Cent fois sur le métier remettez vos correctifs… (Libres conseils 12/42)

Chaque jeudi à 21h, rendez-vous sur le framapad de traduction, le travail collaboratif sera ensuite publié ici même.

Traduction Framalang : ga3lig, Fred, peupleLa, LAuX, Goofy, jcr83, purplepsycho, Jej, Jean-Noël AVILA, Julius22, kalupa, 4nti7rust, lamessen, okram + Cédric Corazza

Écrire des correctifs

Kai Blin

Kai Blin est un bio-informaticien qui mène des recherches sur les antibiotiques dans le cadre de ses activités quotidiennes, tant sur ordinateur qu'au labo. Il est très heureux de pouvoir diffuser le logiciel développé dans le cadre de ses activités professionnelles sous licence open source. Vivant dans la charmante ville de Tübingen, dans le sud de l'Allemagne, Kai passe une partie de ses soirées sur l'ordinateur, à programmer pour le projet Samba. Il consacre la majorité de son temps libre restant au théâtre, où il participe aussi bien à la performance scénique qu'à la construction d'accessoires et à la régie technique dans les coulisses.

Écrire des correctifs et les proposer est souvent la première interaction concrète que vous pouvez avoir avec un projet *open source*. C'est la première impression que vous donnez aux développeurs présents. Proposer de « bons » premiers correctifs, ou tout du moins jugés comme tels par le projet auquel vous contribuez, vous rendra la vie plus facile. Les règles précises d'écriture du correctif, de la façon de le

soumettre au projet et tous les autres détails nécessaires vont sans doute varier selon les divers projets auxquels vous voulez contribuer. Mais j'ai trouvé quelques règles générales que l'on retrouve presque à chaque fois. Et c'est ce dont traite cet article.

Comment tout foirer

Le fil rouge de ce livre est « ce que j'aurais aimé savoir avant de commencer », aussi permettez-moi de commencer par l'histoire de mes premiers correctifs. J'ai été impliqué sérieusement dans l'écriture de code pour la première fois pendant le Google Summer of Code™ de 2005. Le projet Wine avait accepté que j'implémente un chiffrement NTLM basé sur des outils connexes à Samba. Wine est un projet à *committer* unique, ce qui signifie que seul le développeur principal, Alexandre Julliard, possède les autorisations de *commit* sur le

dépôt principal. En 2005, Wine utilisait encore CVS comme système de gestion de versions. Quand le projet a démarré et que j'ai reçu le courriel me disant que j'étais accepté, j'ai contacté mon mentor sur IRC et me suis mis au travail.

J'alignais joyeusement les lignes de code et bientôt j'ai pu implémenter les premières fonctionnalités. J'ai produit un correctif et l'ai soumis à mon mentor pour qu'il fasse une relecture. Au temps du CVS, il fallait renseigner toutes les options de *diff*(1) manuellement, mais je m'étais particulièrement documenté sur la partie cvs diff -N -u > ntlm.patch. De cette façon, j'avais le fichier que je pouvais envoyer à mon mentor. En fait, c'est quelque chose que j'ai bien fait. Et c'est la première chose que vous devriez prendre en compte quand vous préparez un correctif. Le résultat classique de la commande diff est sans doute plus facile à lire pour un ordinateur, mais je n'ai jamais rencontré un humain préférant le résultat classique au résultat d'un diff unifié. Grâce à l'option -u , diff utilise les notations **+ + +** et **- - -**

Par exemple, le diff qui suit est le résultat de la réécriture de « Hello, World! » en Python, version suédoise.

```
diff –git a/hello.py b/hello.py index 59dbef8..6334aa2 100644 --- a/hello.py +++ b/hello.py @@ -1,4 +1,4 @@ #!/usr/bin/env python # vim: set fileencoding=utf-8 -print "Hello, world!" +print "Halla, varlden!"
```

La ligne qui commence par « - » est la ligne supprimée, celle qui commence par « + » est celle qui va être ajoutée. Les autres lignes permettent à l'outil de correction de faire son travail.

J'ai envoyé le diff unifié que je venais de créer à mon mentor qui m'en a fait une *review*(2) en me signalant beaucoup d'éléments à modifier. J'ai effectué les corrections et lui ai renvoyé un nouveau diff peu de temps après. Le cycle d'analyse du code a continué durant toute la durée du GSoC et mon

correctif ne cessait de grossir. Quand la date de livraison est arrivée, je me suis retrouvé avec un correctif monstrueux dans lequel étaient inclus tous mes changements. Naturellement, j'ai eu beaucoup de mal à obtenir une review de mon correctif, sans parler de le faire accepter. Pour finir, Alexandre refusa de regarder plus avant le correctif tant que je ne l'aurais pas scindé. Les règles en usage chez Wine exigent que les correctifs soient de petites étapes logiques ajoutant une fonctionnalité. Chaque correctif doit faire quelque chose d'utile et pouvoir être compilé.

