Développer en logiciel libre : empaquetage et diffusion

François PELLEGRINI
Maître de conférences, ENSEIRB
Projet ScAlApplix, INRIA Futurs
pelegrin@labri.fr

Que faire du logiciel produit ?

- Un laboratoire n'a pas vocation à être un éditeur de logiciel
  - Le coût de la finalisation d'un logiciel peut être important
  - La maintenance applicative n'est pas en tant que telle une activité entrant dans les attributions des chercheurs
- Pistes à envisager :
  - Partenariat exclusif
  - Partenariat privilégié
  - Accès libre
- Avec quelles licences ?

Pourquoi développer sous licences libres

- Outils idéaux pour la préservation du patrimoine intellectuel d'un laboratoire de recherche
  - Aucune renonciation à un usage ultérieur du logiciel
- Coût nul
  - L'ajout des mentions de licences dans chaque fichier source suffit pour bénéficier des termes de la licence
  - Le dépôt à des organismes de type APP (pour un coût dérisoire) apporte une preuve d'antériorité
- Mutualisation de l'effort de développement
- Maximisation des possibilités d'irriguer la société et donc de rentabiliser l'investissement de recherche

Cas type : libre téléchargement

- Logiciel dont le marché est très large, constitué d'entités non concurrentes ou dont ce n'est pas le coeur de métier
- Possibilité de créer une communauté d'utilisateurs et de contributeurs
  - Les libertés d'usage augmentent sa taille et donc sa valeur
- Mutualisation de la maintenance et des développements ultérieurs
- Diffusion sous licences libres persistantes ou diffusives
- Cas des bibliothèques et logiciels de service

Cas type : partenaire exclusif

- Logiciel encore au stade de démonstrateur et dont le marché est très étroit
  - Quelques clients potentiels, tous concurrents directs
- Choix d'un partenaire désireux d'investir dans la finalisation du logiciel
  - L'exclusivité est la contrepartie du risque commercial
  - Si pas de risque, aucune nécessité d'exclusivité car spoliation sans contrepartie d'un bien public
- Distribution sous licence libre évanescente :
  - Le partenaire peut redistribuer sous la forme qu'il veut
  - Les sources initiaux restent utilisables par le laboratoire

Cas type : partenaire privilégié

- Logiciel ou bibliothèque métier potentiellement utilisable par une communauté plus large
- Choix d'un partenaire privilégié fournissant un retour sur expérience
- Distribution double sous licences libres :
  - Fourniture au partenaire sous licence évanescente pour lui permettre l'inclusion du logiciel au sein de produits dont les caractéristiques sont cachées aux concurrents
  - Mise en libre accès sous licence diffusive pour possibles contributions de la communauté et réalisation de logiciels analogues mais au code source accessible à tous

Diffusion du logiciel

- Diffuser quoi ?
  - L'état courant du logiciel ?
  - Seulement les versions stables ?
- Diffuser comment ?
  - Accès aux versions stables seulement ?
  - Accès permanent aux versions de travail (« builds ») ?
- Diffuser à qui ?
  - Utilisateurs ?
  - Contributeurs actuels ?
  - Contributeurs potentiels ?

En amont de la diffusion

Checklist avant décollage

Préparation de la diffusion (1)

- La diffusion d'un logiciel est un processus complexe, dans lequel interviennent des aspects :
  - Techniques
  - Juridiques
  - Administratifs
  - Économiques
- Aucun de ces aspects ne doit être ignoré

Préparation de la diffusion (2)

- Les principales étapes préalables à la diffusion d'un logiciel sont :
  - L'identification des auteurs et ayant droits
  - Le choix du modèle économique sous-jacent
    - Traité dans d'autres exposés
  - Le choix de la licence
    - Découle du choix précédent
    - Conditionné par les licences des composants tiers
  - L'accord des ayant droits
  - La réalisation des contenants

Auteurs et ayant-droits (1)

