Nuovi metodi per distribuire software su Linux

Elvis Angelaccio

elvis.angelaccio@kde.org

22 ottobre 2016

Che problemi ha il Desktop Linux?

- Linux è il *kernel* più diffuso al mondo
- Linux come *desktop* non è altrettanto fortunato
- Perchè programmi come Photoshop non esistono su Linux?

Possibili risposte

Linux viene ignorato per motivi...

1) Politici (modalità complotto ON)
   - Azienda X non sviluppa su Linux a causa di accordi con azienda Y (*coff* Microsoft *coff*)
2) Commerciali
   - Linux non ha abbastanza utenti quindi gli sviluppatori lo ignorano
   - Ma se Linux non ha abbastanza app, sono gli utenti ad ignorarlo...
3) Tecnici (modalità complotto OFF)
   - Tantissimi problemi causati dalla *frammentazione* del desktop Linux
   - Se risolviamo questi problemi, nessuno avrà più scuse!

Background: come funziona un programma

- Ogni programma di una certa complessità ha bisogno di *librerie* per svolgere i suoi compiti.
  - Una libreria è come un singolo pezzo di un puzzle
- Esempi di librerie:
  - GTK/Qt: per implementare GUI
  - SDL: per audio nei videogame
  - Poppler: per rendering di documenti PDF

Linking

- Per essere usata, una libreria va linkata (“appiccicata”) all’applicazione
- Linking statico: l’app include tutte le librerie di cui ha bisogno nell’eseguibile
- Linking dinamico: l’app NON include le librerie di cui ha bisogno.
  - Il sistema operativo (SO) le fornisce quando l’app parte.
  - Il SO fa in modo che più app condividano una sola copia della stessa libreria.

IL PROBLEMA

Una giungla di distribuzioni

- Tecnicamente, non esiste UN desktop Linux
- Esistono *N* distribuzioni che forniscono *N* versioni diverse della stessa libreria
  - Moltiplica ciò per *M* librerie
- Ad esempio, Libreoffice su Ubuntu è diverso dal Libreoffice su Fedora

Una giungla di package managers

- Windows = `.exe`
- Mac = `.dmg`
- Linux = ???
- Ogni distribuzione usa un diverso formato
  - `.deb` (Ubuntu XX.YY)
  - `.deb` (Debian)
  - `.rpm` (Fedora)
  - `.rpm` (openSUSE)
  - …
- Vogliamo parlare di `npm`, `pip` e soci?

E se il software è open?

- “Se rilasci la tua applicazione con licenza open, non avrai problemi!”
  - “I packager delle varie distro si occuperanno della tua app!”
- In pratica, questo modello spesso non funziona
- Anche gli sviluppatori di software open fanno fatica a supportare il desktop Linux
  - In particolare, grossi progetti come KDE o GNOME
  - Paradossalmente *più* fatica rispetto al software closed!

LA SOLUZIONE

Cosa vogliamo

- Lato sviluppatore:
  - Un solo eseguibile che funzioni su ogni distribuzione
  - Pieno controllo sulle dipendenze dell'app
- Lato utente:
  - Ultima versione di ogni app (anche su Debian!)
  - Update automatici (non si buttano le cose buone)
  - Installazione *senza* permessi di root!
  - Sandbox, soprattutto con app proprietarie
  - Bonus: permessi ben definiti

ESEMPI DA ANALIZZARE

Android

- App Android funzionano (quasi) ovunque
- Update automatici
- Installazione senza password di root
- Sandbox
- App hanno permessi ben definiti

Steam

- Solo Ubuntu 12.04 è ufficialmente supportato...
- ... ma funziona benissimo anche altrove
- In pratica, su altre distro Steam scarica e usa la “Steam Runtime”
  - Una marea di librerie prese da Ubuntu 12.04

POSSIBILI SOLUZIONI

Linking statico

- In teoria è la soluzione più semplice
- In pratica non è molto usata
- Problemi di sicurezza
  - Se una libreria ha un bug, bisogna ricompilare *tutte* le app che la linkano staticamente
- Problemi di licenza
  - Un software proprietario non può linkare staticamente librerie LGPL

AppImage

[appimage.org]

“This is just very cool.” (*Linus Torvalds*)

AppImage: pro

- 1 app = 1 file
  - l’appimage contiene tutto ciò di cui l’app ha bisogno
- Non c’è bisogno di installare nulla
- Download → Rendi eseguibile → Esegui
  - Su qualsiasi distribuzione!

AppImage: contro

- Update manuali
  - Come su Windows…
- No sandbox
  - Ma si può integrare con sandbox tipo Firejail
- No ottimizzazione memoria
  - e.g. ogni app KDE caricherà in memoria tutte le librerie KDE…
- Stessi problemi di sicurezza del linking statico

Snaps

`[snapcraft.io]`

- 1 app = 1 file = 1 snap
  - Lo snap contiene tutto ciò di cui l'app ha bisogno
  - Uno snap può anche dipendere da un altro snap
- Snap si possono installare da uno "store"
  - `sudo snap install vlc`
  - Snap è disponibile su quasi tutte le distro
- Creato da Canonical, originariamente per Ubuntu Phone e Ubuntu Cloud (Snappy)
  - Molto simile al Play store di Android

snappy

Snaps: pro

- Update automatici
- Sandbox
  - Ma solo con AppArmor (SELinux non supportato)
- Permessi ben definiti
  - Simili ai permessi delle app Android

Snaps: contro

- Permessi di root necessari per installare uno snap
- Per creare uno snap serve Ubuntu
- Al momento l’unico store è l’Ubuntu store
  - Necessario un account Ubuntu per pubblicare uno snap

