Introduzione a Linux

Modulo 3: come lo uso?

Alessandro Brunengo
Mirko Corosu
INFN – Sezione di Genova

Parte I

Login, logout

Login

- Dopo il boot, il sistema linux presenta un *prompt* (o una finestra grafica se e' attiva l'interfaccia grafica X) che richiede uno username ed una password
- Una volta verificate le credenziali, viene eseguito un processo di terminale, gestito da un programma chiamato "shell", che accetta comandi in input e produce messaggi di output
- In caso di interfaccia grafica, parte il desktop (vedi sezione dedicata ad X)

Login (cont.)

- All'atto del login, il sistema cerca nei database degli utenti (generalmente i file /etc/passwd e /etc/shadow) le caratteristiche dell'utente:
  - password
  - login shell
  - default working directory o home directory
- Il processo iniziale (la shell di login) si colloca nella home directory
- Il processo iniziale e tutti i processi in seguito attivati saranno di proprieta' dell'utente corrispondente allo username fornito

Login (cont.)

- In virtu' delle caratteristiche multi-process e multi-user del kernel linux, possono lavorare contemporaneamente piu' utenti, e piu' connessioni (login) dello stesso utente.
- Piu' connessioni contemporanee dello stesso utente potranno operare ciascuna in modo indipendente dalle altre (ma sugli stessi file e directory!)

Logout

- Quando si desidera terminare la sessione di login, si deve dare un comando di "disconnessione" dal sistema. A seconda del tipo di shell utilizzata questi comandi possono essere:
  - **exit** (dovrebbe funzionare sempre)
  - **CTRL^D** (a volte e' disabilitato)
  - **logout** (caratteristico della csh)
- Il sistema presentera' nuovamente il prompt di login per iniziare una nuova sessione

Logout (cont.)

- In caso di utilizzo di interfaccia grafica X, la sessione viene chiusa utilizzando l'apposito comando di logout (o disconnect) fornito dall'interfaccia grafica
- In seguito al logout, il sistema ripresentera' la finestra grafica per una nuova login

Parte II

La shell

La shell

La shell e' un programma finalizzato ad accettare input sotto forma di comandi di linea, ed a visualizzare messaggi in output.

L'ambiente in cui lavora e' un terminale alfanumerico.

Costituisce una interfaccia tra l'utente ed il sistema operativo

Tipi di shell

Esistono numerosi tipi di shell, simili nelle funzioni essenziali. I principali sono:

- sh (Bourne shell)
- ksh (Korn shell)
- csh (C shell)
- tcsh (Turbo C shell)
- bash (Bourne agan shell)
- rsh (Reduced shell)

Parte II.1

Caratteristiche comuni

il prompt

- la shell utilizza un prompt, cioe' una stringa di caratteri posta all'inizio della riga, per indicare all'utente che e' pronta a ricevere comandi
- spesso le shell utilizzano come prompt i simboli **$** e #
- il prompt e' modificabile a piacere

I comandi

Il comando che viene dato alla shell ha la seguente struttura:

 # cmd [-<flag>] [--<option>] [<parameter>]

dove:

- cmd rappresenta un programma da eseguire, o una funzione interna della shell
- flag e' una lettera preceduta dal segno – che modifica il comportamento del programma
- option, similmente, e' una parola, preceduta da un doppio - con le stesse funzioni della flag
- parameter e' un parametro che specifica l'azione del programma

I comandi (cont.)

- Quando si digita un comando, la shell guarda se il comando corrisponde ad una funzione interna; nel caso, la esegue (es.: cd)
- Altrimenti cerca un file eseguibile (programma) con quel nome in apposite directory; se c'e' lo esegue
- In entrambi i casi passa al programma tutti i parametri e le flag specificate in riga di comando
- Se non trova funzioni o programmi, la shell da un messaggio di errore (*command not found*)

I comandi (cont.)

- flag, options e parameter sono argomenti interpretati dal programma che viene eseguito
- usualmente i programmi possono accettare opzionalmente piu' flag o parametri (piu' raramente options) ed il loro significato dipende dal programma stesso.
- Esempi:

```
# ls /usr
# ls –l –r –t  /usr
# ls –lrt  /usr
# ls --help
```

stdin, stdout, stderr

- stdin (standard input): e' il posto da dove la shell legge l'input; normalmente coincide con l'input da terminale
- stdout (standard output): e' il posto in cui la shell invia l'output; normalmente coincide con la visualizzazione su terminale
- stderr (standard error): e' il posto in cui la shell invia i messaggi di errore; normalmente coincide con la visualizzazione su terminale

redirezione dell'I/O

- la shell interpreta alcuni caratteri posizionati in coda ai comandi per redirigere l'I/O da o verso file
- redirezione di stdin; si opera con il carattere **<** seguito dal nome del file; il comando invece di leggere l'input da terminale, lo legge dal file specificato:

```
# grep root < /etc/passwd
```

redirezione dell'I/O (cont.)

- redirezione di stdout: si opera con il carattere **>** seguito dal nome del file; il comando, invece di scrivere l'output sul terminale, lo scrive sul file specificato:

```
# ls /etc > listato.out
```

- attenzione: se esiste gia' un file con quel nome, il file verra' sovrascritto con il nuovo contenuto

redirezione dell'I/O (cont.)

- redirezione di stderr: si opera con il comando **2>** seguito dal nome del file; il comando, invece di scrivere i messaggi di errore sul terminale, li scrive sul file specificato:

```
# ls /etc/nonesiste 2> listato.err
```

- attenzione: se esiste gia' un file con quel nome, il file verra' sovrascritto con il nuovo contenuto

redirezione dell'I/O (cont.)

