	0:00	sleep
31	root	5	0	376	376	324	S		0	0.0	0.2	0:00	kernelld
194	root	0	0	596	596	496	S		0	0.0	0.4	0:01	syslogd
203	root	0	0	536	536	336	S		0	0.0	0.4	0:00	klogd
214	daemon	0	0	416	416	340	S		0	0.0	0.3	0:00	atd
225	root	1	0	484	472	440	S		0	0.0	0.3	0:00	crond
226	root	0	0	416	416	336	S		0	0.0	0.3	0:00	crond

Linux - 94

Esempio di file comandi (hw)

- È possibile scrivere il comando hello-world anche come file comandi (script)
- Invece di compilarlo, bisogna renderlo 'manualmente' eseguibile
 - `chmod u+x <nome_comando>`

Linux - 95

Esercizio (file comandi)

- Scrivere un file comandi `hw1` che visualizzi la scritta 'Hello World!', attenda 20 secondi, visualizzi 'Finito!' e termini.
 - Comandi da usare:
 - `echo`
 - `sleep`(vedere il manuale on-line)
- Mandare in esecuzione i due comandi, sia nella versione compilata, sia nella versione script, e verificare con `ps` / `top` quali e quanti processi vengono rispettivamente creati. Ci sono delle differenze? Usare `ps -l` per verificare la paternità dei processi

Linux - 96

- `echo Hello World`

- `top &`

Linux - 97

segnali e interruzioni...

- È possibile interrompere un processo (purché se ne abbia il permesso...)
- `kill -s <PID>` provoca l'invio di un segnale (individuato dal parametro `s`) al processo identificato dal PID
- abbiamo già visto `kill -9`: 9 corrisponde a SIGKILL, il che provoca la terminazione incondizionata del processo (segnale non mascherabile) e dei figli (ricorsivamente)

Linux - 98

...segnali e interruzioni

- Alcuni tra i segnali più comuni sono:
 - CTRL-C (invia un SIGINT, `kill -2` del processo attualmente in foreground),
 - CTRL-Z (invia un SIGTSTP, sospensione di un processo, `kill -20`)
- `kill -l` fornisce la lista dei segnali

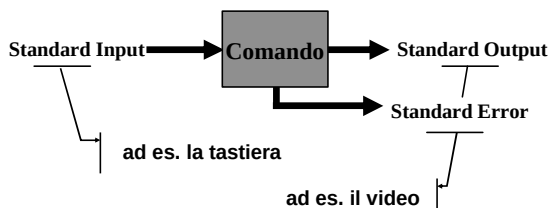
Linux - 99

redirezione e piping

`>, >>, <, |`

Linux - 100

input / output di un comando



- Il comando è visto come un filtro, che trasforma i dati in ingresso (stdin) in dati in uscita (stdout)

Linux - 101

comandi - filtri

■ Alcuni esempi:

- `grep <testo> [<file>...]` Input: (lista di) file (se specificata). Output: video
- `tee <file>` Scrive l'input sia su file, sia sul canale di output.
- `sort [<file>...]` Ordina alfabeticamente le linee. Input: (lista di) file (se specificata). Output: video
- `rev <file>` Inverte l'ordine delle linee di file.
- `cut [-options] <file>` seleziona colonne da file.

Linux - 102

modificare input / output

- È possibile far sì che un comando riceva l'input da un canale diverso dallo standard:
`<comando> < <file>`
- Analogamente, si può cambiare il canale di output:
`<comando> > <file>`

Linux - 103

esempi (comandi - filtri)

```
ptorroni@lab3-linux:~$ cut -c-9 quasimodo  
ciascuno  
trafitto  
ed è subi
```

Esercizio: creare un file di testo (ad es. chiamato `quasimodo`) e provare i seguenti comandi:

- `cut -c-9 < quasimodo`
- `cut -c-9 > temp`
- `cut -c-9 >> temp`
(CTRL+D è il carattere di fine-file)

Linux - 104

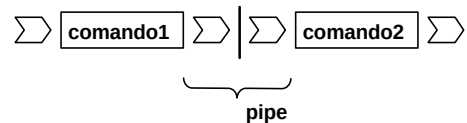
piping

- L'output di un comando può servire da input di un altro comando.
- Soluzione: si può (1) redirigere l'output del primo comando su di un file temporaneo, e (2) redirigere l'input del secondo comando dallo stesso file
- La pipe è uno strumento messo a disposizione da Unix, che realizza un canale virtuale tra due comandi

Linux - 105

<comando1> | <comando2>

- '|' è il simbolo di pipe. Indica che l'output del comando di sinistra è rediretto in input al comando di destra



Linux - 106

esempi (pipe)

```
$ cat f1 | wc -l
    32 // numero di linee di f1
$ wc f1 | cut -c9-16
    194 // numero di parole di f1
$ grep sera < quasimodo | tee f2 | wc
    1      4      17
// numero di linee, parole e caratteri
// dell'output di grep sera < quasimodo
$ cat f2
ed è subito sera
```

Linux - 107

backquote (`)

```
$ echo direttorio corrente: pwd
→ come faccio a dire che pwd è un comando
che va eseguito?
```

- ` (backquote, o apice inverso) serve a far sì che l'interprete consideri il testo contenuto tra apici come un comando (sostituzione di comando):

```
$ echo direttorio corrente: `pwd`
direttorio corrente: /home/staff/ptorroni
```

Linux - 108

apici doppi + backquote

- Gli apici doppi espandono, oltre che le variabili, anche i comandi:

```
$ cd /usr/bin
$ echo "`pwd`"
/usr/bin
$ echo "$TERM: `pwd`"
xterm: /usr/bin // con o senza "
$ echo "$TERM: >>`pwd`<<"
xterm: >>/usr/bin<< // devo usare " !!
```

