
Benvenuti alla prima lezione per la provincia di Massa Carrara del **progetto TAG** (Toscana Area Giovani).

Questo progetto, promosso principalmente dall' **UPI Toscana** (Unione delle Provincie Italiane) in collaborazione con la **Fondazione Sistema** Toscana ed il **Corecom**, promuove la diffusione fra pari di conoscenze inerenti il mondo della teconologia dell'informazione.

In questo incontro parleremo di software libero e faremo un'introduzione a Debian, il sistema operativo basato su Linux libero per eccellenza.

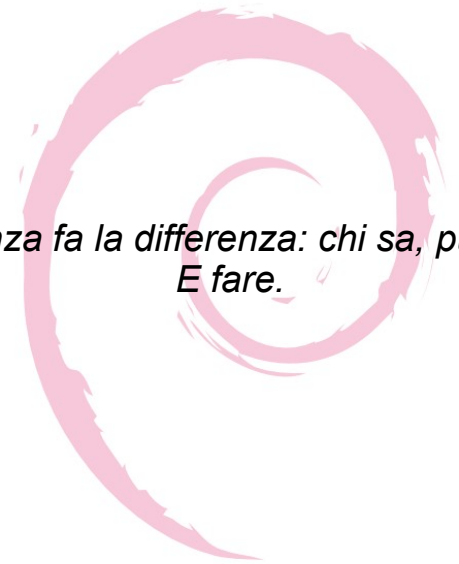
Prima di iniziare vorrei ringraziare l'UPI Toscana, l'amministrazione dell'assessorato alle politiche giovanili della Provincia di Massa Carrara, il Corecom e la Fondazione Sistema Toscana, grazie ai quali è stato possibile far nascere ed organizzare questo progetto.

# Programma

- Introduzione
- Concetti di base
- Installazione
- Primi passi
- Tutto è un file
- Applicativi
- Rete
- Amministrazione
- Customizzazione


*http://www.filippomicheletti.it/tag/fosdinovo/presentazione.odp*
*http://www.filippomicheletti.it/tag/fosdinovo/slides.pdf*
*http://www.filippomicheletti.it/tag/fosdinovo/notes.pdf*

## Introduzione



*La conoscenza fa la differenza: chi sa, può scegliere.*
*E fare.*



Il mondo dell'informatica è vastissimo e si evolve con velocità e dimensioni inimmaginabili.

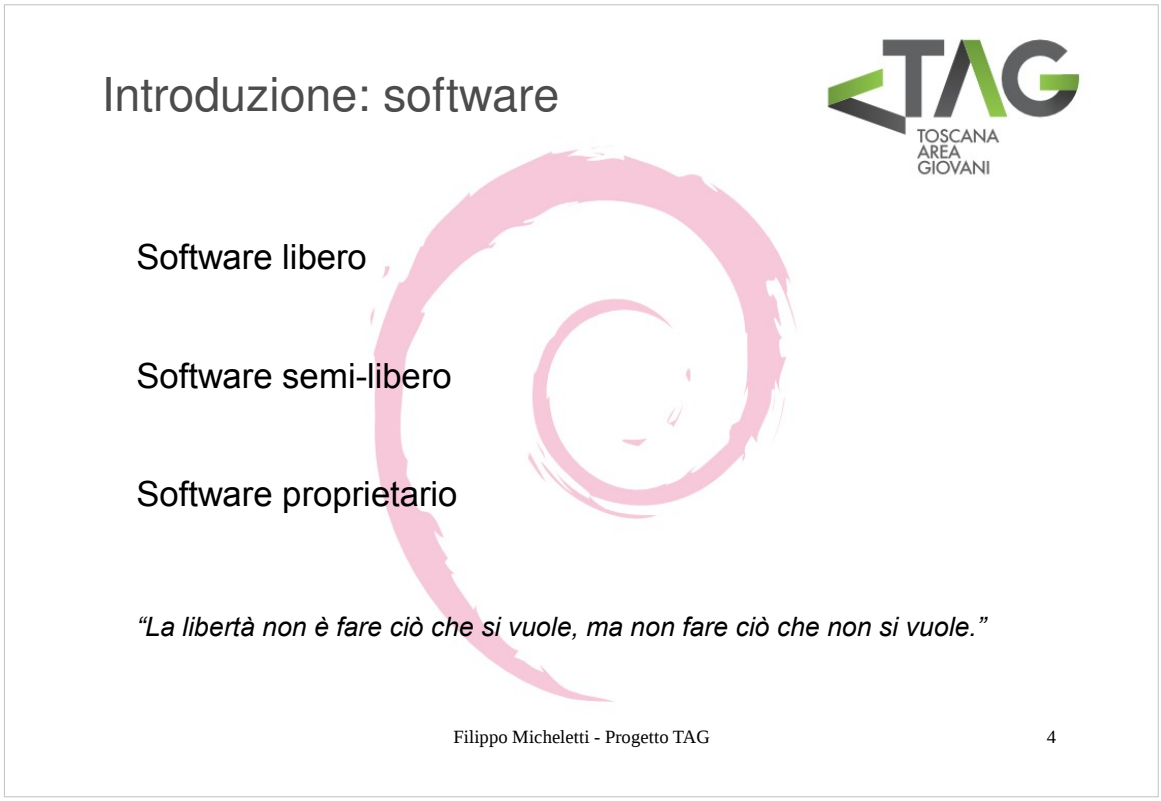
L'unico modo per poter essere "al passo coi tempi" è abituarsi a pensare in maniera flessibile, imparando a percorrere strade ignote che spesso, fino a poco prima, si pensavano sbagliate.

Il mondo dell'informatica (e dell'informazione) è inoltre un mondo in cui convivono aspetti contrapposti, dove i fenomeni che vanno per la maggiore possono avere dietro gigantesche operazioni commerciali (Facebook) così come autosostenersi dall'interesse e la passione di chi le realizza (Linux), dove società ricchissime possono offrire servizi eccezionali gratuitamente (Google) o tentare di raccogliere cerchie di utenti assoggettati da una fede quasi religiosa (Apple), in ogni caso tutto ciò che ha davvero successo nasce dallo stesso seme: la creatività.

E proprio da idee creative ed innovative sono nati progetti come Linux, GNU e Debian.

Nel mondo informatico non esiste solo Linux e sarebbe sbagliato ignorare gli altri Sistemi Operativi.

NON esiste un OS migliore degli altri, esiste quello più adeguato alle proprie necessità e quello sul quale si riesce a fare meglio e più in fretta quello che si deve fare.




Software
Formalmente un **software** è un'insieme di istruzioni per una determinata macchina.
Le istruzioni, scritte in "linguaggio macchina" non sono comprensibili dall'uomo per cui, a monte di esse, c'è un codice scritto in un preciso linguaggio di programmazione: è l'operazione di **compilazione** che "traduce" le istruzioni in linguaggio macchina.

Un software ha quindi una duplice natura: quella di **codice sorgente** e quella di **codice eseguibile**.

Software libero
Il **software libero** è solo quello che fornisce il permesso per chiunque di **utilizzarlo**, **copiarlo**, **modificarlo** e **ridistribuirlo**, in forma originale o dopo averlo modificato, sia gratuitamente che per trarne profitto.

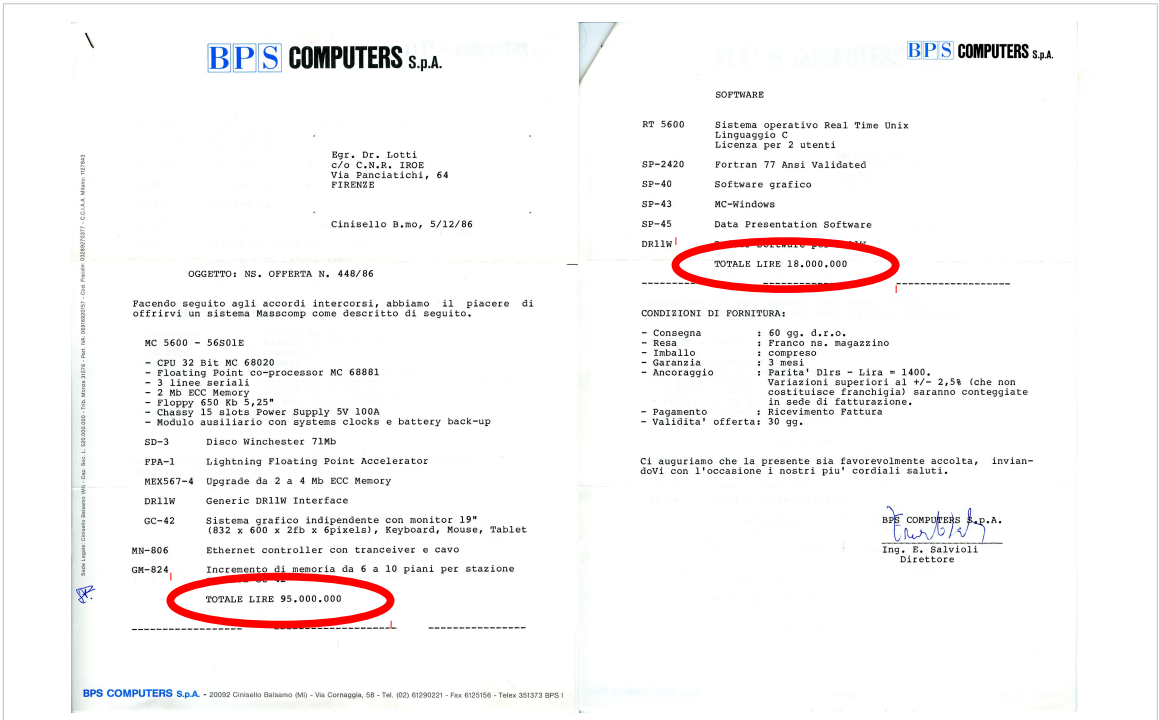
Comunemente ci si riferisce a queste condizioni necessarie a far sì che un software possa essere considerato libero come alle 4 libertà del software libero, che evidentemente può essere considerato tale solo se viene messo a disposizione assieme al codice sorgente ( “*if it’s not source, it’s not software”*).

Software semilibero e software proprietario
Esistono poi altri gradi di semi-libertà, ossia casi in cui il software viene distribuito con limitazioni riguardo ad uno o più dei punti suddetti.

Ci si riferisce a software **semi-libero** quando questo è distribuito con i sorgenti, permette uso, copia, modifica e ridistribuzione per qualsiasi scopo purché senza fine di lucro.

Il **software proprietario** è invece tutto quel software di cui è distribuito solo il codice che per essere utilizzato richiede il possesso di particolari privilegi a pagamento (licenze, chiavi hw ecc).

Esistono alcune varianti semi-libere del software proprietario alle quali si fa riferimento comunemente come **software shareware** quando è concesso un utilizzo gratuito temporaneo (periodo di prova) e/o limitato del software e **software freeware** quando ne è concesso un utilizzo gratuito illimitato.



BPS COMPUTERS S.p.A.

Egr. Dr. Lotti
c/o C.N.R. IROE
Via Panciatichi, 64
FIRENZE

Cinisello B.mo, 5/12/86

OGGETTO: NS. OFFERTA N. 448/86

Facendo seguito agli accordi intercorsi, abbiamo il piacere di offrirvi un sistema Masscomp come descritto di seguito.

MC 5600 - 56S01E

- CPU 32 Bit MC 68020
- Floating Point co-processor MC 68881
- 3 linee Seriali
- 2 Mb ECC Memory
- Floppy 650 Kb 5,25"
- Chassy 15 slots Power Supply 5V 100A
- Modulo ausiliario con systems clocks e battery back-up

| | |
|---|---|
| SD-3 | Disco Winchester 71Mb |
| FPA-1 | Lightning Floating Point Accelerator |
| MEX567-4 | Upgrade da 2 a 4 Mb ECC Memory |
| DR11W | Generic DR11W Interface |
| GC-42 | Sistema grafico indipendente con monitor 19" (832 x 600 x 2fb x 6pixels), Keyboard, Mouse, Tablet |
| MN-806 | Ethernet controller con tranceiver e cavo |
| GM-824 | Incremento di memoria da 6 a 10 piani per stazione |

TOTALE LIRE 95.000.000

BPS COMPUTERS S.p.A. - 20092 Cinisello Balsamo (MI) - Via Cornaggia, 58 - Tel. (02) 61290221 - Fax 6125156 - Telex 351373 BPS I

BPS COMPUTERS S.p.A.

SOFTWARE

| | |
|---|---|
| RT 5600 | Sistema operativo Real Time Unix<br>Linguaggio C<br>Licenza per 2 utenti |
| SP-2420 | Fortran 77 Ansi Validated |
| SP-40 | Software grafico |
| SP-43 | MC-Windows |
| SP-45 | Data Presentation Software |
| DR11W | |

TOTALE LIRE 18.000.000

CONDIZIONI DI FORNITURA:

- Consegna : 60 gg. d.r.o.
- Resa : Franco ns. magazzino
- Imballo : compreso
- Garanzia : 3 mesi
- Ancoraggio : Parita' Dlrs - Lira = 1400. Variazioni superiori al +/- 2,5% (che non costituisce franchigia) saranno conteggiate in sede di fatturazione.
- Pagamento : Ricevimento Fattura
- Validita' offerta: 30 gg.

Ci auguriamo che la presente sia favorevolmente accolta, inviandoVi con l'occasione i nostri piu' cordiali saluti.

BPS COMPUTERS S.p.A.
Ing. E. Salvioli
Direttore



Perché esiste il software proprietario?

Nonostante le nostre abitudini ci portino a pensare "ovvio, per trarne profitto" la risposta a questa domanda non è scontata.

Fino a tutti gli anni settanta, anche se in misura decrescente, la componente principale e costosa di un computer era l'hardware, il quale era comunque inutile in assenza di software. Da ciò la scelta dei produttori di hardware di vendere il loro prodotto accompagnato da più software possibile e di facilitarne la diffusione, fenomeno che rendeva più utili le loro macchine e dunque più concorrenziali. Il software, tra l'altro, non poteva avvantaggiare la concorrenza in quanto funzionava solo su un preciso tipo di computer e non su altri, neanche dello stesso produttore.

L'introduzione dei sistemi operativi rese i programmi sempre più portabili, in quanto lo stesso sistema operativo veniva offerto dal produttore di diversi modelli di hardware. La presenza di sistemi operativi funzionanti per macchine di differenti produttori hardware ampliava ulteriormente le possibilità di usare lo stesso codice in modo relativamente indipendente dall'hardware usato. Uno di questi sistemi operativi era Unix... (cit. Wikipedia)

**Il software proprietario è nato prinipalmente a causa dell'introduzione dei Sistemi Operativi.**

Infatti il sistema operativo permette un'astrazione dall'hardware della macchina offrendo un ambiente in cui è possibile realizzare il codice che poi viene compilato per quella determinata macchina: in questo modo viene meno la convenienza di chi vende la macchina a far si che il software si diffonda.

Inoltre l'avanzamento tecnologico ha permesso un abbattimento dei costi dell'hardware, quindi il costo di un computer è divenuto sempre più accessibile, di conseguenza sono esplose sempre più aziende di produzione di hardware il che ha contribuito all'abbattimento ulteriore del costo e quindi la diffusione del computer, in un circolo vizioso alimentato dal mercato.

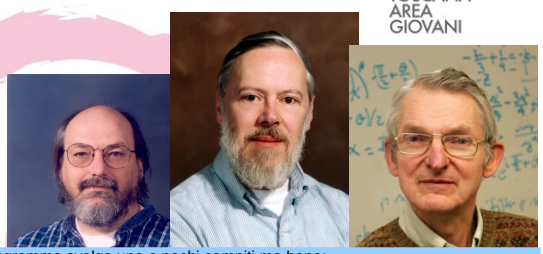
Di conseguenza c'è stata una forte differenziazione delle architetture (ciascuna delle quali col tempo ha trovato la propria strada, ad esempio nelle macchine consumer l'architettura i32 a più recentemente AMD64 hanno avuto la meglio, ARM nei dispositivi mobili e così via) questo fatto ha reso ancora più centrale il ruolo del SO spostando il gioco sul software.

Importanza=guadagno e così è nato il software proprietario, che ha fatto sbocciare migliaia e migliaia di software-house, alcune delle quali sono ad oggi fra le aziende col fatturato più alto del mondo.

## Introduzione: un po' di storia...

TOSCANA AREA GIOVANI

I primi sistemi operativi



UNIX

- si fa in modo che ogni programma svolga uno o pochi compiti ma bene;
- ci si aspetta che l'output di un programma diventi l'input di un altro;
- si sviluppa software con l'idea che esso verrà provato subito: non si esiti a condividere il programma;
- si usano degli strumenti appositi nella programmazione senza perdersi ogni volta nel "reinventare la ruota".

sistemi UNIX-like e UNIX standards

*The Open Software Foundation on Wikipedia (EN)*
*The Open Group on Wikipedia (EN)*
*The Open Group (EN)*
*UNIX (EN)*

Filippo Micheletti - Progetto TAG

UNIX®
Celebrating 40 years uptime

6

Se non si considerano i primi sistemi per il controllo di sistemi batch, come il Fortran Monitor System o l'IBSYS, il primo sistema operativo può considerarsi l' OS/360 della IBM, che introdusse l'implementazione dell'astrazione hardware, con un primo tentativo di multiprogrammazione (1964).
Più o meno contemporaneamente, in ambito accademico, nasceva CTSS del MIT, il primo sistema multiprogrammato, dal quale sarebbe partito il progetto Multics, in collaborazione con GE e Bell Laboratories.

In quegli anni uscivano i primi elaboratori "economici", come il PDP-1, per i quali furono sviluppati numerosi sistemi operativi.

Ma la vera rivoluzione fu **UNIX**. A partire dal 1969 un gruppo di ricercatori della **AT&T** presso i **Bell Laboratories**, tra cui erano presenti **Ken Thompson** (che lavorò anche al progetto Multics), **Dennis Ritchie** e **Douglas McIlroy**, diedero vita ad un sistema operativo con lo scopo di eseguire un programma chiamato "Space Travel", per la simulazione dei movimenti del sole e dei pianeti e la simulazione dell'atterraggio di una navicella spaziale: nasceva UNIX.

UNIX, scritto interamente in **C**, il linguaggio di programmazione sviluppato dallo stesso Dennis Ritchie, nasce come **sistema multiutente**, **multiprogrammato**, **portabile** ed **aperto**, dalle macerie del progetto Multics.

Le idee su cui è nato UNIX, che vedete citate nella slide, riassumono tutto ciò che sta alla base della buona programmazione e fanno capire meglio di ogni altra cosa quanto chiare fosser fin dall'inizio le idee di Tompson, Ritchie e McIlroy.

Con UNIX nasce infatti molto di più del primo vero sistema operativo...

AT&T non volle avere alcuna royalty sull'utilizzo e la modifica di UNIX, permettendo che il codice sorgente di UNIX venisse distribuito gratuitamente per fini di studio presso le università di tutto il mondo.

Per risolvere i problemi di condivisione del codice, si introdusse per la prima volta la copia di file tra sistemi dislocati in parti diverse del mondo attraverso la linea telefonica. Il sistema venne chiamato UUCP (Unix to Unix CoPy) e nonostante venne ben presto surclassato dalla nascente ARPANET (in seguito Internet) costituì di fatto uno degli esperimenti da cui è nata l'attuale Internet.

L'interesse commerciale da parte degli sviluppatori dei sistemi operativi derivati da UNIX (ricevuto GRAUITAMENTE!!!) ha portato a guerre di potere legate ai diritti di autore sul codice, battaglie fratricide spesso senza fine. Da UNIX nacquero così vari sistemi derivati, detti "UNIX-like", fra i quali **BSD** (Berkley Software Distribution) (1978), successivamente scissa per motivi legali in FreeBSD, NetBSD ed OpenBSD, e dalla quale hanno avuto origine SunOS e NextStep, "padre" di MacOS, **Minix**, **Xenix** (Microsoft), ed altri progetti di minor fama.

Attorno al modello UNIX si sono definiti due standard fondamentali: **il linguaggio C** (ISO/IEC 9899) e la definizione del sistema «UNIX» standard, noto come **POSIX** negli anni 1990 (Portable Operating System Interface uniX), ad opera della **OSF** (Open Software Foundation), poi divenuta **The Open Group**, che le ha evolute nelle **SUS** negli anni 2000 (Single Unix Specification).

## Introduzione: un po' di storia...

GNU & FSF

*GNU project*
*Free Software Foundation*
*Richard Stallman on Wikipedia (EN)*



Verso la metà degli anni 80 in un rinnovamento del parco stampanti del MIT venne adottato un nuovo driver proprietario per una stampante XEROX che non implementava più la funzionalità di segnalazione automatica della carta inceppata, molto utile quando una stampante è condivisa da molte persone.

Fra i programmatori del MIT vi era un certo **Richard Stallman**, al quale non andava giù la tendenza che si stava diffondendo da parte delle aziende di assumere i migliori programmatori impiegati dai centri di ricerca delle università per impiegarli nella scrittura di software proprietario.

Il driver della XEROX fu la gocciola che fece traboccare il vaso: Stallman si fece portavoce del problema e si attivò per combattere contro la privatizzazione del software, fondando nel 1985 la **FSF** (Free Software Foundetion), con il preciso scopo di promuovere e sostenere la creazione e diffusione di software libero.

L'idea di Stallman si concretizzò presto con la creazione della licenza **GPL** (General Public License), una licenza a protezione del cosiddetto **copyleft**, una sorta di duale del copyright che punta a riconoscere diritti di paternità più che d'autore, secondo I principi del software libero.

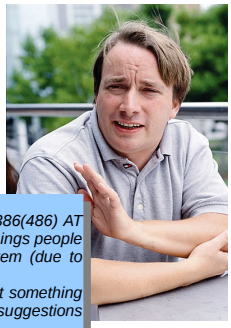
In realtà la FSF non è la prima organizzazione a nascere con questo scopo, un anno prima era nato infatti il progetto **GNU** (GNU is Not Unix), con lo scopo di creare un sistema operativo e un insieme di software che fosse basato su (e compatibile con) Unix, ma completamente indipendente e libero.

Non fatevi ingannare dall'aspetto di Stallman:negli ultimi 25 anni ha ricevuto centinaia di prestigiosissimi riconoscimenti internazionali, fra cui 8 tra lauree e PhD *honoris causa* (uno anche dall'università di Pavia)!

## Introduzione: un po' di storia...

TOSCANA AREA GIOVANI

### Linux



*Hello everybody out there using minix -*
*I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).*
*I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)*
*Linus (torvalds [at] kruuna.helsinki.fi)*
*PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.*

*—Linus Torvalds*

*Linux History (EN)*
*Linus Torwalds on Wikipedia (IT)*
*Tux on Wikipedia (IT)*
*The Linux Foundation (EN)*

THE LINUX FOUNDATION



**Linux** nasce all'inizio degli anni 90 come un progetto personale di studio delle funzionalità di multiprogrammazione dei microprocessori x86-32 da parte di **Linus Torvalds**, all'epoca uno studente all'università di Helsinki, in Finlandia.

Inizialmente il lavoro di Linus Torvalds si basava su un sistema Minix, riscrivendo il kernel e adattando solo successivamente il compilatore e i programmi sviluppati dal progetto GNU.

Il progetto di Torvalds concentrò ben presto una grossa comunità di sviluppatori in tutto il mondo.

Il testo il slide riporta il messaggio originale postato il 25 Agosto 1991 da Torwalds sul newsgroup "comp.os.minix" della Usenet dell'università di Helsinky: il resto è storia...

