

Linux, istruzioni per l'uso

Linux è un Sistema Operativo e, come tale, mette a disposizione degli strumenti per utilizzare una macchina.

Utilizzare una macchina (intesa come calcolatore) significa poter disporre attivamente delle sue risorse, quali:

- la capacità elaborativa

- la memoria

- i dispositivi di input/output

il kernel

- il compito di “gestire” le risorse viene svolto da un programma detto kernel che:
 - ha il controllo completo sull'hardware
 - fornisce i propri servizi attraverso le system call, ovvero porzioni di codice che vengono attivate da opportune sequenze di opcode, secondo un insieme di regole univocamente definite da ogni sistema
 - solitamente (almeno per gli unices classici), non dispone di alcuna interfaccia diretta verso l'utente-operatore

le applicazioni

- ogni programma esterno al kernel è considerato una applicazione
 - le applicazioni possono usufruire dei servizi del kernel tramite le system call
 - ogni applicazione viene messa in esecuzione dal kernel, che le attribuisce una frazione di capacità elaborativa (time slicing) e una o più porzioni di memoria virtuale e/o fisica.
 - l'accesso ai dispositivi di input e output, quali terminali, rete, filesystem, multimedia, avviene tramite i device driver del kernel (con qualche eccezione per le schede video)

il kernel Linux

- Linux è un kernel (solo quello) che appartiene alla famiglia degli “Unix”
 - condivide con gli altri unices, alcuni standard di funzionamento e di interfaccia alle applicazioni (Posix, Single Unix Specification, X/Open, BSD, SVID)
 - è un kernel monolitico, nel senso che non è basato su microkernel, anche se consente l'inserimento e la rimozione di funzionalità a run-time attraverso i moduli

applicazioni “di sistema”

- il kernel e l'utente possono “comunicare” solo attraverso le applicazioni
 - alcune di queste sono pertanto “vitali” alla fruibilità del sistema, come la shell che rappresenta l'interfaccia privilegiata fra l'utente e la macchina
 - gran parte di queste appartengono al progetto GNU e sono disponibili sia per Linux che per altri unices, sia open source che proprietari (open/free/net-Bsd, Solaris, MacOSX) ma anche per Windows attraverso il progetto Cygwin.

la grafica

- a differenza di altri sistemi, la grafica non ha alcun ruolo privilegiato per il kernel
 - l'interfaccia grafica di Linux e di molti unices è demandata ad un applicativo, l'X-server
 - l'utente può eseguire le applicazioni e/o i comandi in modo completamente equivalente sia da interfaccia "terminale" che da interfaccia "grafica"
 - la finestra grafica è solo un "output" su cui inviare dati o immagini – un programma può essere eseguito su di una macchina ed utilizzare video, mouse e tastiera di un'altra!

la linea di comando

- un unix, tutto si può fare attraverso un terminale, ovviamente se si sa come invocare opportunamente le applicazioni
- l'invocazione delle applicazioni si ottiene con una linea di comando, nella quale compaiono i nomi dei comandi, le opzioni, gli argomenti e le eventuali ridirezioni
- la sintassi della linea di comando dipende dalla “shell” che si usa, anche se le differenze sono di norma limitate

la Bourne shell

- la shell è una applicazione che si occupa di tradurre le command line in system call verso il kernel
 - la maggioranza degli unices (forse tutti) dispongono di una shell che utilizza la sintassi della Bourne shell
 - le distribuzioni Linux più utilizzate installano la bash che rappresenta una evoluzione della Bourne shell
 - ogni utente può specificare la shell che preferisce e questa verrà messa in esecuzione ad ogni accesso da terminale

il prompt

- quando la shell è pronta ad accettare comandi, lo segnala mediante l'emissione di una sequenza di caratteri detta “prompt”
- `alessandro@copperbottom:~$ _`
 - le informazioni riportate nel prompt sono personalizzabili ma, solitamente, terminano con `$` o `#` rispettivamente nel caso di utente comune o super-utente
 - il super-utente di unix ha identificativo `0` e normalmente ha per nome “root”

regole sintattiche di sh

- ogni cmdline termina con un “\n”
- gli elementi sono separati da blank (tipicamente)
 - un blank è uno spazio o un tab
- il primo elemento è il nome del comando da eseguire
- gli elementi successivi sono detti argomenti
- se un argomento inizia con -, viene detto opzione

