N°69 NOVEMBRE - DÉCEMBRE 2013

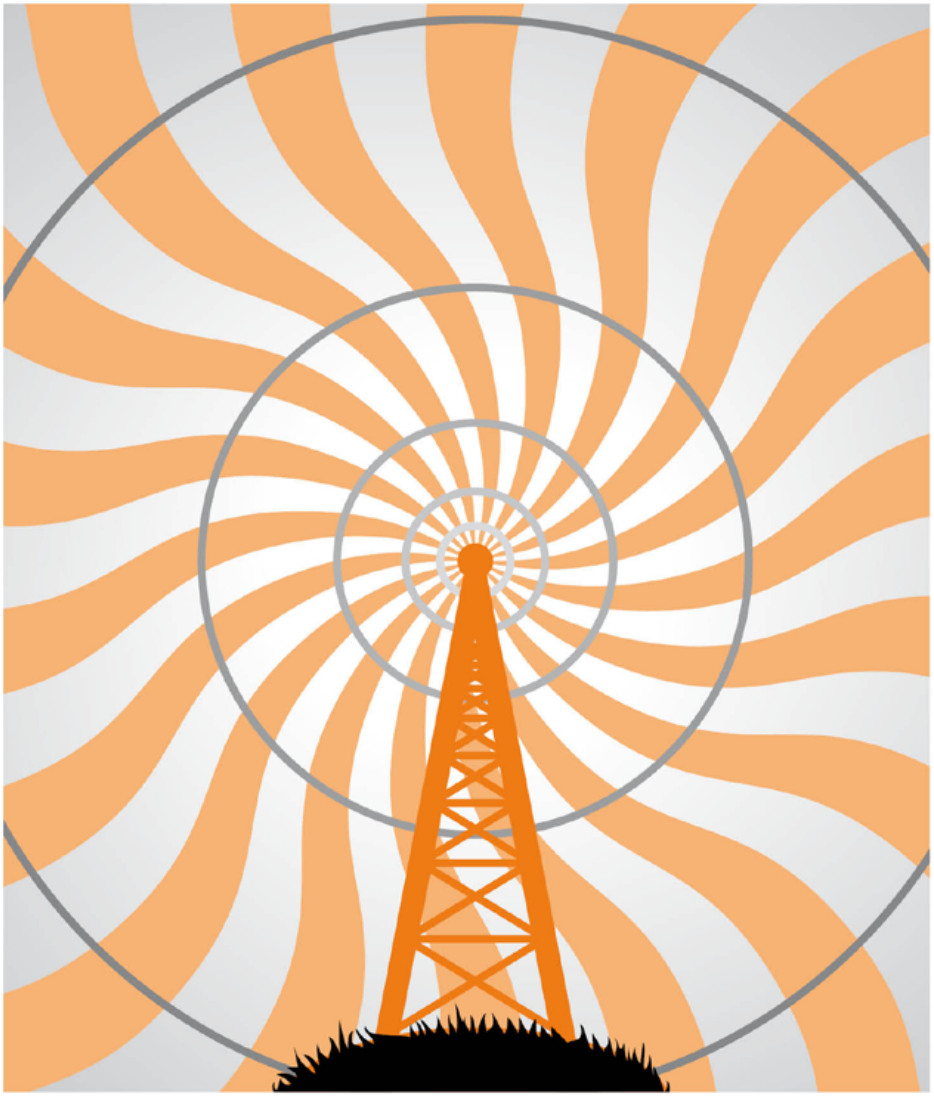
GNU LINUX MAGAZINE / FRANCE HORS SÉRIE

L 15066 - 69 H - F: 8,00 € - RD

ADMINISTRATION ET DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE ET EMBARQUÉS

TOUTES LES BONNES PRATIQUES POUR FAIRE CONNAÎTRE ET EXPLOITER AU MIEUX...

VOTRE PROJET OPEN SOURCE



France METRO : 8 € – CH : 13,80 CHF – BEL/PORT.CONT : 8,90 € – DOM : 8,50 € – CAN : 13,99 $ cad – NCAL/S : 1100 CFP – POL/A : 1600 CFP – TUNISIE : 17,40 TND – MAR : 98 MAD

DÉPLOYEZ VOS PROJETS OPEN SOURCE
avec CloudStack™ Instances operated by Ikoula

PLATEFORME PACKAGÉE
Ressources utilisables à l'heure, sans limitation : routeur, load balancer, firewall, IP.

CLOUDSTACK INSTANCES
Déployez votre Cloud Public avec des instances et des ressources à la demande.

APIs OUVERTES
Bénéficiez d'une interopérabilité avec des infrastructures existantes sans surcoût.

INTERFACE UNIQUE
Gérez votre réseau de Cloud Public en self-service.

EN SAVOIR PLUS SUR : http://express.ikoula.com/cloudstack

*Test valable, sans engagement de durée, une fois par compte client pour les 15 premiers jours à partir de la date de création du compte CloudStack Instances. Passée cette période toute consommation vous sera facturée au tarif en vigueur.

ikoula
Hosting Services

01 84 01 02 50
sales@ikoula.com

NOM DE DOMAINE | MESSAGERIE | HÉBERGEMENT | CERTIFICAT SSL | CLOUD | SERVEUR DÉDIÉ

SOMMAIRE N°69

INTRODUCTION

DÉVELOPPEMENT

COMMUNAUTÉ

DOCUMENTATION

ABONNEMENTS

ÉDITO

Une idée de projet vient de germer dans votre esprit ? Vous codez depuis plusieurs mois, à vos heures perdues, tout seul dans votre coin, et vous vous dites qu'un retour (positif comme négatif) sur votre développement serait le bienvenu ? Votre projet est mûr et fonctionnel, vous aimeriez le faire connaître, le partager ? Alors, qu'attendez-vous pour vous lancer ?

Eh non, ce n'est pas facile de se faire une place sur le marché, parmi la myriade de projets qui existent déjà. D'autant plus que dans le monde de l'open source, la pluralité des solutions dans chaque domaine nécessite de toujours trouver un détail qui doit faire toute la différence, sous peine de susciter peu d'intérêt chez les utilisateurs...

Mais en prenant ce magazine entre les mains, vous êtes déjà sur la bonne voie ! En effet, ce nouveau numéro hors-série de *GNU/Linux Magazine* vous aide à faire le point sur votre projet et sur les moyens de le faire connaître et même, pourquoi pas, d'en tirer quelques revenus. On commencera par aborder toutes les questions à se poser *avant* de vous lancer tête baissée dans l'arène, à savoir le choix du financement et de la licence sous laquelle distribuer votre projet. Tout ceci doit en effet être mûrement réfléchi, bien en amont de votre projet.

Suivent plusieurs articles qui vous permettront non pas de mieux développer, mais plutôt de mieux *encadrer* vos développements : gestion de versions, gestion de bugs et intégration continue sont donc au programme de cette partie pour assurer un suivi rigoureux de votre projet.

Un point crucial ensuite concerne toute la partie « sociale » de votre projet : vous faire connaître auprès du plus grand nombre et susciter la curiosité. Autrement dit, comment communiquer autour de votre projet et aussi, comment le distribuer ou le mettre aisément à disposition des utilisateurs potentiels ?

Et enfin, n'oubliez pas que la qualité d'un projet se mesure bien souvent par la qualité de sa documentation ! Celle-ci reflète en effet la volonté du développeur de rendre son projet accessible, d'anticiper les difficultés des utilisateurs et régulièrement mise à jour, elle témoigne également de la « bonne santé » du projet, ce qui inspire davantage la confiance.

Si vous suivez l'ensemble des conseils et bonnes pratiques évoqués dans ce numéro, nous ne pouvons certes pas vous garantir la réussite, mais au moins vous assurer que vous mettez toutes les chances de votre côté pour parvenir à vos fins !

La rédaction

PS : Cher lecteur, vous avez peut-être déjà eu le plaisir de découvrir nos quelques numéros hors-séries sous forme de « magalivres » ou « mooks ». Aujourd'hui, nous avons le plaisir de vous annoncer que l'ensemble des hors-séries de *GNU/Linux Magazine* vous seront désormais toujours proposés dans ce format haut de gamme ! L'occasion pour vous d'alimenter ainsi votre bibliothèque avec une série de guides thématiques de grande qualité, à lire et à relire...

Rendez-vous début janvier 2014 pour notre prochain guide dédié au langage C !

GNU/Linux Magazine France Hors-série
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – **Fax :** 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – boutique.ed-diamond.com
Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Secrétaire de rédaction : Véronique Sittler
Réalisation graphique : Kathrin Scali

Responsable publicité : Valérie Fréchard, Tél. : 03 67 10 00 27
v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 34 52 34 01
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 0183-0864
Commission paritaire : K78 976
Périodicité : Bimestrielle
Prix de vente : 8,00 €

CHOISIR SA LICENCE LIBRE

par Tristan Colombo

Une fois votre projet parvenu dans une version fonctionnelle, vous voudrez sans doute le distribuer et il faudra indiquer aux utilisateurs quels sont leurs droits par rapport au logiciel. C'est l'ajout d'une licence qui précisera ces droits, mais encore faut-il choisir correctement sa licence dans la jungle de celles existant...

« Qu'est-ce que c'est que le logiciel libre ? Je peux expliquer le logiciel libre en trois mots : liberté, égalité, fraternité. Liberté, parce qu'avec le logiciel libre, l'utilisateur est libre. Égalité, parce que par le logiciel libre, personne n'a de pouvoir sur personne. Et fraternité, parce que nous encourageons la coopération entre les utilisateurs. » C'est en ces termes, ou en tout cas des tournures approchantes, qu'après quelques recommandations débutent les conférences en français de Richard Stallman. Mais qui est ce monsieur et pourquoi en parler dans un article sur les licences libres ?

Richard Stallman est, entre autres, un défenseur acharné des libertés, le fondateur de la *Free Software Foundation*, l'initiateur du projet GNU et le créateur de l'une des premières véritables licences de logiciel libre, la *GNU General Public License*, publiée pour la première fois en 1989. On peut dire qu'il a défini les bases des licences libres qui sont apparues par la suite et qui, pour être considérées comme telles, doivent respecter quatre libertés fondamentales numérotées de 0 à 3 :

- **Liberté 0 - liberté d'utilisation** : vous pouvez utiliser le logiciel pour n'importe quelle tâche, sur n'importe quel système informatique, sans avoir à en référer au développeur ou à la société éditrice. Si vous redistribuez le programme à une tierce personne, vous ne pourrez pas lui imposer vos conditions d'utilisation : elle jouira de la même liberté que vous ;
- **Liberté 1 - liberté d'étude et de modification** : vous pouvez étudier le code source du logiciel et le modifier pour que le programme fasse ce que vous souhaitez ;
- **Liberté 2 - liberté de redistribution** : vous avez la possibilité de redistribuer des copies exactes du programme à une tierce personne ;
- **Liberté 3 - liberté de publication** : vous pouvez publier vos propres versions du logiciel modifié et contribuer ainsi à son évolution. Cette liberté de publication va plus loin puisque, en fonction de la licence du logiciel, vous pourrez le redistribuer sous une autre licence, y compris une licence non libre.

Il ne faut pas confondre logiciel libre et *freeware* (logiciel gratuit), où la liberté de copier est généralement accordée sans aucune restriction et où le code source n'est pas accessible. Il en va de même pour les *sharewares* où en plus, la liberté d'utilisation sera limitée soit au niveau des fonctionnalités, soit dans le temps.

Les logiciels libres, même s'ils sont le plus souvent distribués gratuitement, ne sont pas forcément des logiciels « non commerciaux ». On rentre ici dans un modèle économique particulier, qui est détaillé dans un article de ce hors-série.

Enfin, nous parlons de « logiciel libre », mais existe-t-il une réelle différence avec un logiciel « open source » ? Ces deux termes qualifient pratiquement les mêmes logiciels, mais dans le premier il y a l'acceptation sous-jacente d'une philosophie, de valeurs morales et sociales, alors que le second est axé sur un aspect purement marketing en vantant uniquement les aspects pratiques des logiciels, sans présence d'une idéologie qui pourrait faire peur aux décideurs des entreprises. Suivant les points de vue, le logiciel libre et l'open source sont ainsi plus ou moins éloignés sans être pour autant franchement opposés.

Pourquoi choisir une licence ? Le simple fait de distribuer un logiciel dans le domaine public ne le rend-t-il pas libre ? Bien sûr ! Mais rien ne protège le code ou son auteur : n'importe qui peut récupérer le code, le modifier et le redistribuer sous la forme d'un logiciel privateur.

Nous avons vu les différentes implications d'une licence libre, mais chaque licence a sa ou ses particularité(s) qu'il faut connaître avant de pouvoir faire un choix. Dans la suite, je vais vous présenter quelques licences libres parmi les plus utilisées et pour celles disposant de plusieurs versions, je ne m'attacherai qu'à la dernière version publiée. Pour chacune des licences, j'indiquerai l'organisme la gérant, l'URL du texte original, les spécificités de la licence et les règles en cas de redistribution de votre projet.

Petit glossaire libriste

Copyleft : le copyleft, ou gauche d'auteur, est une méthode permettant de rendre un programme libre et de forcer toutes ces versions suivantes à être également libres.

DRM : il s'agit de la gestion numérique des droits (DRM = *Digital Rights Management*), encore appelée MTP pour Mesures Techniques de Protection, applicables à une œuvre numérique. Ce mécanisme est là pour empêcher la copier privée et restreindre la lecture en fonction d'un élément particulier (marque de lecteur, zone géographique, etc.).

GNU : « GNU is Not Unix » est un système d'exploitation libre (compatible UNIX) initié par Richard Stallman en 1983. Linux est en fait GNU/Linux, car sans les développements du projet GNU, il n'y aurait pas de Linux (d'un autre côté, le noyau de GNU, nommé « Hurd », n'étant jamais sorti en version stable, sans le noyau Linux il n'y aurait pas non plus de GNU/Linux).

Licence libre : contrat dans lequel le détenteur des droits sur un produit expose les droits concédés aux utilisateurs. Ce contrat doit respecter les quatre libertés fondamentales.

Logiciel privateur : logiciel privant l'utilisateur des quatre libertés fondamentales (analogue à logiciel propriétaire).

Tivoïsation : du nom du terminal Tivo permettant d'enregistrer en numérique des programmes télé (utilisé aux États-Unis), principe matériel interdisant l'exécution d'un code sous GNU GPL après modification.

1 Liste des licences libres les plus utilisées

1.1 GNU General Public License (v3)

Troisième version de la Licence Publique Générale GNU, abrégée en **GNU GPL v3**, cette licence est gérée par la Free Software Foundation (texte complet sur http://www.gnu.org/licenses/gpl-3.0.txt).

Cette licence a une portée universelle dans la mesure où une attention toute particulière a été portée à ce qu'aucun concept ou terme juridique à spécificité nationale ne soit employé. Les principaux points forts de cette licence sont les suivants :

- L'article 7, intitulé « Additional Terms », définit sept permissions additionnelles qui pourront être éventuellement supprimées ou ajoutées, ce qui rend la licence plus modulaire ;
- Protection contre la « tivoïsation » : un produit embarquant du code sous licence GNU GPL v3 ne peut pas limiter l'accès à ce code ;
- La licence GNU GPL v3 n'interdit pas la mise en œuvre des DRM... mais elle empêche que ces verrous numériques ne soient imposés aux utilisateurs sans qu'ils ne puissent s'en défaire ;
- Le code ne peut pas inclure des composants distribués sous une autre licence (en tout cas une autre licence non compatible).

Du point de vue de la redistribution du code après modification, il n'est pas possible de changer de licence. Il est interdit de retirer les mentions de propriété intellectuelle et les contributeurs doivent exposer clairement leurs noms et la date, ainsi qu'un texte explicatif des modifications effectuées. Il est possible de commercialiser le nouveau produit à condition de conserver la licence et de fournir le code source.

1.2 GNU Lesser General Public License (v3)

La Licence Publique Générale Limitée GNU, abrégée en **GNU LGPL v3**, est une licence GNU GPL v3 amoindrie en termes de restrictions (texte complet sur http://www.gnu.org/licenses/lgpl-3.0.txt). Par exemple, il est possible d'incorporer des composants distribués sous une autre licence, même si le logiciel complet doit rester sous GNU LGPL v3.

1.3 GNU Affero General Public License (v3)

La Licence Publique Générale Affero GNU, abrégée en **GNU AGPL**, est une version particulière de la licence GNU GPL v3 destinée aux logiciels en ligne (texte complet sur http://www.gnu.org/licenses/agpl-3.0.txt). Elle oblige la mise à disposition du code source, même si le logiciel n'est utilisable que sous la forme d'un service à distance.

1.4 Licence Berkeley Software Distribution (v3)

La licence Berkeley Software Distribution, ou **BSD**, fait partie des premières licences libres (texte complet sur http://opensource.org/licenses/BSD-3-Clause). Cette licence est issue de l'Université de Berkeley qui ne la gère plus. Elle est beaucoup moins restrictive que les licences GNU et son texte en est considérablement allégé. Seules quelques obligations sont mentionnées, permettant de déduire que tout le reste est autorisé.

En cas de redistribution, il faut simplement conserver les mentions de propriété intellectuelle dans l'en-tête des fichiers source. Vous pouvez ensuite inclure des composants utilisant une autre licence, changer de licence, etc. Il s'agit d'une licence sans *copyleft* (appelé également « gauche d'auteur » en

français) : les versions modifiées du programme ne sont pas forcément des logiciels libres. D'un point de vue commercial, vous pouvez faire absolument ce que vous voulez.

Le problème de cette licence dans sa version originale (en dehors de son absence de copyleft) était son attachement à l'Université de Berkeley dans une clause de publicité :

All advertising materials mentioning features or use of this software must display the following acknowledgement : This product includes software developed by the University of California, Berkeley and its contributors.

Les utilisateurs ont bien sûr remplacé « Université de Berkeley » par des références à leur propre organisation, ce qui conduit à une myriade de licences qui doivent toutes être citées différemment.

1.5 Licence Massachussetts Institute of Technology

La licence Massachussets Institute of Technology, ou **MIT**, dont le texte complet est disponible sur http://opensource.org/licenses/mit-license.php, est une licence très permissive, sans copyleft, au même titre que la licence BSD. Elle est globalement identique à la licence BSD tout en décrivant les droits cédés de manière plus précise.

1.6 Licence Apache (v2)

La licence **Apache** est encore une licence permissive, sans copyleft, proche des licences BSD et MIT (texte complet sur http://www.apache.org/licenses/LICENSE-2.0.txt). Par rapport à ces deux dernières, les termes de la licence sont toutefois plus détaillés. On trouve notamment :

- L'obligation de joindre une copie de la licence Apache en cas de redistribution (changement de licence possible, mais la licence Apache sera toujours présente) ;
- Toutes les modifications doivent être tracées, comme dans une licence GNU GPL v3 ;
- Il est expressément indiqué dans la licence que l'on peut proposer une garantie aux utilisateurs du logiciel (cette garantie n'engageant que celui qui la propose).

Pour le reste, comme avec la licence BSD, des composants utilisant d'autres licences peuvent être incorporés et il n'y a pas de restriction pour un usage commercial.

1.7 Mozilla Public License (v2)

La Mozilla Public License, ou **MPL**, gérée par... la Mozilla Foundation, est une licence permissive en ce qui concerne l'intégration de produits, mais non permissive dans la redistribution (texte complet sur http://www.mozilla.org/MPL/2.0/index.txt). Le logiciel modifié est donc obligatoirement redistribué sous licence MPL, mais il peut contenir des éléments utilisant diverses licences (qui s'appliqueront toujours à ces éléments). Il faut obligatoirement joindre le texte de la licence et lister les modifications, ainsi que leurs auteurs.

C'est une bonne licence intermédiaire si vous devez utiliser des composants non compatibles avec une licence GNU : pas trop de copyleft, mais un peu quand même...

2 Des licences libres dans des domaines extérieurs à l'informatique

2.1 GNU Free Documentation License (v1.1)

La documentation des logiciels doit également être protégée par une licence. La GNU Free Documentation Licence, ou **GNU FDL**, garantit le droit d'auteur tout en permettant à l'utilisateur de modifier et de redistribuer les documents (texte complet sur http://www.gnu.org/licenses/fdl-1.3.txt). Cette licence est bien sûr gérée par la Free Software Foundation comme toutes les licences GNU. Elle s'applique à la documentation d'un logiciel, mais de manière plus générale, elle peut s'appliquer à tout document écrit.

En cas de redistribution, le document peut être vendu.

2.2 Licence Art Libre

Licence spécifiquement destinée aux œuvres artistiques (texte complet disponible sur http://artlibre.org/licence/lal). Il s'agit d'une licence avec copyleft, donc pas de changement de licence en cas de redistribution.

2.3 Licence(s) Creative Commons

La licence Creative Commons est un peu particulière et on devrait plutôt parler de licences Creative Commons (les textes sont accessibles depuis http://creativecommons.org/licenses/). Il s'agit d'une licence entièrement paramétrable par quatre conditions qui peuvent être ajoutées et associées :

- **Attribution** : il s'agit d'une indication de paternité. En cas de redistribution, il faudra vous citer ;
- **Pas d'utilisation commerciale** : autorisation d'utilisation, modification et redistribution de l'œuvre (documentation, photo, mémoire, etc.) dans un cadre non commercial, à moins que vous n'en donniez votre consentement ;
- **Partage dans les mêmes conditions** : en cas de redistribution, la licence devra être inchangée, à moins que vous n'en donniez l'autorisation ;
- **Pas de modification** : si quelqu'un souhaite modifier votre œuvre, il devra obtenir votre autorisation.

De ces quatre conditions, on peut obtenir six licences différentes, représentées par six logos différents (voir figure 1) :

- Attribution : Creative Commons By, abrégé en **CC BY**.
- Attribution et pas d'utilisation commerciale : **CC BY-NC**.

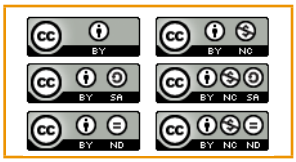


Fig. 1 : Logos des six licences Creative Commons. De gauche à droite et de haut en bas : CC BY, CC BY-NC, CC BY-SA, CC BY-NC-SA, CC BY-ND, et CC BY-NC-ND.

- Attribution et partage dans les mêmes conditions : **CC BY-SA**.
- Attribution, pas d'utilisation commerciale et partage dans les mêmes conditions : **CC BY-NC-SA**.
- Attribution, pas de modification : **CC BY-ND**.
- Attribution, pas de modification et partage dans les mêmes conditions : **CC BY-ND-SA**.

Conclusion

Pour vous aider dans votre choix, le site http://www.tldrlegal.com vous propose pour chaque licence une description très succincte, le texte complet, mais surtout un résumé de ce que vous pouvez faire, ce que vous ne pouvez pas faire et ce que vous êtes dans l'obligation de faire (voir figure 2). Ceci est présenté de manière très claire et même si cela ne vous empêchera pas de devoir lire le texte complet si vous voulez savoir exactement ce que dit la licence, vous aurez une vision synthétique qui vous permettra d'éliminer rapidement certaines licences ne correspondant pas à vos attentes.

Fig. 2 : Résumé de la licence GNU GPL v3 sur le site tldrlegal.com

Choisir une licence, c'est garantir que votre logiciel évoluera dans la direction que vous avez souhaitée. Et n'oubliez pas que vous pouvez tout à fait choisir plusieurs licences pour différentes parties de votre projet (logiciel, extensions, documentation, etc.). ■

FINANCER UN PROJET OPEN SOURCE : LE FINANCEMENT PARTICIPATIF

par Nathael Pajani

Pour trouver des fonds pour un projet open source, une des alternatives est de créer une campagne de financement participatif, du terme anglais « crowdfunding » (littéralement, financement par la foule).

1 Préambule

Dans le but d'englober un maximum de cas, je tiens à faire quelques précisions.

J'utilise le mot « projet » et pas « logiciel », car le financement participatif n'a pas vraiment de limites à ce que l'on peut financer. D'ailleurs, bien peu de projets de développement de logiciels utilisent ce mode de financement, peut-être parce que la majorité des personnes à même de contribuer préfèrent participer directement au développement et donner du temps et de la matière grise que donner des sous, ou parce que le système de dons fonctionne aussi assez bien (cf. Wikipédia). Les dons sont d'ailleurs une forme de financement participatif ! Mais si participer est très simple pour le développement d'un logiciel, ce n'est pas toujours possible, comme dans le cas de la production d'un disque ou de matériel.

Je voulais aussi conserver le terme « open source » qui m'a été donné comme sujet lorsque l'on m'a proposé de rédiger cet article, malgré les remarques faites sur ce sujet, toujours pour englober tous les projets. Sauf que certains projets peuvent ne pas avoir de « sources » au sens du code source. Je pense en particulier à l'art. Mais dans le cas de l'art, comme dans d'autres cas, le terme « libre » proposé ne convient pas forcément. Le terme « libre » tel qu'utilisé dans « logiciels libres » implique normalement quatre libertés. Mais tout travail ne peut pas être publié sous licence libre au sens des quatre libertés fondamentales du logiciel libre.

Pour les sceptiques, Richard Stallman lui-même publie certains de ses textes sous licence CC-BY-ND. Une licence qui interdit les travaux dérivés, et est donc incompatible avec le terme « libre » au sens des quatre libertés. J'utiliserai donc, à partir de maintenant, le terme « ouvert ». Du moins, j'essayerai.

2 Le financement participatif, c'est quoi ?

Le financement participatif est une rupture par rapport aux modes de financement « classique ».

Dans le mode de financement classique, une banque, un investisseur privé, ou un fond d'investissement (ou quelques-uns de ceux-ci) prête(nt) des sous à une personne (ou plus souvent à une entreprise) en échange de la perspective (ou de la garantie dans le cas des banques) d'un très fort « retour sur investissement ». De plus, le choix des projets financés ou soutenus appartient dans ce cas à un nombre très limité de personnes, ayant parfois oublié de changer de siècle.

Dans le cas du financement participatif, le rôle des investisseurs, qui était réservé aux personnes riches ou très riches, est pris par beaucoup de personnes beaucoup moins riches, comme vous et moi. Ce n'est plus une personne qui décide de l'intérêt et de la viabilité d'un projet, mais tout le monde, tous les utilisateurs finaux.

Pour ce qui est du retour sur investissement, il va de nul à non quantifiable. Du don désintéressé, jusqu'au financement d'un projet comme Wikipédia, pour lequel chaque centime aide au maintien ou à la croissance de projets dont la valeur n'est pas monétaire, et potentiellement sans commune mesure avec l'investissement. Dans la majeure partie des cas, il reste cependant équivalent à l'investissement, mais sous une autre forme : bien matériel, accès privilégié à une ressource ou à un événement, contrepartie qui dans tous les cas convient à l'investisseur puisqu'il choisit de contribuer.

Et dans le cas des plateformes de financement dont nous parlerons plus loin, le (ou les) porteur(s) du projet garde(nt) un contrôle total sur le projet. Les personnes qui choisissent de soutenir le projet peuvent leur faire des remarques, des suggestions, mais n'ont aucun pouvoir sur les décisions prises par le(s) porteur(s) de projet.

Le financement participatif est donc finalement quelque chose de très

vieux, mais il a pris un nouvel élan avec l'apparition il y a quelques années de plateformes dédiées. Ces plateformes ont pu voir le jour grâce à Internet et aux réseaux sociaux, qui permettent de diffuser l'information à un très grand nombre de personnes en un temps limité.

Quand on pense financement participatif, on pense souvent Ulule, IndieGoGo, Kickstarter et équivalents. Cependant, il faut savoir que leur mode de fonctionnement n'est qu'une des facettes du financement participatif. Nous l'avons déjà évoqué avec le système de dons (ou mécénat global) utilisé par Wikipédia et de nombreux projets libres, comme Framasoft, et l'illustration ci-contre (Fig. 1), qui se concentre sur les systèmes utilisant une plateforme dédiée, le montre bien.

Pour compléter ce schéma, il faudrait ajouter les divers outils utilisés pour les dons, comme Flattr, qui permet de rémunérer les auteurs (de toute sorte) sur Internet, PayPal, qui facilite les transferts d'argent entre particuliers et donc les dons, et les banques, toujours pour les dons. En effet, les dons sont un mode de rémunération assez répandu pour les développeurs de logiciels, et je pense que ça fonctionne au moins un petit peu.

Remarque

Petit aparté puisqu'on parle de logiciel : il y a une plateforme dédiée au financement des développements de logiciels libres, *Open Funding* (voir lien en fin d'article). Elle permet aux développeurs de proposer une estimation chiffrée du développement de chaque fonctionnalité, et les utilisateurs apportent leur contribution sur les fonctionnalités qui les intéressent. Une fois le montant atteint, le développeur développe la fonctionnalité, et si les utilisateurs valident le développement (ou au bout de deux semaines au plus tard si personne ne répond) le développeur est payé. Il ne s'agit cependant pas du mode de fonctionnement des autres plateformes, c'est donc en dehors du sujet de cet article.

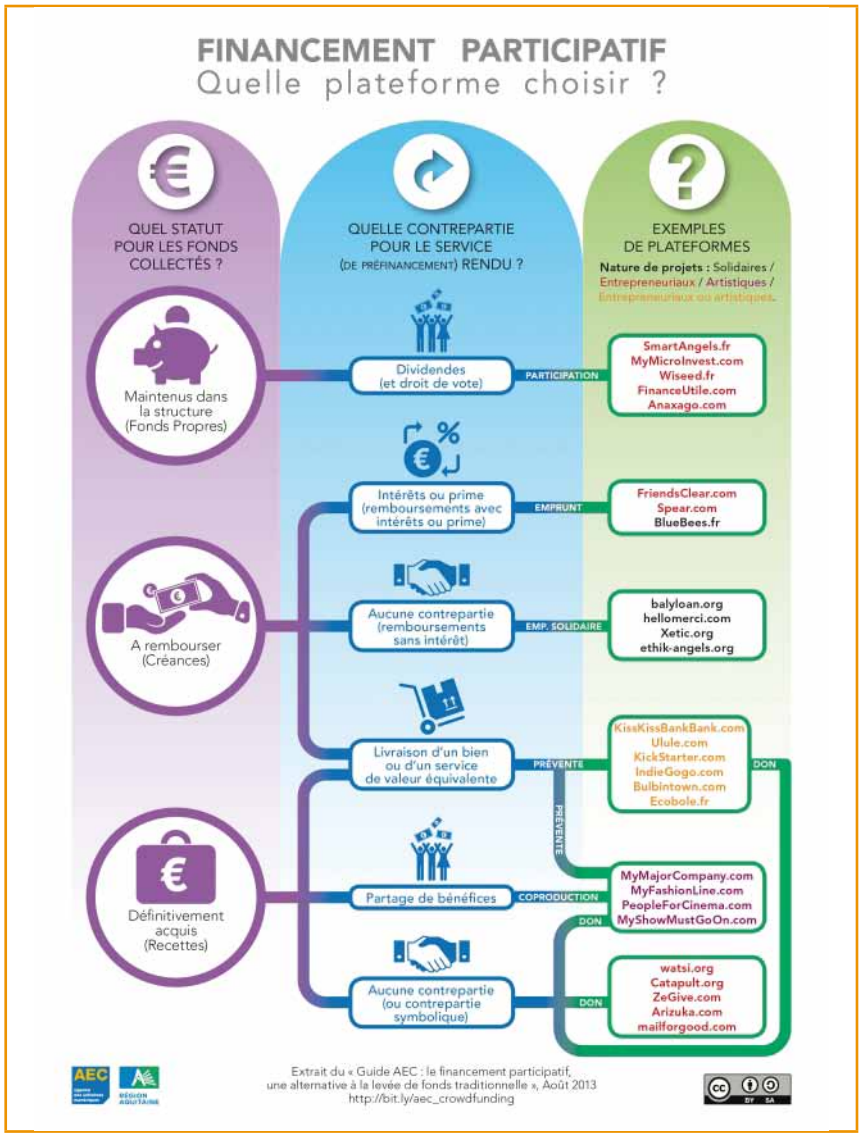


Fig. 1 : Les modes et plateformes de financement participatif

Nous allons désormais nous concentrer sur l'art et la manière de mener une campagne de financement participatif à terme (avec succès si possible) et les écueils à éviter. Nous parlerons uniquement des campagnes de type « prévente », avec livraison d'un bien ou d'un service à la clé, même si les personnes qui supportent un tel projet auront dans la majorité des cas la possibilité de faire des dons sans contreparties.

