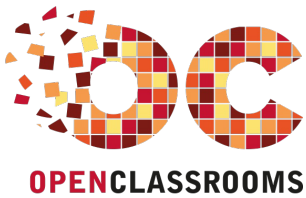
Le guide du contributeur

Par Cygal



www.openclassrooms.com

Dernière mise à jour le 24/09/2009

Sommaire

Le guide du contributeur

Par  Cygal

Mise à jour : 24/09/2009
Difficulté : Facile

Vous utilisez tous les jours des logiciels libres tels que Firefox ou GNU/Linux. Vous vous êtes rendu compte qu'au fur et à mesure des versions, ces logiciels s'amélioraient, obtenaient de nouvelles fonctionnalités, et qu'ils restaient gratuits. Vous avez peut-être aussi entendu dire que ces programmes étaient développés par des personnes, parfois dans leur temps libre, parfois payées par une société qui a intérêt que le logiciel se développe. Pourquoi pas vous ?

Cet article vous montre comment contribuer dans le logiciel libre, pas à pas, en vous montrant la démarche à effectuer. Si vous êtes pressés ou connaissez déjà un peu le sujet, n'hésitez pas à lire en diagonale jusqu'aux parties vous concernant.
Sommaire du tutoriel :

Pourquoi lire cet article ?

Vous avez acquis des connaissances en programmation, quel que soit le langage. Vous pouvez réaliser de petits projets dans votre coin, pour vous améliorer, ou parce que vous avez un objectif particulier. Vous pouvez aussi travailler à plusieurs, entre copains, parce que c'est rigolo, parce que ça vous permet d'apprendre, et parce que ça vous permet d'avoir plus de possibilités.

Utiliser des logiciels libres, cela permet aussi d'améliorer un logiciel qu'on utilise tous les jours, en écrivant de la documentation, une traduction, en ajoutant une fonctionnalité utile, ou en corrigeant un bug. Pour faire simple, cela permet d'augmenter la qualité globale du logiciel, au lieu d'avoir *n* clones propriétaires : les efforts se font sur un seul logiciel. Le but ici n'est pas d'essayer de vous faire croire que dans le libre tout est rose, que tout le monde est gentil, mignon, et que c'est facile (au contraire) mais que cela est une bonne idée, et que tout le monde est gagnant au final.

Plaçons-nous donc dans la peau d'un zéro qui a déjà un certain nombre de compétences qui pourront être utiles à un projet quelconque. Cela peut être le projet d'un zéro, un projet découvert sur la toile, ou n'importe quoi d'autre. Les compétences peuvent être très variées, contrairement à ce que vous pourriez penser :

- la capacité à écrire de la documentation pour permettre aux autres utilisateurs de mieux se servir du logiciel ;
- la volonté d'écrire une traduction dans une autre langue afin que le logiciel puisse être utilisé par davantage de personnes ;
- l'élaboration de contributions multimédias (icônes, images, musiques, etc.) ;
- ou encore la capacité à modifier ou écrire du code pour ce projet, entre autres.

Seulement dans tous les cas il faut un peu d'organisation, et savoir comment les projets libres s'organisent pour permettre tout cela. Cet article se place dans la position du contributeur, du pauvre zéro gentil qui veut essayer d'aider les super-héros qui s'occupent d'un projet quelconque. Vous pouvez aussi utiliser ces connaissances pour mettre en place votre projet, mais cela est plus dangereux, et il y a d'autres problèmes qui ne seront pas traités dans cet article.

Ainsi, vous voulez offrir votre aide pour un projet quelconque, parce que vous êtes convaincu que ce projet a de l'avenir, que votre aide sera appréciée, et que vous apprendrez en y contribuant. Mais avant de vous impliquer, il faut comprendre comment

fonctionnent les interactions entre développeurs au sens large.

Les différents types de projets

Il faut d'abord comprendre quels grands groupes de logiciels libres existent. C'est schématique, et le projet que vous choisirez fonctionnera probablement un peu différemment, mais les éléments qui suivent sont des constantes qui permettent de s'y retrouver dans la jungle des projets libres. Attention, ce n'est pas un ordre chronologique, juste quelques *photographies* de projets types.

Projet individuel

C'est le moment où les premières lignes de code sont écrites, le moment où on ne sait toujours pas si c'est un projet qui vaut le coup, et le moment où on ne sait pas si on va le continuer. Toujours est-il qu'on est excité par toutes les possibilités qui s'offrent à nous, on s'amuse comme des petits fous, etc. Rien de très intéressant à part si le but est dès le départ de faire un projet à plusieurs. À ce moment-là, le projet est en stade *Survie*, tant qu'il n'est pas abandonné.

