Luthaf

Zeste de savoir

Homebrew pour gérer ses logiciels OS X

12 août 2019

Table des matières

Lorsque vous travaillez avec OS X (le système d'exploitation d'Apple), vous installez en général vos logiciels de l'une des manières suivantes :

— En téléchargeant une image `.dmg` et en copiant l'application `Foo.app` n'importe où sur le disque (en général dans `/Applications/` ou `Users/<votre-nom>/Applications/`). C'est la méthode la plus utilisée car la plus simple ;
— En téléchargeant un installeur, qui a comme extension `.pkg` ou `.mpkg`, lequel se charge d'installer pour vous tous les bouts du logiciel là où il le faut. C'est l'option utilisée lorsque les logiciels sont complexes, et ont par exemple besoin de *Frameworks*[1] ou de *Bundles*[2] pour fonctionner.

Toutefois, il existe au moins 2 autres manières d'installer un logiciel :

— Télécharger le code source et le compiler. Cela est surtout utilisé par des projets open source, mais peut être assez compliqué la première fois ;
— Utiliser un gestionnaire de paquets. C'est un logiciel qui se charge d'installer pour vous tout ce que vous lui demandez.

Ce tutoriel présente Homebrew , un des gestionnaires de paquets disponible pour OS X.

1. À qui s'adresse ce tutoriel ?

Aux utilisateurs avancés de OS X, qui ont besoin d'installer des outils complexes, de faire vivre un environnement de développement sans se prendre la tête, et sans passer des heures à chercher sur Internet les différents installateurs de différents projets.

1. Un Framework est un bundle qui contient une bibliothèque dynamique et les fichiers associés (fichiers d'en-tête, …)
2. Un Bundle est un dossier qui obéit à des règles particulières de structure. Par exemple, les applications sont des bundles (vous pouvez en explorer la structure avec un clic droit -> "Afficher le contenu du paquet".

2. Prérequis

Pour suivre ce tutoriel, vous devez savoir vous servir de la ligne de commande sous OS X. Si ce n'est pas le cas, lisez donc n'importe quelle introduction aux systèmes GNU/Linux en vous souvenant que le terminal est situé dans `Applications/Utilitaires/`.

3. Les gestionnaires de paquets

? Qu'est-ce que c'est dis donc un gestionnaire de paquets ?

Un gestionnaire de paquet est comme son nom l'indique un moyen de gérer des paquets. Bon là, on n'est pas beaucoup plus avancé …

Un paquet est une entité qui contient toutes les informations qui permettent d'installer un logiciel sur votre ordinateur. En général, un logiciel a besoin pour fonctionner de bibliothèques généralistes ou d'autres logiciels en plus de son propre code. Ainsi, les interpréteurs Python et Ruby dépendent tous les deux des bibliothèques OpenSSL, du logiciel `readline` et de `pkg-config`. Dans une logique de séparation en paquets, ces bibliothèques sont fournies séparément — par d'autres paquets — et seront partagées entre tous ceux qui en ont besoin (Python, Ruby, …).

La grande force des gestionnaires de paquets par rapport à une compilation depuis les sources repose justement sur cette gestion des dépendances. Prenons par exemple l'utilitaire `wget`. Il s'agit d'une commande UNIX servant à télécharger des ressources en ligne de commande, et non fournie par défaut dans OS X. L'ensemble de ses dépendances est représenté ci-dessous, une flèche exprimant une dépendance :

Figure 3. – Dépendances de wget (source : tutoriel sur MacPort)

On voit bien que même dans un cas simple, la gestion manuelle des dépendances est une prise de tête sans nom : pour un seul logiciel, on a besoin de vérifier 13 dépendances, qui ont elles-même des dépendances, et ainsi de suite. L'utilisation d'un gestionnaire de paquets peut donc simplifier de beaucoup la vie du développeur qui a besoin de bibliothèques diverses ou du bidouilleur qui veut utiliser toute la puissance de la ligne de commande sous OS X.

Je ne parle volontairement pas de ceux qui veulent installer les dernières applications de jeux ou même en général des applications graphiques, car Homebrew n'est pas orienté vers ce type de logiciels. Il existe bien quelques jeux et applications graphiques, mais ils ne forment pas la majorité des paquets disponibles.