3 Pourquoi faire appel au financement participatif ?

Avant de nous lancer dans les différents aspects du montage de la campagne, il nous faut répondre à cette question de la plus haute importance. À quoi sert le financement participatif, ou pourquoi faire une campagne ? Il y a trois réponses à cette question :

- Faire connaître votre projet,
- Récolter des fonds,
- Préparer une autre campagne.

Pour les deux premières, nous reviendrons dessus juste après, car elle semblent évidentes. Pour la troisième réponse, elle peut sembler farfelue au premier abord, mais gérer une campagne de financement n'est pas de tout repos, comme vous allez le voir dans cet article, et si votre objectif réel est ambitieux, alors une première campagne de préparation peut être une bonne idée, pour vous familiariser avec le concept, l'outil, et commencer à créer une petite communauté.

Attention cependant, très peu de monde donnera deux fois pour la même chose, et vous avez donc deux solutions : soit expliquer clairement qu'il s'agit d'un « essai » (mais ce n'est pas très vendeur), soit trouver un sujet suffisamment différent (tout en restant cohérent), pour que les internautes comprennent ce que vous faites, sans faire deux fois la même campagne. Une solution possible est de commencer par lever des fonds pour la fabrication d'un prototype, pour préparer la campagne de prévente de la version finale, ou de lever des fonds pour l'hébergement du site de la communauté que vous voulez créer autour de votre logiciel, avant une campagne plus vaste pour financer le développement.

Et justement, une campagne de financement participatif ce n'est pas que récolter des sous. Les personnes qui soutiendront votre projet seront les premiers membres d'une potentiellement vaste communauté d'utilisateurs ! C'est grâce à eux que votre projet va se répandre, vivre et que de nouveaux participants (ou de nouveaux clients) arriveront.

Vos premiers contributeurs seront autant d'ambassadeurs pour votre projet, fiers de montrer qu'ils vous ont soutenu dès le départ. N'hésitez pas à leur donner un statut privilégié, cela sera apprécié ! Ne négligez pas l'effet d'initiation d'une communauté de contributeurs que peut avoir une campagne de financement participatif.

4 Les différentes plateformes de financement participatif : petite liste non exhaustive

Pour créer une campagne de financement participatif qui aura une chance d'arriver à bon terme, il est important de respecter un certain nombre de règles pour que les internautes ne soient pas perdus. En effet, si vous devez les convaincre de l'intérêt de votre projet, ils devront aussi être convaincus de votre honnêteté.

La méthode la plus simple est d'utiliser une des plateformes existantes, opérée par un organisme extérieur, dont nous allons faire un petit tour rapide. Ils ne sont pas les garants de votre honnêteté ou de votre crédibilité, mais réaliseront un premier filtrage qui rassurera les internautes, qui auront aussi l'assurance de revoir leurs sous si finalement le projet n'est pas financé.

- **Ulule** : commençons par le premier français et européen. Même s'il n'est pas forcément le plus connu, il est l'un des plus accessibles pour nous, dispose d'une communauté respectable, et les échanges avec l'équipe d'administration se font en français, ce qui peut être appréciable pour la compréhension. Côté configuration de la campagne, il y a quelques limitations sur ce qu'il est possible de faire : succès uniquement si l'objectif est atteint, changement de l'objectif uniquement par les administrateurs d'Ulule, et quelques limitations sur la mise en page, qui devra respecter les possibilités offertes par l'interface d'édition. Cela a l'avantage de conserver un site cohérent, et de mettre tout le monde sur un pied d'égalité, et vous empêchera de masquer les éléments essentiels d'une plateforme de financement de ce type : l'accès aux autres projets. C'est cette plateforme que j'ai utilisé pour le projet DomoTab.

Bon à savoir

Pour se financer, les différentes plateformes prélèvent une partie (non négligeable) des fonds récoltés, de 7 à 12 %. Sur Ulule, cependant, les versements par chèque bénéficient d'un tarif avantageux de 5 % seulement de prélèvement, contre 8 % pour les versements par PayPal ou par carte bancaire !

- **IndieGoGo** : un californien sympa, car ouvert à tous (comme Ulule), ayant plus de visibilité à l'international, et dont le prélèvement est relativement bas sur les projets à objectif fixe (*fixed funding*) : 7 %. Il offre aussi une possibilité intéressante, qui se rapproche du principe des dons : le « flexible funding », un type de campagne qui réussira dans tous les cas, quel que soit le niveau de soutien. L'objectif annoncé n'est

alors qu'indicatif... ou presque, car s'il n'est pas atteint, le prélèvement sera de 12 % ! Cela peut être un bon moyen pour indiquer que le projet se fera de toute façon, quel que soit le montant atteint.

- **Kickstarter** : le mastodonte, au sens de la taille de la communauté bien entendu. Je ne sais pas s'il existe encore un américain qui ne connaît pas, et beaucoup de monde en a déjà au moins entendu parler. Un gros avantage si vous voulez toucher un public très vaste et international, et ne pas devoir expliquer systématiquement ce qu'est une plateforme de financement participatif. Le hic... il faut être Américain, Anglais ou Canadien, ou au moins qu'une personne du projet remplisse les conditions établies par Kickstarter. Pour le reste, Kickstarter est très similaire à Ulule (ou bien est-ce l'inverse ?).

- **KissKissBankBank** : le deuxième français (par le nombre de projets financés, la société ayant été créée avant celle qui gère Ulule). Bon, rien de nouveau, on prend les mêmes (principes) et on recommence avec d'autres (personnes)...

Et je vais arrêter ma liste ici, au risque de froisser certaines plateformes de financement. Je n'ai passé en revue que celles que je juge les plus pertinentes pour des projets « technologiques », mais il y en a bien d'autres, pour tous types de projets.

Le souci c'est que la multitude de solutions finit par égarer l'utilisateur et dessert au final les porteurs de projets, qui doivent choisir une plateforme et risquent ainsi de se couper d'une partie des soutiens qu'ils pourraient avoir.

5 Une alternative libre !

Il existe une alternative aux plateformes de financement, libre qui plus est : l'outil **Selfstarter** (voir lien en fin d'article pour le code sur GitHub). Il s'agit d'un site en Ruby, avec apparemment tous les éléments pour créer votre propre campagne si vous voulez utiliser « Amazon Payments » pour gérer les paiements en ligne. Encore une fois, il faut être américain pour utiliser Amazon Payments, cependant il doit être possible de modifier Selfstarter pour utiliser PayPal, ou tout autre moyen de paiement qui vous intéresse (c'est un logiciel libre après tout, c'est aussi fait pour ça). Je ne l'ai pas utilisé, ni mis en place, mais plusieurs projets l'ont déjà utilisé avec succès.

Un projet libre pour récolter les fonds d'un projet libre, quoi de mieux me direz vous ? Difficile d'argumenter contre ceci, mais il faut tout de même bien prendre en compte qu'une plateforme de financement participatif vous donne une visibilité que vous n'aurez pas avec Selfstarter. Il vous faudra faire toute votre communication tout seul(s). Et pour qu'un financement participatif fonctionne, il faut toucher beaucoup, beaucoup de monde. Vos soutiens potentiels ne seront pas rassurés par le « cadre » de la plateforme de financement participatif. N'oubliez pas non plus qu'Amazon Payments gardera entre 2 et 5.5 % de la somme collectée, fonction du montant des souscriptions et du volume global.

Personnellement, je pense qu'un Selfstarter peut être utile en complément d'une autre plateforme de financement, pour vous permettre de récolter des financements « en direct », sans passer par la plateforme, et donc vous affranchir du pourcentage prélevé par la plateforme de financement.

6 Et en pratique, comment fait-on ?

Mener à bien une campagne de financement participatif est une tâche complexe. Attention, faire une campagne de financement prend du temps. Beaucoup de temps. Surtout si vous n'êtes pas connu et que votre projet n'est pas connu. En gros, pour une campagne de deux mois, comptez entre un et trois mois de boulot. À plein temps (c'est-à-dire avec des journées de 12 à 15 heures les bons jours).

Si vous voulez lever un petit millier d'euros pour un projet perso et que vous êtes au chômage, il existe des boîtes d'interim, c'est plus efficace, puisque le côté « création d'une communauté » n'a alors que très peu d'intérêt.

Pour le reste, une campagne se découpe en trois phases chronophages :

- La préparation,
- La campagne en elle-même, son suivi, son animation,
- L'après campagne : les contreparties.

6.1 La préparation de la campagne

Comme vous l'avez maintenant compris, la première étape est le choix de la plateforme de financement. C'est un processus compliqué. Vous pourrez difficilement en tester plusieurs et donc, seulement vous appuyer sur les retours d'expérience que vous feront ceux qui ont déjà tenté l'aventure, et ce que vous pourrez récolter comme informations.

Cette première étape prend du temps, mais dites-vous que si elle torture certains pendant longtemps avant qu'ils ne se lancent, ce n'est qu'une courte étape. Il faut ensuite préparer le projet à soumettre à l'équipe de validation (pour les sites qui le demandent). Ce « dossier » se compose de plusieurs éléments, qu'il vous faudra de toute façon préparer s'ils ne vous sont pas directement demandés.

6.1.1 Choisir l'objectif de la campagne

Il existe différents types de campagnes : avec ou sans échec si l'objectif n'est pas atteint, avec un objectif en termes de montant collecté, ou en nombre de soutiens avec un montant fixe, et choisir le montant. Si vous choisissez un objectif en nombre de soutiens, l'interface Ulule n'affiche pas « ou plus » pour les souscriptions. Cela n'encourage donc pas les utilisateurs à donner plus que ce qui est demandé pour chaque contrepartie.

Fixer le montant n'est pas une partie de plaisir et peut vous torturer les méninges pendant longtemps. Dans le cas où vous avez un budget précis, la question ne se pose pas, il faut demander ce dont vous avez besoin et l'expliquer, c'est toujours apprécié.

Pour les autres, un objectif trop haut, et le risque de ne pas l'atteindre grandit et peut décourager certains soutiens. Trop bas, et l'atteindre ne vous apportera pas grand-chose, surtout si les soutiens potentiels se disent « C'est bon, ils ont levé ce qu'ils voulaient, pas besoin de donner plus ». Il est alors utile de prévoir des paliers dès le départ. Les paliers sont des objectifs intermédiaires, qui ne sont pas intégrés au système fourni par la plateforme de financement, mais très utilisés par les projets qui réussissent vraiment bien.

Ce système consiste à définir un premier palier réaliste à atteindre pour que la campagne soit considérée comme financée par la plateforme, et qui permettra au projet de démarrer. Ensuite, dans la description de la campagne, vous ajouterez la description des paliers suivants (certains ne dévoilent qu'un ou deux palier(s) d'avance), sous la forme « si tel montant est atteint, alors nous ajouterons ceci aux contreparties » (voir plus loin pour les contreparties), de façon à motiver les personnes qui voudraient soutenir le projet à continuer, même si l'objectif initial a été atteint. Attention, il faut cependant que les paliers soient motivants, et que l'objectif initial soit suffisamment intéressant pour que les internautes aient envie de vous soutenir, même si seul cet objectif est atteint, car il validera la campagne.

6.1.2 Choisir la durée de la campagne

Le choix de la durée de la campagne est aussi un élément important. Les plateformes de financement limitent la durée des campagnes à trois mois maximum. N'utilisez pas Selfstarter pour créer une campagne sur un an, mettez en place un système de dons dans ce cas-là, ou une boutique de vente en ligne si vous voulez vendre des objets.

Une campagne de financement participatif est une sorte de sprint, et il faut qu'elle arrive au bout sans s'essouffler. Mais si elle est trop courte, les personnes qui pourraient soutenir n'auront pas forcément vu passer l'information !

Les questions à se poser sont les suivantes : « Combien de personnes connaissent déjà votre projet ? », « Quel est leur intérêt pour votre projet ? » et « Combien de personnes connaissent-elles ? ». Les réponses à ces questions vous donneront une idée de la vitesse à laquelle l'information pourra se répandre et donc, de la durée minimum de votre campagne. Si vous estimez qu'il vous faut plus de trois mois, alors ne lancez pas votre campagne tout de suite, finissez de lire cet article et suivez les conseils que je donne un peu plus loin :)

Je pense que faire une campagne de deux mois est un bon compromis. Cela vous laisse du temps pour contacter beaucoup de monde, pour publier des articles, des news et convaincre du monde, et montre que vous êtes confiant dans le succès de votre campagne, car vous n'avez pas utilisé la totalité du temps disponible. Attention cependant, c'est ce que nous avons fait pour la campagne DomoTab sur Ulule et l'objectif n'était pas atteint au bout des deux mois prévus, il nous a fallu rallonger la campagne de 15 jours (merci à l'équipe d'Ulule d'ailleurs !).

6.1.3 Présenter le projet, le produit et l'équipe

Tous les sites de financement participatif vous demanderont un certain nombre d'informations sur votre projet, et vous aurez tout intérêt à passer beaucoup de temps sur cette présentation, car c'est ce que vos soutiens potentiels verront, ce qui les décidera à vous soutenir ... ou pas. Sachez que la majorité des personnes prendront leur décision en moins de dix secondes. Les visiteurs devront donc comprendre votre projet au premier regard sur votre page.

La présentation devra inclure plusieurs éléments, à savoir : un titre, clair et accrocheur, un sous-titre ou une très courte phrase d'introduction, une image principale et/ou une vidéo de présentation, une présentation courte du projet (sous forme de texte cette fois), une présentation courte de l'équipe, l'état d'avancement du projet, et ce à quoi vous servira l'argent récolté.

La courte phrase d'introduction est là pour expliquer le titre, le projet et donner une phrase à retenir et reprendre sur Twitter et autres blogs. Si cette phrase ne tient pas sur Twitter, elle est trop longue.

La présentation vidéo : un *must*. Nous n'avions pas fait de vidéo par faute de temps, de moyens et d'éléments à présenter. Le temps n'est pas une excuse, les moyens ça se trouve, et si vous n'avez pas d'éléments à présenter, ce n'est peut-être pas le bon moment pour lancer une campagne de financement... Si vous ne savez pas à quoi peut

ressembler une vidéo promotionnelle pour une campagne de financement, regardez-en quelques-unes, en choisissant si possible des campagnes qui ont ((très) bien) réussi (bien que regarder ce qu'il ne faut pas faire permette aussi d'apprendre). N'oubliez pas que si vous souhaitez obtenir le soutien de personnes outre-Atlantique, voire simplement de personnes non francophones, il vous faudra une vidéo en anglais. Et encore plus que sur une vidéo classique, l'image « statique » de présentation de la vidéo devra donner envie de la regarder.

Les visuels : lisibles, clairs et surtout accrocheurs. Le visuel que vous devrez le plus soigner est l'image principale de présentation du projet. C'est ce visuel qui sera diffusé sur le reste de la Toile pour présenter votre projet. Il devra donner envie. Tout le monde doit vouloir cliquer dessus. S'ils le font, c'est 50 % du travail de fait.

Les textes : simples, courts (oups, tout le contraire de ce que j'ai fait) et non exhaustifs (si si !). Les textes que vous écrirez ne devront inclure QUE les éléments importants. Ne décrivez surtout pas toutes les fonctionnalités et toutes les options. Soyez concis, vous avez dix secondes. Mettez des liens. Les personnes qui veulent la liste exhaustive des fonctionnalités suivront les liens et passeront une heure sur votre site (voir plus loin) pour trouver les informations qui les intéressent, et pour vérifier votre sérieux ou les chances de réussite du projet. Les détails techniques ne doivent pas se trouver dans les textes de présentation de la campagne, il s'agit d'un texte commercial !

6.1.4 Bien choisir les contreparties

En échange de leur contribution, vos soutiens attendront une juste contrepartie. Mais chacun doit pouvoir choisir une contrepartie qui corresponde à ses moyens, et à ses besoins. Attention cependant, mettre trop de contreparties différentes rend le projet illisible, brouillon. Il vous faudra définir des contreparties claires, adaptées à différents budgets, et à la fois réalistes pour vous et pour les contributeurs.

Si les personnes susceptibles de soutenir votre projet estiment que vous cherchez à les escroquer, votre campagne n'aboutira pas. Inversement, ne proposez pas de cadeaux que vous aurez du mal à fournir, ou qui vous demanderont un travail que le succès de la campagne ne permettra pas de financer. En cas de besoin, servez-vous du système de paliers pour ajouter des éléments aux récompenses initiales, comme un objet ou une fonctionnalité supplémentaire pour tous si telle somme est atteinte.

Souvent, les contreparties d'un montant supérieur incluent le contenu d'autres contreparties, ou de toutes les précédentes. Ceci n'est cependant pas une obligation. Mais surtout, faites bien attention à ce que les contreparties soient équilibrées (l'achat groupé de trois objets doit coûter moins cher que trois fois un objet !) et ne changez pas vos contreparties pendant la campagne si des personnes les ont déjà choisies. Ils risquent de se sentir lésés et de retirer leur soutien !

Et n'oubliez pas trois choses :

- Le site qui héberge votre campagne gardera 7 à 12 % des versements !
- Si vous vendez un produit en tant qu'entreprise, il y aura de la TVA (pour les ventes en Europe du moins) ;
- N'oubliez pas les frais de port ! (Certains sites vous permettent de les indiquer, sinon faites-le).

6.1.5 Se faire connaître AVANT !

Ce point est peut-être le plus important. Même si l'objectif de la campagne est de se faire connaître, il est important d'avoir un maximum de contacts avant le début de la campagne. Vous avez besoin de (très, énormément) beaucoup de *followers* et surtout de followers actifs. Sauf si vous voulez vendre des tartelettes au citron dans un emballage fait maison en papier mâché, vos contacts Facebook ne vous apporteront pas grand-chose. Vous avez besoin de vrais accros du tweet dans vos contacts, de personnes qui parleront (en bien) de votre projet, et qui vous aideront à le vendre.

Ulule présente assez bien le concept des cercles de financement dans son article intitulé « Les trois Cercles du financement participatif » et d'ailleurs, votre projet ne sera visible publiquement sur leur plateforme que si au moins cinq personnes ont soutenu votre projet. C'est-à-dire que vous devrez faire le travail de diffuser l'information et de convaincre au moins cinq personnes que vous connaissez avant que le reste du monde puisse découvrir votre projet.

Et quand bien même votre petit(e) ami(e) ou votre animal de compagnie trouve votre projet génial, cela ne suffira pas. Il faut que tout le monde croie en vous et au succès de votre projet, et y voie un intérêt. Si les personnes autour de vous (qui pourraient être concernées) n'y croient pas, il est encore temps de laisser tomber.

Une campagne qui échoue n'est pas forcément une catastrophe, mais que penseront les internautes de votre projet ? Qu'il n'était pas (suffisamment) bien ? Qu'il est mort et enterré ?

6.1.6 Le site de présentation exhaustive : une obligation

La dernière étape de préparation concerne un élément extérieur aux plateformes de financement participatif, mais un élément primordial : un site annexe de présentation du projet. Puisque vous n'avez pas mis tous les détails sur le site de la campagne (rassurez-moi, vous suivez mes conseils ?), vous devez donner un moyen aux curieux et aux sceptiques de trouver toutes les informations qui les décideront à soutenir le projet.

Il vous faut donc préparer un site qui présente le projet ou le produit, et QUE ça. Ce site sera la référence pour les curieux. Ce site pourra (devra ?)

inclure des exemples d'utilisation, qui aideront le visiteur à s'approprier l'objet. Il n'est pas nécessaire que ce site soit exhaustif dès le départ, mais il sera tout de même plus complet, et vous pourrez le faire évoluer et l'enrichir pendant la campagne. Dans ce cas-là, les curieux devront pouvoir poser très facilement des questions, et vous devrez y répondre très vite.

Attention, il ne peut pas s'agir d'une page de votre blog, ou d'une page d'un site sans rapport avec votre campagne, et il faut éviter la sous-page d'un site existant. Vous devez montrer votre projet, et que lui. Vous pourrez cependant inclure un blog, qui vous permettra de rendre votre campagne vivante, de donner des liens vers d'autres articles parlant de la campagne, ou de laisser une trace de l'évolution de la campagne de financement.

Surtout, une fois leur curiosité satisfaite, les internautes devront disposer d'un lien immédiat vers la campagne de financement !

6.2 La campagne en elle-même, son suivi et son animation

Pour décider les personnes qui ne vous auront pas soutenu dès les premiers instants de votre campagne de financement, il vous faudra rendre votre campagne vivante. Les plateformes de financement participatif intègrent en général trois outils pour vous aider dans cette tache difficile : une FAQ, un fil de news et un fil de commentaires.

6.2.1 Les news

Le fil de news, c'est vous qui devrez l'alimenter. Vous pouvez prévoir dès le début de votre campagne des éléments à présenter en cours de route, pour faire parler du projet et relancer le « buzz ».

6.2.2 La FAQ

Pour la FAQ, vous pouvez commencer à la remplir en répondant à quelques questions évidentes pendant la préparation. Cela pourra vous servir pour alléger le texte de présentation. Vous devrez aussi y faire apparaître les questions auxquelles vous aurez répondu sur le site de présentation, ou dans les commentaires d'autres sites qui parleront de votre projet. Faites intervenir vos connaissances, en leur demandant de poser des questions, même s'ils connaissent déjà la réponse. S'ils vous font une remarque sur un point qui manque à la présentation, demandez-leur de poser la question dans la FAQ plutôt que de l'intégrer immédiatement dans la description.

6.2.3 Les commentaires

C'est pour les commentaires que vos soutiens les plus proches peuvent (devraient ? doivent ?) intervenir. Plus il y aura de commentaires, plus les visiteurs verront d'intérêt à votre projet. Demandez-leur de poster une phrase, un mot, ou un petit texte expliquant ce qu'ils feront quand ils auront reçu leur contrepartie (car ils vous ont soutenu, n'est-ce pas ?).

6.2.4 Publiez

Pour faire vivre votre campagne, vous devrez en parler. Et pas à votre chat. Twettez, faites tweeter votre chat, utilisez tous vos réseaux sociaux, publiez des articles, proposez du contenu à d'autres sites, essayez d'obtenir des interviews. Faites parler de vous ! Obligez vos amis à parler de vous, menacez-les si besoin (bon, pas trop, ne vous fâchez pas avec eux non plus). Vous n'avez que le temps de la campagne pour collecter des fonds. Quand elle sera finie, ce sera trop tard. Postez des articles sur votre blog aussi, cela pourra donner des idées d'articles à d'autres blogueurs.

6.2.5 Faites vivre votre page

La page de la campagne sur le site de financement participatif n'est pas forcément figée. Vous pouvez ajouter des informations, encourager les visiteurs, remercier vos soutiens, dévoiler de nouveaux paliers, ... la rendre attrayante et surtout, montrer que vous êtes présent, actif et passionné par votre projet. N'oubliez pas de relancer votre campagne quelques jours avant la fin (et pas seulement quelques heures avant). De nombreuses personnes choisissent d'attendre et de voir comment évoluera votre projet avant de se décider. Il vous faut maintenant les re-contacter, car elles ont oublié et risquent de revenir trop tard.

6.3 L'après campagne : livrer les contreparties

Votre campagne a réussi, bravo ! Mais le travail n'est pas terminé, il ne fait que commencer ! Vous recevrez les fonds quelques jours après, mais n'attendez pas pour remercier une première fois vos soutiens. Ensuite, n'oubliez pas d'informer vos soutiens le plus souvent possible, à chaque jalon franchi, chaque difficulté rencontrée, ou au moins une fois de temps en temps si le développement suit son cours normalement. Apportez une attention particulière aux livraisons ! Vos soutiens sont vos premiers clients, ce sont eux qui parleront de vous, qui feront connaître votre projet et pourront publier des informations sur vos produits. Soignez-les !

7 Et l'ouverture dans tout ça ?

Je n'ai pas reparlé d'ouverture, de choix de licence, ou de libertés dans les paragraphes précédents, mais c'est volontaire. Faites de même. Je suis particulièrement attaché au libre, mais comme je l'ai déjà évoqué en introduction, le choix de la licence d'un projet peut être un sujet polémique, ou même seulement le choix des termes utilisés pour en parler. Et vous ne voulez pas de polémique sur la page de présentation de votre projet (si si, je vous assure !). Cette information peut faire partie des éléments de présentation du projet,

mais les détails devront figurer au mieux dans la FAQ, si la question vous est posée, ou de préférence sur le site annexe de présentation du projet.

Un projet libre n'est pas destiné aux seuls fervents défenseurs du libre (ou alors je n'ai pas tout compris), vous ne devez donc pas faire fuir les autres soutiens potentiels. Bien entendu, le choix d'une licence libre est préférable. Cela permettra à d'autres de reprendre votre projet si d'aventure vous n'arriviez pas au bout, et vos contributeurs n'auront ainsi pas tout perdu. Mais si vous lisez ces lignes, je pense que vous êtes déjà convaincu des bienfaits des licences libres, ou au moins des bienfaits de l'ouverture, je ne vous embêterai donc pas plus sur ce sujet.

8 La régulation par l'état, un problème d'actualité

Le dernier point que je voulais aborder, après je vous laisse tranquille. Je n'irai pas dans les détails, puisque je n'ai pas tout suivi, seulement lu un ou deux articles (mais vous aurez au moins un début d'information). Le gouvernement a annoncé l'établissement pour septembre 2013 d'un cadre juridique sécurisé pour le développement de la finance participative en France. Nous sommes en octobre et ce n'est pas encore fait, le débat n'est pas clos, et je n'ai pas vu passer d'information indiquant que des lois avaient été votées. Voir le lien vers l'article sur le site economie.gouv.fr mis à la fin des liens utiles. Voilà, au moins je ne dis pas de bêtises sur ce sujet :) ■

Liens utiles et autres lectures

- http://www.leguideducrowdfunding.com/
- http://fr.slideshare.net/PlaineImages/crowdfunding-web
- http://wiki.april.org/w/Argumentaire_Licence_libre_et_Crowdfunding
- http://wiki.april.org/w/Financement_du_logiciel_libre
- https://github.com/lockitron/selfstarter
- http://funding.openinitiative.com/ et un article sur LinuxFR : https://linuxfr.org/news/open-funding-cherche-des-testeurs-beta
- http://www.economie.gouv.fr/mise-en-ligne-dun-guide-financement-participatif-crowdfunding

GÉRER LES DIFFÉRENTES VERSIONS DU CODE SOURCE DE VOTRE PROJET

par Tristan Colombo

Lorsque l'on développe un logiciel, ce dernier est voué à évoluer. On ne part malheureusement pas de l'idée pour aboutir immédiatement au produit fini. Même si les spécifications sont précises, il y aura toujours de petits bugs à corriger et donc des lignes de codes seront modifiées, supprimées ou ajoutées. Mais que se passe-t-il lorsque plusieurs développeurs travaillent sur le même fichier, ou lorsqu'une correction n'en est pas une et qu'il faut revenir en arrière ? C'est là qu'interviennent les logiciels de gestion de versions concurrentes.

Il existe de nombreux logiciels de gestion de versions libres, parmi lesquels nous pouvons citer CVS, Subversion, Git, Mercurial, et Bazaar. Il faut bien évidemment faire un choix, chacun présentant des avantages et des inconvénients. Si vous n'êtes pas encore formaté à l'usage d'un de ces logiciels, prenez le temps avant de vous décider : on a ensuite beaucoup de mal à changer !

Dans cet article, je ne pourrai pas décrire en détails l'utilisation de tous ces logiciels et je commencerai donc par expliquer leur fonctionnement général et pourquoi ils sont indispensables. Ensuite, je détaillerai l'installation et l'utilisation de deux d'entre eux : **Subversion**, car très utilisé pour des raisons historiques, et **Git**, car faisant partie de la nouvelle vague des gestionnaires de versions et utilisé pour les versions du noyau de Linux.

1 La gestion de versions

Avant de rentrer dans le côté théorique de la gestion de versions, je voudrais vous présenter deux exemples réels d'organisation de développement d'un logiciel. Ces deux exemples se déroulent dans les locaux de deux startups que nous nommerons « Startup_X » et « Startup_Y ». La première entreprise emploie une dizaine de développeurs et la seconde une vingtaine. Je tiens à souligner le fait que ces deux sociétés produisent un logiciel et que leur cœur de métier est donc bien l'informatique.

1.1 Le tableau Velleda

Dans la Startup_X, il n'y a que huit développeurs qui travaillent dans une même pièce (on appelle ça un *open space* pour faire plus « in »). Le projet contient déjà une centaine de fichiers, répartis dans différents répertoires. Le code source est hébergé sur un serveur et chaque développeur a la possibilité de lire ou d'écrire sur ce serveur sans aucun mécanisme de contrôle. Donc, si deux développeurs travaillent en même temps sur le même fichier, le premier des deux qui aura fini recopiera son fichier sur le serveur et lorsque le second aura terminé son travail et qu'il recopiera également son fichier, les modifications de son collègue seront écrasées... Mais dans cette entreprise, il y a des gens qui réfléchissent ! Il a donc été décidé d'acheter un grand tableau Velleda sur lequel chaque développeur vient noter sur quels fichiers il travaille. À partir du moment où un nom de fichier est noté sur le tableau, il est « verrouillé » et plus personne ne doit le modifier en dehors de son « propriétaire » du moment.

Les problèmes de cette démarche : pendant que Jean travaille sur les fichiers **A**, **B** et **C**, Stéphanie travaille sur les fichiers **D** et **E**, mais aurait besoin de modifier le fichier **C**, alors elle attend (« Elle attend que le monde change, elle attend que changent les gens, (...) inexorablement elle attend »). Un développeur est donc immobilisé, ce qui va faire naître du stress et des tensions au sein de l'équipe. De plus, rien n'interdit à un développeur distrait de travailler sur un fichier déjà inscrit au tableau et donc, de perdre son travail ou d'effacer le travail d'un collègue. Un logiciel de gestion de versions aurait réglé le problème.

1.2 La duplication du code

La Startup_Y emploie plus de développeurs que la Startup_X, mais elle vend plusieurs logiciels qui sont développés en parallèle. Cette société utilise un logiciel de gestion de versions concurrentes... Donc, il ne doit pas y avoir de problème de perte de corrections. Eh bien non ! Le logiciel ne résout pas de lui-même tous les problèmes, encore faut-il savoir l'utiliser correctement ! Dans cette entreprise, on développe des fonctionnalités « sur-mesure » quand on livre le logiciel au client final et il a été décidé de créer un nouveau projet par client. Parfois, un client détecte un bug et le support demande qu'il soit corrigé. Quelques semaines plus tard, un autre client détecte le même bug... Comme la correction a été apportée sur le code d'un autre client, il faut soit la retrouver, soit recommencer le travail. Là encore, en dehors de la perte de productivité, il s'ensuit une perte de motivation des équipes de développement qui jouent à Sisyphe à longueur de journée (Sisyphe qui fut condamné à faire rouler un énorme rocher jusqu'au sommet d'une colline, mais qui n'y parvient jamais et doit éternellement recommencer, le rocher dévalant la pente à chaque tentative).

1.3 Fonctionnement général

Un logiciel de gestion de versions, abrégé en VCS en anglais pour *Version Control System*, réalise essentiellement quatre tâches :

1. Il permet de travailler à plusieurs développeurs sur un même code grâce à la gestion de verrous et la gestion des conflits si un même fichier a été modifié plusieurs fois ;
2. Il permet de faire évoluer plusieurs versions d'un même logiciel en parallèle par le biais de ce que l'on appelle des branches. On peut ainsi avoir par exemple une version de développement et une version de production ;
3. Il garantit la sécurité et l'intégrité des données ;
4. Il conserve un historique des modifications de manière à pouvoir reprendre le code avant n'importe laquelle d'entre elles.

De manière générale, les logiciels de gestion de versions utilisent un dépôt (ou *repository*) qui va conserver l'historique des modifications. À chaque fois qu'un développeur enregistre les modifications qu'il a effectuées sur des fichiers, il réalise une mise à jour du code (encore appelé *commit*) et cette action aboutit à une nouvelle révision du projet.