Flatpak

[flatpak.org]

- Concorrente di Snaps
- Aggiunge il concetto di “Runtime”
  - Insieme di librerie che più app possono condividere
- Progetto freedesktop.org, spinto da Gnome (e finanziato da Red Hat)
- In fase di valutazione anche per KDE

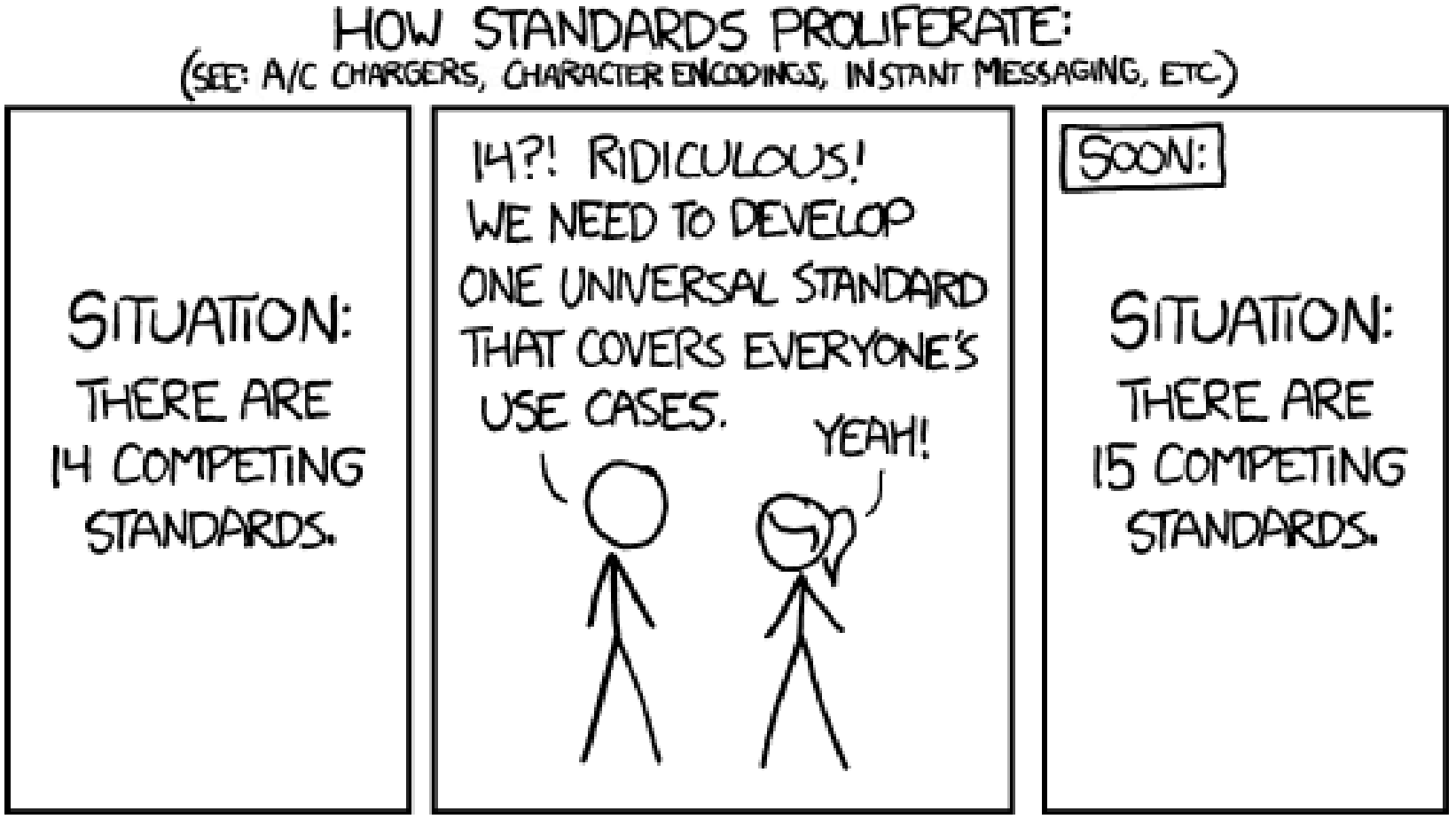
Flatpak: pro

- Installazione senza permessi di root
- Sandbox (tramite systemd)
- Update automatici
- Può essere usato come backend
  - Esempio: Gnome Software
- Appstream Metadata

Flatpak: contro

- App possono usare una sola Runtime
  - Se qualcosa manca, va inclusa nel bundle dell’app
- Ancora non molto diffuso al di fuori di Gnome
- Repository vanno aggiunti manualmente

Una scena già vista...



[https://xkcd.com/927]

Un po' di numeri

- 2007: Alexander Larsson (autore di Flatpak) rilascia la prima versione di Glick
- 2013: Ubuntu Phone introduce Click (precursore di Snaps)
- Settembre 2014: Lennart Poettering (autore di systemd) pubblica un famoso post
  - http://0pointer.net/blog/revisiting-how-we-put-together-linux-systems.html
- Fine 2014: Snaps e Flatpak vengono pubblicati
- Giugno 2016: il progetto Limba annuncia la chiusura in favore di Flatpak

Conclusioni

- AppImage ideale se si vuole un modello simile a Windows
- Snaps/Flatpak competono come “package manager universale”
  - Snaps più simile al modello Android (singolo repository, a.k.a. store)
  - Flatpak più adatto per progetti come Gnome o KDE (repository dedicati)

goo.gl/drRFdB

Domande?

Elvis Angelaccio
elvis.angelaccio@kde.org