- Un logiciel libre n'est pas « libre de droits » ni « domaine public » !
  - Expressions malheureusement encore présentes dans de nombreux contrats de recherche (projets ANR, etc)
  - « Logiciel libre » est le terme juridiquement le plus approprié
- Des droits sont dévolus à ses auteurs et ayant droits
- La licence associée au logiciel engage ses utilisateurs

http://www.aful.org/droit-auteur/index/jacobsen-v-katzer/

Auteurs et ayant-droits (2)

- Les auteurs sont les personnes ayant écrit l'œuvre
  - Titulaires imprescriptibles des droits moraux
  - Pas de droits moraux dans le régime anglo-saxon
  - Droits moraux limités dans le cas des logiciels
- Les ayant droits sont les personnes titulaires des droits patrimoniaux
  - Les employeurs dans le cas des salariés
  - Ce sont les institutions finançant les salaires des personnels, pas les laboratoires d'accueil
    - CNRS, écoles, INRIA, universités, ...

http://www.sg.cnrs.fr/daj/propriete/logiciels/logiciels.htm

Identification des auteurs

- Les auteurs sont l'ensemble des personnes ayant contribué au logiciel
  - Développeurs, graphistes, etc.
  - Cas classique d'une œuvre collective ou de collaboration
- Ce recensement doit se faire par fichier
  - Mention des auteurs dans l'en-tête des fichiers
  - Granularité nécessaire vis-à-vis des décisions éventuelles de réécriture
- Il doit être exhaustif
  - Prise en compte des stagiaires
- Mieux vaut le faire au fil du développement qu'après

Identification des ayant-droits (1)

- Les ayant droits sont les seuls en droit de décider de la diffusion ou non d'un logiciel qu'ils ont financé
  - Nécessité d'obtenir leur accord avant diffusion
- Les ayant droits sont :
  - Les personnes (le plus souvent morales) ayant financé le développement des auteurs salariés
  - Les stagiaires en ce qui concerne leurs propres créations
    - L'« Arrêt Puech » conclut qu'un stagiaire d'un établissement public n'est ni agent ni salarié de celui-ci
      - Quid des indemnités de stage, qui ne sont pas rémunérations ?
    - Jurisprudence extrêmement problématique, et non figée

http://jeanpaulmartin.canalblog.com/archives/2007/05/12/index.html

Identification des ayant-droits (2)

- La présence de fragments de code dont les ayant droits ne peuvent être déterminés ni contactés doit être circonvenue
  - Re-codage des routines considérées par des personnels salariés (thésards, post-doc, agents)
    - Permet souvent un nettoyage du code
  - Extraction des fonctionnalités correspondantes de la version du logiciel vouée à diffusion
    - Externalisation de ces fonctionnalités dans une bibliothèque tierce dont l'obtention est à la charge de l'utilisateur

Identification des ayant-droits (3)

- Une fois les ayant droits connus, il peut être utile de déterminer le prorata de leur participation
  - En cas de dépôt du logiciel, souhaitable
- Ce prorata peut être calculé :
  - Soit sur l'ensemble du logiciel
  - Soit relativement aux modules constituant le logiciel
- On prend en compte les personnels impliqués, leur affiliation et le volume de code impacté
- Pas besoin d'être précis à la troisième décimale...

Choix de la licence

- Le choix de la licence est déterminé par :
  - Les souhaits de valorisation des ayant droits
  - Les licences des modules tiers intégrés au logiciel
- Les modules tiers pouvant compromettre le choix de la licence doivent être déportés dans une bibliothèque tierce
  - Décisions à anticiper plutôt qu'à subir

Accord des ayant droits

- Nécessité d'obtenir un accord des ayant droits avant diffusion selon les termes de la licence choisie
  - Un accord écrit est toujours préférable
- En cas d'ayant droits multiples, un mandat de valorisation peut être demandé par l'ayant droit principal auprès des autres
  - Intérêt d'avoir évalué les contributions respectives
  - Géré par le service « valorisation » de l'entité
  - Permet de ne plus avoir qu'un interlocuteur unique
  - Facilite le dépôt du logiciel

Mentions des fichiers (1)