esempio di cmdline

```
$ ls /usr/bin -l
```

- ls: nome del comando (primo elemento); la shell cerca nelle directory del PATH un file eseguibile con quel nome (/bin/ls)
- /usr/bin: primo argomento passato al comando ls all'atto della sua esecuzione
- -l: secondo argomento passato ad ls, iniziando con - si considera una opzione
- esistono dei meccanismi standard per processare le opzioni (getopt)

accesso al filesystem

- introduciamo qualche comando utile per esplorare il filesystem
- il filesystem è una struttura gerarchica con una unica radice detta root (/)
- la shell (come ogni processo) ha una directory corrente nella quale si cercano i file espressi in forma relativa
- un file può essere espresso in forma assoluta, in tal caso la ricerca avviene dalla root

pwd

- questo comando riporta su standard output il nome assoluto della directory corrente del processo
- se si conosce la directory corrente, ogni nome di file espresso in forma relativa può essere trasformato in forma assoluta
- `pwd = /home/utente`, `filerel = dir/prova.c`
- `fileass := /home/utente/dir/prova.c`
- `filerel2 = ../utente2/dir/prova.c ??`

. e ..

- ogni directory unix può contenere altre directory ma sicuramente contiene due directory particolari:
- .. è un “link” alla directory superiore
- . è un link alla medesima directory

/a/b/../c/d/../e == a/c/e

/a/b./c/d./e == a/b/c/d/e

- utilità di . -> si indica un nome di file

esecuzione di comandi

- la shell cerca i comandi nel PATH
- se non trova il comando nel PATH, non cerca un file eseguibile nella dir corrente (a differenza di dos/win)

\$ prova a b c

cerca il file eseguibile prova nel PATH perchè prova è considerato un comando

\$./prova a b c

cerca prova nella dir corrente perchè “./prova” non può che essere un nome di file

ls

- lista i file di una directory, senza argomenti lista la dir corrente

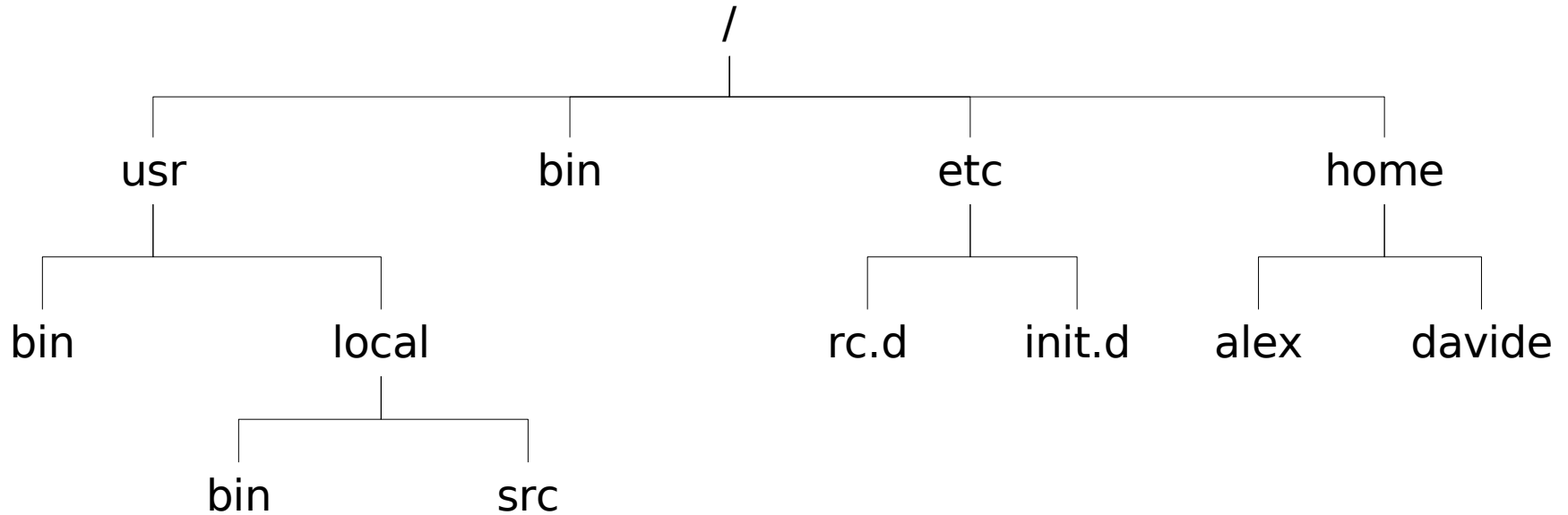
ls -l lista in formato lungo

ls -a lista anche i file che iniziano per . come . e .. o .prova

ls -la come ls -l -a, le opzioni si possono comporre fra loro

ls abc def ghi lista i file/directory indicati (abc,def,ghi). Se c'è una directory fra questi, ls lista il relativo contenuto

