HOWTO sur la publication de logiciels

Eric S. Raymond <esr@thyrsus.com> Traduction par Thierry Bézecourt <thbzcrt@worldnet.fr> 2.4, 12 juillet 2000

Ce HOWTO décrit des méthodes de publication de logiciel convenant à des projets de logiciel libre pour Linux. En adoptant ces règles, vous permettrez à vos utilisateurs de compiler votre code et de l'utiliser plus facilement, et à d'autres développeurs de mieux le comprendre et de vous aider à l'améliorer. Ce document est à lire absolument par les développeurs débutants. Ceux qui ont plus d'expérience devraient le parcourir à nouveau au moment de publier un nouveau projet. Il sera mis à jour périodiquement afin de refléter l'évolution des règles de bonne pratique.

Contents

1 Introduction

1.1 Raison d'être de ce document

Un vaste ensemble de traditions relatives au développement de logiciels libres permet à d'autres personnes de porter le code plus facilement, de l'utiliser et de participer à son développement. Certaines de ces conventions sont des traditions du monde Unix antérieures à Linux ; d'autres ont été suscitées récemment par l'apparition de nouveaux outils et de nouvelles technologies comme le World Wide Web.

Ce document vous aidera à acquérir ces règles d'usage. Il se compose de plusieurs sections thématiques, chacune contenant une liste de points à vérifier. Considérez que ces sections sont pour votre distribution comme la liste de contrôle qu'un pilote d'avion vérifie avant le décollage.

1.2 Nouvelles versions de ce document

Ce document sera envoyé chaque mois dans le forum de discussion `comp.os.linux.answers` . Ce document est archivé sur plusieurs sites FTP Linux, dont `metalab.unc.edu` dans le répertoire `pub/Linux/docs/HOWTO`.

Vous pouvez aussi voir la dernière version, en anglais, de ce HOWTO sur le World Wide Web à l'URL `http://www.linuxdoc.org/HOWTO/Software-Release-Practice.html` . La version française est disponible à l'adresse `http://metalab.unc.edu/pub/Linux/docs/HOWTO/translations/fr` .

Vous pouvez envoyer vos questions et vos commentaires sur ce HOWTO à Eric S. Raymond, *esr@snark.thyrsus.com* `mailto:esr@snark.thyrsus.com` .

1.3 Note du traducteur sur l'usage de l'anglais dans le document

On a choisi de ne pas traduire les noms de fichiers que l'auteur recommande d'inclure dans un logiciel. En effet, le monde des logiciels libres est fortement internationalisé, et il utilise l'anglais comme langue commune. On supposera donc que le lecteur qui souhaiterait mettre en application les conseils de ce HOWTO connaît suffisamment d'anglais pour écrire des documents tels que le fichier README ou le fichier INSTALL. Cela n'interdit pas, bien entendu, d'inclure dans les projets une version française de ces documents, qui sera très appréciée des utilisateurs francophones, qu'ils parlent ou non l'anglais.

2 Règles d'usage pour l'appellation de votre projet et de votre archive

Au fur et à mesure que s'accroît la charge de travail des gestionnaires d'archives comme Metalab, le site PSA ou le CPAN, les soumissions sont de plus en plus souvent traitées, en tout ou en partie, par des programmes (et non en totalité par des humains).

Il est donc très important que le nom de votre projet et celui de votre fichier d'archive suivent des règles précises, afin que des programmes informatiques puissent les analyser et les comprendre.

2.1 Utilisez le style d'appellation GNU, avec un préfixe suivi d'un numéro du type majeur.mineur.patch.

Vous faciliterez la vie à tout le monde en donnant à vos archives des noms dans le style GNU : un préfixe-racine alphanumérique tout en minuscules, suivi par un tiret, puis un numéro de version, une extension et d'autres suffixes.

Supposons que vous ayez un projet nommé "toto", qui en est à la version 1, mise à jour 2, niveau 3. S'il est composé d'une seule archive (sans doute le code source), voici à quoi devrait ressembler son nom :

toto-1.2.3.tar.gz

L'archive des sources

toto.lsm

Le fichier LSM (si vous l'envoyez à Metalab).

