http://clx.asso.fr/spip/?Comment-la-GPL-Maintient-Linux

Comment la GPL Maintient Linux unifié et fort

- Espace membres - Points de vue -

Date de mise en ligne : lundi 9 décembre 2002

L'histoire d'Unix est navrante pour tous ceux qui regrettent qu'un grand systÚme d'exploitation ait ainsi échoué à s'imposer comme standard unifié. Le sort d'Unix est bien connu. ?parpillés par les commerciaux en une multitude de versions incompatibles, les différents Unixes ne sont plus aujourd'hui en concurrence qu'entre eux pour se disputer les parts d'un marché Unix en diminution

Cette histoire pourrait-elle se répéter pour Linux ? Il est vrai que la licence GPL (GNU General Public License) offre aux développeurs Open Source le droit explicite d'utiliser et transformer n'importe quel projet sous GPL, ouvrant ainsi la porte à une "divergence de code" potentiellement nuisible. Pourtant, apparent paradoxe, les développeurs Open Source considèrent ce droit à la divergence comme un mécanisme de protection contre le risque réel de divergence. Et ils fustigent l'absence de ce droit dans la licence SCSL (Sun Community Source License) utilisée pour Java, Solaris et Jini, considérant SCSL comme une piètre alternative à la GPL, pourtant plus permissive.

C'est un défi pour ceux qui veulent implanter Linux en milieu professionnel : il leur faut sans doute se battre pour surmonter les craintes des utilisateurs que GNU/Linux ne s'éparpille en des centaines de versions incompatibles du fait de l'absence de prise en charge par une unique grande société. Et voici où je veux en venir, demandant s'il n'est pas extravagant que les licences Open Source garantissent à quiconque le droit de faire précisément ce que l'on souhaite éviter.

Cela vous semble contradictoire ? C'est vrai, c'est une réponse rapide et grossière. Le détail arrive : Linux ne se divisera pas parce que cela représenterait pour le diviseur un travail excessif pour un profit nul : les améliorations intéressantes seraient absorbées par le courant principal tandis que le reste de la branche divergente, inutile, serait écarté ou ignoré.

Ce phénomène se constate pour Linux, malgré que ce n'ait pas toujours été le cas pour des projets antérieurs, et c'est là précisément l'effet de la licence du code-source de Linux.

DES EXEMPLES REMARQUABLES DE DIVERGENCES

1. Unix â€”> des dizaines d'Unix "propriétaires"

Si vous connaissez l'histoire d'Unix, vous savez qu'Unix était un

produit accidentel des laboratoires Bell d'AT&T, vers 1969. Je vais négliger beaucoup de détails de cette longue histoire, mais ce qu'il y a de plus important à connaître est qu'AT&T était à l'époque sous le coup d'une mesure de justice antitrust (qui a expiré vers 1980) qui lui interdisait d'entrer dans le secteur des ordinateurs et du logiciel. Se trouvant ainsi dans l'impossibilité légale de vendre Unix, AT&T vendit des licences pour son code-source (et aussi par la même occasion le droit d'utiliser le nom déposé "Unix") à des universités comme celle de Berkeley, ainsi qu'à des sociétés privées comme IBM, Apple, DEC, Data General, SGI, SCO, HP, etc.

Ces compagnies obtenaient ainsi le droit de concevoir leurs propres Unixes : IBM édita AIX, Apple sortit A/UX. DEC fit Ultrix, OSF/1 et Digital Unix (renommé plus tard "Compaq Unix" et maintenant "Compaq Tru64"). DG/UX fut créé par Data General, IRIX par SGI, HP/UX par HP, et SCO sortit XENIX qui se transforma en fin de compte en SCO UNIX. On pourrait en citer encore d'autres que je vous épargnerai.

Le problème, c'est que ce sont ces types qui ruinèrent Unix. Chacun d'eux vantait son Unix mutant comme un "Unix plus" â€” tout ce que les autres ont et plus encore. Ayant besoin de différenciation, c'est délibérément qu'ils rendirent leurs Unixes incompatibles, alors même qu'en paroles ils soutenaient les "standards".