- Tout fichier significatif (apportant une valeur ajoutée) du projet doit être étiqueté
- Un en-tête doit être ajouté, qui comporte :
  - La mention de copyright / droit d'auteur, spécifiant la date de publication et les ayant-droits
  - Les mentions de licence, renvoyant au texte intégral de celle-ci
- Cet en-tête doit précéder l'en-tête de fichier classique spécifiant les auteurs et informations de maintenance
  - Il doit être réputé visible par tout utilisateur ou contributeur

Mentions des fichiers (2)

- Une diffusion sous double licence est possible
  - Gestion de jeux de sources différents
- Mise en place de scripts automatisés d'insertion ou de modification de licence
  - Découplage juridique possible entre la version de travail (tous droits réservés faute de mention expresse) et la version de diffusion

```
for i in *.[ch]
do
  csplit $i 2 31
  cat xx00 textelicence.c.txt xx02 > $i
done
```

Dépôt du logiciel

- Le dépôt du logiciel auprès d'organismes agréés permet d'établir une preuve d'antériorité du logiciel face à des revendications de mauvaise foi
  - Doit idéalement avoir lieu avant la diffusion du logiciel
  - Équivalent à un constat d'huissier
  - Concerne le code source mais aussi tous documents préparatoires, manuels, etc.
- Rien à voir avec un dépôt de brevet logiciel...
  - Illégal en Europe, contraire aux libertés fondamentales, coûteux, inutile, etc...

Types de diffusion

- Les licences libres imposent au diffuseur de mettre, à la disposition de qui le demande, le code source du logiciel
  - Le code source n'a pas besoin d'être fourni d'office
- Deux types de diffusion possibles :
  - Diffusion du code source
  - Diffusion des binaires

Diffusion des binaires

- Plus grande simplicité pour l'utilisateur
  - Tout le travail d'adaptation à un système donné a déjà été réalisé et validé
- Mais problèmes de portabilité
  - Rarement disponibles pour tous les systèmes
  - Absence ou différence de version des bibliothèques tierces au sein d'une même famille de système
- Existence d'informations de gestion des dépendances au sein de paquetages précompilés

Diffusion du code source

- Offre la portabilité maximale
  - Possibilité de retoucher le code pour l'adapter à des systèmes différents ou versions différentes
  - Implique plus la communauté
- Travail plus important pour l'utilisateur
  - Configuration manuelle à réaliser dans la plupart des cas
- Existence de paquetages sources pour lesquels le travail de portage et de spécialisation a été déjà réalisé

Conventions de codage (1)

- L'acceptation d'un logiciel par ses utilisateurs est favorisée par le respect de conventions de codage
  - Portabilité du code source lui-même
    - Règles de codage
    - Bibliothèques système normalisées (POSIX)
    - Tester avec plusieurs compilateurs est un plus
  - Portabilité des procédures d'installation
    - Nommage consistant des sous-arborescences et fichiers produits
    - Recours à des outils standard

Conventions de codage (2)

- Certains systèmes imposent des conventions de codage plus strictes que d'autres
  - Pour Debian, il doit exister une « manpage » pour tout binaire généré
- Des ressources, ainsi que des groupes d'utilisateurs et de packageurs existent pour vous aider à réaliser vos portages, voire les faire à votre place
  - Projet Debian Science
  - FreeBSD

Arborescence d'un projet (1)

- L'arborescence d'un projet contient habituellement, à sa racine :
  - Des répertoires :
    - `src/` contenant les sources du projet
    - `doc/` et/ou `man/` contenant les manuels utilisateur et pages de manuel des commandes et fonctions de bibliothèque
  - Des fichiers :
    - `README` contenant des informations importantes à lire préalablement
    - `INSTALL` contenant les informations d'installation
    - `LICENSE` contenant ou pointant sur le texte intégral de la licence selon laquelle le logiciel est distribué

Arborescence d'un projet (2)

- La structure de la sous-arborescence `/src` est libre
- On y trouve le plus souvent un fichier `Makefile` permettant de construire les binaires, utilisable au moyen de la commande `make`
- Le paramétrage de la compilation en fonction de l'architecture peut se faire :
  - Manuellement par l'utilisateur
  - Au moyen de scripts dédiés
  - Par l'outil `Autoconf`, avec le script `./configure`
  - Par l'outil `Cmake`, avec la commande `cmake`

http://www.cmake.org/
http://www.gnu.org/software/autoconf/

Arborescence d'un projet (3)