N'utilisez *pas* les noms suivants :

toto123.tar.gz

Beaucoup de programmes croiront qu'il s'agit du fichier d'archive d'un projet nommé 'toto123', sans numéro de version.

toto1.2.3.tar.gz

Beaucoup de programmes croiront qu'il s'agit de l'archive d'un projet nommé 'toto1' à la version 2.3.

toto-v1.2.3.tar.gz

Beaucoup de programmes prendront cela pour un projet nommé 'toto-v1'.

to_to-1.2.3.tar.gz

Le caractère souligné est difficile à prononcer, à taper, et à retenir.

ToTo-1.2.3.tar.gz

A moins que vous vouliez *vraiment* ressembler à un accroc du marketing. Là encore, c'est difficile à prononcer, à taper et à retenir.

Si vous voulez faire séparément une archive de sources et une archive de binaires, ou différentes archives de binaires, ou encore indiquer un certain type d'option de fabrication dans le nom de l'archive, rajoutez pour cela une extension *après* le numéro de version. Voici quelques exemples :

toto-1.2.3.src.tar.gz
: sources

toto-1.2.3.bin.tar.gz
: binaires, type non spécifié

toto-1.2.3.bin.ELF.tar.gz
: binaires ELF

toto-1.2.3.bin.ELF.static.tar.gz
: binaires ELF liés statiquement

toto-1.2.3.bin.SPARC.tar.gz
: binaires pour SPARC

N'utilisez *pas* des noms comme 'toto-ELF.1.2.3.tar.gz', car les programmes ont beaucoup de mal à séparer un infixe (tel que 'ELF') de la racine du mot.

Un bon schéma d'appellation générique contient, dans l'ordre, les parties suivantes :

1. préfixe du projet
2. tiret
3. numéro de version
4. point
5. "src" ou "bin" (optionnel)
6. point ou tiret (un point de préférence)
7. type de binaire et options (optionnel)
8. extensions relatives au mode d'archivage et de compression

2.2 Mais respectez le cas échéant les conventions locales

Certains projets ou communautés ont des conventions bien établies pour les noms et les numéros de version, et ces conventions ne sont pas toujours compatibles avec les conseils qui précèdent. Par exemple, les modules Apache ont en général des noms du genre mod_foo, et ils ont à la fois un numéro de version propre et le numéro de la version d'Apache avec laquelle ils fonctionnent. De même, les numéros de version des modules Perl peuvent être traités comme des nombres décimaux (par exemple, vous pouvez voir 1.303 à la place de 1.3.3), et les distributions s'appellent en général Foo-Bar-1.303.tar.gz pour la version 1.303 du module Foo::Bar.

Apprenez et respectez les conventions des communautés et développeurs spécialisés ; suivez les règles décrites ci-dessus dans le cas général.

2.3 Choisissez avec le plus grand soin un préfixe unique et facile à taper

Le préfixe-racine devrait être le même pour tous les fichiers d'un projet, et il devrait être facile à lire, à taper et à retenir. N'utilisez pas le caractère "souligné". Et ne mettez pas de majuscules ou de MajusculesIntérieures sans une très bonne raison – cela dérange le trajet naturel de l'oeil humain, et vous aurez l'air de faire du marketing.

C'est difficile de s'y retrouver lorsque deux projets ont le même nom. Assurez-vous donc, dans la mesure du possible, qu'il n'y a pas de conflit de noms avant de publier votre première version. Deux bons endroits pour vérifier ceci sont *l'index de Metalab* `http://metalab.unc.edu/pub/Linux` et l'index des applications (appindex) à *Freshmeat* `http://www.freshmeat.net` . Un autre endroit recommandé est *SourceForge* `http://www.sourceforge.net` , en effectuant une recherche par nom.

3 Règles d'usage pour la licence et le copyright : la théorie

La licence que vous choisissez définit le contrat social que vous souhaitez mettre en place avec vos co-développeurs et vos utilisateurs. Le copyright que vous mettez sur le logiciel sert principalement de déclaration légale de votre droit à fixer les termes de la licence sur le logiciel et sur les oeuvres qui en sont dérivées.

3.1 Les logiciels à code ouvert et le copyright

(Note du traducteur : dans cette section comme dans celles qui suivent, l'expression "(logiciel à) code ouvert" est utilisée pour traduire l'anglais "open source", tandis que l'expression habituelle "logiciel libre" sert à transcrire "free software")

Tout ce qui n'appartient pas au domaine public possède un copyright, voire plusieurs. Selon la Convention de Berne (qui a force de loi aux Etats-Unis depuis 1978), le copyright n'a pas besoin d'être explicite. C'est-à-dire que les auteurs d'une oeuvre sont détenteurs du copyright même s'il n'y a pas de note de copyright.