- redirezione di output in append: si opera con i comandi **>>** e **2>>;** in questo caso l'output (o l'errore) vengono "aggiunti" in coda al file; se il file non esiste, viene creato;

```
# ls /root >> listato.out
# ls /noncisono 2>> listato.err
```

redirezione dell'I/O (cont.)

- i comandi di redirezione sono utilizzabili insieme, per redirigere piu' di un canale di I/O:

```
# ls  /etc  /noncisono > ls.out 2> ls.err
```

pipeline

- la pipeline consiste nel reindirizzare l'output di un comando nell'input di un altro comando. Si opera tramite il carattere **|** posto tra i due comandi:

```
# ls /usr | grep lib
```

- la pipeline puo' essere reiterata:

```
# ls –F /usr/lib | grep gcc | wc -l
```

alias

- la shell permette di definire alias, vale a dire dei sinonimi di comandi:

```
# alias ll "ls –al"
# alias rm "rm –i"
```

- il comando *alias* fornisce in output tutti gli alias correntemente definiti

filename globbing

- la shell interpreta il carattere ***** inserito in un file name come "arbitraria sequenza di caratteri":

```
# ls /usr/bin/l*
# ls /etc/p*d
```

- la shell interpreta il carattere **?** inserito in un file name come "qualunque carattere":

```
# ls /dev/hd?
```

filename globbing (cont.)

- il file name ***** viene espanso in "qualunque file name" tranne che per i nomi che iniziano col punto (**.**)
- l'insieme di tutti i file (e le directory) il cui nome inizia per . e' identificabile con il nome **.***

variabili

- la shell e' capace di utilizzare variabili:

```
# a="/etc/passwd"              (sh, bash)
# set a="/etc/passwd"          (csh, tcsh)
# setenv  a  "/etc/passwd"     (csh, tcsh)
```

  Nell'esempio si e' definita una variabile di nome a e di valore /etc/passwd

- In assenza di spazi, i doppi apici sono opzionali.

espansione delle variabili

- definita una variabile, si istruisce la shell per utilizzare il valore della variabile tramite il carattere **$** seguito dal nome della variabile

```
# a="/etc/passwd"
# echo a
# echo ${a}
# cat $a
```

- In assenza di ambiguita', le parentesi graffe sono opzionali.

espansione dei comandi

- Esiste il modo per inserire un comando nel comando, tramite gli apici `

```
# echo "data: `date`"
```

Questo dice alla shell: prima esegui il comando *date*, quindi sostituisci a `date` l'output del comando, poi esegui il comando *echo*

carattere di escape

- I caratteri

  $ " ' ` ( ) { } % # & > < ! / \ ;

  hanno per la shell un significato particolare, e non possono normalmente essere utilizzati all'interno di una stringa
- per poter utilizzare questi caratteri senza che la shell li interpreti, si deve farli precedere dal carattere di escape ****

```
# a="variable"
# echo "$a non e' uguale a \$a"
```

carattere di escape (cont.)

- Qualora un comando sia troppo lungo per stare sulla riga, si puo' utilizzare il carattere di escape come ultimo carattere della linea per indicare alla shell che il comando prosegue sulla linea successiva:

```
# ls -lrt /usr /lib \
> /etc /bin > tutti.lis
```

Usualmente, quando la shell attende sulla linea successiva il completamento del comando, usa come prompt il carattere **>**

carattere di commento

- Nella shell tutto quello che segue il carattere **#** e' considerato un commento e non viene analizzato
- L'utilita' di questo carattere sta nella possibilita' di inserire commenti negli script (vedi sezione dedicata)

separatore di comandi

- E' possibile inserire piu' di un comando nella stessa linea, separando i comandi con il carattere **;**

```
# date; ls -l; echo OK
```

history

- la shell mantiene una memoria dei comandi dati in precedenza da un utente (history)
- per richiamare l'elenco dei comandi dati in precedenza, si utilizza il comando

  # history

- il numero dei comandi memorizzati puo' essere definito dall'utente

command line editing

- le shell piu' evolute (tcsh, bash) permettono di richiamare i comandi dati in precedenza, e di scorrerne l'elenco avanti ed indietro, utilizzando i tasti *arrow-up* e *arrow-down*
- con queste shell e' anche possibile spostare il cursore avanti ed indietro sulla linea di comando per modificarlo, utilizzando i tasti *arrow-left* ed *arrow-right*

script di startup

- quando la shell viene eseguita dal processo di terminale, questa esegue una serie di comandi che si trovano in file particolari chiamati script di startup
- questi script sono modificabili per configurare il comportamento della shell che si preferisce

script di startup (cont.)

- vi sono script di startup uguali per tutti gli utenti e modificabili solo dall'utente root; questi sono:

  /etc/profile (sh, bash)

  /etc/csh.login e /etc/csh.cshrc (csh, tcsh)

- ci sono poi script di startup specifici dell'utente, collocati nella sua home directory, e modificabili dall'utente stesso:

  .profile (sh, bash)

  .login e .cshrc (csh, tcsh)

command line completion

- con le shell piu' evolute (tcsh, bash) e' possibile chiedere alla shell di tentare di completare un comando scritto parzialmente, utilizzando il **TAB**
- se la shell determina in modo univoco il completamento, lo espande in corrispondenza della digitazione del TAB
- in caso di ambiguita' un doppio **TAB** (o un CTRL^D) ci mostrera' le possibili opzioni
- L'interpretazione - e quindi l'espansione - puo' riguardare nomi di comandi o nomi di files

la home directory

- Nelle shell piu' evolute, la sequenza di caratteri **~/** viene convertita nella home directory dell'utente:

```
# ls ~/.*sh*
```

Parte II.2

Caratteristiche specifiche

le shell

- sh (Bourne Shell): la prima (e si vede...)
- ksh (Korn Shell): introduce la history e gli array di variabili (vecchia anche lei)
- csh (C-shell): utilizza una sintassi leggermente differente, C-like (vecchia)
- tcsh (Turbo C-shell): evoluzione della csh, con history, editing di linea, command line completion

le shell (cont.)

