

GNU Guix

Pacchetti, ambienti e rollback senza paura (anche se usi già Debian)

Francesco P. Lovergine

LinuxDay — Bari, 24 ottobre 2026

Debian Developer & Geodata scientist

- Classe 1967 — laurea 1991
- Ricercatore CNR dal 1997: infrastrutture geospaziali e telerilevamento SAR/ottico
- Debian Developer dal 2000 (DebianGis blend founder)
- OSMapper dal 2008
- Ex presidente LUG Bari (2002–2009)
- Mantiene pacchetti e contribuisce upstream a troppi software per ricordarseli tutti...
- FOSS activist da sempre — geek per nascita :-)

Di cosa parliamo oggi

Il problema

Cos'è GNU Guix

Demo: proviamolo insieme

Insidie comuni per chi inizia

Per concludere

Il problema

Un rituale che conosciamo tutti

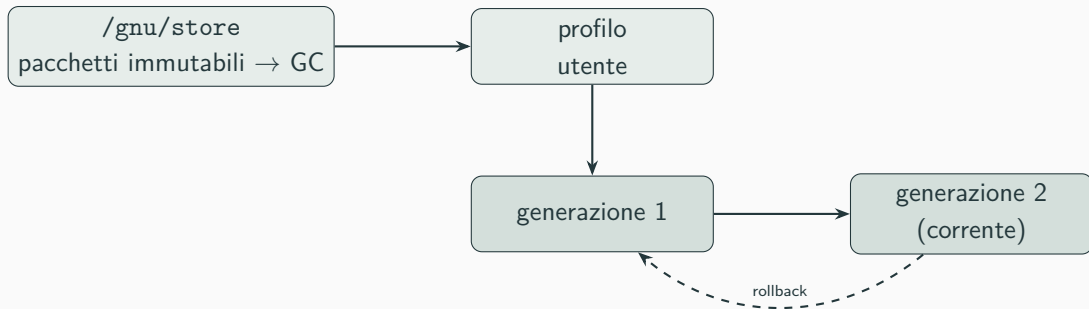
- `apt install ...` poi `apt remove --purge` per "pulire" quando qualcosa non serve più
 - Un ambiente virtuale diverso per ogni linguaggio: `venv`, `rbenv`, `nvm`, ...
 - La sensazione di non poter mai essere sicuri al 100% che una macchina "pulita" si comporti come quella su cui abbiamo sviluppato
 - I sistemi di pacchettizzazione (*deb* o *rpm*) sono dei *PITA* che stanno su con la colla a caldo ($\rightarrow$ `debhelper`).
 - I binari e le dipendenze funzionano qui e adesso (forse), ma nel futuro?
 - La **SBOM** ($\rightarrow$ *reproducible computing*) è ancora una chimera, una triste verità...
- $\rightarrow$ "Funziona sul mio computer" resta un problema irrisolto

Cos'è GNU Guix

Un gestore di pacchetti ispirato alla programmazione **funzionale**
(e, se vuoi, un intero sistema operativo)

- Ogni pacchetto è costruito in isolamento, a partire da un hash calcolato sulle sue dipendenze esatte sulla base dell'albero dei sorgenti
- Installare non tocca né sovrascrive nulla del sistema esistente
- Ogni *transazione* applicata al sistema (install, upgrade, remove) è **atomica e reversibile**
- Stessa configurazione $\Rightarrow$ stesso ambiente, ovunque
- Monorepo `git` per tutto ed estendibile
- Un solo linguaggio (Scheme) per dominarli tutti (GNU Guile, il mio tessssoro!)

I concetti chiave, in un disegno



Ogni cambiamento crea una nuova *generazione* del profilo: tornare indietro è solo puntare a quella precedente. I link simbolici sono usati per rispettare FHS, ma praticamente tutto il sistema è read-only.

Non serve cambiare distribuzione

- Guix si installa **sopra** Debian, Ubuntu, Fedora, ... come gestore di pacchetti aggiuntivo ($\rightarrow$ *foreign distribution*)
- Convive pacificamente con `apt/dnf`: non li sostituisce, li affianca
- Utile per strumenti aggiornati o versioni specifiche senza inquinare il sistema di base
- Il canale `nonguix` estende la disponibilità di software (driver, firmware, ...) oltre i pacchetti liberi ufficiali
- Altri canali (per es. `guix-science`) sono di interesse per applicazioni specifiche, come l'HPC e la scienza.