Il peut être très difficile de déterminer qui peut être compté comme un auteur, surtout lorsque de nombreuses auteurs ont travaillé sur le logiciel. C'est ce qui fait l'importance des licences. En précisant les conditions dans lesquelles l'oeuvre peut être utilisée, elles donnent aux utilisateurs des droits qui les protègent des actions arbitraires que pourraient entreprendre les détenteurs du copyright.

Dans le logiciel propriétaire, les termes de la licence sont formulés de manière à protéger le copyright. Ils permettent de donner quelques droits aux utilisateurs tout en assurant au propriétaire (le détenteur du copyright) la plus grande possibilité d'action possible sur le plan légal. Le détenteur du copyright a une grande importance, et la licence est tellement restrictive dans l'esprit que les détails techniques de ses termes sont généralement sans importance.

Dans le logiciel à code ouvert, la situation est souvent exactement inverse ; le copyright existe pour protéger la licence. Les seuls droits qui sont toujours conservés au détenteur du copyright sont ceux qui permettent de renforcer la licence. Parmi les autres droits, un petit nombre seulement sont réservés, et c'est l'utilisateur qui a la plus grande liberté. En particulier, le détenteur du copyright ne peut pas modifier la licence sur une copie que vous possédez déjà. Par conséquent, le détenteur du copyright dans les logiciels à code ouvert n'a presque aucune importance – contrairement aux termes de la licence.

Normalement, le détenteur du copyright sur un projet est le responsable actuel du projet ou une organisation mécène. Le changement de responsable à la tête d'un projet se manifeste souvent par la modification du copyright. Toutefois ce n'est pas une règle absolue ; dans de nombreux projets à code source ouvert, le copyright revient à de multiples personnes, et il n'y a pas de cas connu où cela ait entraîné des complications sur le plan légal.

Certains projets choisissent de donner le copyright à la Free Software Foundation, avec l'idée que cette fondation a un intérêt dans la défense du logiciel à code ouvert, et possède les avocats pour s'en occuper.

3.2 Déterminer ce qui peut être qualifié comme logiciel à code ouvert

En ce qui concerne la licence, on peut distinguer plusieurs sortes de droits transférables via une licence. Les droits de *copie et de redistribution*, les droits d'*utilisation*, les droits de *modification à usage personnel* et les droits de *redistribution de copies modifiées*. Une licence peut restreindre chacun de ces droits ou les accompagner de conditions.

L' *Open Source Initiative* `http://www.opensource.org`

est le résultat d'un important effort de réflexion sur ce qui fait un "logiciel à code ouvert" ou (dans une terminologie plus ancienne) un "logiciel libre". L'association place les contraintes suivantes sur les licences :

1. Un droit de copie illimité doit être accordé.
2. Un droit d'utilisation illimité doit être accordé.
3. Un droit de modication illimité pour utilisation personnelle doit être accordé.

Ces règles proscrivent les restrictions sur la redistribution de binaires modifiés ; cela correspond aux besoins des distributeurs de logiciels, à qui il faut pouvoir livrer sans entraves du code en état de marche. Cela permet aux auteurs de demander que le code source modifié soit redistribué sous la forme du code source original plus des patchs, ce qui fait apparaître les intentions de l'auteur et, dans un"suivi d'audit", toutes les modifications faites par d'autres personnes.

L'OSD est la définition légale de la marque de certification 'OSI Certified Open Source', et vaut toutes les définitions qu'on a pu faire du "logiciel libre" (note du traducteur : OSI est ici l'Open Source Initiative et n'a rien à voir avec l'Organisme de Standardisation Internationale ou ISO). Toutes les licences courantes (MIT, BSD, Artistic et GPL/LGPL) la vérifient (encore que certaines, comme la GPL, ajoutent d'autres restrictions que vous devriez comprendre avant de les adopter).