Il nome Linux, a dispetto dell'assonanza con il nome di Torwalds, è da attribuire a Ari Lemke, l'amministratore che rese per primo disponibile Linux su Internet via FTP. In particolare Linux era il nome della directory in cui risiedevano i file del nuovo sistema operativo. Il nome scelto da Torvalds era Freax, una combinazione tra "free", "freak" e "x", per indicare la caratteristica di un sistema Unix-like.

Nel 1996, ad opera di Larry Ewing, nasce **Tux** (Torwalds UniX), il pinguino mascotte ufficiale di Linux.

Nel 2007 viene findata la **The Linux Foundation**, un'organizzazione non-profit che sostiene lo sviluppo del kernel Linux

## Introduzione: un po' di storia...

Debian

DFSG

Open Source

*Debian.org*
*Debian project history (EN)*
*DFSG (EN)*
*Open source on Wikipedia (IT)*
*Open Source Initiative (EN)*



Nel 1993 **Ian Murdock** fonda il progetto Debian, con lo scopo di realizzare una distribuzione GNU/Linux libera e accurata.

Debian è la fusione delle parole **Deb** e **Ian**, con cui Murdock volle dedicare il progetto alla allora fidanzata Debra Lynn.

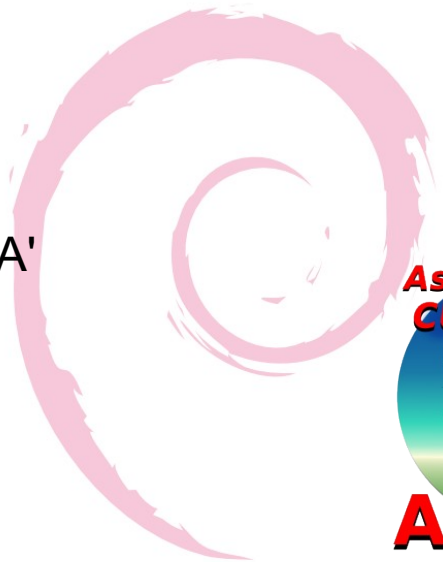
Il progetto Debian prende il via ufficialmente con la pubblicazione del “Manifesto Debian”, ocn il qualeMurdock pubblica le proprie intenzioni presentandoil progetto che ha in mente.

Successivamente, **Bruce Perens** stila la bozza di ciò che attualmente costituisce le «linee guida», per stabilire cosa sia da intendere «software libero» e cosa invece non rientri in questa categoria, ai fini della distribuzione stessa. Tali linee guida sono note con la sigla **DFSG** (Debian Free Software Guidelines).

**Open Source**

Sulla scia del progetto Debian, nel 1998 viene ufficializzata la definizione di Open Source, in seguito alla quale nasce anche un'organizzazione non-profit a cui aderiscono I principali progetti basati su software libero, con lo scopo di diffondere l'educazione all'open source, la **Open source Initiative**.

## Introduzione: pregi e difetti



COMUNITA'


*LugMap Italia (IT)*
*ACROS, LUG Versilia (IT)*
*Linux Day Italia (IT)*



I pregi del mondo Open-Source sono tutti riconducibili a ciò che sta dietro a questo mondo: una **comunità**.

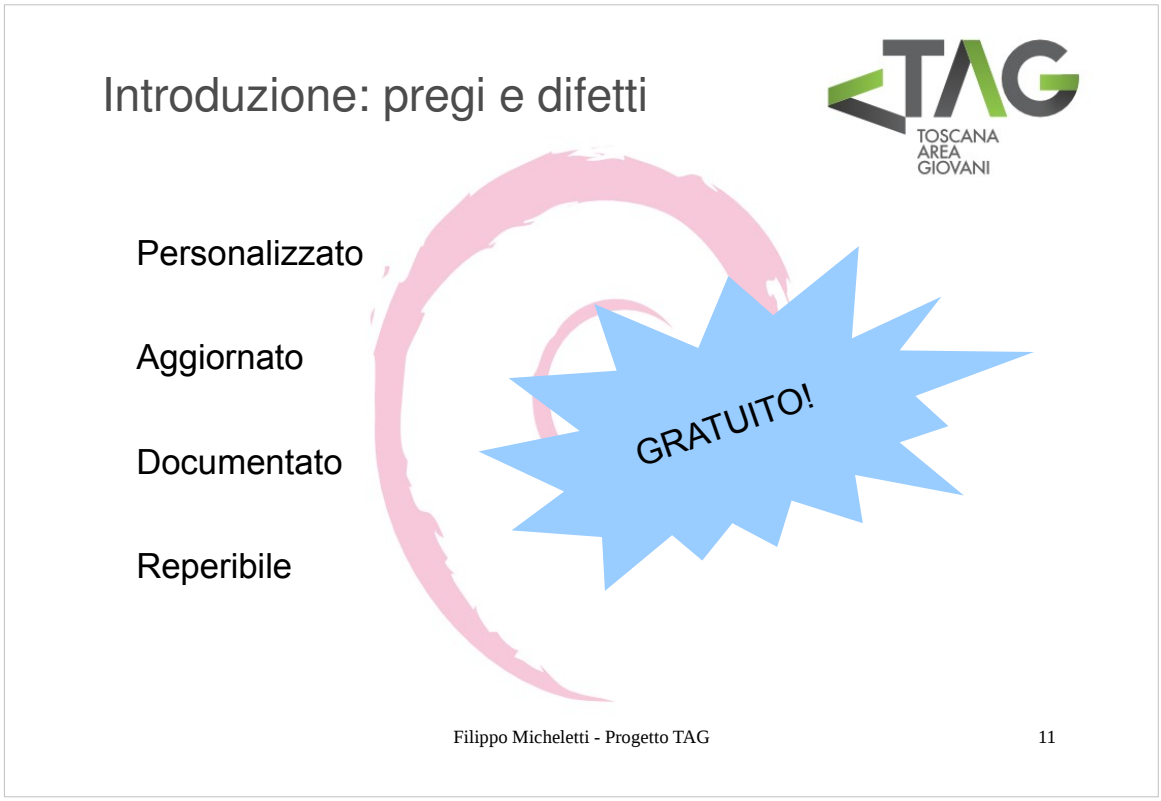
Il software libero nasce infatti dallo sviluppo di idee da parte di gruppi di persone che **collaborano** mettendo a disposizione le proprie idee e competenze in modo aperto ed accessibile.

Allo sviluppo non contribuisce però solo chi **scrive** il software, ma anche chi lo **usa**, che partecipa a comunità virtuali (forum, liste, ecc) dove si condividono esperienze, si propongono novità che rispecchino le necessità di chi usa quel software, si segnalano bug e deficit.

Le comunità non sono però solo virtuali, tutt'altro. Gli utenti di software libero sono coscenti di essere accomunati da qualcosa di speciale e spesso si organizzano nelle forme più varie, da associazioni come I **LUG** (Linux User Group) a vere e proprie società (avete mai sentito parlare della Canonical?).

In questo ambito esistono moltissime iniziative un po' in tutte le direzioni, che vanno dalla promozione del software libero (Linux Day) a eventi di tipo ludico fino a semplici ritrovi, un aspetto UNICO di questo mondo (avete mai sentito parlare di un Windows day?).

Insomma la comunità rappresenta il miglior "customer service" spontaneo del mondo, che non potrà mai essere superato da nessun servizio “costruito” ad hoc.




Le conseguenze di questa collaborazione è che il software libero è il prodotto della collaborazione di molte persone che si impegnano (principalmente) per **passione** per uno scopo in cui **credono**.

Inoltre l'apertura dei progetti fa sì che nelle comunità di sviluppo possano confluire personalità con competenze di tutti I tipi e a tutti I livelli, che trovano velocemente la propria posizione scegliendo di fare quello in cui si sentono più preparate e più appassionate.

Le comunità di sviluppo sono quindi gruppi in cui collaborano moltissime persone, in maniera continuativa e dinamica, ma soprattutto in continuo contatto con chi utilizza il software prodotto, primi fra tutti gli sviluppatori stessi.

Di conseguenza il software libero è generalmente molto più **aggiornato**, **sicuro** (libero da bug) e **performante** degli equivalenti proprietari, inoltre è generalmente possibile interagire direttamente con chi sviluppa il software per richiedere informazioni, segnalare problemi o mancanze del software.

Un altro aspetto che apprezzerete molto se deciderete di avventurarvi in questo mondo è quello della **documentazione**: il software è documentato a tutti I livelli, dalle procedure di sviluppo ai manuali di utilizzo, in modo semplice e chiaro, e la documentazione è spesso presente in molte lingue

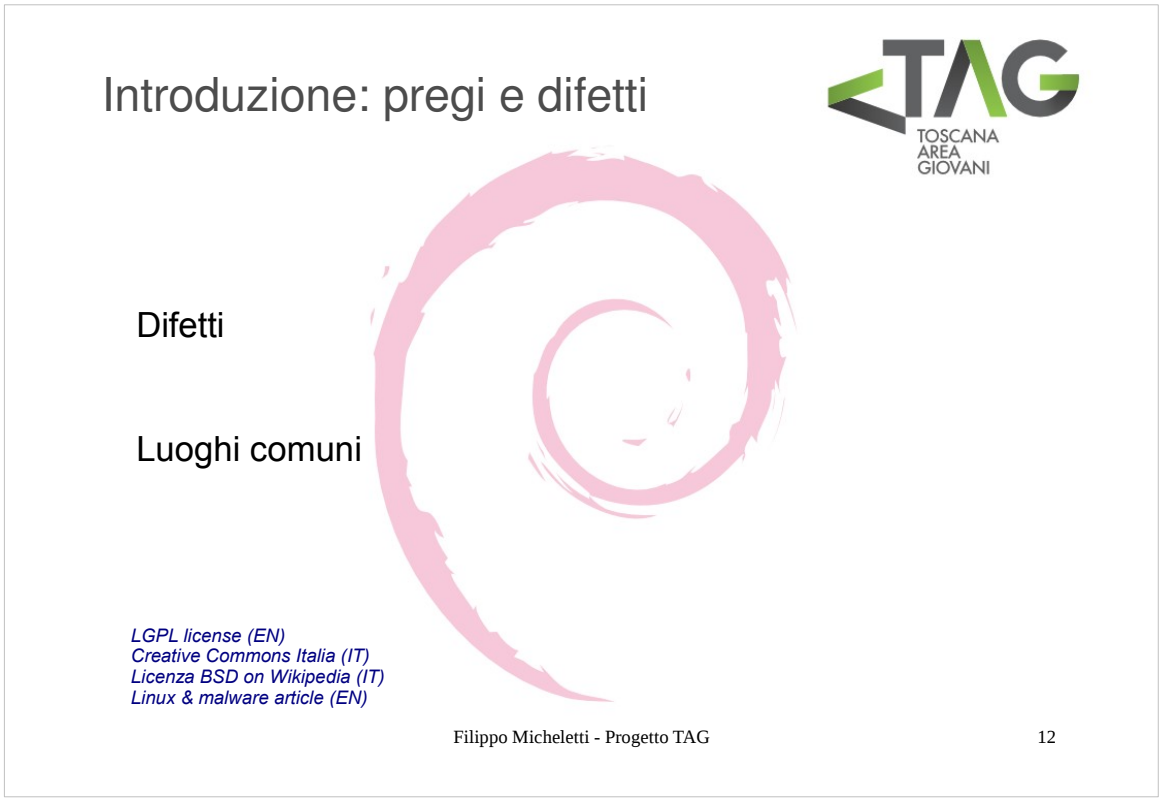
.

Come vedrete spessissimo si ricorre all'utilizzo delle documentazioni offline (man) per operazioni “rutinarie” come l'uso di un comando, o a quelle online (specialmente le guide ufficiali e gli how-to) per operazioni più complesse.

Un altro aspetto che distingue fortemente il software libero da quello proprietario è la **personalizzabilità**: tutto è concepito per essere customizzato in ogni suo aspetto, e anche quando qualche possibilità di customizzazione mancasse c'è sempre qualcuno che si cimenta per implementarla... perché non voi stessi per esempio? Ricordate che il codice è disponibile, basta rimboccarsi le maniche!

Infine il software libero è sempre **reperibile**, in tutte le versioni in cui è stato creato, insieme a tuttla la documentazione originale.
Potete comprarvi un computer del 91 e scaricare la prima distribuzione di Debian per divertirvi un po', come potete trovare il driver della vostra vecchia stampante e compilarlo per il vostro sistema a 64 bit.
Non sarete mai costretti a cambiare il computer perché l'ultima versione del software con cui lavorate non gira sul sistema che avete e il nuovo sistema richiede un hardware più potente del vostro, né dovrete mai cambiare la stampante perché I driver non sono più mantenuti per il sistema operativo che avete.

Ah, quasi dimenticavo... tutto questo E' **GRATUITO**!




E' chiaro che alcuni di questi aspetti possono comportare anche alcuni problemi.

Innanzi tutto la **gestione della comunità** a tutti I livelli non è semplice, e a volte vi entrano a far parte persone poco competenti o che agiscono addirittura contro il progetto per cui c'è sempre bisogno di meccanismi di autocontrollo e autodifesa.

Ovviamente poi c'è l'**aspetto economico**: salvo eccezioni I progetti sopravvivono sostanzialmente di donazioni, sia in termini finanziari che in termini di tempo di chi vi si dedica gratuitamente. Il segreto sta tutto nel minimizzare le spese ed annullare gli sprechi.

Infine credo che sia necessario sfatare alcuni **luoghi comuni**.

**Il software libero è più difficile da usare**: questa è la più grossa assurdità che si sente dire di solito a proposito del software libero. Spesso purtroppo il cambiamento spaventa e molte persone abituate ad usare certi software proprietari preferiscono rassegnarsi al pagamento di licenze o a compiere azioni illegali procurandosi software crackato piuttosto che provare qualcosa di semplicemente diverso. Nella maggior parte dei casi il software libero offre molte alternative che possono semplificare le cose o complicarle per offrire più servizi, in ogni caso troverete sempre qualcosa facile da usare almeno quanto l'equivalente proprietario a cui siete abituati.

**Il software libero non è regolamentato e protetto**: non è assolutamente vero. Il principio fondamentale su cui si basa il software libero è quello del copyleft (o pemrmesso d'autore): un modo che ci si è inventati per rappresentare (e proteggere) chi, mentre difende il proprio diritto di autore, vuole difendere la libertà della sua opera, imponendo che questa e le sue derivazioni restino libere. Sulla base del principio di copyleft sono nate alcune licenze a cui abbiamo già accennato, GPL in primis, estese poi a qualsiasi forma di opera creativa (CreativeCommons, Art Libre, ecc). Un'altra licenza spesso utilizzata per software libero è la BSD.

**Il software libero offre minori garanzie di quello proprietario**: sbagliato! In generale sul software proprietario non vi sono particolari garanzie se non quelle legate all'integraità dei dati sul supporto con cui il software viene venduto... il software libero è reperibile online in qualsiasi momento.

**Non esiste software libero professionale e I software proprietario di questo tipo sono incompatibili con I sistemi liberi**: ancora sbagliato. Esiste una grossa varietà di software libero professionale in grado di rimpiazzare egregiamente il sostituto proprietario, spesso in maniera anche migliore (a questo proposito vi segnalo il Linux Day 2012 il cui filone sarà “Software libero per le piccole e medie imprese”). Inoltre ormai praticamente qualsiasi software professionale viene distribuito compilato per Linux per cui anche volendo (o dovendo) utilizzare un particolare software proprietario è comunque possibile farlo in un ambiente libero. E in casi estremi c'è sempre la virtualizzazione... ma di questo parleremo più avanti!

**Linux è indipendente da malware**: (purtroppo) non è vero, anche se è sicuramente molto più robusto di qualsiasi SO commerciale ed esistono molti meno virus scritti per Linux.


Debian è il primo vero sistema operativo ad essere stato creato nel progetto GNU, nonostante questo non è l'unico.

Quasi contemporaneamente a Debian infatti sono nati altri 2 progetti basati sul kernel Linux: **Slackware** e **Red Hat**.

Da Debian, Slackware e Red Hat si sono poi sviluppati molti progetti che hanno dato vita a distribuzioni basate su una di queste tre ma con variazioni orientate sia ad ambienti e caratteristiche particolari (ad es. Per macchine server o per sistemi embedded), sia più semplicemente a creare qualcosa di diverso, magari perché non si condividevano alcuni punti del progetto base.

A tutte le distribuzioni derivate da Debian, alcune delle quali sono divenute decisamente più popolari di Debian stesso, come Ubuntu, sono dette "**Debian child**" in quanto, per un verso o per un altro, sono figlie, nipoti, bisnipoti di Debian.

Ma allora perché scegliere di divenire utenti Debian?

Debian è l'unica distribuzione che ha mantenuto fissi all'interno del proprio progetto tutti I punti originali del progetto GNU, e che rispetta a pieno tutti I punti di **libertà** del software facente parte del progetto.

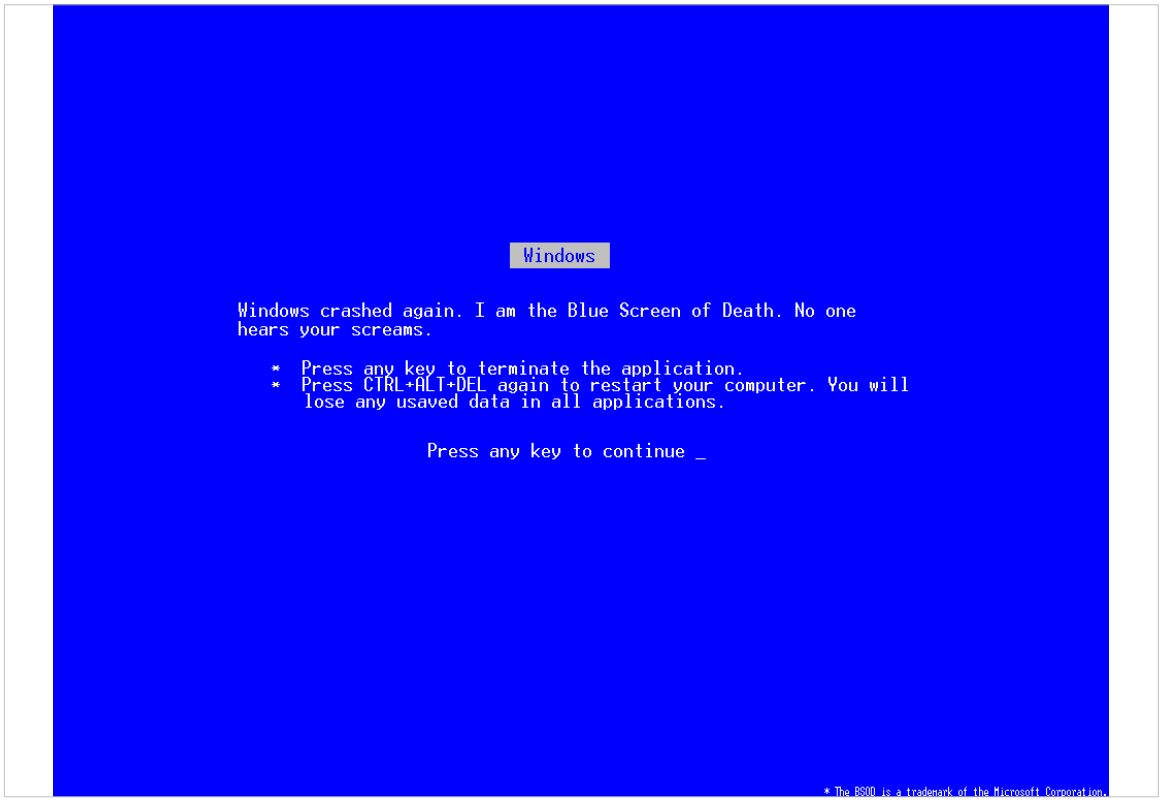
Debian è inoltre una distribuzione **bilanciata** che offre un'**architettura essenziale** ma **non scarna**, dove insomma c'è (solo) tutto quel che serve.

Di conseguenza Debian è una delle distribuzioni più **customizzabili**.

Inoltre Debian è uno dei sistemi operativi disponibile per più architetture in assoluto: la distribuzione stabile attuale è rilasciata per 24 varianti di 10 architetture diverse.

Dulcis in fundo Debian è senza ombra di dubbio il sistema più orientato a **stabilità** e **sicurezza**, cosa che vi farà dimenticare per sempre perdite di dati e crash del sistema e schermate blu.

Una curiosità: ad oggi si stima che le distribuzioni basate su kernel Linux siano oltre 600 (alcuni progetti non sono più attivi)... siete ancora convinti che esistano solo Windows e MacOS?


Windows
Windows crashed again. I am the Blue Screen of Death. No one hears your screams.
* Press any key to terminate the application.
* Press CTRL+ALT+DEL again to restart your computer. You will lose any usaved data in all applications.
Press any key to continue _
* The BSOD is a trademark of the Microsoft Corporation.

Scherzetto :)

## Concetti base: sistema operativo

Sistema operativo

kernel

shell(s)



Una macchina NON ha bisogno di un sistema operativo per poter svolgere un compito: il codice può essere fornito direttamente alla macchina che si occupa dell'elaborazione e della restituzione dei risultati.

Questo era quello che accadeva quando i computer erano macchine grandi quanto un automobile e costose come un aereo: il tempo macchina veniva costosamente affittato a chi doveva svolgere dei calcoli, che doveva aspettare il suo turno.

In questo modo il codice si impossessa letteralmente dell'hardware della macchina, disponendone a piacimento: il processore si occupa soltanto delle operazioni di elaborazione ed indirizzamento, senza alcun controllo sul corretto utilizzo della macchina da parte del codice.

La diffusione di macchine più economiche, e successivamente del personal computer, ha portato alla necessità di condividere le risorse della macchina tra diversi utenti, intesi sia come utenti umani che come processi in esecuzione: è nato così il sistema operativo.

Un **sistema operativo** è formalmente un software che offre un astrazione dell'hardware della macchina su cui è in esecuzione per fornire un ambiente più o meno standardizzato all'interno del quale possano operare i programmi.

Il SO **gestisce le risorse** della macchina assegnandole ai processi che le richiedono secondo determinate politiche (scheduling).

I programmi chiedono di poter accedere alle risorse tramite delle chiamate al SO (system calls).