- Le processus de compilation doit placer ses résultats dans des sous-répertoires spécifiques de la racine :
  - `include/` : fichiers de déclaration pour les fichiers clients
  - `lib/` : fichiers de bibliothèques pour les logiciels clients
  - `bin/` : programmes exécutables
  - `share/` : données indépendantes de l'architecture et programmes interprétés
- L'étape finale de l'installation placera éventuellement les fichiers produits dans l'arborescence principale du système
  - Privilèges d'administrateur nécessaires

Création de fichiers d'archive

C'est pour offrir ?

Fichiers d'archive (1)

- Un fichier d'archive est un fichier servant de contenant à une sous-arborescence
  - Totale liberté sur le contenu
- Ils permettent de transmettre comme un tout :
  - L'arborescence des sources d'un projet
  - Les binaires précompilés
  - La documentation associée
  - Les licences et scripts d'installation
- Le fichier d'archive peut être compressé ou non
  - Certains outils gèrent simultanément les deux fonctionnalités (`zip`, `tar -z`)

Fichiers d'archive (2)

- Les fichiers d'archive servent majoritairement à diffuser les projets sous forme de codes sources
  - Appelés « *tarball* » dans le jargon informatique
- Aucune contrainte de structure, à l'opposé des paquetages
- Intervention manuelle de l'utilisateur nécessaire pour l'installation
  - Configuration nécessaire en fonction du système
  - Mise en place de règles de bon sens pour lui faciliter le travail

Adaptation des sources

- La multiplicité des plate-formes et des versions de logiciels tiers nécessite des adaptations des sources
  - N'ont pas besoin de figurer toutes dans les sources d'origine
  - Nécessité d'un outil automatique pour les réaliser
- La remontée de correctifs par les utilisateurs nécessite également d'indiquer quelles modifications mineures ont été apportées à une arborescence
  - Doivent ici aussi pouvoir être reportées de façon automatique

Outils `diff` et `patch` (1)

- `Diff` est un outil listant les différences existant entre deux fichiers texte
  - Ajout de lignes
  - Suppression de lignes
  - Modification du contenu d'un bloc de lignes
- Le résultat de `diff` est un flot de texte indiquant à quelles lignes se sont produites les modifications par rapport au premier fichier, et quelles elles sont
  - Utile en tant que tel pour comparer deux fichiers

Outils `diff` et `patch` (2)

- `Diff` et patch permettent la distribution de mises à jour de sources sans avoir à télécharger l'intégralité des nouveaux sources de la distribution
  - Le concepteur du logiciel met en ligne un ensemble de fichiers patch correspondant aux différences entre versions successives, à partir de la version de référence téléchargeable en intégralité
  - Pour se mettre à jour, les utilisateurs téléchargent et appliquent en séquence les patches existant entre leur version et la dernière version disponible

Outils `diff` et `patch` (3)

- Le mécanisme d'analyse de contexte de `diff` et `patch` permet d'appliquer des patches différents sur le même fichier source
  - Tant que les modifications ne portent pas sur les mêmes lignes, il n'y a pas de conflit
  - Existence de patches « non officiels » réalisés par des personnes autres que les développeurs principaux, et relatifs à certains aspects du code
    - Extensions pour un matériel particulier
    - Ajout d'une fonctionnalité non encore « officielle »

Création d'un patch

- On se place dans le répertoire de la version d'origine
- On lance la commande `diff` entre le répertoire courant et la sous-arborescence modifiée

```
% cd ~/paral/scotch/distrib/scotch_5.1
% diff -Naur . /tmp/scotch_5.1_envol/ > /tmp/scotch.patch
```

Création de paquetages

Emballez c'est pesé !

