



Composer, l'outil moderne de gestion de dépendances

vendredi 30 novembre 12

C'est à quel sujet ?

- Dépendances, kezako ?
- Mes 1ers pas avec Composer
- Aller un peu plus loin

vendredi 30 novembre 12

Plan de la conférence

Loi des 2 pieds : si la conférence ne t'apporte rien, tu as le droit de changer de salle

Ce que nous allons voir :

- 1/ qu'est-ce qu'une dépendance, et quels problèmes sont liés
- 2/ Commencer à utiliser composer
- 3/ Pousser un peu l'utilisation de composer

La cible de cette conférence est les personnes qui ne connaissent pas/peu composer et qui veulent en savoir plus.

Votre serviteur



Mikael RANDY
@mikaelrandy



vendredi 30 novembre 12

Mikael Randy

Développeur spécialisé PHP, principalement les technologies Symfony (Symfony 1.x, Symfony2, Silex)

6 ans en tant qu'architecte logiciel dans une SSII lyonnaise

Administrateur sur PHPFrance
Coordinateur de l'AFUP Lyon

The Ebook Alternative

‘tea’
the ebook alternative



vendredi 30 novembre 12

The Ebook Alternative
Service de diffusion du livre numérique (B2B et B2C) qui se veut plus ouvert que les solutions existantes.

Basé sur Lyon
PHP/Silex, Ruby, Javascript

The Ebook Alternative



vendredi 30 novembre 12

The Ebook Alternative
CDI / Stages

Dépendance

Kézako ?

vendredi 30 novembre 12

Qu'est-ce qu'une dépendance.

Une dépendance est une librairie/bout de code/programme nécessaire au bon fonctionnement de mon programme