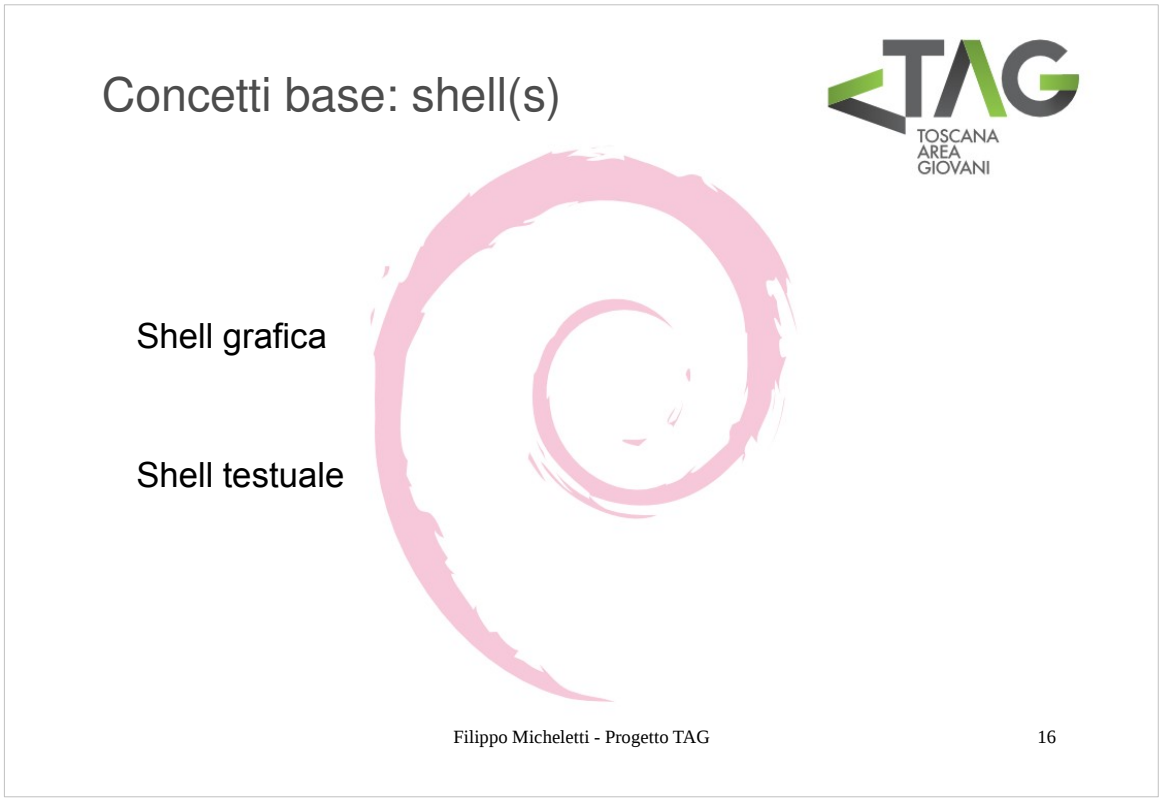
Il SO è composto fondamentalmente di due parti:

- il kernel
- la shell

Il **kernel** (nocciolo) è il cuore del SO, ossia la parte che si occupa di astrarre la struttura hardware per proteggere le risorse del sistema e gestirle distribuendole ai processi che le richiedono.

La **shell** (guscio, conchiglia) è la parte concettualmente esterna del SO, che mette a disposizione un'interfaccia per poter interagire con il kernel attraverso dei comandi.

Kernel e shell sono due parti complementari per un sistema che nn possono prescindere l'una dall'altra: possiamo vedere infatti il kernel come il motore di un'automobile e la shell come I comandi per guidarla, nonostante sia spontaneo pensare che il motore è, tutto sommato, più importante dello sterzo o della leva del cambio, se ci pensiamo un attimo ci rendiamo subito conto che un'auto senza sterzo può avere il più bel motore del mondo ma non serve a nulla...!




Esistono due tipi di shell:

- testuale
- grafica

La shell **testuale** è stata la prima ad essere implementata e per questo motivo spesso con il termine "shell" si fa riferimento a questo tipo di interfaccia.
Il funzionamento di una shell testuale è semplice e diretto: l'utente comunica con il SO tramite dei **comandi** forniti da tastiera, ricevendo i risultati sotto forma di testo stampato a video.

Esistono diverse shell testuali, ognuna con le proprie caratteristiche (sh, csh, bash, ksh, tcsh, rsh, jsh, dtksh, rksh, ...)
Anche in questo caso non esiste una shell migliore di un'altra: l'ambiente Linux contiene di default sh e bash, altri SO Unix-like adottano di default shell diverse (per esempio ksh in Solaris).

Conviene sceglierne una ed imparare ad usarla. Per il nostro corso faremo riferimento a **Bash**.

Le shell **grafiche** sono nate successivamente a quelle testuali, in concomitanza all'introduzione del mouse, con l'idea principale di rendere più intuitivo l'utilizzo di un sistema anche da parte di chi ne ha poca conoscenza.

In questo caso l'utente comunica con il SO tramite un ambiente grafico tipicamente composto da icone e finestre tra le quali è possibile spostarsi tramite un sistema di puntamento (mouse). L'interazione vera e propria con il SO in questo caso avviene tramite i menu contestuali delle finestre e la selezione di icone per compiere operazioni su file o avviare programmi.

La shell grafica di Debian (così come quella di molti altri sistemi Linux-based) è composta da due parti: un **server grafico** (**X**) ed un **DE** (Desktop Enviroment, **GNOME** in Debian).

Spesso chi non conosce I sistemi Linux-based è convinto che l'utilizzo di Linux passi attraverso l'utilizzo della shell testuale, magari perché ha visto qualche amico utilizzare un terminale, e che sia necessario ricordarsi centinaia di comandi da impartire al sistema anche per compiere le operazioni più semplici. Questa convinzione è dovuta al fatto che nei sistemi commerciali che la maggior parte delle persone è abituato ad usare, la shell testauale è praticamente inesistente o così nascosta che I più non sanno nemmeno di averla, per cui pochissimi ne conoscono le potenzialità e anzi ne temono gli aspetti “mnemonici”.

In realtà, ome avremo modo di vedere, la shell testuale permette di svolgere molte operazioni in modo estremamente chiaro e veloce, eliminando quasi completamente l'utilizzo del mouse e l'apparire di finestre su finestre. Ma perché allora la shell testuale è stata messa da parte nei SO più diffusi?

I sistemi commerciali hanno in effetti puntato molto sullo sviluppo delle interfacce grafiche promuovendone una (spesso dubbia) maggior intuitività e semplicità di utilizzo. In un mondo sempre più basato sull'immagine l'aspetto grafico di un sistema può essere determinante per il suo successo e in effetti una grafica (ritenuta) accattivante è ed è stata il cavallo di battaglia di certe case produttrici che hanno dominato per anni il mercato del software.

## Concetti base: shell(s)

Grafico vs testuale

*I pirati della Silicon Valley on Wikipedia (IT)*



Sebbene possa sembrare un aspetto secondario questo è stato uno dei motivi principali che ha limitato fino ad oggi la diffusione di sistemi Linux-based, ritenuti erroneamente più ostici rispetto ad altri SO dall'aspetto invitante e amichevole. In effetti questo effetto è l'emblema della differenza di approccio alla base delle strategie adottate dagli sviluppatori di sistemi Linux rispetto a quelli di software commerciali: nel primo caso, laffove non c'era da conquistare un mercato, si è sempre puntato alla funzionalità del sistema ponendo come priorità sicurezza, stabilità ed affidabilità, nel secondo invece si sono spesso tralasciati questi aspetti per concentrarsi su grafiche accattivanti che invogliassero il consumatore a preferire il prodotto.

Per questi motivi in passato le varie interfacce grafiche su Linux hanno avuto una grafica abbastanza scarna ed essenziale. Oggi la situazione si è decisamente evoluta: l'offerta è fortemente differenziata e si può scegliere fra interfacce ricche fino ad ambienti minimali e performanti. Generlamente le distribuzioni stesse vengono offerte in più soluzioni "preimpacchettate".

I **vantaggi** di una shell **grafica** sono sicuramente
- **intuitività**
- **semplicità** (certe volte solo apparente)
- **aspetto gradevole** (opinabile)

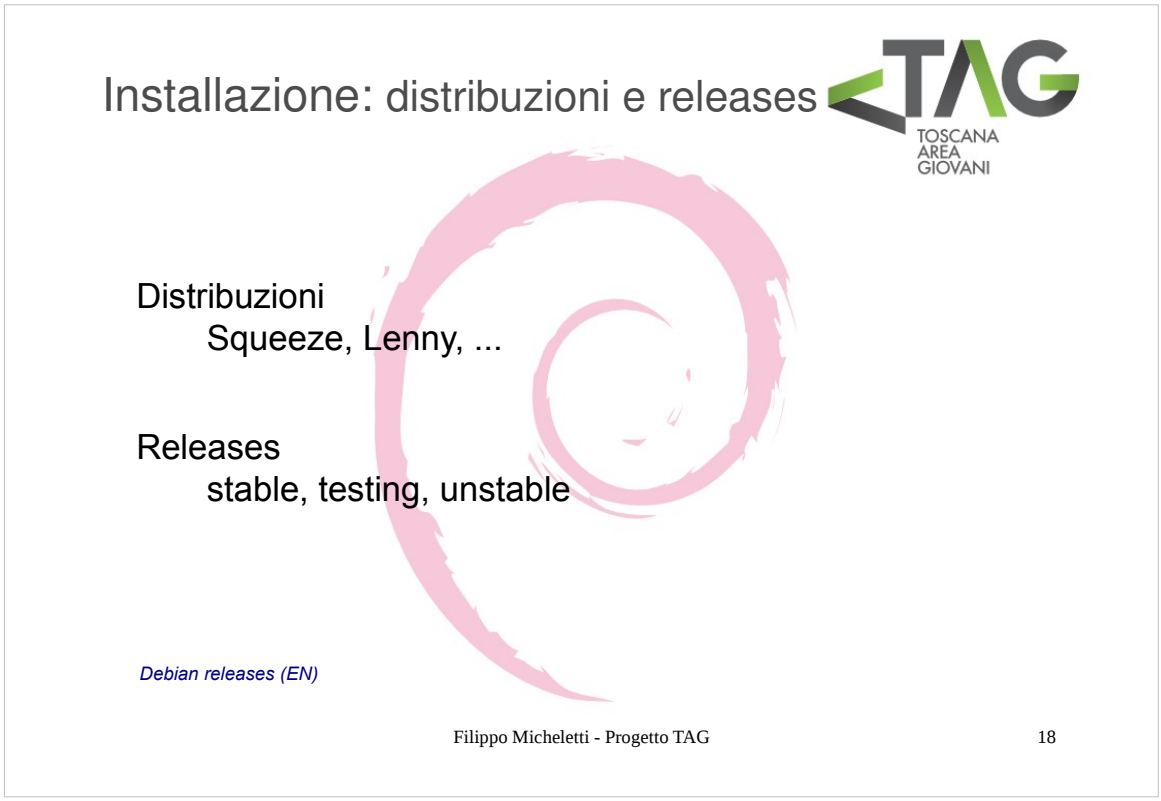
Gli **svantaggi** sono sicuramente
- **pesantezza** (in termini di spazio e di risorse per la gestione)
- **lentezza** per lo svolgimento di operazioni (uso del mouse, navigazione grafica)
- presenza di **oggetti inutili e distrattivi**
- **limitatezza** delle operazioni eseguibili (che comporta) difficoltà nello svolgere operazioni personalizzate o particolari

In sostanza la shell grafica è piacevole ma impone di accettare le limitazioni di chi l'ha realizzata.

**Vantaggi** e svantaggi di una shell **testuale** possono essere ricavati per dualità da quelli della shell grafica.
I principali vantaggi sono
- **leggerezza**
- **velocità**
- **semplicità** grafica (massima chiarezza)
- **potenza** espressiva (possibilità di sfruttare TUTTE le potenzialità offerte dal SO e dagli applicativi)
- **controllo** totale sulle proprie azioni
- **hystory** dei comandi

Fra gli **svantaggi** si hanno sicuramente
- **minor intuitività** (è necessario conoscere l'ambiente in cui ci si muove... ma questo è realmente uno svantaggio?)
- **aspetto** grafico **minimale**

In ogni caso non c'è un'interFaccia migliore per utilizzare il SO: l'interfaccia migliore è quella che meglio si addatta alle proprie esigenze ed è importante conoscere entrambe le interfacce e saper passare da una all'altra “spontaneamente” in base a ciò che si vuol fare.




Debian viene rilasciato sotto forma di **distribuzioni**, versioni del SO rilasciate in successione.

Tra le distribuzioni, rilasciate in successione, si distinguono 3 **releases**:

- stable
- testing
- unstable

La release **stable** è la distribuzione rilasciata attualmente come stabile.
Il concetto di stabilità non è legato tanto all'affidabilità del software quanto al fatto che il software contenuto in questa distribuzione non subirà modifiche se non quelle per l'eliminazione di eventali bug.
vantaggi: distribuzione stabile e sicura, aggiornamenti sporadici, di piccola entità
svantaggi: software non recentissimo

La release **testing** è la distribuzione contenente i pacchetti destinati a far parte della prossima stable.
Il software in quest distribuzione è generalmente poco più aggiornato di quello della stable ma non ne è garantita la stabilità: per questo motivo l'installazione della distribuzione testing non è di solito particolarmente vantaggiosa.

La release **unstable** contiene i pacchetti più recenti.
vantaggi: software recentissimo
svantaggi: possibile presenza di bug (ancora) irrisolti, aggiornamenti consistenti pressoché quotidiani

Con il termine **oldstable** ci si riferisce inoltre alla distriduzione stable precedente alla attuale.

Esiste infine una pseudo-distribuzione detta **experimental**, che in realtà è più un parcheggio per il nuovo software in fase di sviluppo.

L'utente Debian ha due possibilità: scegliere una distribuzione e mantenerla con i relativi aggiornamenti, seguendo la filosofia "squadra che vince non si cambia" oppure scegliere una release e mantenere il sistema aggiornato ad essa.

Vedremo più avanti come sia possibile (e semplice!) seguire una di queste due strade.

## Installazione: distribuzioni & releases

*Disney Toy Story (EN)*
*Personaggi di Toy Story on wikipedia (IT)*



I nomi delle distribuzioni Debian sono mutuati dal nome dei personaggi del film di animazione **Toy Story**.

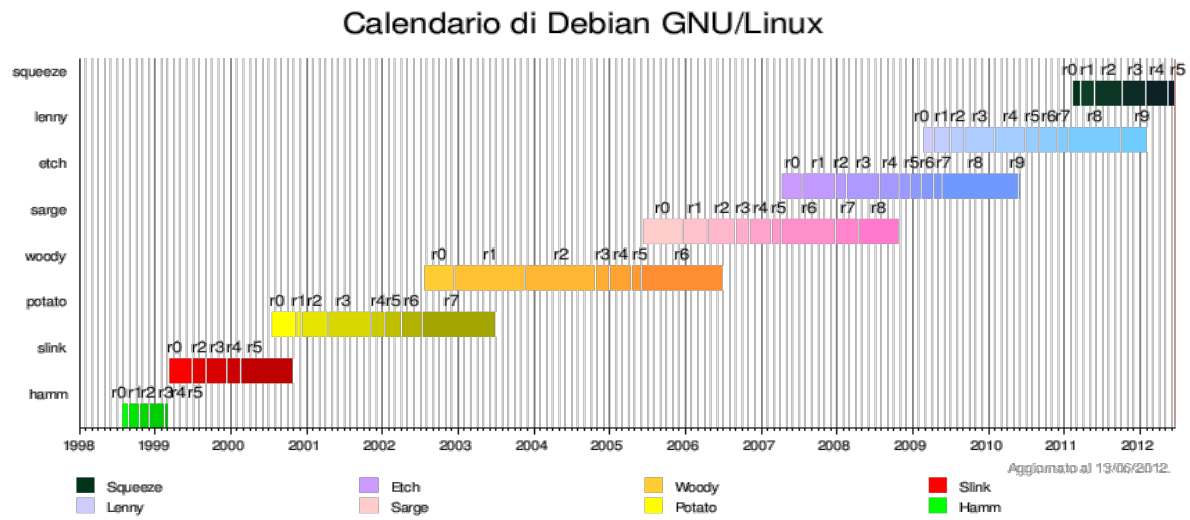
L'elenco delle distribuzioni con le relative date di rilascio a stable è il seguente:

1.1 - **Buzz**, rilasciata il 17 giugno 1996
1.2 - **Rex**, rilasciata il 12 dicembre 1996
1.3 - **Bo**, rilasciata il 2 giugno 1997
2.0 - **Hamm**, rilasciata il 24 luglio 1998
2.1 - **Slink**, rilasciata il 9 marzo 1999
2.2 - **Potato**, rilasciata il 15 agosto 2000
3.0 - **Woody**, rilasciata il 19 luglio 2002
3.1 - **Sarge**, rilasciata il 6 giugno 2005
4.0 - **Etch**, rilasciata l'8 aprile 2007
5.0 - **Lenny**, rilasciata il 14 febbraio 2009
6.0 - **Squeeze**, rilasciata il 6 febbraio 2011

L'attuale distribuzione testing è denominata **Wheezy**.

Mentre tutte le distribuzioni rilasciate portano il nome di un giocattolo, la release unstable ha quello del bambino irrequieto e imprevedibile: **Sid** (che fra l'altro risponde anche l'acronimo di *Still In Developement*).

Infine, periodicamente vengono rilasciati aggiornamenti della distribuzione in corso.



## Installazione: pacchetti e dipendenze

pacchetto

dipendenza




Il pacchetto è l'unità di distribuzione del software in Debian. Con il termine **pacchetto** si indica un insieme di file che vengono distribuiti congiuntamente per ragioni di ordine prevalentemente tecnico.

I pacchetti non sono quasi mai isolati ma dipendono gli uni dagli altri (ad esempio un software che richiede l'uso di una determinata periferica si appoggerà ad altri software che la gestiscono, che saranno condivisi condivisi con altri applicativi quindi contenuti in pacchetti separati): questo approccio permette una modularità del sistema che ne permette il rapido aggiornamento, la reparabilità, ed un forte snellimento.
Tutti I pacchetti da cui dipende un determinato pacchetto si dicono sue **dipendenze**.

**La strada dei nuovi pacchetti**
I nuovi pacchetti vengono inseriti in experimental se hanno bisogno di test approfonditi o non sono considerati stabili dall'autore, altrimenti vengono inseriti direttamente in unstable.

Gli sviluppatori (e gli utenti di unstable) testano i pacchetti presenti in unstable, segnalando bug e modifiche desiderabili.

Il pacchetto viene "promosso" alla testing quando supera alcuni criteri di stabilità (tempo minimo di permanenza in unstable, compilazione riuscita su tute le architetture supportate, dipendenze soddisfatte da pacchetti già presenti in testing, nessun danneggiamento di pacchetti in testing, minor numero di bug release-critical della precedente versione dello stesso pacchetto già presente in testing).

Infine, quando la distribuzione testing ha raggiunto un buon grado di stabilità, viene dichiarata una situazione di freeze durante la quale non si possono apportare modifiche ai pacchetti (tranne per correggere bug critici) e successivamente la testing è promossa a stable, la unstable a testing (con l'assegnazione di un nuovo nome) e viene aperto un nuovo progetto unstable + experimental.

## Installazione: getting Debian




Ottenere Debian è semplice e gratuito: è sufficiente scaricarlo dal sito www.debian.org .

E' possibile scaricare diverse versioni dell'installazione, di tipo e dimensione diversa.

Innanzi tutto è necessario scegliere la **tipologia** d'installazione:

- Scaricando un'immagine delle dimensioni di un cd per un'installazione offline di base
- Scaricando un'immagine delle dimensioni di un dvd per un'installazione offline più completa
- Scaricando un'immagine di tipo netinst (anch eper chiavette usb) per un'installazione online

E' poi necessario scaricare l'immagine per la propria **architettura** (ad es. i386 o AMD64 sono le più diffuse per I laptop).

## Installazione: installer


*Guida all'installazione di Debian on Debianizzati (IT)*

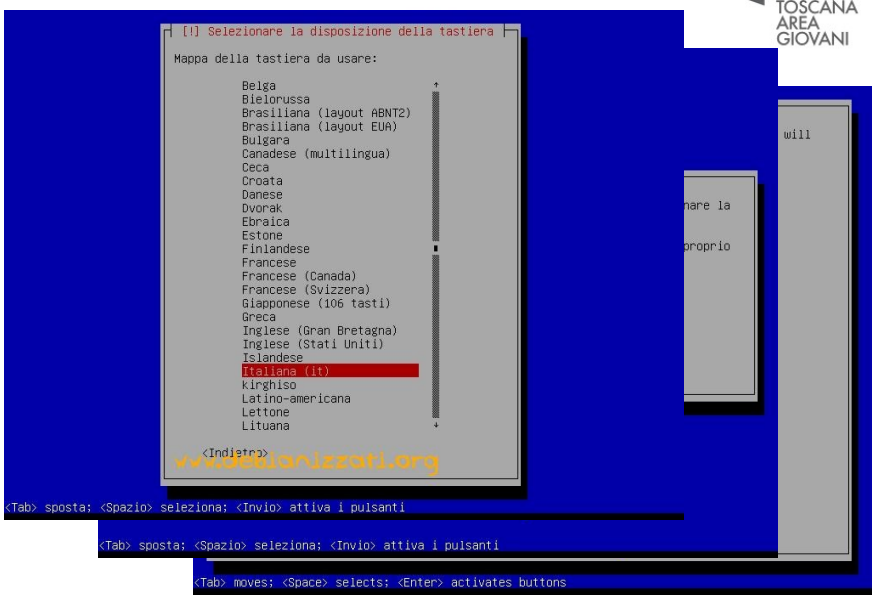


Vediamo I punti principali della procedura d'installazione.

Una volta avviato il computer (con boot da disco o usb a seconda del supporto che avete scelto per l'installazione del sistema) si avvierà l'installer di Debian. Per installare il SO selezionate la voce “Install”.

NB: la procedura d'installazione tradizionale nn prevede l'utilizzo del mouse. Per **scorrere** tra le voci dei menù utilizzate le **frecce**, per **selezionare** una voce utiizzate **invio**, per spuntare una **check-box** utilizzate la **barra spaziatrice**, per **cambiare** menù o passare ai pulsanti di navigazione (avanti, indietro, ok, ecc), utilizzate il tasto **tab**.

# Installazione: lingua

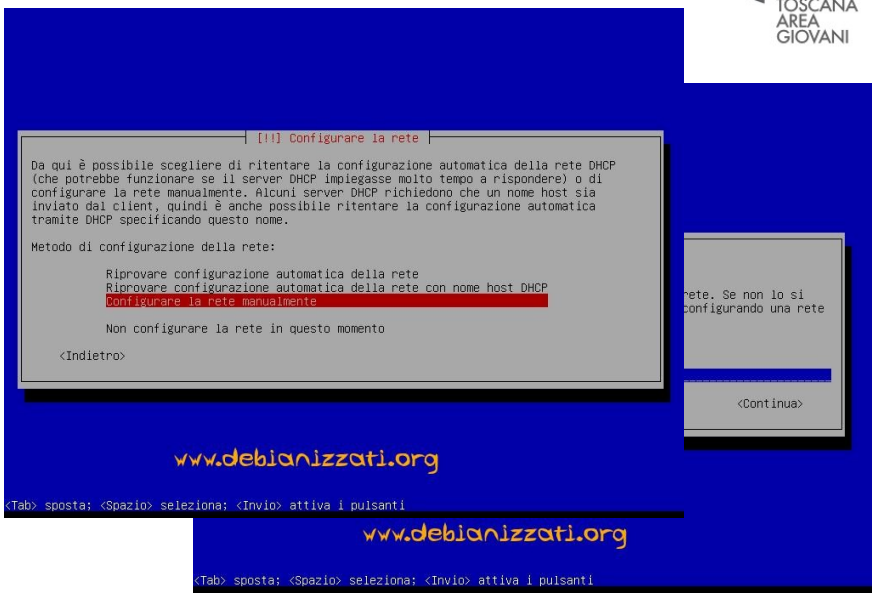





La prima fase della procedura d'installazione consiste nelle impostazioni della **lingua**.

Vi verrà chiesto di impostare la lingua del sistema, la lingua della tastiera e la regione geografica in cui vi trovate per l'impostazione del fuso orario.

