

# Lezione 13

# Gestione software in

# Debian GNU/Linux

Sistemi Operativi (9 CFU), CdL Informatica, A. A. 2023/2024  
Dipartimento di Scienze Fisiche, Informatiche e Matematiche  
Università di Modena e Reggio Emilia

# Quote of the day

(Meditate, gente, meditate...)

**“Fellow Linuxers, this is just to announce the imminent completion of a brand-new Linux release, which I'm calling the Debian Linux Release...”**

**Monday, 16 August 1993**

*Ian Murdock (1973-2015)*

*Ingegnere del software*

*Creatore di Debian GNU/Linux*

*Sviluppatore di OpenSolaris*



# INTRODUZIONE

# Lo scenario

(Nuovo utente Debian, inesperto, davanti al suo PC/laptop)

**Nuovo utente Debian.** Lo studente con una preparazione di base sui comandi UNIX vuole imparare ad usare gli strumenti di gestione del software messi a disposizione da Debian GNU/Linux.



# Lo scenario

(Nel caso in cui ve lo steste chiedendo...)

(Per la cronaca: alla vostra destra potete ammirare un astronauta sulla ISS).

(Sempre per la cronaca: è un utente Debian).

(Ed ancora per la cronaca: per la NASA, Debian è il SO più stabile in assoluto).

Perché la NASA ha effettuato la migrazione a Debian?  
Leggete qui (e fatevi quattro risate):

<http://www.dailytech.com/International+Space+Station+Goes+Open+Source+Dumps+Windows+XP+for+Debian/article31527.htm>

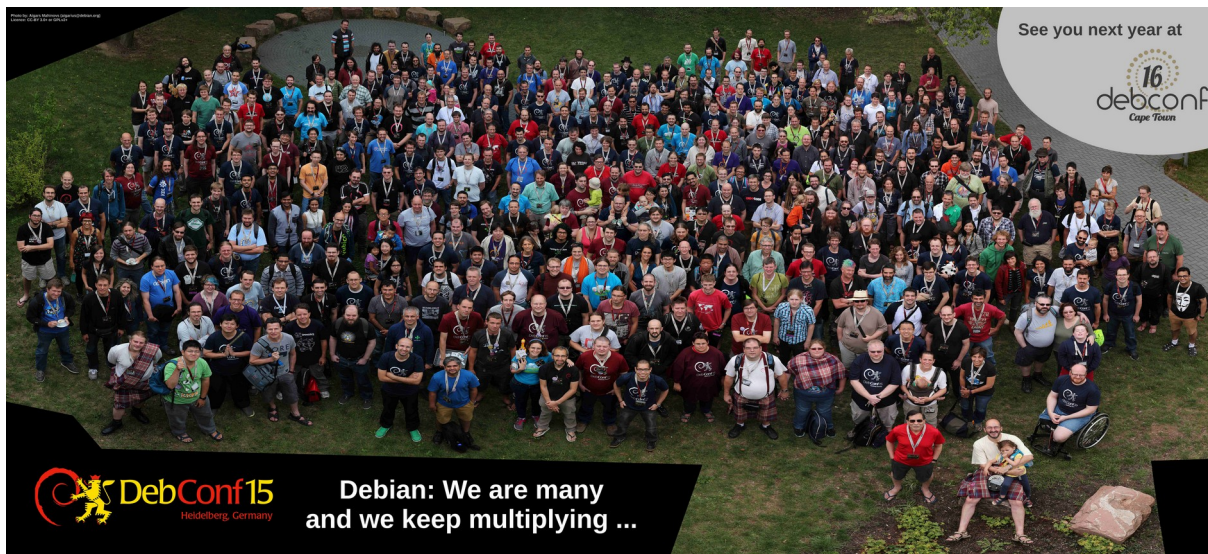




# Interrogativi 1/2

(Come è strutturato l'ecosistema Debian?)

Alcuni aspetti del SO Debian sono incomprensibili se non si conosce il contesto culturale della distribuzione.  
Come è organizzata Debian dal punto di vista sociale e politico?



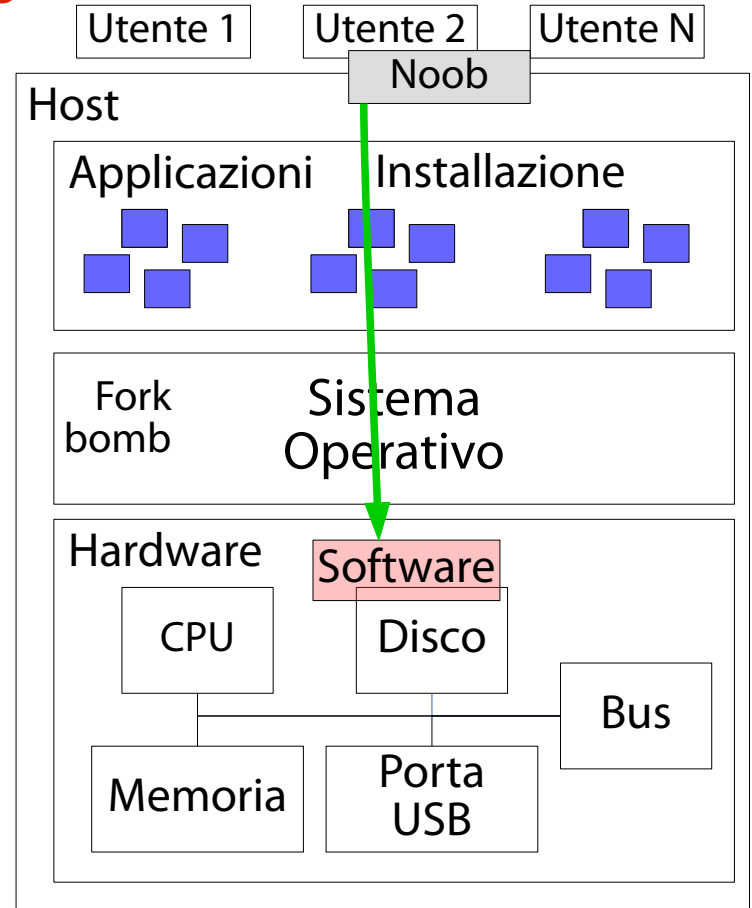
# Interrogativi 2/2

(Quali strumenti esistono per la gestione del software?)

L'utente alle prime armi vuole gestire il software in maniera corretta. Per "gestire" si intende:

- installare;
- rimuovere;
- aggiornare;
- configurare.

Quali strumenti mette Debian a disposizione dell'utente per gestire il software?



# INTRODUZIONE



# Cosa è Debian GNU/Linux

(È la prima grande distribuzione GNU/Linux)

Debian è la **prima** grande **distribuzione GNU/Linux** sviluppata nello spirito “GNU”.

È una distribuzione non commerciale, realizzata da volontari (sviluppatori, amministratori, ricercatori, ...).

Distribuzione “amatoriale” nel miglior senso del termine  
→ Sviluppo fatto per passione.

Debian è un progetto ambizioso, che si pone l'**obiettivo** di **creare il migliore SO “free”** in circolazione.

Debian è una grande **comunità** di sviluppatori.

Home page progetto: <https://www.debian.org>

# Alcuni numeri

(Danno un'idea della complessità del progetto)

Architetture hardware supportate:

10 (ufficiali)

22 (non ufficiali)

<https://www.debian.org/ports/>

Pacchetti software disponibili:

60388 (Debian 10).

```
apt search "^.*$" | grep / | wc -l
```

Righe di codice complessive:

1071348495 (Debian 10).

<https://sources.debian.org/stats/>

Sviluppatori:

1115 (<https://db.debian.org/>)

# Trivia

(Grande capo estiq... è interessato di conoscere questo)

Debian è l'unione di

**DEB**ra Lynn (moglie di Ian Murdock).

**IAN** Murdock.

Ogni versione di Debian prende il nome da uno dei personaggi del cartone animato "Toy Story".

Buzz, Rex, Bo, Hamn, Slink, Potato, Woody, Sarge, Etch, Lenny, Squeeze, Jessie, Stretch, Buster.