Petit groupe

Le développeur principal sait qu'il a envie de continuer son projet. S'il est au courant des pratiques et de l'intérêt du monde libre, il va probablement faciliter l'accès au code source et tenter de recruter quelques personnes pour l'aider. Si ces personnes sont convaincues, motivées, capables d'aider et que le projet est faisable, alors le projet pourra avancer. Du sang neuf pourra envoyer des modifications qui lui sembleront utiles, ou même intégrer l'équipe une fois qu'il aura vu les méthodes de développement.

Gros projet

Certains projets, une fois qu'ils se sont avérés utiles (exemple : WordPress), prometteurs (exemple : 0 A.D), ou autres, et ont su se faire connaître, pourront attirer de nouveaux développeurs. Si ces développeurs sont actifs, le projet aura tout intérêt à mettre en place des outils de développement efficaces pour leur éviter au maximum de se marcher dessus pendant qu'ils travaillent.

Énorme projet

Ceux-là sont très rares, et ont un challenge important à relever : savoir mettre en place une architecture suffisamment efficace pour ne pas ralentir l'arrivée des nouveaux contributeurs. Des responsables sont alors nommés, et ils deviennent des points de contact pour les nouveaux contributeurs.

La théorie

Vous (munis de toute la bonne volonté du monde) avez donc détecté le projet de vos rêves et décidé de tenter l'aventure. La première chose à faire est de récupérer les sources, de réussir à les compiler, et de lancer le tout. Cela permettra plus tard de pouvoir tester vos modifications, et de les envoyer pour que tout le monde en profite. De l'aide est généralement disponible dans les sources elles-mêmes, ou sur le site du projet, pour montrer comment compiler ce dernier. Dans le cas contraire, n'hésitez pas à contacter l'auteur du projet et à écrire de la documentation à ce sujet une fois votre éventuel problème résolu !

Ceci est la seule étape qui peut se faire sans communication. Pour tout le reste, et pour s'assurer que l'échange va se passer sans problèmes, il est vital de communiquer avec les membres du projet ! Eux seuls savent pour l'instant quel est l'avenir souhaité pour le projet, quels sont les objectifs, les défauts, et ce sont eux qui vont relire vos modifications. Voilà d'ailleurs un article intéressant au sujet de la communication dans le logiciel libre (en anglais), orientée sur le GSoC.

Une fois que cela est fait, cherchez quelque chose à faire. Une liste des bugs est généralement tenue, soit dans les sources dans le fichier BUGS, soit sur le *bug tracker*. Une liste des choses à faire peut par exemple être tenue dans le fichier TODO à la racine du projet. Vous pouvez aussi demander directement aux personnes responsables du projet, pour avoir des idées, ou leur demander quelles sont les priorités actuelles.

Si un site est mis en place, pensez à le parcourir, certaines de vos questions y trouveront peut-être réponses. Le site pourra contenir des informations qui vous aideront dans votre démarche. Les projets respectent en général des conventions de codage, peuvent avoir des documents expliquant la conception du projet, la façon dont les contributions ont lieu, des réponses aux questions souvent posées, etc. C'est un endroit à consulter avant de poser ses questions.

Les changements apportés doivent absolument être *atomiques*, c'est-à-dire ne faire qu'une seule chose, et la faire bien, sans artifices inutiles. Cela permet de vérifier que le changement est souhaitable, et surtout correct. C'est quelque chose de très important qui facilite beaucoup la gestion du travail à plusieurs, et permet de répondre à la question « Qu'est-ce qui a été modifié,

et quand ? ». La réponse à cette question est très utile lorsqu'on cherche par la suite à corriger des problèmes ou à comprendre un bout de code.

L'idéal pour le monsieur qui s'occupe du projet, c'est d'avoir votre changement dans les mains, et n'avoir qu'à l'incorporer dans sa copie du code. Le principe est donc de lui envoyer un *diff*, qui contient tous les changements apportés depuis la précédente version. Nous verrons plus tard comment faire cela en pratique. Vous réalisez donc vos changements, qu'ils soient importants ou non, vérifiez que tout vous semble correct en théorie (en relisant attentivement) et en pratique (avec des tests). Vous envoyez ensuite votre code à la personne en charge. Même si la modification fonctionne, il y a un certain nombre de raisons qui peuvent pousser à la refuser. Elle peut ne pas correspondre à la volonté du projet (toutes les fonctionnalités ne sont pas souhaitables, certaines images peuvent ne pas correspondre à la charte graphique), ou être un trop gros changement alors que la priorité est de corriger les problèmes existants pour sortir une nouvelle version. Mais rassurez-vous, si vous avez communiqué avant, vous pouvez être sûr que le changement sera souhaitable.