Notez que les licences n'autorisant qu'un usage non commercial ne peuvent *pas* être qualifiées de licences à code ouvert, même si elles affichent la licence "GPL" ou quelque autre licence courante. Elles font de la discrimination envers des emplois, des personnes et des groupes particuliers. Elles rendent la vie trop compliquée aux distributeurs de CD-ROM et aux autres personnes qui essaient de diffuser commercialement les logiciels à code ouvert.

4 Règles d'usage pour la licence et le copyright : la pratique

Voici comment appliquer dans la pratique la théorie qui précède :

4.1 Donnez le copyright à vous-même ou à la FSF

Dans certains cas, si vous avez derrière vous une organisation mécène qui possède des avocats, vous pouvez choisir de donner le copyright à cette organisation.

4.2 Choisissez une licence conforme à l'Open Source Definition

L'Open Source Definition (Définition du Code Ouvert) est la règle d'or pour les licences. L'OSD n'est pas une licence en soi ; elle définit plutôt un ensemble minimal de droits qu'une licence doit garantir afin d'être considérée comme une licence à code ouvert. On peut trouver L'OSD, avec des documents complémentaires, sur le site Web de l' *Open Source Initiative* `http://www.opensource.org` .

4.3 N'écrivez pas votre propre licence si vous pouvez l'éviter.

Les licences compatibles à l'OSD et connues de tous ont des traditions d'interprétation bien établies. Les développeurs (et, dans la mesure où ils s'y intéressent, les utilisateurs) savent ce qui en découle, et mesurent les risques et les inconvénients qu'elles comportent. Par conséquent, utilisez si possible l'une des licences standards sur le site de l'Open Source Initiative.

Si vous devez écrire votre propre licence, prenez soin de la faire certifier par l'Open Source Initiative. Cela vous épargnera de nombreuses discussions et des coûts importants. Si vous n'êtes jamais passé par là, vous ne pouvez pas imaginer pas à quel point un débat sur les licences peut tourner au vinaigre ; les gens s'enflamment, parce que les licences sont considérées comme des pactes presque sacrés qui touchent aux valeurs essentielles de la communauté des logiciels ouverts.

De plus, l'existence d'une tradition d'interprétation bien établie pourrait se révéler importante si un jour votre licence faisait l'objet d'un procès. A la date où ces lignes sont écrites (septembre 1999), il n'y a pas d'exemple de décision judiciaire qui ait confirmé ou invalidé une licence de logiciel à code ouvert. Toutefois, c'est un principe de droit (au moins aux Etats-Unis, et sans doute dans d'autres pays de droit coutumier comme l'Angleterre et le reste du Commonwealth) que les cours de justice doivent interpréter les licences et les contrats en fonction des attentes et des pratiques de la communauté qui les a produits.

5 Règles d'usage du développement

La plupart de ces règles visent à assurer la portabilité, non seulement entre les différentes distributions de Linux, mais aussi avec d'autres variétés d'Unix. La portabilité vers Unix n'est pas seulement une question de professionnalisme ou de savoir-vivre entre programmeurs, c'est aussi une assurance non négligeable contre les évolutions futures de Linux lui-même.

D'autres personnes *finiront* par essayer de compiler votre code dans d'autres environnements que Linux ; avec la portabilité, vous recevrez moins de messages ennuyeux de la part d'utilisateurs perplexes.

5.1 Ecrivez soit en C ANSI pur, soit dans un langage de script portable

Pour des raisons de portabilité et de stabilité, il est fortement recommandé d'écrire soit en C ANSI pur, soit dans un langage de script dont la portabilité soit garantie par une implémentation multi-plateforme unique.

Parmi les langages de script, Python, Perl, Tcl, Emacs Lisp et PHP respectent ce critère. Le bon vieux shell *ne convient pas* ; il existe trop d'implémentations différentes, chacune ayant des particularités subtiles, et l'environnement du shell est souvent transformé de manière imprévisible par des configurations propres à chaque utilisateur, comme les alias.

Java promet beaucoup sur le plan de la portabilité, mais les implémentations disponibles sur Linux sont encore sommaires et mal intégrées dans le système d'exploitation. Java est encore un choix exotique, bien qu'il ait de fortes chances de gagner en popularité lorsqu'il aura plus de maturité.

5.2 Respectez les règles de portabilité du C

Si vous écrivez en C, n'hésitez pas à utiliser à fond les fonctionnalités décrites dans la norme ANSI – y compris les prototypes de fonction, qui vous aideront à repérer les incohérences entre modules. Les compilateurs du type K&R relèvent du passé.

