



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

# Sviluppo software in gruppi di lavoro complessi<sup>1</sup>

Mattia Monga

Dip. di Informatica  
Università degli Studi di Milano, Italia  
[mattia.monga@unimi.it](mailto:mattia.monga@unimi.it)

Anno accademico 2020/21, I semestre

<sup>1</sup> © 2020 M. Monga. Creative Commons Attribuzione — Condividi allo stesso modo 4.0 Internazionale.

<http://creativecommons.org/licenses/by-sa/4.0/deed.it>



Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

## Lezione VIII: *Dependency hell*



# Dove siamo?

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

Come ci si organizza? “*The tar pit*” Sviluppare *software* necessita sforzi collettivi coordinati: gruppi di lavoro complessi con obiettivi in rapida evoluzione e innumerevoli *concern* intrecciati rendono molto difficile la divisione del lavoro

Come si gestiscono i manufatti? La produzione del *software* consiste principalmente nella modifica di *file*: i sistemi di *configuration management* permettono di tenere sotto controllo l’evoluzione delle revisioni



# Collaborare in un gruppo di lavoro complesso

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

La collaborazione ordinata richiede spesso parecchio lavoro aggiuntivo.

- Un caso in “famiglia”: <https://github.com/scipy/scipy/pull/6658>
- Anche un programmatore eccezionalmente dotato come Sebastiano Vigna, deve spendere parecchie energie per *incastrare* il proprio contributo nello sforzo collettivo.
- Le *policy* aziendali (o di “kibbutz”) sono ormai diventate una componente essenziale del lavoro dello sviluppatore



# Dipendenze

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

Qualsiasi applicazione **dipende** da componenti *software* fuori dal controllo del produttore:

- kernel
- device driver
- librerie di sistema
- **librerie di supporto**



# Gnome-calculator

```
$ ldd $(which gnome-calculator)
libgtk-3.so.0 => /lib/x86_64-linux-gnu/libgtk-3.so.0 (0x00007f1bbc404000)
libgdk-3.so.0 => /lib/x86_64-linux-gnu/libgdk-3.so.0 (0x00007f1bbc10d000)
libpango-1.0.so.0 => /lib/x86_64-linux-gnu/libpango-1.0.so.0 (0x00007f1bbbec1000)
libatk-1.0.so.0 => /lib/x86_64-linux-gnu/libatk-1.0.so.0 (0x00007f1bbbc9b000)
libgio-2.0.so.0 => /lib/x86_64-linux-gnu/libgio-2.0.so.0 (0x00007f1bbb904000)
libgobject-2.0.so.0 => /lib/x86_64-linux-gnu/libgobject-2.0.so.0 (0x00007f1bbb6b1000)
libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x00007f1bbb39d000)
libcalculator.so => /usr/lib/x86_64-linux-gnu/gnome-calculator/libcalculator.so (0x00007f1bbb
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f1bba884000)
libpangocairo-1.0.so.0 => /lib/x86_64-linux-gnu/libpangocairo-1.0.so.0 (0x00007f1bba677000)
libcairo.so.2 => /lib/x86_64-linux-gnu/libcairo.so.2 (0x00007f1bba363000)
libgdk_pixbuf-2.0.so.0 => /lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0 (0x00007f1bba140000)
libxml2.so.2 => /lib/x86_64-linux-gnu/libxml2.so.2 (0x00007f1bb9d85000)
libgmodule-2.0.so.0 => /lib/x86_64-linux-gnu/libgmodule-2.0.so.0 (0x00007f1bb9b81000)
...

```

In totale 83 componenti!

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo



# Dipendenze di sviluppo

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

Va un po' meglio con i linguaggi interpretati: alle dipendenze di sistema generalmente sopperisce l'interprete (ma non sempre: con la macchina virtuale Java per esempio può essere piuttosto faticoso utilizzare specifiche librerie grafiche).

- Un'applicazione usa librerie per non 'reinventare la ruota'
- Evitare la sindrome NIH
- Ma anche evitare le dipendenze inutili: <https://redd.it/4bjss2>

Le dipendenze vanno il più possibile esplicitamente documentate e motivate



# Dipendenze e pacchettizzazioni

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

Abbiamo già discusso che distributori come Debian devono gran parte del loro successo alla ricca documentazione delle dipendenze:

- Ogni pacchetto è regolato da un *control* file, che specifica le caratteristiche
- le dipendenze: *Depends*, *Recommends*, *Suggests*, *Enhances*, *Pre-Depends*
- gli script da eseguire per mantenere l'integrità del sistema: *preinst*, *postinst*, *prerm*, *postrm*
- la priorità: *Required*, *Important*, *Standard*, *Optional*, *Extra*



# Il “packaging” nello sviluppo

Svigruppo

Monga

Riassunto

Dipendenze

Il “packaging”  
nello sviluppo

Il problema esiste non solo a livello di *sistema*, ma anche di singola *applicazione*.  
**(DLL hell)**

- Riproducibilità
- Ambienti di “scripting” per i quali non sono possibili compilazioni “statiche”
- Gestione di installazioni concorrenti di diverse versioni



# Python

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

Esaminiamo il caso di Python, ma considerazioni analoghe valgono ormai per moltissime piattaforme di sviluppo (`npm`, `stack`, ...).

Onnipresenti poi i sistemi di distribuzione centralizzata:

- PHP Pear
- CPAN Perl
- CTAN T<sub>E</sub>X
- MELPA Emacs <sup>1</sup>
- ...



# Python: documentazione delle dipendenze

Python fornisce un meccanismo standard per documentare le dipendenze di un'applicazione: `setup.py`

```
from setuptools import setup
```

```
setup(  
    name="MyLibrary",  
    version="1.0",  
    install_requires=[  
        "requests",  
        "bcrypt",  
    ],  
    # ...  
)
```

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo



# Python: distribuzione centralizzata

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

Esistono poi dei punti di distribuzione centralizzata: per esempio PYPI (Python Package Index <https://pypi.python.org/pypi>)

E naturalmente un package manager: pip install requests



# Python: virtualenv

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

Ma sempre più spesso non vogliamo installazioni “system-wide”, ma “user-wide” o addirittura “application-specific”.

```
$ cd ~/usr/local/src/app/  
$ virtualenv env  
New python executable in env/bin/python  
Installing setuptools.....done.  
Installing pip.....done.  
$ pip ....  
$ python ....
```

(Con `source ./env/activate` si può semplificare la chiamata dei programmi)



# Python: riproducibilità

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

```
$ pip install pippo  
$ pip freeze > requirements.txt  
$ pip install -r requirements.txt
```



# Virtualenv: problemi

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

- Può non essere banale tenere aggiornato un virtualenv  $\rightsquigarrow$  pipenv
- *Source distribution vs. Wheel (egg)*
- Moltissime duplicazioni  $\rightsquigarrow$  virtualenvwrapper
- Sistemi più generali, cross-platform: CONDA



# Dependency hell (cont.)

Svigruppo

Monga

Riassunto

Dipendenze

Il "packaging"  
nello sviluppo

## Versionamento semantico <http://semver.org/spec/v2.0.0.html>

- Numero di versione con tre token MAJOR.MINOR.PATCH
- MAJOR cambia quando ci sono cambiamenti incompatibili nelle API
- MINOR cambia con nuove funzionalità (ma *backwards-compatible*)
- PATCH solo bugfix