Pour les utilisateurs, la messe était dite, et Microsoft s'engouffra dans la brèche de la désunion comme un char Sherman.
C'est l'exemple classique de divergence, celui qui vient tout de suite à l'esprit des gens.

2. BSD â€”> FreeBSD, NetBSD, OpenBSD, BSD OS, MachTen, NeXTStep (qui s'est récemment transformé en Apple Macintosh OS X Server), et SunOS (appelé maintenant Solaris)

Les mesures antitrust limitaient donc AT&T qui, ne pouvant vendre Unix, diffusa des licences très bon marché de son code-source à des universités, notamment Berkeley. Berkeley joua le rôle de meneur dans le monde universitaire : ayant accès au code-source, ils réalisèrent rapidement qu'ils pouvaient le réécrire pour l'améliorer, ce qu'ils firent peu à peu. Leur version était qualifiée de "BSD" (Berkeley Software Distribution) et ils étaient heureux de la partager avec quiconque ayant comme eux une licence d'AT&T pour les sources Unix.

Or leur travail était en général largement meilleur que celui des laboratoires Bell, en partie du fait qu'il bénéficiait des retours des universités du monde entier, d'une façon très proche du modèle Open Source. Au bout de quelques années, sans qu'il y ait eu - de prime abord - d'intention délibérée, la quasi-totalité du code d'AT&T était réécrit.

Un beau jour, Keith Bostic, un étudiant inspiré par le projet GNU de Richard M. Stallman (vous voyez de qui je parle ?), vint trouver les responsables du développement de BSD et leur proposa de remplacer ce qu'il restait du travail d'AT&T dans BSD afin de créer un BSD vraiment libre. Craignant des problèmes avec AT&T, les responsables tentèrent de se dérober en soulignant à Bostic, afin de le dissuader, l'énormité de la tâche consistant à réécrire des fonctions clé de BSD. Cet argument se retourna cependant contre eux car c'est exactement ce que Bostic fit, et rapidement. Alors, ils durent terminer le travail en bougonnant. Ils essayèrent ensuite d'éviter qu'AT&T ne se rende compte de ce qu'ils avaient fait.

Bien sûr AT&T s'en rendit compte, s'en alarma et attaqua en justice. Encore une longue histoire sur laquelle il vaut mieux passer. Sous la pression du procès, le freeware BSD se scinda en trois camps : FreeBSD, NetBSD, et OpenBSD. Mais il y avait aussi des branches privées car la licence BSD de Berkeley en autorisait la création. SunOS de Sun Microsystems, MachTen de Tenon Intersystems, BSD OS de BSDI et NeXTStep de NeXT Computer furent commercialisé sans accès public au code-source, alors qu'ils étaient pourtant tous basés sur le code-source BSD de Berkeley.

Notez bien la différence : si vous écrivez un programme et que vous mettez en circulation son code-source sous licence GPL (GNU General Public License), quelqu'un d'autre qui vendrait ou distribuerait des travaux dérivés du vôtre se trouve dans l'obligation d'en publier le code-source sous les conditions de la GPL. Ce n'est pas le cas si vous publiez sous licence BSD : n'importe qui peut alors créer une version dérivée de votre travail et refuser d'en publier les modifications de code-source. En d'autres termes, il est autorisé à créer des dérivés "propriétaires".

Un mot à propos des trois variantes de BSD libres : tous étaient des groupes divergents d'un projet maintenant éteint appelé 386BSD. S'ils avaient envisagé un temps de refusionner afin d'éviter le travail en double, ils continuent à l'heure actuelle séparément car ils se sont spécialisés : FreeBSD recherche la plus grande stabilité possible sur les processeurs Intel x86, NetBSD tente de fonctionner sur autant de types de processeurs différents que possible alors que OpenBSD vise la sécurité la plus stricte possible. En d'autres termes, le projet 386BSD reste divisé car il existe des raisons convaincantes faisant que tout le monde a à y gagner.