Buzz, il protagonista di "Toy Story" 11

# Le tre caratteristiche di Debian

(Qualità, Libertà, Indipendenza)

In Debian vige la **cultura dell'eccellenza tecnica** (parecchi sviluppatori sono esperti a livello mondiale del pacchetto software che gestiscono).

**“We release when it's ready”** (il rilascio di software non è condizionato dal fattore tempo).

# Le tre caratteristiche di Debian

(Qualità, **Libertà**, Indipendenza)

Sviluppatori ed utenti godono delle libertà descritte nel **contratto sociale (Debian Social Contract)**.

È un contratto vero e proprio che definisce gli impegni di Debian nei confronti della comunità Free Software.

# Le tre caratteristiche di Debian

(Qualità, Libertà, **Indipendenza**)

Lo sviluppo non è influenzato da alcun ente o ditta. Le principali fonti di introito sono donazioni e gift-economy.

Il marchio “Debian” è in mano alla organizzazione non-profit **Software in the Public Interest (SPI)**.

Il processo decisionale è aperto a tutti i membri della comunità.

Il processo decisionale è democratico (**democracy**).

Il processo decisionale è meritocratico (**do-ocracy**).

# Debian Free Software Guidelines

(Redatte da Bruce Perens nel 1997)

Le **Debian Free Software Guidelines (DFSG)** sono linee guida per stabilire se una licenza software sia “free” o meno.

1. Ridistribuzione libera.
2. Inclusione del codice sorgente.
3. Modifica e creazione di software derivato.
4. Integrità della versione originale del software.
5. Nessuna discriminazione verso persone o gruppi.
6. Nessuna discriminazione verso le imprese.
7. La licenza si applica a tutti i destinatari del software.
8. La licenza non deve essere esclusiva a Debian.
9. La licenza non deve contaminare altro software.
10. GPL, BSD, Artistic → free.



# Debian Social Contract

(Redatto da Bruce Perens nel 1997)

**Il contratto sociale Debian (Debian Social Contract)** è un vero e proprio contratto stipulato fra Debian e i suoi utenti.

1. Debian è e resterà free al 100% (aderenza alle DFSG).
2. Debian renderà alla comunità Free Software (rilascio di software con licenza free, comunicazione con i vari upstream, creazione del miglior SO possibile, ...).
3. Debian non nasconderà mai i problemi (è possibile leggere apertamente l'elenco dei bug).
4. Le priorità di Debian sono gli utenti ed il free software.
5. Le opere che non rispettano gli standard free software sono incluse a parte.

# Debian Constitution

(Debian è una democrazia in miniatura)

La costituzione Debian (Debian Constitution) descrive la struttura organizzativa (organigramma).

→ Identifica i ruoli all'interno della comunità (presidente, sviluppatori, amministratori, ...) ed i relativi diritti/doveri.

→ Specifica i processi decisionali (votazioni).

Organigramma Debian:

<http://commons.wikimedia.org/wiki/File:Debian-organigram.svg>

# Canali di comunicazione

(No, non le reti sociali, per favore...)

La comunicazione “pubblica” fra sviluppatori e utenti avviene attraverso i seguenti strumenti:

- E-mail (mailing list).

- Chat (canali IRC).

La comunicazione “privata” avviene spesso in forma cifrata (GnuPG).

La comunicazioni interna con l'infrastruttura software avviene in forma cifrata (GnuPG).

**GnuPG: GNU Privacy Guard**, pacchetto software contenente funzionalità crittografiche.

# Le diverse versioni di Debian

(Diverse versioni? Ma non era uno il SO?)

Il progetto Debian è presente in diverse versioni. Ciascuna versione corrisponde ad un insieme di pacchetti software accuratamente scelti, con i seguenti obiettivi.

- Avere un sistema il più “stabile” possibile, a scapito della modernità del software.

- Avere un sistema più “moderno possibile”, a scapito della stabilità del software.

- Avere una “via di mezzo” fra stabilità e modernità.

L'utente si sceglie la versione che vuole usare.

# Stable, testing, unstable

(Eh, no! Sono tre i SO!)

**Distribuzione stable.** È la versione ufficiale, contenente software datato ma stabile. Per “stabile” si intende “testato in tutte le condizioni operative possibili”.

**Distribuzione unstable.** È la versione di sviluppo. Contiene software recente che non causa danni evidenti al sistema (si sperimenta l'interazione fra nuovi sw). Per “unstable” si intende “sembra funzionare, ma non è testato in tutte le condizioni operative possibili”.

**Distribuzione testing.** È una via di mezzo fra versione stable ed unstable.

# Repository

(Archivi Web in cui sono contenuti metadati e pacchetti)

Le versioni di Debian sono mantenute in depositi software, detti **repository**.

Un repository è un server Web ospitante documenti.

Ad esempio: <http://ftp.debian.org/debian>

Il sito è soggetto ad un carico bestiale; pertanto viene copiato su siti nazionali detti **mirror**. Il mirror italiano è:

[http://ftp."countrycode".debian.org/debian](http://ftp.)

<http://ftp.it.debian.org/debian>

L'intera architettura Debian (repository, formato pacchetti binari e sorgente) è spiegata nel **Debian**

**Policy Manual**: <https://www.debian.org/doc/debian-policy/>

# Struttura del repository 1/2

(Ad altissimo livello)

Un repository contiene diverse directory.

**dists**: contiene i file di controllo usati dai meccanismi di pacchettizzazione (elenco compresso dei pacchetti disponibili).

**doc**: contiene documentazione di base Debian.

**indices**: è un indice di tutti i file contenuti nei pacchetti (usato dal software **apt-file** per individuare il pacchetto contenente un dato file).

**non-US**: software la cui esportazione dagli USA è vietata per legge (gli algoritmi crittografici usati da SSH e GPG).

Directory obsoleta da una decina d'anni, oramai.



# Struttura del repository 2/2

(Ad altissimo livello)

Un repository contiene diverse directory.

**pool**: l'archivio vero e proprio dei pacchetti, suddiviso per lettera iniziale del pacchetto. Contiene le versioni attuali dei pacchetti nella distribuzione unstable.

**Project**: spazio per gli sviluppatori (contiene la directory **experimental**, in cui gli sviluppatori possono caricare le ultimissime versioni dei loro pacchetti per scopi di test).

**tools**: strumenti (obsoleti) per la creazione di floppy di avvio da DOS.

# Suddivisione di un repository

(main, non-free, contrib)

All'interno della directory **pool**, i pacchetti software sono divisi in altrettante sottodirectory, per componenti.

**main**: l'elenco principale dei pacchetti DFSG compliant.

**non-free**: software considerato non-free secondo le DFSG.

**contrib**: software free che dipende da altro software non-free.

# Il caricamento di un pacchetto

(L'inizio dell'avventura)

Periodicamente uno sviluppatore responsabile di un pacchetto “carica” con il protocollo Secure FTP (SFTP, FTP criptato) una nuova versione nella directory **incoming** dell'archivio.

- Directory non visibile agli utenti normali.

- Coda di upload dei pacchetti.

- Controllo batch dei pacchetti in incoming.

Controlli effettuati:

- il pacchetto è firmato digitalmente?

- il pacchetto è malformato?

- il pacchetto rispetta le DFSG?

# L'ingresso in unstable

(Se il pacchetto sopravvive ai controlli in incoming)

Se il pacchetto non ha difetti evidenti, viene copiato automaticamente nella directory `pool` dell'archivio, che è pubblicamente visibile.

Nel giro di un giorno, il contenuto della directory viene replicato (mirrored) in tutto il mondo.

**I pacchetti copiati in pool fanno inizialmente parte della distribuzione unstable.**

# La distribuzione testing

(Un ulteriore, rigidissimo, controllo)

Il codice in unstable è il più recente e il più fragile possibile.

Un pacchetto, per poter far parte della versione ufficiale di Debian, deve “passare” per la versione intermedia **testing**.

Il passaggio da unstable a testing non è manuale, ed è dettato da un insieme di regole piuttosto rigide.