Il est aussi très possible que la modification ne soit pas acceptée d'emblée. La façon de faire peut ne pas être optimale, peut ne pas respecter les conventions de codage utilisées par le projet, et peut surtout poser des problèmes auxquels vous n'aviez pas pensé, étant donné que vous êtes nouveau dans le projet. Soyez préparé à ce genre de réponses, même si elles peuvent paraître sèches, le seul but est d'améliorer la qualité de votre proposition sans digressions « inutiles ». Si on vous demande d'apporter des modifications, essayez de comprendre en quoi votre travail n'était pas complet, et retravaillez-le. C'est un travail qui peut être long, surtout pour les premières fois. Une fois que tout le monde est satisfait, on peut incorporer la modification dans le projet, et ça y est, vous aurez réalisé votre première contribution à un projet libre !

Vous pouvez ensuite continuer à travailler en collaboration avec les personnes en charge du projet, envoyer quelques autres patchs pour aider. Au bout de quelques temps, il est possible que vous soyez estimé comme une personne de confiance : on vous donnera alors la possibilité d'envoyer vous-même vos modifications. Vous serez alors devenu un développeur *open source*, qui travaillez en équipe et dans laquelle vous puisez votre motivation. Vous pourrez ensuite vous occuper des nouveaux et les accueillir comme vous l'avez été.

La pratique

L'envoi d'un patch

Vous vous demandez peut-être à présent comment il est possible « d'envoyer des modifications », et comment est-ce qu'il est possible de garder les données synchronisées entre les différents développeurs. Cela s'applique à tout projet collaboratif, qu'il soit libre ou non. Le principe est de stocker les données dans un endroit qui contiendra la dernière version du projet. Tous les développeurs pourront alors s'y synchroniser régulièrement pour profiter des améliorations des autres, et être sûrs que personne ne se désynchronise. Le travail pour récupérer les changements des autres en conservant les siens pourrait alors être considérable.

À chaque fois qu'un changement est fait, un développeur pourra donc envoyer sa version. On utilise un terme anglais pour dire ça, *commit*, qui représente la validation de vos changements qui seront envoyés sur le serveur. Je vais garder les mauvaises habitudes prises et utiliser ce mot comme un nom français dans la suite du texte. Un logiciel nommé *diff* va s'occuper de regarder les différences entre la version locale et la version du serveur, et noter les modifications dans un fichier.

Code : C++

```
Index: plugins/plugin_utils/plugin_utils.h
===================================================================
--- plugins/plugin_utils/plugin_utils.h (revision 20449)
+++ plugins/plugin_utils/plugin_utils.h (revision 20450)
@@ -144,9 +144,9 @@
 inline bool isPrintable(const char c)
 {
-   if (isVisible(c) || isWhitespace(c)) {
-    return false;
-   }
-  return true;
+  return (isVisible(c) || isWhitespace(c));
 }

 const std::vector<std::string> bzu_standardPerms (void);
```

Ce code représente une modification faite sur un projet sur lequel je travaille, pour donner un exemple. J'ai retiré quatre lignes, et j'en ai ajouté une, pour corriger un bug et alléger le code. Ce fichier est ensuite envoyé sur le serveur, qui va chercher le fichier décrit par le diff : "plugins/plugin_utils/plugin_utils.h". Il vérifie que le fichier de diff est bien applicable à la version du serveur (il ne faut pas par exemple que quelqu'un ait modifié ce code avant moi), et l'applique. Ainsi tout le monde peut profiter de la correction.

Supposons que le logiciel utilisé pour synchroniser les développeurs entre eux soit Git et que Z soit sous GNU/Linux, pour regarder les modifications, il suffit de taper *git diff* dans les sources du projet. Z peut ensuite envoyer son diff à un mainteneur du projet.

Pour plus de détails sur l'envoi et la rédaction de patchs, merci de vous référer à l'article de bluestorm qui vous expliquera en détail comment faire accepter votre code dans un projet libre.

La gestion des bugs

La façon la plus rudimentaire de gérer les bugs, c'est d'avoir un fichier qui les liste. Cela permet aux utilisateurs d'avoir connaissance des bugs : s'ils en rencontrent un, ils peuvent savoir s'il est connu ou non, et sinon, en parler aux développeurs. C'est à ce moment-là qu'un outil pour gérer les bugs devient utile : les utilisateurs passent par un site web pour poster leur bugs, et les développeurs peuvent leur répondre. Voilà un exemple de *bug tracker*. C'est une simple liste de bugs non résolus, et une fois qu'ils sont corrigés (ou marqués comme non valides, etc.), ils disparaissent de la liste, pour laisser place aux bugs à résoudre.