En revanche, ne supposez *pas* que certaines caractéristiques spécifiques à GCC comme l'option '-pipe' ou les fonctions imbriquées sont disponibles. Cela vous poursuivra et vous vous en repentirez le jour où quelqu'un portera votre logiciel vers un système autre que Linux, ou dépourvu de GCC.

5.3 Utilisez autoconf/automaker/autoheader

Si vous écrivez en C, utilisez autoconf/automaker/autoheader pour assurer la portabilité, vérifier la configuration du système et affiner vos makefiles. De nos jours, les gens qui compilent à partir des sources s'attendent à devoir juste taper "configure; make" et que tout se compile proprement - sans la moindre erreur.

5.4 Soignez la rigueur de votre code avant chaque nouvelle version

Si vous écrivez en C, faites une compilation de test avec l'option -Wall et corrigez les erreurs résultantes, au moins une fois avant chaque nouvelle version. On trouve comme cela un nombre surprenant d'erreurs. Pour être vraiment complet, compilez aussi avec l'option -pedantic.

Si vous écrivez en Perl, vérifiez votre code avec perl -c (voire -T dans les cas adéquats). Utilisez perl -w et 'use strict' religieusement (consultez la documentation de Perl pour plus d'informations).

5.5 Soignez votre documentation et vos README avant la livraison

Passez-les au correcteur d'orthographe. Si vous donnez l'impression de ne pas connaître l'orthographe et de vous en moquer, les gents penseront que vous avez aussi bâclé votre code.

6 Règles d'usage pour la mise au point de la distribution

Les indications qui suivent montrent à quoi votre distribution devrait ressembler lorsqu'on la récupère et qu'on la décompacte.

6.1 Assurez-vous que vos archives se décompactent toujours dans un répertoire nouveau et unique.

L'erreur la plus agaçante que font les développeurs novices est de fabriquer des archives qui décompactent les fichiers et répertoires de la distribution dans le répertoire courant, avec le risque d'écraser des fichiers déjà présents. *Ne faites jamais cela !*

A la place, faites en sorte que les fichiers de vos archives partagent le même répertoire, avec un nom dérivant de celui du projet. Ainsi, ils se placeront dans un seul répertoire, juste *en-dessous* du répertoire courant.

Voici un moyen de réaliser cela dans un makefile, en supposant que le répertoire de votre distribution est 'toto' et que SRC est une variable contenant la liste des fichiers de votre distribution. Vous devez avoir GNU tar 1.13.

```
VERS=1.0
toto-$(VERS).tar.gz:
        tar --name-prefix='toto-$(VERS)/' -czf toto-$(VERS).tar.gz $(SRC)
```

Si votre version de tar est plus ancienne, faites quelque chose dans ce genre :

```
toto-$(VERS).tar.gz:
	@ls $(SRC) | sed s:^:toto-$(VERS)/: >MANIFEST
	@(cd ..; ln -s toto toto-$(VERS))
	(cd ..; tar -czvf toto/toto-$(VERS).tar.gz `cat toto/MANIFEST`)
	@(cd ..; rm toto-$(VERS))
```

6.2 Ecrivez un README

Fournissez un fichier nommé README ou READ.ME, qui donne une vision d'ensemble de votre distribution. C'est une vieille convention, et c'est le premier fichier que l'intrépide explorateur lira après avoir extrait les sources.

Voici quelques éléments qu'il est bon d'avoir dans un README :

- Une brève description du projet.
- Un lien vers le site Web du projet, le cas échéant.
- Des indications sur l'environnement de compilation du développeur et sur les possibles problèmes de portabilité.
- Un plan d'ensemble des fichiers et sous-répertoires importants.
- Les instructions de compilation et d'installation, ou bien un lien vers le fichier les contenant (habituellement INSTALL).
- Le nom des responsables et des contributeurs, ou un lien vers le fichier contenant ces noms (habituellement CREDITS).
- Les dernières nouvelles relatives au projet, ou un lien vers un fichier les contenant (habituellement NEWS).

6.3 Adoptez les conventions courantes d'appellation des fichiers

Avant même d'avoir regardé le README, votre intrépide explorateur aura parcouru la liste des fichiers dans le répertoire principal de votre distribution. Ces noms, par eux-mêmes, contiennent de l'information. En suivant certaines règles d'appellation, vous donnerez à l'explorateur de bons indices pour orienter son parcours.