Ha anche dei difetti

- Piccolo team di sviluppo, ma molto inclusivo
- Lisp non è il linguaggio di programmazione più convenzionale del mondo...
- I pacchetti sono tanti, ma non sempre aggiornati
- In Guix SD dimenticati systemd, un pro o contro a seconda dei PV
- Alcune scelte sono da puristi, e quindi molto opinionate (non-free, no-AI)
- Le operazioni di pull/upgrade non sono fulmini di guerra...
- Anche con i substitutes capita di doversi fare i build in casa

Demo: proviamolo insieme

Per questo vi rimando ad un semplice mio post, con guida passo-passo:

`https://lovergine.com/creating-your-own-guix-substitute-server.html`

con apposita sezione generale di installazione su Debian e derivate.

Demo dal vivo

1. Installare, rimuovere, aggiornare
2. Rompere qualcosa ... e tornare indietro
3. Un ambiente di sviluppo riproducibile in 2 minuti
4. `guix shell` in tutte le salse, anche in container
5. Riproducibilità nel tempo con `guix time-machine`
6. (*se c'è tempo*) un assaggio di Guix System e `guix home`

Demo 1 — I comandi base

```
$ guix install hello  
$ hello  
$ guix remove hello  
$ guix upgrade
```

- Cosa succede "sotto": build/download in /gnu/store, poi un link nel profilo dell'utente
- Niente viene sovrascritto: il pacchetto precedente resta disponibile finché non viene ripulito con `guix gc`

Demo 1bis — Cercare e capire i pacchetti

```
$ guix search editor markdown  
$ guix show hello  
$ guix size hello  
$ guix graph hello | dot -Tpng > hello.png
```

- `guix search` / `guix show`: ricerca e scheda del pacchetto (versione, licenza, sorgenti, descrizione)
- `guix size`: quanto pesa **con tutte** le dipendenze
- `guix graph`: il grafo delle dipendenze, lo stesso concetto del disegno visto prima, ma per un pacchetto vero

Demo 2 — Il rollback

```
$ guix install cowsay  
$ guix upgrade % qualcosa va storto...  
$ guix package --roll-back  
$ guix package --list-generations
```

Il punto della demo

Nessun backup, nessuna reinstallazione: si torna indietro con un comando perché ogni generazione precedente è ancora lì, intatta.

Demo 2bis — Generazioni e profili

```
$ guix package --list-generations
$ guix package --switch-generation=2
$ guix package --delete-generations=1m
$ guix install -p ~/profili/lavoro emacs
$ GUIX_PROFILE=~/profili/lavoro
$ . $GUIX_PROFILE/etc/profile
```

- Ogni operazione crea una nuova **generazione**: si può tornare a una qualsiasi, non solo alla precedente
- `-p`: profili separati (lavoro, test, ...) con pacchetti e rollback indipendenti
- `--delete-generations` + `guix gc`: recuperare spazio quando serve

Demo 3 — Un ambiente riproducibile

manifest.scm

```
;; ambiente per un progetto Python
(specifications->manifest
 (list "python"
       "python-numpy"
       "python-requests"))
```

```
$ guix shell -m manifest.scm
$ python3 -c \
    "import numpy; \
    print(numpy.__version__)"
```

- Il file `manifest.scm` si versiona insieme al progetto
- Un collega ottiene **esattamente** lo stesso ambiente con lo stesso comando

Demo 3bis — guix shell, altri usi

```
$ guix shell python python-numpy -- python3
$ guix shell --pure coreutils bash -- bash
$ guix shell -D hello
$ guix shell --container ruby -- irb
```

- Senza `-m`: basta elencare i pacchetti per un ambiente "usa e getta", un comando e via
- `--pure`: azzera le variabili d'ambiente dell'host, niente PATH "sporco" che si infila di nascosto
- `-D pacchetto`: tira dentro le dipendenze di **sviluppo** di un pacchetto esistente — utile per mettere le mani sui suoi sorgenti senza installarle una a una
- `--container`: isolamento totale, anche dal filesystem dell'host — la riproducibilità portata all'estremo

Demo 3ter — Container: isolamento vero

```
$ guix shell -C python -- python3
$ guix shell -C -N curl -- curl https://guix.gnu.org
$ guix shell -C --expose=/etc/os-release=/etc/os-release \
    coreutils -- cat /etc/os-release
$ guix shell -C --share=$HOME/progetto python -- bash
```

- -C: nuovo namespace, solo i pacchetti richiesti e la directory corrente sono visibili
- Rete **disattivata** di default; -N la riattiva
- --share (lettura/scrittura) e --expose (sola lettura): si decide cosa mostrare dell'host
- Utile per provare software non fidato, o per verificare che un progetto non dipenda da nulla di nascosto