Mais, là où c'est possible, ces trois projets frères collaborent sur les tâches les plus coriaces - et ils collaborent également avec les programmeurs GNU/Linux. Quelques-uns des meilleurs pilotes de matériels du noyau Linux sont en fait des pilotes BSD. Il existe un haut niveau de compatibilité entre les trois BSDs, ainsi qu'entre eux et GNU/Linux : à la différence des vendeurs d'Unix "propriétaires", les programmeurs BSD et GNU/Linux sont enclins à éliminer les incompatibilités et à

encourager les standards.

3. emacs â€”> Lucid emacs â€”> xemacs, autres emacses "propriétaires" pour la plupart oubliés, maintenant â€”> GNU emacs

Appeler emacs "éditeur de texte", c'est un peu comme appeler "bateau" le paquebot Queen Elizabeth II. Ce programme n'est en principe qu'un simple programme de manipulation de texte conçu pour traiter des macros (d'où son nom emacs : éditeur de macros) mais vous pouvez passer tout votre temps et accomplir à peu près tout ce qu'on peut faire avec un ordinateur sans jamais quitter le programme.

Il a été écrit autrefois par le légendaire programmeur Richard M.Stallman (vous voyez qui je veux dire ?), à une époque où quiconque entrant en relation avec Stallman était présumé partager avec tous le code-source qu'il produisait, et le faisait la plupart du temps, d'ailleurs. En fait, la seule licence pour les premières versions de emacs était un tacite "Faites ce qu'il vous plaît".

Malheureusement pour Stallman, nombre de sociétés voulaient utiliser son travail, et en faire des dérivés "propriétaires". Il en fut surpris et déçu, et ce fut là l'une des raisons pour lesquelles il rédigea le texte de la licence GNU GPL, puis initia le projet GNU (dans le but de créer un système d'exploitation à la Unix, appelé "GNU", et dont le caractère libre soit garanti de façon pérenne).

Les différents emacs propriétaires sont tous morts aujourd'hui, à une exception près : "Lucid Emacs" fut vendu à Sun Microsystems par Lucid Corporation, avant sa disparition. Sun décida de le renommer "xemacs", appellation portant à confusion (car xemacs n'intègre pas plus de support particulier pour le système X Window que les autres emacses), pour en fin de compte en faire cadeau à quelques programmeurs qui décidèrent de remettre son code-source en circulation comme logiciel libre sous la licence GNU GPL de Richard M. Stallman.

Pendant ce temps, Stallman poursuivait le développement de l'emacs original sous le nom de "GNU emacs", et plaçait la totalité de ses nouveaux travaux sous la licence GNU GPL, précisément pour éviter qu'une catastrophe comme celle de Lucid ne puisse se reproduire.

On se demande souvent pourquoi les gens de xemacs ne fusionnent pas de nouveau leur travail avec la version de Stallman, pour éviter des tâches qui doublonnent. Les obstacles sont à la fois personnels et techniques, et parfois difficiles à distinguer. D'abord, Stallman est vraiment un autocrate. Peut-être d'ailleurs que seul quelqu'un de vraiment difficile et obstiné était à même d'accomplir autant que lui, bâtissant tout un monde logiciel libre à partir de presque rien. Par ailleurs, le code-source de xemacs a été écrit en suivant les principes de programmation orientée objet, ce qui rend aisément possible à des

développeurs nombreux de se répartir les responsabilités. Le code-source de GNU emacs, au contraire, est de type classique, et il est tellement vaste que seul un programmeur génial pouvait espérer le développer. Ce programmeur-là, seul parmi des milliards, c'est naturellement Richard M.Stallman.

4. NCSA httpd â€"> serveur Web Apache

De nos jours, le standard mondial du logiciel de serveur Web s'appelle Apache, maintenu par un groupe entièrement constitué de bénévoles - ce qui ne signifie pas qu'ils ne gagnent pas d'argent : lorqu'ils sont sollicités comme consultants à propos du Web, des membres du groupe Apache comme Brian Behlendorf se voient proposer un pont d'or, et ces sollicitations proviennent d'une renommée bien méritée.