Commençons par planter le décor, avec un cas pratique

Problème des dépendances

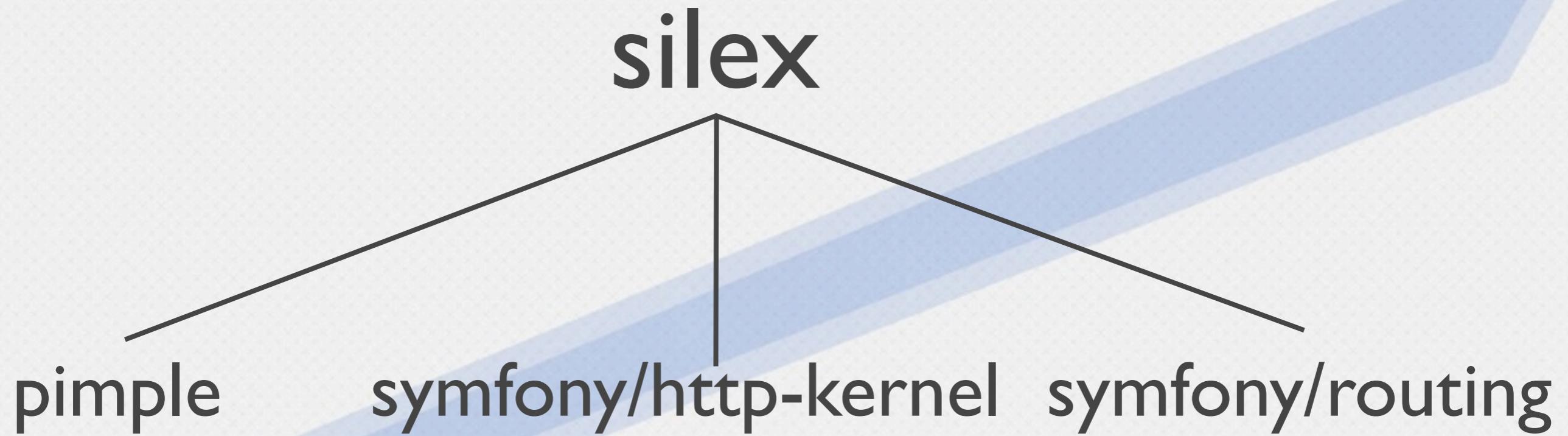
silex

vendredi 30 novembre 12

Démarrage d'un nouveau projet utilisant Silex

La seule dépendance est : Silex

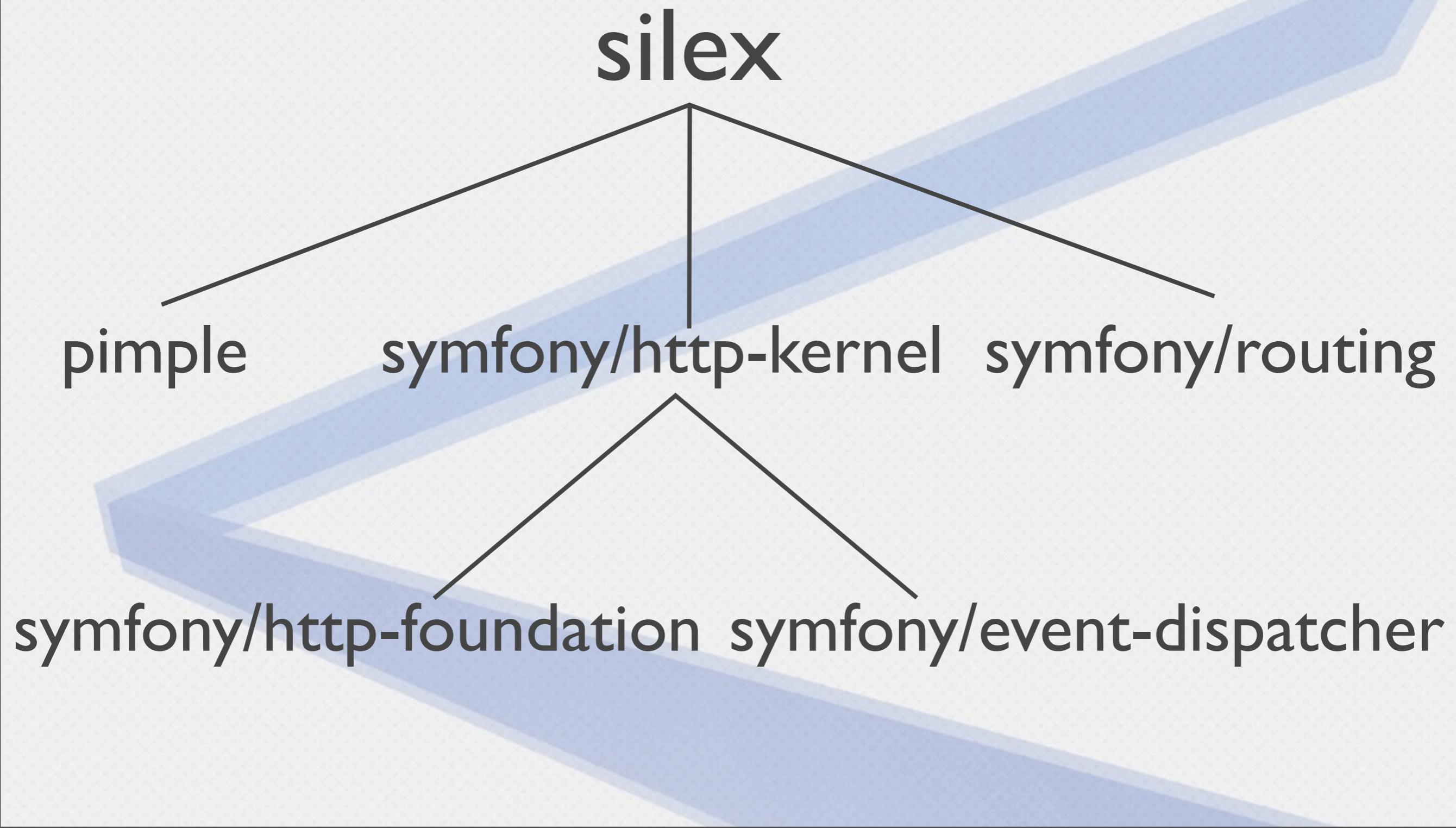
Problème des dépendances



vendredi 30 novembre 12

Zut, ma dépendance a des dépendances
Pour que mon projet fonctionne, il faut les installer

Problème des dépendances



vendredi 30 novembre 12

Les dépendances de ma dépendance ont des dépendances

A l'origine, je voulais juste installer Silex ! Et je me retrouve à installer 6 dépendances ...

Maintenance



vendredi 30 novembre 12

Quand on retire une dépendance, ou qu'on en met à jour

Maintenance



vendredi 30 novembre 12

Jusqu'au jour où ... tout s'écroule

- * Les versions ne sont plus compatibles
- * Est-ce que cette dépendance est utilisée ?

Comment faisons-nous ?

- Téléchargement et installation à la main
- Submodules des gestionnaires de versions

vendredi 30 novembre 12

A la main
-> archaïque

Submodules (svn externals)
-> Un peu mieux
-> Un minimum d'automatisation

Mais dans tous les cas
* Il faut se soucier des dépendances de dépendances
* Comment s'assurer que la même version est toujours installée ?