Voici quelques noms recommandés pour les fichiers du répertoire principal, avec leur signification. Tous ne sont pas indispensables dans chaque projet.

README ou READ.ME

le plan d'ensemble, à lire en premier

INSTALL

instructions de configuration, de compilation et d'installation

CREDITS

liste des contributeurs du projet

NEWS

dernières nouvelles

HISTORY

histoire du projet

COPYING

termes de la licence (convention GNU)

LICENSE

termes de la licence

MANIFEST

liste des fichiers

FAQ

"Foire Aux Questions" pour le projet, au format texte.

TAGS

fichier de tags généré automatiquement, pour être utilisé par Emacs ou vi

Remarquez que la convention générale est que les fichiers dont le nom ne comporte que des majuscules sont des fichiers d'information sur le projet lui-même, et ne sont pas des éléments du système à compiler.

La présence d'une FAQ vous soulagera beaucoup. Quand une question relative au projet revient fréquemment, rajoutez-la dans la FAQ. Puis demandez aux utilisateurs de lire la FAQ avant de poser des questions ou d'envoyer des rapports de bogues. Une FAQ bien entretenue peut réduire d'au moins un ordre de grandeur la charge du support pour les mainteneurs du projet.

Il est bon d'avoir un fichier HISTORY ou NEWS avec un historique du projet mis à jour à chaque nouvelle version. Entre autres choses, il pourrait vous aider à prouver votre antériorité si vous étiez pris dans un procès pour contrefaçon (ce n'est jamais encore arrivé à personne, mais autant y être préparé).

6.4 Prévoyez les mises à jour

Votre logiciel évoluera dans le temps au rythme des versions successives. Certains des changements ne seront pas compatibles avec l'existant. Par conséquent, réfléchissez bien à l'organisation du logiciel afin que plusieurs versions puissent être installées et coexister sur le même système. C'est particulièrement important pour les bibliothèques de fonctions : vous ne pouvez pas compter que tous les programmes clients soient mis à jour d'un seul coup pour s'adapter aux modifications de vos interfaces.

Les projets Emacs, Python et Qt ont adopté une bonne convention pour traiter ce problème, celle des répertoires numérotés par version. Voici à quoi ressemble une hiérarchie de bibliothèques Qt (${ver} est le numéro de version) :

```
/usr/lib/qt
/usr/lib/qt-${ver}
/usr/lib/qt-${ver}/bin          # Emplacement de moc
/usr/lib/qt-${ver}/lib          # Emplacement des .so
/usr/lib/qt-${ver}/include      # Emplacement des fichiers d'en-tête
+
```

Une telle organisation vous permet de faire coexister plusieurs versions. Les programmes clients doivent spécifier quelle version de la bibliothèque ils désirent, mais c'est un faible prix à payer en comparaison des incompatibilités d'interface qu'ils évitent.

6.5 Fournissez des RPM

Le format standard de facto pour les distributions de binaires à installer est celui qu'utilise le Red Hat Package Manager, RPM. Il est présent dans les distributions Linux les plus populaires, et il est supporté en pratique par toutes les autres (sauf Debian et Slackware ; et Debian peut faire des installations à partir de RPM).

Par conséquent, c'est une bonne idée de fournir sur le site de votre projet des RPM installables en même temps qu'une archive des sources.

C'est aussi une bonne idée d'inclure dans votre archive de sources le fichier de spécification du RPM, avec dans le Makefile une entrée qui fabrique les RPM à partir de lui. Le fichier de spécification devrait avoir l'extension '.spec' ; c'est comme cela que l'option -t de rpm le trouve à l'intérieur de l'archive.

Encore un point de style : générez votre fichier de spécification avec un script shell qui insère automatiquement le numéro de version en analysant le Makefile ou un fichier version.h.

Note : si vous fournissez des RPM sources, utilisez BuildRoot pour fabriquer le programme dans /tmp ou /var/tmp. Si vous ne le faites pas, l'installation, réalisée au cours de la partie install de votre fabrication, se déroulera dans les répertoires finaux. Ceci se produira même en cas d'homonymie de fichiers ou si vous ne désirez pas réellement installer le produit. Une fois la fabrication terminée, les fichiers seront installés à leur emplacement définitif, et la base de données RPM du système ne sera pas mise à jour. Des SRPM ayant ce mauvais comportement sont des champs de mines et doivent être évités.