- bash (Bourne Again Shell): e' la shell di default su linux; deriva dalla sh, con l'aggiunta di tutte le features gia' viste, e di altre. Estremamente comoda e flessibile. Problemi (dal manuale):

BUGS: It's too big and too slow

linea di comando

Nel seguito vedremo una serie di comandi utili per lavorare da linea di comando.

Lavorare con la shell ha diversi vantaggi:

- e' piu' rapido
- permette di accedere a tutti i programmi
- e' sempre disponibile

C'e' anche uno svantaggio:

- si deve sapere come fare cio' che si desidera

linea di comando (cont.)

Nel seguito vedremo comandi secondo la sintassi della bash.

Tuttavia quasi tutte le cose che vedremo valgono anche per le altre shell.

Verranno sottolineate le sintassi specifiche della bash (rispetto alla tcsh)

Parte II.3

Comandi generici

il manuale on line

- La cosa piu' importante che si deve imparare ad usare e' il manuale on line.
- Ogni comando, funzione, system call, file di configurazione, o funzionalita' di un programma, fornisce una pagina di manuale, richiamabile tramite il comando *man*
- La sintassi del comando e'

  # man <comando>

il manuale on line (cont.)

- Il manuale on line e' costituito da 8 sezioni differenti, ciascuna dedicata a diversi argomenti:
  - sezione 1: pagine di manuale sui comandi utilizzabili dall'utente
  - sezione 2: pagine si manuale sulle system call
  - sezione 3: pagine di manuale sulle funzioni della libreria del C
  - sezione 4: pagine di manuale sui device
  - sezione 5: pagine di manuale sul formato dei file di configurazione

il manuale on line (cont.)

- sezione 6: pagine di manuale sui giochi
- sezione 7: pagine di manuale su argomenti vari
- sezione 8: pagine di manuale sulla amministrazione di sistema

- Per richiamare la pagina di manuale di una sezione specifica il comando man deve essere seguito dal numero della sezione desiderata:

```
# man 1 printf
# man 3 printf
```

il manuale on line (cont.)

- Se non si conosce il comando esatto, si puo' utilizzare la flag **k** seguita da qualche parola relativa alla funzione che il comando svolge:

  # man –k print

  Si avra' in output l'elenco delle pagine di manuale che hanno attinenza alle parole inserite
- Un sinonimo di *man –k* e' il comando *apropos*

il manuale on line (cont.)

- Per vedere come puo' essere utilizzato man:

```
# man man
```

Si avra' la pagina di manuale relativa al comando man

scrivere su terminale

Il comando *echo*, seguito da una serie di parole, scrive su stdout la serie di parole, dopo aver eventualmente espanso i comandi e le variabili:

```
# echo Hello world
# echo "data: `date`" > data.out
# echo –n "la home dir e\' ${HOME}"
```

data e ora

Il comando *date* serve a visualizzare la data e l'ora, o a modificarle

```
# date
# date 10261137
```

E' possibile definire il formato della data in output:

```
# date +"%Y %m %d %H %M %S"
```

operazioni algebriche

- Il comando *expr* serve ad effettuare calcoli tra numeri interi

```
# expr 6 / 2
# count=10; count=`expr $count + 1`
```

- I valori numerici, o le variabili espanse, ed i simboli delle operazioni vanno separate da spazi
- Esistono *calcolatrici* piu' evolute (*bc*)

pausa della shell

- Il comando *sleep* seguito da un numero provoca la sospensione della shell per quel numero di secondi

```
# date; sleep 10; date
```

- Trascorso questo intervallo di tempo, la shell ritorna il prompt

trovare il comando

- Il comando *which* seguito dal nome di un comando serve a visualizzare il nome completo del programma che verrebbe eseguito in corrispondenza di quel comando:

```
# which ls
```

- Quando il comando corrisponde ad una funzione interna, la shell lo comunichera':

```
# which cd
```

Parte II.4

Comandi sul file system

spostarsi nel file system

- il comando *cd* serve a spostarsi in una directory

```
# cd /usr      # vai in /usr
# cd ..        # vai un livello sopra
# cd           # vai nella home dir
# cd -         # vai nella dir di prima
```

- Il comando *pwd* visualizza la current working directory

```
# cd /usr/lib; pwd
```

visualizzare il contenuto di una directory

- il comando *ls* serve a visualizzare il contenuto di una directory

```
# ls              # lista la current dir
# ls /home        # lista la dir /home
```

- sono disponibili numerose flag

```
# ls –l           # mostra le proprieta'
# ls –a           # mostra tutti i files
# ls --color      # lista i nomi usando colori
```

creare e rimuovere una directory

- il comando *mkdir* serve a creare una directory

```
# mkdir dir1
# mkdir dir1/dir2
```

la flag **-p** provoca la creazione di tutto il sottoalbero in un colpo (se necessario)

```
# mkdir –p dir1/dir2/dir3/dir4/dir5
```

- il comando *rmdir* serve a rimuovere una directory

```
# rmdir dir3/dir4  # rimuove solo dir4
```

la directory verra' rimossa con rmdir solo se e' vuota

trovare un file

- i comandi *locate* e *find* servono a trovare la posizione di uno o piu' file

```
# locate passwd
# locate lib
```

- find ha una sintassi molto piu' complessa

```
# find /usr –name "*libm*.a" -print
```

obbligatorio studiarne la man page

creare un file

- il modo piu' semplice per creare un file e' dare un comando che genera output, redirigendo stdout in un file

```
# echo OK > nuovo_file
```