Et donc, composer ?



vendredi 30 novembre 12

Composer, c'est cool

Et donc, composer ?



vendredi 30 novembre 12

Avec composer, on est plus zen

(cette slide ne sert à rien, mais je m'étais promis de mettre un lolcat dans ma présentation)

Installer composer

```
$ curl -s https://getcomposer.org/installer | php
```

```
$ php composer.phar
```

```
$ php composer.phar self-update
```

vendredi 30 novembre 12

Une seule ligne de commande pour installer composer

* Par défaut, il s'installe localement. Autant l'installer à la racine du projet

* Il est possible de l'installer globalement (mv composer.phar /usr/bin/)

Une fois installé, l'utilisation est aussi simple que lancer l'exécution du fichier PHAR

La mise à jour est aussi simple

Un fichier nommé plaisir

composer.json

```
{  
    "require": {  
        "silex/silex": "1.0.x-dev"  
    }  
}
```

vendredi 30 novembre 12

On a déjà notre fichier composer.phar à la racine du projet
Rappel : la dépendance que nous voulions à l'origine : Silex

«require» : indique que nous avons besoin de cette dépendance
«silex/silex» : indique le package que nous voulons télécharger. «vendorName/packageName»
«1.0.x-dev» : indique la version

Les packages

- « dépendance »
- Archive / Repository Git / Dépôt SVN
- Version
 - 1.0.x-dev
 - 1.0.0
 - $\geq 1.0, < 2.0$

vendredi 30 novembre 12

Un package est ce que j'appelle une dépendance depuis le début de la conférence
Package : lien entre un nom et la localisation d'une librairie (source)

* Archive à télécharger (adresse vers l'archive)

* Repository Git/SVN

Version

* Pour une archive, permet de choisir le fichier exacte

* Pour un repository Git, permet de choisir la branche / un tag

* *-dev convention quand on pointe sur une branche

* version exacte pointe sur un tag

* Il est possible d'utiliser des plages de valeurs

L'installation des dépendances

```
$ php composer.phar install
```

```
Loading composer repositories with package information
Installing dependencies
- Installing symfony/routing (v2.1.3)

- Installing symfony/http-foundation (v2.1.3)

- Installing symfony/event-dispatcher (v2.1.3)

- Installing symfony/http-kernel (v2.1.3)

- Installing pimple/pimple (1.0.0)

- Installing silex/silex (dev-master 0829298)
  Cloning 0829298b374167464842ad2c52e574dc6140dccf
```

vendredi 30 novembre 12

Une seule commande, et tout se déroule

- * Vérification du fichier
- * Résolution des dépendances
 - * Recherche des packages listés dans le composer.json, et recherche récursive des dépendances des packages
 - * Installation de tout les packages nécessaires

Tips

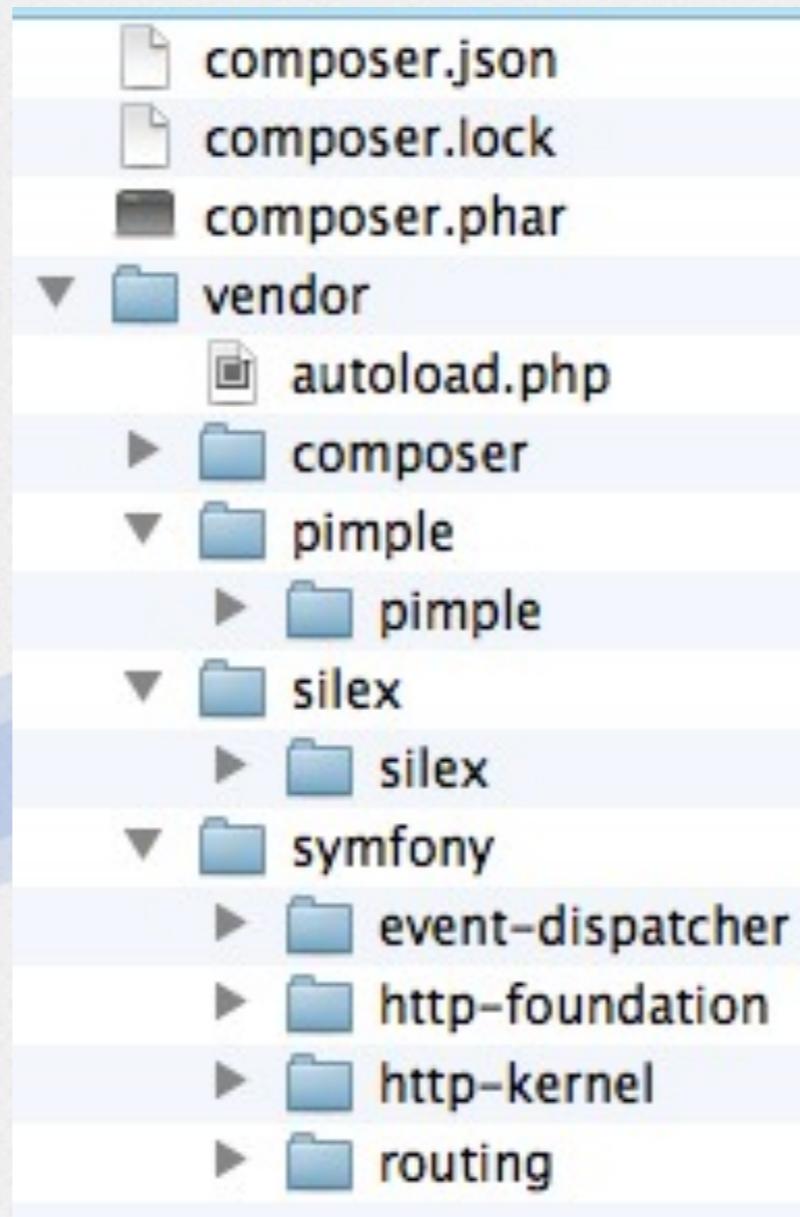
```
$ php composer.phar install --dry-run
```