Homebrew est donc un gestionnaire de paquets pour OS X et a d'ailleurs choisi comme nom de projet :

> brew - The missing package manager for OS X

Ainsi, que vous ayez besoin de Qt, d'OpenSSL, de bibliothèques MPI ou du logiciel automake, Homebrew peut l'installer pour vous, avec toutes ses dépendances.

4. La concurence : MacPort et Fink

La plupart des distributions GNU/Linux proposent un gestionnaire de paquets officiel, qui est utilisé par défaut : `apt-get` pour Debian et consorts ; `yum` pour les distributions basées sur RedHat ; …

OS X pour sa part ne propose pas de gestionnaire de paquets par défaut. Pour répondre à ce besoin, plusieurs gestionnaires différents ont donc été créés. Parmi eux, je vais en présenter trois, les plus connus : Homebrew, MacPort et Fink.

4.1. Avantages et inconvénients de ces gestionnaires

4.1.1. Homebrew (3 490 paquets, hors dépôts supplémentaires)

Avantages

- Homebrew est simple d'utilisation ;
- Tente de s'intégrer autant que possible à l'écosystème Apple, en utilisant les bibliothèques fournies au lieu de les réinstaller ;
- Installe les logiciels dans `/usr/local/bin`. Il n'est pas nécessaire de changer son `PATH` ;
- Il est *très* facile d'ajouter un nouveau paquet.

Inconvénients

- Il n'y a pas autant de paquets qu'avec les autres gestionnaires. Ce point doit toutefois être modéré par le fait que chacun peut installer son propre dépôt de paquets et le partager. Il y a par exemple 365 autres paquets dans `Homebrew/Science`, et 416 dans `Homebrew/PHP`.

4.1.2. MacPort (20 505 paquets)

Avantages

- Énormément de paquets sont disponibles ;

Inconvénients

- MacPort est plus compliqué à utiliser : il défini son propre shell et défini de nombreuses commandes ;
- Les paquets sont pour la plupart compilés depuis les sources : l'installation peut être (*très*) longue.

4.1.3. Fink (21 622 paquets)

Je ne connais pas vraiment ce gestionnaire, mais il est très ancien. C'est l'un des premiers gestionnaires de paquet pour OS X, qui semble toujours fonctionner sous les versions les plus récentes. Pour en savoir plus, reportez-vous au site officiel .

4.2. Conclusion

Par rapport à MacPort , Homebrew présente donc l'avantage de la simplicité, et de la rapidité d'installation. Là où MacPort installera toutes les bibliothèques depuis les sources, Homebrew va utiliser au maximum les bibliothèques natives, et propose des paquets précompilés.

Enfin, il est très simple d'ajouter des paquets à Homebrew, et les utilisateurs sont fortement encouragés à participer à l'amélioration de la base de paquets.

5. Installer Homebrew

5.1. Les dépendances

Malheureusement, même les gestionnaires de paquets ont des dépendances. Dans le cas de Homebrew, il s'agit principalement des outils en ligne de commande pour développeurs : le compilateur C/C++/Objective-C `clang`, les gestionnaires de version[3] `git` et `svn`, etc.

5.1.1. Outils en ligne de commande : clang, git, ...

Commencez donc par ouvrir un terminal, et installez les outils en ligne de commande d'Apple :

```
xcode-select --install
```
Une fenêtre va s'ouvrir, cliquez sur ||Installer|| et allez au bout de l'installation. Si vous avez comme erreur :
```
Impossible d'installer ce logiciel car il n'est pas disponible actuellement depuis le serveur de mise à jour de logiciels.
```
C'est que les outils en question sont déjà installés — oui, je sais, on peut faire plus explicite comme message d'erreur …

Les deux dépendances suivantes sont optionnelles, bien que très fortement recommandées pour ce qui est de XQuartz.