filesystem e nomi relativi/assoluti



Directory corrente: /usr

/usr/local/bin assoluto

local/bin relativo

gerarchia standard del filesystem

- /bin eseguibili principali di sistema
 - /dev file speciali per i dispositivi (o device)
 - /etc file di configurazione del sistema e dei programmi
 - /lib librerie di sistema, statiche e dinamiche
 - /tmp file temporanei
 - /usr file di uso comune per tutti gli utenti come programmi, documentazione, esempi.
- Spesso /usr viene montata in sola lettura per migliorare la velocità di accesso in lettura e la sicurezza; in tal caso viene affiancata dalla gerarchia /var per la parte writable.

collocazione dei file nel filesystem

	sistema	utente	dedicati
programmi eseguibili	/bin	/usr/bin	/usr/local/bin
programmi di sistema	/sbin	/usr/sbin	/usr/local/sbin
file di configurazione	/etc	/etc	/etc
file di supporto	/usr/share	/usr/share	/usr/local/share
documentazione	/usr/share/man	/usr/share/man	/usr/local/share/man
info di esecuzione	/var/run	/var/run	/var/run
log	/var/log	/var/log	/var/local/log

gerarchie alternative

- / gerarchia principale, solitamente riservata agli applicativi di sistema; l'installazione in questa gerarchia viene normalmente fatta attraverso i tool della distribuzione (apt, rpm, dpkg, pkgtool, ecc...)
- /usr come / ma contiene le utility
- /usr/local contiene file ed applicativi eventualmente compilati dall'utente (personalizzati); installazione con: make install
- /opt gerarchia totalmente sotto il controllo dell'utente
- /home/xxx totalmente libera per xxx

come gestire le priorità

- ogni utente può personalizzare la variabile PATH dell'environment
- l'ordine in cui appaiono le directory nel PATH corrisponde all'ordine di ricerca:

PATH=

`/home/xxx/bin:/opt/bin:/usr/local/bin:/usr/bin:/bin`

- il primo percorso (`/home/xxx/bin`) è il primo candidato alla ricerca per l'utente (`xxx`) che utilizza questo setting
- se esistono diversi eseguibili in altrettanti percorsi distinti del PATH, quello che si trova elencato per primo ha la prevalenza

quale eseguibile si usa

- visto che ogni utente può personalizzare la variabile PATH di ogni sua shell (eventualmente anche in modo differente!), esiste il comando

which

- che restituisce il nome assoluto del file eseguibile che sarà eseguito per il comando indicato
- Attenzione: l'installazione di un “pacchetto” potrebbe alterare il valore della variabile PATH

come modificare PATH

- modifica interattiva:
 - `PATH=nuovdir:$PATH`
- trasmissione della variabile PATH alle sotto shell:
 - `export PATH`
- valore “iniziale”
 - `/etc/profile` (per `sh`, `bash`, `ksh`, `ash`, ...)
 - `~/.bash_profile` `~/.bash_login` `~/.profile` (login shell)
 - `/etc/bash.bashrc` and `~/.bashrc` (non login shell)
- in nessun modo una sotto shell può alterare le variabili di ambiente (come PATH) di una shell antenata

esercizio: sostituire il comando ls

- l'eseguibile del comando ls si trova in /bin/ls
- indichiamo una directory in cui abbiamo i diritti di scrittura in posizione precedente nel PATH (es. /home/utente/bin)

```
export PATH=/home/utente/bin:$PATH
```