Paquetage

- Un paquetage est un fichier d'archive destiné à simplifier la mise en œuvre d'un logiciel sur une famille de systèmes particulière
- Il contient :
  - Le logiciel proprement dit, sous forme de code source ou de binaire précompilé
  - Des modifications de celui-ci (« patches ») pour corriger des bogues ou l'adapter au système destination
  - Des informations sur le contenu du paquetage et ses dépendances
  - Des informations sur les fichiers à (dés)installer et les commandes à exécuter pour cela

Création des paquetages

- Étapes de base
  - Installation du tarball des sources
  - Paramétrage de l'installation
  - Création du paquetage source ou binaire
- Ne jamais créer un paquetage en tant qu'administrateur du système (« root »)
  - Une bêtise est toujours possible...
- Exemple pratique avec les paquetages Mandriva
  - Utilise le système de paquetages de RedHad

http://wiki.mandriva.com/fr/Tutoriel_RPM

Préalables

- Installation du paquetage rpm-build

```
# urpmi rpm-build
```

- Création dans votre répertoire personnel du fichier `.rpmmacros`

```
% cat > ~/.rpmmacros
%_topdir            %(echo $HOME)/rpm
%_tmppath           %(echo $HOME)/rpm/tmp
%packager           Francois Pellegrini <pelegrin@labri.fr>
%distribution       Mandriva Linux
%vendor             LaBRI - INRIA
^D
```

Création de l'arborescence de travail

- La création et l'assemblage des paquetages RPM nécessitent une arborescence normalisée

```
% mkdir -p ~/rpm/{BUILD,RPMS/
{i586,noarch,x86_64},SOURCES,SRPMS,SPECS,tmp}
```

  - Répertoire des sources pour la création des paquetages : SOURCES
  - Répertoire des fichiers de configuration : SPECS
  - Répertoires de travail : BUILD, tmp
  - Répertoire destination pour les paquetages sources créés par **rpm** : SRPMS
  - Répertoires destination pour les paquetages binaires créés par **rpm** : ~/rpm/RPMS/*architecture*

Installation du tarball d'origine

- Le tarball source et les fichiers patches doivent être placés dans le répertoire ~/rpm/SOURCES

```
% cp scotch_5.1.1.tar.bz2
~/rpm/SOURCES/scotch-5.1.1.tar.bz2
```

  - Il est préférable d'utiliser le format de compression .bz2 plutôt que .gz pour le tarball des sources
- Ne pas compresser les autres fichiers tels que les « patches »
  - Ne permet pas une gestion fine des versions
  - Pénalise leur gestion par SVN

Création du fichier de configuration (1)

- Le fichier de configuration `.spec` contient les informations nécessaires à **rpm** pour :
  - Compiler le programme
  - Créer les RPM source et binaire
  - Installer et désinstaller le programme sur un système Mandriva
- Fichier à placer dans le répertoire `~/rpm/SPECS`
- Il est préférable de s'inspirer de squelettes ou de fichiers existants
  - Nombreuses macros et options

http://wiki.mandriva.com/en/Development/Packaging/Tools/skel.spec

Création du fichier de configuration (2)

- Fichier SPECS/scotch-5.1.1.spec, début

```
%define name    scotch
%define version 5.1.1
%define release %mkrel 1

Name:           %{name}
Summary:        Sequential and parallel graph partitioning and sparse
matrix ordering software
Version:        %{version}
Release:        %{release}
Source0:        %{name}-%{version}.tar.bz2
Patch0:         scotch-5.1.1-bugfix0.patch
URL:            http://gforge.inria.fr/projects/scotch/

Group:          Sciences/Computer science
BuildRoot:      %{_tmppath}/%{name}-%{version}-%{release}-buildroot
License:        CeCILL-C
Requires:       zlib

%description
English blabla describing Scotch.
```

Création du fichier de configuration (3)

- Fichier SPECS/scotch-5.1.1.spec, milieu

```
%prep
%setup -q -n scotch_5.1
%patch -p0

%build
ln -s Make.inc/Makefile.inc.i686_pc_linux2 src/Makefile.inc
%make -C src
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/usr
%makeinstall -C src
%clean
rm -rf $RPM_BUILD_ROOT
```

Création du fichier de configuration (4)