Extensions pour Java

3. Un gestionnaire de version est un logiciel permettant de gérer différentes versions d'un code source : retours en arrière, différences entre deux états, … `git`, `svn` et `mercurial` sont trois gestionnaires

```
13
14 Si vous voulez compiler les extensions Java de certains logiciels
       (subversion, Cmake, ...), vous aurez besoin d'installer le
       logiciel `Java Developer Update` d'Apple.
15
16 ### XQuartz
17
18 La dernière dépendance de Hombrew est
       [XQuartz](https://xquartz.macosforge.org/landing/), une
       implémentation du serveur de fenêtre X.org pour OS X. Ce
       dernier sera nécessaire pour utiliser certains logiciels avec
       interface graphique conçus pour GNU/Linux.
19
20 ## Installer et tester Homebrew
21
22 Maintenant, vous pouvez oublier tous les problèmes de dépendances
       et installer Homebrew qui s'en chargera pour vous ! Pour ce
       faire, lancez dans un terminal la commande
23 ```
24 ruby -e "$(curl -fsSL
       https://raw.githubusercontent.com/Homebrew/install/master/install)"
25 ```
26 Le script demandera confirmation avant de commencer l'installation.
       Ce script installe la commande principale de Homebrew : `brew`,
       et télécharge la liste des paquets de base.
27
28 Une fois l'installation finie, lancez la commande de debuggage de
       Homebrew :
29 ```
30 brew doctor
31 ```
32 Et lisez le rapport effectué ! Vous pouvez ignorer la plupart des
       avertissements si vous avez déjà installé des logiciels dans
       `/usr/local`. Il se peut que vous ayez besoin d'effectuer
       quelques corrections, mais en général tout se passe bien.
33
34
35 # Utiliser Homebrew
36
37 Maintenant que vous avez installé Homebrew sur votre ordinateur,
       nous allons pouvoir nous en servir. Le point d'entrée de ce
       gestionnaire de paquets est la commande `brew`, qui s'utilise
       avec des sous-commandes, un peu dans le style de `git` :
38
39  - `brew install` pour installer un logiciel ;
40  - `brew remove` pour supprimer un logiciel ;
41  - et tout plein d'autres commandes.
42
43 ## Univers de la bière
44
```

```
45 Homebrew signifie en anglais "Brassé à la maison". C'est
       normalement un terme qui s'applique à la fabrication de la
       bière, et plusieurs termes utilisés par le projet sont liés à
       la bière. En voici un court récapitulatif :
46
47 Anglais   | Français   |   Concept
48 ----------|------------|--------------
49 *formula* | recettes   | Fichier décrivant les étapes et les
       dépendances pour l'installation d'un paquet
50 *cellar*  | cellier    | Dossier regroupant tous les paquets
       installés. En général `/usr/local/Cellar/`.
51 *bottles* | bouteilles | Versions précompilées d'un paquet
52 *taps*    | bondes     | Dépôts de recettes supplémentaires
53
54 ### Recettes et paquets
55
56 J'utiliserai dans ce tutoriel les termes paquets, recettes ou
       logiciel de manière indifférente pour désigner ce que l'on
       cherche à installer. En pratique, une recette peut fournir
       différentes choses :
57
58  - Des binaires, *i.e.* des logiciels en tant quel tels ;
59  - Des fichiers d'en-tête (fichiers `.h`) pour l'utilisation des
        bibliothèques en C, C++ et autres langages ayant un
        préprocesseur ;
60  - De bibliothèques dynamiques (`.dylib` et `.so`) ou statiques
        (`.a`, ...).
61
62 ## Installer un logiciel
63
64 Avant toute chose, lancez la commande `brew update` qui met à jour
       les recettes afin de pouvoir installer les dernières versions.
65
66 Nous prendrons dans cette partie l'exemple d'un développeur
       souhaitant installer Python 3 sur son Mac.
67
68 ### Recherche du logiciel
69
70 Avant d'installer un logiciel, il faut commencer par chercher s'il
       existe une formule correspondant à ce logiciel. Cette recherche
       se fait avec la commande
71 ```
72 brew search <recette>
73 ```
74 Par exemple, chez moi, la recherche de `python` donne :
75 ```
76 $ brew search python
77 boost-python        gst-python010        python                python3
78 homebrew/apache/mod_python
       homebrew/python/python-dbus          homebrew/python/vpython
```

```
```

Il y a deux types de résultats ici : des formules directement
    disponibles, et des formules qu'il est possible d'installer
    depuis une source alternative. Les secondes se reconnaissent
    par la présence du caractère `/` dans leur nom. Ainsi,
    `python3` est directement disponible, et `vpython` est
    disponible dans la source alternative `homebrew/python`.