vendredi 30 novembre 12

Tout ce que fait un « install » mais sans le téléchargement final

- * Vérification du fichier
- * Résolution récursive des dépendances

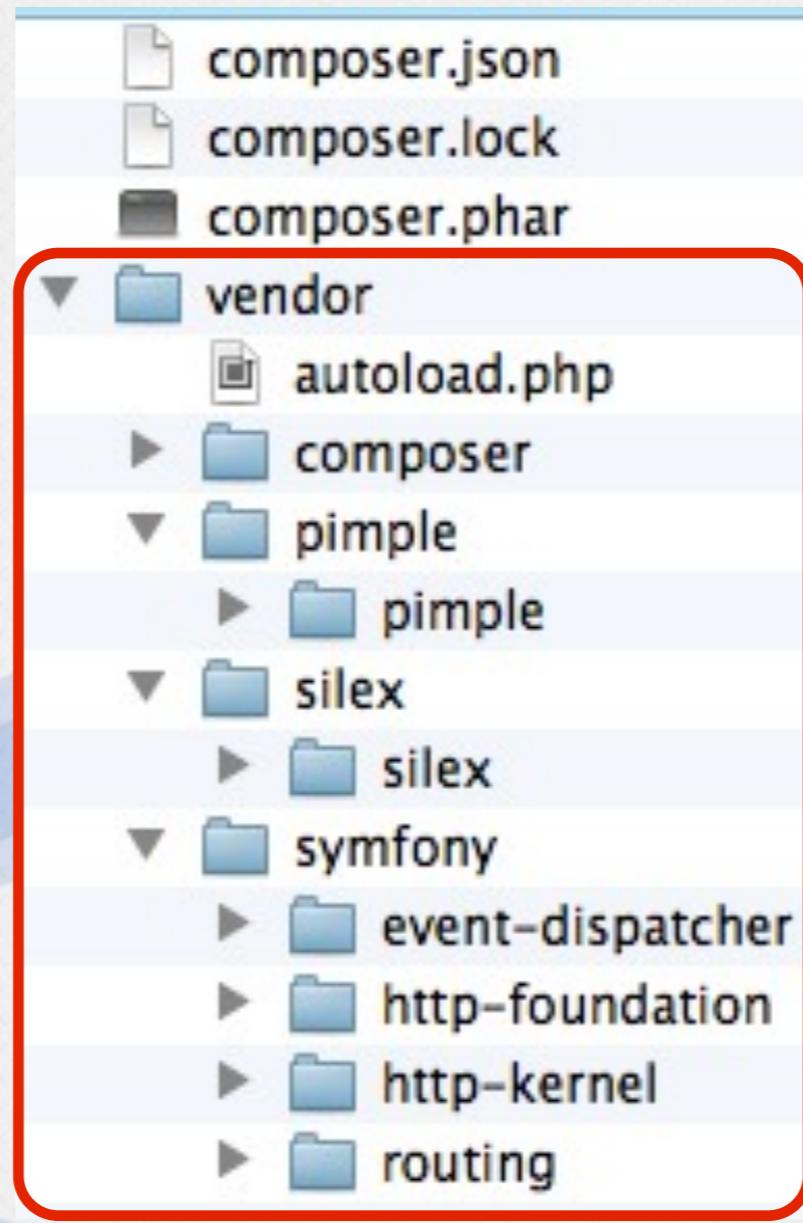
Mes dépendances



vendredi 30 novembre 12

Résultat de l'installation

Mes dépendances

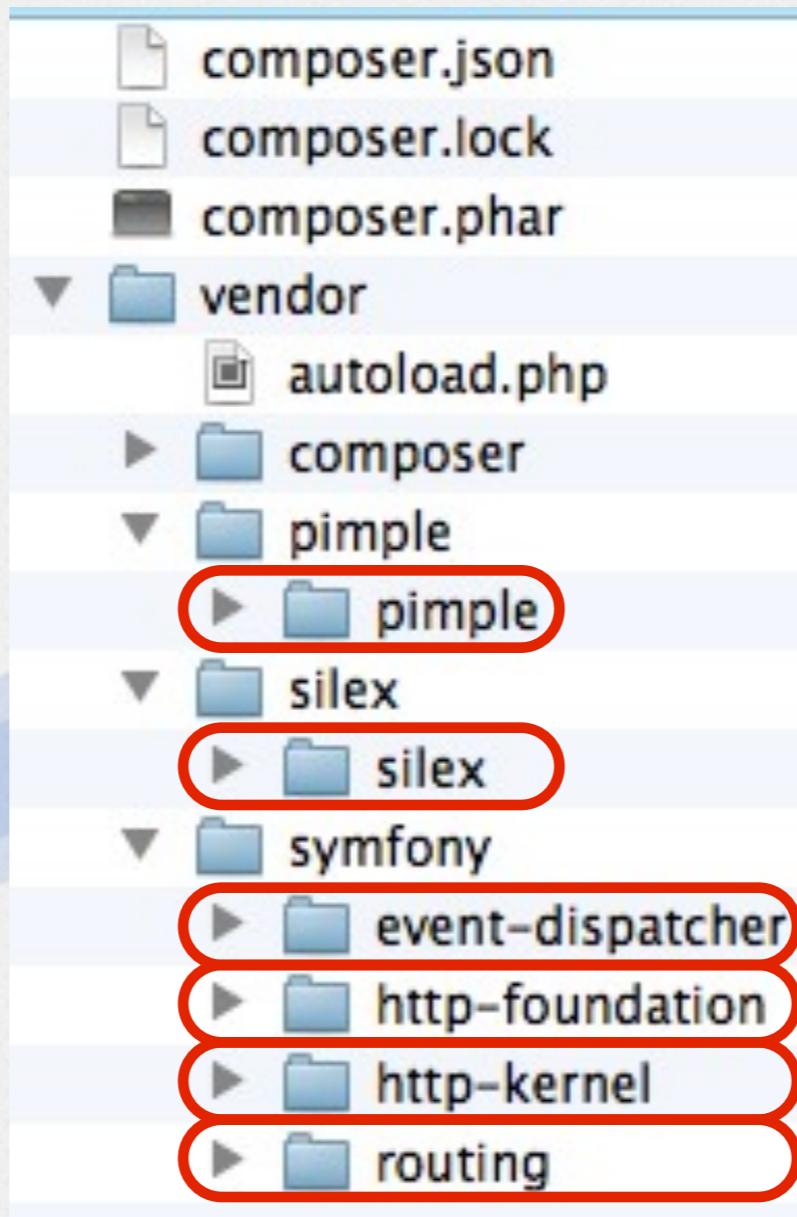


vendredi 30 novembre 12

Dossier vendor

* Toutes les dépendances sont installées ici

Mes dépendances

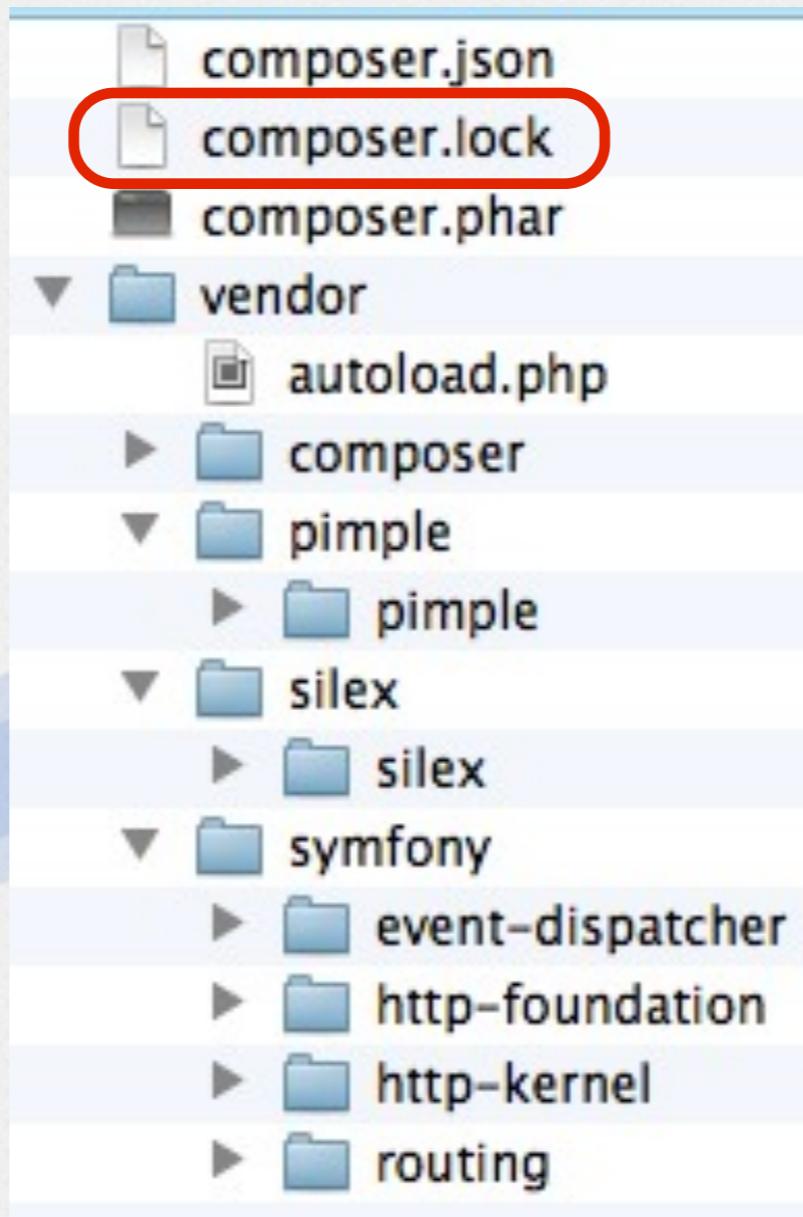


vendredi 30 novembre 12

Dossier vendor

* Chaque package est installé dans une architecture de la forme «`vendorName/packageName`»

Mes dépendances



vendredi 30 novembre 12

Nouveau fichier «composer.lock»

- * Résultat de la résolution des dépendances
- * Image exacte de ce qui a été installé

composer.lock

```
{  
    "name": "silex/silex",  
    "version": "dev-master",  
    "source": {  
        "type": "git",  
        "url": "git://github.com/fabpot/Silex.git",  
        "reference": "46ac91f06d2b5b37bde03d7f185b82645f10f691"  
    },  
    "dist": {  
        "type": "zip",  
        "url": "https://github.com/fabpot/Silex/archive/46ac91f06d2b5b37bde03d7f185b82645f10f691.zip",  