- creiamo uno script eseguibile in /home/utente/bin con nome "ls"
- da una nuova shell (bash ha una sorta di cache dei comandi per cui nella medesima shell il trucco non funziona ... solo per certe versioni di bash) invochiamo il comando ls che, questa volta, sarà sostituito dal nostro eseguibile

pacchetti: eseguibili+file di supporto

- D.Knuth (ideatore del Pascal) affermava che “algoritmi + strutture dati = programmi”
- un software spesso necessita di fonti accessorie, documentazione, esempi, template, librerie, ecc...
- per questo motivo si indicano con il termine “pacchetti” l'insieme di tutti i file che devono essere installati nel sistema per un corretto funzionamento del software
- ... per ora abbiamo imparato come cercare, trovare e sostituire l'eseguibile ma per i file di supporto come possiamo fare? La soluzione non è banale!

esempio: man page

- i progettisti di tool unix hanno mantenuto la buona abitudine di fornire insieme all'eseguibile una pagina di manuale (man page)
- in tutti gli unix, pertanto, esiste una qualche implementazione del programma “man” che cerca, visualizza o stampa le man page
- dove trovo la man page di un comando?
 - normalmente è un problema di “man” (/etc/manpath.config) ma se siamo curiosi:
whereis
 - che ci dice dove troviamo un comando e la sua man page (e i sorgenti, se presenti nel sistema)

qualche consiglio per iniziare

- acquisire il “metodo” unix non è difficile: i comandi non sono ne' semplici ne' intuitivi ma sono funzionali e documentati (standard ... vedi oltre)
 - cercare sempre le informazioni che ci servono nella man page dei comandi, spesso (ma non sempre) la documentazione allagata è completa e ben fatta
 - il progetto GNU ha introdotto “info” che fornisce una alternativa alle man page sotto forma di ipertesti; se abbiamo installato info ed esiste la documentazione in tale formato di un certo tool, possiamo invocare:

info <nometool>

- guardare, capire e riutilizzare il più possibile ciò che è già stato fatto (open source) o documentato (howto)

standard: a chi servono?

- una pubblicità di qualche tempo fa recitava: “diffidate dalle imitazioni!”. Siamo certi che le imitazioni siano realmente dannose?
- unix è stato “copiato” più e più volte, sia come kernel che come applicazioni (shell, utility, ecc.) ma ogni diversa implementazione ha solo migliorato l'idea originale (magari con percorsi non proprio monotoni -> a volte si peggiora anche!)
- aderire ad uno standard o rendere pubbliche le specifiche o, addirittura, i sorgenti di un software fornisce a chiunque la possibilità di lavorare, migliorare o semplicemente adattare ed estendere il principio originale.

lo standard al servizio dell'utente

- l'utente che impara ad operare con uno strumento “standard”, potrà contare sulla “portabilità” della propria preparazione
- un sistemista unix degli anni '80, anche dopo un ipotetico letargo di oltre 20 anni sarebbe oggi ancora utile.
- i pochi coraggiosi che impararono le API di Windows 3.1 negli anni '90 oggi sarebbero disoccupati!
- all'utente conviene utilizzare strumenti standard perchè in questo modo la sua professionalità ed esperienza andranno difficilmente in obsolescenza!

lo standard al servizio del vendor?

- il vendor che si allinea ad uno standard si espone alla concorrenza di altri vendor che possono emulare (ma, attenzione, non copiare) il proprio sw
 - la competizione fra vendor si fa dunque sulla qualità visto che tutti possono fornire la propria versione equivalente ma non uguale (copyright)
 - se un vendor non si allinea agli standard, sarà l'unico a poter fornire quel sw ma avrà grosse difficoltà a vendere il proprio prodotto agli utenti intelligenti. Questi, infatti, consci del pericolo di monopolio/trust non adotteranno mai soluzioni e formati proprietari.
- i siti dei nostri uffici pubblici, scuole, università, amministrazioni, forze armate, ecc. sono pieni di file .doc ... saranno utenti intelligenti?! e noi?

unix non standard

- esistono implementazioni unix che si discostano dagli standard, ad esempio per quanto riguarda il fhs (struttura del filesystem)
- alcune differenze implementative fra gli unix di derivazione Berkeley e quelli AT&T introducono ancora oggi difficoltà ai progettisti
- Consiglio: utilizzare ed adottare le implementazioni non standard solo se queste introducono vantaggi adeguati -> ogni differenza richiede un apprendimento mnemonico aggiuntivo quindi uno sforzo che deve essere giustificato ... ecco perchè oggi c'è ancora chi usa il vi e la shell ;-)