Prima Esercitazione

GNU/Linux e
linguaggio C

Stefano Monti
smonti@deis.unibo.it

Unix e GNU/Linux

Unix: sviluppato negli anni '60-'70 presso Bell Labs di AT&T, attualmente sotto il controllo del consorzio The Open Group

- **UNIX certified**: sistemi fully-compliant (IBM AIX, Solaris 10, ...)
- **Unix-like**: sistemi NON fully-compliant (GNU/Linux, BSD, ...)

GNU (**G**NU's **N**ot **U**nix)

- progetto annunciato da Stallman '83; obiettivo la creazione di un sistema operativo **free Unix-like**
- obiettivo raggiunto nel '92, con l'**adozione del kernel Linux**

Linux

- primo kernel rilasciato nel 1991 ad opera di Torvalds
- successivamente, adozione di software dal progetto GNU
- attualmente **kernel ufficiale Linux** nome in codice **Vanilla**

Distribuzioni GNU/Linux

Attualmente varie **distribuzioni GNU/Linux** (comunemente *distro*):

- personalizzazione del **kernel** *vanilla*
- collezione di **pacchetti** (applicativi) **software**: archivi compressi usati per **automatizzare** e **semplificare** l'installazione di applicazioni (compilazione dei sorgenti, impostazione delle variabili di ambiente, configurazione di permessi, ecc...)
- alcuni esempi: Redhat/Fedora, Slackware, Debian, Gentoo, Ubuntu, SUSE, ecc...

Gestori di pacchetti: sono pacchetti a loro volta

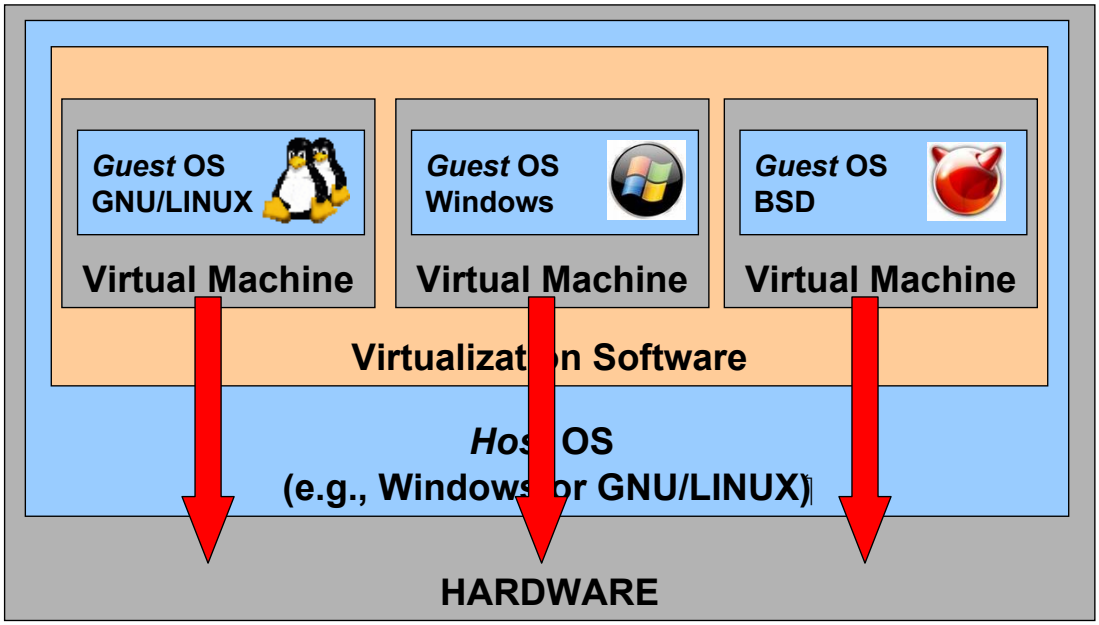
- differenti per *famiglie* di distribuzioni (RPM, APT, Portage,...)
- operazioni di installazione, rimozione, aggiornamento di pacchetti software
- gestione dipendenze tra pacchetti

Ai fini del corso...