        "reference": "46ac91f06d2b5b37bde03d7f185b82645f10f691",  
        "shasum": ""  
    },  
    "require": {  
        "php": ">=5.3.3",  
        "pimple/pimple": "1.*",  
        "symfony/event-dispatcher": ">=2.1,<2.3-dev",  
        "symfony/http-foundation": ">=2.1,<2.3-dev",  
        "symfony/http-kernel": ">=2.1,<2.3-dev",  
        "symfony/routing": ">=2.1,<2.3-dev"  
    },  
}
```

vendredi 30 novembre 12

Extrait du fichier composer.lock de l'exemple courant (366 lignes pour le fichier composer.json d'exemple)
Peu lisible, mais pas à lire

l'installation d'un existant

```
$ php composer.phar install
```

vendredi 30 novembre 12

Lors du processus d'installation, si un fichier composer.lock est présent, composer l'utilise pour savoir quoi télécharger

S'il ne le trouve pas, il réalise la résolution des dépendances, puis génère le fichier composer.lock

Et les mises à jour alors ?

```
$ php composer.phar update
```

```
Loading composer repositories with package information
Updating dependencies
- Updating silex/silex dev-master (0829298 => 46ac91f)
  Checking out 46ac91f06d2b5b37bde03d7f185b82645f10f691

Writing lock file
Generating autoload files
```

vendredi 30 novembre 12

Le composer update ignore le fichier composer.lock et relance la résolution complète, puis met à jour le fichier composer.lock

Versionnement

```
$ cat .gitignore  
vendor/
```

```
$ git add composer.*
```

vendredi 30 novembre 12

Comment versionner mes dépendances ?

- * `composer.phar` : exécutable d'installer
- * `composer.json` : description des dépendances
 - * pour connaître les dépendances d'un point de vue macro
- * `composer.lock` : détail exact de l'installation
 - * pour connaître les dépendances d'un point de vue micro
 - * nécessaire pour partager l'état exact d'un projet

Avec cette configuration, toute personne qui récupère votre projet, exécute un «`php composer.phar install`» se retrouve dans la même version que vous

Trouver mes dépendances



Packagist
The PHP package archivist.

vendredi 30 novembre 12

Packagist est la source officielle (repository par défaut)
Par défaut, la résolution d'une dépendance va chercher des informations sur cette source
Tout le monde peut déposer son package
Comment créer un package ? (slide suivant)