- si puo' utilizzare il comando *touch*, seguito dal nome di un file inesistente: crea un file vuoto con quel nome

```
# touch /tmp/nonesiste
```

rinominare e rimuovere un file

- il comando *mv* serve a rinominare un file o una directory

```
# mv file1 file2
```

questo comando cambia il nome di file1 in file2;

- il comando *rm* serve a rimuovere un file

```
# rm file*
# rm –i file*    # chiede conferma
```

osservazione su mv

- poiche' non esiste il concetto di "extension" di un file, il comando

```
# mv *.lis *.txt
```

non funziona. Per fare questo il comando corretto e':

```
# for old in `ls`; do
> new=`echo $old | sed s/\.lis$/\.txt/`
> mv $old  $new
> done
```

... a volte la shell non e' semplicissima

rimozione recursiva

- la flag **-r** applicata a rm su una directory provoca la rimozione di tutto il sottoalbero

```
# rm -r ~/*
```

questo comando rimuove tutti i file e tutte le directory dalla home directory

visualizzare un file

- il comando *cat* serve a visualizzare sul terminale il contenuto di un file

```
# cat /etc/passwd
```

- i comandi *more* e *less* permettono di scorrere il contenuto di un file avanti ed indietro

```
# less /etc/profile
```

Sono quasi editori di testo

ordinare il contenuto di un file

- il comando *sort* serve a ordinare sul terminale il contenuto di un file

```
# sort /etc/passwd
```

- dispone di flag che permettono di ordinare il file selezionando i campi (secondo un separatore) o il criterio di ordinamento

```
# sort -t: -r -n -k3,4 /etc/passwd
```

questo comando ordina al contrario (-r) con criterio numerico (-n) sui campi 3 e 4 (-k3,4) usando come separatore il carattere **:**

cercare in un file

- il comando *grep* serve a ricercare in un file le righe che contengono una data stringa

```
# grep root /etc/passwd
```

- dispone di flag che permettono di selezionare diversi criteri di ricerca

confrontare file

- il comando *diff* serve a cercare le differenze tra due file

  # diff file1 file2

- l'output non e' di facile comprensione, ma spesso viene utilizzato solo per verificare che due file siano uguali: nel caso non c'e' output.

contare righe, parole, caratteri

- il comando *wc* permette di effettuare un conteggio delle righe, delle parole e dei caratteri contenuto in un file

```
# wc /etc/profile      # conta tutto
# wc -l /etc/profile   # conta le righe
```

per i piu' temerari...

- cut: programma che permette di selezionare e visualizzare solo alcuni campi delle riche di un file
- awk: come cut, ma piu' flessibile e piu' complicato da usare
- sed: programma che permette di modificare una stringa di caratteri; richiede eroismo

andarsi a vedere le man pages...e non solo!

modificare le caratteristiche dei file

- chmod: modifica le protezioni di un file
- chown: modifica il proprietario del file
- chgrp: modifica il gruppo del file
- touch: modifica il tempo di ultimo accesso al file

... andarsi a vedere le man pages...

occupazione disco

- il comando *df* serve a visualizzare l'occupazione dei file system

  ```
  # df <directory o file>
  ```

  mostra l'occupazione del file system che contiene la directory o il file specificato

  ```
  # df
  ```

  mostra l'occupazione di tutti i file system montati

occupazione disco (cont.)

- il comando *du* serve a visualizzare lo spazio occupato da un sottoalbero

  ```
  # du -s <directory>
  ```

  mostra l'occupazione di tutto il sottoalbero a partire dalla directory specificata (la flag **-s** specifica solo il totale: senza la flag mostra l'occupazione di ciascun file)

occupazione disco (cont.)

- il comando *quota* serve a visualizzare lo spazio occupato da tutti i file di uno user

```
# quota
```

La quota occupata verra' mostrata solo se il sistema e' configurato per controllare le quote (tipico di server di login centrale, usualmente non utilizzato su macchine personali).

Parte II.5

Comandi di job control

controllare i processi

- quando si fornisce un comando alla shell, questa crea un processo che esegue il programma corrispondente
- e' possibile eseguire piu' programmi contemporaneamente
- e' possibile controllare lo stato dei processi creati, metterli in pausa o farli terminare

foreground e background

- un processo si dice *in foreground* quando possiede il controllo del terminale;

  quando si esegue un comando in foreground, non si ha disponibile il prompt fino al termine della sua esecuzione
- un processo si dice *in background* quando viene eseguito in modo da lasciare il controllo del terminale alla shell;

  in questa condizione l'utente puo' lanciare un altro processo, che sara' eseguito contemporaneamente

esecuzione in foreground

- normalmente l'esecuzione di un comando provoca la creazione di un processo in foreground
- il prompt non e' disponibile fino alla fine della esecuzione del processo
- e' normalmente possibile interrompere definitivamente l'esecuzione di un processo in foreground utilizzando la combinazione di tasti CTRL ^C

esecuzione in foreground (cont.)

- e' possibile interrompere temporaneamente l'esecuzione di un processo in foreground tramite la combinazione di tasti CTRL^Z
- in questa condizione il processo si blocca, e la shell ritorna il prompt
- un processo in pausa puo' essere rimesso in esecuzione in foreground tramite il comando

```
# fg
```

esecuzione in background

- quando il comando che viene digitato viene terminato con il simbolo **&**, la shell eseguira' il programma corrispondente in background
- in queste condizioni, la shell vi offrira' il prompt immediatamente

esecuzione in background (cont.)