# Installazione: rete

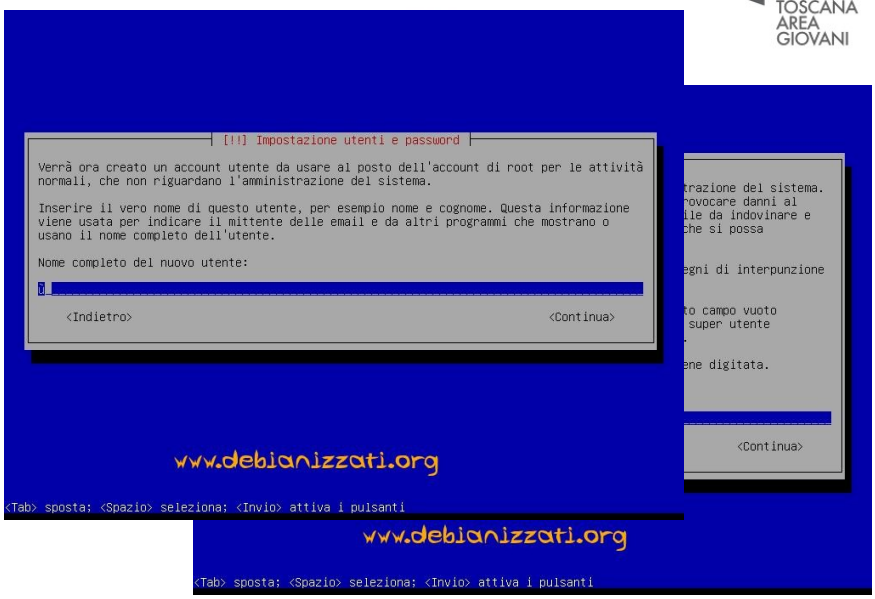





Segue quindi l'impostazione della **rete**.

A meno che non abbiate deciso di eseguire un'installazione online questo passo può essere fatto sucessivamente, per cui scegliete il **nome host** (il nome con cui verrà identificato il vostro computer sulla rete locale a cui vi collegherete) e selezionate la voce “Non configurare la rete in questo momento”.

## Installazione: utenti

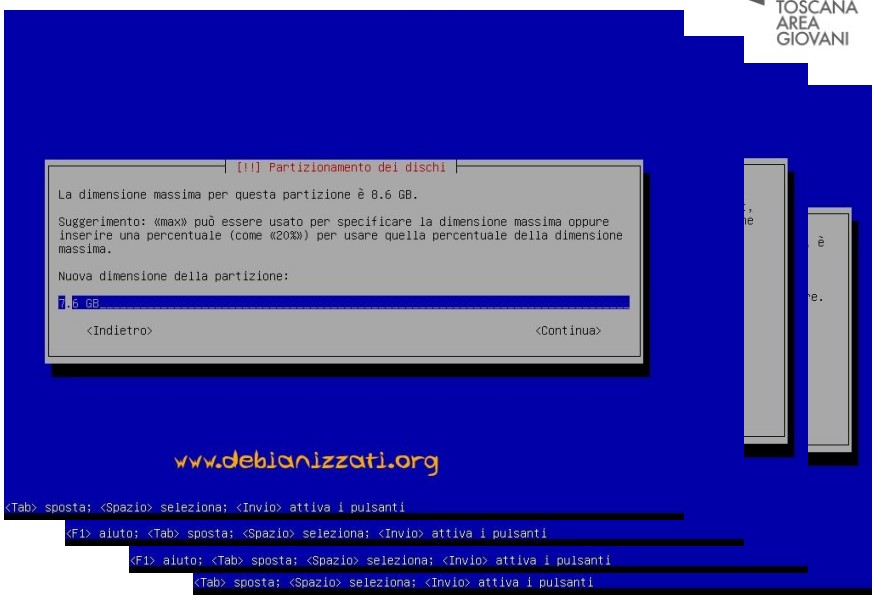





Vi verrà quindi chiesto di creare almeno un **utente**.

Scegliete nome utente e password per il vostro account personale (potrete aggiungerne altri successivamente se prevedete di condividere la macchina con altre persone o avete comunque la necessità di mantenere delle utenze separate) e impostate quindi la password per l'**utente di root**: parleremopiù in dettaglio più avanti di questo aspetto, per ora è sufficiente saper eche l'utente di root è colui che svolge I compiti di amministrazione del sistema e possiede quindi I privilegi più ampi su di esso.

## Installazione: partizionamento






La fase successiva è la procedura di **partizionamento** del disco.
Scegliete l'opzione “manuale” e procedete.

Il partizionamento base richiesto per l'installazione di un SO Linux-based è composto da due partizioni:

| | |
|---|---|
| / | partizione di **root** |
| /swap | partizione di **swap** |

NB: in realtà questa osservazione non è del tutto corretta: è infatti possibile forzare il kernel Linux a realizzare la memoria virtuale su un normale file della partizione di root; in questo modo si potrebbe installare l'intero sistema su una sola partizione ma la gestione della memoria virtuale è notevolmente più efficiente se questa è realizzata su di una partizione separata per cui la partizione di swap è fortemente raccomandata.

Tuttavia un partizionamento un po' più articolato permette di sfruttare alcuni dei più grossi vantaggi di Linux, riducendo al minimo le operazioni di manutenzione e la possibilità di perdita dei dati.

Ci sono infatti diversi buoni motivi per separare fisicamente certe sezioni logiche del sistema.

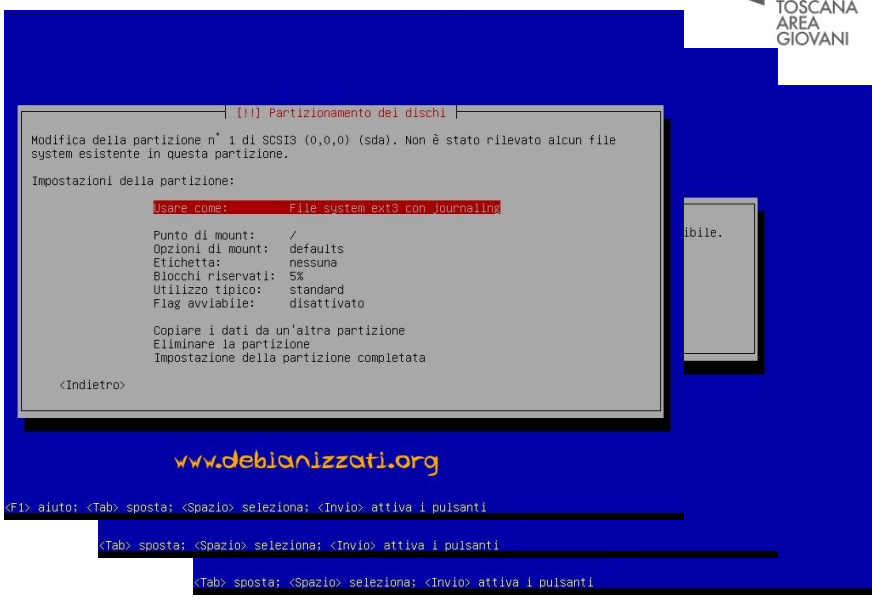
Prima di tutto la **sicurezza**: se si incontrassero problemi su un filesystem questi coinvolgerebbero soltanto una parte del SO per cui ripristinando quella parte (tramite un'opportuna copia di backup) si potrebbe ripristinare (rapidamente) tutto il sistema.
Un altro buon motivo è la **gestione**: la separazione di certe sezioni del sistema permette un maggior controllo su di esse e una gestione più chiara del sistema stesso in tutti i suoi aspetti.
Infine, soprattutto per quanto riguarda la home directory, la sicurezza nei confronti della **perdita dati**: tenere separati i propri dati (compese le impostazioni di tutti i programmi in uso!) dal resto del sistema permette di avere una certa tranquillità che anche se un giorno ci fossero problemi che compromettessero il sistema in modo grave da richiedere una nuova installazione tutto l'aspetto "personale" non andrebbe perduto.

Il disco può essere partizionato in fase d'installazione oppure è possibile utilizzare un qualsiasi tool di partizionamento con Il quale è possibile anche ridistribuire lo spazio successivamente nel caso in cui ce ne fosse la necessità; uno di questi, open-source ovviamente, è **GParted** (Gnome PARTtition EDitor), di cui esiste anche una versione live.

Ad ogni modo, per partizionare in fase d'installazione, scegliete lo spazio libero su disco (se intendete eliminare/modificare partizioni precedenti potete farlo sempre da questo menù, selezionando la partizione in questione e scegliendo quindi l'operazione voluta), quindi “crea una nuova partizione” e impostate la dimensione voluta.
Fate attenzione a non eccedere il massimo (4) di partizioni primarie, quindi create la partizione a partire dall'inizio dello spazio libero su disco, selezionate il filesystem con cui formattare la partizione e selezionate quindi “Impostazione della partizione completata”. Ripetete la procedura per ciascuna partizione da creare.

Una nota sul filesystem: I più “adatti” a Linux sono gli ext, attualmente **ext3** è di sicuro il migliore in quanto a stabilità ma potete scegliere anche ext4. Ricordate però che questi filesystem non sono supportati nativamente dalla maggior parte dei SO commerciali, per cui se prevedete di condividere dati con un altro SO di questo tipo ripiegate su FAT.

## Installazione: partizionamento






La partizione di **swap** non dovrebbe avere dimensioni inferiori a quelle della ram installata sulla macchina. Visto che viene utilizzata come memoria virtuale è di solito anche inutile che sia di dimensioni maggiorni per cui la regola del pollice è farla esattamente delle stesse dimensioni della RAM.

La partizione di **root (/)** deve sempre contenere fisicamente le directory /etc, /bin, /sbin, /lib e /dev.
Sono sufficienti circa 150–250 MB per questa partizione.

**/usr** contiene tutti i programmi utente (/usr/bin), le librerie (/usr/lib), la documentazione (/usr/share/doc), ecc. Questa è la parte del file system che di solito occupa più spazio su disco per cui tutto dipende dall'utilizzo che si farà del sistema. Se decidete di non separarla (lasciandola quindi inclusa nella partizione di root) fate attenzione ad aumentare opportunamente la dimensione di quest'ultima.

**/var** contiene i dati variabili, sostanzialmente tutte le cache (siti web, cache del sistema di gestione dei pacchetti, ecc.), database e log. La dimensione da scegliere dipende fortemente dal tipo di uso che si farà del sistema, ma per la maggior parte degli utenti il fattore principale di cui tenere conto è il funzionamento del sistema di gestione dei pacchetti: se si intende installare in una sola sessione tutto il software fornito da Debian, dovrebbero bastare 2 o 3 GB di spazio per /var. Se invece si intende installare il sistema a più riprese (ad esempio, installare le utilità di sistema, poi quelle per la gestione dei documenti, poi il sistema X, ecc), è sufficiente riservare da 300 a 500 MB. Se si intende risparmiare al massimo lo spazio su disco e non si hanno in programma massicci aggiornamenti del sistema, è possibile riservare anche solo 30 o 40 MB.

**/tmp** contiene i dati temporanei creati dai programmi. La maggior parte degli applicativi non genera grosse moli di file emporanei (al massimo dell'ordine di grandezza delle centinaia di MB), tuttavia ad esempio i software per la masterizzazione sfruttano questa directory per memorizzare l'immagine di un disco da copiare per cui si è sempre al sicuro con 6 GB.

Infine la **/home**: ogni utente conserverà i propri dati personali in una sybdirectory di questa directory (il rispettivo Desktop, la propria cartella di Downloads e una serie di cartelle nascoste contenenti i file di impostazioni personali di tutti i programmi in uso). La sua dimensione dipende dall'utilizzo della macchina, dal numero di utenti che utilizzeranno il sistema e dal tipo di file che saranno conservati nelle loro directory.

In base a queste considerazioni, per un sistema personale tipo workstation con un HD da almeno 60 GB dove si prevede l'installazione di molti applicativi un buono schema di partizionamento potrebbe essere questo:

| | |
|---|---|
| / | 25 GB |
| /swap | dimensione pari a quella della RAM installata |
| /boot | 1 GB |
| /var | 2 GB |
| /tmp | 6 GB |
| /home | tutto lo spazio restante (digitare *max* nel campo di selezione della dimensione) |

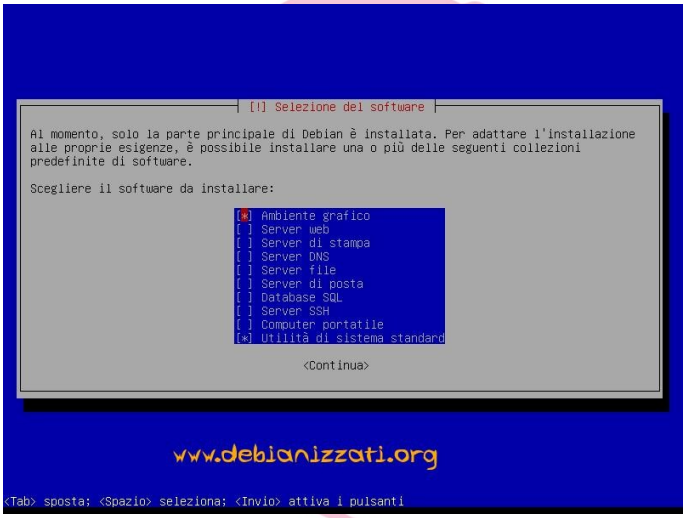
Per esigenze e scopi diversi è comunque abbastanza facile ridimensionare questa valutazione ripercorrendo i ragionamenti visti.

# Installazione: popularity-contest




Terminato il partizionamento vi verrà chiesto di partecipare o meno al **popularity-contest**, per supportare lo sviluppo del progetto Debian tramite l'invio di statistiche anonime, bug-report e quant'altro. Se avete a disposizione una (qualsiasi) connessione vi sonsiglio di partecipare... ne va anche del vostro bene!

# Installazione: selezione software






Fra gli ultimi passaggi è necessario scegliere il **software** da installare: qui dipende dall'utilizzo che verrà fatto della macchina, l'installazione prevede diverse configurazioni preimpostate con I pacchetti più comunemente utilizzati in questi casi, fra cui diversi tipi di server. Per le nostre esigenze è sufficiente selezionare l'*ambiente grafico*, le *utilità di sistema standard* e se necessario l'opzione *computer portatile*.

# Installazione: grub

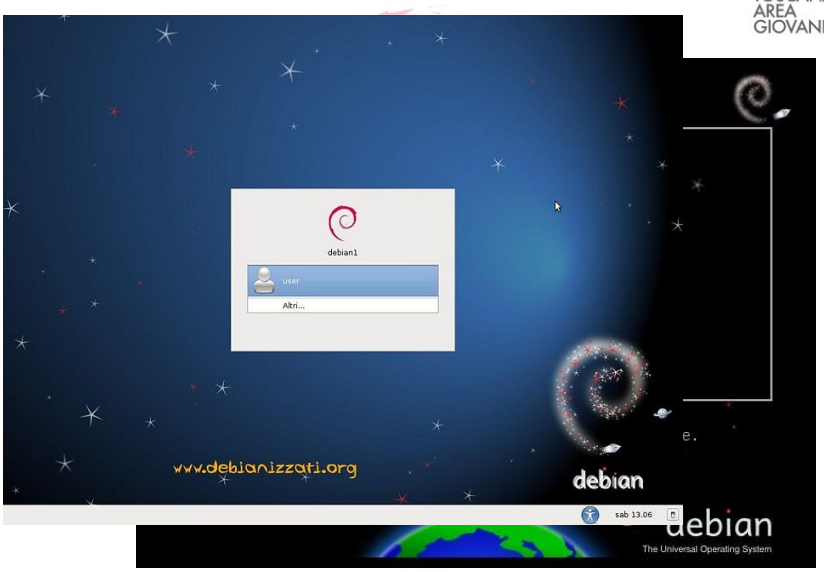



Infine vi verrà chiesto di installare GRUB, un software boot loader necessario per gestire la presenza contemporanea di più sistemi operativi ma anche per poter scegliere alcune opzioni di avvio (ad esempio l'avvio in recovery-mode).

L'installazione di GRUB è consigliata anche nel caso in cui non si intenda installare altri SO sul computer, e non comporta sprechi di spazio (GRUB si installa sul MBR (Master Boot Record) del disco, una porzione di spazio che rimarrebbe comunque riservata e quindi inaccessibile).

Vedremo fra poco come comortarsi nel caso in cui si voglia far convivere Debian con uno o più altri SO.

# Installazione: avvio






Terminata l'installazione all'avvio del comuter si passerà prima dalla schermata di GRUB, quindi a quella di login di Debian: il sistema è pronte per l'uso!

## Installazione: distribuzioni live

Debian live

provare, recuperare, riparare...


*Debian Live Project (EN)*



**Debian live**

Quando si parla di sistemi live ci si riferisce ad un SO che può essere avviato da un supporto removibile o da rete senza alcuna installazione sul disco fisso della macchina e quindi senza alcuna modifica del computer.

In pratica una live è un sistema che si installa sulla memoria volatile (RAM) del computer.

Questa risulterà una grande novità per chi mette piede per la prima volta nel mondo Linux in quanto i SO più diffusi non offrono alcun supporto live, che invece risulta utilissimo, specialmente in caso di problemi con il SO installato sul computer.

A cosa serve una distribuzione live?
A moltissimo. L'uso più banale è quello di provare un sistema operativo senza modificare il proprio computer. Molto più interessante è invece l'utilizzo per cui sono state pensate le distribuzioni live: **recupero** dati e **ripristino** di sistemi danneggiati.
Quandi si ha un problema che impedisce l'avvio del SO infatti è possibile utilizzare una live per cercare di risolverlo agendo sui file che compongono il SO oppure, se non c'è proprio speranza, recuperare I dati dalle partizioni interessate per potervi reinstallare il sistema.

In Debian il progetto live è relativamente recente ed offre un supporto limitato alle 2 architetture più popolari i386 (32 bit) e amd64 (64 bit), ma con una disponibilità ampia di personalizzazioni a seconda dell'uso che si intende fare della live: sono disponibili infatti versioni complete con l'ambiente grafico standard di Debian (X+GNOME) adatte a chi vuol provare per la prima volta Debian o ad utenti meno esperti, così come versioni con DE più leggeri, quali LXDE, o addirittura senza X Windows System e DE (versioni rescue e standard), adatte alla riparazione di sistemi danneggiati.

## Installazione: multi-boot

multi-boot

bootloaders

GRUB

*GRUB Project (EN)*

```
default=0          imposta come default la prima "label" sotto indicata
timeout=10         imposta a 10 secondi il tempo di attesa prima di caricare
                   automaticamente l'entry di default.
splashimage=(hd0,2)/boot/grub/splash.xpm.gz      path della schermata di boot
password --md5 $1$6ðòüZßXÈ$bXTLL8IbDhnwmjyaNNcPG.
                   password criptata da fornire per poter accedere
                   al menu o alla command-line
title Debian GNU/Linux, with Linux 2.6-32-5-686
                   titolo della prima scelta del menu ("label")
root (hd0,2)       HD (primary master) e partizione (terza) del device di root.
kernel /boot/vmlinuz-2.4.7-10 ro root=/dev/hda3      path del kernel
initrd /boot/initrd-2.4.7-10.img
                   path del file system da montare su Ram Disk al boot.
title DOS          titolo della seconda scelta del menu
rootnoverify (hd0,0)     HD (primary master) e partizione (prima) del device di
root.
...
```



Si parla di **multi-boot** quando su una macchina convivono più SO (non necessariamente diversi!)

La situazione più diffusa è (purtroppo) quella del multi-boot fra un sistema Linux-based ed un sistema commerciale, dovuta specialmente fra gli utenti alle prime armi alla paura di compiere qualche errore che faccia perdere il controllo del proprio computer (…). Senza giudicare le scelte altrui è comunque sconsigliato scegliere questa opzione, specialmente quando si è alle prime armi, perché spesso porta a evitare di risolvere le difficoltà che si possono incontrare quando si è alle prime armi per tornare ad usare il proprio vecchio sistema proprietario.

Altre volte il multi-boot è considerato necessario da alcuni per la necessità di utilizzare software propretari che non vengono istribuiti per Linux. Anche in queto caso il multi-boot non è assolutamente neessario, e come vedremo è possibile (e avolte anche vantaggioso) utilizzare su Linux software compilati per altri SO.

Molto meno comune è invece la situazione in cui si abbiano installate più versioni di una stessa distribuzione, o più distribuzioni diverse di sistemi Linux.

In ogni caso è ovvio che su una macchina potrà essere in esecuzione un solo SO operativo alla volta (questo non è del tutto vero, più avanti diremo qualcosa di più...) e la “convivenza” di più SO installati è gestita da un software detto **Boot Loader** che si occupa di avviare il SO (**bootstrap**).

Il boot loader si trova nella settore di boot del dispositivo. In un hard disk ciascuna partizione primaria contiene un settore di boot, il primo della partizione, inoltre l'intero disco contiene un **MBR** (Master Boot Rrecord): è il boot loader presente in questo settore che viene avviato dal bios.

Esistono diversi loader che eseguono il bootstrab del sistema operativo per Linux:

- su sistemi Intel Based: LILO, GRUB, LOADLIN, SYSLINUX, BOOTLIN;
- su sistemi Alpha: MILO;
- su sistemi Sparc: SILO.

Il boot loader attualmente distribuito di default con Debian è **GRUB**, che ha sostituito lo storico LILO, superandolo per semplicità e flessibilità.

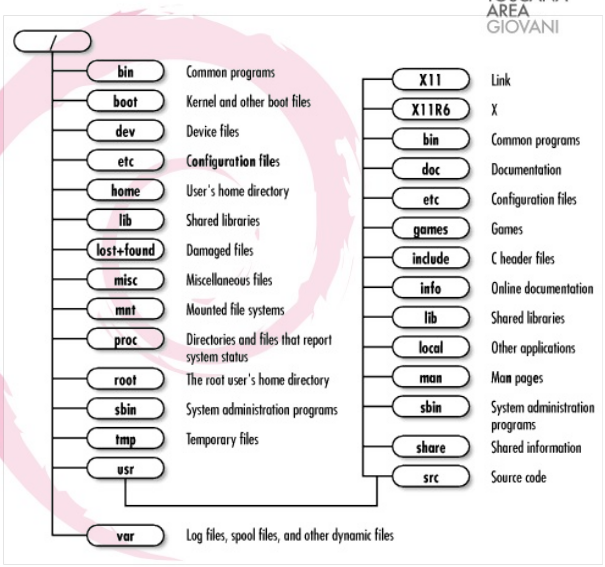
Tutte le impostazioni di GRUB possono essere configurate tramite il file /etc/grub.conf

In slide è riportato un esempio commentato (in corsivo) di questo file, non entriamo qui nel dettaglio del funzionamento e della gestione di GRUB, ma limitiamoci ad un consiglio: se volete installare Debian in multiboot con un SO commerciale installate PRIMA il sistema commerciale in modo che GRUB si sovrascriva al Boot Loader proprietario del SO commerciale che altrimenti ne impedirebbe l'avvio rendendo necessarioreinstallare e configurare manualmente il Boot Loader.