Transformer un projet en package

- Dis, papa, c'est quoi un package ?
- Mon fils, c'est un projet avec un nom

vendredi 30 novembre 12

Un package est un projet qu'il est possible de retrouver par son nom

Un package, c'est :

- * du code source
- * qui rend un service
- * qui est identifiable

Transformer un projet en package

```
{  
    "name": "PhpTourNantes/presentation",  
    "require": {  
        "silex/silex": "1.0.x-dev"  
    }  
}
```

```
{  
    "require": {  
        "PhpTourNantes/presentation"  
    }  
}
```

vendredi 30 novembre 12

Un package est un projet qu'il est possible de retrouver par son nom

Un package, c'est :

- * du code source
- * qui rend un service
- * qui est identifiable

Une fois uploadé sur packagist, il est possible de l'inclure dans un projet, comme dépendance

Transformer un projet en package

```
{  
    "name": "PhpTourNantes/presentation"  
    "require": {  
        "silex/silex": "1.0.x-dev"  
    }  
}
```

```
{  
    "require": {  
        "PhpTourNantes/presentation"  
    }  
}
```

vendredi 30 novembre 12

Il suffit d'utiliser le nom donné au package dans le composer.json

Vous voyez, une librairie, c'est un projet avec un nom !

Et mes packages privés ?

github.com/composer/satis

```
{  
    "repositories": [  
        { "type": "vcs", "url": "http://github.com/phptournantes/presentation" }  
    ],  
    "require": {  
        "phptournantes/presentation": "*",  
    }  
}
```

```
{  
    "repositories": [ { "type": "composer", "url": "http://packages.example.org/" } ],  
    "require": {  
        "PhpTourNantes/presentation"  
    }  
}
```

vendredi 30 novembre 12

Satis

- * serveur de package statique, installable localement
- * sert à faire le lien entre un nom de package et son repository

Pour le faire fonctionner, il suffit de :

- * cloner le projet
- * paramétriser son composer.json
 - * Url des repositories où trouver les sources
 - * les packages qui sont gérés par satis, et dans quelles versions

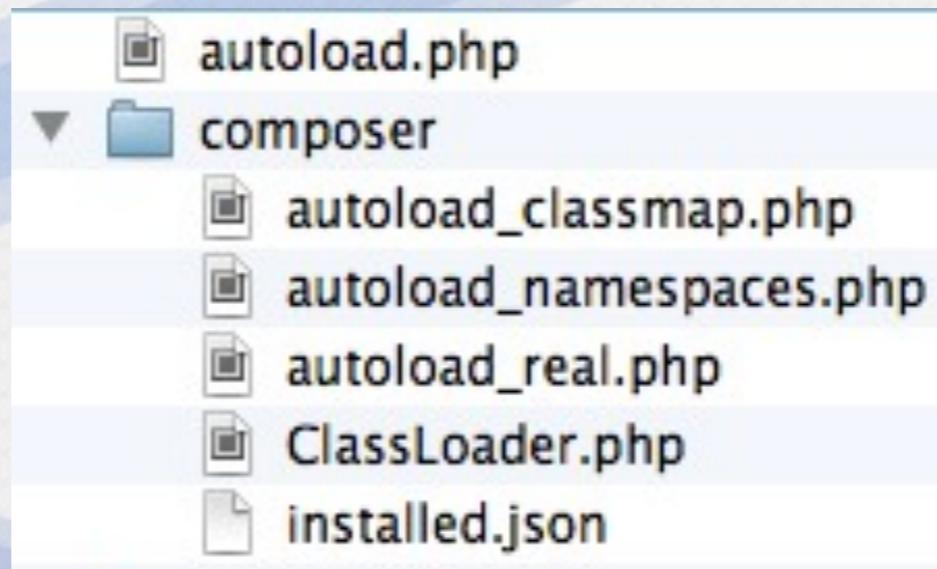
Il suffit ensuite, dans le composer.json, de préciser le repository que l'on veut consulter en priorité

- * Si plusieurs sont définis, ils sont pris dans l'ordre
- * packagist est toujours ajouté à la fin (donc, possible de surcharger des packages packagist)

Utiliser ses dépendances

```
<?php
```

```
require __DIR__.' /vendor/autoload.php '
```



vendredi 30 novembre 12

Il suffit d'inclure un unique fichier central pour disposer des dépendances

le fichier «vendor/autoload.php» s'occupe d'inclure les classes d'autoloading de composer
Ces classes se chargent d'autoloader les classes des dépendances

Il suffit alors d'utiliser les codes définis dans les dépendances, sans se soucier de les charger

Autoloading de mes sources

```
{  
    "name": "PhpTourNantes/presentation",  
    "require": {  
        "silex/silex": "1.0.x-dev"  
    },  
    "autoload": {  
        "psr-0": { "AFUP": "src/AFUP" },  
        "classmap": ["src/foobar", "lib/", "Something.php"],  
        "files": ["foobar/myFunctions.php"]  
    }  
}
```

vendredi 30 novembre 12

Un projet ne se résume pas à ses dépendances. Il y a des sources propres au projet.