- e' possibile mettere in background un processo attualmente in foreground con i seguenti passi:
  - mettere in pausa il processo tramite CTRL^Z
  - digitare il comando:

```
# bg
```

controllo dei sottoprocessi

- i processi in background fatti partire da una sessione di shell (sottoprocessi o job) possono essere visualizzati con il comando

  # jobs

- ogni sottoprocesso e' identificato con un numero (il job-id), a partire da 1
- e' possibile riportare in foreground un job attualmente in background con il comando

  # fg <job-id>

  dove job-id e' il numero identificativo del job
- Il comando jobs mostra solo i sottoprocessi della shell corrente

controllo di tutti i processi

- i processi in esecuzione sul sistema possono essere visualizzati tramite il comando ps:

```
# ps -e
```

- ogni processo ha un identificativo numerico (il PID: Process IDentifyer)
- attenzione a non confondere il PID con il job-id (identificativo dei sottoprocessi di una shell)

signal

- linux permette di inviare ad un processo 32 diversi *messaggi* (signals), tramite il comando kill:

  # kill -<signal-id> %<job-id>

  # kill -<signal-id> <PID>

- il signal-id e' un numero che identifica il signal che si vuole inviare (da 1 a 32)
- il job-id e' il numero che identifica il job (PID e' il numero che identifica il processo) a cui si vuole mandare il signal

signal (cont.)

- l'elenco dei segnali inviabili ad un processo, ed il loro significato, sono elencati nella man page di signal, sezione 7:

```
# man 7 signal
```

- esempi:
  - il CTRL^C equivale al segnale 2 (SIGINT)
  - il CTRL^Z equivale al segnale 17 (SIGSTOP)
  - il segnale 9 (SIGKILL) provoca la brutale terminazione del processo

top

- esiste una utility usualmente presente nelle installazioni di linux, che permette di visualizzare l'elenco dei processi e la loro occupazione di risorse:

  ```
  # top
  ```

  top permette all'amministratore del sistema di monitorare ed intervenire sui processi

Esecuzione ritardata

- E' possibile dire al kernel di eseguire un comando ritardando il momento di esecuzione di un intervallo di tempo definito, o farlo eseguire in un certo momento, tramite il comando *at*:

  # at -f <programma> <time>

- L'indicazione del tempo puo' essere
  - assoluta: HH:MM
  - relativa: + 3 hours
  - keyword: now, midnight, noon, teatime

Esecuzione ritardata (cont.)

- Si possono visualizzare i comandi sottomessi tramite *at* con il comando

  # atq

  Si otterra' in output la lista dei comandi dello user in attesa di esecuzione
- Si possono rimuovere i comandi sottomessi tramite *at* con il comando

  # atrm <job>

  Dove <job> e' l'indentificativo del comando sottomesso, visualizzato da *atq*

Esecuzione ripetuta

- E' possibile dire al kernel di eseguire un comando (generalmente uno script) ad intervalli di tempo regolari, tramite il programma cron
- Si modifica la lista dei comandi da eseguire ciclicamente tramite:

```
# crontab -e
```

- Questo fara' partire un editore (per default vi) con cui inserire, rimuovere o modificare i job da eseguire ciclicamente

Esecuzione ripetuta (cont.)

- Per visualizzare la lista dei comandi eseguiti ciclicamente si utilizza:

  ```
  # crontab -l
  ```

- Per rimuovere completamente la lista:

  ```
  # crontab -r
  ```

- La sintassi (non banalissima) per specificare quando eseguire i comandi va vista nella man page di crontab

Esecuzione a bassa priorita'

- E' possibile dire al kernel di eseguire un comando a priorita' diversa da quella della shell corrente, tramite il comando *nice*:

  # nice -n <priority> <command>

- <priority> e' un numero che va da -20 (alta) a 19 (bassa); il default e' 0
- Solo root puo' sottomettere comandi a priorita' negativa

Parte II.6

L'environment

environment di un processo

- Per environment di un processo si intende l'insieme di una serie di variabili e del loro valore, che vengono associate al processo
- Il programma in esecuzione puo' accedere al valore di queste variabili in qualsiasi momento
- Il programma puo' modificare l'environment, definendo nuove variabili, rimuovendole o modificandone il valore

environment di un processo (cont.)

- Quando un processo genera un altro processo (ad esempio quando la shell esegue un comando), l'environment del processo padre (la shell) viene passato al processo figlio (il processo che esegue il programma corrispondente al comando)

environment della shell

- Come gia' visto, la shell dispone della capacita' di utilizzare variabili:

```
# lavoro="~/work"
```

- E' possibile rendere una variabile parte dell'environment (cioe' delle variabili che verranno esportate ai sottoprocessi della shell):

```
# export lavoro
```

environment della shell (cont.)

- E' possibile esportare una variabile all'atto della sua definizione:

```
# export lavoro="~/work"
```

- L'elenco delle variabili definite si ha con:

```
# set
```

- L'elenco delle variabili esportate, cioe' l'environment, si ha con:

```
# env
```

environment della shell (cont.)

- E' possibile rimuovere una variabile dall'environment tramite il comando *unset*:

```
# unset lavoro
```

- Il comando *unset* elimina anche la definizione della variabile, non solo la sua appartenenza all'environment

variabili in csh e tcsh

- In csh (e tcsh) i due insiemi di variabili e variabili di environment sono separati:

```
# set lavoro="~/work"
# setenv lavoro "~/work"
```

- Per rimuovere le variabili dai due insiemi si usano comandi diversi:

```
# unset lavoro
# unsetenv lavoro
```

variabili in csh e tcsh (cont.)

- L'elenco delle variabili definite si ottiene come per la bash:

```
# set          # variabili
# env          # environment
```

Principali variabili di environment

- HOME: indica la home directory
- HOSTNAME: il nome del calcolatore
- USER: lo username dell'utente
- TERM: il tipo di terminale utilizzato
- HISTSIZE: la lunghezza della history
- DISPLAY: il display utilizzato dai client X

Principali variabili di environment (cont.)