## Primi passi: struttura logica

Albero directory

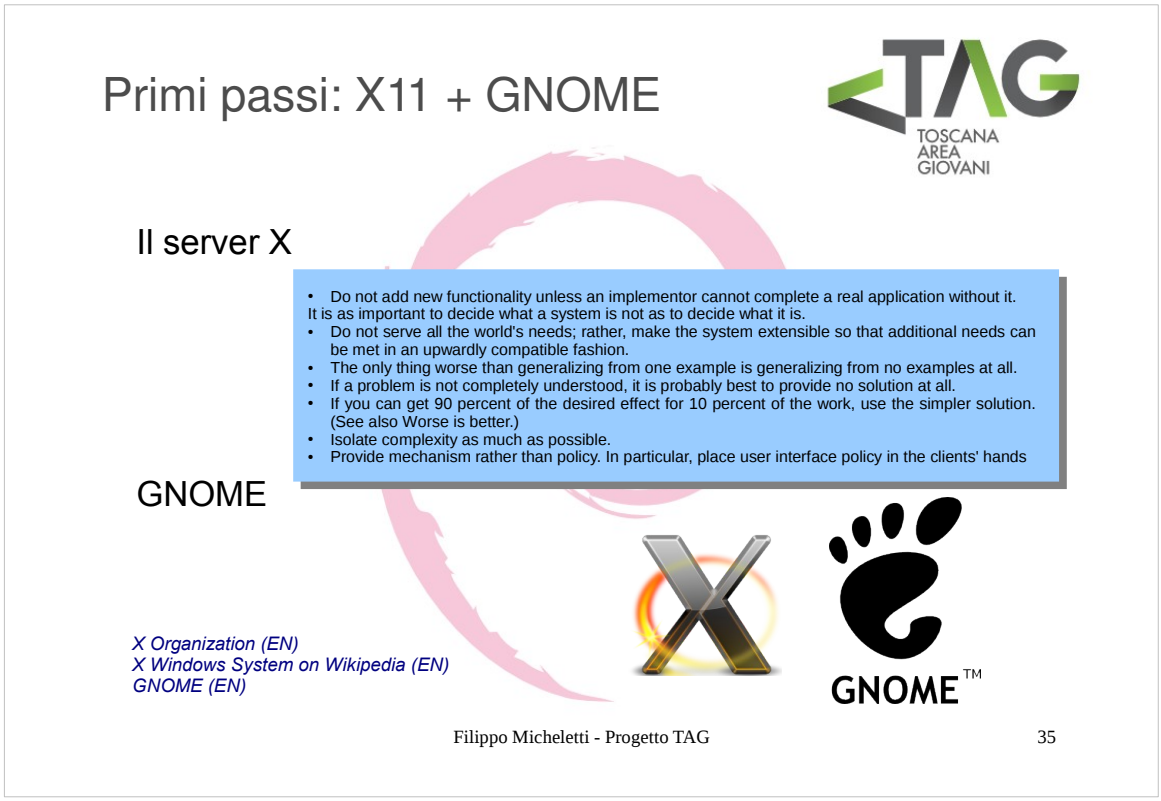
Dove e come






L'albero delle directory:

| | |
|---|---|
| **/bin** | file binari dei comandi essenziali |
| **/boot** | file statici del bootloader |
| **/dev** | devices |
| **/etc** | file di configurazione del sistema (host) |
| **/home** | home directory degli utenti |
| **/lib** | librerie condivise e moduli del kernel |
| **/lost+found** | files danneggiati |
| **/media** | punti di mount di dispositivi removibili |
| **/mnt** | punti di mount di dispositivi fissi e filesystem temporanei |
| **/opt** | pacchetti applicativi opzionali |
| **/proc** | filesystem virtuale per le informazioni di sistema (aree di memoria processi, immagine della ram,ecc) |
| **/root** | home directory dell'utente di root |
| **/sbin** | file binari essenziali per il sistema |
| **/srv** | file di servizio |
| **/sys** | directory virtuale per le informazioni di sistema |
| **/tmp** | file temporanei (di sistema) |
| **/usr** | librerie, file binari e quant'altro installato dagli utenti |
| **/var** | dati variabili |




In Debian (e in molti altri SO Linux-based) la shell grafica è fatta di due componenti: un **server grafico** e un **DE** (Desktop Enviroment)

Il server grafico si occupa della gestione dei dispositivi di IO (sostanzialmente mouse, tastiera e schermo) e ne fornisce le primitive di utilizzo, mentre I DE si occupa di creare e gestire gli oggetti grafici sullo schermo.

**Il server X**

X è nato nei laboratori del MIT nel 1984 come evoluzione di W. Dopo varie vicissitudini legate all'interesse industriale per X si è arrivati all'attuale Xorg Foundation, fondazione nata nel 2004 per lo sviluppo dell'X Windows System come software open source.

La versione attuale del protocollo implementata su diversi SO tra cui Debian è una release della versione 11 del protocollo, X11, risalente al 1987.

Il funzionamento di X è basato sul paradigma server/client: il server X, in esecuzione sulla macchina, gestisce le periferiche di input (tastiera, mouse, touchscreen) e output (schermo), definendo le primitive grafiche per l'interazione tra i dispositivi di input e gli oggetti grafici sullo schermo, ma non definisce delle specifiche per l'aspetto dei componenti dell'interfaccia grafica, che sono invece demandati al DE.

Il server X serve i client che richiedono l'utilizzo dell'interfaccia grafica, i quali possono trovarsi in esecuzione sulla stessa macchina in cui si trova il server (per esempio un'applicazione come un browser) o in remoto.

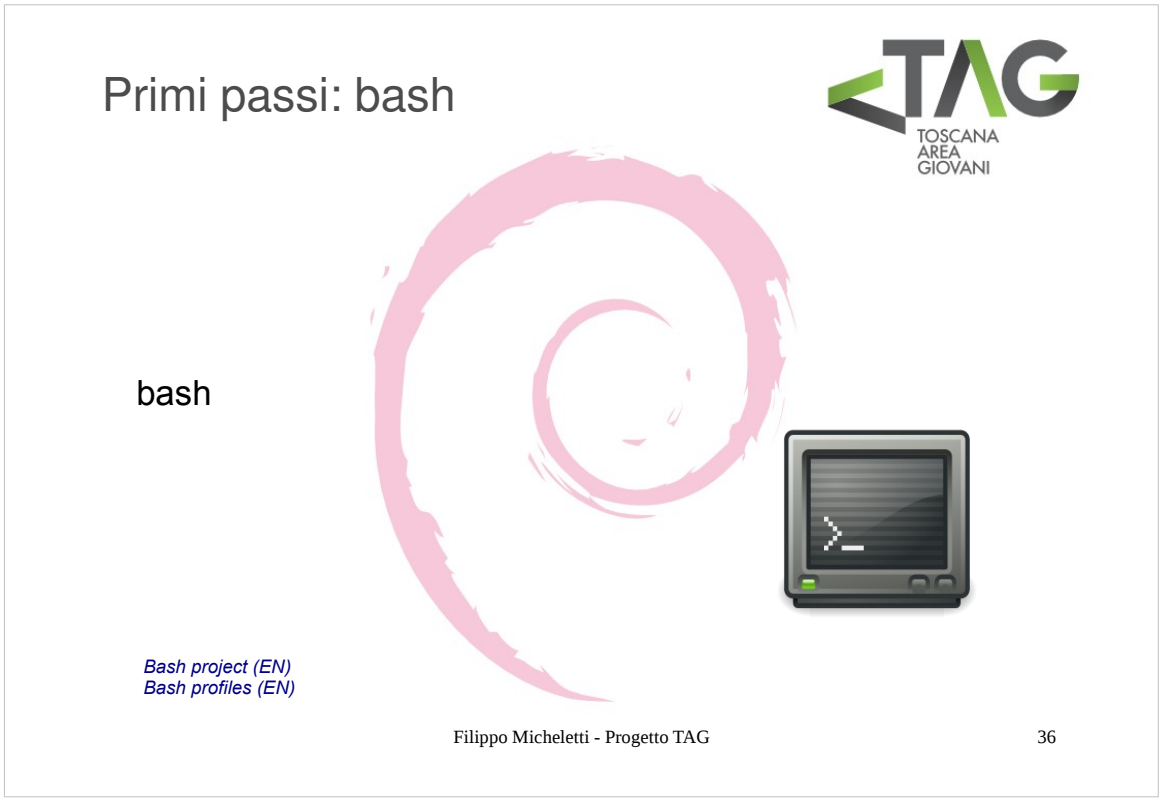
L'idea sulla quale è basato il funzionamento del sistema X (client/server) risale alle necessità di un epoca in cui il costo dei computer ne limitava il proliferare: succedeva allora (ad esempio nei laboratori di un'università) che molte persone dovevano condividere la stessa macchina. Con la nascita dei SO multiprogrammati l'utilizzo da parte di più utenti sarebbe potuto avvenire in parallelo ma l'utilizzo fisico di uno stesso computer da parte di più persone risultava difficile per ovvi motivi. Nacque allora l'idea del server X e dell'X-term: più utenti condividevano una (potente) macchina centrale attraverso dei semplici terminali collegati ad essa tramite una rete, detti X-term, e costituiti da uno schermo, una tastiera, eventualmente un mouse, ed un'interfaccia dedicata alla gestione del solo X server: in questo modo ciascun utente disponeva della propria (economica) interfaccia personale con la workstation centrale che eseguiva i calcoli...

**GNOME**

GNOME è il DE di default del progetto Debian. Si tratta di un DE basato sulle librerie GTK+ creato nel 1997 da Miguel de Icaza e Federico Mena come alternativa a KDE.

Il progetto GNOME fornisce: l'ambiente grafico GNOME, un desktop intuitivo ed invitante per gli utenti e la piattaforma di sviluppo GNOME, un framework per creare applicazioni che si integrano all'interno del desktop

GNOME risulta un ottimo compromesso fra performance e completezza.




**Bash** (Bourne Again SHell) è la shell testuale più diffusa in ambiente Linux.

Bash è un'evoluzione di sh (bourne SHell, la shell più semplice) che integra le funzionalità di csh (C SHell, una shell evoluta con funzionalità di programmazione derivate dal linguaggio C).

La scelta di bash come shell testuale di default per Linux è motivata dal fatto che bash è una shell completa e ricca ma al tempo stesso di semplice utilizzo.

Fra le potenzialità della bash possiamo citare:

- **autocompletion** dei comandi
- **history** infinita dei comandi inseriti
- **programmabilità**

In Debian sono disponibili di default 6 shell testuali indipendenti, fra le quali è possibile switchare tramite la combinazione di tasti **crtl+alt+fx** (con x da 1 a 6), mentre **ctrl+alt+f7** comporta il passaggio all'ambiente grafico

La shell testuale è disponibile anche in versione emulata dalla shell grafica, tramite l'aplicazione **terminale**, che di fatto nell'utilizzo quotidiano del sistema è il modo più diffuso di utilizzare bash.

## Primi passi: utente di root

su





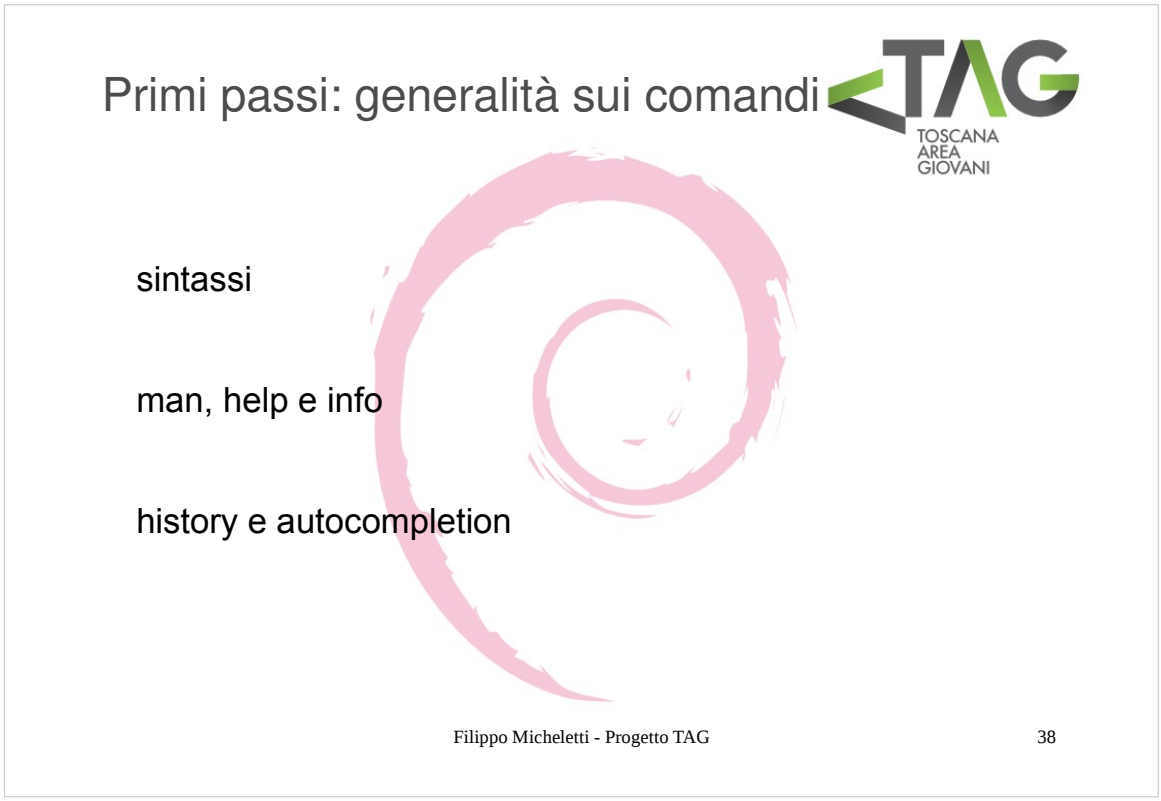
L'accesso al sistema avviene, di norma, tramite un utente “tradizionale”.

Tuttavia durante l'utilizzo del sistema spesso si devono svolgere mansioni di amministrazione che richiedono “privilegi esclusivi” per poter eseguire operazioni che non sono (normalmente) permesse agli utenti qualsiasi.

Approfondiremo più avanti la questione dei privilegi, per il momeno ci basti sapere che ad un utente sono permesse, di default, solo operazioni sui file di cui detiene la proprietà e azioni che non possano mettere a rischio la sicurezza del sistema.

Per poter eseguire operazioni potenzialmente rischiose è necessario acquisire temporaneamente l “privilegi di root”, accedendo al sistema come amministratore o meglio come **super user**, il quale può compiere qualsiasi azione su qualsiasi file del sistema.

L'accesso da super user si ottiene tramite il comando **su** e si rilascia tramite il comando **exit**, la password per accedere com super-user è quella scelta in fase d'installazione per l'utente di root.




D'ora in avanti vedremo come si gestiscono I principali aspetti del sistema e er farlo utilizzeremo principalmente la riga di comando, vedremo quindi alcuni comandi bash ed il loro utilizzo.

In generale un comando è composto da 3 parti:

- il comando
- le opzioni
- il (o i) target

Il **comando** corrisponde al nome di un eseguibile binario, ossia un file compilato per il sistema, che esegue un determinato software. Alcuni comandi sono condivisi nel sistema e sono identificati da una parola chiave (il nome del comando) unica per cui per chiamarli è sufficiente digitarla. Altri comandi sono invece corrispondenti a script binari non condivisi per cui è necessario specificare il percorso del file binario.

Le **opzioni** specificano il comportamento del comando. Un comando può non accettare opzioni come può accettarne molte e in generale si “scoprono” le funzionalità di un comando offerte attraverso le opzioni man mano che lo si utilizza, leggendo la documentazione.

Il (o i) **target** è l'obiettivo su cui agisce Il comando. Anche n questo caso possono esistere comandi che eseguono operazioni senza bisogno che venga specificato un target così come esistono comandi che possono lavorare su tag multipli o con comportamenti iterati, ricorsivi, ecc.

Per conoscere le capacità di un comando si può utiizzare la documentazione online (dalle guide ufficiali ai forum) ma si hanno a disposizione anche delle importanti risorse direttamente sul sistema, costituite dalla documentazione offline.

La fonte principale di documentazione offline è costituita dai manuali dei comandi, brevi descrizioni concise delle funzionalità di un comando,spesso con esempi di utilizzo. Si accede al manuale di un comando tramite il comando **man *nome_comando***.

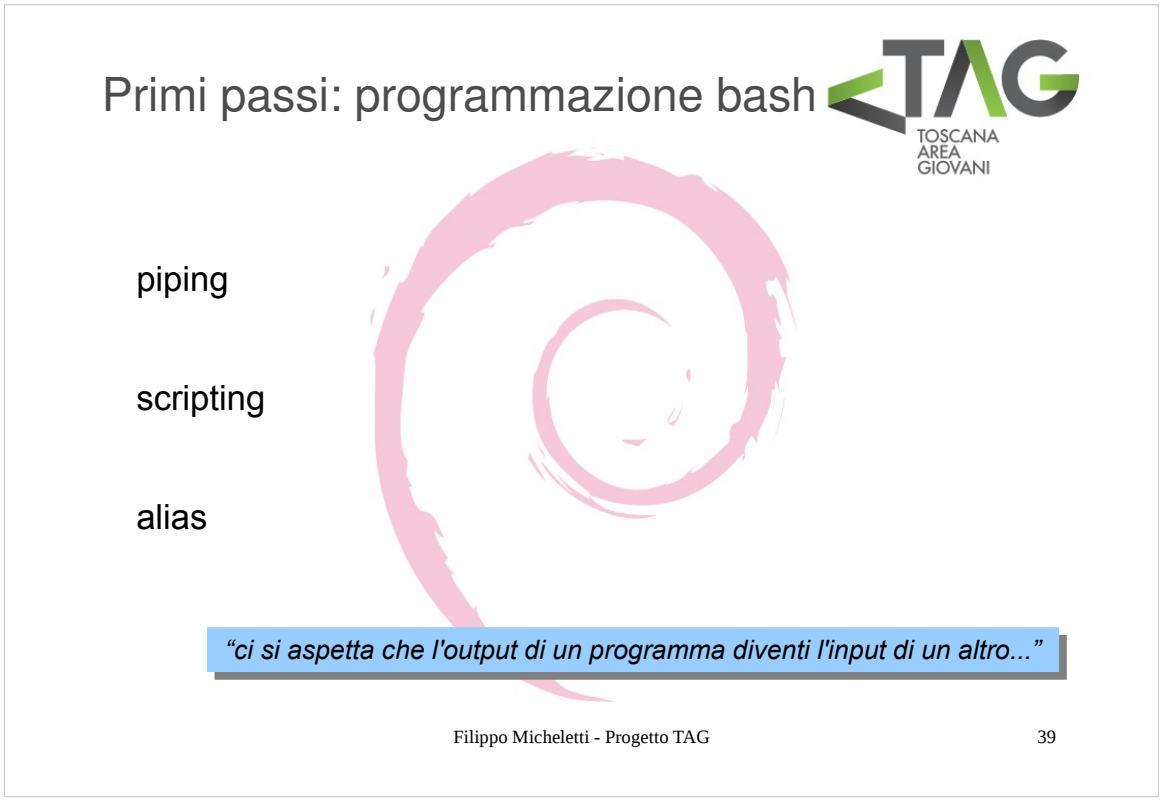
Molti comandi hanno a disposizione anche una documentazione più dettagliata, visionabile tramite **help *nome_comando*** e **info *nome_comando*** .

Un'altra caratteristica utile della bash è l'**autocompletamento** (**autocompletion**), che permette di non doversi ricordare con precisione il nome di qualsiasi comando e di velocizzarne moltissimo la digitazione. L'autocompletamento si esegue premendo il tasto di tabulazione quando si è iniziato a scrivere il nome del comando. Le impostazione dell'autocompletamento sono contenute nel file /etc/bash_completion .

Infine tramite Il comando **history** è possibile visionare la cronologia dei comandi utilizzati (contenuta nel file ~/.bash_history )

Mensioniamo subito inoltre il comando **clear**, che pulisce la schermata.

Dovrebbero iniziare ad essere chiari certi vantaggi della riga di comando...




Tra le idee su cui è stato sviluppato UNIX si trova quella per cui l'output di un programma diventi l'input di un altro.

Questo semplice ragionamento può sembrare logico e scontato ma non lo è affatto. Nel tempo infatti, per motivi legati alla corsa a sviluppare software che offrissero sempre più funzionalità ed invogliassero da un lato l'utente a preferire quel dato programma rispetto ad uno concorrente, dall'altro a giustificarne il costo, si è affermata la tendenza ad integrare in un unico software più funzionalità possibili,la maggior parte delle quali spesso di dubbia utilità o di poca attinenza con lo scopo principale dell'applicativo.
D'altronde un approccio grafico spinge di per sé all'integrazione di più operazioni sotto un'unica interfaccia.

Nei sistemi Linux-based non è così. I software rispettano fortemente l'idea riportata in slide e la maggio parte degli applicativi sono binari di piccola entità che accettano un'input, eseguono un'(unica) operazione e producono un output.

Operazioni più elaborate sono eseguite tramite l'esecuzione sequenziale di comandi.

Questa modularità rende il sistema molto più flessibile, sicuro ed aggiornabile.

I modi di far “cooperare” più comandi sono tanti, accenniamoi più immediati: il piping e lo scripting.

Per **piping** s'intende il passagio diretto dell'output di un comando all'input di un altro, come se vi fosse un tubo.

Non sempre questo passaggio può essere effettuato direttamente e il piping è comodo per mettere in cascata rapidamente due o più comandi una tantum.

Per ora interessiamoci al piping nella sua forma più semplice, che consiste nello scrivere in cascata I comandi separati dall'operatore | .

Per **scripting** si intende invece una forma più articolata di esecuzione sequenziale di comandi, con la possibilità di utilizzare regole sintattiche di un vero e proprio linguaggio di programmazione di alto livello.

In uno script bash I comandi da eseguire sono scritti sequenzialmente in un file che inizia con la riga: #!/bin/bash .

Lo scripting è una tecnica abbastanza elaborata che non si può imparare certo in pochi minuti quindi per ora è sufficiente sapere cos'è uno script. Se ci sarà tempo approfondiremo un pochino l'argomento alla fine del corso.

Infine un modo semplice di “raggruppare” uno o più comandi sotto un'unico nome è quello di usare il comando **alias**, che permette di definire un nuovo comando.

es

```
alias spegni="shutdown –h now"
```

## Primi passi: la bash paranoia...

evitiamola!

… nonostante tutto: nano

*nano project (EN)*



Ok, la riga di comando è potentissima e al prezzo di un piccolo sforzo iniziale per “disabituarsi” ad avere una finestrella per qualsiasi cosa permette di lavorare con un'immediatezza assoluta.

E' bene però ricordarsi che esiste anche l'interfaccia grafica.

In particolare è importante imparare ad **alternare** l'utilizzo delle due shell in funzione dell'applicazione, ma soprattutto **non fissarsi** a voler eseguire da riga di comando operazioni “intrinsecamente grafiche”.

Alcuni applicativi infatti sono imprescindibili dalla grafica (tutto ciò che riguarda immagini e video) o si sono fortemente orientati ad essa (tutto ciò che riguarda il web) per cui non ha senso cercare in qualche modo di ricondurne l'utilizzo alla riga di comando (per es. se dovete aprire un browser la cui icona è sul desktop apritela e basta senza lanciare da terminale il comando corrispondente...).

In altri casi invece l'utilizzo di un'interfaccia grafica agevola fortemente le cose (il caso più frequente è sicuramente quello dell'editing di un testo).

Per cui, sintetizzando: **evitiamo la bash paranoia!**

Nonostante questo però è bene conoscere anche come si possono svolgere da riga di comando alcune operazioni di norma più comode da interfaccia grafica, in particolare, in merito proprio all'editing di testi, sarebbe bene saper utilizzare (almeno) un editor senza interfaccia perché spesso può capitare di dover editare un file di configurazione senza aver a disposizione un'interfaccia grafica... e allora Nano!