# Passaggio da unstable a testing 1/2

(Un bagno di sangue)

1. Il pacchetto deve essere rimasto in unstable per un dato periodo di tempo. La durata esatta dipende dall'"urgenza" dell'aggiornamento (funzionalità, sicurezza).
2. Il pacchetto non deve avere un numero di bug critici per il rilascio della distribuzione (release-critical bugs) maggiore di quello della versione attualmente in testing.

**Release-critical bug:** un baco che impedisce, di fatto, il rilascio della prossima versione di Debian (non DFSG compliant, malfunzionante, non si integra bene, ...).

3. Il pacchetto deve essere compilabile su tutte le architetture da esso supportate.

# Passaggio da unstable a testing 2/2

(Vuole anche una fetta di ... vicino all'osso?)

4. Tutte le dipendenze del pacchetto devono essere soddisfatte da pacchetti già presenti in testing (!), oppure dal gruppo di pacchetti che sta per entrare in testing (insieme al pacchetto in questione).
5. L'installazione del pacchetto in testing non deve corrompere in alcun modo l'installazione di altri pacchetti già presenti in testing.



# Il periodo di freeze

(Si chiude il rubinetto unstable → testing)

Nel periodo immediatamente precedente il rilascio di una nuova distribuzione ufficiale stable, la distribuzione testing entra in uno stato noto con il nome di **freeze**.

Sono permessi solo ed esclusivamente aggiornamenti di tipo correttivo, non di tipo aggiuntivo.

Quando le DFSG, gli obiettivi, le strategie di rilascio sono rispettati in pieno:

- il contenuto di testing viene riversato nella nuova distribuzione ufficiale, stable.

- testing ottiene il nuovo nome e “scongelata”.

L'intero ciclo di vita del pacchetto è riassunto qui:

[http://commons.wikimedia.org/wiki/File:Debian\\_package\\_cycle.svg](http://commons.wikimedia.org/wiki/File:Debian_package_cycle.svg)

# Il rilascio della distribuzione stable

(Finalmente!)

Periodicamente il **Release Manager** (capo del team di rilascio, **Release Team**), in accordo con gli sviluppatori dei pacchetti principali (kernel, libreria del C, desktop, ...), pubblica le linee guida per il rilascio della nuova distribuzione stable.

Si correggono obiettivi e strategie della nuova distribuzione.

Si decide un eventuale rilascio di distribuzione (quando il numero di release critical bug si avvicina a zero).

Si decide il nome della prossima distribuzione stable.

# Il numero di release critical bug

(Un'immagine vale più di mille parole)

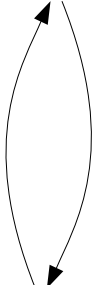
È disponibile all'indirizzo:

<https://bugs.debian.org/release-critical/>

Si può anche vedere l'andamento nell'intera storia del progetto (o, perlomeno, a partire dall'istante in cui si è iniziato a monitorare i bug critici).

Basta cliccare su “Graph with all the history”.

→ Andamento a “dente di sega”.



- Si comincia con tanti bachi.
- I bachi sono corretti.
- I bachi tendono allo zero.
- Si rilascia la nuova distribuzione e unstable popola testing.

# Pacchettizzazione del software

(Risolve il problema della distribuzione del software)

Debian pacchettizza il software in un formato detto **DEB**.

Un file **.deb** è un archivio UNIX contenente:

- i file di un progetto software installati sul file system;
- alcuni file di controllo che coordinano la procedura di installazione.

Esistono due famiglie di comandi per la gestione dei pacchetti: DPKG e APT.

# DPKG e APT

(I capisaldi)

## **DPKG (1993, Ian Jackson).**

Gestione a basso livello di singoli pacchetti (installazione, rimozione, configurazione).

Client da linea di comando.

Non sono gestite le dipendenze fra pacchetti.

## **APT (1998, Brian White).**

Estensione di DPKG (libreria C++, diversi client da linea di comando).

Gestione delle dipendenze, aggiornamento di intere distribuzioni, uso di repository non ufficiali.

Front-end di APT testuali e grafici (2000-).

# **GESTIONE DEI REPOSITORY**

# Scenario e domande

(Esistono strumenti di ausilio alla scelta di una distribuzione GNU/Linux?)

**Scenario:** l'utente ha appena installato Debian GNU/Linux e vuole svolgere le seguenti operazioni:  
configurare le sorgenti dei repository;  
sincronizzare localmente il repository.

## **Interrogativi:**

Quali file di configurazione impostano le sorgenti?

Quali comandi permettono di sincronizzare localmente il repository?



# Il file `/etc/apt/sources.list`

(Definisce i repository usabili dal SO Debian installato)

Il file `/etc/apt/sources.list` contiene la definizione dei repository usabili dal SO Debian installato.

È un elenco di righe, ciascuna definente uno specifico repository.

Il formato generico di una riga è il seguente:  
`deb[-src] <URI> <distrib.> [componenti]...`

# Primo campo

(Repository di pacchetti binari o sorgenti?)

Il primo campo è una stringa che può assumere uno dei valori seguenti.

**deb**: definizione di un repository per pacchetti binari.

**deb-src**: definizione di un repository per pacchetti sorgente.

# Secondo campo

(L'URI del repository, che lo identifica in maniera univoca)

Il secondo campo definisce la locazione del repository. Si usa un **URI (Uniform Resource Indicator)**, ossia la generalizzazione di un URL di una pagina Web.

Il formato dell'URI è il seguente:

*protocollo:sorgente*

in cui:

*protocollo* dettaglia la natura del mezzo che ospita il repository (cdrom, http);

*sorgente* specifica quale sorgente si vuole usare (un sito Web piuttosto che un altro, uno specifico CD).

# Terzo campo

(La distribuzione che si intende usare)

Il terzo campo definisce la distribuzione che si intende usare.

**stable**: la distribuzione stabile corrente.

**sid**: la distribuzione instabile corrente.

**testing**: la distribuzione di test corrente.

È possibile altresì usare il nome in codice di una specifica distribuzione: **jessie**, **wheezy**, ...

# Quarto campo

(Quali componenti abilitare fra main, contrib, non-free)

Il quarto campo è un elenco di componenti della distribuzione da abilitare.

**main:** pacchetti che rispettano le DFSG.

**non-free:** pacchetti che non rispettano le DFSG.

**contrib:** pacchetti che, pur rispettando le DFSG, dipendono a loro volta da pacchetti che non le rispettano.

# Un esempio di configurazione

(Con repository in rete)

```
# Repository ufficiali presi da un mirror italiano
deb http://ftp.it.debian.org/debian/ stable main
deb-src http://ftp.it.debian.org/debian/ stable main

# Aggiornamenti di sicurezza
deb http://security.debian.org/debian/ stable/updates
main
deb-src http://security.debian.org/debian/
stable/updates main

# Aggiornamenti non di sicurezza
deb http://ftp.it.debian.org/debian stable-updates main
deb-src http://ftp.it.debian.org/debian stable-updates
main
```

# Un esempio di configurazione

(Tipo di repository)

```
# Repository ufficiali presi da un mirror italiano  
deb http://ftp.it.debian.org/debian/ stable main  
deb-src http://ftp.it.debian.org/debian/ stable main
```

Il primo campo identifica il tipo di repository.

**deb**: repository contenente pacchetti binari (archivi compressi che contengono un file system da copiare sulla partizione di root).

**deb-src**: repository contenente pacchetti sorgente (contengono il codice sorgente originale e le modifiche apportate da Debian; permettono di ricompilarsi un pacchetto binario).

Solitamente, un repository ospita sia i pacchetti binari che quelli sorgente.

# Un esempio di configurazione

(URI della sorgente repository)

```
# Repository ufficiali presi da un mirror italiano  
deb http://ftp.it.debian.org/debian/ stable main  
deb-src http://ftp.it.debian.org/debian/ stable main
```

Il secondo campo identifica l'URI del repository.

Nel caso di repository Web, si fornisce l'URL del server Web ospitante il repository.

Un'altra fonte possibile è il CDRom. Ad essa corrisponde un URI del tipo **cdrom: [nome del CD] /**.