- PATH: l'elenco delle directory in cui la shell cerca i programmi da associare ai comandi, separate dal simbolo **:**

```
# echo $PATH
/home/brunengo/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/X11R6/bin:/cern/pro/bin:/usr/local-ge/bin:/usr/local-ge/mathematica/bin:/usr/local/root/bin
```

Principali variabili

- PS1: il valore di questa variabile viene espanso ed utilizzato come prompt principale; es:

```
# PS1="`hostname`> "
# PS1="\u@\H:\w> "
```

- PS2: il valore di questa variabile viene espanso ed utilizzato come prompt secondario (quando la shell aspetta ulteriore input)

Parte II.7

Lo scripting

Lo script

- Uno script e' un file ASCII costituito da una serie di comandi
- Se il file e' eseguibile (permission x attivata) puo' essere eseguito come un qualsiasi programma
- In esecuzione il file viene letto riga per riga, ed ogni riga processata come se fosse un comando dato in linea

L'interprete dello script

- Il programma che deve interpretare lo script deve essere specificato nella proma riga dello script:

  #!<interprete>

  Ad esempio:

  #!/bin/bash
- Se non specificato, l'interprete utilizzato sara' la shell di login
- La shell e' solo uno degli interpreti disponibili

Interpretazione forzata

- E' possibile dire alla shell di eseguire una serie di comandi scritti su un file anche se questo non e' eseguibile, tramite i comandi:

```
# source comandi.lis
# .  comandi.lis
```

- In questo caso l'interprete utilizzato sara' la shell correntemente usata

Istruzioni di controllo di flusso

- La shell dispone di istruzioni per controllare il flusso dei comandi da eseguire in funzione di condizioni
- Questo, unitamente alla possibilita' di definire subroutine, fa della shell un vero e proprio linguaggio di programmazione

Operatore di test

- Le condizioni vengono controllate tramite l'operatore **[ ]** detto test
- La sintassi dell'operatore e'

  [<condizione>]

  Questa operazione ritornera' vero o falso in base alla condizione

Operatore di test (cont.)

- Le condizioni possono essere:
  - confronto tra stringhe (=, !=, >, <)

    ["str1" = ${variabile}]
  - confronto numerico (-eq, -lt, -gt, -neq)

    [${count} -lt 100]
- Vedere la man page della bash

Esecuzione condizionale

- L'esecuzione condizionale si ottiene tramite l'istruzione *if*

```
if [ <condizione> ]
then
    <istruzioni>
else                        # opzionale
    <istruzioni>            # opzionale
fi
```

cicli

- E' possibile ripetere piu' volte un blocco di istruzioni tramite l'istruzione *for*

```
for var in val1 val2 val3
do
    echo ${var}
done
```

In questo esempio l'istruzione echo verra' eseguita una volta per ogni parola che segue *in*; in ogni ciclo la variabile val avra' di volta in volta il valore delle parole val1 val2 val3

cicli (cont.)

- Si puo' utilizzare come elenco di valori del ciclo l'output di un comando, tramite l'espansione data dagli apici inversi:

```
for var in `ls /usr/lib`
do
    echo "libreria: ${var}"
done
```

In questo esempio l'istruzione *echo* verra' eseguita una volta per ogni file in /usr/lib. La variabile var avra' via via per valore il nome dei file

cicli (cont.)

- Un ciclo puo' essere realizzato anche tramite l'istruzione while:

```
while [ <condizione> ]
do
    <istruzioni>
done
```

- Il ciclo verra' ripetuto finche' la condizione sara' vera

interruzione dei cicli

- Esistono due istruzioni che interrompono l'esecuzione di un ciclo

  continue

  L'istruzione *continue* provoca l'interruzione del blocco di istruzioni del ciclo, per proseguire col ciclo successivo

  break

  L'istruzione *break* provoca l'interruzione e l'uscita dal ciclo, per proseguire con le istruzioni successive al ciclo

subroutine

- In bash possibile definire subroutine, cioe' blocchi di istruzioni che possono essere eseguiti con un comando che li richiama:

```
sub1()
{
    <istruzioni>
}
```

- Queste istruzioni definiscono una subroutine di nome sub1

subroutine (cont.)

- Definita una subroutine, si attiva l'esecuzione delle istruzioni che la compongono semplicemente digitando il suo nome, come se fosse un comando comune:

```
echo "Inizio subroutine sub1"
sub1
echo "Fine subroutine sub1"
```

parametri

- Qualunque script e' capace di ricevere parametri: basta richiamare lo script facendo seguire al suo nome l'elenco dei parametri che si desidera passargli:

```
# ./script.sh par1 par2 par3
```

Con questo comando si chiama in esecuzione lo script *script.sh* passandogli come parametri le tre stringhe par1, par2 e par3

parametri (cont.)

- All'interno di uno script si hanno a disposizione delle variabili per trattare gli eventuali parametri passati allo script:
  - $#: numero di parametri passati
  - $0: nome con cui e' stato eseguito lo script
  - $1: parametro in posizione 1
  - ...
  - $9: parametro in posizione 9

parametri (cont.)

- Ad esempio, lo script:

```
#!/bin/bash
echo "nome: $0"
echo "parametri: $1, $2, $3"
```

chiamato con il comando

```
# ./script.sh a 35 /etc/passwd
```

produrra' come output:

```
nome: ./script.sh
parametri: a, 35, /etc/passwd
```

parametri alle subroutine

- Anche le subroutine possono essere chiamate con parametri
- Il meccanismo di funzionamento e' lo stesso dello script:
  - la subroutine viene chiamata facendo seguire al suo nome la lista dei parametri
  - all'interno della subroutine, i parametri saranno accessibili tramite le variabili $1, $2...

Parte III

X-Windows

In principio era la SHELL....