Pour les développements en parallèle, il est possible de créer des branches. On peut ainsi conserver plusieurs versions d'un même projet et continuer à corriger des problèmes sur d'anciennes versions. La figure 1 montre un exemple de branches créées pour un projet : la version 1.0 est la version du code de départ. On passe ensuite soit sur la branche 1.1, qui est la version suivante du programme, soit à la branche 1.a, qui constitue une branche de test ou de prototypage. Par la suite, des modifications sont encore apportées à ces branches et on aboutit respectivement aux versions 1.2 et 1.b. À partir de la version 1.1, il a été décidé de débuter une nouvelle version majeure (la version 2.0), mais pour pouvoir continuer à maintenir le code de la version 1.x, il a fallu créer une nouvelle branche.

On pourrait aussi imaginer une fusion de branches : après avoir fait diverger le code, on choisit de ne maintenir plus qu'une seule version qui contiendra les améliorations des deux branches précédentes.

Pour nommer les différentes branches et définir les versions du projet, on utilise un système de tags : des étiquettes qui permettent de repérer les fichiers dans un certain état.

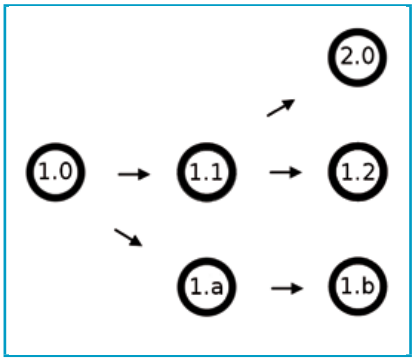


Fig. 1 : Exemple de projet contenant plusieurs branches de développement

Chaque fois qu'un développeur travaille sur le code source, il possède sa propre copie des fichiers. Lors du commit, si un développeur a travaillé sur le même fichier et mis à jour celui-ci, il y aura un conflit et le logiciel n'arrivera pas à le gérer tout seul. Voici un tableau illustrant le déroulement des modifications sur un fichier : voir tableau ci-dessous.

Développeur A	Développeur B
Récupère le code du fichier `index.html`. La première ligne contient : `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`	Récupère le code du fichier `index.html`. La première ligne contient : `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`
Modifie la première ligne en : `<!-- Page d'index -->`	Modifie la première ligne en : `<!doctype html>`
Modifie une autre ligne.	Valide les modifications.
Valide les modifications.	
Conflit ! La première ligne a déjà été modifiée par ailleurs.	

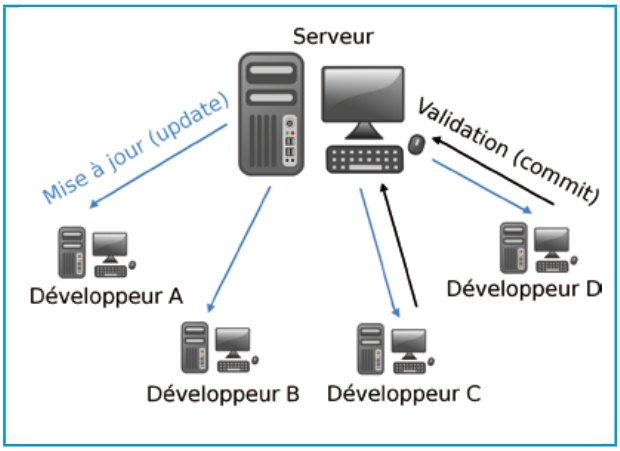


Fig. 2 : Modèle de gestion de versions centralisé

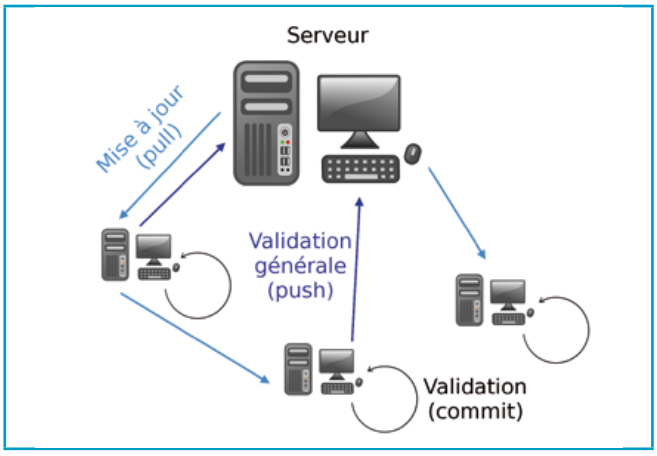


Fig. 3 : Modèle de gestion de versions décentralisé

Ici, il faut que le dernier développeur validant son travail indique expressément quelle est la bonne version : la précédente, la sienne ou une fusion des deux. Il est impossible de réaliser un commit avant d'avoir résolu le conflit.

Les logiciels de gestion de versions adoptent essentiellement deux approches : soit un modèle **centralisé** avec un serveur accessible à distance et qui va centraliser les informations dans le dépôt, soit un modèle **distribué** où chaque développeur possède ses propres copies du dépôt incluant tout l'historique. Le premier modèle, dont le fonctionnement est illustré en figure 2, est celui utilisé par exemple par Subversion. Le second modèle, illustré en figure 3, est celui utilisé par la nouvelle génération de gestionnaires de versions tels que Git et Mercurial.

Intéressons-nous maintenant à deux implémentations de ces modèles : Subversion pour le modèle centralisé et Git pour le modèle décentralisé.

2 Subversion

Subversion a été développé dans les années 2000 pour remplacer et combler les lacunes de l'antique CVS (pour *Concurrent Versions System*) créé en 1990. CVS n'est plus maintenu de manière officielle, mais malgré cela, des projets continuent à l'utiliser... par peur de migrer à un autre système. Subversion, dont le nom abrégé est SVN, utilise le même modèle que CVS et il s'en est fortement inspiré. C'est un peu le CVS 2.0.

2.1 Installation

Pour les distributions basées sur Debian, l'installation se fait à l'aide d'un simple **aptitude** :

```
$ sudo aptitude install subversion
```

Subversion va utiliser un dépôt qu'il nous faut définir. Vous pouvez par exemple stocker les versions dans **/var/svn** :

```
$ sudo mkdir /var/svn
```

Pour l'instant, Subversion fonctionne seulement localement, mais nous pouvons déjà le tester en utilisant la commande **svnadmin** qui permet de créer des projets :

```
$ sudo svnadmin create /var/svn/test_svn
```

Si vous vous rendez dans le répertoire **/var/svn**, vous verrez qu'un nouveau répertoire **test_svn** a été créé et que celui-ci contient des fichiers et des répertoires. La configuration du dépôt pour ce projet se trouve dans le fichier **/var/svn/test_svn/conf/svnserve.conf**. Éditez ce fichier et vous constaterez que toutes les lignes sont commentées.

Si vous ne souhaitez pas que n'importe qui puisse se connecter à votre serveur, il va falloir modifier quelques options :

```
01: [general]
02: anon-access = none
03: auth-access = write
04: password-db = passwd
05: realm = test_svn repository
```

En ligne 2, nous indiquons que les utilisateurs anonymes (donc non identifiés) ne peuvent plus rien faire (les choix possibles sont (**none**, **read** ou **write**)). En ligne 3, nous indiquons les permissions pour les utilisateurs identifiés. La ligne 4 permet de spécifier le nom du fichier qui contiendra les couples identifiant/mot de passe des utilisateurs autorisés.

Il faut bien sûr ensuite créer la liste des utilisateurs dans le fichier **/var/svn/test_svn/conf/passwd** :

```
01: [user]
02: linus = linusSeCrET
03: toto = MoTdEpAsSe
```

Les mots de passe étant stockés en clair, il faut rendre ce fichier non accessible en lecture :

```
$ sudo chmod o-x /var/svn/test_svn/conf/
passwd
```

Pour pouvoir utiliser Subversion, il faut ensuite lancer le serveur en tant que *daemon* (option **-d**) et en précisant le répertoire racine du dépôt (option **-r**) :

```
$ sudo svnserve -d -r /var/svn
```

Cette ligne devra être exécutée à chaque démarrage de la machine...

Il serait donc intéressant que ce service se lance de manière automatique. Malheureusement, il va falloir effectuer cette opération à la main en créant un fichier **/etc/init.d/svnserve** :

```
01: #!/bin/sh
02:
03: do_start () {
04:     svnserve -d -r /var/svn --pid-file
/var/run/svnserve.pid
05: }
06: do_stop () {
07:     start-stop-daemon --stop --quiet
--pidfile /var/run/svnserve.pid
08: }
09:
10:
11: case "$1" in
12:     start)
13:         do_start
14:         ;;
15:     stop)
16:         do_stop
17:         exit $?
18:         ;;
19:     restart)
20:         do_stop
21:         sleep 1s
22:         do_start
23:         ;;
24:     reload|force-reload)
25:         echo "Error: argument '$1' not
supported" >&2
26:         exit 3
27:         ;;
28:     *)
29:         echo "Usage: $0
start|stop|restart" >&2
30:         exit 3
31:         ;;
32: esac
```

Il faut ensuite rendre ce fichier exécutable et indiquer qu'il doit être exécuté à chaque démarrage :

```
$ sudo chmod +x /etc/init.d/svnserve
$ sudo update-rc.d svnserve defaults
```

Vous avez désormais accès au serveur (avec le bon couple identifiant/mot de passe) et vous pouvez récupérer une copie des sources (certes vide pour l'instant...) :

```
$ svn checkout svn://localhost/test_svn
Authentication realm: <svn://localhost:3690>
test_svn repository
Username: linus
Password for 'linus':
Checked out revision 0.
```

Sur la sortie de la commande, nous pouvons voir que le port par défaut de **svn** est le port 3690. Vous trouverez dans votre répertoire courant un nouveau répertoire **test_svn** vide, qui correspond au projet que nous avions créé. À partir de ce répertoire, vous pourrez mettre à jour votre code et valider des modifications... Mais nous verrons cela après avoir réalisé une installation pour un serveur distant.

2.2 Serveur distant

L'installation précédente doit avoir été effectuée au préalable.

La manière la plus simple d'utiliser Subversion avec un serveur distant est d'installer le module Apache pour Subversion et de l'activer :

```
$ sudo aptitude install apache2 libapache2-svn
$ sudo a2enmod dav_svn
$ sudo a2enmod authz_svn
```

Éditez ensuite le fichier **/etc/apache2/mods-enabled/dav_svn.conf**, dé-commentez et modifiez les lignes suivantes :

```
01: <Location /svn>
02:   DAV svn
03:   Require valid-user
04:   SVNParentPath /var/svn
05:   AuthType Basic
06:   AuthName "Subversion Global Repository"
07:   AuthUserFile /var/svn/conf/htpasswd
08:      <IfModule mod_authz_svn.c>
09:        AuthzSVNAccessFile /var/svn/conf/access
10:      </IfModule>
09: </Location>
```

La première ligne indique que le serveur sera accessible par http://monserveur.com/svn où **monserveur.com** représente le nom de votre serveur (que vous pouvez remplacer par son adresse IP). Les lignes 3, 5, et 7 à 10 permettent de ne plus accepter que des utilisateurs authentifiés présents dans le fichier **/var/svn/conf/htpasswd**. Pour créer le premier utilisateur, il faudra exécuter :

```
$ sudo mkdir /var/svn/conf
$ sudo htpasswd -c /var/svn/conf/htpasswd linus
New password:
Re-type new password:
adding password for user linus
```

Par la suite, pour ajouter d'autres utilisateurs, il faudra omettre le paramètre **-c** qui indiquait une création.

Une fois les utilisateurs créés, il faudra leur associer des droits d'accès aux projets. Ceci est fait dans le fichier **/var/svn/conf/access** que nous avons indiqué en ligne 8 du fichier **dav_svn.conf** :

```
01: [groups]
02: dev = linus, tonton_richard
03:
04: [test_svn:/]
05: linus = rw
06: * = r
07:
08: [autre_projet:/]
09: @dev = rw
```

Vous pouvez définir les droits pour chaque utilisateur, comme pour le projet **test_svn** dans les lignes 4 à 6 (où * représente les droits pour les autres utilisateurs), ou bien utiliser des groupes qui se définissent dans une section **[groups]** (lignes 1 et 2). Pour utiliser un groupe, on préfixe son nom par le caractère **@**. Ainsi, la ligne 9 est équivalente à :

```
linus = rw
tonton_richard = rw
```

Pour que vos modifications soient actives, il faut redémarrer Apache :

```
$ sudo service apache2 restart
```

La connexion au projet **test_svn** se fera alors par l'adresse http://monserveur.com/svn/test_svn :

```
$ svn checkout http://monserveur.com/svn/test_svn
Authentication realm: <http://monserveur:80>
test_svn repository
Username: linus
Password for 'linus':
Checked out revision 0.
```

Voyons maintenant les commandes de base qui permettent de travailler avec Subversion.

2.3 Utilisation

Nous avons déjà vu la commande **checkout**, qui permet de créer une copie locale du dépôt d'un projet. Modifier cette copie ne va pas modifier le code du dépôt automatiquement, il va falloir exécuter une commande particulière :

```
$ svn add nouveau_fichier
```

Si vous choisissez d'ajouter un répertoire, l'ensemble des fichiers de ce répertoire sera ajouté récursivement.

Pour supprimer un fichier, ce sera :

```
$ svn delete fichier
```

Et pour que ces données soient propagées sur le serveur, il faudra les valider par un commit :

```
$ svn commit -m "Message expliquant les modifications"
```

Si vous souhaitez récupérer la dernière version des fichiers sur le dépôt (ce qu'il faut toujours faire avant de commencer à travailler sur le code), il faudra exécuter :

```
$ svn checkout
```

Pour récupérer une version plus ancienne d'un fichier, il faudra préciser le numéro de révision et pour connaître le numéro de révision approprié (sous la forme **rXX**), la commande **log** fournira l'historique des modifications :

```
$ svn log fichier
------------------------------------------------------------------------
r12 | linus | 2013-08-27 16:33:12 +0200 (mar, 27 août 2013) | 4 lines

Modification de la fonction de calcul du poids
------------------------------------------------------------------------
r11 | linus | 2013-08-26 10:10:54 +0200 (lun, 26 août 2013) | 1 line

Correction bug affichage
------------------------------------------------------------------------
...
$ svn update -r numero_revision fichier
```

Pour créer de nouvelles branches et leur ajouter un tag, il faudra utiliser la commande **copy** :

```
$ svn copy http://monserveur.com/svn/test_svn/trunk   http://
monserveur.com/svn/test_svn/branches/nom_de_la_branche -m
"Etiquette de la branche"
```

Enfin, si vous travailliez sur un projet qui n'était pas géré par Subversion, vous pouvez l'importer pour créer un nouveau dépôt par :

```
$ svn import repertoire_projet http://monserveur.com/svn
```

Je n'ai présenté ici que les commandes de base. Pour plus d'informations, vous pourrez consulter le livre (en français) « Gestion de versions avec Subversion » de Collins-Sussman, Fitzpatrick et Pilato sur http://svnbook.red-bean.com/.

2.4 Interfaces graphiques

Il existe de très nombreuses interfaces graphiques côté client pour Subversion (sans compter les extensions pour Eclipse). Il y a par exemple rapidSVN, qui s'installe depuis les dépôts :

```
$ sudo aptitude install rapidsvn
```

Le lancement de l'interface se fera par un appel à la commande **rapidsvn**. L'intérêt de cette interface est sa simplicité d'utilisation et de configuration, comme le montre la figure 4.

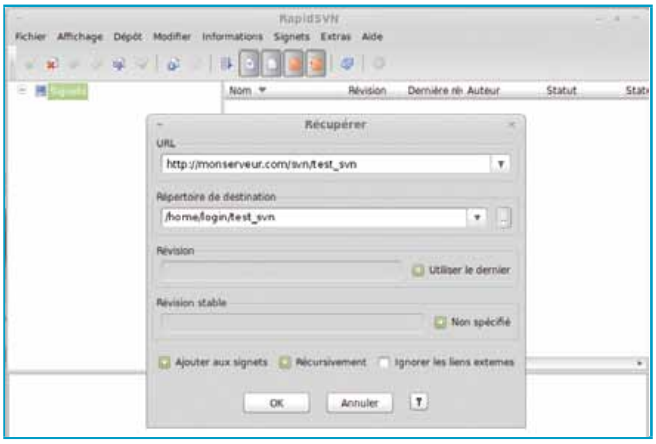


Fig. 4 : RapidSVN, une interface simple pour Subversion

Si vous utilisez Nautilus comme navigateur de fichiers, RabbitVCS permet d'intégrer Subversion directement à l'intérieur. Tout est disponible depuis les dépôts :

```
$ sudo aptitude install rabbitvcs-nautilus nautilus-script-collection-svn
```

Vous devrez ensuite activer le support de Subversion :

```
$ nautilus-script-manager enable Subversion
```

Après redémarrage de Nautilus, vous aurez directement accès à un menu spécial ***Subversion*** en effectuant un clic droit sur un fichier ou un dossier, comme le montre la figure 5.

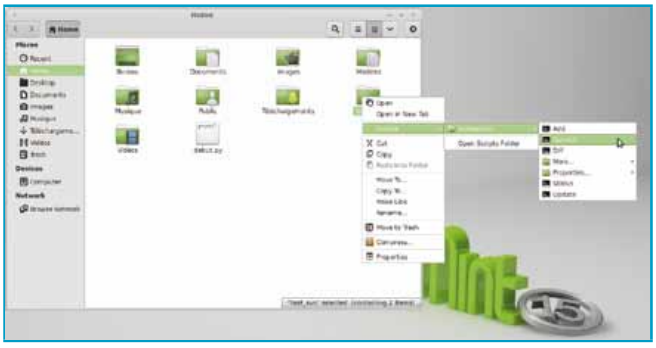


Fig. 5 : Intégration de Subversion dans Nautilus grâce à RabbitVCS

3 Git

Git est né en 2005, suite à l'abandon du gestionnaire de versions BitKeeper pour suivre les modifications du noyau Linux. Git est une initiative de Linus Torvalds, qui visait à créer un logiciel capable de gérer de très grands projets tout en restant rapide. Au final, les contraintes de départ furent parfaitement suivies et ont abouti à un logiciel de versions très rapide et très souple d'utilisation.

3.1 Installation

L'installation en local se fait classiquement par **aptitude** :

```
$ sudo aptitude install git
```

La configuration est ensuite minimale. On va commencer par activer la coloration des messages :

```
$ git config --global color.ui auto
```

Ensuite, nous déclarerons les utilisateurs qui seront amenés à travailler avec Git. Ici, deux options sont possibles :

- Soit les utilisateurs travailleront sur tous les projets et ils sont alors déclarés de manière globale :

```
$ git config --global user.name "Moi"
$ git config --global user.email "moi@
mon_adresse.com"
```

- Soit les utilisateurs seront déclarés pour un projet particulier. Nous prendrons ici pour exemple un projet se trouvant dans **/home/login/dev** :

```
$ cd /home/login/dev
$ git config user.name "Moi"
$ git config user.email "moi@mon_adresse.com"
```

Vous pouvez maintenant tester votre installation en créant un premier projet :

```
$ cd /home/login/dev
$ git init --shared=group
```

Si votre répertoire contient déjà des fichiers, il faudra les ajouter pour une première validation :

```
$ git add .
$ git commit -m "Commit initial"
```

Si vous le souhaitez, vous pouvez définir un modèle de message pour tous vos commits en créant un fichier **.gitmessage.txt** dans votre répertoire personnel :

```
01: Titre du commit
02:
03: Description
04:
05: # S'il s'agit d'une résolution de ticket
décommenter la ligne suivante
06: #[ticket: X]
```

Déclarez ensuite ce fichier en tant que modèle :

```
$ git config --global commit.template
~/.gitmessage.txt
```

Lors des commits suivants, vous n'aurez plus à indiquer directement un message (tapez simplement **git commit**) : votre éditeur par défaut s'ouvrira sur le modèle que vous pourrez modifier. Si vous voulez changer d'éditeur, vous pourrez le faire à l'aide de la commande :

```
~$ git config --global core.editor mon_editeur
```

En fait, à chaque fois que vous invoquez une commande **git config**, des informations sont inscrites dans le fichier **.gitconfig** de votre répertoire personnel. On peut configurer beaucoup de choses dans ce fichier. Pour en savoir plus, vous pourrez consulter le livre « Pro Git » de Scott Chacon (en français) sur http://git-scm.com/book/fr.

Nous verrons dans la suite d'autres commandes de travail, mais vous pouvez constater que cette phase est beaucoup plus simple qu'avec Subversion. Voyons maintenant ce qu'il en est si nous choisissons d'installer Git sur un serveur distant.

3.2 Serveur distant

Le paquetage de Git étant installé, il faut simplement créer un répertoire qui contiendra les dépôts :

```
$ sudo mkdir /var/git
```

Par défaut, il existe un groupe **git**. Nous allons donner les fichiers de **/var/git** à ce groupe :

```
$ cd /var
$ sudo chgrp -R git git
$ sudo chmod -R g+swX git
```

Nous initions ensuite un projet de test :

```
$ mkdir /var/git/test_git
$ cd /var/git/test_git
$ git init --shared=group
```

Comme vous devez vous connecter à votre serveur en SSH, vous pouvez tester la connexion à Git depuis un ordinateur externe en clonant le dépôt de test :

```
$ git clone ssh://tristan@server/var/git/test_git
Cloning into 'test_git'...
warning: You appear to have cloned an empty
repository.
```

Ici, « server » est le nom ou l'adresse IP de votre serveur. Attention : si vous avez modifié le port de connexion SSH, il faudra le préciser. Par exemple, pour un serveur dont l'IP est 14.22.123.12 et le port SSH 1515, nous devrons utiliser : **ssh://tristan@14.22.123.12:1515/var/git/test_git**.

À chaque accès le mot de passe vous sera demandé, mais vous pourrez automatiser l'identification avec un système de clé publique/clé privée (voir la commande **ssh-keygen**). Cette partie, un peu plus longue, ne sera pas traitée dans cet article.

3.3 Utilisation

Dans les commandes précédentes, nous avons déjà vu comment cloner un projet, ajouter des fichiers et valider des modifications. Le fait de valider les modifications avec un commit ne transfère pas ces modifications sur le serveur. Pour cela, il faut appeler une autre commande, le « push » :

```
$ git push origin master
```

Ici, **origin** est le nom du dépôt distant et **master** le nom de la branche (branche par défaut). Du coup, vous pourriez être tenté de créer vos propres branches... Pour créer une branche et s'y placer, vous devrez effectuer deux commandes :

```
$ git branch ma_nouvelle_branche
$ git checkout ma_nouvelle_branche
```

Ou alors, vous pouvez utiliser le raccourci suivant :

```
$ git checkout -b ma_nouvelle branche
```

Vous pouvez connaître la liste des branches accessibles en utilisant l'option **-a** de la commande **branch** :

```
$ git branch -a
* master
  ma_nouvelle_branche
```

Pour fusionner une branche dans une autre (après résolution d'un problème par exemple), il faut se placer dans la branche cible et effectuer un « merge » :

```
$ git checkout master
$ git merge ma_nouvelle_branche
```

Pour ajouter une étiquette à une révision, il faudra utiliser la commande **tag** :

```
$ git tag -a v1.0.1 -m "Version 1.0.1"
```

La liste des tags est donnée par l'option **-l** et il faut savoir que les étiquettes ne sont pas automatiquement transmises au serveur lors du « push ». Il faut utiliser l'option **--tags** :

```
$ git push origin --tags
```

Git fonctionnant en utilisant des objets, il faut parfois le forcer à faire un peu le ménage :

```
$ git repack && git prune
```

Avec la commande précédente, vous gagnerez de l'espace disque (à effectuer plus ou moins régulièrement en fonction du nombre de commits).

Revenons sur la gestion des fichiers : effacer ou déplacer un fichier dans l'arborescence de votre projet ne suffit pas... Il faut le signaler à Git ! Pour cela, les commandes **git rm** et **git mv** fonctionnent exactement comme dans un shell, où la première retire un fichier et la seconde le déplace.

Il faut également savoir que si vous souhaitez que certains fichiers ne soient pas partagés, il suffit de les lister dans un fichier **.gitignore** à la racine de votre projet (un nom de fichier par ligne).

L'historique des révisions est donné par la commande **git log** :

```
$ git log
commit 8bcd0214643bf5595fefc200f40fad75ba54ab67
Author: Tristan <moi@mon_mail.com>
Date:   Fri Aug 23 20:24:32 2013 +0200

    maj roadmap

commit 907457d33876c12e897eab0c3b2133150653c04bf
...
```

Enfin, je ne pouvais pas ne pas parler des « hooks », qui sont de petites actions programmées. Si vous vous rendez dans le répertoire de votre dépôt **/var/git/test_git/.git/hooks**, vous verrez de nombreux fichiers du type **pre-commit.sample**, **post-commit.sample**, etc. Si vous voulez envoyer un mail après chaque commit, vous créerez par exemple un fichier **post-commit** contenant :

```
01: #!/bin/bash
02:
03: mail -s "Nouveau commit !" user@mail.com < "Mettez à jour votre
projet"
```

Une fois le fichier rendu exécutable (par **chmod -x**), l'action sera déclenchée automatiquement.

3.4 Interfaces graphiques

Comme avec Subversion, de nombreuses interfaces graphiques existent. On peut voir sur la figure 6 que l'une de ces interfaces, GitG, va permettre d'exécuter des actions de manière graphique et fournir un affichage peut-être plus lisible (pour ceux qui sont allergiques au mode console).

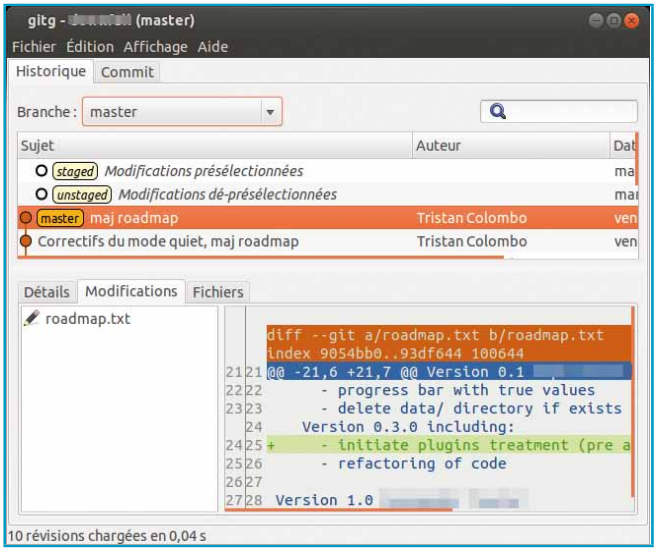


Fig. 6 : Interface graphique pour Git GitG

Pour terminer, là encore comme avec Subversion, vous pouvez installer RabbitVCS pour intégrer directement Git au navigateur de fichier Nautilus.

Conclusion

Un gestionnaire de versions concurrentes n'est pas un gadget, c'est un outil indispensable pour faire vivre tout projet, à partir du moment où il contient plus d'un fichier ou plus d'un développeur (et même là, il peut être utile...).

Pour ma part, j'utilise Git en mode console et j'en suis très satisfait pour sa souplesse, sa rapidité et son ergonomie. Comme avec tout logiciel, vous trouverez des fanatiques de telle ou telle autre solution, qui vous diront par exemple que Git est nul et qu'il faut choisir Mercurial, ou inversement. Prenez le temps de tester quelques solutions et de vous faire votre opinion avant de faire un choix définitif car, au final, c'est vous qui l'utiliserez tous les jours... ■

Abonnez-vous !

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Consultez l'ensemble de nos offres sur : boutique.ed-diamond.com !

11 Numéros de GNU/Linux Magazine

60€*

au lieu de 86,90 €* en kiosque

Économie 26,90€

Économisez plus de 30%*

* Sur le prix de vente unitaire France Métropolitaine

Nouveauté 2014

TOUS LES HORS-SÉRIES PASSENT EN GUIDE !

Un nouveau format avec une reliure de luxe pour ces Guides de référence de 128 pages à conserver dans votre bibliothèque !

Découvrez tous les Guides déjà parus sur

boutique.ed-diamond.com

***OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE. Pour les tarifs hors France Métropolitaine, consultez notre site : boutique.ed-diamond.com**

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- Économisez 26,90 €/an ! (soit plus de 3 magazines offerts !)

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur boutique.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletter des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles de nos partenaires.

GNU LINUX MAGAZINE / FRANCE

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **www.ed-diamond.com/cgv** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

Abonnez-vous !

➡Tous les abonnements incluant GNU/Linux Magazine :

offre LM

60€* au lieu de **86,90€**** en kiosque

Économie 26,90€

INCLUS : GNU/Linux Magazine (11nos)

offre LM+

115€* au lieu de **164,30€**** en kiosque

Économie 49,30€

INCLUS : GNU/Linux Magazine (11nos) + ses 6 Guides

NOUVEAUTÉ 2014
TOUS LES HORS-SÉRIES PASSENT EN **GUIDE !**

offre T

198€* au lieu de **277,10€**** en kiosque

Économie 79,10€

INCLUS : GNU/Linux Magazine (11nos), Open Silicium (4nos), MISC (6nos), Linux Pratique (6nos) et Linux Essentiel (6nos)

offre T+

299€* au lieu de **411,20€**** en kiosque

Économie 112,20€

INCLUS : GNU/Linux Magazine (11nos) + ses 6 Guides, Linux Pratique (6nos) + ses 3 Guides, MISC (6nos) + ses 2 Hors-Séries, Open Silicium (4nos) et Linux Essentiel (6nos)

* Tarifs France Métro (F) ** Base tarifs kiosque zone France Métro (F)

Consultez l'ensemble de nos offres d'abonnements sur :

boutique.ed-diamond.com

Vous pouvez également commander par Tél. : +33 (0)3 67 10 00 20 / Fax : +33 (0)3 67 10 00 21

➡ Nos Tarifs s'entendent TTC et en euros

		F	OM1	OM2	E	RM
		France Métro	Outre-Mer		Europe	Reste du Monde
LM	Abonnement GLMF	60 €	75 €	96 €	83 €	90 €
LM+	Abonnement GLMF + GLMF HS (6 Guides)	115 €	147 €	190 €	160 €	173 €
T	Abonnement GLMF + MISC + OS + LP + LE	198 €	253 €	325 €	276 €	300 €
T+	Abonnement GLMF + GLMF HS (6 Guides) + MISC + MISC HS + OS + LP + LP HS (3 Guides) + LE	299 €	382 €	491 €	415 €	448 €

• OM1 : Guadeloupe, Guyane française, Martinique, Réunion, St Pierre et Miquelon, Mayotte

• OM2 : Nouvelle Calédonie, Polynésie française, Wallis et Futuna, Terres Australes et Antarctiques française

MA FORMULE D'ABONNEMENT :

Offre	Zone	Tarif
☐ LM		
☐ LM+		
☐ T		
☐ T+		

J'indique la somme due : (Total) €

Exemple :
Je souhaite m'abonner à l'ensemble des magazines + tous les Hors-série/Guides et je vis en Belgique. Je coche donc l'offre **T+** (la totale avec tous les Hors-Série/ Guides) puis ma zone (E), le montant sera donc 415 euros.

Je choisis de régler par :

❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond

❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

LES OUTILS DE GESTION DE TICKETS

Tristan Colombo

Une des bases du fonctionnement du logiciel libre est la participation des utilisateurs à l'amélioration du logiciel qu'ils utilisent. Cela peut aller du simple avertissement d'un bug rencontré, jusqu'à la proposition de patches. L'utilisation d'un gestionnaire de tickets permet une communication plus simple entre les utilisateurs et les développeurs, ou même simplement entre les développeurs.

Ce que l'on nomme « ticket » est en fait une note aux développeurs signalant une anomalie quelle qu'elle soit : bogue, fonction mal implémentée, régression, documentation erronée ou absente, problème d'ergonomie, etc. Les tickets peuvent également être utilisés pour signaler une demande d'évolution.