# Un esempio di configurazione

(Nome della distribuzione)

```
# Repository ufficiali presi da un mirror italiano  
deb http://ftp.it.debian.org/debian/ stable main  
deb-src http://ftp.it.debian.org/debian/ stable main
```

Il terzo campo identifica il nome in codice della distribuzione. Si può usare uno dei nomi delle ultime distribuzioni stabili (ad es., “wheezy” o “jessie”).

In alternativa, si può usare uno dei sinonimi seguenti:

<b>stable</b>	→ punta sempre alla distribuzione stable corrente.
<b>testing</b>	→ punta sempre alla distribuzione testing.
<b>sid</b>	→ punta sempre alla distribuzione unstable.

# Un esempio di configurazione

(Componenti della distribuzione)

```
# Repository ufficiali presi da un mirror italiano
deb http://ftp.it.debian.org/debian/ stable main
deb-src http://ftp.it.debian.org/debian/ stable main
```

Il quarto campo (e successivi) elenca i nomi dei componenti software installabili dal repository. Di default Debian fornisce un SO free; pertanto l'unico repository a disposizione è **main**.

È possibile aggiungere altri due componenti:

**non-free** → software non libero.

**Contrib** → software libero che dipende da software non libero.

Occhio! Non tutti i repository hanno questi componenti!

## Esercizio 1 (2 min.)

Aprite il file `/etc/apt/sources.list` della vostro SO Debian. Configurate il file in modo tale da recuperare i repository seguenti da rete:

- repository ufficiale;

- repository degli aggiornamenti di sicurezza;

- repository degli aggiornamenti non di sicurezza.

Inoltre, abilitate le seguenti componenti:

- software libero;

- software non libero;

- software libero che dipende da software non libero.

# Sincronizzazione dei repository

(Si usa il comando **apt update**)

L'aggiornamento dei repository Debian è una operazione propedeutica a qualunque interazione col gestore dei pacchetti.

Si scarica sul SO locale l'elenco aggiornato dei pacchetti, facendo uso dei repository definiti in **/etc/apt/sources.list**.

A tal scopo si usa un comando fornito da APT:

**apt update**

# Dove è memorizzato il repository?

(Qualche informazione tecnica)

I repository scaricati sono memorizzati nella directory **/var/lib/apt/lists**. Ogni repository contiene diversi file.

Un file **Nome\_Repository\_Packages** contenente la descrizione di ogni pacchetto binario fornito.

Un file **Nome\_Repository\_Sources** analogo per la descrizione di ogni pacchetto sorgente fornito.

Un file **Nome\_Repository\_Translation** con la traduzione delle descrizioni dei software nelle lingue abilitate.

Un file **Nome\_Repository\_Release** contenente gli hash MD5 di ciascun file contenuto nel repository.

Un file **Nome\_Repository\_Release.gpg** contenente la firma digitale del repository (è verificata automaticamente).

# **INSTALLAZIONE E RIMOZIONE DEI PACCHETTI BINARI**

# Scenario e domande

(Quali comandi permettono di installare e rimuovere un pacchetto binario?)

**Scenario:** l'utente ha appena installato Debian GNU/Linux e vuole svolgere le seguenti operazioni:  
installare un pacchetto binario;  
rimuovere un pacchetto binario.

## **Interrogativi:**

Quali comandi permettono di installare un pacchetto binario?

Quali comandi permettono rimuovere un pacchetto binario?

# Firma digitale dei pacchetti

(Ovvero come risolvere quel fastidioso warning emesso da apt-get)

Può capitare che APT si lamenti di “firme non verificate”. In Debian, ogni pacchetto è firmato digitalmente dallo sviluppatore che lo ha creato.

Per verificare la firma digitale, l'utente deve scaricarsi le chiavi pubbliche degli sviluppatori coinvolti ed applicare un comando.

APT farebbe automaticamente la verifica per voi, ma ha bisogno delle chiavi pubbliche degli sviluppatori, che vanno pertanto installate.



# O vi installate le chiavi a mano...

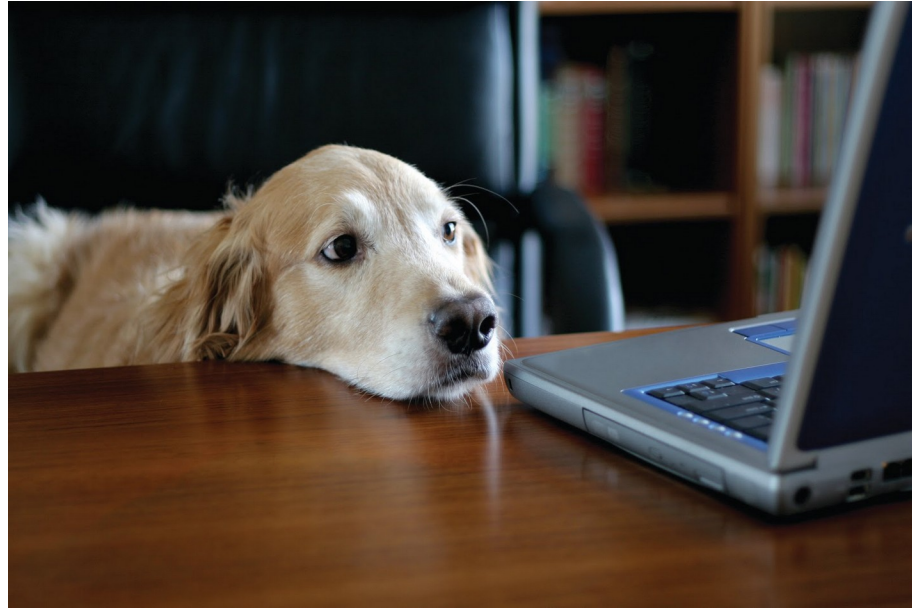
("Ma sì, dajè, famola alla c.... di c...")



# ... o si installa un pacchetto software

(Che fornisce le chiavi!)

Domanda: qual è il pacchetto software che fornisce le chiavi pubbliche degli sviluppatori Debian responsabili del repository Debian?



# Installazione di un pacchetto binario

(Si usa il comando **apt install**)

Per installare pacchetti, si utilizza il seguente comando (da amministratore).

```
apt install PACCHETTO1 PACCHETTO2 . . .
```

Operazioni effettuate dal comando:

- risoluzione e stampa delle dipendenze.

- recupero dei pacchetti da sorgenti remote.

- pre-configurazione (script).

- spacchettamento.

- configurazione.

- post-configurazione (script).

# Una osservazione doverosa

(Che si rivelerà utile in seguito)

Se **apt install** decide di installare più di un pacchetto, il comando opera in questo modo:

- recupera tutti i pacchetti;
- pre-configura tutti i pacchetti;
- spacchetta tutti i pacchetti;
- configura tutti i pacchetti;
- post-configura tutti i pacchetti.

# Installazione delle chiavi

(Seguendo fedelmente le istruzioni di debian-multimedia)

Si installi il pacchetto binario avente il nome **debian-keyring**, contenuto nel repository appena configurato.

```
apt install debian-keyring
```

Questo pacchetto software installa le chiavi pubbliche GPG degli sviluppatori Debian con uno script.

# Aggiornamento dei repository

(Questa volta il warning sparisce)

Si aggiorni nuovamente l'elenco dei repository.

```
apt update
```

Due osservazioni degne di nota.

Il warning precedente è sparito (segno che le firme digitali sono state verificate).

Gli archivi non sono scaricati, in quanto già presenti sul file system locale.

## Esercizio 2 (1 min.)

Installate il pacchetto software **hello**.

# Rimozione di un pacchetto binario

(Si usa il comando **apt remove**)

Per rimuovere pacchetti, si utilizza il seguente comando (da amministratore):

```
apt remove PACCHETTO1 PACCHETTO2 . . .
```

Operazioni effettuate dal comando:

- stampa l'elenco dei pacchetti che verranno rimossi (incluse le dipendenze non più utilizzate);

- stampa lo spazio su disco liberato dal processo di rimozione dei pacchetti;

- rimuove tutti i file contenuti in tutti i pacchetti.