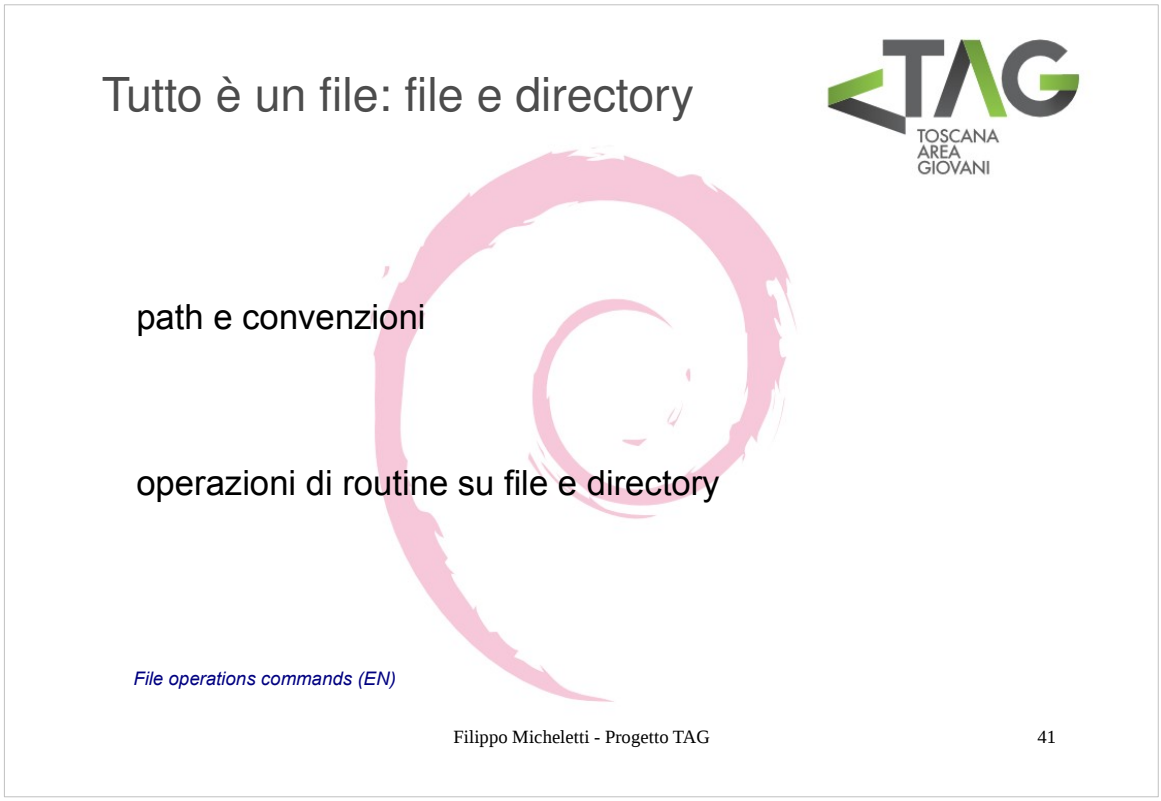
**Nano** è il text-editor testuale ufficiale del progetto GNU, nato come alternativa libera a Pico.

L'utilizzo di Nano può apparire inizialmente un po' ostico ma è piuttosto semplice:

- ci si sposta nel teso utilizzando le frecce
- si cancella normalmente tramite I tasti backspace e delete
- si salva tramite la sequenza Ctrl+o
- si esce con Ctrl+x

Ovviamente dovendo editare un file di testo in condizioni normali avendo a isposizione un'interfaccia grafica è consigliabile non complicarsi la vita inutilmente, ma quando l'interfaccia non parte o vi connettete in remoto a una macchina che non ce l'ha...

Impara Nano e mettilo da parte!




Gestione di file e directory

I comandi principali per la gestione di file e directory sono:

| | |
|---|---|
| **cp** [opzioni] sorgente destinazione | copia file e directory |
| **mv** [opzioni] sorgente destinazione | sposta e rinomina file o directory |
| **rm** [opzioni] file | elmina file e directory |
| **ls** [opzioni] [path] | elenca contenuto di file e directory |
| **tree** [opzioni] directory | elenca contenuto di file e directory con visualizzazione ad albero |
| **cd** directory | cambia directory corrente |
| **mkdir** [opzioni] directory | crea una directory |
| **rmdir** [opzioni] directory | climina una directory |

**ATTENZIONE:** su Linux come su Unix, di default, non esiste un concetto analogo al **cestino**, i file cancellati sono (quasi) irrimedibilmente perduti per cui prestate attenzione a quello che fate!

Per quanto riguarda il path (percorso) di file e directory vale la pena di ricordare alcune cose:

| | |
|---|---|
| **/** | Indica la root, la dir principale alla base di tutto il filesystem |
| **..** | La directory del livello superiore al corrente (directory parent) |
| **.** | La directory corrente |
| **~** | La home directory dell'utente corrente |

Un **percorso assoluto** parte sempre dalla directory di root: /etc/fstab
Un **percorso relativo** parte sempre dalla directory corrente: cd .. o ./nome_file

Infine può tornare utile il comando **pwd** che restituisce il path corrente (normalmente il path è indicato nel prompt della shell, ma quando diviene lungo viene automaticamente abbreviato).

## Tutto è un file: links



link fisico

link simbolico



Un **link** (collegamento) ad un file o ad una directory non è altro che un alias per quel file/directory, ossia un oggetto (eventualmente di nome diverso) che permette di accedervi.

Tutte le informazioni riguardanti un oggetto contenuto in un filesystem (file o directory) sono contenute in un inode del filesystem.

Nei sistemi Unix based si hanno 2 tipi di link:

- hard link
- soft link

Un **hard link**, che può essere fatto solo per un file, condivide lo stesso inode del file linkato (un hard link può quindi trovarsi solo sullo stesso filesystem contenente il file linkato).
L'hard-link è realmente solo un alias per il file linkato.

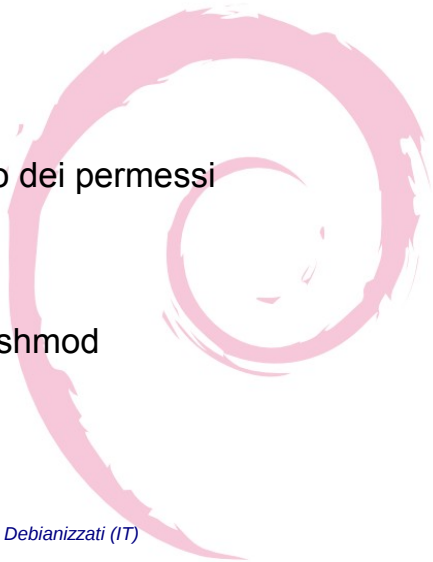
Un **soft link** è invece in pratica un file locato nel path del link che contiene "l'indirizzo" (il path) dell'oggetto linkato.
Di conseguenza un soft link può essere fatto per qualsiasi oggetto e può trovarsi anche su un filesystem diverso da quello dell'oggetto linkato.

Hard link e soft link si creano entrambi con il comando **ln**, ma passando l'opzione -s per I link simbolici.

| | |
|---|---|
| ln nome_file [nome_link] | crea un link fisico |
| ln -s nome_file [nome_link] | crea un link simbolico |

NB: nome_file e nome_link contengono anche il path dell'oggetto da linkare; nome_link può essere omesso ottenendo un link nella path corrente dello stesso nome dell'oggetto linkato.

## Tutto è un file: permessi



Funzionamento dei permessi

chown, chgrp, shmod

*Guida ai permessi sui file on Debianizzati (IT)*



Linux è un sistema **multiutente**.
In ambinete Linux vale una regola: tutto è un file.

Per rendere il sistema sicuro facendo in modo che chiunque possa modificare solo ciò che gli compete a ciascun file sono associati 3 tipi di permesso:

- **lettura** (r)
- **scrittura** (w)
- **esecuzione** (x)

Esistono 3 livelli di utenza più uno "speciale":

- **utente** proprietario (u)
- **gruppo** proprietario (g)
- gli **altri** (o)

Il livello speciale è l'utente di **root** (amministratore del sistema) che possiede tutti i permessi su tutti i file del sistema.

Per conoscere i premessi associati ad un file si utilizza il comando ls con l'opzione -l:

ls -s nome_file

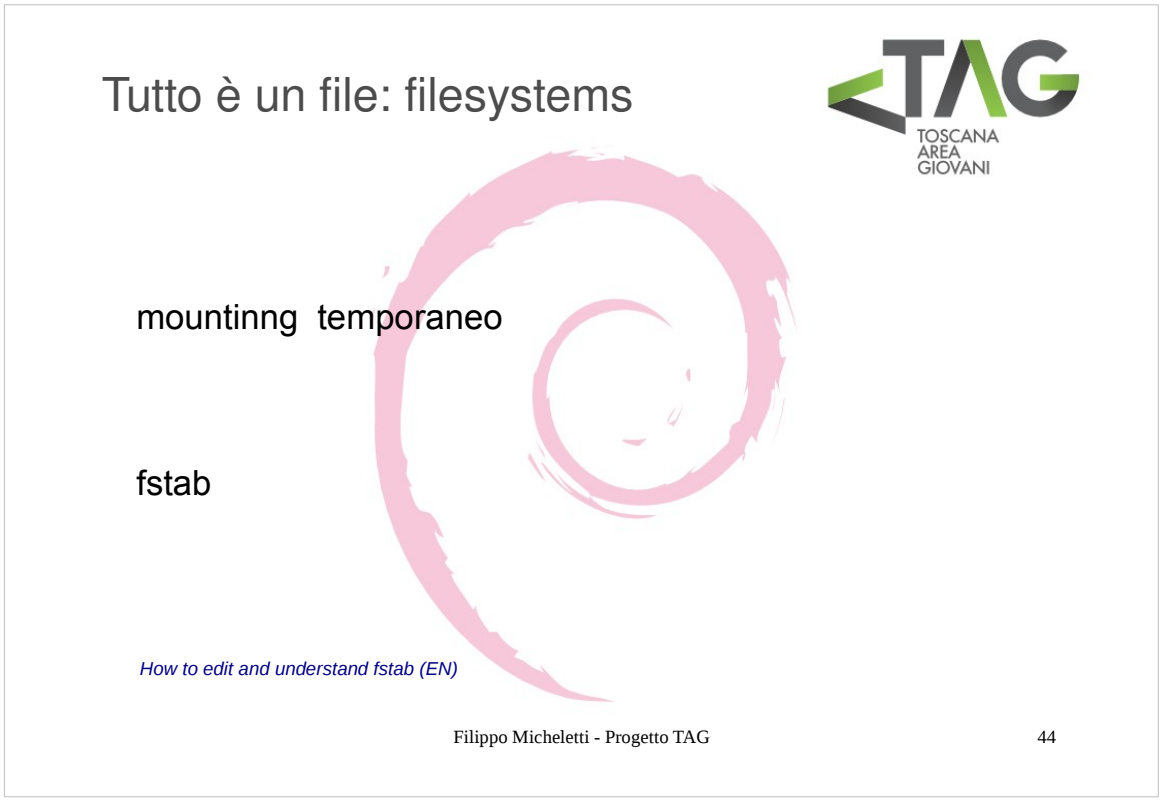
Es: -rw-r--r-- 1 root root 77266 Dec 13 17:18 /etc/passwd

L'output esteso di ls dà varie informazioni sul file

- attributi (il primo carattere a sinistra);
- permessi (i successivi 9 caratteri, raggruppati per 3, indicano rispettivamente i permessi per l'owner, per il gruppo e per gli altri utenti (in questo caso l'owner può scrivere e leggere, e gli altri possono solo leggere);
- numero di file con lo stesso inode (in questo caso 1);
- nome dell'owner (in questo caso root);
- nome del gruppo (in questo caso root);
- dimensioni in byte del file (in questo caso 77266);
- data dell'ultima modifica (13 Dicembre dell'anno in corso alle 17:18);
- nome del file ( /etc/passwd ).

Per modificare i premessi si utilizzano i comandi

**chown** [opzioni] [utente] nome_file cambia il proprietario del file nome_file
**chgrp** [opzioni] [utente] nome_file cambia il gruppo proprietario del file nome_file
**chmod** [opzioni] [utente] nome_file cambia I permessi associati al file nome_file




Ciascuna unità disco per poter essere accessibile comtiene un filesystem, in parole povere un sistema di organizzazione dei dati in file e directory che mette a disposizione le funzioni di accesso ai dati contenuti nel disco (lettura, scrittura, creazione, rimozione, ecc ecc di files e directories).

Per poter essere utilizzato un filesystem deve essere montato in un punto del sistema, detto punto di mount.

Il punto di mount può essere una qualunque directory del sistema, anche non vuota (in questo caso tuttavia il contenuto della directory non è più accessibile fin quando il filesystem non viene smontato).

I comandi principali per montare e smontare un filesystem sono:

| | |
|---|---|
| **mount** -t [tipo fs] [opzioni] device dir | Monta un dispositivo a blocchi su un file system |
| **umount** [ opzioni ] device | Smonta un dispositivo (necessario) |
| **df** [opzioni][file] | Verifica lo spazio libero su disco |
| **du** [opzioni][file] | Visualizza lo spazio occupato da file e directory |
| **fsck** [opzioni] dispositivo | Verifica l'integrità e ripara il Filesystem |
| **mkfs** [opzioni] dispositivo | Crea un Filesystem (formatta) |

Il file **fstab**
Montare manualmente un filesystem è un'operazione che ha senso per dispositivi normalmente non collegati alla macchina.
Quando si hanno unità disco normalmente collegate al computer (ad esempio più hard-disk o più partizioni) si può far sì che queste vengono montate in automatico, con le impostazioni preferite, all'avvio editando il file fstab.

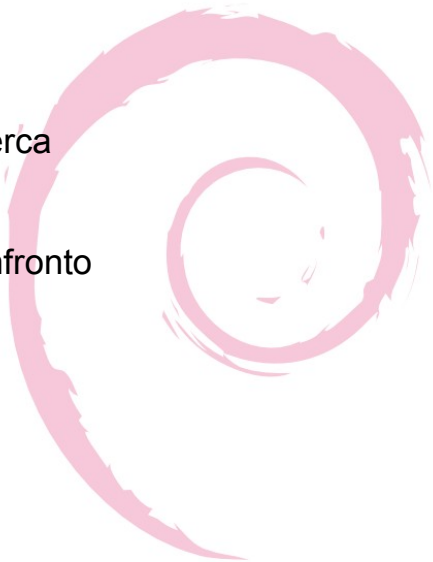
Nel file /etc/fstab vengono configurate le informazioni sui vari file system preimpostati sul sistema, vengono definiti i mount point, il tipo di file system ed altre informazioni.

La sintassi per il file /etc/fstab è la seguente:

dispositivo | mount point | filesystem | opzioni | backup flag | check flag

**Problemi comuni** con I filesystem
I problemi più comuni riguardano la compatibilità del filesystem con il sistema.
Alcuni filesystem non sono supportati nativamente da Linux per motivi di diritti: ntfs ad esempio è un filesystem proprietario Microsoft per cui non si ha alcun supporto nativo.
Esistono comunque dei pacchetti appositamente creati per utilizzare i filesystem più diffusi, sempre per ntfs ad esempio esiste il pacchetto ntfs-3g, contenuto nel repository non-free.

## Tutto è un file: ricerca e confronto



comandi di ricerca

comandi di confronto

crittografia

*Examples using grep (EN)*



Su Linux esistono molteplici comandi per la ricerca e il contronto di file.

I principali comandi per la ricerca sono

| | |
|---|---|
| **find** [path][expression] | Ricerca di file o directory in tutto il file system in base a un certo numero di criteri, come il nome, la data di creazione e la dimensione. |
| **locate** filename | Ricerca file o directory tramite il db costruito da updatedb |
| **updatedb** | Crea/Aggiorna un database (contenuto in /var/lib/slocate/slocate.db ) contenente tutti i path di tutti i file nel file system |
| **whereis** [opzioni] filename | Visualizza i path di binari,sorgenti e manuali per un comando |
| **sort** [opzioni] [file] | Ordina con un certo criterio le righe di un file ASCII |
| **strings** [opzioni] filename | Cerca pattern di testo nei file binari |
| **grep** | Cerca righe contenenti un pattern specificato |

es

| | |
|---|---|
| grep "parola_chiave" /path_file | cerca la stringa parola_chiave nel file path_file |
| grep "altra_parola" -r /path | cerca la stringa altra parola ricorsivamente in tutti i file contenuti nell'albero con radice la directory in /path |

Comandi di confronto file

| | |
|---|---|
| **cmp** file1 file2 | confronta i file file1 e file2 byte a byte |
| **diff** file1 file2 | confronta i file di testo file1 e file2 per righe |

Infine è utile conoscere l'utilizzo del tool di generazione e verifica di chiavi crittografiche più diffuso: **md5sum**

md5sum è programma per la creazione e verifica di chiavi crittografiche (hash) tramite l'algoritmo MD5 (Message Digest 5).

Le hash-keys sono stringhe alfanumeriche calcolate elaborando un file attraverso un algoritmo specifico con lo scopo di verificarne l'integrità: al momento della creazione del file viene calcolata la md5sum; quando il file viene distribuito assieme ad esso viene fornita anche la md5sum calcolata, così chi lo riceve può calcolarla nuovamente e confrontare la propria con quella fornita: se le due chiavi non coincidono il file è corrotto, altrimenti è a posto.

Creazione della md5sum

md5sum <nome_file> > <file_chiave> (NB: l'estensione del file chiave è indifferente)

Verifica automatica della md5sum

md5sum -c <file_chiave> (NB: il file chiave ed il file origine devono trovarsi nella stessa directory)

# Tutto è un file: regular expression

RegEx

metacaratteri

*Regular Expression on Wikipedia (EN)*



Le **Regular Expressions** (RegExp) sono un sistema di regole rivolte alla creazione di pattern di ricerca utili nella ricerca di stringhe.

Sebbene possa sembrare un'applicazione limitata, la ricerca di stringhe è in realtà una delle operazioni più frequenti in qualsiasi applicazione dell'informatica.

Nella ricerca di stringhe tipicamente si passa la stringa cercata (chiave) in argomento ad un tool di ricerca (come grep), insieme all'indirizzo in cui cercare (path di ricerca): nella la chiave la maggior parte dei caratteri corrisponde semplicemente a sé stessa, mentre alcuni caratteri assumono un significato assimilabile ad un comando per la ricerca. Questi caratteri particolari vengono detti **metacaratteri**, che, utilizzati insieme alle chiavi di ricerca, formano le regular expressions.

Metacaratteri di definizione

| | |
|---|---|
| **.** | identifica un singolo carattere |
| **^** | identifica l'inizio riga |
| **** | escape, neutralizza il significato del metacarattere seguente (ad esempio se la chiave contiene un \, allora questo andrà scritto come \\) |
| **$** | identifica la fine della stringa |
| **[]** | identifica qualsiasi carattere indicato tra parentesi |
| **[^]** | identifica tutti i caratteri non specificati nell'insieme |
| **[0-9]** | identifica ogni carattere compreso nell'intervallo tra 0 e 9 compresi (sono ammessi sia intervalli di numeri che di lettere) |

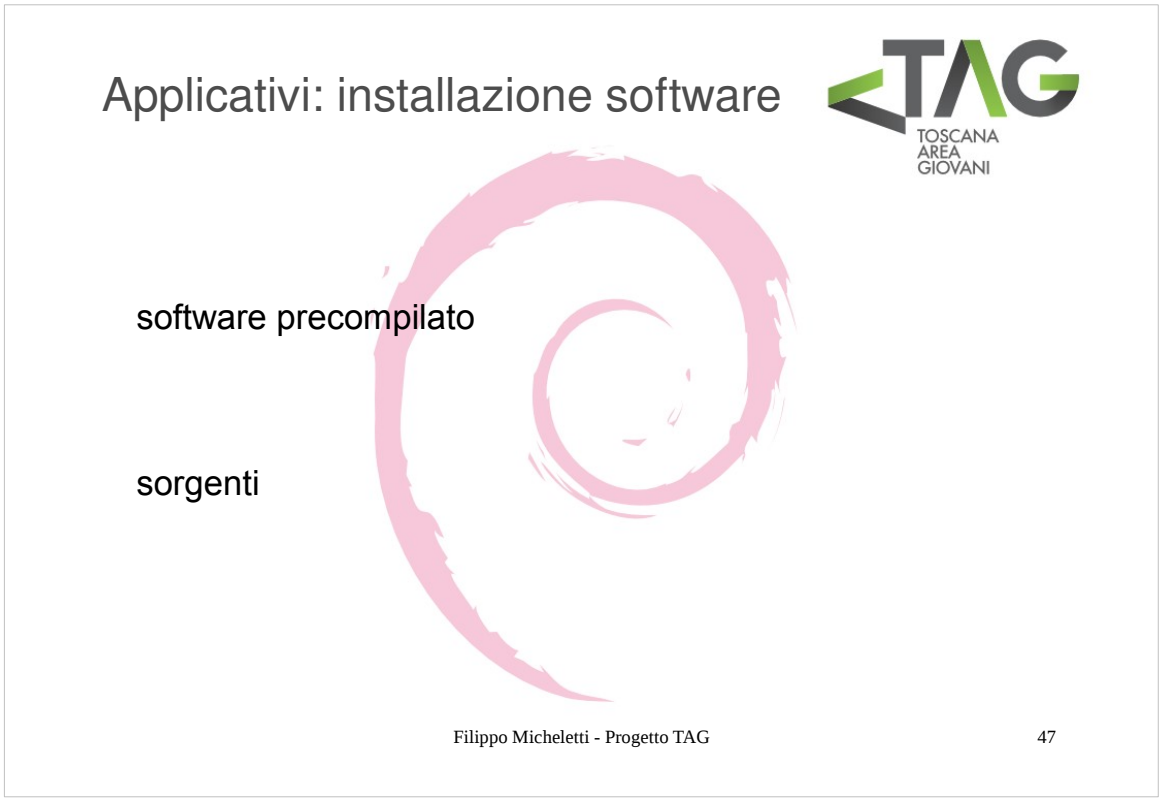
Modificatori

| | |
|---|---|
| **?** | zero o una volta |
| **+** | una o più volte |
| ***** | zero o più volte |
| **\|** | or logico, indica uno o l'altro elemento prima e dopo il segno \| |

esempi

| | |
|---|---|
| egrep -i '(string1\|string2)' file | Ricerca e visualizza in un file string1 e string2 |
| grep '^1' list.txt | Ricerca in list.txt le righe che iniziano con 1 |
| egrep '^2[234]' list.txt | Ricerca le righe che iniziano con 22,23,24 |
| grep '^Linux$' list.txt | Ricerca le righe che contengono SOLO la parola Linux |
| grep -c '^$' list.txt | Ricerca il numero di righe vuote in list.txt |
| grep '^[^0-9]' list.txt | Ricerca le righe che NON iniziano con un numero |
| grep '\<[Ll]inux\>' list.txt | Ricerca le righe che contengono la parola singola Linux o linux, ma non visualizza quelle con parole che CONTENGONO la chiave di ricerca, come per esempio, LinuxOS |

Le regular expression possono essere utilizzate con moltissimi dei comandi più diffusi.




Come abbiamo visto un software "nasce" da un codice sorgente scritto con un determinato linguaggio di programmazione; è la compilazione a generare un file binario "adatto" ad essere eseguito nel sistema per cui è avvenuta la compilazione.

Ne segue che  quindi esistono due modi per ottenere del software: tramite il codice sorgente o tramite il codice precompilato.

**Installazione di software precompilato**

Iniziamo da qui in quanto il software precompilato è l'unica modalità con cui viene distribuito il softare proprietario (o comunque non open-source) e ognuno di noi che abbia avuto a che fare con un computer si sarà trovato almeno una volta nella vita a dover installare qualche programma.