Une autre manière de rechercher un logiciel est d'utiliser une
    [expression
    régulière](fr.wikipedia.org/wiki/Expression_rationnelle) pour
    chercher dans la liste des recettes. Cela se fait avec la
    commande :
```
brew search /regex/
```
où `regex` est une expression régulière encadrée par deux barres
    obliques `/`.

Enfin, `brew search` affiche la liste de toutes les recettes
    connues par votre installation de Homebrew. Il existe aussi une
    [interfaces web](http://braumeister.org/) pour effectuer des
    recherches dans la liste des recettes.

Installation du logiciel

Python 3 semble disponible pour une installation directe, allons-y
    !
```
brew install python3
```
Là, Homebrew va se lancer dans le téléchargement de toutes les
    dépendances de Python 3 (SQLite, OpenSSL, ...) et installer
    toutes ces dépendances pour vous. Pour chacune des dépendances,
    il affiche un résumé sous la forme :
```
==> Installing python3
==> Downloading
    https://downloads.sf.net/project/machomebrew/Bottles/python3-
    3.4.2_1.mavericks.bottl
######################################################################
    100,0%
==> Pouring python3-3.4.2_1.mavericks.bottle.tar.gz
==> Caveats
Pip has been installed. To update it
  pip3 install --upgrade pip

You can install Python packages with
  pip3 install <package>
```

```
109
110 They will install into the site-package directory
111   /usr/local/lib/python3.4/site-packages
112
113 See: https://github.com/Homebrew/homebrew/wiki/Homebrew-and-Python
114
115 .app bundles were installed.
116 Run `brew linkapps` to symlink these to /Applications.
117 ==> /usr/local/Cellar/python3/3.4.2_1/bin/python3 -m ensurepip
        --upgrade
118 ==> Summary
119 /usr/local/Cellar/python3/3.4.2_1: 3866 files, 67M
120 ```
121
122 La section la plus intéressante de ce rapport est la section
        `Caveats` qui donne toutes les informations utiles pour
        utiliser la recette fraichement installée. Ainsi, pour Python
        3, nous savons qu'il existe des applications (`.app`) qui n'ont
        pas été reliées au dossier `/Applications/`, et que l'on peut
        utiliser `pip3` pour installer des packages Python.
123
124 ### Recettes et bouteilles
125
126 Lors de l'installation, Homebrew nous a affiché ce message :
127 ```
128 ==> Pouring python3-3.4.2_1.mavericks.bottle.tar.gz
129 ```
130 Cela signifie que l'installation effectuée a utilisé une bouteille,
        c'est-à-dire un paquet précompilé. Ce type d'installation est
        le plus rapide qu'une compilation complète, laquelle peut
        prendre jusqu'à plusieurs dizaines de minutes pour un paquet
        (55min pour `gcc` chez moi).
131
132 Les installations depuis les sources peuvent être repérées par des
        messages du type :
133 ```
134 ==> ./configure --prefix=/usr/local/Cellar/<...>
135 ==> make install
136 ```
137
138 ### Plus d'options
139
140 La plupart des recettes offrent des options de compilation. On peut
        voir ces options avec la commande
141 ```
142 brew options <formula>
143 ```
144 Pour notre exemple, les options sont les suivantes :
145 ```
146 $ brew options python3
```

```
--quicktest
        Run `make quicktest` after the build
--universal
        Build a universal binary
--with-brewed-tk
        Use Homebrew's Tk (has optional Cocoa and threads support)
--without-gdbm
        Build without gdbm support
--without-readline
        Build without readline support
--without-sqlite
        Build without sqlite support
--without-xz
        Build without xz support
--HEAD
        install HEAD version
```

Toutes les options permettent de modifier le comportement par défaut. Ainsi, l'option `--without-sqlite` permet de compiler Python 3 sans le support de `sqlite`, ce support est donc activé par défaut. Pour les recettes les plus populaires, les versions par défaut existent sous la forme de bouteilles précompilées. Si jamais vous modifiez les valeurs des options, cela déclenchera une compilation complète depuis les sources.

Homebrew et les liens symboliques

Par défaut, Homebrew installe les logiciels dans son cellier, et créé un [lien symbolique](fr.wikipedia.org/wiki/Lien_symbolique) dans `/usr/local/bin` pour chacun des binaires installés, dans `/usr/local/include` pour chaque fichier d'en-tête, et dans `/usr/local/lib` pour chaque bibliothèque. Cela permet d'activer/désactiver rapidement et facilement un paquet en cas d'incompatibilité lors de l'installation ou de la compilation d'un autre logiciel.