## Esercizio 3 (1 min.)

Cancellate il pacchetto binario **hello** appena installato.

# Rimozione di un pacchetto binario

(Si usa il comando **apt purge**)

L'opzione **--purge** di **apt remove**:

effettua tutte le operazioni di **apt remove** (se il pacchetto non è ancora stato rimosso);

cancella tutti i file di configurazione generati dagli script eseguiti dal pacchetto binario durante l'installazione (se non sono già stati rimossi).

```
apt remove --purge PACCHETTO1 . . .
```

Come alternativa equivalente si può usare il comando

```
apt purge PACCHETTO1 PACCHETTO2 . . .
```

# Un esempio concreto

(Il pacchetto **lm\_sensors** misura la temperatura dei componenti hardware)

Il pacchetto binario **lm-sensors** installa strumenti per la misurazione della temperatura dei componenti hardware installati nella macchina.

Fornisce il comando **sensors**.

Occhio! Su macchine virtuali non è detto che funzioni.

Questo pacchetto:

- installa il file binario **/usr/bin/sensors**.

- installa il file **/etc/init.d/lm-sensors** (script shell di inizializzazione del monitor di temperatura).

- genera dinamicamente un altro file di configurazione **/etc/rcS.d/S19lm-sensors** (attiva il monitor quando il SO è nella modalità di ripristino)

# Rimozione normale

(**apt remove**)

Eseguendo **apt remove lm-sensors**:

viene rimosso il binario **/usr/bin/sensors**;

il file **/etc/init.d/lm-sensors** rimane;

il file **/etc/rcS.d/S19lm-sensors** rimane.

Ora si installi di nuovo **lm-sensors** per eseguire la purga successiva.

**apt install lm-sensors**

Si noti come il pacchetto binario non venga più scaricato dal Web. È già presente in locale ed APT lo spacchetta direttamente.

# Rimozione completa

(**apt purge**)

Eseguendo **apt purge lm-sensors**:

viene rimosso il binario **/usr/bin/sensors**;

viene rimosso il file **/etc/init.d/lm-sensors**;

viene rimosso il file **/etc/rcS.d/S19lm-sensors**.

→ La configurazione è stata interamente cancellata.

# Autorimozione

(Si usa il comando **apt autoremove**; con l'opzione **--purge** è pericoloso)

In seguito alla cancellazione di un pacchetto, APT cerca di capire se esistono pacchetti che non servono più a niente.

→ Servivano per il pacchetto cancellato e a nient'altro.

APT suggerisce di rimuoverli con il comando:

```
apt autoremove
```

Il comando seguente cancella **lm-sensors**, la sua configurazione e tutti i pacchetti che servivano solo per installarlo ed eseguirlo (con le relative configurazioni):

```
apt autoremove --purge PACCHETTO1 . . .
```

## Esercizio 4 (2 min.)

Installate il pacchetto binario **apache2** (Web server Apache, v2.4).

Verificate la presenza della directory seguente:

**/etc/apache2**

Rimuovete il pacchetto con il comando:

**apt remove apache2**

La directory è ancora presente?

Se è ancora presente, cosa dovete fare per rimuoverla?

# Una avvertenza

(State calmi; non avete sbagliato niente)

Se, al termine dello svolgimento del precedente esercizio, rimane qualche residuo nella directory `/etc/apache2`, non preoccupatevi.

Un altro pacchetto ha installato file di configurazione in `/etc/apache2`; la rimozione del pacchetto `apache2` non tocca tali file.



# Don't panic!

(OK, panic!)



# Dove sono i pacchetti scaricati?

(Nella directory `/var/cache/apt/archives`)

Una volta scaricati, i pacchetti sono conservati in una apposita directory "cache":

`/var/cache/apt/archives`

Visualizzate il contenuto di tale directory:

```
ls -l /var/cache/apt/archives
```

# Quanti sono i file?

(Tanti quanti sono i pacchetti scaricati almeno una volta)

Quanti sono i file? Sono uno per ogni pacchetto installato almeno una volta.

Contateli:

```
ls /var/cache/apt/archives/*.deb | wc -l
```

# Quanto spazio occupano?

(Diverse centinaia di MB)

Quanto spazio su disco occupano?

`apropos -s1 disk usage` → si scopre il comando `du`.

`man du` → si scoprono le opzioni `-s` (consumo totale) e `-h` (stampa "human readable").

→ Il comando seguente stampa il consumo totale dei pacchetti:

```
du -hs /var/cache/apt/archives
```

# Cancellazione cache dei pacchetti

(Si usa il comando **apt clean**)

Il comando **apt clean** rimuove i file **\*.deb** dalla directory cache del sistema APT:

**/var/cache/apt/archives**

Eseguite il comando periodicamente, soprattutto al termine del passaggio da una distribuzione alla successiva.

## Esercizio 5 (1 min.)

Pulite la cache dei pacchetti.

Quanto spazio su disco è stato recuperato dopo la pulizia?

# Aggiornamento dei pacchetti

(Upgrade; non considera le mutate dipendenze fra pacchetti)

Per aggiornare tutti i pacchetti installati, si utilizza il seguente comando (da amministratore):

```
apt upgrade
```

L'opzione **-u** forza la stampa dell'elenco dei pacchetti da aggiornare. Nella versione corrente di APT è attivata di default.

Non ci si dimentichi dell'**apt update** preliminare!

# Aggiornamento distribuzione

(Full-upgrade o dist-upgrade; considera le mutate dipendenze fra pacchetti)

Per aggiornare ad una nuova distribuzione (dopo aver modificato opportunamente il file di configurazione **sources.list**), si utilizza il seguente comando (da amministratore):

```
apt full-upgrade
```

Per ragioni di compatibilità è accettata anche il sottocomando precedentemente in uso:

```
apt dist-upgrade
```



# Aggiornamento distribuzione

(Dist-upgrade; considera le mutate dipendenze fra pacchetti)

Perché dist-upgrade e non un semplice upgrade?

Nel cambio di distribuzione possono subentrare modifiche nelle dipendenze fra pacchetti (alcune librerie spariscono, ne entrano altre, ...).

Dist-upgrade gestisce tali modifiche (rimuove le vecchie dipendenze ed installa le nuove).

Upgrade non gestisce tali modifiche (prova ad avanzare tutti i pacchetti alla nuova versione).

## Esercizio 6 (3 min.)

Sincronizzate il repository locale della vostra distribuzione con quello remoto.

Aperte la pagina di manuale di **apt** ed individuate una opzione per simulare l'aggiornamento della distribuzione.

Eseguite una simulazione di aggiornamento della distribuzione e valutate quanti pacchetti binari dovrebbero essere aggiornati.

# **RICERCA E ANALISI DEI PACCHETTI**

# Scenario e interrogativi 1/2

(È possibile individuare il nome esatto di un pacchetto binario?)

**Scenario:** l'utente non riesce ad individuare i nomi esatti dei pacchetti binari e non riesce, pertanto, ad installarli.

## **Interrogativi:**

Esistono comandi che permettono di individuare il nome esatto di un pacchetto binario?

Partendo, magari, da singoli termini di ricerca?

Oppure da espressioni regolari?

# Scenario e interrogativi 2/2

(È possibile estrarre i metadati di un pacchetto binario?)

**Scenario:** l'utente vuole estrarre tutti i metadati di un pacchetto binario, ad esempio:

- indirizzo di posta elettronica autore;
- hash del pacchetto;
- elenco dipendenze;

...

## **Interrogativi:**

Quali comandi permettono di estrarre i metadati di un pacchetto binario?

# Ricerca sui pacchetti binari

(Si usa il comando **apt search**)

Il comando seguente:

```
apt search REGEX1 REGEX2 . . .
```

stampa:

nome;

descrizione breve (una riga);

di ciascun pacchetto che verifica le espressioni regolari  
**REGEX1, REGEX2, ...**

Quali fonti sono usate per la ricerca?

Il nome del pacchetto.

La descrizione testuale di un pacchetto.