Il est possible d'utiliser le principe d'autoloading de Composer pour charger ses propres ressources

Se principe permet un autoloading selon 3 règles :

- * PSR-0
 - * correspondance namespace/arborescence de dossiers
 - * PHP Framework Interop Group : <https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md>
- * Classmap
 - * Construit A PRIORI un mapping classe => chemin vers un fichier
- * Fichier simple
 - * il est possible de demander l'inclusion d'un seul fichier

Tips

```
$ php composer.phar install --optimize-autoloader
```

vendredi 30 novembre 12

Astuce pour gagner un peu en performance en production

Utile en production pour gagner du temps à l'exécution

* le temps de résolution de l'autoloading est fait à l'installation des dépendances, pas à l'exécution

Les mauvais côtés

- Pas de proxy efficace
 - Dépendance aux repositories distants
 - Pas de mutualisation possible

vendredi 30 novembre 12

Pas de proxy efficace

- * Stone / Brocker / Satis
- * Toutes les solutions ont des défauts
 - * Il faut changer la repository
 - * et/ou il faut mettre à jour à la main les dépendances sur le proxy
- * Installation en entreprise/déploiement automatisé
 - * autant de vrai temps de chargement que d'installation

Les mauvais côtés

- En développement constant
 - Pas de version stable

vendredi 30 novembre 12

Composer évolue de jour en jour

- * 30 commits sur la dernière semaine
- * Possibilité de non-compatibilité ascendante lors d'une mise à jour de composer
 - * il est possible qu'un composer.json ou un composer.lock ne fonctionne plus alors qu'il fonctionnait avant
 - * Prendre en compte qu'une mise à jour de composer.phar peut demander une mise à jour des dépendances
 - * Installer composer.phar localement et pas globalement
 - * Difficile de se figer sur une version de composer
 - * Il faut prendre la version du moment

Composer, et plus encore

```
{  
    "name": "PhpTourNantes/presentation",  
    "require": {  
        "silex/silex": "1.0.x-dev",  
        "php": ">=5.4",  
        "ext-gd": "*",  
        "lib-curl": "*"  
    },  
}
```

vendredi 30 novembre 12

Avec composer, il est également possible de vérifier quelques configuration du système

- * Version de PHP
 - * Même gestion des versions que pour les packages
- * Les extensions : «ext-*»
 - * Les versions ne sont pas fonctionnelles, il faut toujours utiliser «*»
- * Les librairies PEAR : «lib-*»
 - * Idem pour les versions

Dépendances de développement

```
{  
    "name": "PhpTourNantes/presentation",  
    "require": {  
        "silex/silex": "1.0.x-dev"  
    },  
    "require-dev": {  
        "atoum/atoum": "dev-master"  
    }  
}
```

```
$ php composer.phar install --dev
```

vendredi 30 novembre 12

Certaines dépendances ne sont pas utiles au fonctionnement, mais sont utiles pour le développement

Exemple : outils de tests (Atoum est une excellente alternative à PHPUnit)

Ces dépendances ne sont installées que lorsque l'installation/la mise à jour est faite avec l'option «--dev»

Hooks

```
{  
    "name": "PhpTourNantes/presentation",  
    "require": {  
        "silex/silex": "1.0.x-dev"  
    },  
    "scripts": {  
        "post-update-cmd": "MyVendor\\ MyClass::postUpdate",  
        "post-package-install": [  
            "MyVendor\\ MyClass::postPackageInstall"  
        ],  
        "post-install-cmd": [  
            "MyVendor\\ MyClass::warmCache",  
            "phpunit -c app/"  
        ]  
    }  
}
```

vendredi 30 novembre 12

Composer autorise l'utilisation de hooks autour de son execution

Le composer.json permet de définir la configuration de ces hooks

- * une ligne de commande
- * un callback PHP
- * L'autoloading de composer sera déjà en place

Conclusion

- Composer, c'est bien !
- Lire la doc
 - <http://getcomposer.org/doc/>
- Suivre le développement
 - <https://github.com/composer/composer>

vendredi 30 novembre 12

Lire la doc permet d'approfondir les sujets que nous venons d'effleurer

Suivre le développement permet de proposer des améliorations, de comprendre ce qui change, ...

Merci à vous

Question ?



@mikaelrandy

joindin <https://joind.in/talk/view/7247>

vendredi 30 novembre 12