Les outils de gestion de tickets permettent de centraliser ces diverses demandes, de leur associer un identifiant unique appelé « numéro de ticket » et ils fournissent une interface graphique permettant de saisir la description de l'anomalie ou de la demande d'évolution. Un ticket possède un degré d'urgence et un statut :

- Le **degré d'urgence** indique la rapidité avec laquelle l'anomalie devra être résolue (ou que l'on souhaiterait voir résolue). Une fois ce degré fixé par l'utilisateur, l'administrateur peut le modifier. En général, il existe trois niveaux d'urgence : critique, urgent et normal.
- Le **statut** indique l'état du ticket : celui-ci a-t-il été lu ou pas (dans ce cas il est nouveau) ? A-t-il été accepté par les développeurs (s'il est reproductible, s'il ne s'agit pas d'un doublon, etc.) ? A-t-il été assigné à un ou plusieurs développeur(s) ? Est-il résolu (le ticket est alors fermé), ou le problème est-il réapparu (on « ré-ouvre » un ticket) ? En général, les statuts que l'on peut associer à un ticket sont les suivants : *new*, *accepted*, *assigned*, *closed* et *reopened*.

La communication entre la personne qui ouvre le ticket et les développeurs est facilitée par la possibilité de laisser des messages qui apparaîtront dans l'historique du ticket. Il sera ainsi possible de demander des informations supplémentaires pour pouvoir traiter l'anomalie plus rapidement et en cas de régression, l'échange sera toujours accessible.

Enfin, le fait d'identifier une anomalie par un ticket comportant un numéro permet de reporter ce numéro lors de la validation des fichiers modifiés à l'aide du gestionnaire de versions : en cas de régression, il sera alors très simple de retrouver les fichiers impactés et les modifications effectuées.

Il existe pléthore de logiciels libres de gestion de tickets fonctionnant sur la base d'une interface web. Il est d'ailleurs fréquent que pour un projet dont le site est monprojet.org, la gestion des tickets soit mise en place sur le sous-domaine issues.monprojet.org. Je ne présenterai ici que les gestionnaires les plus utilisés et je supposerai que votre serveur est correctement installé et opérationnel.

1 Bugzilla, la gestion de tickets de chez Mozilla

Bugzilla **[1]** est le gestionnaire de tickets de la fondation Mozilla utilisé pour suivre ses projets. D'autres projets célèbres utilisent également ce gestionnaire : GNOME, KDE, Linux Kernel, etc. Il est distribué sous licence MPL (*Mozilla Public License*). Pour l'installation, nous utiliserons une base de données PostgreSQL sur une distribution basée sur Debian. La première étape est de s'assurer que le système possède bien les logiciels requis :

```
$ sudo aptitude install perl postgresql
```

Il faut ensuite télécharger le code source, le décompresser et lancer un script qui va vérifier que tous les modules Perl nécessaires sont présents :

```
$ wget http://ftp.mozilla.org/pub/mozilla.
org/webtools/bugzilla-4.4.tar.gz
$ tar -zxvf bugzilla-4.4.tar.gz
$ cd bugzilla-4.4/
$ ./checksetup.pl --check-modules
```

Si le système vous signale que des modules sont manquants, exécutez la commande suivante pour les télécharger et les installer :

```
$ sudo perl install-module.pl --all
```

Si seulement quelques modules sont manquants, au lieu de la commande précédente vous pourrez exécuter pour chaque module **nom_module** :

```
$ sudo perl install-module.pl nom_module
```

Vous pouvez vérifier à nouveau que tous les modules sont présents (en toute logique la réponse est oui). Il est probable que cette commande vous indique des modules optionnels manquants mais, comme leur nom l'indique, ils sont optionnels et n'empêcheront pas le bon fonctionnement de Bugzilla. Vous pourrez les installer par la suite si vous avez besoin des fonctionnalités qu'ils procurent.

```
$ ./checksetup.pl --check-modules
```

Relancez maintenant la commande **checksetup.pl** sans paramètre pour générer le fichier de configuration **localconfig** :

```
$ ./checksetup.pl
```

Il faut ensuite éditer le fichier **localconfig** pour indiquer les paramètres de connexion à la base de données :

```
# Nom du groupe Apache
$webservergroup = 'www-data';

# Spécifie le type de base de données :
mysql, pg pour PostgreSQL, Oracle ou Sqlite
$db_driver = 'pg';

# Indiquez ici localhost pour une
installation locale ou l'adresse de votre
serveur
$db_host = 'localhost';

# Nom de la base de données
$db_name = 'bugs';

# Nom de l'utilisateur
$db_user = 'bugs';

# Mot de passe
$db_pass = 'mon_MoTDePaSse';

# Port sur lequel Bugzilla sera servi (0 =
port par défaut)
$db_port = 0;
```

Nous nous apercevons ici que nous n'avons pas paramétré de base de données PostgreSQL... Il faut donc remédier à cela en restant cohérent avec les données que nous avons indiquées dans le fichier de configuration :

```
$ sudo su - postgres
$ createuser -U postgres -S -D -R -E -P bugs
Enter password for new role:
Enter it again:
$ createdb -U postgres -O bugs -E UTF-8 bugs
$ exit
```

Un nouvel appel à **checksetup.pl** va créer les tables de Bugzilla et vous demander de renseigner les informations relatives au compte administrateur :

```
$ sudo ./checksetup.pl
```

Si vous n'aviez pas installé Bugzilla dans un répertoire lisible par Apache, il est temps de le déplacer :

```
$ cd ..
$ sudo mv bugzilla-4.4 /var/www/bugzilla
$ sudo chown -R www-data:www-data /var/www/
bugzilla
```

Il faut maintenant configurer Apache en utilisant le module CGI qui est normalement actif par défaut. Nous allons quand même l'activer au cas où :

```
$ sudo a2enmod cgi
Module cgi already enabled
```

Il faut ensuite créer un VirtualHost dans **/etc/apache2/sites-available/bugzilla** :

```
<VirtualHost localhost:80>
  ServerAdmin tristan.colombo@info2dev.com
  ServerName localhost
  DocumentRoot /var/www/bugzilla

  <Directory /var/www/bugzilla>
    AddHandler cgi-script .cgi
    Options +ExecCGI
    DirectoryIndex index.cgi index.html
    AllowOverride Limit FileInfo Indexes Options
  </Directory>
</VirtualHost>
```

Pour une installation non locale, vous n'aurez qu'à remplacer **localhost** par le nom de votre serveur et éventuellement ajouter les lignes suivantes pour utiliser le module **proxy_http** si vous utilisez un sous-domaine :

```
ProxyRequests Off
<Proxy *>
   Order deny,allow
   Allow from all
</Proxy>

ProxyPreserveHost On
ProxyPass / http://localhost:80
ProxyPassReverse / http://localhost:80
```

L'activation du site se fait par :

```
$ sudo a2ensite bugzilla
$ sudo service apache2 reload
```

En lançant un navigateur sur http://localhost (à condition que vous ayez configuré Bugzilla en local sur le port par défaut), vous obtiendrez la page d'accueil du gestionnaire de tickets. Pour vous connecter en administrateur, vous devrez cliquer sur le lien « Log in » en bas à droite ou en haut à droite, puis indiquer l'adresse mail que vous aviez renseignée lors de la création de l'administrateur. Vous accéderez alors aux réglages des préférences : thème, champs du formulaire de saisie des tickets, etc.

Un système d'extensions permet d'apporter des fonctionnalités supplémentaires à Bugzilla (voir https://wiki.mozilla.org/Bugzilla:Addons). Il existe également des extensions non-officielles, comme FreedomSponsors disponible sur https://github.com/freedomsponsors/freedomsponsors-bugzilla-plugin. Cette extension ajoute automatiquement à chaque ticket un lien vers le site freedomsponsors.org pour que les utilisateurs puissent proposer une participation financière en échange d'un développement spécifique.

D'un point de vue visuel, il faut reconnaître que c'est quand même bien chargé une fois passée la page d'accueil (voir figure 1). L'équipe de développement sait qu'il y a un problème de design et la prochaine version majeure (la version 5.0) contiendra un nouveau modèle. Si vous ne pouvez pas attendre jusque-là (et que vous disposez du temps nécessaire), il est tout à fait possible de créer son propre thème.

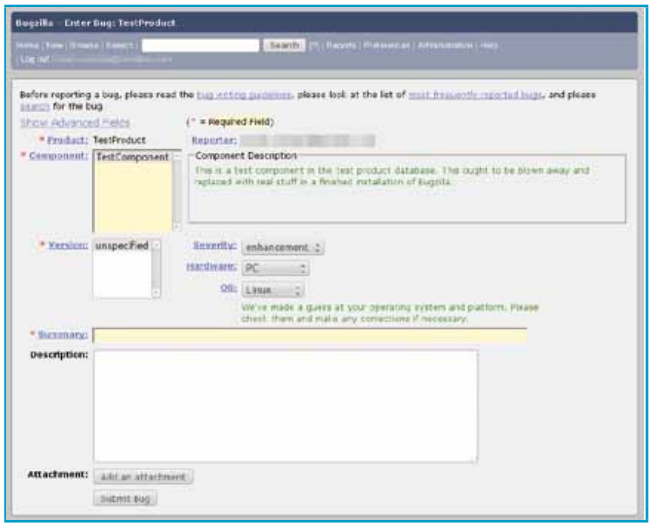


Fig. 1 : Saisie d'un ticket dans Bugzilla

2 Trac

Trac **[2]** est un logiciel de gestion de tickets assez ancien (2006) distribué sous licence BSD. Il est écrit en Python et l'installation se fait soit directement depuis les dépôts Debian, soit via **pip** (logiciel permettant d'installer les scripts de PyPi, *PYthon Package Index*) :

```
$ sudo aptitude install python python-setuptools
$ sudo easy_install pip
$ sudo pip install trac
```

Il faut ensuite configurer le projet et répondre à quelques questions (nous utiliserons la base de données SQLite, qui ne nécessite aucune installation supplémentaire) :

```
$ sudo trac-admin /var/www/trac initenv
$ sudo chown -R www-data:www-data /var/www/trac
```

Il faut maintenant créer au moins un utilisateur administrateur. Pour un fonctionnement local, les utilisateurs sont créés à l'aide de la commande **htpasswd** de Apache :

```
$ htpasswd -cb /home/login/.pwdtrac admin mot_de_passe
```

Vous pouvez ensuite ajouter d'autres utilisateurs à l'aide de la même commande (sans l'option **-c**), puis lancer Trac par la commande :

```
$ sudo tracd --port=8000 --basic-auth="nom_du_projet,/home/login/.
pwdtrac,admin" /var/www/trac
```

Ouvrez ensuite un navigateur web et faites-le pointer vers l'adresse http://localhost:8000. Ici, la commande est lancée en tant qu'administrateur pour avoir les droits de lecture et d'écriture sans configuration supplémentaire, puisque le gestionnaire est voué à être installé sur un serveur et non simplement en local (cette partie d'installation étant très semblable à celle d'Apache Bloodhound, vous pourrez vous reporter à la description de cette installation, en fin d'article, pour configurer Trac sur un serveur).

Comme le montre la figure 2, au premier coup d'œil vous verrez que l'écran est chargé... Mais il faut noter que Trac dispose d'un wiki intégré et que tout est paramétrable dans le menu administrateur. De plus, des extensions sont disponibles pour apporter de nouvelles fonctionnalités (par exemple **freedomsponsors**).

Fig. 2 : Page d'accueil de Trac

Trac dispose de toutes les fonctionnalités nécessaires pour un gestionnaire de tickets ; mais si, comme moi, l'aspect graphique vous rebute, jetez donc un œil à Apache Bloodhound en fin d'article. Vous pourrez également installer l'extension **ThemeEnginePlugin** avec un thème tel que **CrystalX** (voir http://trac-hacks.org/wiki/CrystalxTheme).

3 Mantis

Mantis **[3]**, de son nom complet Mantis Bug Tracker ou Mantis BT, est un gestionnaire de tickets écrit en PHP et distribué sous licence GNU GPL v2. Sur les distributions basées sur Debian, ce sera le plus simple des gestionnaires à installer puisqu'il est directement présent dans les dépôts :

```
$ sudo aptitude install mantis
```

Le processus d'installation vous demandera de créer un administrateur en indiquant un identifiant et un mot de passe, puis vous n'aurez plus qu'à ouvrir un navigateur à l'adresse http://localhost/mantis/admin/install.php pour achever la configuration (remplacez bien sûr **localhost** par l'adresse IP de votre serveur le cas échéant).

Après vous avoir demandé de vous authentifier en tant qu'administrateur, vous pourrez configurer l'accès à votre base de données. Par défaut, c'est MySQL qui sera proposé, mais vous pouvez choisir dans le menu déroulant PostgreSQL (étiqueté en « expérimental », donc à utiliser à vos risques et périls).

Pour la première connexion, vous devrez utiliser l'identifiant « administrator » et le mot de passe indiqué dans l'étape

de configuration. Vous pourrez alors créer de nouveaux utilisateurs dans le menu ***Manage***, puis ***Manage Users***. Les fonctionnalités sont ensuite les mêmes qu'avec les autres gestionnaires de tickets : possibilité d'ajout d'extensions (https://github.com/mantisbt-plugins), réglage des champs pour la saisie des tickets, etc. Et comme le montre la figure 3, comme pour les autres, au niveau du design c'est laid !

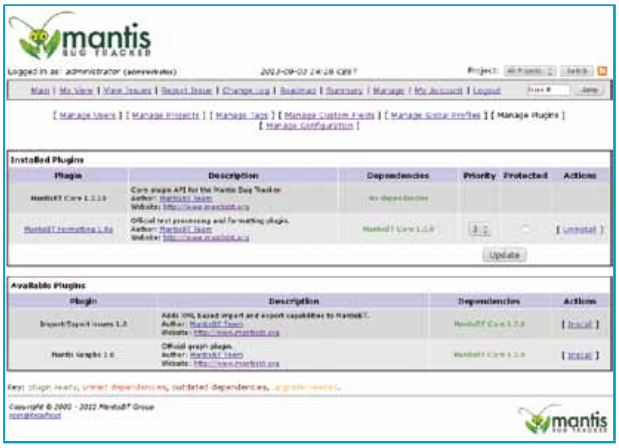


Fig. 3 : Écran de gestion des extensions de Mantis

Avec Mantis, grâce à Tim Pietrusky qui a développé une extension de gestion des thèmes, nous allons pouvoir changer simplement l'apparence des pages. Pour installer cette extension, il faut la télécharger (vous pouvez aussi cloner le projet) et la copier dans le bon répertoire. Il faut faire de même avec au moins un thème :

```
$ wget http://github.com/TimPietrusky/MantisThemeManager/zipball/master
$ unzip master
$ sudo mv TimPietrusky-MantisThemeManager-0b74494 /usr/share/mantis/www/
plugins/MantisThemeManager
$ wget http://github.com/TimPietrusky/mantisbt-is-a-rockstar/zipball/master
$ unzip master
$ sudo mvkdir /usr/share/mantis/www/css/themes
$ sudo mv TimPietrusky-mantisbt-is-a-rockstar-9f034f5/rockstar /usr/share/
mantis/www/css/themes
$ sudo chown -R www-data /usr/share/mantis/www/css/themes
```

Retournez ensuite sur la page des extensions et cliquez sur le lien ***install*** se trouvant sur la ligne de l'extension « MantisBT Theme Manager ». Celle-ci va être déplacée dans la section des extensions installées et en cliquant sur son nom, vous aboutirez à l'écran de sélection des thèmes. Sélectionnez le thème « rockstar », puis cliquez sur le bouton ***Change theme***... Vous voilà revenu en 2013 !

4 Apache Bloodhound, le petit dernier

Apache Bloodhound **[4]** est le dernier des logiciels de gestion de tickets libres puisqu'il est apparu en 2012... Mais sous son apparente nouveauté se cache un gestionnaire plus ancien. Il est basé sur Trac (donc en Python), son installation est très simple et son design est très sobre et épuré. Comme son nom peut le laisser deviner, il est distribué sous licence Apache.

Il va falloir installer les paquetages nécessaires à l'installation (nous utiliserons ici une base de données PostgreSQL) :

```
$ sudo aptitude install python python-virtualenv postgresql python-
psycopg2 python-setuptools
$ sudo easy_install pip
```

Il faut ensuite ajouter une base de données et un utilisateur pour accéder aux données. Par simplicité, la base et l'utilisateur porteront le même nom... **bloodhound**. Voici la suite de commandes que nous avons déjà utilisée pour l'installation de Bugzilla :

```
$ sudo su - postgres
$ createuser -U postgres -S -D -R -E -P bloodhound
Enter password for new role:
Enter it again:
$ createdb -U postgres -O bloodhound -E UTF-8 bloodhound
$ exit
```

Normalement, le système est prêt à accueillir Bloodhound (qui fera lui-même ensuite quelques installations supplémentaires). Il nous faut donc récupérer le code source :

```
$ wget http://wwwftp.ciril.fr/pub/apache/bloodhound/apache-bloodhound-0.7.tar.gz
```

Ici, la commande est donnée pour la dernière version stable disponible au moment de la rédaction de cet article (version 0.7). Un bouton « download now » est disponible sur la page d'accueil du site pour pouvoir sélectionner le numéro de version. On peut ensuite passer à l'installation proprement dite :

```
$ tar -zxvf apache-bloodhound-0.7.tar.gz
$ cd apache-bloodhound-0.7/installer/
$ virtualenv --system-site-packages bloodhound
$ source ./bloodhound/bin/activate
$ pip install -r requirements.txt
```

Pour ceux d'entre vous qui ne seraient pas familiers des environnements virtuels Python, nous avons créé un environnement nommé **bloodhound** dans lequel nous nous sommes placés pour installer les modules manquants. Ce mécanisme permet de ne pas polluer le système avec des versions de modules qui seraient éventuellement incompatibles avec d'autres programmes Python. Pour sortir de cet environnement, il faut utiliser la commande **deactivate**.

Après avoir attendu le téléchargement et l'installation des différents modules, vous pourrez lancer le script de configuration et répondre aux différentes questions. Si vous avez créé la base et l'utilisateur **bloodhound** sous PostgreSQL, vous pourrez utiliser beaucoup de réponses par défaut :

```
$ python bloodhound_setup.py
...
Do you want to install a PostgreSQL database [Y/n]:
DB user name [bloodhound]:
...
```

À la fin de cette étape, vous pouvez tester localement Bloodhound en lançant la commande :

```
$ tracd ./bloodhound/environnements/main
--port=8000
```

En ouvrant un navigateur web à l'adresse http://localhost:8000/main, vous parviendrez sur la page d'accueil de Bloodhound. Pour pouvoir effectuer des réglages, connectez-vous en tant qu'administrateur (lien « login » en haut à droite), puis allez dans le menu ***Admin*** (lien « Admin. » en haut à droite, à ne pas confondre avec votre identifiant « admin »). Attention, car tous les éléments de configuration ne sont pas forcément accessibles simplement. Il faut notamment cliquer sur la liste déroulante de gauche (voir figure 4) pour accéder à la configuration détaillée (type de tickets, degrés d'urgence, etc.).

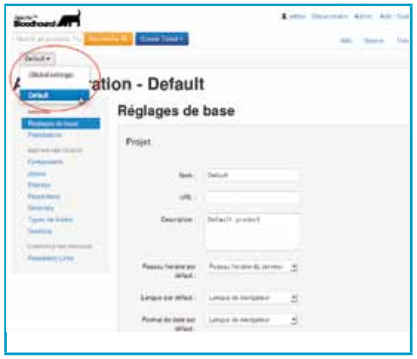


Fig. 4 : Configuration d'Apache Bloodhound

Pour une installation sur un serveur, vous devrez déployer Bloodhound depuis l'environnement virtuel (pensez éventuellement à créer un lien symbolique depuis **/var/www** sur votre répertoire Bloodhound ou déplacez celui-ci) :

```
$ trac-admin ./bloodhound/environments/main
deploy ./bloodhound/site
```

Activez ensuite les modules **proxy_http** (pour la redirection de localhost:8000 vers issues.monprojet.org), **wsgi** et **auth_digest** d'Apache :

```
$ sudo a2enmod proxy_http
$ sudo a2enmod wsgi
$ sudo a2enmod auth_digest
$ sudo service apache2 restart
```

Pour accéder à Bloodhound, il faut créer un VirtualHost dans **/etc/apache2/sites-available/issues.monprojet.org** :

```
01: <VirtualHost issues.monprojet.org:80>
02:   WSGIDaemonProcess bh_tracker user=www-data
python-path=/var/www/bloodhound/lib/python2.7/site-
packages
03:   WSGIPythonHome /var/www/bloodhound/
04:   WSGIScriptAlias /bloodhound /var/www/
bloodhound/site/cgi-bin/trac.wsgi
05:
06:   <Directory /var/www/bloodhound/site/cgi-bin>
07:     WSGIProcessGroup bh_tracker
08:     WSGIApplicationGroup %{GLOBAL}
09:     Order deny,allow
10:     Allow from all
11:   </Directory>
12:
13:   <LocationMatch "/bloodhound/[^/]+/login">
14:     AuthType Digest
15:     AuthName "Bloodhound"
16:     AuthDigestDomain /bloodhound
17:     AuthUserFile /var/www/bloodhound/
environments/main/bloodhound.htdigest
18:     Require valid-user
19:   </LocationMatch>
20:
21:   ProxyRequests Off
22:   <Proxy *>
23:     Order deny,allow
24:     Allow from all
25:   </Proxy>
26:
27:   ProxyPreserveHost On
28:   ProxyPass / http://localhost:8000/main/
29:   ProxyPassReverse / http://localhost:8000/main/
30: </VirtualHost>
```

Enfin, il faut activer ce VirtualHost :

```
$ sudo a2ensite issues.monprojet.org
$ sudo service apache2 reload
```

C'est un gestionnaire qui est très agréable à utiliser et qui remplit parfaitement son rôle. La modification des pages se fait à l'aide du wiki qui est incorporé. Il faut noter que l'extension FreedomSponsors de Trac, bien qu'apparemment compatible avec Bloodhound, n'ajoute aucun lien, ce qui est bien dommage...

Conclusion

Dans cet article, je ne me suis pas attaché à un gestionnaire de tickets en particulier, mais j'ai préféré décrire leur installation pour que vous puissiez vous faire vous-même une idée. Tous ont les mêmes fonctionnalités qu'il faut aller chercher parfois à des endroits forts différents, mais ils gèrent bien les tickets.

Comme vous avez pu le voir, lorsque l'on gère des tickets on n'est pas là pour rigoler ! C'est vrai que saisir un rapport d'erreur dans une interface conviviale et ergonomique aurait pu influencer négativement l'utilisateur. On nous propose donc des interfaces des années 1990 contenant sur chaque page un maximum d'informations (ce qui permet de n'en voir aucune). Deux projets tirent leur épingle du jeu : Mantis (en le modifiant avec l'extension MantisBT Theme Manager) et Apache Bloodhound, qui est une surcouche à Trac. Personnellement, je préfère le second, car je trouve son interface plus claire et, bien que ne présentant pas encore de version stable, le projet semble prometteur.

Un petit mot également sur un autre gestionnaire de tickets, JIRA **[5]**, dont je n'ai pas parlé dans l'article. Ce logiciel est utilisé notamment par la fondation Apache. Il est complet (comme les autres) et bénéficie d'une interface utilisable... Mais il souffre d'un problème majeur : il est open source, mais non libre. C'est pourquoi, par choix philosophique, je préfère me tourner vers Apache Bloodhound.

Sachez enfin que si vous hébergez votre projet sur GitHub, ce dernier met à votre disposition un gestionnaire de tickets. À vous maintenant de trouver le gestionnaire de tickets qui vous convient ! ■

Liens

[1] Site officiel de Bugzilla : http://www.bugzilla.org/

[2] Site officiel de Trac : http://trac.edgewall.org/

[3] Site officiel de Mantis : http://www.mantisbt.org/

[4] Site officiel d'Apache Bloodhound : http://bloodhound.apache.org/

[5] Site officiel de JIRA : https://www.atlassian.com/software/jira

INTÉGRATION CONTINUE AVEC JENKINS

par Tristan Colombo

Lorsque l'on développe, le code source évolue au gré des améliorations et des corrections de bugs. Mais comment s'assurer qu'une modification n'a pas entraîné de régression ? Cette question se pose lorsque l'on développe seul, mais elle est d'autant plus vraie lorsque l'on travaille au sein d'une équipe... L'intégration continue permet d'effectuer des contrôles à chaque modification du code, pour détecter au plus tôt les problèmes. Jenkins est le logiciel qui va nous aider à mettre en place cette méthode.

L'intégration continue, connue sous le nom de *Continuous integration* ou CI chez les anglo-saxons, est issue du génie logiciel. Elle a été créée pour répondre à une problématique simple : détecter et corriger le plus rapidement possible les différents bugs d'un projet et du coup, réduire de manière drastique le stress des équipes de développement lors de la sortie d'une version d'un logiciel ou lors d'une livraison chez le client.

À chaque fois qu'une modification du code, aussi infime soit-elle, est détectée dans le gestionnaire de versions concurrentes, l'application est automatiquement testée et un message avertit le développeur en cas de problème. Ce dernier peut donc immédiatement corriger cette erreur, alors que les modifications qu'il a effectuées sont encore fraîches dans sa tête. Pour que tout cela fonctionne, il faut bien sûr utiliser un gestionnaire de versions concurrentes (Git, SVN, etc.) et que les développeurs l'utilisent correctement en mettant à jour régulièrement leur code.

Quitte à analyser le code pour détecter les erreurs, l'intégration continue permet également d'analyser la qualité du code et la couverture des tests (on parle bien sûr de tests unitaires). Les informations sont présentées sous forme de tableaux de bord, ce qui permet à tous les membres impliqués dans le développement du projet - y compris les non développeurs - d'avoir accès aux données... Et puis, pour les développeurs, il faut quand même avouer qu'il est plus simple de lire rapidement un graphe plutôt qu'une flopée de chiffres bruts...

Cette méthode peut être employée pour poursuivre le parcours de l'application en automatisant le déploiement (chaque fois que tous les tests réussissent, la version de développement passe en production), ou même la mise en production chez le client (là, c'est tout de même un peu plus risqué et il faut vraiment avoir une grosse confiance dans le logiciel de CI... Moi, je ne le ferais pas, mais dans l'absolu, rien ne l'interdit).

Vous l'aurez compris, la mise en place de l'intégration continue nécessite une acceptation complète du processus par tous les membres de l'équipe : en fonction du langage utilisé dans le projet, faire en sorte que le code puisse être compilé automatiquement, écrire les tests unitaires, mettre à jour régulièrement le code après modification et, en cas d'erreur, accepter de la corriger immédiatement ! En effet, si à chaque message indiquant une erreur vous vous dites que vous la corrigerez le lendemain, l'intégration continue ne vous apportera rien si ce n'est la lourdeur d'utilisation d'un logiciel supplémentaire.

Dans cet article, je vais donc vous présenter un logiciel d'intégration continue : Jenkins. Il en existe bien sûr de nombreux autres, tels que CruiseControl ou Apache Continuum, mais Jenkins est vraiment simple à utiliser et il dispose d'un système d'extensions permettant de le configurer finement en fonction des besoins. Cerise sur le gâteau, étant très répandu, sa communauté est importante et vous trouverez très facilement de la documentation sur Internet. Je commencerai donc par présenter ce logiciel avant que nous ne l'installions et que nous ne le configurions.

1 Jenkins

Peut-être connaissez-vous Jenkins, mais sous un autre nom, celui d'Hudson. Toutefois, le logiciel Hudson existe toujours. Comment se fait-il alors que ce soit l'ancien nom de Jenkins ? L'histoire est malheureusement devenue courante dans le monde du logiciel libre et si je

vous parle d'OpenOffice.org et de LibreOffice, vous comprendrez immédiatement de quoi il s'agit. Hudson a été développé chez Sun à partir de 2004 et il est rapidement devenu le logiciel d'intégration continue le plus utilisé. En 2009, Oracle rachète Sun et prend des décisions de développement du logiciel contraires aux idées des équipes de développement. Un *fork* s'ensuit où la majorité des développeurs du noyau et des extensions claquent la porte et renomment leur projet Jenkins. Actuellement, une large majorité des anciens utilisateurs d'Hudson sont passés à Jenkins : encore une belle victoire d'Oracle...

2 Installation

Jenkins est écrit en Java et il faut donc vous assurer que vous disposez bien d'une version de Java installée sur votre machine. En général, **openjdk** est installé par défaut sur les distributions, mais il est préférable d'utiliser la version Oracle (pour une fois...).

```
$ java -version
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
```

Ici, la version 1.7 est installée. Java n'est pas installé chez vous si vous obtenez le message suivant :

```
-bash: java : commande introuvable
```

Les instructions d'installation seront données pour une distribution basée sur Debian et il faudra donc éventuellement les adapter en fonction de votre distribution. Sous Debian, Java (Oracle) n'est plus disponible dans les dépôts et il faut donc ajouter un nouveau dépôt au fichier **/etc/apt/sources.list**. En fonction de vos habitudes, vous pouvez ajouter les lignes à l'aide de commandes **echo "...." | tee -a /etc/apt/sources.list**, ou simplement éditer le fichier et ajouter :

```
## Oracle Java
deb http://ppa.launchpad.net/webupd8team/java/ubuntu precise main
deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu precise main
```

Il faut ensuite ajouter la clé publique pour authentifier le dépôt :

```
# apt-key adv --keyserver keyserver.ubuntu.com --recv-keys EEA14886
Executing: gpg --ignore-time-conflict --no-options --no-default-
keyring --secret-keyring /etc/apt/secring.gpg --trustdb-name /etc/apt/
trustdb.gpg --keyring /etc/apt/trusted.gpg --primary-keyring /etc/apt/
trusted.gpg --keyserver keyserver.ubuntu.com --recv-keys EEA14886
gpg: requête de la clé EEA14886 du serveur hkp keyserver.ubuntu.com
gpg: clé EEA14886: clé publique " Launchpad VLC " importée
gpg:        Quantité totale traitée: 1
gpg:                    importée: 1  (RSA: 1)
```

Mettez en suite à jour votre gestionnaire de paquetages :

```
# aptitude update
```

Il ne vous reste plus qu'à installer Java :

```
# aptitude install oracle-java7-installer
```

Après acceptation des licences, Java 7 sera téléchargé et installé.

Si vous disposez de plusieurs versions de Java, il faut indiquer quelle est la version à utiliser par défaut :

```
# update-java-alternatives -s java-7-oracle
```

À titre d'information, l'option **-l** de la commande **update-java-alternatives** vous indique la liste des versions de Java installées sur votre machine.

Pour configurer automatiquement les variables d'environnement Java, il faut installer un paquetage supplémentaire :

```
# aptitude install oracle-java7-set-default
```

En lançant la commande **env**, vous verrez apparaître de nouvelles variables (la modification de la variable **PATH** est indiquée en jaune) :

```
DERBY_HOME=/usr/lib/jvm/java-7-oracle/db
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/
usr/lib/jvm/java-7-oracle/bin:/usr/lib/jvm/java-7-oracle/db/bin:/
usr/lib/jvm/java-7-oracle/jre/bin
JAVA_HOME=/usr/lib/jvm/java-7-oracle
J2SDKDIR=/usr/lib/jvm/java-7-oracle
J2REDIR=/usr/lib/jvm/java-7-oracle/jre
```

Nous disposons maintenant d'une version de Java Oracle correctement installée. Nous aurons également besoin d'un logiciel de gestion de versions concurrentes, donc autant l'installer dès maintenant. Vous pouvez utiliser votre logiciel habituel, j'utiliserai ici Git qui s'installe très simplement :

```
# aptitude install git
```

Si vous souhaitez vérifier si Git est installé sur votre machine, il suffit de taper :

```
$ git --version
git version 1.7.2.5
```

Tout est prêt pour lancer ou installer Jenkins.

2.1 Lancer Jenkins en tant qu'application web

Une version de Jenkins utilisant la technologie *Java Web Start* est disponible. Elle permet surtout de tester Jenkins, vous vous doutez bien qu'il sera difficile de travailler en équipe en utilisant ce procédé. Théoriquement, vous devriez donc pouvoir jeter un œil à Jenkins simplement en vous rendant sur la page https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins. À un peu plus de la moitié de la page, vous trouverez une section intitulée « Test Drive » et contenant un bouton orange « Launch ». Cliquez sur ce

bouton, le téléchargement de l'application démarrera, on vous demandera de confirmer l'exécution de l'application et, l'application devrait se lancer... En théorie...