7 Comment bien communiquer

Votre logiciel n'apportera pas grand-chose à l'univers si vous êtes le seul à connaître son existence. De plus, en établissant une présence visible sur Internet pour votre projet, vous pourrez recruter plus facilement des utilisateurs et des co-développeurs. On le fait habituellement comme ceci :

7.1 Faites une annonce dans c.o.l.a et sur Freshmeat

Annoncez vos nouvelles versions dans *comp.os.linux.announce* `news:comp.os.linux.announce` . Non seulement ce forum est lu par un grand nombre de personnes, mais c'est aussi un fournisseur important pour des sites Web d'information comme *Freshmeat* `http://www.freshmeat.net` .

7.2 Faites une annonce dans un forum de discussion adéquat

Trouvez un forum USENET dont le thème de discussion est directement concerné par votre application, et faites-y aussi votre annonce. N'envoyez votre message qu'aux endroits où la *fonction* remplie par votre logiciel est pertinente, et restez mesuré.

Si (par exemple) vous publiez un programme en Perl qui interroge des serveurs IMAP, vous devriez probablement envoyer un message dans comp.mail.imap. Mais sûrement pas dans comp.lang.perl, à moins que le programme utilise de manière instructive des techniques Perl avancées.

Votre annonce devrait aussi contenir l'URL du site Web de votre projet.

7.3 Ayez un site Web

Si vous comptez établir une communauté substantielle d'utilisateurs ou de développeurs autour de votre projet, celui-ci devrait avoir son site Web. Voici des éléments que l'on trouve habituellement sur un site Web :

- La charte du projet (pourquoi il existe, quelle est son audience, etc).
- Des liens pour le téléchargement des sources.
- Des instructions relatives à l'inscription à la ou les liste(s) de diffusion.
- Une FAQ (Foire Aux Questions).
- Une version en HTML de la documentation.
- Des liens vers des projets proches et/ou concurrents.

Certains projets ont même une URL pour un accès anonyme à l'arborescence principale du code source.

7.4 Hébergez des listes de diffusion pour votre projet

Il est d'usage d'avoir une liste de développement privée qui permet aux collaborateurs du projet de communiquer et d'échanger des patchs. Vous voudrez peut-être créer en plus une liste d'annonces pour les gens qui veulent être informés de la progression du projet.

7.5 Publiez dans les archives les plus importantes

Depuis plusieurs années, le *site Metalab* `http://www.metalab/unc.edu/pub/Linux/` est le plus important des endroits d'échange de logiciels pour Linux.

Voici quelques autres sites notables :

- le site *Python Software Activity* `http://www.python.org` (pour les logiciels écrits en Python).
- le *CPAN* `http://language.perl.com/CPAN` ou Réseau d'Archives Perl Global (pour les logiciels écrits en Perl).

8 La bonne gestion d'un projet

Bien gérer un projet lorsque tous les participants sont bénévoles présente des défis particuliers. Le sujet est trop large pour être traité dans le cadre d'un HOWTO. Heureusement, il existe des documents de réflexion qui vous aideront à comprendre les principaux points.

Pour un essai sur l'organisation de base du développement et du principe 'distribuez tôt, mettez à jour souvent' propre au 'mode bazar', lisez *The Cathedral and the Bazaar* `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/` .

Pour une réflexion sur les motivations psychologiques, des coutumes de la communauté et de la résolution des conflits, lisez *Homesteading the Noosphere* `http://www.tuxedo.org/~esr/writings/homesteading/` .

Pour un exposé des principes économiques et des modèles commerciaux à adopter, lisez *The Magic Cauldron* `http://www.tuxedo.org/~esr/writings/magic-cauldron/` .

Si vous êtes allergique à la langue de Shakespeare, vous pourrez trouver des traductions de ces documents sur le site *Linux France* `http://www.linux-france.org/article/these/` .

Ces papiers ne prétendent pas se poser comme les références ultimes à propos des développements à code source ouvert. Mais ils constituent les premières analyses sérieuses écrites sur le sujet, et n'ont pas encore été dépassés (l'auteur espère toutefois qu'ils le seront un jour).