

# Cours Logiciels Libres

## Écosystèmes de logiciels libres

Théo Zimmermann

Télécom Paris, Institut Polytechnique de Paris

Vendredi 17 mars 2023



# Écosystèmes logiciels

*Une collection de projets logiciels qui sont développés et évoluent conjointement dans le même environnement.*

(thèse de Mircea Lungu, 2009)

- Le terme provient de l'écologie, en passant par les sciences économiques et de gestion et le monde de l'entreprise.
- Les **écosystèmes de logiciels libres** existent depuis presque aussi longtemps que le logiciel libre :
  - Le système TeX ou des langages de programmation comme Perl ont accumulé de nombreuses **extensions/bibliothèques libres** dès les années 80.
  - À partir des années 90, l'arrivée des premières "archives complètes" puis des gestionnaires de paquets simplifie et accélère le partage.

# Définitions

- **Paquet** : un composant logiciel qui est “empaqueté” de manière à rendre son installation facile.
- **Gestionnaire de paquets** : un logiciel qui sert à installer et supprimer des composants logiciels de façon cohérente.
  - Une innovation cruciale des gestionnaires de paquets “modernes” est la **gestion automatique des dépendances**.
  - *Exemples* : pip (Python), Maven (Java), opam (OCaml), mais aussi APT (Debian / Ubuntu), RPM (RedHat / Fedora).
- **Registre de paquets** (ou **archive** ou **dépôt** de paquets) : un serveur qui centralise les paquets pour faciliter leur distribution.
  - *Exemples* : PyPI (Python), MavenCentral (Java), opam-repository (OCaml), opam-coq-archive (Coq).
- **Index de paquets** : interface (web) par-dessus un registre de paquets permettant de rechercher les paquets disponibles.

# Histoire des gestionnaires de paquets

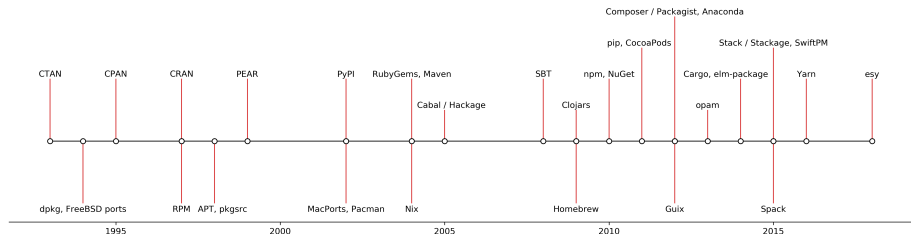


Figure 1: En haut, les gestionnaires de paquets spécifiques à une application, en bas ceux qui permettent de gérer un système d'exploitation

# Distributions Linux

- Les distributions Linux mettent à la disposition de leurs utilisateurs une **collection de logiciels** à travers un gestionnaire de paquets.
- Les mainteneurs de distributions créent des paquets pour les logiciels populaires.
- Ces mainteneurs de paquets sont **indépendants** des mainteneurs des logiciels.

## Écosystèmes de paquets

- En général, on parle d'écosystème de paquets pour désigner les paquets disponibles à travers un gestionnaire et un registre de paquets qui sont **spécifique à une application** (typiquement à un langage de programmation).
- Les paquets sont en général créés par les mainteneurs des logiciels.
- La plupart des paquets dépendent eux-mêmes d'autres paquets (disponibles dans le même écosystème). Par conséquent, il y a un **besoin de coordination** au niveau de l'écosystème.

# Coordination au niveau des écosystèmes

Les mécanismes de base (dans tous les écosystèmes) :

- Communication **descendante** (des mainteneurs de paquets vers ceux qui en dépendent) :
  - Notes de version.
  - Numéro de version (*semver*).
- Communication **ascendante** :
  - Rapports de bug.

# Exemples d'écosystèmes de paquets

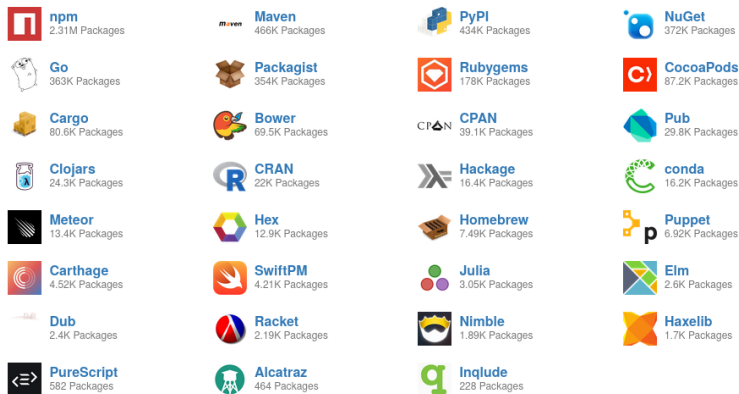


Figure 2: Les registres de paquets indexés par <https://libraries.io>

Les écosystèmes populaires peuvent atteindre plusieurs **centaines de milliers de paquets**. En comparaison, les distributions Linux en comportent quelques milliers à quelques dizaines de milliers.