Linux - 109

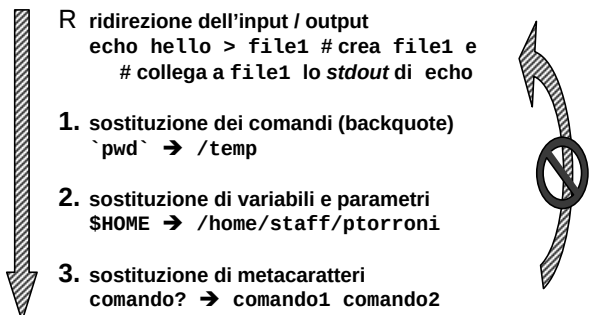
apici dentro le stringhe

```
oggi e\' giovedì\' # 3 stringhe
oggi e\' \'giovedì\'\' # 3 stringhe

"oggi e\' giovedì'" # 1 stringa
"oggi e\' \'giovedì\'" # 1 stringa
"oggi e\' \'giovedì\'" # 1 stringa
```

Linux - 110

fasi del *parsing* della shell

- 
- R ridirezione dell'input / output
echo hello > file1 # crea file1 e
collega a file1 lo *stdout* di echo
1. sostituzione dei comandi (backquote)
`pwd` → /temp
 2. sostituzione di variabili e parametri
\$HOME → /home/staff/ptorroni
 3. sostituzione di metacaratteri
comando? → comando1 comando2

Linux - 111

apici e fasi del parsing

- ' e " sono di fatto dei comandi per il controllo del parsing:
- ' : nessuna sostituzione (blocca tutte le fasi: R, 1, 2, e 3)
- " : vengono sostituiti comandi e variabili, mentre i caratteri speciali, compresi quelli per la ridirezione, non vengono considerati tali (blocca solo le fasi R e 3)

Linux - 112

esempi (parsing)

```
echo `pwd` > "f1">
# R: crea f1, poi stdout_echo= f1> ; echo `pwd`
# 1: echo /usr/bin
# 2: echo /usr/bin
# 3: echo /usr/bin

test -f `pwd`/$2 -a -d "$HOME/dir?"
# R: test -f `pwd`/$2 -a -d "$HOME/dir?"
# 1: test -f /temp/$2 -a -d "$HOME/dir?"
# 2: test -f /temp/arg_n2 -a -d \
"/home/staff/ptorroni/dir?"
# 3: test -f /temp/arg_n2 -a -d \
/home/staff/ptorroni/dir?
```

Linux - 113

cicli (for)

```
for <var> [in <list>]
do
    <comandi>
done
```

■ list = lista di stringhe

Linux - 114

esempi (for)

```
for i in *
# esegue per tutti i file nel direttorio corrente

for i                # cioè: for i in $*
# esegue per tutti i parametri di invocazione

#file crea
for i
do > $i            # ridirezione di input su $i
done
```

Linux - 115

esempi (for)

```
for i in `cat file1`
do <comandi per ogni parola del file
    file1>
done

for i in `ls s*`
do cat $i | grep $2 | wc -w
done
```

Linux - 116

cicli (while)

```
while <lista_comandi>
do
  <comandi>
done

■ while : esegue un loop infinito:
while :
  do echo "e' passato un minuto"
    sleep 60s
  done
```

Linux - 117

esempi (while)

```
while [ ! -f $1 ]
do
  sleep 10s; echo file assente
done

# ripete finché non compare un file di nome $1
```

Linux - 118

esempio di file comandi

```
# nuovo comando: loop <directory>
# avvisa se sono stati creati o
# eliminati file in una directory

if [ $# -ne 1 ] # controllo formato
then
  echo "usage: $0 nomedir"
  exit 1 # formato errato
fi
```

Linux - 119

esempio di file comandi

```
if [ ! -d $1 ] # controllo parametro
then
  echo "$1 is not a valid directory"
  exit 2 # parametro errato
fi

# creazione file temporaneo
if [ ! -e loop.$$tmp ]
then echo 1 > loop.$$tmp
fi
```

Linux - 120

esempio di file comandi

```
while :
do
  sleep 5s
  if [ `ls $1 | wc -w` -ne `cat loop.$$tmp` ]
  then
    ls $1 | wc -w > loop.$$tmp
    echo in $1 sono presenti \
`cat loop.$$tmp` file
  fi
done
```

Linux - 121

esercizio (monitor di un dirett.)

- scrivere un comando che ogni 5 secondi controlla se sono stati creati o eliminati file in un direttorio: nel caso, visualizza un messaggio su stdout (quanti file sono presenti nel direttorio)
- deve poter essere invocato con uno e un solo parametro: il direttorio da controllare (→ controllo dei parametri)
- uso di un file temporaneo, in cui tengo traccia

Linux - 122

esempio di file comandi

```
# sintassi: loop <directory>
# avvisa se sono stati creati o
# eliminati file in una directory

if [ $# -ne 1 ] # controllo formato
then
  echo "usage: $0 nomedir"
  exit 1 # formato errato
fi
```

Linux - 123

esempio di file comandi

```
if [ ! -d $1 ] # controllo parametro
then
  echo "$1 is not a valid directory"
  exit 2 # parametro errato
fi

# creazione file temporaneo
if [ ! -e loop.$$tmp ]
then echo 1 > loop.$$tmp
fi
```

Linux - 124

esempio di file comandi

```
while :  
do  
  sleep 5s  
  if [ `ls $1 | wc -w` -ne `cat loop.$$tmp` ]  
  then  
    ls $1 | wc -w > loop.$$tmp  
    echo in $1 sono presenti \  
  `cat loop.$$tmp` file  
  fi  
done
```

Linux - 125