- Agli albori Linux forniva solamente un'interfaccia utente testuale, capace di interpretare tutti i comandi necessari all'utilizzo del sistema (shell).
- La shell e' utile ma poco comoda, specialmente per utenti poco esperti....

L'interfaccia grafica

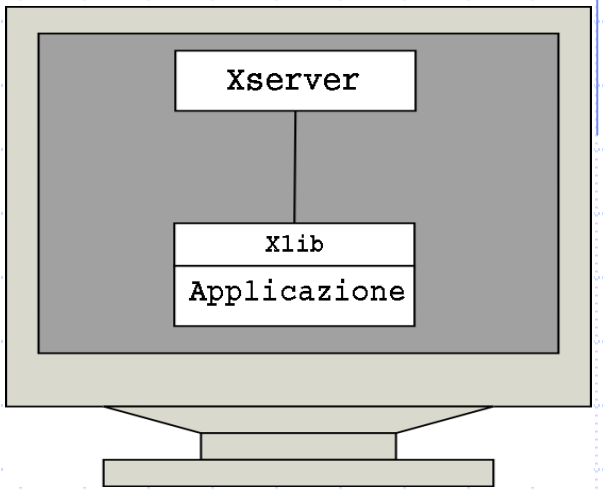
- Nasce l'esigenza di adottare un sistema che gestisca l'hardware grafico dei PC e fornisca un'interfaccia piu' comoda.
- Viene deciso di introdurre l' X Window System (o piu' semplicemente X11), il sistema di gestione della grafica dei sistemi UNIX; in particolare XFree86, una sua implemetazione opensource.

Cos'e' X11?

- E' un insieme di programmi che, seguendo le specifiche determinate dal protocollo X11R6 consentono l'utilizzo di device quali: scheda video, monitor, mouse, tastiera, ecc...
- In parole povere permette di gestire un sistema grafico a finestre come Windows o Macintosh ma offrendo una maggiore flessibilita'.

Come funziona X11

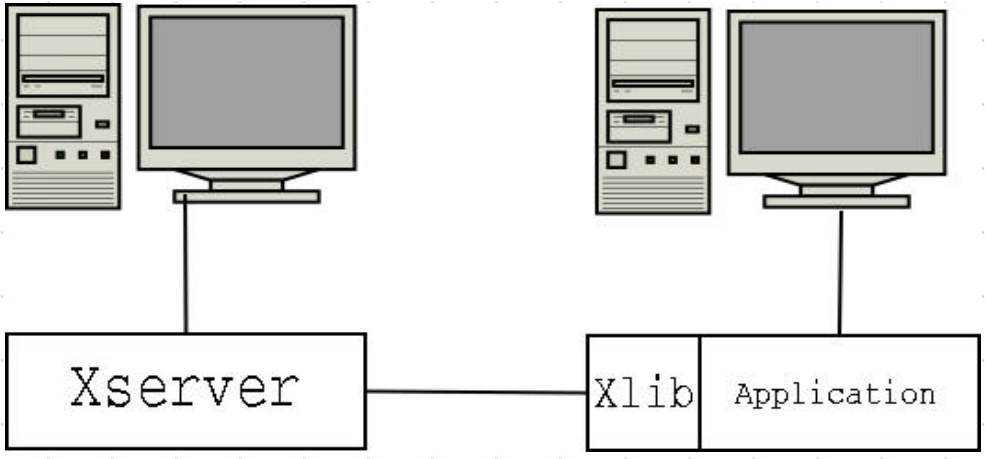
- Architettura client/server



- Il “server” si occupa di gestire i device attraverso i driver.
- Ogni applicazione e' un client.

Come funziona (continua)

- In questo modo un'applicazione si puo' trovare anche su una macchina diversa da quella sulla quale gira il server.



Il window manager

- E' una speciale applicazione (client dell' X server) che si occupa di gestire gli oggetti sullo schermo (aprire, spostare e ridimensionare finestre, visualizzare oggetti)
- E' indispensabile per poter utilizzare X
- Ne esistono vari tipi: twm, fvwm, blackbox, sawfish, metacity....
- Da solo ancora non permette un utilizzo veramente completo del sistema

Il desktop environment

- E' un insieme di programmi che rendono facilmente fruibili le risorse del PC attraverso un'interfaccia grafica.
- Di solito e' composto da alcuni elementi fondamentali:
  - Un pannello
  - Un file manager
  - Una scrivania (desktop appunto)

I desktop environment per Linux

- Esistono diversi tipi di desktop environment per il sistema operativo Linux.
- Due sono i piu' utilizzati
  - GNOME
  - KDE

GNOME (il file browser)

- Gnome mette a disposizione un file browser chiamato Nautilus che permette di:
  - Navigare attraverso file e directory
  - Vedere le immagini in anteprima
  - Visualizzare file di testo
  - Visualizzare html
  - Riprodurre file sonori
  - ……

GNOME (il pannello)

- Il pannello di GNOME (gnome-panel) permette di:
  - Lanciare applicazioni
  - Gestire le finestre iconizzate
  - Selezionare desktop
  - .....

KDE

- KDE (K Desktop Environment) mette a disposizione i medesimi elementi di GNOME. Il file browser (Konqueror) ed il pannello hanno le stesse funzionalita'.
- Konqueror e' anche web browser (come Netscape o IE) di buona qualita'.

KDE o GNOME ?