De fait, scinder un énorme correctif existant en différentes parties cohérentes individuellement et qui peuvent être compilées représente beaucoup de travail. C'était même d'autant plus de travail que je ne connaissais qu'une seule manière de le faire : écrire un petit correctif, créer le diff, le proposer à la validation, mettre à jour ma contribution locale et écrire alors le correctif suivant. Peu de temps après que j'ai commencé à envoyer mes premiers petits correctifs, Wine est entré dans une phase de gel des fonctionnalités d'un mois menant à la version 0.9.0 bêta. Je suis resté sur le correctif suivant pendant un mois avant de pouvoir continuer et j'ai finalement envoyé mon dernier correctif en novembre. Complètement frustré par cette expérience, j'ai décidé que je ne voulais plus jamais avoir à faire avec la communauté Wine.

Ma frustration perdura jusqu'à ce que des personnes qui utilisaient réellement mon code commencent à me poser des questions sur celui-ci en février 2006. Mon code était vraiment utile ! Ils voulaient également davantage de fonctionnalités. Quand Google annonça qu'il reconduirait le GSoC en 2006, mes projets pour l'été étaient clairs. Maintenant que Wine avait basculé de diff à git en décembre 2005, je savais que je ne serais pas ennuyé par des gels de fonctionnalités, puisque je pouvais finalement créer tous mes petits correctifs localement. La vie était belle.

Ce n’est que lorsque je suis tombé sur une interface de git (appelée porcelaine dans le langage git) qui émulait le comportement de Quilt que j’ai appris qu’il y avait des outils qui auraient pu rendre ma vie plus facile, même en 2005.

Comment NE PAS tout foirer

Maintenant que je vous ai raconté comment j’ai réussi à me planter avec l’envoi de correctifs, permettez-moi de poursuivre avec quelques conseils pour éviter les pièges.

Règles pour la soumission de correctifs

Mon premier conseil est de lire attentivement toutes les directives de soumission de correctifs que peut avoir le projet auquel vous voulez contribuer. Celles-ci, ainsi que les normes de style de codage, doivent être consultées *avant même* de commencer à coder.

Des diffs unifiés

Même si ce n’est pas explicitement indiqué dans les directives de soumission des correctifs, vous devez vraiment, vraiment envoyer le résultat d’un diff unifié. Je n’ai encore rencontré aucun projet qui préfère la sortie non unifiée du diff. Les diffs unifiés rendent la révision du correctif beaucoup plus facile. Ce n’est pas par hasard que les programmes de gestion de version modernes utilisent automatiquement ce format dans leurs commandes diff par défaut.

Utilisez un contrôle de version distribué

En ce qui concerne la gestion de versions moderne, vous devriez vraiment utiliser un système de gestion de versions distribué (DVCS) pour travailler localement sur le code. Git et Mercurial sont les choix les plus populaires de nos jours, quoique Bazaar puisse aussi très bien faire l’affaire. Même si le projet auquel vous voulez contribuer utilise toujours un

système de gestion de version centralisé, être en mesure d'envoyer vos changements de manière itérative est une très bonne chose. Tous les outils de gestion de versions distribués mentionnés ci-dessus devraient être capables d'importer des changements depuis un SVN ou un CVS. Vous pourrez y aller et apprendre doucement Quilt mais, sérieusement, le futur passe par les systèmes de gestion de versions distribués.

De petits correctifs, pour faire une chose à la fois

Quand je dois examiner des correctifs, les plus ennuyeux à traiter sont ceux qui sont trop gros ou qui tentent de faire de nombreuses choses en même temps. Les correctifs qui ne font qu'une chose à la fois sont plus faciles à traiter. Au final, ils vous faciliteront la vie quand vous devrez déboguer les erreurs qu'auront manquées à la fois l'auteur et le vérificateur.

Suivez votre correctif

Après avoir proposé votre correctif, gardez un œil sur les canaux de communication du projet et sur votre correctif. Si vous n'avez eu aucun retour au bout d'une semaine, vous devriez poliment en demander un. En fonction de la façon dont le projet gère les propositions de correctif, celui-ci pourrait être noyé dans le bruit. N'espérez pas voir votre correctif retenu du premier coup. Il faut, en général, quelques essais pour s'habituer au style d'un nouveau projet. En tant que contributeur néophyte, personne ne vous en voudra pour ça, à condition d'avoir presque tout bon. Assurez-vous seulement de corriger ce que les développeurs vous ont indiqué et envoyez une seconde version du correctif.

Conclusion

Écrire de bons correctifs n'est pas difficile. Il y a deux ou trois choses à prendre en considération. Mais après en avoir écrit quelques-uns vous devriez être au point sur celles-ci. Un système moderne de contrôle de version (distribué) et le *workflow* (Ndt : flux de production) qui en résulte gèreront de fait la plupart des choses que j'ai mentionnées. Si vous n'avez qu'une chose à retenir, c'est ceci :

- Utilisez un système de contrôle de version distribué pour gérer vos correctifs.
- Écrivez vos correctifs en petites étapes indépendantes.
- Suivez les normes de codage en vigueur.
- Répondez rapidement aux commentaires sur vos correctifs.

Les quelques lignes directrices ci-dessus devraient vous aider à bien faire la plupart des choses, sinon toutes, quand vous soumettrez vos premiers correctifs. Bon codage.

(1) Un diff (abréviation de différence) est un fichier qui affiche le résultat d'une comparaison entre deux éléments (en général, des lignes de code). Pour en savoir plus : http://fr.wikipedia.org/wiki/Diff

(2) *Review* : révision minutieuse http://fr.wiktionary.org/wiki/review