Débuggage

Si jamais une installation se passe mal, les commandes suivantes peuvent être utiles :

 - `brew doctor` vous indique tous les problèmes possibles à considérer ;
 - `brew home <formula>` vous conduit à la page principale du logiciel que vous tentez d'installer.

Si vous ne trouvez toujours pas où est le problème, consultez la [documentation](https://github.com/Homebrew/homebrew/wiki/troubleshooting) à ce sujet.

````
180
181 ## Supprimer un logiciel
182
183 Il n'y a rien de plus simple que de supprimer un paquet avec
        Homebrew :
184 ```
185 brew remove <formula>
186 ```
187 À la place de `remove`, vous pouvez aussi utiliser `uninstall` ou
        `rm`, ce sont des alias strictement équivalents.
188
189 Comme Homebrew utilise par défaut des liens symboliques, il peut
        aussi être intéressant de désinstaller un paquet sans le
        supprimer. Cela se fait avec la commande
190 ```
191 brew unlink <recette>
192 ```
193
194 ### Faites attention aux dépendances
195
196 Lorsque vous supprimez un paquet, Homebrew ne vous dira pas si ces
        paquets sont encore utilisés par d'autres. C'est donc à vous
        d'y faire attention, pour ne pas casser tout votre système.
        Toutefois, Homebrew peut vous y aider, avec les commandes
        suivantes :
197
198  - `brew leaves` affiche une liste des recettes qui peuvent être
         supprimé sans problème ;
199  - `brew missing` affiche une liste des recettes manquantes,
         c'est-à-dire les recettes qui sont une dépendance d'un autre
         paquet installé, mais qui ne sont pas installés ;
200  - `brew uses foo` affiche une liste des recettes qui dépendent de
         `foo`.
201  - `brew deps bar` affiche la liste des dépendances de `bar`.
202
203 Il est possible de regrouper ces informations pour avoir plus
        d'informations. Par exemple, `join <(brew uses foo) <(brew
        list)` affichera la liste des paquets installés qui dépendent
        de `foo`.
204
205 Retenez au moins cela : il est possible de supprimer sans problème
        tout paquet présent dans `brew leaves`, et de trouver les
        paquets installés qui dépendent d'un paquet `foo` avec `join
        <(brew uses <foo>) <(brew list)`.
206
207 ### Supprimer un paquet et toutes ses dépendances
208
````

````
On peut aussi vouloir supprimer un paquet `foo` avec toutes ses dépendances. Une action naïve consisterait à supprimer tous les paquets affichés par `brew deps foo`, puis à supprimer `foo`. Seulement, en faisant cela il est fort possible que vous supprimiez un paquet qui est encore utilisé. Il faut donc trouver l'intersection entre les paquets qui ne sont pas des dépendances, et des dépendances de `foo`. Cela nous sera donné par `join <(brew leaves) <(brew deps foo)`. La commande finale sera donc :
```
brew remove foo
brew remove $(join <(brew leaves) <(brew deps foo))
```

Il faut commencer par `brew remove foo` afin que les dépendances inutiles soit bien détectées par `brew leaves`.

## Définir des commandes personnalisées

Homebrew permet de définir des commandes personnalisées qui pourront être appelées sous la forme `brew ma_commande --option1 <recette>`. Ces commandes personnalisées sont des scripts Shell exécutables (ou des scripts Ruby), qui doivent être mis dans le `$PATH`, et avoir comme nom de fichier `brew-ma_commande`.

Par exemple, créons donc la commande `brew cleandeps` qui se charge de supprimer une recette et toutes les dépendances inutiles. Dans le fichier `/usr/local/bin/brew-cleandeps`, écrivez donc le script suivant
```
#!/bin/bash
formula=$1