# Un esempio concreto

(Ricerca di tutti i pacchetti binari il cui nome/descrizione inizia per **tmux**)

Si cerchino tutti i pacchetti il cui nome esatto (oppure la cui descrizione esatta) inizia con la stringa **tmux**.

```
apt search '^tmux.*$'
```

Descrizione  
breve

Nome esatto  
pacchetto /  
Distribuzione

```
studente@debian:~$ apt search '^tmux.*$'
Ordinamento... Fatto
Ricerca sul testo... Fatto
python-tmux/stable 1.5.0a-1 all
  tmux session manager (Python 2)
python3-tmux/stable 1.5.0a-1 all
  tmux session manager (Python 3)
tmux/stable 2.8-3 amd64
  multiplexer per terminale
tmux-plugin-manager/stable 3.0.0-1 all
  gestore di plugin per tmux basato su git
tmux-themepack-jimeh/stable 0+git20180910-126150d-1 all
  pack of various themes for tmux by jimeh
tmuxinator/stable 0.15.0-1 all
  crea e gestisce facilmente sessioni di tmux
tmuxp/stable 1.5.0a-1 all
  tmux session manager
```

# Il match dell'espressione regolare

(All'inizio del nome del pacchetto o della descrizione breve)

L'espressione regolare `^tmux.*$` è applicata sia al nome esatto del pacchetto sia alla descrizione breve. Se una delle due stringhe la verifica, viene stampato l'output.

```
studente@debian:~$ apt search '^tmux.*$'
Ordinamento... Fatto
Ricerca sul testo... Fatto
python-tmuxp/stable 1.5.0a-1 all
  tmux session manager (Python 2)

python3-tmuxp/stable 1.5.0a-1 all
  tmux session manager (Python 3)

tmux/stable 2.8-3 amd64
  multiplexer per terminale

tmux-plugin-manager/stable 3.0.0-1 all
  gestore di plugin per tmux basato su git

tmux-themepack-jimeh/stable 0+git20180910-126150d-1 all
  pack of various themes for tmux by jimeh

tmuxinator/stable 0.15.0-1 all
  crea e gestisce facilmente sessioni di tmux

tmuxp/stable 1.5.0a-1 all
  tmux session manager
```



# Esclusione della descrizione breve

(Si usa l'opzione **--names-only** del comando **apt search**)

L'opzione **--names-only** esclude la descrizione breve del pacchetto dal parco stringhe da controllare.

```
apt --names-only search '^tmux.*$'
```

```
studente@debian:~$ apt --names-only search '^tmux.*$'
Ordinamento... Fatto
Ricerca sul testo... Fatto
tmux/stable 2.8-3 amd64
multiplexer per terminale
tmux-plugin-manager/stable 3.0.0-1 all
gestore di plugin per tmux basato su git
tmux-themepack-jimeh/stable 0+git20180910-126150d-1 all
pack of various themes for tmux by jimeh
tmuxinator/stable 0.15.0-1 all
crea e gestisce facilmente sessioni di tmux
tmuxp/stable 1.5.0a-1 all
tmux session manager
```

Manca il  
pacchetto  
binario

**python3-tmuxp**

## Esercizio 7 (3 min.)

Cercate tutti i videogiochi disponibili nei repository configurati.

Raffinate la ricerca e cercate un gioco di biliardo.

Installate i pacchetti software corrispettivi ed eseguite i giochi.

Avete incontrato particolari difficoltà?

# Convenzioni nei nomi di pacchetti

(Pacchetti che iniziano con **lib**: librerie dinamiche)

I pacchetti che:

- iniziano con la stringa **lib**

- non terminano con la stringa **-dev**

- non terminano con la stringa **-dbg**

- non terminano con la stringa **-doc**

offrono librerie dinamiche.

Eccezione notevole: LibreOffice (pacchetto **libreoffice**).

# Convenzioni nei nomi di pacchetti

(Pacchetti iniziati con **lib** e terminanti con **-dev**: librerie statiche, file include)

I pacchetti che:

- iniziano con la stringa **lib**;

- terminano con la stringa **-dev**;

offrono:

- le corrispondenti versioni statiche delle librerie

- (file con nomi del tipo **libXYZ.a**);

- i file include della libreria.

# Convenzioni nei nomi di pacchetti

(Pacchetti che terminano con **doc**: documentazione)

I pacchetti che terminano con la stringa **-doc** contengono documentazione relativa ad uno specifico pacchetto.

Ad esempio: il pacchetto binario **libx11-doc** offre la documentazione per i contenuti del pacchetto binario **libx11**.

Di che documentazione si parla?

Pagine di manuale aggiuntive.

Manuali in formato TXT, HTML, TROFF, PS, PDF.

# Convenzioni nei nomi di pacchetti

(Pacchetti che terminano con **dbg**: tabelle dei simboli)

I pacchetti che terminano con la stringa **-dbg** contengono le tabelle dei simboli di software specifici.

Ad esempio: il pacchetto binario **libc6-dbg** offre la tabella dei simboli della libreria del C, contenuta nel pacchetto binario **libc6**.

La tabella dei simboli facilita il debugging del software tramite il debugger **gdb**.

## Esercizio 8 (2 min.)

Installate la documentazione relativa alla Java Virtual Machine OpenJDK.

# Stampa metadati di un pacchetto

(Si usa il comando **apt show**)

Una volta identificato il nome esatto di un pacchetto, è possibile stampare i suoi metadati con il comando:

```
apt show PACCHETTO1 PACCHETTO2 . . .
```

Come si può facilmente notare, l'output del comando è esaustivo.

Chi è interessato ad approfondire lo studio dei metadati, può trovare le informazioni nei Capitoli 3, 5 e 7 del Debian Policy Manual.



# Dipendenze dirette

(Si usa il comando **apt depends**)

Le **dipendenze dirette** dirette di un pacchetto binario P sono i pacchetti binari  $P_1, \dots, P_n$  di cui è richiesta l'installazione per poter eseguire il software contenuto in P.

Le dipendenze dirette di un pacchetto possono essere stampate con il comando **apt depends**:

```
apt depends PACCHETTO1 PACCHETTO2 . . .
```

# Un esempio concreto

(Quali sono le dipendenze dirette di **bash**?)

Quali sono le dipendenze di bash?

**apt depends bash**

```
studente@debian:~$ apt depends bash
bash
Pre-dipende: libc6 (>= 2.15)
Pre-dipende: libtinfo6 (>= 6)
Dipende: base-files (>= 2.1.12)
Dipende: debianutils (>= 2.15)
Va in conflitto: bash-completion (<< 20060301-0)
Raccomanda: bash-completion (>= 20060301-0)
Consiglia: bash-doc
Sostituisce: bash-completion (<< 20060301-0)
Sostituisce: bash-doc (<= 2.05-1)
```

# Un esempio concreto

(Quali sono le dipendenze dirette di **bash**?)

Quali sono le dipendenze di **bash**?

**apt depends bash**

```
studente@debian:~$ apt depends bash
bash
Pre-dipende: libc6 (>= 2.15)
Pre-dipende: libtinfo6 (>= 6)
Dipende: base-files (>= 2.1.12)
Dipende: debianutils (>= 2.15)
Va in conflitto: bash-completion (<< 20060301-0)
Raccomanda: bash-completion (>= 20060301-0)
Consiglia: bash-doc
Sostituisce: bash-completion (<< 20060301-0)
Sostituisce: bash-doc (<= 2.05-1)
```

Il pacchetto binario **bash** pre-dipende da **libc6** e da **libtinfo6**.

“Pre-Dipende” forza l'installazione completa dei pacchetti utili alla installazione di **bash**, prima della installazione di **bash**. Ad esempio, **libc6** viene scaricato, spacchettato, configurato prima di **bash**.

# Un esempio concreto

(Quali sono le dipendenze dirette di **bash**?)

Quali sono le dipendenze di **bash**?