C'est un moyen simple et pourtant efficace de gérer les bugs et assurer leur suivi, ne pas les oublier, et montrer à l'utilisateur qu'ils ont étés corrigés. D'autres outils plus puissants que celui-là existent, je me contenterais de citer Trac (qui permet d'ajouter d'autres attributs aux bugs, pour s'adapter aux besoins du projet) et Bugzilla, qui possède des fonctionnalités de bugs évoluées, permettant de lier les bugs entre eux, ou encore Quilt, etc. Trac est utile pour un projet de taille moyenne, et Bugzilla pour un projet de grande taille (citons Linux et KDE), ce qui le rend inutile pour des projets de petite taille.

Saviez-vous que le Site du Zéro utilise un système de ce type, nommé Redmine, qui permet notamment de gérer les bugs ?

En dehors des bugs, les utilisateurs peuvent également soumettre des patchs. Cela peut être lié au système de gestion des patchs, ou non. L'important est que les utilisateurs puissent en envoyer. Au pire, les mails peuvent être utiles, mais ils ne permettent pas aux utilisateurs de commenter les patchs.

Revue de code

C'est un processus qui permet d'augmenter la qualité du logiciel : faire en sorte que pour chaque modification de code, un certain nombre de personnes lisent ces modifications et vérifient que le *commit* n'apporte pas de problème. N'hésitez pas à souligner ceci dans votre projet, c'est quelque chose de très utile et qui augmentera la qualité du logiciel : vous aurez moins de bugs. De la même manière, des outils existent pour cela, même si dans un premier temps vous pouvez « imaginer » n'importe quelle façon de le faire ; l'important étant que ce processus ait lieu.

Moyens de communication

Il est important pour l'équipe de garder contact, et il existe bien sûr de nombreux moyens de faire cela. Vous connaissez sûrement la messagerie instantanée à travers des services comme Jabber ou MSN. Cela permet aux personnes du groupe de parler entre elles, mais c'est plus adapté à la communication entre deux personnes. Les projets libres aiment donc utiliser d'autres outils.

IRC

Le principe est le même que MSN par exemple, mais il est plus adapté à la conversation entre plusieurs personnes traitant d'un même sujet. Par exemple, les membres du projet Alpha (et leurs fans) peuvent se retrouver sur le canal #alpha et y discuter librement. Si tout le monde n'est pas là, ce n'est pas gênant, mais cela permet souvent une réponse rapide des membres présents. Si vous comptez utiliser IRC pour votre projet libre, pensez au réseau Freenode qui est fait pour ça. Si vous utilisez ce réseau pour autre chose qu'un projet libre, pensez à mettre deux # devant le nom, par exemple ##siteduzero.

Mailing lists

Le principe est simple : abonner tous les membres du projet à une *liste* pour leur permettre de discuter entre eux. Si un membre envoie un mail à la liste, tous les abonnés vont le recevoir et être capables d'y répondre. Cela permet de toucher tous les

membres du projet, mais le temps de réponse sera plus lent que sur IRC. À utiliser pour les discussions qui sont amenées à être plus longues, plus techniques ou nécessitant que tous les membres du projet y prennent part. La revue du code peut s'effectuer par mail par exemple.

Il existe bien sûr tout un tas d'autres moyens, citons par exemple les forums.

Tirer profit d'une forge

Comme vous pouvez le voir, de nombreux outils peuvent faciliter le développement d'un logiciel libre. Tous ne seront pas utiles, et il faut veiller à ce que leur mise en place ne soit pas un fardeau. C'est pour cela que des « forges » existent. Ce sont des sites Internet qui vous permettent d'héberger vos projets, en vous offrant un serveur de gestion des sources, un module pour gérer les bugs, les patchs, etc.

La forge qui est la plus utilisée actuellement (pour des raisons historiques), est SourceForge (sf.net). Elle offre un site web par projet, de quoi gérer vos sources avec CVS, SVN, Git, de quoi gérer vos bugs avec Trac ou leur système de gestion des patchs, présentés plus haut, des *mailing lists*, etc. Les libristes purs et durs voudront l'éviter en raison de son statut non libre.

TuxFamily est une forge française qui offre un site web, des *mailing lists* et de la gestion de code. Google Code offre en plus de la *review*, un wiki et un *bug tracker*. Il en existe tout un tas d'autres, notamment Gna!, Savannah, OcamlForge (dédié aux programmes OCaml), GitHub qui se démarque par son intégration avec Git, etc.
Vous savez désormais comment se passe de manière générale le développement des logiciels libres. À vous de comprendre pourquoi ce fonctionnement-là existe, comment l'améliorer pour vos projets, et continuer à apporter votre contribution au monde libre.

Toutes les suggestions et commentaires sont les bienvenus pour améliorer cet article.

Merci à bluestorm pour sa motivation et ses conseils, et à Poulpette pour tout son travail de zcorrection sur les tournures et les fautes de cet article.

Happy Hacking!

Partager

Ce tutoriel a été corrigé par les zCorrecteurs.