brew remove $formula
brew remove $(join <(brew leaves) <(brew deps $formula))
```
`formula=$1` permet ici de stocker le premier argument du script dans la variable `$formula`, ce qui nous permettra d'utiliser notre commande avec `brew cleandeps <recette>`.

Puis rendez le script exécutable :
```
chmod +x /usr/local/bin/brew-cleandeps
```

Vous pouvez maintenant utiliser la commande `brew cleandeps foo` pour supprimer `foo` et toutes ses dépendances.
````

```
238 Pour une utilisation plus avancée, je vous renvoie encore une fois
        à la [documentation
        officielle](https://github.com/Homebrew/homebrew/wiki/External-Commands).
239
240 ## Mettre à jour les paquets
241
242 Pour mettre à jour des paquets, deux commandes sont utiles :
243
244  - `brew outdated` affiche une liste de tous les paquets obsolètes
         ;
245  - `brew upgrade` met à jour tous les paquets et `brew upgrade foo`
         met à jour le paquet `foo`.
246
247 Vous pouvez empêcher la mise à jour de certaines recettes en les
        *épinglant*.`brew pin <recette>` fixe la version de la recette
        et empêche les mises à jour tandis que `brew unpin <recette>`
        supprime le blocage introduit par `brew pin` et permet de
        nouveau les mises à jour. Cela peut être utile si vous avez
        besoin d'une version spécifique d'un paquet.
248
249 ## Conclusion
250
251 Vous devriez à présent être armé pour gérer votre cellier, en
        installant, supprimant ou modifiant avec Homebrew tous les
        logiciels qui vous sont utiles.
252
253 Si vous avez besoin de documentation sur les commandes, vous pouvez
        utiliser `brew commands` pour une liste des commandes, et `man
        brew` pour une doc complète.
254
255 Les prochaines parties sont optionnelles, et vous indiqueront
        comment avoir accès à plus de recettes, et comment écrire vos
        propres recettes.
256
257
258 # Encore plus de recettes : les Taps
259
260 Si jamais vous ne trouvez pas votre bonheur dans les 3000 recettes
        proposées par défaut, ne perdez pas courage ! Il existe
        d'autres dépôts de recettes, plus ou moins liés au projet
        officiel. Ces dépôts sont appelés `taps`, c'est-à-dire robinet
        ou bonde. Les Taps sont des dépôts Github, et sont identifiés
        par un nom sous la forme `User/Tap`.
261
262 ## Ajouter et supprimer des taps
263
264 Lorsque vous effectuez une recherche qui propose des résultats dans
        des taps, vous devez commencer par ajouter ce tap à votre
        cellier avant de pouvoir l'utiliser pour installer des
        recettes.
```

````
265
266 Comme d'habitude, rien n'est plus simple que cet ajout :
267 ```
268 brew tap <user>/<tap>     # Clone le dépôt <user>/homebrew-<tap>
269 brew untap <user>/<tap>   # Supprime la copie locale du dépôt
270 ```
271 Puis :
272 ```
273 brew install ma_formule_qui_provient_d_un_tap
274 ```
275
276 Par exemple, lors de la recherche de `python`, le résulat
        `homebrew/python/vpython` nous avait été proposé. Imaginons que
        l'on veuille installer ce paquet. Les commandes à lancer sont
        simplement :
277 ```
278 brew tap homebrew/python
279 brew install vpython
280 ```
281
282 ## Liste des taps les plus intéressants
283
284  - `Homebrew/versions` pour avoir plusieurs versions d'un seul
         logiciel : Node.js 0.4 et GCC 4.3 sont disponibles ici ;
285  - `Homebrew/games` quelques jeux, dont le célèbre jeu en console
         DwarvesFortress ;
286  - `Homebrew/science` plein de paquets scientifiques : Physique,
         Géosciences (NetCDF), Chimie (LAMMPS), Mathématiques (Maxima),
         Biologie ...
287  - `Homebrew/php` : plusieurs versions de PHP, ainsi que quelques
         extensions ;
288  - `Homebrew/apache` et `Homebrew/nginx` : des extensions pour les
         serveurs web Apache et Nginx.
289
290 Et plusieurs autres, vous pouvez même héberger le votre !
291
292
293 # Écrire sa propre recette
294
295 Pour Homebrew, les recettes sont de simples scripts Ruby, et il est
        très facile d'en écrire de nouvelles. Si jamais vous ne trouvez
        vraiment pas la recette qu'il vous faut, pas même dans les
        taps, vous pouvez donc écrire facilement la vôtre !
296
297 Nous allons faire l'expérience ensemble et écrire une recette
        simple. Les recettes sont des scripts Ruby, et si vous ne
        connaissez pas ce langage, ce n'est pas grave, moi non plus !
        Plus sérieusement, l'interface est suffisament simple pour être
        rapidement apréhendée.
298
````

````
Par contre, pour suivre cette partie vous devrez avoir des bases dans l'utilisation de git et dans la compilation de logiciels en ligne de commande. Nous allons créer un projet minimaliste, en héberger l'archive sur Github et utiliser Homebrew pour l'installer.

## Préparer un projet

### Les sources

Notre première recette sera un petit logiciel ayant tout de même une dépendance. J'ai choisi de faire un `Hello-Word` en appelant l'API Lua depuis le C. La dépendance à satisfaire sera donc `lua`. Heureusement, il existe une [recette homebrew](https://github.com/Homebrew/homebrew/blob/master/Library/Formula/ pour installer Lua !


Dans un fichier `hello.c` copiez le code suivant :
```c
#include "lua.h"
#include "lualib.h"
#include "lauxlib.h"