Il software precompilato viene di solito distribuito sotto forma di **pacchetti**.
Il formato standard dei pacchetti Debian è **.deb**: un pacchetto .deb è unn sostanza un archivio ottenuto dalla concatenazione di due sottoarchivi, uno contenente il codice compilato e uno dei dati aggiuntivi necessari all'installazione.

L'installazione di pacchetti può essere eseguita in due modi:

- **manualmente**
- **automaticamente**

In generale la gestione automatica dei pacchetti è preferibile perché offre degli indubbi vantaggi, in particolare:

- soddisfacimento automatico delle dipendenze
- aggiornamento automatico dei pacchetti

Tuttavia non tutto il software di cui si può avere bisogno è disponibile su sistemi di gestione automatica per cui in alcuni casi l'installazione manuale è necessaria.

**Installazione di software da sorgente**

Ci dovrebbe essere ormai chiaro che un sorgente ha bisogno prima di tutto di esseere compilato e solo in seguito, se questa operazione è andata a buon fine, installato.

L'installazione di software da codice sorgente è pertanto in generale meno automatizzata e di solito si preferisce quando non è disponibile un pacchetto precompilato per la propria architettura o si vogliono impostare /modificare alcune parti del codice per ottenere software customizzati (ad esempio per ottimizzare l'utilizzo delle risorse disponibili sul proprio sistema).

## Applicativi: APT



APT

*APT Debian wiki (EN)*



Esistono diversi front-end per la gestione automatica dei pacchetti sia dotati d'interfaccia grafica, come synaptic, che da riga di comando, come aptitude, dselect e APT, noi facciamo riferimento ad **APT** (Advance Package manager Tool) in quanto è sicuramente uno dei più versatili.

NB: la direzione di sviluppo è quella di portare aptitude ad integrare apt, attualmente tuttavia aptitude risulta più una complicazione che un'evoluzione di apt per cui a parer mio conviene l'utilizzo di quest'ultimo.

APT offre una serie di tool per la manipolazione dei pacchetti, vediamo i principali.

| | |
|---|---|
| **apt-get install** pacchetto_1 pacchetto_2 ... | installa i pacchetti in argomento |
| **apt-get -t** nome_repository **install** p1 p2 ... | installa l pacchetti dal repository specificato |
| **apt-get -f install** | installa le dipendenze mancanti ad una precedente installazione |
| **apt-get remove** pacchetto_1 pacchetto_2 … | rimuove i pacchetti in argomento (tutti i file associati tranne i file d'impostazione) |
| **apt-get purge** pacchetto_1 pacchetto_2 ... | rimuove i pacchetti in argomento (tutti i file associati) |
| **apt-get autoremove** p1 p2 … | rimuove i pacchetti in argomento con tutte le dipendenze non necessarie ad altri pacchetti installati |
| **apt-get source** pacchetto_1 pacchetto_2 ... | preleva i sorgenti dei pacchetti in argomento (necessario abilitare i repository deb-src) |
| **apt-get update** | aggiorna la lista dei pacchetti contenuti nei repository |
| **apt-get upgrade** | esegue l'upgrade dei pacchetti installati sul sistema (solo quelli installati!) |
| **apt-get dist-upgrade** | esegue l'upgrade dei pacchetti installati sul sistema eventualmente installandone di mancanti per soddisfare nuove dipendenze degli aggiornamenti dei pacchetti installati |

Downgrade di un pacchetto

http://www.crazysquirrel.com/computing/debian/downgrade.jspx

La **cache** di APT

APT genera di default una cache locale (in /var/cache/apt/archives) mantenendo i pacchetti scaricati dai repository per facilitarne la reinstallazione e limitare lo spreco di banda.

Nella maggior parte dei casi questi pacchetti non serviranno più per cui è buona norma tenere pulita la cache tramite i comandi

| | |
|---|---|
| **apt-get autoclean** | rimuove dalla cache i pacchetti obsoleti |
| **apt-get clean** | elimina tutta la cache |


La gestione automatica dei pacchetti in Debian è basata sui **repository**: depositi di pcchetti da cui è possibile attingere per aggiornare il proprio software, installarne di nuovo o ottenerne i sorgenti.

APT, così come tutti gli altri strumenti di gestione automatica dei pacchetti, attingono dai repository per ottenere I pacchetti.

Da un punto di vista del contenuto esistono 2 tipi di repository:

- **deb**: contengono i pacchetti precompilati
- **deb-src**: contengono il codice sorgente

NB: ogni repository (o perlomeno di ogni repository contenente software open source) contiene per ciascun pacchetto sia la versione compilata per la propria architetture (nel repositori deb) che i sorgenti (nel deb-src). Se non si è interessati allo sviluppo i repository deb-src possono essere per il momento tralasciati.

Dal punto di vista della distribuzione i repository possono onvee essere classificati in:

- **ufficiali**
- **non ufficiali**

Nei repository ufficiali sono contenuti i pacchetti ufficialmente riconosciuti dalla distribuzione Debian, ad esempio perché soddisfano alle DFSG, o perché sono testati e approvati per una specifica distribuzione.

I repository ufficiali si dividono in 3 sezioni:

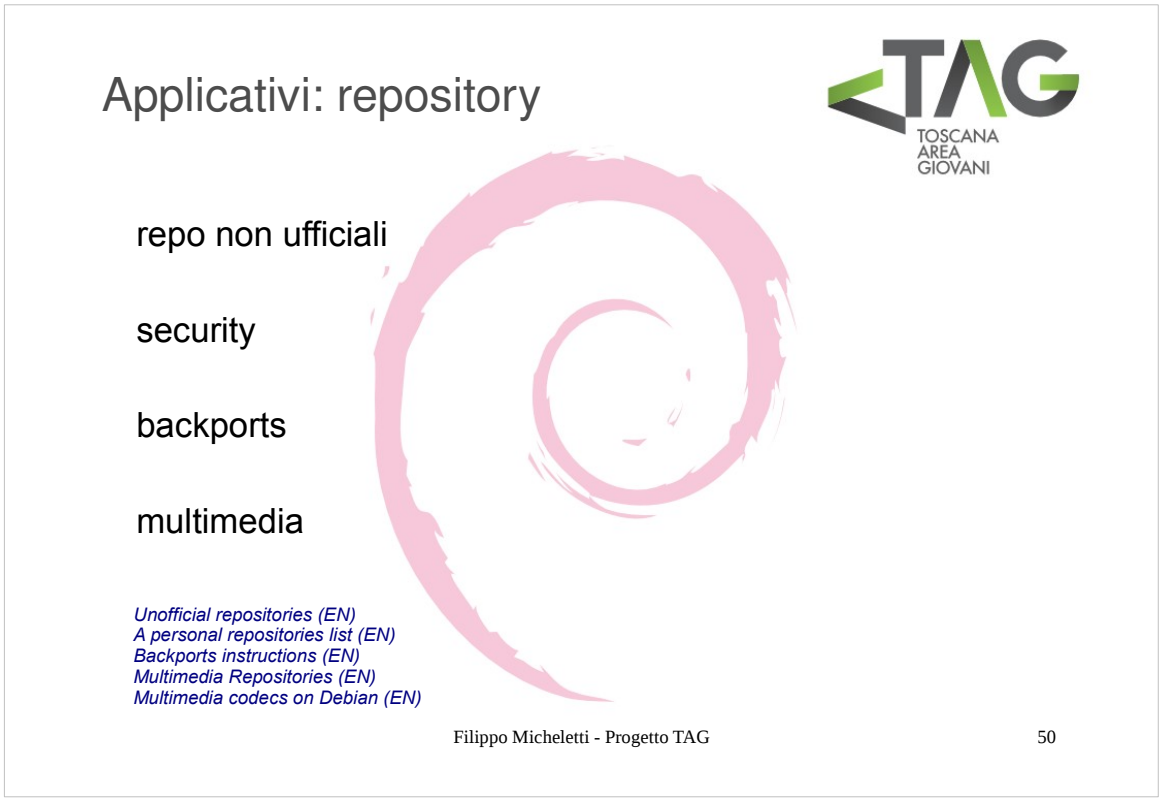
- **main**: solo software libero senza alcuna dipendenza non libera
- **contrib**: software libero ma dipendente da software non libero
- **non-free**: software non libero (ma comunque distribuito gratuitamente)

E' possibile far convivere nel proprio file sources.list stessi repository di distribuzioni o versioni diverse: di default apt sceglie quello della propria distribuzione, ma è possibile specificare da dove si vuole attingere.

I repositories permettono di mantenere aggiornato il sistema in maniera semplice e veloce

NB: solo il software installato da repo verrà aggiornato! Per il software installato manualmente l'aggiornamento può avvenire solo installando una versione più recente.

Qualche tempo fa si è parlato della possibilità di seguire una release o una distribuzione.
Per seguire una distribuzione è sufficiente che la chiave passata ai repository sia il nome della distribuzione da seguire, ad esempio per Squeeze, come nell'esempio riportato in slide, seguire una release è altrettanto semplice, passando il nome del tipo di release come chiave, ad esemio stable.




I repository di cui abbiamo parlato finora sono quelli **ufficiali** del progetto Debian: essi contengono solo ed esclusivamente software approvato e mantenuto dalla comunità di sviluppo del progetto e soddisfacente alle DFSG (o a parte di esse per la sezione non-free). Il software contenuto in questi repository è quindi da considerarsi attendibile e compatibile al 100%.

Esistono poi moltissimi repository **non ufficiali** che distribuiscono o redistribuiscono software di ogni natura, mantenuti da gruppi di sviluppo, associazioni o addirittura singoli individui (potete farne uno voi stessi). Per utilizzare questi repository è sufficiente aggiungerli al file sources.list con la dovuta attenzione riguardo alla provenienza e attendibilità del software offerto.

**Security**
Esiste un repository per la sicurezza del sistema:

```
deb http://security.debian.org/ squeeze/updates main contrib non-free
```

I **backports**
Se si utilizza la versione stabile spesso si incontra il problema della mancanza di pacchetti aggiornati adatti a soddisfare le dipendenze di software esterno alla distribuzione. Per sopperire a questa mancanza esiste il repository backports, che contiene una serie di pacchetti, principalmente legati allo sviluppo, provenienti dai progetti testing e unstable, ma ricompilati per stable. Di conseguenza il repository backports esiste solo per la release stable e si riferisce alla specifica distribuzione.

Per avere a disposizione i backports aggiungere al proprio sources.list la riga

```
deb http://backports.debian.org/debian-backports squeeze-backports main
```

In definitiva i backports sono una (buona) soluzione per avere pacchetti più recenti su una distribuzione stable. Installando un pacchetto dai backports è comunque bene tenere presente che si introduce un elemento di instabilità nel sistema (i pacchetti provenienti da testing e unstable possono contenere bug con più probabilità rispetto a quelli della stable) per cui è buona regola limitare il più possibile l'utilizzo di pacchetti provenienti da questa fonte.
Per questo motivo, anche una volta aggiunti i backports al proprio sources.list, i pacchetti contenuti in questo repository sono ignorati di default da apt, e se si vogliono installare è necessario esplicitarlo tramite l'opzione -t

```
apt-get install -t squeeze-backports <nome_pacchetto>
```

**Multimedia**
Nell'ambito della multimedialità le licenze giocano un ruolo pesante.
Nonostante esistano plugin open-source per praticamente qualsiasi formato ci si trova a volte ad avere la necessità di ottenere codec per formati proprietari, sopratutto legati a Windows (es. w32codecs).
Questi pacchetti non sono inclusi nella distribuzione ufficiale di Debian per ovvi motivi, ma si possono aggiungere tramite i repository multimedia:

```
deb ftp://ftp.deb-multimedia.org squeeze main non-free
```

# Applicativi: installazione manuale



dpkg

Pacchetti tar



Il gestore di pacchetti .deb di Debian è **dpkg** (Debian PacKaGe manager).

dpkg permette di eseguire molte operazioni sui pacchetti, fra queste le più importanti sono l'installazione, la rimozione e l'elenco dei pacchetti installati.

| | |
|---|---|
| **dpkg -i** nome_pacchetto | installa il pacchetto nome_pacchetto |
| **dpkg -r** nome_pacchetto | rimuove il pacchetto nome_pacchetto (tutti i file associati tranne i file d'impostazione) |
| **dpkg -P** nome_pacchetto | rimuove il pacchetto nome_pacchetto (tutti i file associati) |
| **dpkg --get-selections** | elenca i pacchetti presenti sul sistema ed il relativo stato. |
| **dpkg -L** nome_pacchetto | elenca i file sul sistema facenti parte del pacchetto nome_pacchetto |
| **dpkg --version** nome_pacchetto | restituisce la versione del pacchetto nome_pacchetto |
| **dpkg --print-architecture** | restituisce l'architetture della macchina |

Pacchetti distribuit in formato archivio
I pacchetti precompilati vengono distribuiti anche sotto forma di **archivio**, compresso (.tar.gz o .tar.bz2) o non compresso (.tar).

In questo caso per installare il pacchetto è necessario dearchiviarne il contenuto e lanciare lo script d'installazione che si trova al suo interno, che di solito ha uno dei seguenti nomi:

install.sh
setup.sh
install
setup
install.bin
setup.bin

o simili.


I **sorgenti** di un software vengono rilasciati sotto forma di archivio compresso o non.

Una volta dearchiviati i sorgenti è necessario **compilarli** e quindi **installarli**.

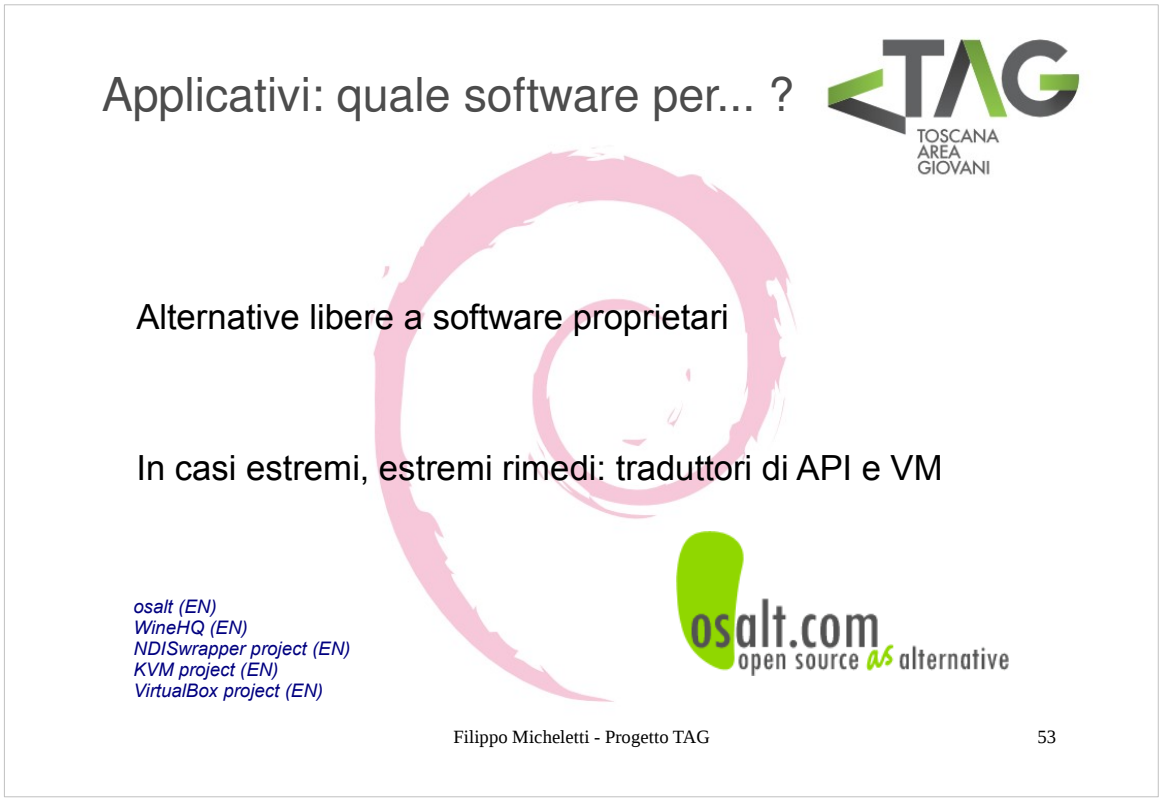
Nella cartella contenente i sorgenti è di solito presente un file README contenente informazioni sulla procedura di compilazione (ad esempio per la soluzione a problemi comunemente riscontrati o particolarità legate alla compilazione): si parte sempre leggendo questo file e impostando quanto necessario nei file di configurazione per la compilazione.

Il file di configurazione si trova sotto il nome ***configure***, e contiene uno script che imposta alcune variabili legate alla compilazione. Una volta eseguite le eventuali modifiche al file configure in base alle proprie esigenze occorre lanciarlo (ad esempio, se si è posizionati nella directory contenente I sorgenti, digitando ./configure).

NB: I parametri contenuti nel file configure sono stati decisi da chi ha scritto il software. Benché esistano delle linee guida ogni software può richiedere l'impostazione di parametri diversi e l'unico modo per conoscerli è leggere il README ed il file configure stesso. Un parametro che comunque è sempre presente è -prefix che permette di scegliere dove installare i file del software una volta compilato e installato: a meno che non si voglia installare qualcosa che debba essere disponibile per tutti gli utenti del sistema è buona norma fare in modo che tali file vadano nella cartella /usr/local, lanciando quindi il file con il comando configure -prefix=/usr/local

Se la configurazione è andata a buon fine verrà generato un **makefile**, il quale contiene le istruzioni di compilazione (e installazione) vere e proprie; si può quindi procedere alla compilazione lanciando il comando **make**.

Compilati sorgenti è possibile installare il software, tramite il comando **make install**.




Ora che sappiamo **come** installare un software ci manca solo una cosa: sapere di **quale** programma abbiamo bisogno.

Un elenco di corrispondenze tra I più noti software proprietari e gli “analoghi” liberi non avrebbe gran senso, l'importante è sapere che esistono alternative libere a qualsiasi applicativo, e che spesso la versione libera è anche più performante.

osalt.com (Open Source As Alternativa) offre un motore di ricerca studiato appositamente per trovare alternative libere a software proprietari: è sufficiente digitare il  nome del software proprietario di cui si cerca l'alternativa ibera per ottenere I risultati disponibili.

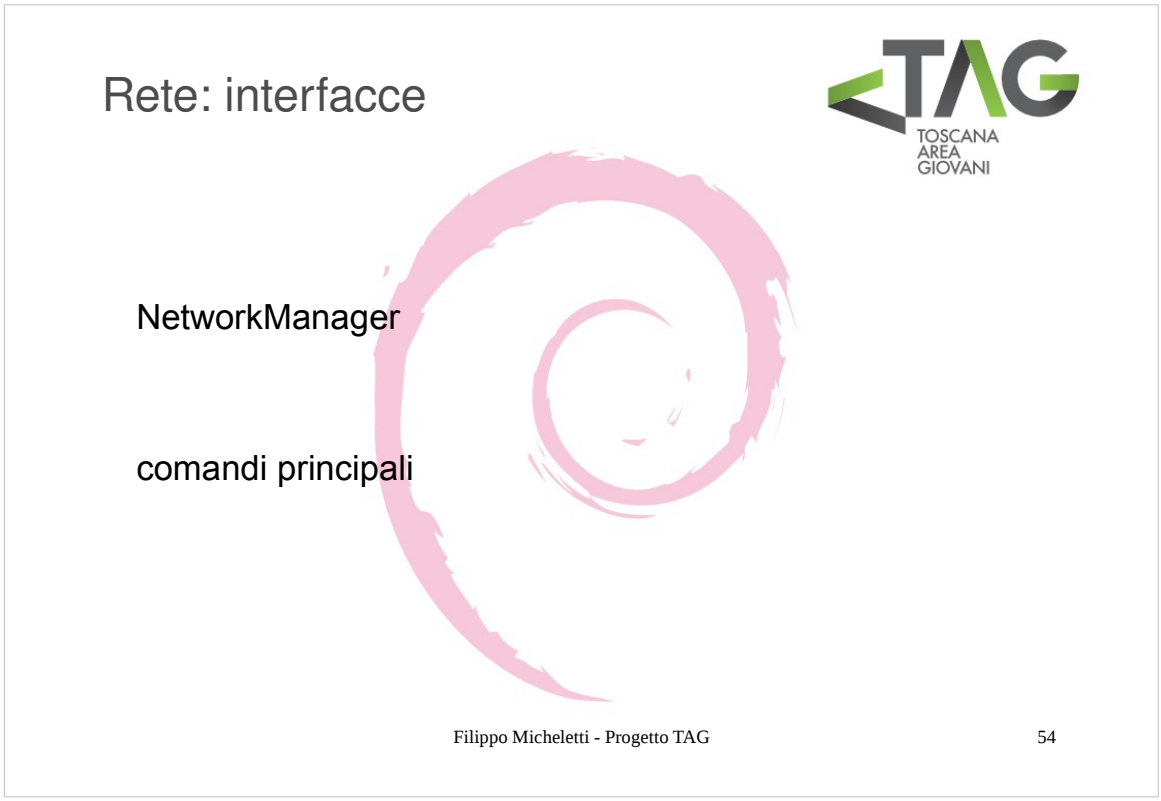
E se proprio si dovesse essre costretti ad utilizzare del software proprietario?

La maggior parte del software proprietario utilizzato ad esempio in campo professionale è ormai disponibile anche per sistemi Linux-based per cui è possibile procurarselo attraverso gli stessi canali del software utilizzato su altri SO.

Ma c'è di più. Nel caso in cui si avesse l'inevitabile necessità di utilizzare del software proprietario disponibile solo per SO proprietari è possibile “simulare” l'ambiente software per cui è realizzato l'applicativo ed utilizzarlo sul nostro sistema!

A questo scopo esistono principalmente 2 approcci:

- utilizzare un traduttore di API, che traduca letteralmente le system-call del SO per cui è realizzato il software in system-call Linux e viceversa (I tool più diffusi sono WINE per l'installazione di software Windows sotto Linux e NDISwrapper, per l'installazione di moduli del kernel (driver) Windows sotto Linux)
- virtualizzare il SO per cui è scritto il software di cui si ha bisogno all'interno del nostro sistema utilizzando una Virtual Machine (in questo caso si tratta sostanzialmente di installare quel dato SO “all'interno” del nostro e quindi installare su di esso il software; le VM più diffuse sono KVM e VirtualBox della Oracle)




L'impostazione di un'interfaccia di rete è uno dei passi fondamentali per la gestione del sistema.

La maggior parte delle operazioni “comuni” di gestione di un'interfaccia possono essere eseguite tramite interfaccia grafica, in particolare di default in GNOME si trova **NetworkManager** che offre le impostazioni essenziali in modo discreto.

In molte occasioni è comunque utilissimo saper gestire un'interfaccia di rete da riga di comando. I comandi principali sono:

| | |
|---|---|
| **ifconfig** | configura un'interfaccia di rete |
| **iwconfig** | configura un'interfaccia wireless |
| **iwlist** | ottiene (e imposta) informazioni da un'interfaccia wireless |
| **dhclient** | client dhcp |
| **-v** | verbose |
| **-r** | release |


**WEP**

Per impostare la password di una connessione protetta con crittografia WEP è sufficiente aggiungere l'opzione key <chiave> al comando iwconfig.

**WPA**

La gestione della crittografia WPA è delegata al demone **wpasupplicant**.