En effet, vous pourrez constater que quels que soient vos efforts, ça ne fonctionnera pas. Utilisant Firefox, j'ai pensé que le problème venait du plugin IcedTea. Nous avons installé la version Oracle de Java pour les applications de bureau, donc il faut faire la même chose pour les applications web. Tout est déjà installé, il suffit d'indiquer à Firefox comment trouver le bon programme en créant un lien :

```
$ ln -s /usr/lib/jvm/java-7-oracle/jre/lib/amd64/libnpjp2.so
~/.mozilla/plugins/
```

Ensuite, il faut retirer OpenJDK et tous les paquetages relatifs à IcedTea (pour cela, nous utiliserons une commande un peu longue à écrire, mais ô combien pratique pour ne pas avoir à recopier les noms de tous les paquetages) :

```
# aptitude remove openjdk-6-jre default-jre default-jre-headless
default-jdk openjdk-6-jdk
# aptitude remove `dpkg -l | grep 'ii' | grep 'icedtea' | tr -s '
' | cut -d' ' -f2`
```

Lorsque nous relançons l'exécution de Jenkins, Firefox nous propose bien d'utiliser Oracle Java 7 Web Start... Mais ça ne marche toujours pas !

J'ai testé également sous plusieurs distributions et même sous Windows 8 (oui, j'ai honte, mais pour ma décharge ce n'est pas sur mon ordinateur) : ça ne marche pas mieux !

À ce stade, même s'il aurait peut-être été plus sage de le faire avant, on se dit qu'il va falloir saisir un ticket sur https://issues.jenkins-ci.org pour signaler l'erreur et l'on recherche donc si le bug a été signalé. Le ticket **JENKINS-14924** correspond à notre problème. Il a été ouvert le 24 août 2012, personne n'a été affecté à cette tâche bien qu'étant considérée comme critique. Je pense donc que si vous voulez absolument lancer Jenkins en application web, le plus simple sera de télécharger le code source et de corriger vous-même le bug...

2.2 Tester Jenkins localement

Vous pouvez lancer la dernière version de Jenkins localement en téléchargeant le fichier http://mirrors.jenkins-ci.org/war/latest/jenkins.war et en l'exécutant dans une console :

```
$ wget http://mirrors.jenkins-ci.org/war/latest/jenkins.war
$ java -jar jenkins.war
Running from: /home/login/jenkins.war
...
Jenkins home directory: /home/login/.jenkins found at: $user.
home/.jenkins
...
INFO: HTTP Listener started: port=8080
...
INFO: Jenkins is fully up and running
```

Parmi les lignes s'affichant, certaines contiennent des informations intéressantes :

- *Jenkins home directory* : Jenkins crée un répertoire **.jenkins** dans votre répertoire utilisateur de manière à ce que toutes les modifications effectuées dans Jenkins soient sauvegardées et puissent être réutilisées lors du prochain lancement ;
- *HTTP Listener started* : Jenkins sera accessible via un navigateur web sur le port **8080** ;
- *Jenkins is fully up and running* : tout est prêt, vous pouvez tester Jenkins.

En ouvrant le page http://localhost:8080 dans un navigateur web, vous obtenez enfin la page d'accueil de Jenkins (Fig. 1).

Fig. 1 : Page d'accueil de Jenkins

Comme dit dans le titre de cette partie, il s'agit seulement de tester Jenkins. Je ne m'étendrai donc sur les différentes fonctionnalités qu'après la véritable installation.

2.3 Installer Jenkins sur un serveur

Jenkins ne dispose pas d'un paquetage dans les dépôts Debian standards. Il faut donc ajouter un nouveau dépôt en commençant par la clé publique :

```
# wget -qO - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | apt-key add -
OK
```

Puis, en modifiant le fichier **/etc/apt/sources.list** :

```
## Jenkins
deb http://pkg.jenkins-ci.org/debian binary/
```

Il ne reste plus qu'à rafraîchir la liste des paquetages et à installer Jenkins :

```
# aptitude update
# aptitude install jenkins
```

Normalement, tout devrait fonctionner correctement et en vous rendant sur la page http://localhost:8080 vous devriez accéder à la page d'accueil de Jenkins. Mais imaginons que votre serveur ait un nom de domaine : accéder à http://monserveur.com:8080, ce n'est pas très joli... Nous préférerions http://jenkins.monserveur.com. Il va falloir configurer un petit peu Apache.

Pour commencer, dans votre fichier de configuration de Bind9, en général **/etc/bind/db.monserveur.com**, il faut ajouter le sous-domaine en pensant à incrémenter le numéro de version (ligne 3).

```
01: $ttl 21600
02: monserveur.com. IN  SOA   id32324.serveur.com. webmaster.monserveur.com. (
03:                           2013061107
04:                           21600
05:                           3600
06:                           604800
07:                           86400 )
08: ...
09: jenkins         IN  A     xxx.xxx.xxx.xxx
```

On recharge ensuite Bind9 :

```
# service bind9 reload
Reloading domain name service...: bind9.
```

Nous disposons ainsi de notre sous-domaine et nous pouvons maintenant configurer Apache en utilisant les modules **proxy** et **proxy_http** qui vont servir de proxy inverse pour transformer http://jenkins.monserveur.com en http://monserveur.com:8080. **proxy_http** dépendant de **proxy**, il suffit d'installer ce premier :

```
# a2enmod proxy_http
Considering dependency proxy for proxy_http:
Enabling module proxy.
Enabling module proxy_http.
Run '/etc/init.d/apache2 restart' to activate new configuration!
```

Nous allons maintenant créer le fichier de configuration **/etc/apache2/sites-available/jenkins.monserveur.com** relatif à notre installation de Jenkins :

```
01: <VirtualHost jenkins.monserveur.com:80>
02:     ServerAdmin admin@monserveur.com
03:
04:     ProxyRequests Off
05:     <Proxy *>
06:         Order deny,allow
07:         Allow from all
08:     </Proxy>
09:
10:     ProxyPreserveHost On
11:     ProxyPass / http://localhost:8080/
12:     ProxyPassReverse / http://localhost:8080/
13: </VirtualHost>
```

Faites très attention de bien indiquer le caractère **/** à la fin des lignes 11 et 12 sous peine d'obtenir un affichage sans styles et un message d'erreur dans le fichier de log d'Apache :

```
# tail /var/log/apache2/error.log
[Thu Aug 15 16:50:22 2013] [error] [client xx.xx.xx.xx] proxy: DNS
lookup failure for: localhost:8080static returned by /static/364a9e8f/
images/jenkins.png, referer: http://jenkins.monserveur.com/
```

N'oubliez pas d'activer votre site et rechargez la configuration d'Apache :

```
# a2ensite jenkins.monserveur.com
# service apache2 reload
```

L'installation est terminée. Il va falloir maintenant configurer Jenkins. Faites attention à une chose : si pour une quelconque raison vous interrompez votre travail à ce moment précis, l'installation de Jenkins n'est pas sécurisée et n'importe qui peut y avoir accès. Vous pouvez alors désactiver Jenkins par :

```
# service jenkins stop
```

Pour relancer Jenkins, il faudra bien sûr employer la même commande suivie de **start** :

```
# service jenkins start
```

3 Configurer et utiliser Jenkins

La première des choses à faire va être de sécuriser l'application. Cliquez sur le lien ***Administrer Jenkins*** dans la colonne de gauche. Dès le chargement de la page, vous verrez apparaître un message vous indiquant que votre installation de Jenkins n'est pas sécurisée (Fig. 2). Cliquez sur le bouton ***Configuration de la sécurité*** et cochez la case ***Activer la sécurité*** pour voir apparaître les différentes options. La partie qui nous intéresse est le ***Contrôle de l'accès***. Pour l'instant, n'importe qui peut se connecter à Jenkins et faire absolument ce qu'il veut. Pour chaque option disponible, l'icône point d'interrogation vous affichera un descriptif détaillé du fonctionnement.

Vous pouvez principalement choisir que les utilisateurs se connectent à Jenkins grâce à leur identifiant et leur mot de passe Linux, utiliser un annuaire LDAP, ou utiliser une base de données interne à Jenkins. J'ai choisi cette dernière option pour plus de souplesse : tous les utilisateurs Linux n'ont pas à avoir accès à Jenkins. Une sous-option ***Autoriser les utilisateurs à s'inscrire*** est disponible. Si elle est cochée, les utilisateurs peuvent demander une inscription qui sera transmise à l'administrateur pour validation ; il faut laisser cette option active pour le moment.

Fig. 2 : Première étape de la configuration : la sécurisation de l'installation

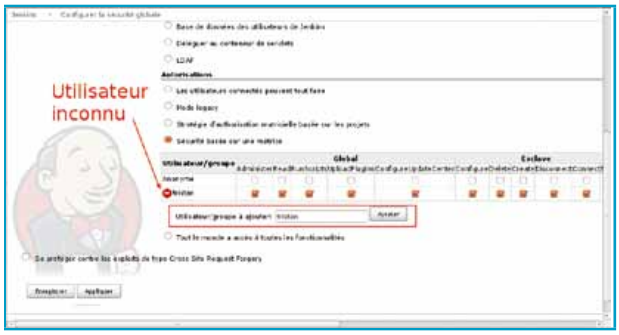


Fig. 3 : Configuration des droits et création à la volée du premier utilisateur

Fig. 4 : Écran d'erreur après oubli d'attribution du droit de lecture global à l'utilisateur « Anonyme »

Il faut maintenant définir les droits de lecture/modification. Là encore, plusieurs options sont disponibles avec par défaut tous les droits associés à tous les utilisateurs. L'option ***Sécurité basée sur une matrice*** permet de régler finement les droits de tous les utilisateurs (l'option ***Stratégie d'autorisation matricielle basée sur les projets*** est un raffinement de cette option permettant de détailler les droits par projet) . On peut ainsi avoir des administrateurs qui auront le contrôle complet du système et pour les autres un simple accès en lecture. J'ai bien entendu choisi cette option, mais qui dit contrôle d'accès dit identifiant et mot de passe.

Nous n'avons pas encore créé d'utilisateur, mais on peut utiliser le champ ***Utilisateur/groupe à ajouter*** pour en définir un. Il n'existera pas encore réellement, comme l'indique le petit sens interdit à la gauche de l'identifiant dans la matrice (Fig. 3), mais il sera créé après validation de nos choix de sécurité. Cet utilisateur sera bien sûr l'administrateur et nous pouvons lui donner tous les droits en cochant toutes les cases. Un bouton se trouve en bout de ligne pour inverser les coches de toutes les cases ; ce mécanisme permet par exemple de sélectionner les droits à retirer, puis en cliquant sur le bouton, d'obtenir les droits complémentaires. Vous devez obligatoirement donner le droit de lecture (***Global***) à l'utilisateur **`Anonyme`** pour le moment (nous le retirerons après). Si vous ne le faites pas, vous aboutirez à l'écran de la figure 4 vous signalant une erreur et vous devrez alors suivre la procédure de réinitialisation de la configuration donnée dans la suite.

Après avoir cliqué sur le bouton ***Enregistrer***, la page de connexion apparaîtra. Comme nous avons autorisé les utilisateurs à s'enregistrer, nous allons créer l'utilisateur que nous avons ajouté dans la matrice de droits (pour moi il s'agit de **`tristan`**). Pour cela, il faut cliquer sur le lien ***Créez un compte*** (Fig. 5) et remplir le formulaire qui apparaît. On vous propose alors de retourner sur la page d'accueil et vous remarquerez dans le coin supérieur droit l'apparition de votre identifiant, suivi du lien ***Se déconnecter***.

Si vous vous déconnectez, vous verrez que n'importe qui a accès à des données en lecture et que n'importe qui peut demander la création d'un compte... Retournez dans la partie ***Administrer Jenkins***, puis ***Configurer la sécurité globale*** et, dans la partie ***Contrôle de l'accès***, désactivez ***Autoriser les utilisateurs à s'inscrire***. De même, dans la matrice des droits, retirez le droit de lecture global de l'utilisateur **`Anonymous`**.

Avant de cliquer sur le bouton ***Enregistrer***, n'oubliez pas de cocher la case ***Se protéger contre les exploits de type Cross Site Request Forgery***. Puisqu'on vous le propose, autant en profiter...

Maintenant, si vous vous déconnectez, vous aurez accès à un écran de connexion ne présentant aucune information, comme le montre la figure 6.

Voilà une bonne chose de faite ! Mais vous vous doutez qu'il reste encore pas mal de travail...

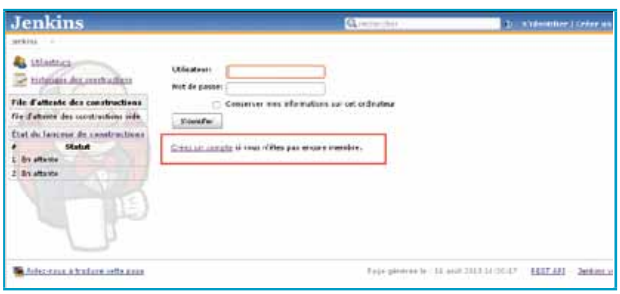


Fig. 5 : Accès à la création du compte du premier utilisateur

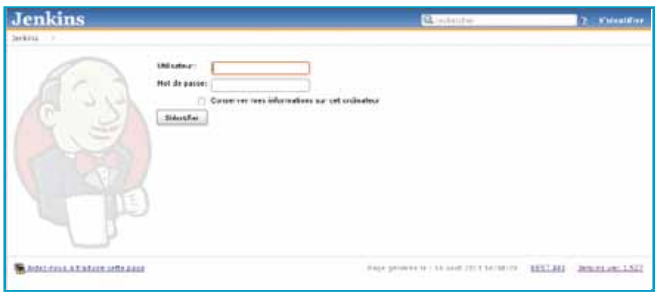


Fig. 6 : Écran de connexion après configuration de la sécurité

Procédure de réinitialisation de la configuration

Si vous vous êtes trompé lors de cette étape et que vous n'avez plus accès à rien, pas de panique ! Voici la procédure permettant de réinitialiser la configuration :

1. Arrêtez Jenkins :

```
# service jenkins stop
```

2. Éditez le fichier **/var/lib/jenkins/config.xml** et remplacez la ligne **<useSecurity>true</useSecurity>** par **<useSecurity>false</useSecurity>** (normalement en ligne 7). Supprimez les lignes des balises **<authorizationStrategy>** (ligne 8) et **<securityRealm>** (lignes 9 à 12). Sauvegardez les modifications.

3. Si vous avez créé des utilisateurs et que vous voulez les supprimer, détruisez leurs répertoires respectifs dans **/var/lib/jenkins/users**. Par exemple, pour supprimer l'utilisateur **linus** :

```
# rm -R /var/lib/jenkins/users/linus
```

4. Relancez Jenkins :

```
# service jenkins start
```

5. Si ça ne marche toujours pas, arrêtez à nouveau Jenkins, supprimez le fichier **/var/lib/jenkins/config.xml** et relancez Jenkins. Cette solution est plus radicale, mais si vous aviez déjà un peu travaillé sur la configuration (plus que simplement la sécurité), vous perdrez tout...

3.1 La gestion de versions concurrentes

Par défaut Jenkins peut s'interfacer avec CVS ou Subversion dans la page ***Configurer le système*** de la page « ***Administrer Jenkins*** ». Pour pouvoir utiliser Git, il va falloir ajouter une extension... Et vous allez voir que cela se fait de manière très simple :

1. Allez dans ***Administrer Jenkins > Gestion des plugins*** ;
2. Cliquez sur l'onglet ***Disponibles*** ;
3. Dans la zone de filtre, tapez le nom de l'extension recherchée sans appuyer sur la touche **<Return>** (la mise à jour de la page se fait automatiquement). Dans le cas de l'extension pour Git que nous voulons installer, vous pourrez bien sûr taper « git » ;
4. Cochez la ou les case(s) de la ou des extension(s) à installer (ici **Git Plugin**), puis cliquez sur le bouton ***Install without restart*** ou ***Download now and install after restart*** (tous les messages ne sont pas traduits...).

La figure 7 montre l'écran de sélection de l'extension et sur la figure 8 on peut voir l'écran résumant l'installation des extensions. Ici, on nous demande de redémarrer Jenkins à cause des mises à jour des extensions « Credentials Plugin » et « SSH Credentials Plugin » (cochez simplement la case de redémarrage).

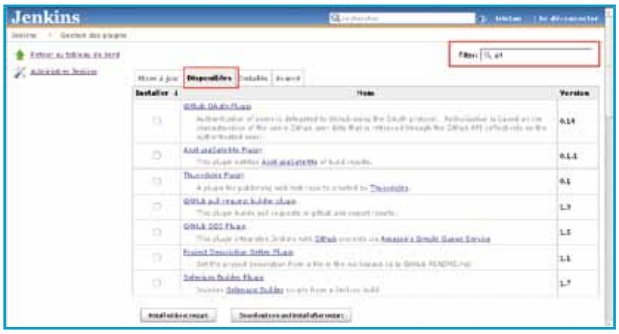


Fig. 7 : Installation d'une nouvelle extension

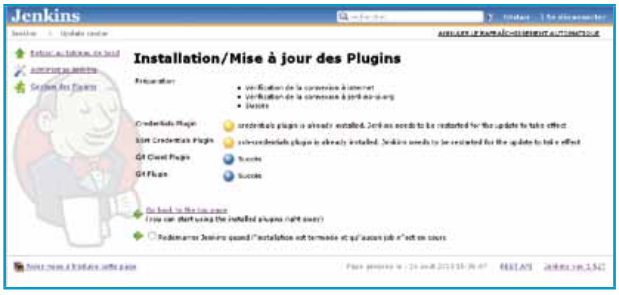


Fig. 8 : Écran de suivi d'installation des extensions

Dans ***Administrer Jenkins > Configurer le système***, une nouvelle partie ***Git*** apparaît prouvant l'installation correcte (Fig. 9). Il faut maintenant pouvoir se connecter automatiquement au serveur Git.

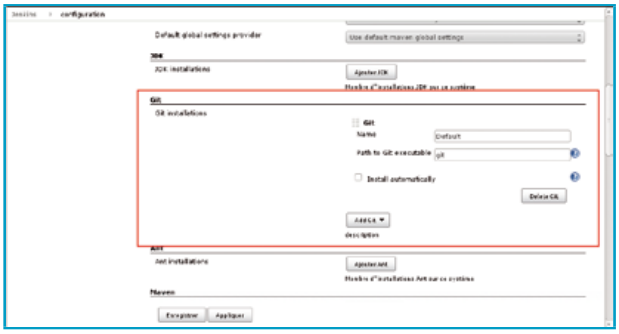


Fig. 9 : Configuration de Git dans « Configurer le système » après ajout de l'extension

3.1.1 Configuration du serveur Git

Nous avions installé le paquetage de Git au début de cet article, mais nous n'avions rien configuré. Il est temps de configurer le serveur ! Tout d'abord, nous activons la coloration des messages :

```
# git config --global color.ui auto
```

Ensuite, nous créons un répertoire qui contiendra les dépôts :

```
# mkdir /var/git
```

Par défaut, il existe un groupe **git**. Nous allons donner les fichiers de **/var/git** à ce groupe :

```
# cd /var
# chgrp -R git git
# chmod -R g+swX git
```

Nous initions ensuite un projet de test :

```
# mkdir /var/git/test
# cd /var/git/test
# git init --shared=group
```

Comme vous devez vous connecter à votre serveur en SSH, vous pouvez tester la connexion à Git depuis un ordinateur externe en clonant notre dépôt de test :

```
$ git clone ssh://tristan@server/var/git/test
Cloning into 'test'...
warning: You appear to have cloned an empty repository.
```

Ici, « server » est le nom ou l'adresse IP de votre serveur. Attention : si vous avez modifié le port de connexion SSH, il faudra le préciser. Par exemple, pour un serveur dont l'IP est 14.22.123.12 et le port SSH 1515, nous devrons utiliser : **ssh://tristan@14.22.123.12:1515/var/git/test**.

Nous savons que notre serveur Git fonctionne correctement, repassons à Jenkins. Lors de l'installation de Jenkins, un utilisateur **jenkins** a été automatiquement créé. C'est lui qui exécute toutes les tâches de Jenkins. Connectez-vous en tant que **jenkins** et créez un répertoire **.ssh** dans votre répertoire personnel (qui sera **/var/lib/jenkins**, vous pouvez lancer un **pwd** pour vous en assurer). Ce répertoire **.ssh** va nous permettre de stocker les clés SSH :

```
# su jenkins
$ pwd
/var/lib/jenkins
$ mkdir .ssh
$ cd .ssh
$ ssh-keygen -t rsa -C "Jenkins_git"
Generating public/private rsa key pair.

Enter file in which to save the key (/var/lib/jenkins/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/jenkins/.ssh/id_rsa.
Your public key has been saved in /var/lib/jenkins/.ssh/id_rsa.pub.
The key fingerprint is:
ab:g4:r4:ec:gf:a4:97:78:ff:45:af:d6:84:h2:22:3c Jenkins_git
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|                 |
|          X      |
|        A| u i . |
|      ::  a + a  |
|     ..y (   ) . |
|     .#.5 _   E  |
|     =S!s?    r  |
+-----------------+
```

Nous générons la clé dans le répertoire où nous nous trouvons (**.ssh**) et il faut bien prendre garde de ne pas donner de mot de passe pour que Jenkins puisse se connecter automatiquement au serveur. La clé est ensuite générée et nous obtenons deux fichiers **id_rsa** et **id_rsa.pub**. Il faut ensuite créer le fichier **known_hosts** pour que la connexion au serveur Git depuis Jenkins se fasse automatiquement à la première connexion :

```
jenkins@server:/var/lib/jenkins/.ssh$ ssh-copy-id -i ~/.ssh/id_rsa.pub
"-p 1515 jenkins@14.22.123.12"
Now try logging into the machine, with "ssh '-p 1515
jenkins@14.22.123.12", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

J'ai utilisé ici la syntaxe permettant de spécifier un port de connexion (notez les guillemets encadrant l'option **-p** et l'identifiant et l'IP du serveur). Une petite remarque qui pourrait faire gagner beaucoup de temps à certains d'entre vous : si vous avez restreint les utilisateurs pouvant se connecter en SSH, pensez à éditer le fichier **/etc/ssh/sshd_config** et à ajouter l'utilisateur **jenkins** :

```
AllowUsers tristan jenkins
```

Il faut ensuite recharger la configuration de **ssh** par :

```
# service ssh reload
```

Enfin, nous déclarons un nouvel utilisateur Git pour Jenkins :

```
# git config --global user.email "tristan.colombo@info2dev.com"
# git config --global user.name "jenkins"
```

Pour tester que tout fonctionne correctement, connectez-vous en tant qu'utilisateur **jenkins** et essayez de cloner le dépôt :

```
$ git clone ssh://14.22.123.12 :1515/var/git/test
Cloning into test...
warning: You appear to have cloned an empty repository.
```

Le message d'avertissement est normal puisque notre projet est vide pour l'instant.

3.2 La qualité du code

La vérification de la qualité du code se fait à l'aide de l'extension **Violations** que vous pourrez installer en suivant la même procédure que pour **Git Plugin**. Cette extension supporte de nombreux langages via notamment **checkstyle** (Java), **jslint** (JavaScript), **pylint** et **pep8** (Python). Les programmes que vous souhaitez utiliser doivent bien sûr être installés sur le serveur... Dans cet article, nous allons tester Jenkins avec Python et nous allons donc installer les outils compatibles avec Python :

```
# aptitude install pylint pyflakes pep8
```

Il nous faut donc un petit code pour que nous puissions obtenir des informations sur la qualité de celui-ci. Nous allons créer le fichier **/var/git/test/monCode.py** :

```
01: def add(a, b):
02:   print("Operation : {} + {}".format(a, b))
03:   return a+b
04:
05: if __name__ == "__main__":
06:   print "Petit programme de test"
```

N'oubliez pas ensuite de transférer votre nouveau fichier sur Git (par abus de langage on « commite ») :

```
$ git add .
$ git commit -m "Fichier exemple"
```

Nous avons installé deux extensions, il est temps de tester leur fonctionnement dans Jenkins avant d'en ajouter d'autres.

3.3 La construction du projet

Le mécanisme central de Jenkins est le « Job ». Nous allons pouvoir définir différentes tâches relatives à un même projet et ces tâches seront automatiquement exécutées. C'est là que nous indiquerons que nous souhaitons lancer les tests unitaires, tester la qualité du code, générer la documentation, etc.

Pour mettre en pratique ces jobs, il va falloir déterminer quel langage nous souhaitons utiliser. À ce niveau, les configurations pour les différents langages seront bien entendu fort différentes : on ne compile pas du C comme on compile du Java et... on ne compile pas du Python !

La première à chose à faire est de créer un nouveau job. Pour cela, il faut cliquer sur le lien ***Nouveau Job*** dans la colonne de gauche, donner un nom à la tâche et indiquer quel est le type de tâche (Fig. 10). ***Construire un projet free-style*** est l'option que vous utiliserez le plus souvent, les autres options permettant respectivement de construire un projet avec Maven (donc pas pour Python), construire un projet avec des spécificités de configuration associées à des environnements multiples, ou contrôler une tâche extérieure à Jenkins.

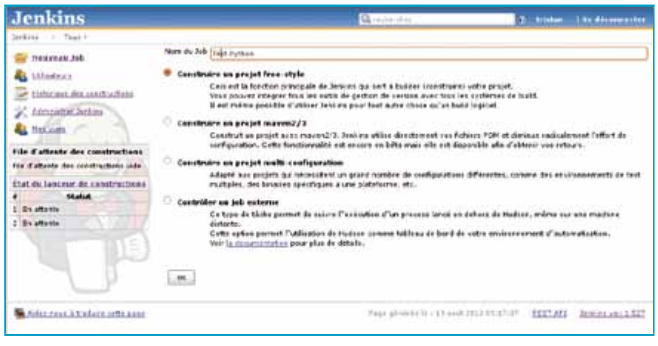


Fig. 10 : Création d'un nouveau job

Dans l'écran de configuration du projet, vous serez invité à ajouter une petite description puis dans la section ***Gestion de code source***, vous devrez sélectionner ***Git*** et indiquer l'URL de votre dépôt. En cas d'erreur dans cette URL ou de mauvaise configuration de Git, vous obtiendrez un message d'erreur dès la sortie du champ, comme le montre la figure 11.

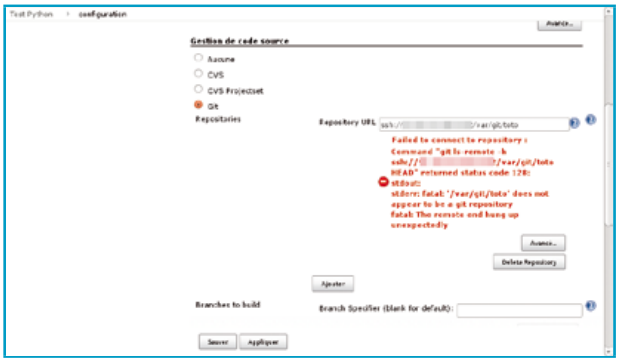


Fig. 11 : URL de dépôt Git erronée dans la configuration d'un projet

Cliquez sur le bouton ***Avancé*** de la configuration de Git pour renseigner les champs utilisateur Git et adresse mail (mêmes données que ce que nous avions indiqué avec **git --global user.name** et **user.email** (**jenkins** et **tristan.colombo@info2dev.com** pour moi). Ces paramètres permettront d'ajouter des tags aux différentes constructions. Si vous souhaitez désactiver ce mécanisme, cochez la case ***Skip internal tag***.

Il faut ensuite indiquer quand lancer les traitements dans la section ***Ce qui déclenche le build***. Vous pouvez choisir de cocher la case ***Construire périodiquement*** et dans le champ qui apparaît, vous pourrez alors indiquer un planning en utilisant la syntaxe cron. Pour une exécution toutes les heures, nous pourrons utiliser indifféremment **0 * * * *** ou le raccourci **@hourly**. Si vous n'êtes pas familier de la syntaxe cron, l'aide associée à ce champ est très détaillée. Le choix de cette option ne permet pas de lancer des constructions à chaque changement dans le code source.

Pour que les traitements du projet se mettent à jour à chaque « commit », cochez la case ***Scrutation de l'outil de gestion de version*** permettant de relancer automatiquement les traitements à chaque modification validée dans le gestionnaire de versions. Ne définissez aucun planning : ce procédé est trop gourmand en ressources ! On peut faire beaucoup mieux avec les actions programmées de Git, les « hooks ». Rendez-vous dans le répertoire de votre dépôt **/var/git/test/.git/hooks** et créez un fichier **post-commit** contenant le code suivant :

```
01: #!/bin/bash
02:
03: curl --user "identifiant_user_jenkins:mot_de_passe" -s http://jenkins.
monserveur.com/git/notifyCommit?url=ssh://14.22.123.12 :1515/var/git/test
```

Il faut ensuite rendre le fichier exécutable :

```
$ chmod +x post-commit
```

Testez le script pour vous assurer que vous n'avez pas commis d'erreurs dans l'URL ou les paramètres de connexion :

```
$ ./post-commit
No git jobs found
```

Nous n'avons effectivement pas encore enregistré notre job, donc le message est correct et nous indique que tout fonctionne correctement. Revenons à la configuration du job. Dans la section ***Build***, cliquez sur ***Ajouter une étape au build***, puis sélectionnez ***Exécuter un script shell***. C'est ici que nous allons pouvoir indiquer les traitements à effectuer. Pour l'instant, nous allons nous contenter de lancer **`pylint`** et de sauvegarder le fichier de résultats. Voici les commandes à indiquer :

```
rm -f pylint.log
for f in `find . -name *.py|egrep -v '^./tests/'`; do
    pylint --output-format=parseable --reports=y $f >> pylint.log
done || :
```

Pylint est exécuté sur l'ensemble des fichiers Python (excepté ceux du répertoire **`tests`**) et le résultat de l'analyse est stocké dans le fichier **`pylint.log`**. Les caractères **`|| :`** en fin de ligne forcent le shell à renvoyer la valeur de succès **`0`** (sinon Jenkins arrête la construction). Dans la section ***Action à la suite du build***, il ne reste plus qu'à ajouter un ***Report Violations*** et à indiquer en vis-à-vis de **`pylint`** le fichier **`pylint.log`** (Fig. 12). Une fois cela fait, il faut enregistrer le job en cliquant sur le bouton ***Sauver***.

Fig. 12 : URL de dépôt Git erronée dans la configuration d'un projet

Vous vous retrouvez alors sur la page du projet « Test Python ». Il faut lancer une première construction en cliquant sur le lien ***Lancer un build*** dans le menu de gauche. Un nouveau choix ***Violations*** apparaît et, à chaque nouvelle construction, la fenêtre ***Historique des builds*** va contenir les différentes constructions ayant réussi ou échoué. Sur la figure 13, on peut voir qu'il y a eu quatre constructions (les deux premières n'ont pas abouti et sont notées en rouge), et que le code contient onze problèmes (l'icône du petit soleil se cachant derrière les nuages donne une indication du nombre de problèmes sur un projet : il vaut mieux avoir un grand soleil qu'un orage !).

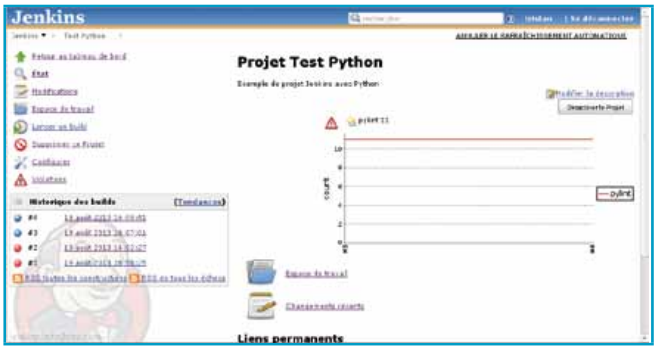


Fig. 13 : Résumé des traitements du projet « Test Python » : Pylint a détecté onze problèmes.

Pour obtenir des informations détaillées, cliquez dans l'historique sur une construction (en général, on choisit la dernière), puis cliquez sur le graphe du nombre de problèmes détectés en fonction du numéro de construction. Un nouveau graphe apparaît vous indiquant le nombre de problèmes classés en fonction de leur niveau de gravité (Fig. 14). En bas de page (encadré en rouge sur la figure 14), le nom du fichier (ou des fichiers si vous en avez plusieurs) apparaît : en cliquant sur celui-ci, vous obtiendrez le détail des problèmes détectés ligne à ligne. En plaçant la souris sur un des panneaux de danger, le détail sera encore plus fin, comme le montre la figure 15.

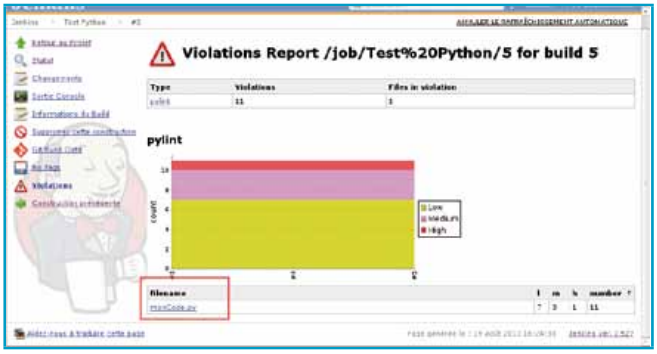


Fig. 14 : Problèmes du projet classés par ordre de gravité

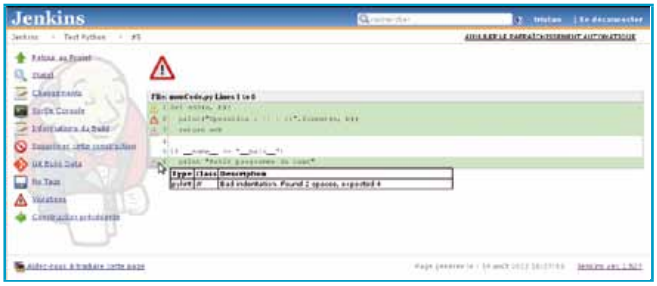


Fig. 15 : Détail des problèmes du projet dans le fichier monCode.py

Nous allons corriger quelques-uns des problèmes de notre code pour nous assurer de la construction automatique du projet. Voici le code corrigé (les modifications sont notées en rouge) :