Mais, avant qu'il n'y ait Apache, vous lanciez soit "NCSA httpd" (HyperText Transport Protocol daemon) du National Center for Supercomputing Applications de l'Université de l'Illinois à Urbana-Champlain, soit "CERN-httpd" du CERN de Genève. Le daemon NCSA était plus petit et plus rapide, mais celui du CERN était réputé, principalement parce qu'il était associé au nom du créateur du Web, Tim Berners-Lee, qui travaillait comme chercheur au CERN.

"CERN-httpd" (appelé plus tard "W3C httpd") fut toujours un logiciel du domaine public (c'est à dire que personne n'en était propriétaire). N'étant plus maintenu, c'est un projet mort. On ne sait pas de manière claire ce qu'était la licence de NCSA-httpd à l'origine, mais quand le projet mourut (1996) sa licence était du type "gratuit pour un usage non commercial seulement".

De toutes façons, des programmeurs d'un groupe on-line qui avaient conçu des patches (modifications) pour le NCSA-httpd décidèrent soudain de produire en 1995 leur propre variante, faisant diverger le code. "Apache" n'était à l'origine que le nom de code provisoire donné par Brian Behlendorf au projet, mais ses amis développeurs en soulignèrent la justesse ("a-patchy" server = "apache" ; vous saisissez ?), et le nom demeura.

C'est un bon exemple de comment et pourquoi les projets Open Source divergent, sans malice, pour de bonnes raisons : le développement au NCSA s'était arrêté après que le créateur original, Rob McCool, ait quitté le Centre. Si c'était arrivé à un produit "propriétaire", il aurait disparu en laissant tous ses utilisateurs sur le carreau. Or, compte tenu de l'utilité du produit, le Projet Apache récupéra le code-source et continua d'aller de l'avant. Apache surpasse maintenant tous les serveurs Web, sans se soucier de leur marketing et de leurs budgets de développement.

5. gcc â€"> pgcc â€"> egcs â€"> gcc

Voilà un exemple bizarre. Richard M. Stallman (vous vous souvenez ?) fonda en 1984 le Projet GNU, qui produisit l'incomparable compilateur C nommé GNU C Compiler ("gcc"). gcc est conçu pour fonctionner sur n'importe quel ordinateur vaguement plausible, pas seulement les séries Intel x86. Et puis d'autres priorités conduisirent peut-être à différer l'amélioration du support Intel. En tout cas, en 1997, le mieux que gcc pouvait faire en matière d'optimisation du code sur plate-forme Intel était de régler le compilateur pour les puces 486. Stallman recevait supplique sur supplique pour que soit développée l'optimisation pour le Pentium, mais il les ignorait obstinément.

Alors, un groupe se créa spécialement pour le compilateur Pentium (Pentium Compiler Group, auquel participait la société CYGNUS Corporation que venait juste d'acheter Red Hat Software, Inc.). Le groupe développa tout d'abord une variante très rapide de gcc appelée "pgcc" (Pentium gcc), puis pour enterrer la hache de guerre avec Stallman développa "egcs" (Experimental GNU Compiler System), destiné à être réabsorbé par gcc.

Pour une raison ou une autre, la Free Software Foundation de Stallman (les développeurs du Projet GNU) continua d'agir comme si egcs n'existait pas. Mais des distributions GNU/Linux basées sur egcs commencèrent d'apparaître, et le monde du logiciel libre se mit à progressivement délaisser gcc.

Ceci peut être vu comme une variante de l'expérience d'Apache. La capacité à diverger signifie qu'un développeur qui refuse d'aller de l'avant ne peut bloquer le progrès : quelqu'un d'autre pourra, aussi courtoisement que possible, assumer le rôle de leader et si nécessaire faire diverger le projet.

Le coup de force fut évité dans le cas de egcs. En avril 1999, la FSF réintégra egcs dans la (future) branche principale de gcc, et en confia tout développement futur à l'équipe egcs, résolvant ainsi le conflit.