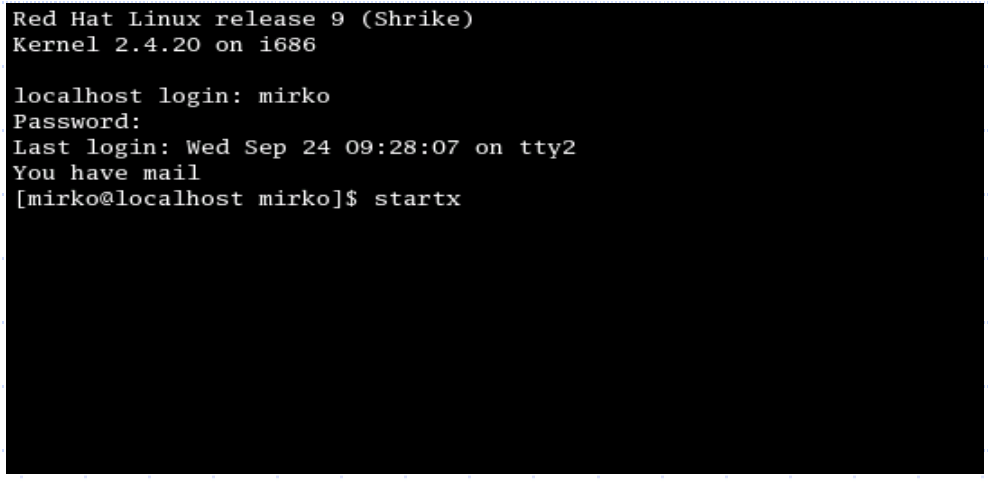
- Molto dipende dal gusto personale e dallo stato dei due progetti.
- KDE e' piu' personalizzabile, sono disponibili in rete numerosi temi del desktop.
- GNOME e' piu' flessibile, i suoi componenti possono essere eseguiti al di fuori dell'ambiente.

Come faccio partire il mio desktop?

- All'avvio il sistema Linux puo' essere configurato per partire in modalita' grafica o in modalita' testo.
- In entrambi i casi, per far partire il desktop environment dobbiamo eseguire alcune operazioni.

Modalita' testo

- Per far partire X in modalita' testo, dopo aver eseguito il login, e' necessario digitare il comando 'startx' seguito dal tasto 'enter':



Modalita' grafica

- In modalita' grafica X parte automaticamente. La prima schermata che viene presentata e' quella di login.
- Selezionare il Desktop Manager preferito.
- Digitare username e password.
- Premere il tasto 'Login' o 'enter' sulla tastiera.

Si parte!

- Compare lo spash screen:

- E (se tutto va come deve...) si parte!

Configurazione di X

- Se X parte correttamente esistono configuratori grafici (diversi a seconda della distribuzione di Linux).
- E' necessario avere privilegi di amministratore.
- Per RedHat il tool si raggiunge dal pannello di GNOME o KDE oppure digitando 'redhat-config-xfree86' da console.
- E possibile cambiare la risoluzione dello schermo e/o la frequenza di refresh del monitor.

Configurazione (continua)

- E' possibile inoltre configurare l'aspetto del nostro desktop, del file browser e del pannello.
- KDE e GNOME mettono a disposizione strumenti di configurazione del 'look and feel'.
- Il modo di rimuovere una errata configurazione del desktop e' cancellare tutte le directory che iniziano con '.gnome' o '.kde' all'interno della propria home.

E se ci sono problemi?

- Legge di Murphy: “Se qualcosa puo' andar male, lo fara'”
- Se X non parte e' necessaro agire attraverso l'interfaccia testuale. Per la distribuzione RedHat digitare il comando 'redhat-config-xfree86'.
- Parte un'interfaccia grafica minimale che contiene una finestra di configurazione.

Risoluzione problemi (continua)

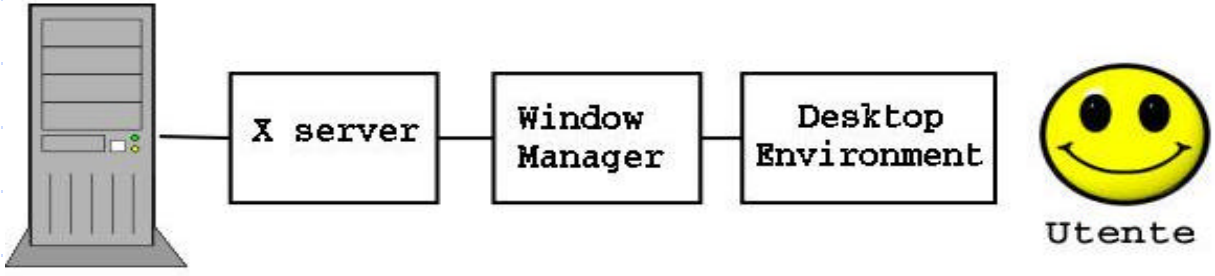
- Raccogliere tutti i dati dell'hardware a disposizione:
  - Modello scheda grafica e quantita' di memoria della scheda
  - Modello dello schermo.
  - Risoluzione ottimale dello schermo e frequenza di refresh verticale
- Se non si conoscono i dati il consiglio e':
  - Scheda svga con 4Mb di memoria
  - Risoluzione 1024x768 alla frequenza di 60Hz

Risoluzione problemi (continua)

- Corollario alla legge di Murphy: “Niente e' facile come sembra”.
- Se X persiste nel non voler funzionare correttamente potrebbe essere un problema di driver. A questo punto l'unica soluzione e' cercare su Internet il driver per la scheda di grafica installata sul PC.

In breve....

- Linux e' dotato di un'interfaccia grafica.
- Le componenti principali di questa interfaccia sono:
  - X server: gestione dell'hardware
  - Window Manager: gestione degli oggetti grafici
  - Desktop Environment: interazione utente



In breve.....(continua)

- I Desktop Environment piu' utilizzati sono GNOME e KDE che possiedono tutte le caratteristiche per rendere il sistema Linux 'amichevole' anche per i meno esperti.
- X puo' partire automaticamente all'avvio oppure puo' essere avviato dall'utente.
- La configurazione di X avviene attraverso applicazioni differenti a seconda della distribuzione Linux