Demo 4 — Un viaggio nel tempo: Python nel 2022

```
$ guix shell python -- python3 --version  
Python 3.12.12 # Guix di oggi
```

```
$ guix time-machine -q \  
  --commit=8e2f32cee982d42a79e53fc1e9aa7b8ff0514714 -- \  
  shell python -- python3 --version  
Python 3.9.9 # Guix 1.4.0, 18/12/2022
```

- Il commit è la **release 1.4.0**: stesso comando, ma con i pacchetti di allora
- Niente installato, il sistema attuale **non viene toccato**
- `guix describe -f channels > channels.scm + -C`: la stessa versione si ritrova fra due anni

(Opzionale) Un assaggio di Guix System

```
(operating-system
  (host-name "esempio")
  (timezone "Europe/Rome")
  (packages (cons* (specification->package "vim")
                   %base-packages))
  ;; ... resto della configurazione
)
```

- L'intero sistema operativo descritto in un unico file
- `guix system reconfigure` applica la configurazione in modo altrettanto atomico e reversibile
- *Slide "cuscinetto": salta se il tempo stringe*

(Opzionale) guix home: lo stesso, ma per voi

```
home-configuration.scm
```

```
(use-modules (gnu home)
             (gnu services))

(home-environment
 (packages
  (specifications->packages
   (list "git" "htop"
         "ripgrep"))))
(services
 (list (service
       home-bash-service-type))))
```

```
$ guix home reconfigure \
    home-configuration.scm
$ guix home list-generations
$ guix home roll-back
```

- Come Guix System, ma per il vostro \$HOME: pacchetti, dotfile e servizi utente in un unico file
- Funziona anche su una *foreign distribution*: non serve Guix System
- Stesse garanzie: generazioni e rollback, ma a livello di profilo utente

Insidie comuni per chi inizia

Cosa può sorprendere all'inizio

- **Certificati CA:** su una distro "foreign", `nss-certs` di Guix può entrare in conflitto col bundle di sistema — spesso conviene continuare a usare i certificati dell'host per `git/https` (dettagli nella slide seguente)
- **Desktop pesanti:** GNOME porta catene di dipendenze lunghe e più delicate da aggiornare; un desktop più leggero (XFCE) è spesso più stabile in fase di apprendimento. Particolarmente insidioso in caso di distribuzioni *roll-on* come Guix SD.
- **Niente è implicito:** nella filosofia dichiarativa di Guix, se un servizio non è nella configurazione, semplicemente non c'è — è un vantaggio, ma spiazza chi è abituato ad altro
- Il processo `init` non è `systemd` ed eventuali servizi vanno gestiti seguendo la logica del sistema ospite.

Insidie: i certificati CA su una distro “foreign”

- **Sintomo:** git, curl ... installati con Guix danno “*certificate verification failed*”
- **Causa:** i pacchetti Guix non guardano in /etc/ssl/certs, e su Debian non c'è lo store di Guix
- **Da manuale:** guix install nss-certs + variabili d'ambiente verso il suo bundle
- **Il rovescio:** è un bundle **diverso** da quello di Debian: una CA aggiunta al sistema (aziendale, universitaria) non c'è, e i siti interni falliscono

```
# di solito basta: usare i certificati dell'host
export SSL_CERT_FILE=/etc/ssl/certs/ca-certificates.crt
export SSL_CERT_DIR=/etc/ssl/certs
export GIT_SSL_CAINFO=$SSL_CERT_FILE
```

Per concludere

Cosa portate a casa stamattina

- Installare Guix accanto alla vostra distribuzione attuale
- Gestire pacchetti in modo reversibile, senza paura di rompere nulla
- Creare un primo ambiente di sviluppo riproducibile con `guix shell`

- <https://guix.gnu.org> — sito e manuale ufficiale
- <https://guix.gnu.org/manual/devel/it/> — manuale (anche in italiano, se disponibile per la versione corrente)
- Canale nonguix: <https://gitlab.com/nonguix/nonguix>
- Canali IRC/Matrix e liste help-guix@gnu.org

Grazie!

✉ frankie@fsfe.org

🌐 <https://loverage.com/>

📧 @gisgeek@floss.social

CC BY-NC-SA 4.0 Slide distribuite con licenza CC BY-NC-SA 4.0

<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>

ORGANIZZATORI



PugliaTechs

CON IL PATROCINIO DI



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

SPONSOR



DESOTECH



UGO LOPEZ.IT

COMMUNITY PARTNER



BARI
OPEN
INNOVATION
HUB



PEOPLE
IDEAS
SPACE

Unplug