6. glibc â€"> Linux libc â€"> glibc

C'est un cas presque exactement inverse. Tout système Unix repose de façon extrême sur une bibliothèque de fonctions essentielles appelée bibliothèque C ("C library"). Le Projet GNU de Richard M.Stallman (vous vous rappelez de lui ?) entreprit dans les années quatre-vingts l'écriture de la GNU C library, ou glibc. Quand Linus et ses amis programmeurs commencèrent à travailler sur le système GNU/Linux (en utilisant le noyau "Linux" de Linus), ils cherchèrent des bibliothèques C libres, et choisirent celle de Stallman. Cependant, ils trouvèrent que la bibliothèque de Stallman (qui en était à la version 1-point-quelque_chose) n'était pas suffisamment rapide, estimèrent qu'ils pouvaient l'adapter eux-mêmes au noyau Linux, et décidèrent alors d'en faire diverger leur propre version, estampillée "Linux

libc". Ils poursuivirent leur effort au cours des versions 2.x, 3.x, 4.x et 5.x mais, en 1997-98, ils constatèrent quelque chose de bizarre : la glibc de Stallman, bien qu'encore dans les numéros de version 1-point-quelque_chose, avait développé des qualités stupéfiantes : ses fonctions internes disposaient d'un système de repérage des versions autorisant l'ajout de nouvelles versions sans perdre le support des plus anciennes applications, elle supportait mieux le multilingue, et permettait de volumineux enchaînements d'exécution [NdT].

Les programmeurs GNU/LINUX décidèrent alors que, même si leur variante avait paru une bonne idée, cela avait été une erreur stratégique. Ajouter les améliorations de Stallman à leur version divergente aurait été possible, mais il était plus facile de simplement se remettre au standard de la glibc. Et ainsi les bibliothèques glibc 2.0 et ultérieures ont progressivement été adoptées comme bibliothèque C standard par les distributions GNU/Linux.

Le numéro de version était un problème mineur : les gens de GNU/Linux avaient déjà atteint 5.4.47, tandis que Stallman touchait juste au 2.0. Ils envisagèrent probablement l'espace d'une milliseconde de demander à Stallman de numéroter sa prochaine version 6.0 pour les arranger. Puis ils se mirent à rire en disant : "C'est de Stallman que nous parlons, n'est-ce pas ?", et décidèrent que tenter d'aller contre l'obstination de Richard n'était une idée judicieuse. Et c'est pour cela que, par convention, la version 6.0 de la libc Linux et la glibc 2.0 sont identiques.

7. Sybase â€”> Microsoft SQL Server

Woody Allen a dit : "Le lion peut s'étendre à côté de l'agneau, mais l'agneau ne va pas beaucoup dormir". On peut dire à peu près la même chose des sociétés qui ont conclu des "alliances industrielles" avec Microsoft Corporation. L'une des sociétés à commettre cette erreur a été Sybase Corporation, l'éditeur du langage de gestion de bases de données "Sybase Structured Query Language" (SQL) pour de nombreux Unixes et pour NetWare. Selon cet accord, Microsoft vendait Sybase à ses clients, sous l'étiquette Microsoft SQL Server, et avait libre accès au code-source de Sybase sous réserve d'une clause de non-divulgation.

Bien sûr, comme on pouvait le prévoir, Microsoft rompit l'accord lorsqu'il eut appris de Sybase tout ce qui était possible, et relança Microsoft SQL Server comme son produit maison, concurrent de Sybase. Je ne sais pas si les versions actuelles de MS SQL Server sont réécrites de zéro ou si elles comportent du code Sybase sous licence, auquel cas ce ne serait pas un cas légitime de divergence (laissons de côté Open Source), mais c'est suffisamment similaire pour m'avoir incité à en parler.

ANALYSE : POURQUOI LA DIVERGENCE EN OPEN-SOURCE EST À LA FOIS PEU FRÉQUENTE ET SALUTAIRE

Vous, lecteur, pouvez faire diverger n'importe quel projet Open Source quand vous le voulez. Mais il n'y a pas à s'en inquiéter. Prouvons-le. Téléchargez une copie du noyau Linux courant depuis ftp://ftp.kernel.org/. Renommez-le Bidon OS par exemple, puis expédiez des messages dans tous les endroits auxquels vous pouvez penser annonçant que Bidon OS s'est séparé de Linux, et tout le bénéfice que l'on peut attendre de cette situation.