```
01: # -*- coding:utf-8 -*-
02:
03: def add(val1, val2):
04:     print("Operation : {} + {}".format(val1, val2))
05:     return val1 + val2
06:
07: if __name__ == "__main__":
08:     print "Petit programme de test"
```

Ensuite, il ne faut pas oublier de transmettre les modifications :

```
$ git add .
$ git commit -m "Corrections apres passage Pylint"
...
 1 files changed, 6 insertions(+), 4 deletions(-)
```

Si vous retournez sur Jenkins, vous verrez que la construction a été effectuée automatiquement et que le graphe des problèmes a été mis à jour (Fig. 16).

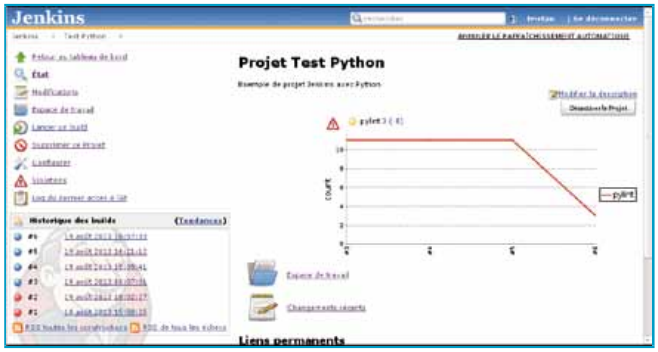


Fig. 16 : Mise à jour automatique des informations de Jenkins après une validation de code dans Git

Nous avons mis en place le cadre de fonctionnement de Jenkins. Il ne reste plus qu'à améliorer les traitements par l'ajout d'extensions et à configurer ces dernières ! Par exemple, pour compléter les vérifications de qualité du code, nous pouvons ajouter le traitement de **pep8** de la même manière que **pylint** en cliquant sur le lien ***Configurer***, puis dans la section ***Build*** en ajoutant en traitement de script shell :

```
rm -f pep8.log
for f in `find . -name *.py|egrep -v '^./tests/'`; do
    pep8 $f >> pep8.log
done || :
```

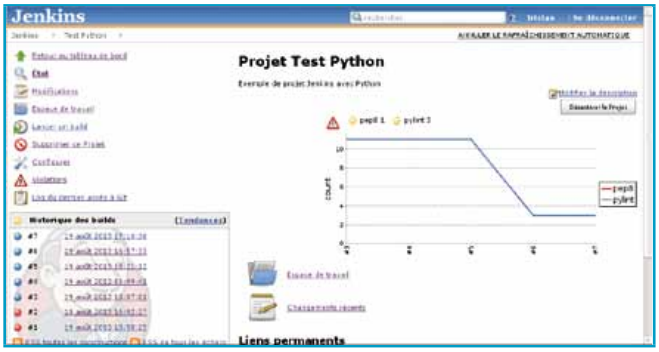


Fig. 17 : Un rapport de qualité du code amélioré à l'aide de pep8

Pour finir, il faut spécifier dans l'onglet ***Report Violations***, en face de **pep8** le fichier **pep8.log**. À la prochaine construction, votre rapport se verra modifié avec l'ajout des informations de **pep8** (Fig. 17).

3.4 Les erreurs

Pour la détection des erreurs, nous allons utiliser l'extension **Warnings** (qui installera également **Static Code Analysis**). Par défaut, Pyflakes n'est pas pris en compte et il faut donc créer une configuration particulière pour **Warnings**. Allez dans ***Administrer Jenkins > Configurer le système*** et dans la section ***Compiler Warnings***, cliquez sur le bouton ***Ajouter*** se trouvant en face de ***Parsers***. Un formulaire apparaît dans lequel nous renseignerons les données suivantes :

- ***Name*** : **pyflakes**
- ***Link name*** : **Pyflakes Warnings**
- ***Trend report name*** : **Pyflakes Warnings**
- ***Regular Expression*** : **^(.*):([0-9]*):(.*)$**
- ***Mapping Script*** :

```
import hudson.plugins.warnings.parser.Warning
import hudson.plugins.analysis.util.model.Priority

String fileName = matcher.group(1)
String category = "PyFlakes Error"
String lineNumber = matcher.group(2)
String message = matcher.group(3)

return new Warning(fileName, Integer.parseInt(lineNumber),
category, "PyFlakes Parser", message, Priority.NORMAL);
```

- ***Example Log Message*** : **monCode.py:3: 'random' imported but unused**

Sur la même page, vous pouvez également configurer les seuils indiquant que la construction est considérée comme instable, ou qu'elle est ratée (Fig. 18).

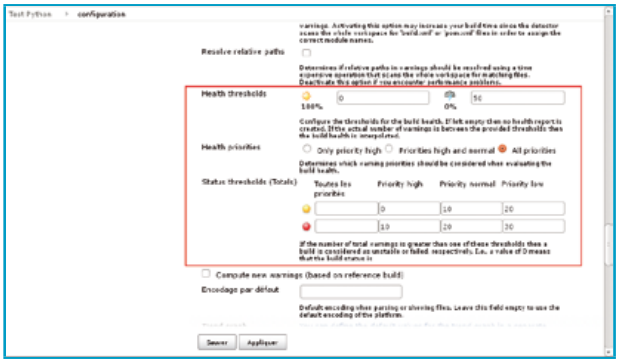


Fig. 18 : Réglage des seuils d'avertissement de l'extension Warnings

Il faut ensuite retourner sur la page de configuration du projet et ajouter le traitement shell suivant :

```
find . -name *.py | egrep -v '^./tests/' | xargs pyflakes  >
pyflakes.log || :
```

Cliquez ensuite sur le bouton ***Add post-build action*** pour ajouter ***Scan for compiler warnings***. Dans la partie ***Scan workspace files***, indiquez le nom de fichier `pyflakes.log` pour ***File pattern*** et, dans la liste déroulante de ***Parser***, sélectionnez l'entrée ***Pyflakes*** que nous avons créée. À la prochaine construction, un nouveau lien ***Pyflakes Warnings*** apparaîtra dans le menu et la page de rapport contiendra un nouveau graphe comme le montre la figure 19 (j'ai ajouté une ligne d'import inutile en ligne 3 du fichier `monCode.py`). En cliquant sur le lien ***Pyflakes Warnings***, vous obtiendrez des détails sur les différents avertissements signalés par l'extension (nom du fichier, numéro de ligne et message d'avertissement).

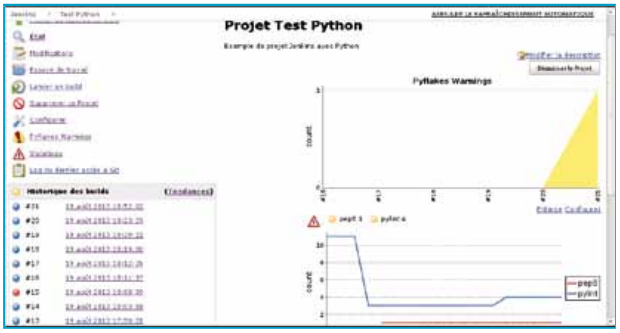


Fig. 19 : Ajout du graphe des rapports d'erreurs

3.5 Les notifications

La notification par courrier électronique est le moyen le plus simple de recevoir des informations depuis le serveur. Pour configurer l'envoi de mails, cliquez sur le lien ***Administrer Jenkins > Configurer le système*** et dans la section ***Notification par email***, complétez le champ ***Serveur SMTP***. Si le serveur Jenkins est aussi votre serveur SMTP, il suffira d'indiquer `localhost`. Vous avez ensuite la possibilité de tester la bonne configuration de votre serveur en cochant la case ***Tester la configuration en envoyant un e-mail de test*** et en renseignant une adresse électronique. Après avoir cliqué sur le bouton ***Tester la configuration***, vous devriez voir le message « Email was successfully sent » et dans votre boîte mail vous aurez un mail ayant pour objet « Test email #1 » et contenant le message « This is test email #1 sent from Jenkins ». N'oubliez pas de sauvegarder vos modifications avant de sortir de la page !

Dans la configuration du projet, vous pouvez maintenant ajouter une action post-construction de type « Notifier par email » (en bas de page). Indiquez alors la liste des destinataires (en les séparant par un caractère espace) qui seront prévenus lors d'un échec de la construction.

Ce mécanisme de notification par mail est vraiment très basique. Il existe une extension permettant d'améliorer son comportement et d'autres méthodes pour recevoir des alertes.

3.5.1 Notification par mail améliorée

L'extension **Email-ext** permet de gérer de manière beaucoup plus fine l'envoi des courriers électroniques. Une fois installée, une nouvelle section intitulée ***Extended E-mail Notification*** apparaîtra sur la page de configuration du système. Vous pourrez alors décider d'envoyer des mails en HTML, de définir une liste d'adresses qui recevront les réponses éventuelles aux mails, de déterminer des adresses qui ne devront pas recevoir les mails, de définir précisément le sujet et le corps du message en utilisant des variables telles que `$PROJECT_NAME` pour le nom du projet, ou encore `$BUILD_NUMBER` pour le numéro de la construction, etc.

Une fois que vous aurez configuré vos différentes options, rendez-vous dans le menu de configuration de votre projet, désactivez éventuellement la notification « standard » par mail et ajoutez le traitement post-construction ***Editable Email Notification***. Vous pourrez configurer là encore les destinataires des mails, le sujet (s'il est différent de celui défini de manière globale), etc., mais notez surtout la possibilité d'ajouter des traitements (*triggers*) en fonction du statut de la construction (avant la construction, construction réussie, construction ayant échoué, etc.). Pour chacun des cas, vous pourrez définir la liste des destinataires, le sujet et le message... Difficile de faire plus configurable !

3.5.2 Messagerie instantanée

L'extension **Instant Messaging** permet d'utiliser la messagerie instantanée pour vos projets (pour utiliser Jabber, il faudra installer également l'extension **Jabber** et pour IRC ce sera bien sûr l'extension **IRC**). Ce mécanisme est plus rapide que la notification par courrier électronique... en fonction de la façon dont vous gérez vos mails et les messages instantanés.

3.5.3 SMS

Vous pouvez être averti des activités de Jenkins par SMS. Pour cela, la solution la plus simple consiste à utiliser le service (payant) fourni par le site http://www.twilio.com (l'équivalent existe avec http://www.hoiio.com). Créez un compte en quelques secondes et vous obtiendrez un numéro de téléphone « twilio ». Conservez la page de votre compte ouverte, nous aurons besoin d'une information qu'elle contient plus tard.

Installez ensuite l'extension **Twilio Notifier** et rendez-vous sur la page de configuration générale dans la partie ***Twilio-Notifier***. Recopiez les informations ***Account SID*** et ***Auth Token*** qui se trouvent sur votre page personnelle du site twilio.com (cliquez sur le cadenas pour rendre lisible le jeton d'authentification). Le numéro de téléphone à renseigner est celui qui vous a été fourni lors de votre inscription. Pour le retrouver, vous pouvez cliquer par exemple sur ***Send a text message*** dans le cadre ***API Explorer***.

Dans la configuration du projet, ajoutez un traitement post-construction de type ***TwilioNotifier*** et indiquez le ou les numéro(s) de téléphone du ou des destinataire(s) et le message (les variables **%PROJECT%** et **%STATUS%** désignent respectivement le nom du projet et son état ; la variable **%CULPRITS%** contient la liste des développeurs qui ont provoqué cette construction). Vous pouvez ensuite choisir d'envoyer un SMS ou un appel où le texte du message sera lu (seule cette option est disponible gratuitement : ça peut être amusant au début pour tester, mais je vous le déconseille en production !).

3.5.4 Smartphones

L'application **Hudson Helper**, compatible avec Jenkins et disponible sous Android et iPhone, permet de se connecter à un serveur Jenkins et de consulter simplement ses rapports, ou même déclencher des constructions (Fig. 20).

Fig. 20 : Consultation des rapports de construction du projet « Test Python » depuis un smartphone Android

3.6 Les tests automatiques

Nous utiliserons l'extension **Cobertura** pour afficher les statistiques des tests unitaires et **Nose** pour lancer les tests. Nose est un module de tests qui propose sa propre syntaxe, mais qui peut s'interfacer avec **doctest** ou **unittest**. L'installation se fait à l'aide d'**aptitude** :

```
# aptitude install python-nose
# pip install -U coverage
```

Pour pouvoir appliquer cette extension, nous allons ajouter à notre code un répertoire **tests** qui contiendra un fichier **testMonCode.py** :

```
01: # -*- coding:utf-8 -*-
02:
03: import unittest
04: import monCode
05:
06: class TestMonCode(unittest.TestCase):
07:
08:     def test_add(self):
09:         self.assertEqual(monCode.add(2, 2), 72)
10:
11: if __name__ == "main":
12:     unittest.main()
```

Nose a été construit de manière à parcourir automatiquement les fichiers à la recherche des tests. Donc, pour lancer ce test, il vous suffit de vous rendre dans le répertoire du projet et de lancer la commande suivante (avant d'exécuter le test, j'ai corrigé mon code Python qui était écrit à moitié en Python 2 et en Python 3) :

```
$ nosetests
F
======================================================================
FAIL: test_add (testMonCode.TestMonCode)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/var/git/test/tests/testMonCode.py", line 9, in test_add
    self.assertEqual(monCode.add(2, 2), 72)
AssertionError: 4 != 72
-------------------- >> begin captured stdout << ---------------------
Operation : 2 + 2

-------------------- >> end captured stdout << -----------------------

----------------------------------------------------------------------
Ran 1 test in 0.004s

FAILED (failures=1)
```

Ajoutons maintenant un traitement shell à notre projet pour générer le fichier de résultat des tests :

```
nosetests --with-xunit --all-modules --traverse-namespace --with-coverage --cover-inclusive
-xunit-file=nosetests.xml || :
coverage xml
```

La commande **coverage xml** permet de convertir le fichier de résultat **.coverage** généré par **nosetests** en fichier XML lisible par l'extension Cobertura.

Il faut ensuite ajouter un rapport Cobertura en cliquant sur ***Publish Cobertura Coverage Report*** en tant qu'action post-construction et spécifier dans le champ ***Cobertura XML report pattern*** le nom de fichier **coverage.xml**. Ajoutez également une action ***Publier le rapport des résultats des tests JUnit*** que vous configurerez avec le fichier **nosetests.xml**. Au lancement de la prochaine construction, vous obtiendrez un nouveau graphe résumant les tests effectués et la couverture du

code comme le montre la figure 21. En cliquant sur l'un ou l'autre de ces graphes, vous obtiendrez des informations plus détaillées.

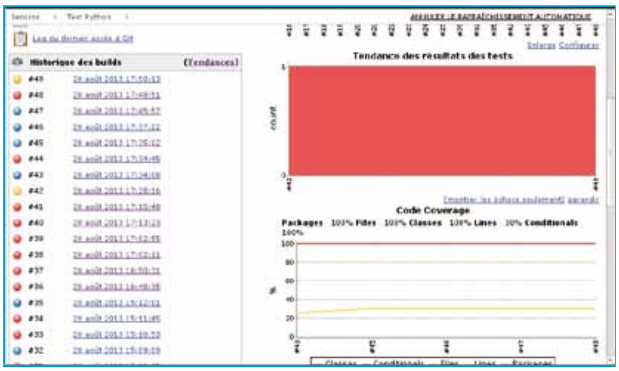


Fig. 21 : Tests unitaires et couverture de code avec l'extension Cobertura

3.7 Génération de documentation technique

La documentation technique est très importante dans un projet... Mais encore faut-il qu'elle soit mise à jour de manière suffisamment régulière ! Alors pourquoi ne pas intégrer la génération de la documentation à la construction du projet ?

Ajoutez votre documentation Sphinx comme vous en avez l'habitude à l'aide de **`sphinx-quickstart`**, puis générez une première fois la documentation pour éviter les problèmes de droits à la création des différents répertoires et fichiers. Ensuite, dans la configuration du projet, dans la section ***Build***, ajoutez un traitement de script shell qui contiendra :

```
cd /var/git/test
make html
```

La première ligne permet de se déplacer dans le répertoire du projet et la ligne suivante génère la documentation. Si vous n'avez pas sélectionné la création du **`Makefile`**, votre commande de génération ressemblera à :

```
sphinx-build -b html ./source ./build
```

Si la construction échoue, vous aurez des informations sur le déroulement des commandes shell en cliquant sur le lien de la construction, puis ***Sortie Console***. À utiliser impérativement pour savoir réellement ce qui s'est passé !

Une fois ces modifications effectuées, vous disposerez d'une documentation qui se mettra à jour automatiquement à chaque commit.

3.8 Utiliser les environnements virtuels

Comme pour la documentation, l'utilisation des environnements virtuels est en fait totalement transparente en ajoutant quelques traitements shell. Je supposerai ici que vous travaillez avec **`virtualenvwrapper`** dans le répertoire **`/var/virtualenvs`** (il doit y avoir un **`export WORKON_HOME=/var/virtualenvs`** dans le fichier **`~/.bashrc`** de l'utilisateur qui gère les environnements virtuels) et que le nom de notre environnement est **`test_python`**.

Pour chaque opération nécessitant l'utilisation de l'environnement virtuel, il vous suffira de l'activer dans un traitement de construction sous la forme d'une commande shell :

```
workon test_python
```

Bien sûr, cette ligne devra être suivie de ce que vous souhaitez effectuer comme action. Par exemple, vous pouvez choisir de mettre à jour les modules d'après le fichier **`requirements.txt`** :

```
pip install -r requirements.txt
```

Vous pouvez également choisir d'exécuter des tests :

```
python test.py
```

Pour la mise à jour du contenu du fichier **`requirements.txt`**, il est plus judicieux de créer un hook **`post-commit`** de Git avec la commande :

```
pip freeze > requirements.txt
```

3.9 Des extensions, des extensions, des extensions !

Vous l'avez vu, par l'ajout d'extensions on peut complètement paramétrer le comportement de Jenkins. Je citerai ici pêle-mêle des extensions moins importantes que les précédentes, qui peuvent toutefois vous être utiles.

3.9.1 Compter le nombre de lignes de code

L'extension **SLOCCount** vous permettra de connaître le nombre de lignes de chacun des langages utilisés dans votre projet. Il faut installer le programme **`sloccount`** :

```
# aptitude install sloccount
```

Dans la configuration du projet, il faut ensuite ajouter l'exécution du script shell suivant :

```
sloccount --duplicates --wide --details . | fgrep -v .git | fgrep -v
coverage.xml | fgrep -v nosetests.xml | fgrep -v clonedigger.xml >
soccount.sc
```

Enfin, ajoutez l'affichage des statistiques SLOCCount en faisant référence au fichier **`sloccount.sc`**.

3.9.2 Rechercher le code dupliqué

Pas besoin d'extension supplémentaire pour la recherche de code dupliqué : l'affichage des résultats est pris en compte

par l'extension **Violations**. Par contre, il va falloir installer **Clone Digger** :

```
# pip install -U clonedigger
```

Ajoutez ensuite dans votre projet un traitement shell :

```
clonedigger --cpd-output . -o clonedigger.xml
```

Pour finir, dans la section de configuration de l'extension **Violations**, ajoutez le nom du fichier `clonedigger.xml` dans le champ ***cpd***.

3.9.3 Afficher les tâches à finir

Vous ajoutez des tags `TODO`, `FIXME`, etc., dans vos commentaires ? Alors utilisez l'extension **Task Scanner** qui vous affichera un rapport des tâches qui doivent être terminées (vous pouvez régler le niveau de priorité dans les paramètres de l'extension). Pour ajouter ce rapport à un projet (lien ***Tâches ouvertes*** dans le menu), il faudra cliquer sur ***Recherche des tâches ouvertes dans le workspace***.

3.9.4 Et tout le reste...

En recherchant dans la liste des extensions disponibles, vous verrez qu'il y a vraiment beaucoup de choix ! Vous verrez qu'il y a même la possibilité de modifier l'apparence de Jenkins...

4 Un petit peu de style : personnaliser l'affichage

La modification de l'affichage se gère encore à l'aide d'extensions. L'extension **Green Balls**, toute simple, permet par exemple de remplacer les boutons bleus de réussite de construction en boutons verts. Ça a l'air tout bête, mais ça améliore nettement la lisibilité des informations !

Pour modifier le style un peu plus en profondeur, il faudra installer l'extension **Simple Theme**. Vous pouvez ensuite trouver des thèmes CSS prêts à l'emploi, tels que `isotope-style` **[2]** qu'il suffira d'adapter. Copiez ce fichier dans le répertoire `userContent/style` de l'utilisateur `jenkins` par exemple et appelez-le `mon-style.css`. Il est très important d'utiliser le répertoire `userContent`, car celui-ci est ensuite servi dans http://jenkins.monserveur.com/userContent. Dans la page de configuration du système Jenkins, dans la section ***Theme***, vous n'aurez plus qu'à ajouter l'URL de votre fichier de styles : http://jenkins.monserveur.com/userContent/style/mon-style.css. Si vous le souhaitez, vous pouvez également ajouter un fichier JavaScript personnalisé.

Vous n'avez plus qu'à modifier le fichier CSS pour obtenir le rendu attendu (Fig. 22).

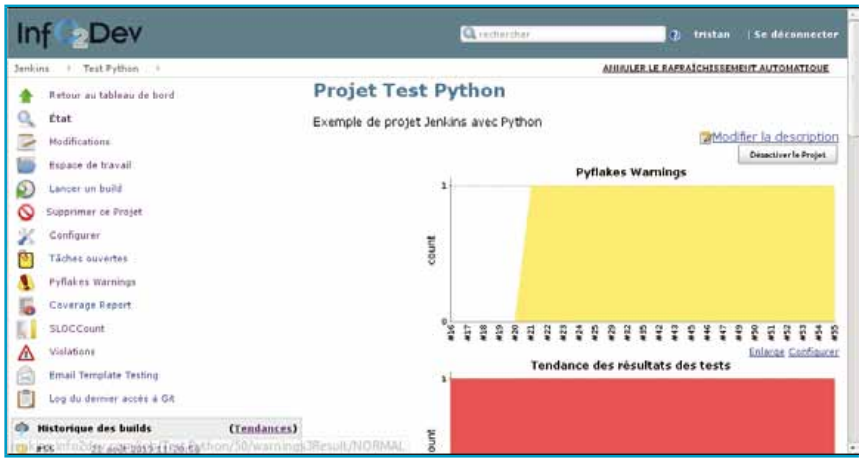


Fig. 22 : Modification de l'apparence de Jenkins

Conclusion

Je n'ai pas détaillé absolument toues les fonctionnalités de Jenkins certaines, comme la gestion des vues, étant très simples à mettre en place et d'autres bien trop complexes. Avec le contenu de cet article, vous devriez être capable de mettre en place un serveur Jenkins et de l'exploiter pleinement dans le cadre d'un projet de développement.

La mise en place est longue, très longue, surtout quand la documentation des extensions est inexistante ou éparpillée de par le Web. Par exemple, il faut savoir que les pages officielles des extensions n'apportent pratiquement pas d'informations... Heureusement que la documentation de Jenkins est, elle, plutôt complète. Je pense avoir suffisamment détaillé les différentes étapes de la configuration pour que celle-ci se déroule sans accroc chez vous. Si ce n'était pas le cas, n'oubliez pas de consulter la console de chaque job ou le fichier de log `/var/log/jenkins/jenkins.log`.

Essayez Jenkins et vous verrez que la qualité de vos développements en sera grandement améliorée... Tout en ayant beaucoup moins de travail ! ■

Liens

[1] Ticket d'erreur de l'application web : https://issues.jenkins-ci.org/browse/JENKINS-14924

[2] Thème « isotope-style » : https://github.com/isotope11/jenkins-isotope-style/blob/master/jenkins-isotope-style.css

HÉBERGEZ VOS PROJETS DANS VOTRE PROPRE SOURCEFORGE

par Benoît Benedetti

OpenOffice, FileZilla, VideoLAN, Nagios, Wine, etc. En plus d'être libres, ces projets phares du mouvement open source présentent la particularité d'être tous hébergés sur la plate-forme SourceForge. Car même si GitHub, propriétaire, a fait l'effet d'un raz-de-marée avec son aspect social, SourceForge reste la forge logicielle historique de référence depuis 1999. Nous allons profiter de cet article pour présenter Allura, la nouvelle version de la plate-forme au cœur de SourceForge.

1 Présentation

Allura est le moteur de la plate-forme SourceForge depuis mi-2011. Il offre toutes les fonctionnalités attendues par une forge logicielle : dépôt de code, gestion des utilisateurs, wiki, gestion des bugs, des tickets, etc.

L'équipe de SourceForge a développé sa propre plate-forme, car elle n'arrivait pas à trouver un projet existant qui répondait complètement à ses attentes. Tout en s'inspirant de projets existants, Allura est créée suivant une architecture d'extensions : chaque projet possède un ensemble de fonctionnalités de base, mais si votre projet nécessite une gestion des droits différente, un flux de travail différent, ou encore son propre type de documents, vous pourrez le mettre en place dans votre projet à l'aide d'une extension.

Allura est écrit en Python, avec le framework Pyramid (anciennement Pylons) et de nombreux autres projets Python. Allura utilise également RabbitMQ pour l'architecture de messages, MongoDB pour le stockage des données, Solr pour l'index et la recherche dans les projets et différentes solutions de gestion de code source comme Git et Subversion.

Allura est développée par la communauté du Libre, pour la communauté. L'équipe de développement veut vraiment qu'Allura appartienne à la communauté et, à ce titre, a fait don d'Allura à la fondation Apache. Allura est depuis 2012 dans l'incubateur d'Apache, en attendant de faire complètement partie de la fondation.

2 Installation

Allura est disponible en version 0.1 ; elle a de nombreuses dépendances, qui nécessitent de nombreuses manipulations d'installation **[1]**. Par chance, l'équipe de développement met à disposition une machine virtuelle Ubuntu sous VirtualBox, au format Vagrant. Il nous faut donc installer VirtualBox et Vagrant.

On commence par installer VirtualBox. Sur ma machine physique hôte qui est une Debian, ça nous donne :

```
$ sudo aptitude -y install virtualbox
```

La version de la machine virtuelle d'Allura utilisée dans cet article a été générée avec Vagrant 1.1.5. La version disponible dans les dépôts Debian est plus ancienne, on récupère donc une version plus récente directement depuis le site officiel (la 1.3.3 lors de la rédaction de l'article) **[2]** :

```
$ wget http://files.vagrantup.com/packages/db8e7a9c79b23264da129f55cf8569167fc22415/
vagrant_1.3.3_x86_64.deb
$ sudo gdebi vagrant_1.3.3_x86_64.deb
```

On crée ensuite un répertoire de travail, avec pour nom la version de la machine virtuelle Ubuntu :

```
$ mkdir allura-ubuntu-1204-server-amd64-20130807
$ cd allura-ubuntu-1204-server-amd64-20130807
```

Dans lequel on récupère les sources de l'application, qui seront utilisées depuis la machine virtuelle :

```
$ sudo aptitude -y install git
$ git clone https://git-wip-us.apache.org/repos/asf/incubator-allura.git allura
```

On peut maintenant récupérer la machine virtuelle au format Vagrant :

```
$ vagrant box add allura-ubuntu-1204-server-amd64-20130807 http://sourceforge.net/projects/
allura/files/vagrant/allura-ubuntu-1204-server-amd64-20130807.box
```

La commande suivante va créer une configuration de base Vagrant dans le répertoire courant :

```
$ vagrant init allura-ubuntu-1204-server-amd64-20130807
```

On peut ensuite démarrer la machine virtuelle :

```
$ vagrant up
```

Et s'y connecter à la machine virtuelle en SSH :

```
$ vagrant ssh
(env-allura)vagrant@vagrant-ubuntu-precise-64:~/src/allura$
```

Vous êtes désormais sur la machine virtuelle. On commence par vérifier que la plate-forme est à jour :

```
$ (env-allura)vagrant@vagrant-ubuntu-precise-64:~/src/allura$ ./update.sh
```

Puis, on peut démarrer les différents services :