- Fichier SPECS/scotch-5.1.1.spec, fin

```
%files
%defattr(-,root,root)
%doc README.txt LI* doc/*user*
%{_mandir}/man1/*.1*
%{_bindir}/[agm]*
%{_libdir}/*scotch*
%{_includedir}/*scotch*

%changelog
* Fri Oct 10 2008 Francois Pellegrini <pelegrin@labri.fr> 5.1.1-1mdk
- Upgraded to 5.1.1
- Created spec file
```

Création des paquetages

- Une fois le fichier `.spec` créé, on crée les paquetages au moyen de la commande `rpmbuild`

```
% rpmbuild -ba SPECS/scotch_5.1.1.spec
```

- On peut voir le contenu du paquetage créé au moyen de la commande `rpm -qpl`

```
% rpm -qpl RPMS/i586/scotch-5.1.1-1mdv2008.1.i586.rpm
```

Liens avec les logiciels tiers (1)

- Lorsqu'on utilise un logiciel tiers qui est lui-même un projet en développement, la gestion des évolutions concurrentes peut être problématique
  - Modification de l'interface dans la nouvelle version, qui fait échouer la compilation
- La mise à disposition de patches permet d'assurer la transition au niveau du code source
  - On modifie le source dès que la nouvelle version du logiciel tiers est disponible
  - On fait un `diff` avec l'ancienne version pour proposer un patch permettant le retour arrière

Liens avec les logiciels tiers (2)

- Lorsque le logiciel est diffusé sous forme de paquetage binaire, il faut pourvoir gérer la transition
  - Conservation sur son site et mise à disposition des utilisateurs des anciennes versions des paquetages tiers pour lesquelles la compilation et l'installation fonctionnent
- Les dépendances en chaîne entre paquetages sont source de nombreux problèmes
  - Il faut être vigilant et réactif

Diffusion du logiciel

Oyez, oyez !

Diffusion du logiciel

- Il existe plusieurs moyens de mettre le logiciel à la disposition de ses utilisateurs
  - Diffusion à la demande
    - Présente fort peu d'intérêts vis-à-vis d'un logiciel sous licence libre
  - Mise en ligne des paquetages
  - Accès à un dépôt de sources

Mise en ligne du paquetage (1)

- Le tarball ou le paquetage est librement téléchargeable par les utilisateurs
- L'accès aux fichiers peut être :
  - Libre : l'URL du paquetage peut être communiquée telle quelle
  - Filtré : l'utilisateur doit fournir certaines informations (adresse courriel) avant que le paquetage ne soit envoyé à son adresse ou qu'une URL particulière ne soit mise à sa disposition

Mise en ligne du paquetage (2)

- Le paquetage peut être mis à disposition :
  - Sur un site local
    - Possibilité d'instrumenter le site à sa guise
  - Sur un site tiers
    - Site en général bien adapté en terme de stockage et de bande passante
    - Intégration à des projets thématiques
      - Distribution Debian Science, etc.
- Les sources tierces empêchent l'établissement de statistiques d'usage mais améliorent la popularité

Dépôts de sources (1)

- Les dépôts de type `CVS`, `SVN` ou `Linux-GIT` permettent à l'équipe de développement une gestion fine des versions du code source
- Une vue en consultation sur le dépôt peut être accordée à certains utilisateurs ou au public
  - Permettent la mise à jour des sources chez les utilisateurs au gré des mises à jour du dépôt
  - Cas des forges logicielles, comme l'INRIA Gforge

Dépôts de sources (2)

- Il est souvent préférable de disposer de deux projets :
  - Un projet privé, sur lequel s'effectue le développement collaboratif
    - Version « expérimentale » parfois non compilable
  - Un projet public, sur lequel les utilisateurs ont un accès en lecture
    - Historique de versions stables
- Le transfert entre projets permet de vérifier la nature des modifications apportées au projet

Publicité

- Afin de maximiser le retour sur investissement de la dépense publique, il est nécessaire d'informer les utilisateurs potentiels de toute mise à disposition
- Les moyens pour cela sont :
  - La mise en ligne d'informations sur le site web des instituts et des personnels
  - L'envoi d'annonces à des listes de diffusion
  - Le recensement du logiciel dans des annuaires
    - Par exemple : Framasoft, Plume, Relier, ...