Puis attendez les réactions. Attendez encore un petit peu. Écoutez le tic-tac de l'horloge. Triez votre collection de peluches. Ouvrez alors l'arborescence du code-source, pensez un peu à tout ce que pourriez faire avec, et essayez d'imaginer comment vous aller bien pouvoir faire pour en trouver le temps.

Bon, c'est un peu injuste. Vous n'êtes probablement pas un programmeur. Mais imaginons que vous en êtes un. Vous êtes un programmeur ninja avec un puissant code-fu, toute l'énergie fondamentale pour réussir et une équipe disciplinée de partisans enthousiastes férus de programmation. Et donc vous ne faites pas qu'écouter le tic-tac de l'horloge, vous abattez un sacré bon travail. Et, en fin de compte, vous améliorez le noyau de façon sensible.

Alors les gens de Linux sourient largement, et vous disent sincèrement "Merci beaucoup." Comme tous les programmeurs du monde, ils savent que la programmation est un travail difficile. Ils sont paresseux, d'une façon constructive, c'est à dire qu'ils ne sont pas orgueilleux et sont ravis d'utiliser le travail des autres â€” quand c'est autorisé.

Oh ! Vous aviez oublié que votre travail était sous licence GPL ? Travaillant sur un programme dérivé d'une production sous GPL (le noyau Linux), vous êtes contraint de publier vos améliorations sous GPL également, afin que chacun puisse librement les utiliser. C'est pourquoi vous étiez le seul à croire travailler à la création de Bidon OS : en réalité, vous travailliez à améliorer Linux.

Et voilà pourquoi les divergences sont rares courantes en code OpenSource, et encore moins (précisément) si le code est sous licence GPL : les perfectionnements amenés par un groupe soi-disant divergent sont librement disponibles pour la communauté entière.

Mais, comme le montrent la plupart des exemples non-GPL ci-dessus, la divergence n'est pas toujours un choix. C'est parfois une soupape de sécurité vitale dans le cas ou les développeurs en charge du projet arrêtent de travailler ou encore décident d'en interrompre la

progression. Le fait que d'autres puissent s'en saisir est une bonne chose.

Il existe une troisième raison qui explique l'existence de divergences et qui est susceptible d'atteindre la communauté GNU/Linux en fin de compte : la spécialisation. Vous vous rappelez que c'est ce qui s'est passé avec les trois versions de BSD libre â€" quoique le choc de titans du procès entre AT&T et Berkeley rende cette situation-là singulière.

Mais c'est comme ça, n'importe qui peut un jour proposer à l'équipe en charge du noyau Linux une quelconque extension hors du champ du projet. Si des ressources associées sont créées et s'il y a des raisons suffisantes à son existence, cette branche progressera.
Dans ce cas, Linux divergera â€" et ce sera une bonne chose car il y aura alors deux projets forts plutôt qu'un seul, chacun d'entre eux se concentrant sur un rôle important que l'autre ne peut remplir.

Et si cela arrive, les branches divergentes partageront certainement leur code-source et leurs informations, exactement comme le font les différentes variantes de BSD, parce qu'il est logique de procéder ainsi.

Le monde en sera plus riche, grâce à la divergence et au partage.

Rick Moen
Légèrement modifié par Zack Brown et Tom Davey.

Rick Moen (rick@linuxmafia.com) est un membre très actif de la communauté Linux. Auteur établi à San Francisco, il était auparavant
responsable de l'administration système chez Linuxcare.

Adaptation française Jean Peyratout
27 novembre 1999

Post-scriptum :

Article original sur http://www.linuxmafia.com/ rick/ess...
Traduction : Jean PEYRATOUT de l'ABUL. Diffusé sur la liste du CLX, et publié sur le site le 28 Novembre 1999.

Voir aussi à ce propos la chronologie des systÚmes Unix.

[NdT] ? glibc "supported multiple execution threads"