Il ocmando èer generare la chiave a partire da password e essid é:

```
wpa_passphrase <essid> <mypassphrase> > <file_name.conf>
```

Altrimenti, per evitare di digitare la password come parametro del comando (così che non venga salvata nell'history della bash) è possibile omettere il parametro passphrase (la password verrà chiesta da wpasupplicant). Un percorso standard per il file di configurazione è /etc/wpa_supplicant.conf (anche se personalmente preferisco salvarlo nella home).Si otterrà un file di configurazione contenente qualcosa del tipo:

```
network={
        ssid="essid"
        #psk="passphrase"
        psk=c011444e5467ce03a421db0b90e8a1166e6f5431aa9546ff0b8888ca45ea6c25
        }
```

A questo punto lanciare il demone wpasupplicant con il comando:

```
# wpa_supplicant -D[driver] -i[device] -c[/path/to/config]
```

(utilizzare l'opzione -B per avviare il demone in background)

Configurare l'interfaccia come già visto.

Per impostare il tutto automaticamente è necessario editare il file /etc/network/interfaces .

Es:

```
auto eth1
iface eth1 inet dhcp
up wpa_supplicant -ieth1 -c<file_name.conf> -Bw
down killall wpa_supplicant
```

## Amministrazione

### I compiti dell'amministratore

*Compiti base per l'amministratore di sistema (IT)*



Amministrare un sistema significa gestirne gli gli aspetti fondamentali ad un funzionamento sicuro, efficiente e continuativo.

I compiti dell'amministratore sono fra I più vari e vanno da operazioni routinarie di “normale amministrazione” come la creazione di un utente, o la modifica dei permessi di n file, a compiti meno frequenti come verifiche di integrità sui filesystem, fino a operazioni rare, come l'upgrade della distribuzione.

Ma chi è l'amministratore?

Concettualmente l'amministratore è una figura distinta dall'utente abituale del sistema, il quale ...


Uno dei compiti dell'amministratore di un sistema è quello della gestione di utenti e gruppi.

La gestione consiste sostanzialmente nella creazione o rimozione di un utente o gruppo, nell'aggiunta o rimozione da un gruppo, nel cambio della password e nella modifica dei permessi associati all'utente.

**Utenti**

| | |
|---|---|
| **useradd** [opzioni] nomeutente | aggiunge il nuovo utente nomeutente |
| **userdel** [opzioni] nomeutente | rimuove l'utente nomeutente (la home viene mantenuta) |
| **usermod** [opzioni] nomeutente | modifica le impostazioni per l'utente nomeutente |
| **passwd** [opzioni] nomeutente | modifica la password dell'utente nomeutente |

/etc/passwd File che contiene le informazioni dell'utente, uno per riga. Formato: username : password : UserID : GroupID : commento : homedirectory : comando di login

NB: in Debian il campo passwd di ciascun utente è costituito da una x. Questo perché Debian, come altri SO Linux-based, utilizza il meccanismo di shadow delle password, un sistema che sfrutta un database criptato (contenuto nel file /etc/shadow) per la memorizzazione delle password utenti.

**chsh** [opzioni] nomeutente cambia il tipo si shell di default al login dell'utente nomeutente

/etch/shells File contenente una lista delle shell disponibili

**Gruppi**

| | |
|---|---|
| **groups** [opzioni] nomeutente | elenca i gruppi a cui appartiene l'utente nomeutente |
| **groupadd** [opzioni] nomegruppo | crea il nuovo gruppo nomegruppo |
| **groupdel** [opzioni] nomegruppo | rimuove il gruppo nomegruppo |

/etc/group File contenente le informazioni sui gruppi e le loro relazioni. Formato: NomeGruppo : commento : GroupID: Utenti,Del,Gruppo

La creazione di un nuovo utente corrisponde ai seguenti passi:
- Editare /etc/passwd aggiungendo una riga per il nuovo utente;
- Editare /etc/group aggiungendo un nuovo gruppo per il nuovo utente (non indispensabile);
- Se esiste il file /etc/shadow editarlo aggiungendo una nuova riga per l'utente;
- Creare la home del nuovo utente: mkdir /home/nomeutente;
- Ricreare l'ambiente base nella nuova home: cp /etc/skel /home/nomeutente;
- Modificare il proprietario della home: chown -R nomeutente:nomegruppo /home/nomeutente;
- Modificare i permessi della home: chmod -700 /home/nomeutente;
- Modificare la password dell'utente: passwd nomeutente

# Amministrazione: processi



processi



I **processi** sono il meccanismo base su cui si basa il multitasking: a ciascun programma in esecuzione sul SO vengono associati uno o più processi ai quali il sistema assegna, tramite un meccanismo detto di scheduling, una porzione di tempo di esecuzione. La rapida schedulazione dei vari processi appare di fatto all'utente come un'esecuzione simultanea dei programmi.

Dal punto di vista del sistema un processo è composto dal codice eseguibile, dalla posizione di esecuzione, un'area di memoria con i dati in essa allocati allocati, una serie di file utilizzati, l'ambiente di esecuzione, le credenziali.

Ciascun processo è identificato univocamente sul sistema da un numero, detto PID (Process IDentifier), assegnato al momento della sua creazione.

Vita di un processo
Nei sistemi Unix-like i processi possono essere generati solo da altri processi tramite un meccanismo detto fork. Di conseguenza ciascun processo ha un processo che lo ha generato e che viene detto processo padre.
Fa eccezione il processo init, generato all'avvio del sistema, che costituisce "il padre di tutti i processi".

Come qualsiasi cosa un processo nasce, vive e muore. Per tenere traccia della condizione di un processo si utilizza una variabile di stato che può assumere tre valori:
R - running, il processo è in esecuzione;
S - sleeping, il processo è in attesa (input dell'utente, conclusione di altri processi ecc..)
Z - zombie, il processo è morto ed aspetta che il parent chieda un codice d'uscita.

Meccanismo di schedulazione
Per schedulare i processi assegnando a ciascuno di essi l'utilizzo della macchina si applicano diversi algoritmi, ad ogni modo per gestire in pratica la schedulazione a ciascun processo è associata una variabile di priorità che può assumere un valore fra -20 e 19 e che determina quanto tempo macchina verrà assegnato a quel dato processo al momento in cui verrà servito.
La priorità è gestita dal SO in base alla natura del processo.

# Amministrazione: processi

gestione processi



Monitoraggio:

**ps** [opzioni] visualizza un elenco dei processi in esecuzione con alcune caratteristiche
**pstree** visualizza i processi in esecuzione con una struttura ad albero, evidenziando parent e child
**top** [opzioni] visualizza un elenco dei processi in esecuzione con le risorse ad essi associate. Tramite le opzioni si può scegliere il criterio con cui viene ordinato l'elenco:
N - Ordina secondo PID
A - Ordina secondo età (per prima i più recenti)
P - Ordina secondo utilizzo CPU
M - Ordina secondo memeoria utilizzata
**vmstat** [opzioni]genera un rapporto sullo stato della memoria virtuale associata ai processi.
Tra le varie opzioni può essere impostato un intervallo per la generazione periodica del rapporto:
vmstat -n intervallo_secondi

Modifica dello stato:

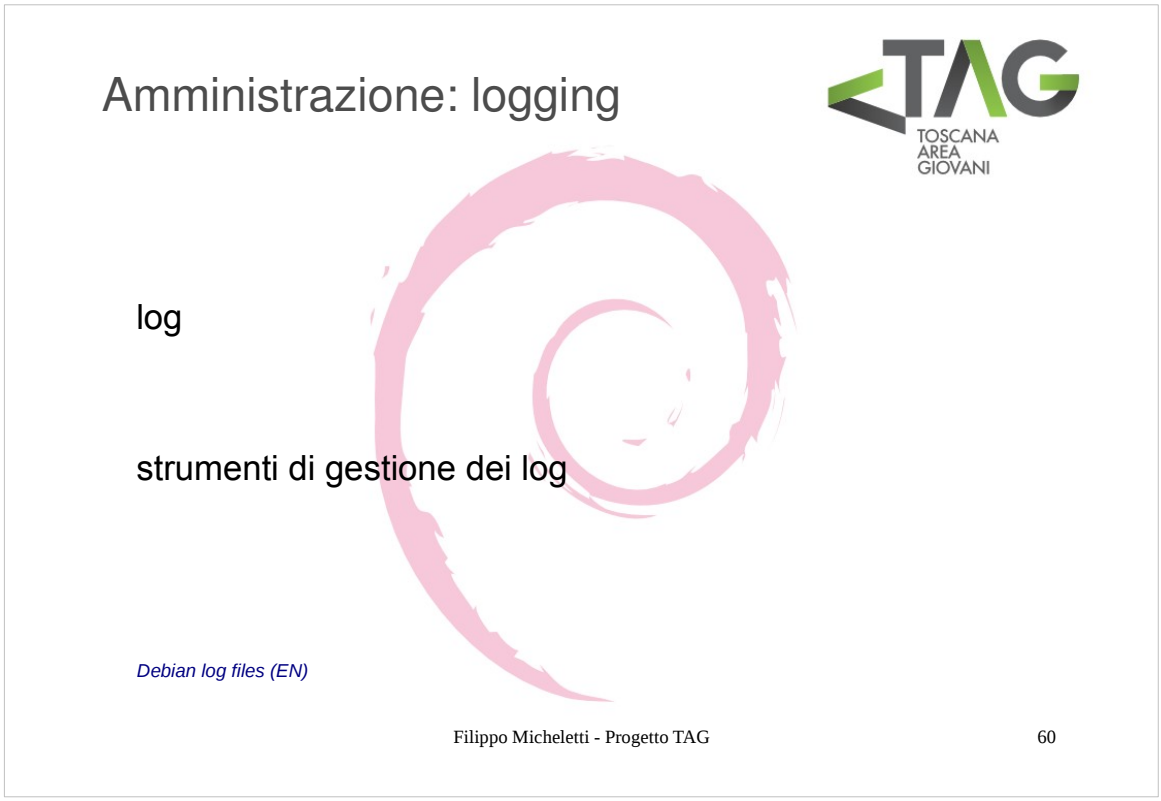
**kill** -[signal] process_pid invia un segnale al processo con pid process_pid. Se non vengono specificati argomenti il segnale di default è SIGKILL, per terminare il processo. Una lista completa dei segnali che possono essere inviati al processo si può ottenere tramite kill -l

**killall** [opzioni] nome invia un segnale a tutti i processi che contengono nome nel proprio identifier. Il segnale di default è sempre SIGKILL.

**CTRL + z**
**bg** <process number>
**fg** <process number>
**Jobs** …
La combinazione di tasti CTRL+z interrompe momentaneamente il processo in esecuzione (dalla shell corrent). Per ripristinarlo si devono usare i comandi bg e fg. Il primo fa eseguire il comando in background ed il secondo fa eseguire il comando in foreground. La shell assegna un ID ad ogni job corrente (processo aperto dall'utente) e prevede diversi comandi (fg, bg, jobs, stop...) per gestirli.

**nohup** <command>
Si può utilizzare prima di qualsiasi per evitare che il processo determinato dal comando non cada al segnale SIGHUP. E' utile quando si lavora su macchine remote e si devono eseguire comandi che possono durare a lungo. Evita l'interruzione del processo quando per problemi di rete o normale operatività ci si sconnette dalla sessione remota (via telnet o ssh). Lo standard error e tutti gli output che verrebbero stampati a schermo vengono scritti sul file nohup.out

**nice** <priority> <command> assegna la priorità specificata al comando command da eseguire




Nei sistemi Linux-based riveste una grande importanza l'attività di **logging**.
Il logging è svolto da praticamente ogni processo in esecuzione sul sistema e dal sistema stesso, ed è così importante perché permette di diagnosticare problemi (e quindi risolverli).
Se vogliamo il meccanismo di log è uno dei simboli di apertura del software: sapere in ogni momento cosa sta succedendo sul proprio sistema è l'unico modo per averne il pieno possesso e gestirlo a proprio piacimento.

Log di sistema
I log di sistema riguardano le operazioni svolte sul sistema (login utenti, cambio di password, ecc) o da strumenti direttamente correlati al suo funzionamento (per esempio apt). I log di sitema si trovano nella cartella /var/log

I log sono gestiti dal demone **rsyslogd**, un'evoluzione di syslog che si occupa di gestire altri demoni di log e l'organizzazione dei log. Le impostazioni di rsyslogd si trovano nel file /etc/rsyslogd.conf

La gestione e l'analisi dei log è un'attività sistemistica che richiede le sue attenzioni.
Su macchine ad alto traffico, o sotto attacco DOS o con particolari problemi ricorrenti, le dimensioni dei log possono crescere a dismisura in pochissimo tempo, fino a saturare il file system. Per questo motivo è sempre consigliabile montare la directory /var in una partizione indipendente, che, anche se riempita, non blocca il funzionamento del sistema.
E' inoltre importante poterli gestire, ruotandoli ad intervalli fissi e compattando i log vecchi.
Questi compiti sono svolti dall'applicazione **logrotate**.
Il file di configurazione è /etc/logrotate.conf inoltre nella cartella /etc/logrotate.d sono contenuti vari altri file d'impostazione per la gestione dei singoli log da parte di logrotate.
Logrotate viene eseguito periodicamente da script inseriti nel crontab.

Esistono diversi strumenti per analizzare i log e verificare se contengono informazioni critiche o se sono stati in qualche modo modificati (traccia di una potenziale intrusione esterna).
**Logwatch** è un sistema di monitoraggio dei log customizzabile ed estendibile che permette l'analisi dei log di sistema e la notifica via email all'amministratore. Il file di configurazione è /etc/log.d/conf/logwatch.conf. Logwatch può essere installato dai repository.
Nella directory /etc/log.d/conf/services/ ci sono le configurazioni per i diversi servizi i cui log possono essere processati da logwatch, in /etc/log.d/conf/logfiles/ ci sono le configurazioni sui log relativi ai servizi, in /etc/log/scripts/ ci sono i filtri predefiniti per analizzare diversi servizi e logfiles.

Alcuni log files:

/var/log/message: General message and system related stuff
/var/log/auth.log: Authenication logs
/var/log/kern.log: Kernel logs
/var/log/cron.log: Crond logs (cron job)
/var/log/boot.log : System boot log
/var/log/secure: Authentication log
/var/log/utmp or /var/log/wtmp : Login records file

## Amministrazione: esecuzione programmata

chron

```
# m h dom mon dow user       command
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
#
```

anacron

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# These replace cron's entries
#period  delay  job-identifier  command
1       5      cron.daily  nice run-parts --report /etc/cron.daily
7       10     cron.weekly       nice run-parts --report /etc/cron.weekly
```

*Crontab quick reference (EN)*



L'esecuzione (schedulazione) programmata di processi su Linux è possibile tramite diversi tool; noi ci occuperemo di **cron**, un programma in grado di eseguire compiti configurabili dall'utente ad intervalli regolari.

Il demone crond viene eseguito all'avvio del sistema ( /etc/rc.d/init.d/crond ) e, ogni minuto, analizza il suo file di configurazione predefinito /etc/crontab per verificare se deve eseguire dei comandi. Il file /etc/crontab ha una sintassi come quella riportata in slide. La prima riga (che è un commento) riporta la sintassi delle righe del file crontab, il significato dei campi è il seguente:

| | |
|---|---|
| **m** | minuto (minut), di valore compreso in [0, 59] |
| **h** | ora (hour), di valore compreso in [0, 23] |
| **dom** | giorno del mese (day of month), di valore compreso in [1, 31] |
| **mon** | mese (month), di valore compreso in [1, 12] |
| **dow** | giorno della settimana (day of week), di valore compreso in [0, 6] |
| **user** | utente con cui eseguire il comando; cron viene eseguito come root e root è l'utente di default (se viene omesso il campo), è possibile eseguire comandi come qualsiasi utente del sistema |
| **command** | il comando da eseguire (con gli argomenti); il comando run-parts esegue tutti i comandi presenti nella directory specificata come argomento. |

Ciascuno dei primi 5 campi può essere sostituito da un asterisco ***** che, in accordo con le regexp, significa 'qualsiasi'.
Inoltre per ciascuno di questi campi è possibile inserire un elenco separato da virgole o un'intervallo separato da un meno **-**.

Le cartelle /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly e /etc/cron.monthly contengono una serie di link simbolici agli script da eseguire, in maniera analoga al meccanismo usato da init per avviare e terminare processi al cambio di runlevel.

Il file /etc/crontab è quello di sistema ed è preferibile non modificarlo se non in per aggiungere operazioni d'interesse generale, ogni utente può editare un proprio crontab file, facendo uso del comando **crontab**. Alcuni comandi:

| | |
|---|---|
| crontab -e | Edita il proprio file crontab (con l'editor predefinito) |
| crontab -l | Visualizza il proprio file crontab |
| crontab -r | Rimuove il proprio file crontab |

**Anacron** è un'evoluzione (o meglio una versione modificata) di cron. In pratica ricalca le funzionalità di cron, con l'aggiunta che è in grado di schedulare processi che non è stato possibile schedulare al momento previsto (tipicamente perché la macchina era spenta). Il file di configurazione di anacron si trova in /etc/anacrontab, la sintassi è quella riportata in slide e I campi significano:

| | |
|---|---|
| **period** | giorni |
| **delay** | minuti |
| **job-identifier** | idntifica il lavoro svolto nei messaggi di anacron |
| **comand** | commando da eseguire |

In pratica cron avvia periodicamente (tramite il meccanismo di schedulazione visto) anacron, che si occupa di gestire i processi programmati e non schedulati da cron.


Una volta caricato il kernel viene eseguito il processo **init** (/sbin/init), il processo radice del sistema, ossia il processo padre di tutti i processi che verranno avviati sul sistema. Init ha il compito di inizializzare il sistema avviando i servizi base.

Sulle distribuzioni Linux che seguono le specifiche **Unix System V** (fra cui Debian) il funzionamento del sistema è organizzato in **runlevels**: ogni runlevel rappresenta un stato di funzionamento del sistema.

A ciascun runlevel sono associati dei **servizi** da avviare e/o terminare al momento in cui la modalità operativa del sistema si sposta su quel determinato runlevel. Init si occupa dunque di avviare o terminare i servizi associati al runlevel in cui viene portato il sistema.

Di default i runlevel di un sistema sono 7 più due speciali:

| | |
|---|---|
| Runlevel **0**: | avvia la sequenza di arresto del sistema (shutdown) |
| Runlevel **1**: | modalità singolo utente (con servizio di rete disabilitato) |
| Runlevel **2**: | modalità multiutente (con servizio rete attivo ma file sharing disabilitato) |
| Runlevel **3**: | modalità testuale (tutti i servizi attivi) |
| Runlevel **4**: | inutilizzato. Può essere dedicato ad usi personali |
| Runlevel **5**: | modalità grafica multiutente (tutti i servizi attivi) |
| Runlevel **6**: | avvia la sequenza di reboot |
| Runlevel **N**: | none |
| Runlevel **S**: | single user |

Inizialmente il sistema viene portato nel runlevel N (None) dove avviene l'inizializzazione tramite le impostazioni del file /etc/inittab; successivamente si passa dal runlevel S (single-user), dove avvengono le impostazioni secondarie (hardware ecc) e infine si passa al runlevel impostato come default nel file /etc/inittab sotto la voce "initdefault" (normalmente il 5 o il 2).

A ciascun runlevel è associata una directory (/etc/rc#.d per il runlevel #) la quale contiene dei link simbolici a degli script contenuti nella cartella /etc/init.d, ciascuno dei quali è rinominato per iniziare con la lettera K o S, seguita da un numero a due cifre e dal nome originale dello script (per esempio in link ad acpid in /etc/init.d è S19acpid in /etc/rc3.d).

La modalità con cui avviene l'inizializzazione del sistema è semplice: all'ingresso in un determinato runlevel il sistema accede alla directory ad esso associata ed esegue gli script in essa contenuti secondo due regole:
gli script che iniziano con K sono eseguiti nell'ordine indicato dal numero presente nel nome dello script o in ordine alfabetico per il nome dello script a parità di numero, con l'argomento start (e quindi i corrispondenti servizi avviati)
gli script che iniziano con S sono eseguiti nell'ordine indicato dal numero presente nel nome dello script o in ordine alfabetico per il nome dello script a parità di numero, con l'argomento stop (e quindi i corrispondenti servizi terminati)

I parametri di default per ciascuno script contenuto nella cartella /etc/init.d si trovano nella cartella /etc/default in un file avente lo stesso nome dello script a cui si riferiscono.

## Customizzazione: startup/shutdown

Personalizzare startup e shutdown

*Making scripts run at boot time (EN)*



In base a quanto visto personalizzare i servizi avviati all'avvio o allo spegnimento del sistema si esegue in 2 passi:

- **aggiungere lo script** da eseguire nella directory /etc/init.d
- **creare un link** (simbolico) allo script nella directory corrispondente al runlevel in cui si vuole che sia eseguito/terminato, modificando opportunamente il nome del link

NB: è possibile utiizzare l'utility **update-rc.d** per aggiungere/rimuovere in automatico gli script da più runlevel, con la possibilità di utilizzare opzioni di default. Creando link manualmente si ottiene comunque un controllo più completo sul risultato evitando complicazioni inutili.

## Customizzazione: moduli del kernel

module-assistant

comandi di gestione

*Kernel modules (EN)*



Un **modulo** del kernel è fondamentalmente un file oggetto, cioè una parte di codice scritto per interagire con variabili e funzioni del kernel, e compilato per una data architettura.

Un modulo assolve a una funzione specifica e può essere visto come una sorta di “driver” per gestire un componente hardware o software del sistema.

Saper modificare I moduli è fondamentale sia per poter integrare funzionalità specifiche legate alla macchina sia per eliminare componenti inutili del sistema.

Per questo scopo si possono utilizzare tool specifici come **module-assistant** che offrono procedure guidate, o agire direttamente tramite I seguenti comandi:

| | |
|---|---|
| **lsmod** | elenca i moduli del kernel caricati (elenca i file contenuti in /proc/modules) |
| **modinfo** <nome_modulo> | restituisce informazioni sul modulo nome_modulo |
| **modprobe** <nome_modulo> | aggiunge il modulo <nome_modulo> |
| **rmmod** <nome_modulo> | rimuove il modulo <nome_modulo> (equivalente a modprobe -r <nome_modulo>) |

I principali file di configurazione dei moduli del kernel sono I seguenti:

| | |
|---|---|
| /etc/modules.conf | contiene una lista di moduli da caricare automaticamente al boot |
| /etc/modprobe.d/ | contiene una serie di file di configurazione riguardanti i moduli |
| /etc/modprobe.d/blacklist.conf | elenco di moduli da NON caricare automaticamente |

## Customizzazione: cambio DE



LXDE

Fluxbox

*LXDE (EN)*
*how to install LXDE (EN)*
*how to install FluxBox (EN)*
*Configurare FluxBox (IT)*



Come abbiamo visto il DE standard di Debian è GNOME.

GNOME è un ottimo DE, sicuramente uno dei migliori in quanto a giusto compromesso fra completezza e semplicità, tuttavia si può avere la voglia (o la necessità) di qualcosa di più leggero e performante oppure di più accattivante.

Personalmente prefrisco la prima scelta, per cui vediamo come installare un DE molto leggero: **LXDE** (Lightweight X11 Desktop Enviroment)

Da root installate il pacchetto **lxde** dai repo: questo installerà LXDE con le applicazioni più diffuse, altrimenti, se volete il solo DE, installate il pacchetto **lxde-core**.

Analogamente è possibile installare qualsiasi altro DE, per esempio **FluxBox** (sul quale è basato fra l'altro LXDE).

Non tutti I DE (ne esistono veramente tanti!) sono ovviamente presenti nei repository ufficiali di Debian per cui per alcuni di questi sarà necessaria un'installazione manuale.