int main(){
    lua_State * L = luaL_newstate();
    luaL_openlibs(L);

    lua_getglobal(L, "print");
    lua_pushstring(L,"Hello Zeste de Savoir ! From Lua to C with Homebrew !");

    lua_call(L, 1, 0);

    lua_close(L);
    return 0;
}
```
C'est un exemple minimaliste d'utilisation de l'API Lua 5.2 depuis le C, où l'on se contente d'afficher une chaîne de caractères.

À côté de ce fichier, créez un fichier `configure` (sans extension) contenant :
```python
#!/usr/bin/env python

import sys

prefix = None
for arg in sys.argv:
````
````

```
337     if "--prefix" in arg:
338         prefix = arg.split("=")[1]
339 
340 if not prefix:
341     sys.exit(1)
342 
343 with open("Makefile", "w") as f:
344     f.write("""# Auto-generated makefile
345 all:hello-zds
346 
347 hello-zds:hello.c
348 \t$(CC) hello.c -o hello-zds -llua
349 
350 install:hello-zds
351 \tmkdir -p {prefix}/bin/
352 \tcp hello-zds {prefix}/bin/
353 """.format(prefix=prefix))
354 
355 sys.exit(0)
356 ```
357 Et changez les droits d'exécution de ce fichier : `chmod +x
        configure`. Ce fichier est un script Python qui imite
        l'interface du projet `automake` afin de générer un fichier
        `Makefile` qui installera `hello-zds` dans un répertoire
        personnalisé.
358 
359 Remarquez qu'il n'y a pas besoin d'indiquer dans le `Makefile` les
        répertoires des fichiers d'en-tête ou des bibliothèques (les
        options `-I` et `-L`) : en effet, `/usr/local/include` et
        `/usr/local/lib` sont déjà dans les `PATH` des compilateurs et
        des éditeurs de liens.
360 
361 ### L'hébergement
362 
363 Pour notre recette, nous allons utiliser les possibilités
        d'hébergement et de téléchargement de
        [Github](http://github.com/).
364 Créez donc un compte sur ce site si vous n'en avez pas, vous
        pourrez le supprimer à la fin du tutoriel. Une fois inscrit,
        créez un
365 nouveau dépôt (plus en haut à droite, "New repository"), appelez-le
        `Hello-ZDS` et ne créez pas de README.
366 
367 Puis en ligne de commande dans le dossier où il y a les fichiers
        `hello.c` et `Makefile` :
368 ```
369 git init
370 git add hello.c configure
371 git commit -m "First commit"
372 git remote add origin https://github.com/<USERNAME>/Hello-ZDS.git
```

````
git tag 1.0
git push -u --tags origin master
```
N'oubliez pas de remplacer `<USERNAME>` par votre nom d'utilisateur.

Vous pouvez à présent télécharger une archive contenant le code source à l'adresse `https://github.com/<USERNAME>/Hello-ZDS/archive/1.0.tar.gz`.

Si vous n'avez pas envie de faire tout ça, vous pouvez utiliser pour la suite le dépôt que j'héberge, en remplaçant `<USERNAME>` par `Luthaf`.

## Créons notre première recette

L'utilitaire Homebrew pour créer des recettes prend comme argument l'URL à laquelle il peut télécharger une archive contenant les sources du logiciel, et les fichiers de configuration pour la compilation (`configure` ou `CMakeList.txt`).