**apt depends bash**

```
studente@debian:~$ apt depends bash
bash
Pre-dipende: libc6 (>= 2.15)
Pre-dipende: libtinfo6 (>= 6)
Dipende: base-files (>= 2.1.12)
Dipende: debianutils (>= 2.15)
Va in conflitto: bash-completion (<< 20060301-0)
Raccomanda: bash-completion (>= 20060301-0)
Consiglia: bash-doc
Sostituisce: bash-completion (<< 20060301-0)
Sostituisce: bash-doc (<= 2.05-1)
```

Il pacchetto binario **bash** dipende da **base-files** e **debianutils**.

“Dipende” forza l’installazione completa dei pacchetti utili alla esecuzione di **bash**, insieme alla installazione di **bash**.

Ad esempio , **base-files** viene scaricato, spaccettato, configurato insieme a **bash**.

# Un esempio concreto

(Quali sono le dipendenze dirette di **bash**?)

Quali sono le dipendenze di **bash**?

**apt depends bash**

```
studente@debian:~$ apt depends bash
bash
Pre-dipende: libc6 (>= 2.15)
Pre-dipende: libtinfo6 (>= 6)
Dipende: base-files (>= 2.1.12)
Dipende: debianutils (>= 2.15)
Va in conflitto: bash-completion (< 20060301-0)
Raccomanda: bash-completion (>= 20060301-0)
Consiglia: bash-doc
Sostituisce: bash-completion (< 20060301-0)
Sostituisce: bash-doc (<= 2.05-1)
```

Il pacchetto binario **bash** condivide alcuni file e/o directory con **bash-completion**.

**/usr/bin**

**/usr/share/doc**

Pertanto, se si spacchetta **bash** si distrugge un file di **bash-completion** (con effetti disastrosi).

“Va in conflitto” blocca l’installazione.

# Un esempio concreto

(Quali sono le dipendenze dirette di **bash**?)

Quali sono le dipendenze di bash?

**apt depends bash**

```
studente@debian:~$ apt depends bash
bash
Pre-dipende: libc6 (>= 2.15)
Pre-dipende: libtinfo6 (>= 6)
Dipende: base-files (>= 2.1.12)
Dipende: debianutils (>= 2.15)
Va in conflitto: bash-completion (<< 20060301-0)
Raccomanda: bash-completion (>= 20060301-0)
Consiglia: bash-doc
Sostituisce: bash-completion (<< 20060301-0)
Sostituisce: bash-doc (<= 2.05-1)
```

L'installatore raccomanda (ed, a tutti gli effetti pratici, esegue) l'installazione di **bash-completion** a partire dalla versione **20060301-0**.

Tale pacchetto offre funzionalità di completamento dei comandi molto avanzate.

# Un esempio concreto

(Quali sono le dipendenze dirette di **bash**?)

Quali sono le dipendenze di bash?

**apt depends bash**

```
studente@debian:~$ apt depends bash
bash
Pre-dipende: libc6 (>= 2.15)
Pre-dipende: libtinfo6 (>= 6)
Dipende: base-files (>= 2.1.12)
Dipende: debianutils (>= 2.15)
Va in conflitto: bash-completion (<= 20060301-0)
Raccomanda: bash-completion (>= 20060301-0)
Consiglia: bash-doc
Sostituisce: bash-completion (<= 20060301-0)
Sostituisce: bash-doc (<= 2.05-1)
```

Per una migliore fruizione del software offerto dal pacchetto **bash**, il gestore dei pacchetti consiglia l'installazione di **bash-doc**.

**Occhio!** Il gestore dei pacchetti consiglia, ma non installa.

# Un esempio concreto

(Quali sono le dipendenze dirette di **bash**?)

Quali sono le dipendenze di bash?

**apt depends bash**

```
studente@debian:~$ apt depends bash
bash
Pre-dipende: libc6 (>= 2.15)
Pre-dipende: libtinfo6 (>= 6)
Dipende: base-files (>= 2.1.12)
Dipende: debianutils (>= 2.15)
Va in conflitto: bash-completion (<< 20060301-0)
Raccomanda: bash-completion (>= 20060301-0)
Consiglia: bash-doc
Sostituisce: bash-completion (<< 20060301-0)
Sostituisce: bash-doc (<= 2.05-1)
```

La relazione “Sostituisce” indica al gestore dei pacchetti di non considerare un errore il fatto che bash sovrascriva **/usr/bin** e **/usr/share/doc**. Non solo: il gestore dei pacchetti include **bash** nell'elenco dei “possessori” di tali file.



## Esercizio 9 (1 min.)

Individuate le dipendenze dirette del pacchetto software **libreoffice**.

# Caratteri presenti nell'output

(I)

L'output di **apt depends** nell'esercizio precedente contiene il carattere |, ad esempio:

```
|Dipende: libreoffice-avmedia-backend-gstreamer
```

```
Dipende: libreoffice-avmedia-backend-vlc
```

Che significato ha questo carattere?

# Caratteri presenti nell'output

(I)

Il carattere | introduce l'OR di più pacchetti. Nell'esempio in questione, **libreoffice** dipende da uno dei due pacchetti seguenti:

**libreoffice-avmedia-backend-gstreamer**

**libreoffice-avmedia-backend-vlc**

Se almeno uno di questi due pacchetti è installato, le dipendenze sono soddisfatte.

Se nessun pacchetto installato, APT installa il primo nell'elenco (il backend Gstreamer).

# Caratteri presenti nell'output

(<>)

L'output di **apt depends** nell'esercizio precedente contiene i caratteri <>, ad esempio:

Consiglia: <hyphen-hyphenation-patterns>

Che significato ha un pacchetto incastrato tra i due caratteri <>?

# Caratteri presenti nell'output

(<>)

I caratteri <> delimitano il nome di un pacchetto virtuale (cfr. sezione “Pacchetti binari speciali” nel seguito della lezione).

# Dipendenze inverse

(Si usa il comando `apt rdepends`)

Le **dipendenze inverse** di un pacchetto binario  $P$  sono i pacchetti binari  $P_1, \dots, P_n$  che richiedono l'installazione di  $P$  per poter eseguire i propri software.

È il contrario di “dipendenza diretta”.

Le dipendenze inverse di un pacchetto possono essere stampate con il comando `apt rdepends`:

```
apt rdepends PACCHETTO1 PACCHETTO2 . . .
```

# Un esempio concreto

(Quali sono le dipendenze inverse di **bash**?)

Quali sono le dipendenze di BASH?

```
apt rdepends bash
```

Sono mostrati i (tanti) pacchetti software che dipendono da **bash** per poter funzionare.

## Esercizio 10 (1 min.)

Individuate le dipendenze inverse del pacchetto software **libreoffice**.



# Il grafo delle dipendenze dirette

(Come si calcola? Come si visualizza?)

È possibile calcolare e disegnare il grafo delle dipendenze dirette di un pacchetto binario.

Si usano due comandi.

Comando **debtree** (pacchetto **debtree**): calcola il grafo delle dipendenze di un pacchetto.

Comando **dot** (pacchetto **graphviz**): disegna grafi.

Si installino i due pacchetti software:

```
apt install debtree graphviz
```

# Calcolo del grafo

(Black magic)

Per calcolare il grafo delle dipendenze di **bash**:

```
debtree --no-recommends
```

```
--no-conflicts bash > bash.dot
```

Il risultato è un file testuale (nel linguaggio DOT) rappresentante il grafo delle dipendenze.

Per mantenere ragionevole la dimensione del grafo, sono esclusi i pacchetti raccomandati e quelli in conflitto.

# Disegno del grafo

(More black magic)

Per produrre un'immagine del grafo delle dipendenze di **bash**:

```
dot -Tsvg bash.dot > bash.svg
```

Il risultato è un'immagine vettoriale (**Scalable Vector Graphics, SVG**) mostrante il grafo delle dipendenze di **bash**.

# Visualizzazione del grafo

(Deepest black magic)

Per visualizzare l'immagine prodotta:

**inkscape bash.svg**

Comandi di **inkscape**:

Tasto +	→ aumenta lo zoom
Tasto -	→ diminuisce lo zoom
tasti cursore	→ spostano l'immagine

## Esercizio 11 (3 min.)

Calcolate, disegnate e visualizzate il grafo delle dipendenze dirette del pacchetto binario **coreutils**. Leggete la pagina di manuale del comando opportuno e trovate un modo per calcolare il grafo contenente anche le dipendenze inverse.