```
$ (env-allura)vagrant@vagrant-ubuntu-precise-64:~/src/allura$ ~/start_allura
```

Allura devrait démarrer et écouter sur le port 8080 de la machine virtuelle. Par le jeu des redirections de port automatiquement mises en place par Vagrant, vous pouvez accéder à la page d'accueil d'Allura depuis la machine hôte physique, en entrant http://localhost:8080 dans votre navigateur (Fig. 1).

Plusieurs utilisateurs sont disponibles par défaut, dont l'utilisateur administrateur **admin1**. Connectez-vous avec celui-ci et le mot de passe **foo**.

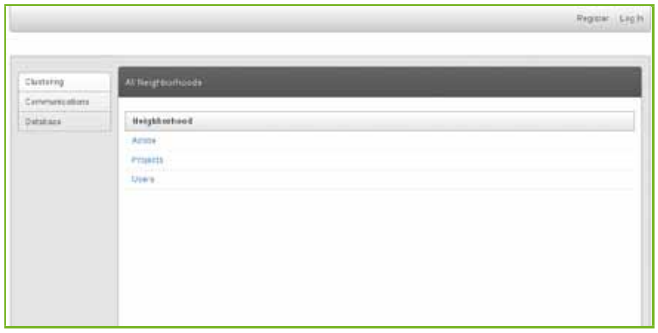


Figure 1

3 Allura

Sur la page d'accueil, le menu latéral gauche permet de naviguer dans les différents projets suivant leur catégorie (***Clustering***, ***Communications*** et ***Database***). Aucun projet n'a de catégorie par défaut, donc ce menu n'affiche rien. En haut à droite, vous pourrez gérer votre compte utilisateur via le lien ***Account*** : informations, mot de passe, clé SSH ou encore abonnements à différents projets (Fig. 2).

De retour sur la page d'accueil, au centre, sont listés des ***Neighborhoods*** (voisinages) : des projets regroupés dans un ensemble commun. Dans la version actuelle d'Allura, il me semble qu'il n'est pas possible de créer un voisinage par l'interface web, il faut directement taper dans la base MongoDB.

Dans notre Allura, par défaut, trois voisinages sont disponibles : ***Adobe*** est un voisinage exemple ; pour chaque utilisateur créé, un projet personnel du même nom de l'auteur est automatiquement créé. ***Users*** est un voisinage qui regroupe tous ces projets personnels ; le dernier voisinage, ***Projects***, est le voisinage dans lequel tous les projets sont

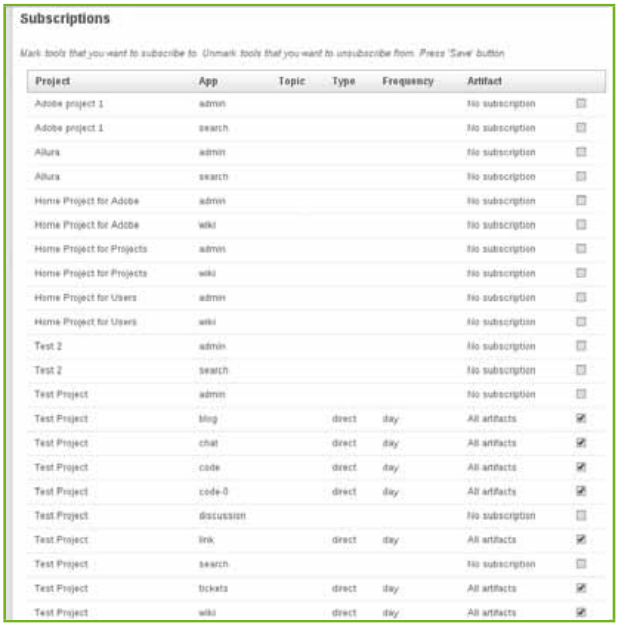


Figure 2

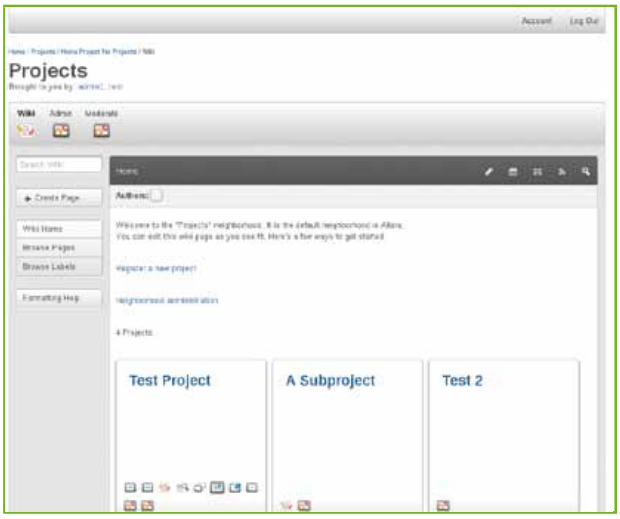


Figure 3

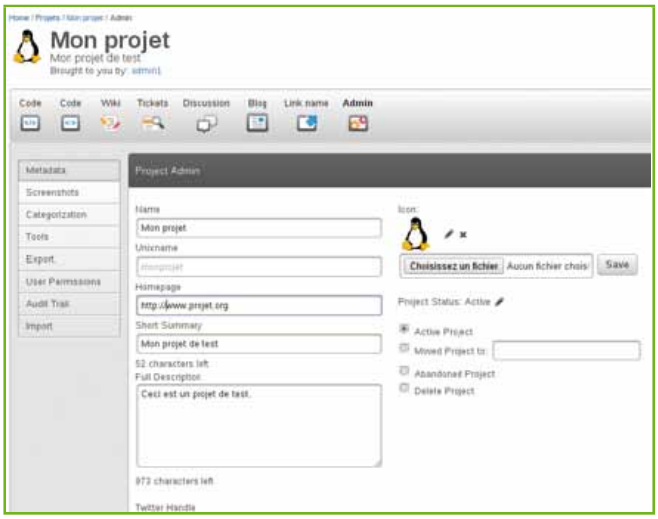


Figure 4

censés être créés par défaut. C'est dans ce voisinage que nous allons continuer à découvrir Allura.

En cliquant sur le voisinage ***Projects***, vous êtes automatiquement redirigé vers son wiki (Fig. 3). On remarque que les URL sous Allura peuvent être de la forme http://localhost:8080/IDENTIFIANT_VOISINAGE/OUTIL, où ici **`IDENTIFIANT_VOISINAGE`** est simplement **`p`** (diminutif de ***Projects***) et **`OUTIL`** vaut **`wiki`**. Donc, l'URL du wiki de notre projet est http://localhost:8080/p/wiki.

Que vous soyez dans une section voisinage, projet, ou sous-projet, l'affichage est similaire : en haut, un menu horizontal avec tous les outils activés de la section en cours (ici ***Wiki***, ***Admin*** et ***Moderate*** activés par défaut pour notre section voisinage ***Projects***, ***Wiki*** étant en gras car c'est l'outil en cours) ; au centre la page de l'outil en cours (ici la page d'accueil par défaut de notre wiki qui liste, entre autres, les projets de notre voisinage) ; enfin, le menu latéral gauche vertical, qui offre des actions pour la section en cours (ici, créer, éditer, lister des pages de notre wiki).

Si vous cliquez sur ***Admin***, dans le menu supérieur, vous pourrez gérer votre voisinage : ajouter ou modifier des outils, gérer les permissions. Plutôt que de faire l'inventaire des différentes fonctionnalités d'Allura depuis la page d'administration du voisinage, nous allons découvrir celles-ci en créant et configurant un projet fictif, ce qui sera plus pratique et didactique.

3.1 Créer un projet

Toujours dans la page d'administration du voisinage, cliquez sur ***Add Project***. Vous arrivez sur la page de création d'un projet, sur laquelle vous devez lui donner un nom, un identifiant, et indiquer quels outils activer (Fig. 4).

La création validée, notre projet est accessible à l'adresse http://localhost:8080/p/IDENTIFIANT (http://localhost:8080/p/monprojet pour notre exemple) et vous êtes automatiquement redirigé vers la section d'administration de votre projet, page ***Metadata***, qui vous permet de préciser les informations de votre projet (Fig. 5).

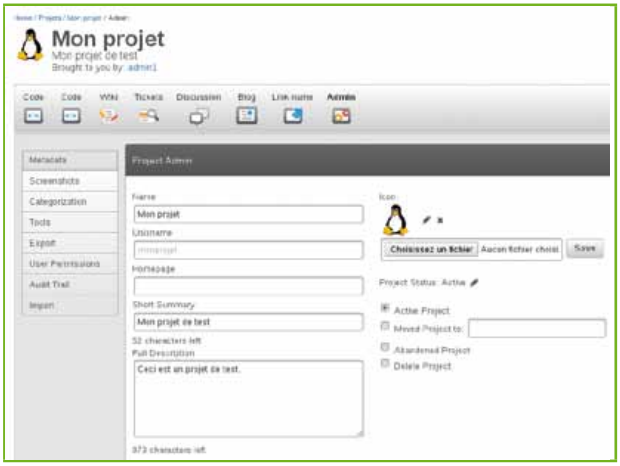


Figure 5

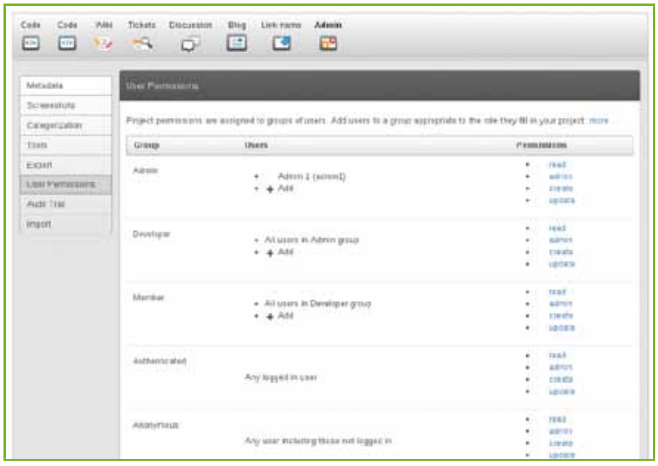


Figure 6

En plus de ***Metadata***, d'autres pages sont disponibles dans le menu vertical gauche pour gérer votre projet : pour lui ajouter des captures d'écran, lui assigner des catégories, etc. Le lien ***Tools*** de ce menu permet d'activer, désactiver, ainsi que configurer les outils. Vous pourrez par exemple changer leur titre (label), réordonner leur place dans le menu supérieur, ou assigner les permissions utilisateurs pour chaque outil **[3]**. Ces permissions se donnent sous la forme de groupes (pour assigner un utilisateur à un groupe, cela se passe via le lien ***User Permissions*** (Fig. 6)).

Notez que vous pouvez activer autant d'instances d'un outil que vous le désirez. Pour cela, vous donnez un nouveau nom, ainsi qu'un nouveau point de montage (c'est-à-dire une URL) à la nouvelle instance. Par exemple, nous avons activé un wiki par défaut pour notre projet fictif. Il a pour nom par défaut « Wiki », il est accessible via le lien ***Wiki*** du menu supérieur. Il a pour point de montage **`wiki`**, il est donc accessible à l'URL http://localhost:8080/p/monprojet/wiki. Si nous voulions ajouter un deuxième wiki, nous pourrions lui donner comme nom « Tutoriels » et comme identifiant **`tutoriels`** (Fig. 7). Il serait donc accessible à l'URL http://localhost:8080/p/monprojet/tutoriels. Dès que plusieurs instances d'un même outil existent, le menu supérieur se voit doté automatiquement d'une liste pour choisir parmi l'une des instances (Fig. 8).

Figure 7

Figure 8

3.2 Communiquer

Que serait un projet sans moyens de communication ? Vos utilisateurs se sentiraient ignorés, ou votre projet semblerait à l'abandon.

L'outil ***Discussion*** permet de maintenir un forum pour vos utilisateurs, afin de collaborer entre développeurs ou échanger avec les utilisateurs finaux. Vous pouvez également héberger votre propre documentation grâce à l'outil ***Wiki*** et votre propre blog grâce à l'outil... ***Blog***. Vous pourrez écrire vos pages de wiki (Fig. 9), vos messages de forum, ou vos billets de blog au format Markdown **[4]**.

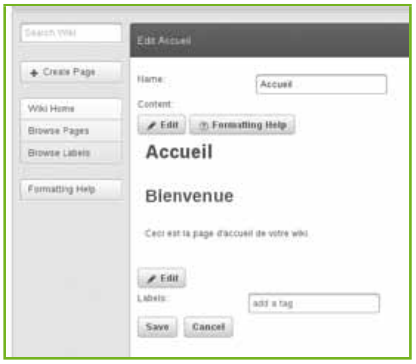


Figure 9

Il existe également l'outil ***External Link*** (Lien externe) qui prend la forme d'un lien dans le menu supérieur, nommé ***Link name*** par défaut. Cet outil permet à vos utilisateurs d'accéder facilement au site web externe de votre choix : votre site personnel hébergé sur un autre site, un lien vers un projet annexe, ou encore vers un service de paiement externe pour soutenir votre projet (Fig. 10). Pour configurer ce lien, allez à la page ***Tools*** de la section d'administration de votre projet.

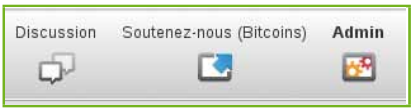


Figure 10

3.3 Tickets

C'est bien beau de communiquer avec vos utilisateurs via un wiki, un blog, ou qu'ils puissent vous poser des questions sur un forum. Un système de tickets reste le meilleur moyen pour mettre en place un suivi des tâches à accomplir : développement, bugs à résoudre, mise à jour de documentation, nouvelles fonctionnalités, etc.

L'outil ***Tickets*** d'Allura permet de mettre en place un tel système pour votre projet (Fig. 11). Lors de la création d'un ticket, vous pourrez lui attribuer toutes les informations utiles : jalons de versions (*milestones*), statut (ouvert, fermé, etc.), propriétaires,... Si vous avez besoin de définir de nouveaux milestones ou statuts, il faut aller dans l'administration de votre projet, page ***Tools***.

De cette page, vous pourrez également mettre en place un système de vote, pour que vos utilisateurs soutiennent ou non telle ou telle requête (Fig. 12, page suivante). Ainsi qu'activer une sorte de mini *mailing list* pour chaque ticket, afin que chaque participant d'un ticket soit tenu au courant par mail de l'évolution de son statut. Chaque ticket peut également être lié à un e-mail, pour pouvoir mettre à jour facilement via e-mail son évolution **[5]**.

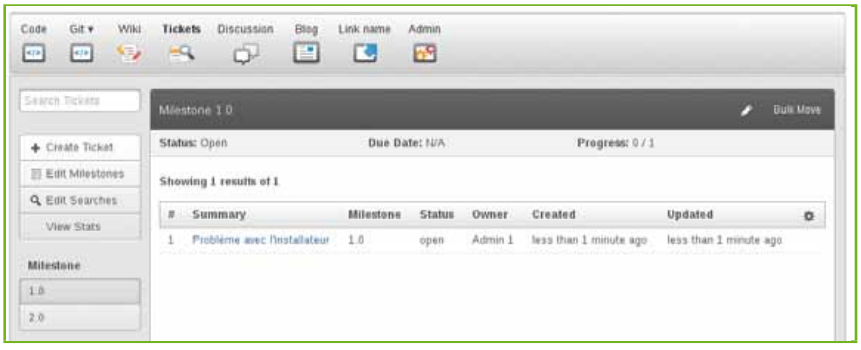


Figure 11

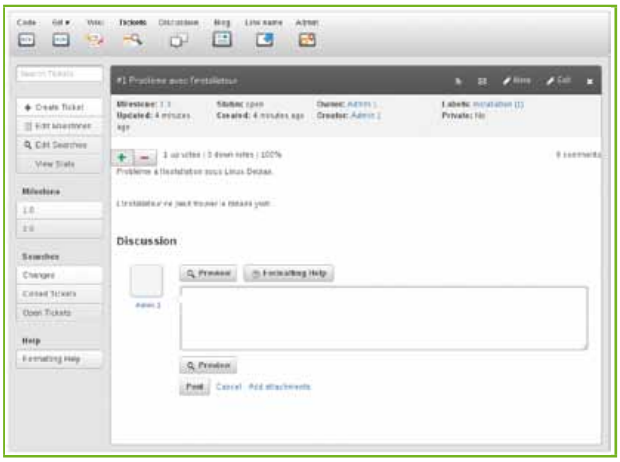


Figure 12

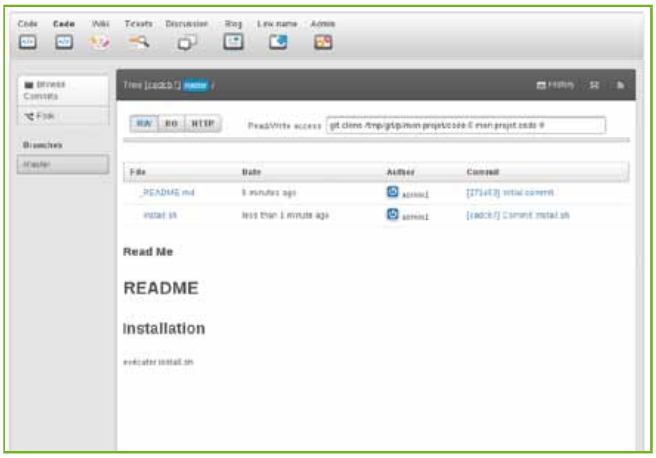


Figure 13

3.4 Dépôts de source

Allura est une forge logicielle...et jusqu'à présent, nous n'avons pas abordé ses fonctionnalités d'hébergement de code. Par défaut, Allura supporte les gestionnaires Git et Subversion. Lors de la création de notre projet fictif, nous avons d'ailleurs automatiquement activé un dépôt Git et un dépôt Subversion, qui apparaissent tous les deux avec le lien ***Code***, dans le menu supérieur d'outils (pour rappel, depuis la page ***Tools*** d'administration de votre projet, vous pouvez changer ces labels ***Code***, peu explicites, du menu supérieur, et ajouter autant de dépôts supplémentaires Subversion ou Git que nécessaire).

Vous pourrez donc héberger votre code, donner des permissions de lecture ou écriture sur les dépôts, et naviguer graphiquement dans l'historique de vos commits via l'interface web (Fig. 13).

Vous pourrez également lier vos tickets à vos commits, en précisant le numéro du ticket dans le message de votre commit. Vous pouvez voir que pour le commit dans la figure 14, son message « *Modif instruction installation* » a été préfixé par ***[#1]***, qui fait référence à notre ticket fictif précédent (Fig. 11 et 12).

En fait, ce système de liens est utilisable depuis n'importe quel outil, pour référencer n'importe quel élément d'Allura (Fig. 15). Une page d'un blog, une page d'un wiki, un commit, un ticket : tous ces éléments sont des objets d'un outil. Un tel objet est appelé un « artefact » selon la terminologie d'Allura. Vous pouvez référencer n'importe quel artefact, depuis un artefact du même ou d'un autre outil, qu'il appartienne au même projet, à un sous-projet, ou d'un autre voisinage **[6]**.

3.5 Accès aux données

Pour être le plus intégré possible, Allura propose différents moyens pour interfacer ses données. Ces différentes formes d'import et d'export sont vraiment fournies dans l'esprit de faire d'Allura la forge la plus ouverte possible **[7]**.

3.5.1 API

Allura expose une grande partie de ses données sous la forme d'une API web REST, au format JSON. Cela va vous

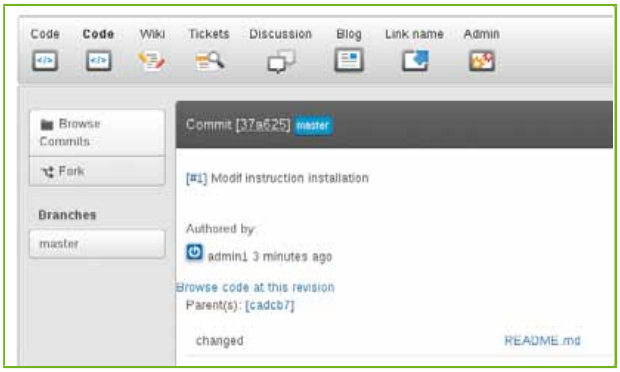


Figure 14

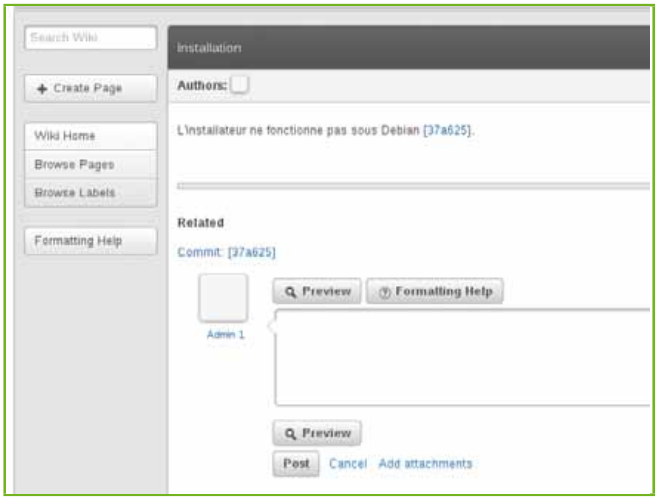


Figure 15 : Une page de wiki de notre projet, qui référence le commit 37a625 de notre dépôt.

permettre d'accéder à ses données sans passer par l'interface web classique. Les données des voisinages, projets, sous-projets, outils, artefacts sont pour la plupart accessibles via cette API, il suffit d'ajouter le préfixe /rest/ à l'URL de l'élément désiré.

Par exemple, en ligne de commandes, vous pouvez utiliser **wget** pour afficher la liste des tickets de notre projet :

```
$ wget http://localhost:8080/rest/p/mon-projet/tickets -q -O -
{"tickets": [{"summary": "Problème avec l'installateur", "ticket_num": 1}], "count": 1,
"milestones": [{"due_date": null, "complete": false, "name": "1.0", "closed": 0, "total":
1, "description": null}, {"due_date": null, "complete": false, "name": "2.0", "closed": 0,
"total": 0, "description": null}], "tracker_config": {"_id": "5242d54f100d2b0d7cbebcc3",
"options": {"ordinal": 5, "mount_point": "tickets", "EnableVoting": false,
"TicketMonitoringType": null, "mount_label": "Tickets"}}, "limit": 100, "saved_bins":
[{"sort": "mod_date_dt desc", "_id": "5242d54f100d2b0d7cbebccc", "terms": "!status:wont-fix
&& !status:closed", "summary": "Changes"}, {"sort": "", "_id": "5242d54f100d2b0d7cbebccb",
"terms": "status:wont-fix or status:closed", "summary": "Closed Tickets"}, {"sort": "",
"_id": "5242d54f100d2b0d7cbebcca", "terms": "!status:wont-fix && !status:closed", "summary":
"Open Tickets"}], "page": 0}
```

En plus d'être accessible en lecture, l'intérêt d'une telle API REST est d'être aussi accessible en écriture, ce qui va vous permettre d'automatiser certaines tâches, comme ajouter, supprimer ou éditer des tickets, des pages de wiki, de forum, depuis la ligne de commandes **[8]**.

3.5.2 Export

Vous pouvez exporter vos données en une archive au format JSON. Rendez-vous dans la section d'administration de votre projet, page ***Export***. Puis, choisissez quelles données vous souhaitez exporter : wiki, tickets, forum, ...

3.5.3 Import

Vous pouvez, depuis le menu ***Admin*** d'un projet ou sous-projet, importer les tickets et/ou les dépôts de services externes comme Git, Google Code ou Trac, ou importer les données précédemment exportées depuis une autre instance d'Allura (Fig. 16).

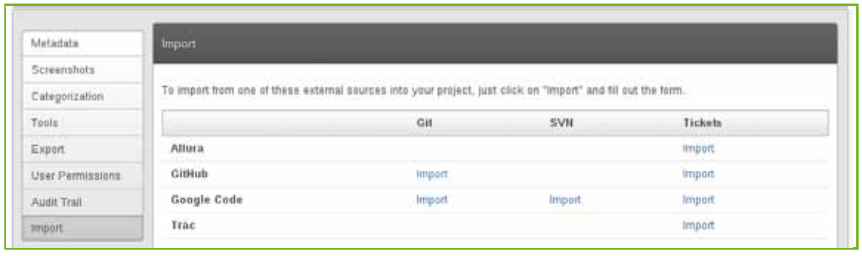


Figure 16

Conclusion

Allura semble très prometteur. Même s'il est utilisable en production, il semble ne pas être complètement mature, mais le fait que le framework soit utilisé pour faire tourner SourceForge donne assurance qu'il est en développement actif. Son aspect modulaire lui permet de pouvoir être étendu, certaines extensions tierces sont téléchargeables **[9]** et un tutoriel est disponible pour développer votre propre extension **[10]**.

Allura est ouvert, n'hésitez pas à participer à son évolution **[11]**, en contribuant au code **[12]**, en créant ou en votant pour des tickets **[13]**, ou simplement en discutant de son futur **[14]** ! ■

Références

[1] https://forge-allura.apache.org/p/allura/git/ci/master/tree/README.markdown

[2] http://downloads.vagrantup.com/

[3] https://forge-allura.apache.org/p/allura/git/ci/master/tree/Allura/docs/guides/permissions.rst

[4] http://sourceforge.net/p/forge/documentation/markdown_syntax/

[5] https://forge-allura.apache.org/p/allura/git/ci/master/tree/Allura/docs/installation.rst

[6] http://sourceforge.net/blog/allura-feature-highlight-artifact-linking/

[7] https://forge-allura.apache.org/p/allura/git/ci/master/tree/Allura/docs/migration.rst

[8] http://sourceforge.net/p/forge/documentation/Allura%20API/

[9] https://forge-allura.apache.org/p/allura/wiki/Extensions/

[10] http://sourceforge.net/u/vansteenburgh/allura-plugin-development/

[11] http://sourceforge.net/p/allura

[12] https://forge-allura.apache.org/p/allura/wiki/Contributing%20Code/

[13] http://sourceforge.net/p/allura/tickets/

[14] https://forge-allura.apache.org/p/allura/wiki/Asking%20Questions/

FAIRE CONNAÎTRE SON PROJET GRÂCE À TWITTER

par Sébastien Chazallet et Alicia Florez

Les réseaux sociaux sont devenus le moyen privilégié pour s'informer et donc, l'outil idéal pour faire connaître son projet. Cet article va présenter des utilisations de Twitter par des communautés libres et vous donner quelques clés pour en faire de même.

1 Introduction

Twitter est un outil de microblogage dont le principe consiste simplement à partager, ou « tweeter », des petits messages de 140 caractères, des « tweets » ; ses utilisateurs sont appelés « Twittos ». Il permet d'échanger rapidement des informations, humeurs ou autres joyeusetés et est perçu comme étant ludique. Chaque message que vous lisez peut être re-tweeté de manière à le partager avec ceux qui vous suivent. Ce processus permet une diffusion très rapide de l'information. On pourra reprocher le phénomène de « buzz » qui pollue souvent l'actualité en donnant plus d'importance à des épiphénomènes qu'à des informations plus importantes, mais c'est le propre de l'Homme.

En ce qui nous concerne, le fait d'utiliser Twitter va simplement nous permettre d'être visibles par la très large communauté de ses utilisateurs. Donc, si vous n'avez pas de compte Twitter, à vos claviers ! Normalement, une minute plus tard, avec une adresse mail et une description, vous avez votre compte. Pas besoin de tutoriel. Premier point important pour travailler votre visibilité : mettez une vraie description, même si vous êtes « un astronaute amateur de bons vins et papa comblé ».

Maintenant, il ne vous reste plus qu'à vous rapprocher de votre communauté et pour ça, allez les chercher !

2 Comment les communautés utilisent Twitter ?

Pour comprendre comment utiliser l'outil Twitter pour promouvoir son logiciel, il faut savoir comment les Twittos utilisent Twitter. Une des choses qui va caractériser votre compte sera vos abonnements ; s'ils s'articulent autour d'un même thème, cela apportera de la cohérence au contenu de votre compte. Vous pourrez alors retweeter les contenus que vous souhaitez partager à la fois pour les diffuser, mais aussi pour vous y associer. Vous associerez alors votre image, ou celle du logiciel pour lequel vous avez créé le compte, à celle des auteurs de ces tweets.

Il est donc temps de faire un petit tour des différents comptes, à commencer par les « gourous » du logiciel libre ou des organisations importantes .

On peut aussi vous inviter à regarder la manière dont certaines fondations qui conçoivent des logiciels libres communiquent. L'exemple par excellence est la fondation Mozilla, avec les comptes **@mozilla**, **@MozillaPR_FR** et **@firefox**. On voit comment la fondation utilise plusieurs comptes pour gérer des aspects totalement différents dans sa communication.

L'étape suivante consiste à regarder comment cela se passe pour une communauté moins structurée que celle de Mozilla, par exemple celle de GIMP. On peut citer le compte **@GimpChat**, qui permet d'échanger à propos du logiciel en général - ce qui est utilisé comme une alternative à un forum - ainsi que **@GimpScripts**, qui permet d'échanger sur la réalisation de greffons pour GIMP et qui est donc plutôt réservé à des développeurs.

On peut maintenant regarder comment cela se passe pour des entreprises, qui elles aussi doivent gérer leur communication. On peut citer l'exemple d'OpenERP, où comme pour la fondation Mozilla, on retrouve plusieurs comptes tous dédiés à des utilisations particulières. Vous verrez qu'il y a beaucoup de comptes (Fig. 1), parmi lesquels on peut identifier :

- le compte générique d'OpenERP, dédié à la communication de l'entreprise (recrutement, marchés, ...) ;
- celui de la *team* : **@openerpteam**, dédié à la communication autour de la sortie de nouveautés, ou à des événements de l'entreprise ;
- celui lié à leur gestionnaire d'anomalies : **@openerpbugs**, on reparlera de ce type de fonctionnalité ;

Figure 1

- ceux liés à des personnalités de l'entreprise, comme Fabien Pinckaers, le créateur d'OpenERP (**@fopenerp**) ;
- et bien d'autres...

Si le système a fait ses preuves pour les logiciels libres renommés, on peut se demander ce qu'il en est pour les petits logiciels, qui n'ont pas encore de communauté... Et effectivement, difficile de croire que du jour au lendemain vous la trouverez, ainsi que la popularité, le succès et que vous ouvrirez la nouvelle entreprise de référence dans la Silicon Valley !

Le travail permettant de se faire connaître se fait surtout à l'extérieur, en animant des sites communautaires, des blogs, en participant au développement de logiciels libres, bref, en se faisant un nom dans la communauté ciblée. Après, on pourra utiliser des outils pour intégrer ses « feeds » Twitter à son blog ou site.

Il est important de ne pas douter de l'utilité d'un compte Twitter, même pour les petites communautés. On peut citer par exemple le très récent *Fait Main Magazine* (**@FaitMainMag**), dont le compte Twitter permet d'informer sur l'avancement des articles, la sortie des numéros, mais permet également aux auteurs de partager les informations entre eux.

3 Les greffons utiles pour intégrer Twitter

3.1 Blogs

On en est donc arrivé au point où l'on est conscient que Twitter est plus utile s'il est intégré à d'autres moyens de communication. Et quel meilleur moyen de communiquer que le fait de créer un blog ? L'idée est donc d'utiliser Twitter pour informer du fait que vous avez publié un nouvel article, ceci automatiquement.

Cela se fait simplement en installant un greffon et en le configurant proprement. On peut citer le greffon *Publisher*, réalisé pour le moteur de blog WordPress (un des plus connus). Ce dernier a la particularité de ne pas gérer uniquement Twitter, mais une bonne quinzaine d'autres réseaux sociaux. Son intégration permet également de vous informer sur l'ensemble des statistiques hebdomadaires, afin de suivre combien de fois les tweets émis à partir de l'outil ont été retweetés, ou mis en favoris.

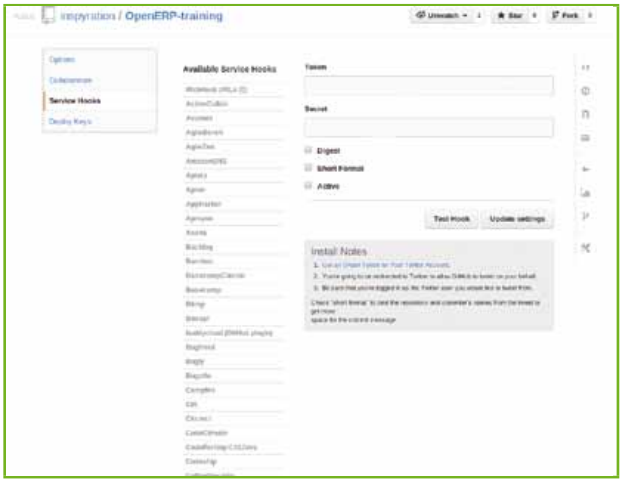


Figure 2

3.2 Planet

Un *Planet* est un outil généralement géré par une communauté reconnue, qui agrège des contenus relatifs aux thèmes qui intéressent la communauté, par exemple planet.debian.org. Ces pages sont généralement très suivies par les membres des communautés, ou ceux qui s'y intéressent de près ou de loin.

Pour ce qui concerne cet article, il faut noter que des planets comme planet.python.org twittent la parution de chaque article. Si vous êtes capable d'identifier un planet qui corresponde au profil de votre projet et que vous pouvez publier dessus, vous gagnerez automatiquement de la visibilité sur Twitter !

3.3 Dépôts de code

Il peut également être utile d'informer à propos de l'avancement de vos développements. C'est ce que proposent des outils comme PyPI (*Python Package Index*, le site qui indexe l'ensemble des modules externes du langage Python qui sont packagés), qui à grande échelle, publie sur Twitter chaque fois qu'une nouvelle version d'un des modules est publiée (autant dire plusieurs dizaines de tweets chaque jour !).

En tant que lecteur, ceci est extrêmement utile lorsque l'on fait de la veille technologique.

À plus faible échelle, des outils comme GitHub ou Bitbucket, que vous ne pourrez faire autrement qu'utiliser, vous permettent également de twitter chaque commit. Libre à vous de créer un compte pour les commits et un pour votre communication officielle.

GitHub met à disposition un service « Hooks », qui contient plus d'une centaine de services dont Twitter. Pour l'utiliser, rendez-vous dans les paramètres du projet, section ***Service Hooks***, choisissez le service ***Twitter***, cliquez sur ***Get an OAuth Token for Your Twitter Account***. Pour le reste, laissez-vous guider (Fig. 2 et 3).

Figure 3

Bitbucket est tout aussi simple d'utilisation. Cela se passe toujours dans les paramètres du projet, dans la section ***Hooks***. Sélectionnez ***Twitter***, puis cliquez sur ***Add hook***, puis sur ***Authorize your Twitter account to communicate with Bitbucket***. Sauvegardez.

4 Utilisez Twitter de la manière dont vous le décidez

Si les greffons sont très utiles, il est possible qu'il n'en existe pas pour son propre site ou que l'on ait un besoin très particulier qui ne soit pas couvert. Ou alors qu'on trouve un logiciel qui fasse ce que l'on veut, mais que ce dernier soit payant. Pas de panique. Faites le travail vous-même! C'est d'ailleurs assez simple (pour un développeur, s'entend). Il suffit de faire sa propre application ou son propre greffon.

La première étape consiste à gérer la manière de s'authentifier sur Twitter. Pour cela, allez sur le site suivant https://dev.twitter.com. Authentifiez-vous et vous tomberez sur une page vierge contenant vos applications et un bouton vous permettant d'en déclarer une, ce que nous vous proposons de faire.

Une fois les 4 champs renseignés (ne rien mettre dans le *Callback* pour les premiers essais), on voit une page contenant l'ensemble des informations utiles. Vous pourrez remplir l'onglet ***Settings*** pour les compléter, puis après avoir sauvegardé, vous pourrez cliquer sur le bouton en bas à gauche, lequel permet de générer les *tokens*. Les plus importantes se trouvent sous le carré noir de la figure 4.

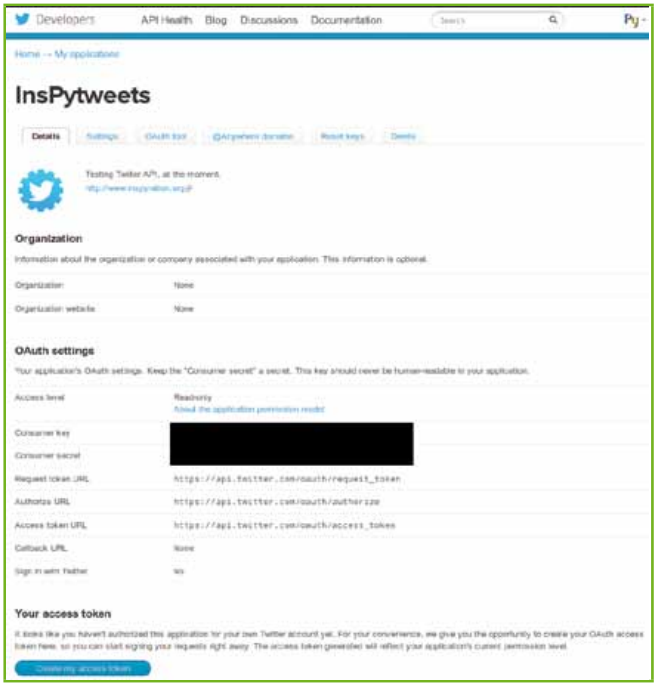


Figure 4

Pour le reste, cliquez sur l'onglet ***Oauth tool*** et copiez les variables qui sont proposées dans votre code source (vous vous doutiez bien qu'à un moment donné, on en viendrait à du code source, non ? Rassurez-vous, c'est du Python !). Pour cela, il faut installer le module dédié :