Pour créer cette recette, nous aurons besoin de la somme SHA1 de l'archive. Cette somme permet de vérifier l'intégrité de l'archive et de vérifier le bon déroulement du téléchargement. Téléchargez donc la version `.tar.gz` du projet sur Github en allant la charger à l'adresse [https://github.com/<USERNAME>/Hello-ZDS/archive/1.0.tar.gz](https://githu
Pour la suite, je suppose que cette archive est dans le dossier `Téléchargements` (ou `Downloads` lorsque vous y accédez via la ligne de commande). La somme SHA1 de l'archive peut être calculée en ligne de commande :
```
openssl sha1 ~/Downloads/Hello-ZDS-1.0.tar.gz
```

Créons la recette de notre projet `Hello-ZDS` :
```
brew create https://github.com/<USERNAME>/Hello-ZDS/archive/1.0.tar.gz
```
Votre éditeur par défaut devrait alors s'ouvrir et vous permettre de modifier la formule ainsi créée. Nous allons supprimer dans cette formule tout ce qui n'est pas utile. Essayez donc de le faire, les formules sont facile à comprendre.

Chez moi, le fichier final est le suivant :

[[secret]]
|
| ```ruby
````
````

```
| require "formula"
|
| class HelloZds < Formula
|    homepage "https://github.com/Luthaf/Hello-ZDS"
|    url "https://github.com/Luthaf/Hello-ZDS/archive/1.0.tar.gz"
|    sha1 "dc4ab680b5ebe572555c93eb15756a8d4e0ecb92"
|
|    depends_on "lua"
|
|    def install
|       system "./configure", "--prefix=#{prefix}"
|       system "make", "install"
|    end
| end
|```
|
| J'ai en particulier supprimé les tests car il n'y en a pas, ainsi
    que les options de `configure` non supportées.

Nous pouvons à présent utiliser notre première formule. Tout
    d'abord testons la formule avec :
```
brew audit hello-zds
```
Homebrew nous indique ici s'il n'est pas content de notre formule.

Puis installons ce superbe logiciel avec :
```
brew install hello-zds
```

Vous devriez alors pouvoir appeler `hello-zds` depuis la ligne de
    commande, et voir votre propre logiciel installé avec Homebrew
    s'exprimer !


### Un peu de ménage

Supprimons donc cette nouvelle recette, car elle ne nous sert plus
    à rien :
```
brew remove hello-zds
rm -f `brew --prefix`/Library/Formula/hello-zds.rb
```

## Créer une recette pour un vrai logiciel

```

```
444 Ce tutoriel ne propose qu'une introduction à l'écriture de vos
        propres recettes. Pour créer une recette pour un vrai logiciel,
        les étapes sont principalement les mêmes. Je vous conseille
        d'aller consulter la [documentation
        officielle](https://github.com/Homebrew/homebrew/wiki/Formula-Cookbook)
        pour en savoir plus.
445
446
447 Si vous avez des difficultés, allez poser des questions sur les
        canaux de discussion liés à Homebrew ou au logiciel que vous
        essayez de porter . Vous aurez beaucoup plus de réponses que
        sur les forums de ce site.
448
449 ## Modifier une recette préexistante
450
451 Vous pouvez modifier toutes les recettes disponibles chez vous,
        afin par exemple d'ajouter ou de modifier des options. Il
        suffit d'utiliser la commande `brew edit <recette>`, votre
        éditeur par défaut devrait s'ouvrir automatiquement.
452
453 Si jamais vous corrigez un bug, n'oubliez pas de faire une
        *pull-request* vers le dépôt original pour en faire profiter
        tout le monde !
454
455
456
457
458
459
460 ---------
461
462 À présent, vous êtes armé pour utiliser Homebrew. Je trouve que
        c'est un outil qui fait gagner énormément de temps dans la vie
        d'un développeur : si je dois compiler un logiciel qui a besoin
        de `gettext` je tape `brew install gettext` et c'est bon !
        D'autre part, il me permet d'installer facilement et rapidement
        des logiciels *juste pour tester*, et de les supprimer
        proprement après.
463
464 Merci à [Marypop](http://zestedesavoir.com/membres/voir/Marypop/)
        et à [Andr0](http://zestedesavoir.com/membres/voir/Andr0/) pour
        leurs relectures et leurs conseils dans l'écriture de ce
        tutoriel.
```