Necessità di utilizzare un sistema operativo **Unix-like**; varie possibilità:

- **Installazione** di una distribuzione Linux su una macchina fisica:
  - maggiore apprendimento ma complessità e problematiche maggiori (partizionamento del disco fisso, dual booting, ecc...)
- Uso distribuzioni **Linux Live CD**
  - nessuna installazione, ambiente di lavoro “stateless” e ripetibile caricato in RAM, elevati requisiti hardware, prestazioni penalizzate
- **Virtualizzazione**
  - installazione necessaria ma priva di implicazioni per la macchina fisica (configurazione dual boot, partizionamento del disco), prestazioni ragionevoli

Virtualizzazione



VMWare

Software di virtualizzazione largamente diffuso

- diversi prodotti, sia commerciali che gratuiti
- per piattaforme Windows e GNU/Linux

  http://www.vmware.com/products/free_virtualization.html
- **VMWare Player**: esecuzione di macchine virtuali esistenti
- **VMWare Server**: creazione e configurazione di nuove macchine virtuali
- **Virtual Appliances Marketplace**: repository di macchine virtuali "preconfezionate" (ad es., installazioni di base delle maggiori distribuzioni)
- **VMWare Converter**: creazione di macchine virtuali a partire da macchine fisiche (solo per piattaforma Windows) già esistenti

Shell - comandi di base

- `man <nome_comando>`
  - manuale del comando e opzioni
- `cd <nome_directory>`
  - Permette di spostarsi all'interno del file system
  - Posizioni relative:
    - . (direttorio corrente) e .. (direttorio padre)
- `mkdir <nome_nuova_directory>`
  - Creazione di un nuovo direttorio
- `ls <parametri>`
  - Lista il contenuto del direttorio corrente
- `emacs` (o `vi`) `<nome_file>`
  - Programmi per l'editing; se il file non esiste, lo creano

Compilazione sorgenti - C

- Comando `gcc <file>`
  - Compilatore C e C++
  - Compila `<file>` producendo il **file eseguibile** `a.out`
  - Per dare un nome diverso al file prodotto opzione `-o`
- Es: `gcc file_exec.c -o f_ex`
- Esecuzione: `./f_ex <parametri>`

Compilazione sorgenti - Java

- Comando `javac <file>.java`
  - Compila il sorgente `<file>.java` producendo il **file bytecode** `<file>.class`
  - possibilità di wildcard e/o indicazione esplicita di più file sorgenti (es. "*`javac *.java`*" oppure "*`javac file1.java file2.java ...`*")
  - parametro -d per esplicitare la directory di output
- Comando `java <file> [param1 ... paramN]`
  - **interpreta (esegue) il bytecode** contenuto in `<file>.class` sulla macchina virtuale Java
- Es: `javac MyProgram.java`
- Esecuzione: `java MyProgram myparam1 4`

Esercitazione 1- Obiettivi

Programmazione C

- Gestione dei parametri in ingresso
  - argomenti ***argc*** ed ***argv***
  - controllo di correttezza dei parametri in ingresso
- Gestione delle stringhe
  - libreria string.h

Esercitazione 1 – Testo (1/2)

Si realizzi un programma C che abbia un'interfaccia del tipo

listaTreni treno1 .... trenoN

e che prenda in ingresso un numero arbitrario di stringhe rappresentanti il codice di un treno, nel formato:

<TIPOTRENO><NUMEROTRENO>

- <TIPOTRENO> stringa di due caratteri che rappresenta il tipo di treno (per semplicità, si assuma che abbia i valori "IC", "ES", "RG", rispettivamente per le tipologie Intercity, Eurostar e Regionale)
- <NUMEROTRENO> identificativo numerico univoco del treno di 4 cifre

Esercitazione 1 – Testo (2/2)

Il programma deve:

- controllare che sia stato passato ***almeno un treno***
- controllare che ogni codice passato sia ***conforme alle caratteristiche*** sopra indicate (in particolar modo, rispetti la lunghezza di 6 caratteri)
- stampare a video i soli identificativi dei treni, ***raggruppati per categoria***