## Spécifier les dépendances

Exemple package.json (npm) :

```
{
  "name": "mon_paquet",
  "version": "1.0.2",
  "description": "Voilà ce que fait mon paquet.",
  "main": "index.js",
  "license": "MIT",
  "dependencies": {
    "super-bibliotheque": "^1.0.2",
    "autre-bibliotheque": "~2.1.0",
    "encore-une-autre": "0.8"
  }
}
```

## Spécifier les dépendances

Exemple `projet.opam` (opam) :

```
opam-version: "2.0"
synopsis: "Description courte"
description: "Description longue"
maintainer: "moi@mondomaine.me"
authors: ["Moi"]
homepage: "https://github.com/Moi/MonProjet"
depends: [
  "ocaml"
  "dune"    { >= "2.5"    }
  "ppxlib"  { >= "0.22.0" }
  "base"    { >= "v0.14.1" & < "v0.15.0" }
]
...
```

## Dépendances : risques et bénéfiques

- Réutiliser des bibliothèques existantes permet d'être **plus productif**. C'est pourquoi la plupart des projets logiciels en utilisent (ne dépendent pas seulement de la bibliothèque standard).
- Mais ajouter des dépendances comporte aussi des risques :
  - Licences des bibliothèques : doivent être comprises et respectées.
  - Le paquet sera-t-il **mis à jour** pour régler les bugs / les vulnérabilités / s'adapter à l'évolution de l'environnement ?
  - Des vulnérabilités pourraient-elles être introduites par accident / volontairement dans de  **futures versions**  ?
- Il est essentiel de savoir **de quoi** et **de qui** on dépend.

## Dépendances transitives : un risque caché

- Les dépendances ont elles-mêmes des dépendances : ce sont les **dépendances transitives**.
- Il faut aussi en respecter la licence.
- Elles peuvent aussi créer des problèmes si elles sont mal maintenues (vulnérabilités, incompatibilités avec l'évolution d'autres bibliothèques, etc.).

## Exemples de situations réelles

### How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

[Chris Williams, Editor in Chief](#)

Wed 23 Mar 2016 // 01:24 UTC

### A post-mortem of the malicious event-stream backdoor



**Danny Grander, Liran Tal**

December 6, 2018

### Ruby off the Rails: Code library yanked over license blunder, sparks chaos for half a million projects

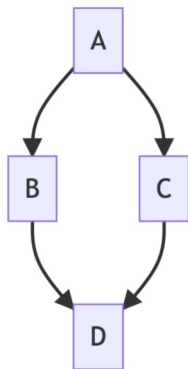
Devs scramble for replacement mimetype data package

[Thomas Claburn in San Francisco](#)

Thu 25 Mar 2021 // 08:24 UTC

## “Dependency hell”

- Le projet A dépend des bibliothèques B v1 et C v1.
- B v1 et C v1 dépendent toutes les deux de la bibliothèque D v1.
- D est mise à jour (v2).
- B est mise à jour (v2) pour dépendre de D v2.
- C n'est pas mise à jour.
- Le projet A **ne peut pas mettre à jour** B tant que C n'est pas mise à jour pour être compatible avec D v2.



# Tester la compatibilité avec les dépendances inverses

Le projet A dépend de la bibliothèque B :

- B est une dépendance de A
- A est une **dépendance inverse** de B.

Les mainteneurs de bibliothèques très utilisées peuvent choisir :

- d'**éviter** tout changement cassant la compatibilité ;
- d'**évaluer** les changements cassant la compatibilité **par le test** avec des dépendances inverses ;
- de **réparer** (certains) paquets cassés par les changements.

# Mécanismes sociaux de coordination

Certains écosystèmes cherchent à éviter le “dependency hell” en favorisant la coordination et la mise à jour des paquets pour être compatibles avec les **dernières versions** de leurs dépendances.

Exemples :

- Stackage (Haskell),
- CRAN (R),
- Plateforme Coq.



## Plateforme Coq

- Une “distribution” de Coq et de **paquets standards**.
- La version de chaque paquet est fixée. Des tests vérifient que tous les paquets inclus sont **compatibles** entre eux.
- À chaque nouveau cycle de préparation de la Plateforme, les mainteneurs des paquets sont **informés** pour qu'ils / elles indiquent quelle version doit être incluse.
- Éventuellement, les mainteneurs produisent une **nouvelle version** (par exemple si la dernière n'était pas compatible avec des mises à jour dans des dépendances).
- Les utilisateurs savent que les versions distribuées dans la Plateforme sont une bonne base sur laquelle s'appuyer.

## Solutions aux problèmes de maintenance

Lorsqu'un logiciel open source utilisé par de nombreuses entreprises ou pour des applications critiques est "mal maintenu" (manque de ressources), les utilisateurs (entreprises) peuvent contribuer à **fournir des ressources** :

- **financières** (dons à une fondation responsable du logiciel),
- **humaines** (développeurs pour participer à la maintenance).

La deuxième solution est plus efficace que la première car tant que les ressources financières n'atteignent pas de quoi **payer un salaire**, cela ne permettra pas d'ajouter plus de monde pour la maintenance du projet.

## Solutions aux problèmes d'abandon

- Un **projet open source abandonné** peut toujours être forké par ses utilisateurs pour **poursuivre sa maintenance**.
- Si le projet est suffisamment important et attire assez de volontaires, ça peut lui donner une **nouvelle jeunesse** (exemple : le serveur Apache).
- Mais si le projet abandonné par son auteur initial est forké par une seule personne, comment s'assurer qu'il ne sera pas **abandonné à nouveau** ?

# Organisations communautaires de maintenance de paquets

- En rassemblant les forks de projets abandonnés dans une organisation communautaire, il est possible de plus facilement assurer leur avenir, car **le mainteneur peut plus facilement être remplacé**.
- Ce type d'organisations a d'autres avantages :
  - Les auteurs de projets qui ne sont plus disponibles pour les maintenir peuvent les **léguer** à l'organisation.
  - Des membres de la communauté peuvent plus facilement aider un grand nombre de projets sur des **tâches de routine** et l'application de **bonnes pratiques**.