# Ricerca del contenuto dei pacchetti

(Si usa il comando **apt-file**)

Il comando **apt-file** (contenuto nel pacchetto binario **apt-file**) permette di effettuare ricerche più sofisticate sui pacchetti.

Lo si installi con il comando seguente:

```
apt install apt-file
```

# Aggiornamento dell'indice

(Si usa il comando **apt-file update**)

Per poter operare, **apt-file** usa un indice locale che associa un file al nome del pacchetto che lo contiene. L'indice è scaricato da un sito Web.

L'utente deve aggiornare periodicamente l'indice con il comando seguente (lanciato da amministratore):

```
apt-file update
```

# Elenco file di un pacchetto

(Si usa il comando **apt-file list**)

Per capire quali file sono forniti da un pacchetto software potete usare l'argomento **list** con l'opzione **-x** (ricerca di espressione regolare):

```
apt-file list -x '^libc6$'
```

Il comando funziona anche su pacchetti non installati:

```
apt-file list -x '^screen$'
```



# Ricerca del pacchetto a partire dal file

(Si usa il comando **apt-file search**)

Per capire quale pacchetto fornisce un file specifico potete usare l'argomento **search** con l'opzione **-x** (ricerca di espressione regolare):

```
apt-file search -x '^/usr/bin/top$'
```

## Esercizio 12 (2 min.)

In quale pacchetto binario si trova l'eseguibile `dd`?

# **PACCHETTI BINARI SPECIALI**

# Scenario e interrogativi 1/2

(È possibile installare software complessi in un colpo solo?)

**Scenario:** l'utente vorrebbe installare un software complesso, composto da svariati pacchetti binari. Classico esempio: un ambiente desktop (GNOME, KDE, ...).

## **Interrogativi:**

È possibile installare in un colpo solo il software? Oppure è necessario installare a mano tutti i pacchetti relativi?

# Scenario e interrogativi 2/2

(È possibile scegliere una variante di un software fra quelle disponibili?)

**Scenario:** pacchetti software distinti possono fornire funzionalità molto simili, se non identiche.

Esempi: le varie edizioni della Java Virtual Machine, gli editor testuali, ...

## **Interrogativi:**

È possibile capire quali varianti distinte di un software esistono?

È possibile scegliere una variante opportuna del software?

# Metapacchetti

(Un pacchetto installa un gruppo di pacchetti)

Un **metapacchetto** è un pacchetto che installa pochi file (tipicamente, documentazione) e dipende direttamente da un gruppo di pacchetti. Il suo scopo primario è quello di installare il gruppo di pacchetti.

# Un esempio concreto

(Analisi del pacchetto software **gnome**)

Si consideri il pacchetto binario di nome **gnome**. Si elenchi il contenuto del pacchetto con il comando **apt-file**:

```
apt-file list -x '^gnome$'
```

Quanti file contiene il pacchetto? Una manciata.

Di per sé, il pacchetto non installa alcunché di significativo.

# Un esempio concreto

(Analisi del pacchetto software **gnome**)

Si stampi l'elenco delle dipendenze del pacchetto binario **gnome**:

```
apt depends gnome
```

Quante dipendenze espone il pacchetto? Una marea; l'intero gruppo di pacchetti contribuente alle funzionalità del desktop GNOME.



## Esercizio 13 (2 min.)

Analizzate il pacchetto binario **kde-full**.

Quali file fornisce?

Da quali pacchetti binari dipende?

Secondo voi, è un metapacchetto?

# Pacchetti virtuali

(Un pacchetto può essere soddisfatto da tante alternative possibili)

Un **pacchetto virtuale** è un pacchetto fantasma che, di per sé, non fornisce file, bensì una lista di pacchetti che possono soddisfarlo.

In questo aspetto differisce dal metapacchetto (che, comunque, installa file sul file system).

Un pacchetto virtuale è un modello generico di pacchetto che fornisce una data funzionalità (editor, JVM, browser Web, ...) ed indica quali pacchetti offrono tale funzionalità.

# Un esempio concreto

(Analisi del pacchetto software **editor**)

Si consideri il pacchetto binario di nome **editor**. Esso rappresenta la categoria degli editor di testo.

Si visualizzino i suoi metadati:

```
apt show editor
```

Si ottiene un errore; **editor** è un pacchetto fantasma, e come tale non esiste in concreto.

Tuttavia, se si prova ad installare il pacchetto:

```
apt install editor
```

APT elenca le alternative disponibili.

# Installazione di un pacchetto virtuale

(Fatta a mano dall'utente)

L'utente sceglie a mano una delle alternative dall'elenco precedente e la installa con il consueto comando **apt install**.

È possibile installare nello stesso SO Debian due o più alternative di **editor**?

Ad esempio, **vim** e **emacs-gtk**?

# Installazione di un pacchetto virtuale

(Fatta a mano dall'utente)

Certo che è possibile!

Installate entrambi i pacchetti:

```
apt install vim emacs-gtk
```

Entrambe le alternative forniscono il supporto al pacchetto virtuale **editor**. Il campo “Provides” dei metadati contiene, infatti, la stringa **editor**.

```
apt show vim emacs-gtk | less -Mr
```

# Implementazione delle alternative

(Un sistema alquanto complesso)

Una delle alternative disponibili viene scelta come “default di sistema” e viene collegata simbolicamente al comando `/usr/bin/editor`, generato dal pacchetto virtuale `editor`.

```
ls -l /usr/bin/editor
```

In realtà, questo file punta ad un file intermedio nella directory `/etc/alternatives` che contiene un file per ogni possibile pacchetto virtuale.

Il file è `/etc/alternatives/editor`, e punta a sua volta a `/bin/nano`.

Scrivendo il comando `editor`, parte `nano`.

# Orrore (neanche tanto, alla fine)!

(L'editor di sistema è `/bin/nano???`)



# Gestione delle alternative

(Si usa il comando **update-alternatives**)

Il comando **update-alternatives** permette di:

- elencare tutti i pacchetti virtuali e l'alternativa di default corrispondente;
- elencare tutte le alternative di un pacchetto virtuale;
- selezionare come default una fra le alternative disponibili.



# Elenco dei pacchetti virtuali disponibili

(Si usa il comando `update-alternatives --get-selections`)

Il comando seguente elenca i nomi di tutti i pacchetti virtuali disponibili sul SO Debian:

```
update-alternatives --get-selections
```

Come si può notare, esistono numerosi comandi virtuali (come **editor**), ciascuno dei quali fornito da una delle sue tante alternative.

Ben 95 sulla macchina del docente!

# Elenco delle alternative installate

(Si usa il comando `update-alternatives --list`)

Il comando seguente elenca tutte i pacchetti alternativi già installati per un dato pacchetto virtuale sul SO Debian (ad esempio, `editor`):

```
update-alternatives --list editor
```

Come si può notare, sono già stati installati alcuni editor.

# Scelta dell'alternativa di default

(Si usa il comando `update-alternatives --config`)

Come già visto in precedenza, scrivendo **editor** il SO fa partire l'editor alternativo scelto per default.

L'editor di default è inizialmente scelto in base ad un sistema interno di priorità.

Il comando seguente permette di modificare l'alternativa di default attuale per **editor**:

```
update-alternatives --config editor
```

# Scelta dell'alternativa di default

(Si usa il comando `update-alternatives --config`)

Ad esempio, per cambiare l'alternativa di default di `editor` da `nano` a `vim.basic`, si esegue:

```
update-alternatives --config editor
```

Dopodiché, si immette il numero di selezione per l'editor con eseguibile `/usr/bin/vim.basic`.

Infine, si esegue `editor`.

Dovrebbe partire `vim` al posto di `nano`.

## Esercizio 14 (3 min.)

Individuate il pacchetto virtuale del browser Web grafico.  
Individuate le possibili alternative e l'alternativa di default.

Cambiare l'alternativa con un browser a scelta.

Eseguite il comando virtuale.

Verificare l'esecuzione del browser scelto.