```
$ pip install twitter
```

En réalité, on installe également pas mal d'outils, qui permettent d'accéder à Twitter via la console, mais je vous laisse découvrir ça et jouer avec. L'option **`--help`** devrait suffire à vous indiquer comment les utiliser.

Ensuite, on peut écrire notre code source (Python 3) et commencer par importer ce module :

```
from twitter import *
```

Le script commence tout naturellement par s'authentifier, ce qui est fait ici, en admettant que les constantes sont bien renseignés :

```
auth=OAuth(ACCESS_TOKEN, ACCESS_TOKEN_SECRET, CONSUMER_KEY, CONSUMER_SECRET)
```

Vous pouvez maintenant commencer par aller chercher tous les tweets présents sur votre Home, dans twitter :

```
t = Twitter(auth=auth)
home_timeline = t.statuses.home_timeline()
```

Si l'on s'attarde un peu sur cet objet, on voit qu'il s'agit de ça :

```
>>> type(home_timeline)
<class 'twitter.api.WrappedTwitterResponse'>
```

Mais en fait, pas de panique ! On n'a ici que des chaînes de caractères, des listes et des dictionnaires. Rien de bien sorcier, donc. Voici comment aller à l'essentiel (parce qu'il y a beaucoup d'informations, en réalité) :

```
[(d['created_at'], d['user']['name'], d['text']) for d in home_timeline]
```

Comme les objets peuvent être un peu gros, ce n'est pas forcément facile de les lire, mais ceci devrait vous aider :

```
home_timeline[0].keys()
home_timeline[0]['user'].keys()
```

Ainsi, pas à pas, on peut réussir à deviner tout seul la manière de bien utiliser ces objets et on peut formater les réponses que l'on obtient. Après, il n'y a plus qu'à lire l'API (https://dev.twitter.com/docs/api/1.1) pour savoir comment utiliser ces objets. Et hop ! Un petit pot-pourri :

```
retweets_of_me = t.statuses.retweets_of_me()
followers= t.followers.ids()
friends = t.friends.ids()
suggestions = t.users.suggestions()
```

Pour lire ces objets, c'est toujours le même principe. On a de la chance, Python permet de faire de l'introspection très facilement ! Voici un petit dernier pour la route :

```
user_timeline  =  t.statuses.user_timeline(screen_name='alex_gaynor')
```

Et pour le comprendre, toujours la même source : https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline.

Conclusion

Twitter est un outil d'une très grande simplicité et c'est ce qui fait sa force. Accessible, agréable à utiliser, il a su fédérer un très grand nombre d'utilisateurs. Utiliser Twitter, c'est donc accéder à ces utilisateurs et les informer sur votre projet. Les premiers temps, vous vous ferez connaître par des actions extérieures, comme l'écriture d'articles sur votre blog, ou votre activité sur la plateforme publique de dépôt de code.

Pour être pleinement efficace, il sera utile de lier votre compte Twitter à ces outils, comme on l'a vu. Ensuite, lorsque vous aurez votre communauté de *followers*, votre communication sera facilitée par ceux qui vous re-tweeteront. C'est un travail de communication long, parfois fastidieux, mais c'est un travail nécessaire pour promouvoir votre logiciel. ■

DISTRIBUER SON PROJET PYTHON

par Sébastien Chazallet

Vous avez réussi à créer un produit intéressant, votre code source fonctionne correctement, vous avez passé les premiers tests avec succès, mais voilà, ce n'est « que » du code source. Si vous voulez que votre produit rencontre son public, il est nécessaire, selon la cible, d'industrialiser sa mise en production ou de faire en sorte qu'il soit facilement installable par les utilisateurs. Cet article se propose de montrer comment distribuer son code source et faire de son produit quelque chose de vraiment abouti et prêt à l'emploi. Il est axé sur la distribution de code Python.

1 Introduction

Pour réaliser cette tâche, on va utiliser ce que l'on nomme « un utilitaire » de distribution. Le but de cet utilitaire est de formaliser la manière dont votre future application doit s'installer. En effet, pour réaliser votre produit, vous avez certainement utilisé des modules ou paquets particuliers, ce que l'on considère comme des « dépendances ». Vous devez les déclarer afin que ceux-ci s'installent en même temps que votre propre produit.

On distingue deux types de cibles, les utilisateurs étant sous Windows et les autres. Pour les premiers, il faut produire des fichiers exécutables, lesquels devront être téléchargés par les utilisateurs, puis exécutés. Ces fichiers devront contenir l'ensemble de votre produit, mais également l'ensemble des dépendances, de manière à ce que tout puisse marcher de manière autonome.

Pour les seconds, les dépendances sont traitées comme le sont celles des paquets Linux. C'est-à-dire que l'on se contente de demander à ce que les dépendances soient installées en même temps que notre produit. La différence peut paraître subtile, mais elle ne l'est pas.

Dans les deux cas, il est possible qu'une des dépendances soit écrite en C et doive donc être compilée et il faut gérer une procédure de désinstallation. Le problème est donc loin d'être trivial !

Toujours est-il que l'utilitaire de distribution va totalement masquer la complexité. Il va en effet gérer automatiquement la construction d'un exécutable Windows, d'un paquet Debian, Fedora ou Mac, ou plus simplement d'un paquet Python, lequel ressemble beaucoup à un paquet Linux. L'utilisateur n'aura plus qu'à installer votre application comme n'importe quelle autre application.

2 Commencer par le début

Pour un projet contenant simplement un fichier **spam.py**, il suffit de se créer un fichier **setup.py** contenant ceci :

```
from distutils.core import setup
setup(name='spam',
      version='1.0',
      py_modules=['spam'],
      )
```

Il s'agit d'une version minimaliste. Il est nécessaire maintenant de l'agrémenter un peu :

```
#!/usr/bin/env python

from distutils.core import setup

setup(name='spam',
    version='0.42',
    description='H2G2',
    author='moi',
    author_email='moi@me.net',
#    maintainer='lui',
#    maintainer_email='lui@him.net',
    url='http://www.monsite.org/applis/spam/',
    download_url='http://www.monsite.org/
    download/spam/',
    description='une présentation du produit,
    simple et efficace',
    long_description='une description plus
    détaillée',
    licence='AGPL',
    packages=['pkg1', 'pkg1.subpkg'],
     )
```

Toutes ces informations seront utiles pour la génération de l'installateur.

Si l'auteur n'est pas unique, le mainteneur doit être précisé. Les deux informations ne cohabitent pas, le mainteneur étant l'information qui est retenue en cas de conflit.

Pour de petits projets, à la place d'utiliser **packages** pour référencer les packages au sens Python du terme (répertoires), on peut utiliser **py_modules** pour référencer des modules au sens Python du

terme (fichiers). Si on doit inclure d'autres fichiers, on peut utiliser **package_data** ou encore **data_files**.

Enfin, on en vient à ce qui est le plus important pour bien référencer votre projet dans l'écosystème Python, les catégories. En effet, plusieurs catégories sont prédéfinies. Voici un exemple tiré de la documentation officielle, qui permet de mesurer d'un clin d'œil la manière dont elles doivent être utilisées :

```
classifiers=[
          'Development Status :: 4 - Beta',
          'Environment :: Console',
          'Environment :: Web Environment',
          'Intended Audience :: End Users/Desktop',
          'Intended Audience :: Developers',
          'Intended Audience :: System Administrators',
          'License :: OSI Approved :: Python Software Foundation License',
          'Operating System :: MacOS :: MacOS X',
          'Operating System :: Microsoft :: Windows',
          'Operating System :: POSIX',
          'Programming Language :: Python',
          'Topic :: Communications :: Email',
          'Topic :: Office/Business',
          'Topic :: Software Development :: Bug Tracking',
          ],
```

Enfin, vous pouvez créer un fichier **setup.cfg** qui vous permettra de passer des options par défaut à l'utilitaire de distribution. Il s'agit simplement d'un fichier comprenant des sections dont le nom est entre crochets et des lignes **clé=valeur**. Pour connaître la liste des clés, faites :

```
$ python setup.py --help COMMANDE
```

en utilisant le nom de la commande qui vous intéresse (et que l'on va présenter plus tard).

Une fois que tout ce travail préparatoire est réalisé, il ne reste plus qu'à générer une distribution de source, c'est-à-dire un fichier d'archive contenant tout ce qu'il faut. Cela nous permettra de vérifier que rien n'est oublié. Voici la commande :

```
$ python setup.py sdist
```

Le mot **sdist** est une des commandes dont il était question quelques lignes plus haut. Par défaut, cette commande va générer un fichier ZIP sous Windows et un fichier .tar.gz sous Unix. Il est possible de préciser le format attendu, voire d'en demander plusieurs en les séparant par une virgule. Voici comment tous les demander :

```
$ python setup.py sdist --formats=gztar,zip,bztar,ztar,tar
```

On obtient, respectivement, des fichiers .tar.gz, .zip, .tar.bz2, .tar.Z et .tar (non compressé).

3 Créer un fichier installable

Une fois le travail préparatoire réalisé, il suffit de procéder comme suit, pour créer un fichier installable :

```
$ python setup.py bdist
```

Oui, une lettre de différence, on est bien d'accord. Mais c'est ce qui fait toute l'importance. Comme pour **sdist**, on génère ici un fichier installable conforme à la plateforme sur laquelle on se trouve. On a le choix entre les options suivantes (entre autres) :

```
$ python setup.py bdist --formats=rpm,
pkgtools,sdux,wininst,msi
```

Ceci permet de créer, respectivement, des fichiers pour Fedora, Solaris, HP-UX, Windows sous la forme de fichiers auto-extractibles et enfin, Windows encore, mais sous la forme de fichier Microsoft Installer.

Pour être complet, il faut savoir qu'il y a une myriade de petits détails à configurer si on veut entrer dans les détails et faire quelque chose d'adapté. Parmi les moins petits de ces détails, il y a le fait qu'il soit utile d'écrire un fichier **SPEC** pour donner à votre RPM toutes les informations pour qu'il s'intègre bien à votre système, et le fait qu'il soit possible de demander l'élévation de privilèges pour les utilisateurs de Windows Vista et supérieur.

Enfin, comme vous pouvez le constater, il n'est malheureusement pas possible de générer directement et facilement un fichier pour Debian. Vous devrez en passer par l'utilisation d'un produit tiers comme **alien** pour générer un fichier DEB à partir d'un RPM.

Heureusement, sous Linux, l'installation de quelques fichiers à l'aide des sources seules est quelque chose de trivial et on passera donc par là.

Conclusion

L'utilisation de ces outils est loin d'être triviale. Même s'ils cachent beaucoup la complexité, il est nécessaire d'avoir une bonne culture générale sur le fonctionnement des systèmes cibles et de s'y casser les dents un petit moment avant de bien maîtriser ces outils. Mais au final, c'est un maigre prix à payer pour élargir considérablement la cible des potentiels utilisateurs de votre produit, simplement en faisant en sorte qu'il soit facile à installer. ■

LES OUTILS INDISPENSABLES POUR DISTRIBUER SON PROJET

par Sébastien Chazallet

La distribution d'un projet est l'élément principal qui fait que le projet est réellement open source. En effet, un projet - même sous licence libre - dont le code source n'est pas distribué n'est pas un projet open source. À ce jour, il existe un assez grand nombre de plateformes reconnues. Nous allons en présenter trois : GitHub, Bitbucket et Launchpad.

1 Introduction

1.1 Forges logicielles

Le positionnement des trois plateformes est clair : il s'agit de « forges logicielles ». Leur but est de permettre de déposer du code et de faciliter le travail autour du code. On est donc, a priori, uniquement sur un registre purement technique.

Quel est l'intérêt d'aller déposer son code sur l'une de ces plateformes alors qu'il suffirait d'installer sa propre solution, comme Trac, par exemple ? La réponse est simple : chacune de ces plateformes héberge de très nombreux projets, dont les plus importants du monde open source. Il y a donc des milliers de personnes qui sont des contributeurs potentiels et qui pourront mieux connaître votre projet, voire y contribuer s'ils n'ont pas besoin de découvrir une nouvelle plateforme, de se créer un nouveau compte, ...

Mettons-nous à leur place : sur ces plateformes, chacun est maître de son destin. Le principe consiste à dupliquer un projet (*fork*) pour le mettre dans son portefeuille et on est alors libre d'y ajouter n'importe quelle contribution, que le propriétaire du projet originel soit d'accord ou non. Bien sûr, le but est de soumettre toute modification (`merge request`) au propriétaire du projet originel, de manière à ce qu'elle soit intégrée. Mais s'il ne le souhaite pas, le contributeur n'est pas bloqué et peut continuer à gérer son propre dépôt en recevant toutes les améliorations apportées dans le projet originel (ceci grâce aux gestionnaires de versions décentralisés, sans qui rien de tout cela ne serait possible).

En tant que créateur d'un logiciel libre, le fait qu'un utilisateur puisse ouvrir un ticket pour signaler une anomalie, pour offrir une suggestion d'amélioration, ou pour apporter un commentaire est quelque chose d'essentiel. Par ce geste, l'utilisateur devient contributeur et là encore, le fait d'être sur une plateforme de renom permet d'élargir le nombre potentiel d'utilisateurs/contributeurs.

Il ne faut pas oublier que bien que l'on soit à l'ère informatique, c'est le bouche à oreille qui est le principal vecteur de diffusion d'un logiciel libre, même si cela se fait par chat, mail, vidéo-conférence, ou tout autre moyen technologique. Prendre en considération les avis ou demandes de ces contributeurs est donc essentiel à la réussite d'un projet libre et c'est également une véritable tâche qui demande beaucoup de temps.

À ce titre, cet article contiendra quelques passages techniques, pour expliquer comment utiliser ces forges, mais sans entrer dans les détails. Si vous souhaitez apprendre à maîtriser un gestionnaire de versions, consultez les anciens numéros de *GNU/Linux Magazine*.

Il est à noter que la création d'un compte n'est pas obligatoire pour consulter un projet, naviguer dans les différentes fonctionnalités, ou encore pour cloner un dépôt de manière anonyme (via HTTPS). Par contre, il est nécessaire d'avoir un compte pour créer un ticket d'anomalie, poster un commentaire (selon la configuration) ou encore cloner un dépôt en lecture et écriture (via SSH).

1.2 Outils de communication

On pourrait faire l'erreur de croire que ces plateformes ne sont que des forges logicielles, que des outils purement techniques. En réalité, publier son projet sur une telle plateforme lui donne de la visibilité. Alors, il faut soigner sa

communication, ce qui passe par deux choses essentielles : expliquer à quoi sert votre logiciel et informer sur votre avancement, votre vision de l'évolution du projet, vos objectifs, ...

À ce titre, chaque plateforme met à disposition certains outils, qu'il convient d'utiliser de manière mesurée. On peut ainsi utiliser le site comme une vitrine pour mettre en avant les avantages de notre création et mettre des liens vers les sites relatifs, en particulier le site qui contiendra la documentation, élément vital du projet.

Pour terminer, il faut noter que parmi les trois exemples qui seront abordés dans cet article, GitHub et Bitbucket se sont clairement tournés vers des fonctionnalités orientées réseau social, alors que Launchpad s'est plutôt orienté vers les professionnels. On va voir tout ceci plus précisément.

2 GitHub

2.1 Généralités

GitHub est LA plateforme de distribution de code la plus populaire. Elle est la plus importante en termes de nombre de contributeurs et de projets. Parmi eux, se trouvent de nombreux projets open source de référence.

L'offre est claire. Il est possible d'héberger autant de projets publics que l'on souhaite. Par contre, tout dépôt privé est payant. On dispose de quelques outils permettant de rechercher des projets, de faire des statistiques par rapport aux langages utilisés, sur l'activité des projets, le tout étant relativement séduisant. Il s'agit d'une plateforme moderne et qui n'est pas seulement un dépôt de code. Comme son slogan le laisse présager, elle est orientée réseau social. Il est en effet possible de communiquer sur son projet, de commenter, ou amender les autres très simplement, sans être un expert du développement.

On trouve également quelques outils à destination des développeurs. Par exemple,

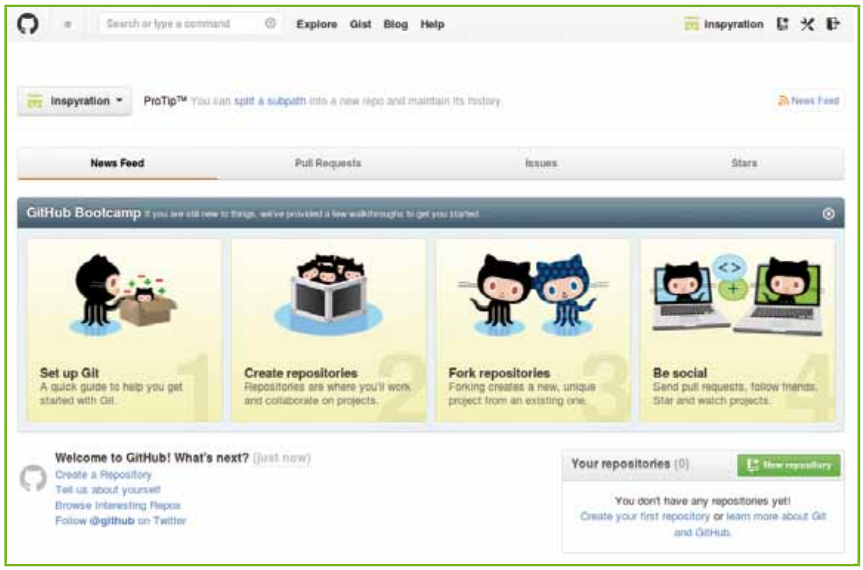


Figure 1

l'outil **Gist** permet de copier-coller du code (hors de tout contexte projet) et de le partager afin d'étayer une discussion entre deux personnes, par exemple. Alors que ce partage de code ne serait pas évident par chat, ou polluerait une boîte courriel, cette solution permet de partager le code en profitant de la coloration syntaxique.

On a également accès à un blog, qui relaie les nouveautés de la plateforme et qui stimule son attractivité.

Avant de passer aux choses sérieuses, on pourra créer son compte, ce qui peut prendre un certain temps. Le temps de trouver un pseudo qui ne soit pas déjà pris, en fait. La chose la plus difficile à faire de tout ce qui est présenté dans l'article ! Ce pseudo sera utilisé en tant que préfixe pour l'ensemble de vos projets. Il sera donc un fort élément d'identification et est à choisir en conséquence.

Un point cependant ne doit pas être minoré : lorsque l'on débarque pour la première fois sur la plateforme, on arrive à une page permettant d'accéder à une aide très complète. Mais il y a des chats en costumes. Des chats... *Sérieusement ?*

Pour une entrée en douceur, les aspects techniques les plus avancés ne seront abordés que dans la partie suivante, portant sur Bitbucket. *Sérieusement, des chats ?*

2.2 Créer un projet

Cela prend environ une minute. Il suffit de remplir un petit formulaire avec le nom du projet et sa description, choisir s'il doit être public ou non. Il y a cependant une petite subtilité : il est possible d'ajouter automatiquement un fichier `README`, ainsi que de choisir une licence et de configurer un fichier `gitignore` automatiquement en fonction de son langage, ce qui peut faire gagner pas mal de temps. *Des chats en costumes, sérieusement ?*

Figure 2

Ceci fait, on peut aller déposer notre projet. Pour cela, trois étapes, et une préliminaire : repérer l'adresse de clone de notre projet. Elle se trouve en bas à droite sur la capture et on choisira

SSH de manière à pouvoir lire et écrire dans le dépôt. Une fois cette information dans le presse-papiers, il suffit de configurer le serveur distant sur notre poste :

```
$ git remote add github git@github.com:inspyration/OpenERP-training.git
```

Git fait cela très bien. Ensuite, il faut mettre à jour les données de notre dépôt local :

```
$ git fetch github
```

Et il ne reste plus qu'à récupérer les trois fichiers créés automatiquement (**README**, le **gitignore** et la licence) et à pousser notre code :

```
$ git pull github master
$ git push github master
```

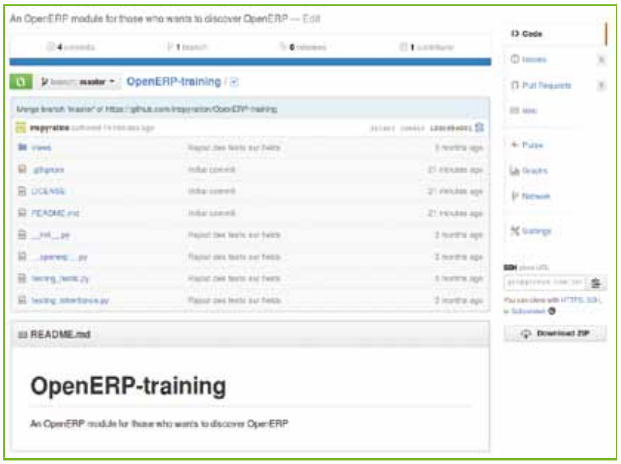


Figure 3

2.3 Organiser ses branches

Créer une branche sur GitHub se fait en allant sur la page présentant l'arborescence du projet, en cliquant sur le nom de la branche (en haut, juste au-dessus de l'arborescence) et en donnant un nouveau nom de branche. L'étape qui suit est d'informer le dépôt sur son ordinateur de travail de la création de la nouvelle branche, ce qui sera présenté dans le chapitre sur Bitbucket.

Le fait de créer une branche permet de réaliser des évolutions différentes et plus généralement, de séparer les opérations de maintenance correctives de celles de maintenance évolutives. Le fait de pouvoir raccrocher (*merger*) différentes branches permet de capitaliser le travail effectué à divers endroits, l'idée étant d'éviter toute gestion manuelle de patch ou pire, d'éviter des copier-coller.

Si sur un plan technique la création d'un nombre très important de branches peut se faire sans problème, on limitera ceci au poste de travail, lequel doit différencier les évolutions, par exemple. Sur une plateforme publique, les branches seront organisées, elles aussi, à des fins de communication. Une branche par version, par jalon (*milestone*), par exemple, permet de donner de la visibilité à l'avancement de vos travaux.

2.4 Wiki

Ce qui nous intéresse particulièrement sur la plateforme, ce sont ses outils de communication. Et l'outil par excellence mis à disposition est simplement un wiki. Cela nous permet de générer quelques pages directement sur la plateforme. Là où GitHub fait fort, c'est qu'il permet d'écrire son wiki dans plusieurs langages, dont le sublime reST (utilisé également pour Sphinx) et le merveilleux Markdow, présenté succinctement à la section suivante. L'outil est très succinct, mais très facile à utiliser et donc, particulièrement efficace. *Des costumes sur des chats, sérieusement ?*

Il est à noter que le wiki est lui-même un dépôt Git (indépendant du dépôt principal, qui est votre projet et qui ne sera pas pollué). À ce titre, il peut être cloné, modifié, puis poussé. Cette astuce est pratique si l'on souhaite travailler sur son poste ou en mode hors ligne. Cela permet également, entre autres, d'ajouter des images dans le projet et ainsi, de faire une mise en page attractive, mais ces images ne devraient sérieusement pas avoir quoi que ce soit à voir avec des chats (voire des costumes)...

Il s'agit là d'un canal de communication essentiel avec la communauté, qui doit pouvoir trouver toute l'information nécessaire pour utiliser le projet et y contribuer. Le wiki peut également faire référence à votre site Internet ou à la documentation du projet.

Un autre canal de communication essentiel est la mise en forme du fichier **README**. On s'attend à trouver des informations capitales sur la manière d'installer le logiciel, mais il doit rester minimaliste.

2.5 Gestionnaire de projet

GitHub permet également de faire de la gestion de projet minimaliste : on peut déclarer des jalons (*milestones*) et y attacher des évolutions qui sont en réalité des fiches qui tiennent plus de la *todolist* que d'autre chose. Le tout permet néanmoins d'anticiper une feuille de route. Le même outil permet aux contributeurs de signaler des anomalies, ou bien de faire des suggestions en créant des fiches de la même manière que précédemment. La possibilité de faire des commentaires sur ces fiches permet d'effectuer un suivi de l'anomalie, l'évolution ou la suggestion.

En termes de communication, le fait que quelqu'un vous signale une erreur fait de lui un contributeur. En ce sens, pour le succès du projet, il est important de prendre en considération sa demande, c'est-à-dire de lui répondre dans des délais corrects, voire d'effectuer des corrections, par exemple.

Figure 4

La réactivité dans ce type de situation est l'un des critères de maturité d'un projet libre. Il ne faut néanmoins pas perdre de vue qu'il s'agit là d'un vrai travail, consommateur de temps et par conséquent, il convient de l'anticiper.

Si vous gérez bien vos *milestones* et vos fiches, vous donnerez une excellente visibilité de votre projet à tous les utilisateurs qui le consultent. Par contre, cet outil reste vraiment minimaliste et s'il est suffisant pour la plupart des besoins, il n'est pas un outil de gestion de projet au sens où il pourrait être utilisé en entreprise ou encore au sein d'une communauté importante et structurée, lesquelles auraient besoin d'un véritable outil pour prioriser, organiser, planifier et suivre les développements. On regrettera également qu'il n'y ait pas d'outils d'intégration continue (tests unitaires, outils de qualimétrie, ...).

L'autre aspect de cet outil est que toutes les activités sont tracées et que l'on peut les visualiser à l'aide de graphiques. Ceci est utile pour déterminer à quel point un projet open source est actif, pour identifier les contributeurs les plus prolifiques (et leur proposer de rejoindre votre entreprise ?), ...

Des chats... Sérieusement ?

3 Bitbucket

3.1 Généralités

En anglais, *bit bucket* est un synonyme de « trou noir » ou d'« hyperespace », dans le sens particulier où ce trou noir est réputé être responsable des disparitions inexpliquées de données.

Bitbucket est une plateforme créée pour gérer des dépôts de code Mercurial. Elle permet également de gérer les dépôts Git depuis fin 2011. Elle est écrite en Python à l'aide du framework Django. Il est possible d'y créer des dépôts publics sans limitations et des dépôts privés en quantité infinie, mais limités en nombre d'utilisateurs (5). Au-delà, il faudra utiliser une version payante.

La plateforme est sans conteste la plus utilisée après GitHub et compte également dans son catalogue nombre d'applications open source de référence. Comme pour GitHub, la création d'un compte se fait en moins d'une minute, puisque le login est simplement une adresse de courriel et non un pseudo. On trouvera cependant un peu moins d'options de configuration.

Les deux plateformes sont très proches, autant au niveau de leur organisation qu'au niveau des fonctionnalités, lesquelles sont souvent un peu plus poussées sur GitHub. Par conséquent, l'ensemble des remarques faites sur GitHub sont généralement valables pour Bitbucket. Sauf qu'ici, pas de trace de chats ! Il y a cependant une différence très importante : Bitbucket gère à la fois Mercurial (historique), mais également Git. Et c'est un point positif extrêmement important !

3.2 Créer son projet

Là encore, créer son projet est une opération qui ne prend pas plus d'une minute. Par contre, il faut dans un second temps le configurer et en particulier, configurer les outils proposés. Cela se passe en cliquant sur l'icône en forme d'engrenage en haut à droite. La première chose à faire est de décrire son projet rapidement, de déclarer le dépôt public ou privé et de choisir la page sur laquelle l'utilisateur arrive lorsqu'il vient sur le projet.

On peut ensuite réaliser un certain nombre de paramétrages, en particulier dire si l'on souhaite utiliser ou non un wiki et un gestionnaire d'anomalies/d'évolutions. On peut également déclarer un certain nombre de métadonnées utiles pour ces outils, comme les versions ou les jalons (*milestones*).

Ensuite, il ne reste plus qu'à pousser son code vers son nouveau dépôt. Voici un exemple avec Git, pour lequel il suffit de paramétrer un nouveau serveur distant (appelé ici « bitbucket » et que vous pouvez appeler « origin » si vous souhaitez que ce soit celui par défaut) :

```
$ git remote add bitbucket git@bitbucket.
org:schazallet/flake8_for_gedit.git
```

Ensuite, il suffit de pousser son code, ce qui est fait ici selon les recommandations du site lui-même (tous les détails de la réponse faite par le serveur apparaissent ici) :

```
$ git push -u bitbucket --all
Password for 'git@bitbucket.org':
To https://schazallet@bitbucket.org/
schazallet/flake8_for_gedit.git
 * [new branch]      master -> master
Branch master set up to track remote branch
master from bitbucket.
$ git push -u bitbucket --tags
Password for 'git@bitbucket.org':
Everything up-to-date
```

Il ne reste plus qu'à visualiser ce qui vient d'être fait (il y avait déjà une branche *origin* et on vient d'ajouter la branche *bitbucket*) :

```
$ git branch -a
* master
  remotes/bitbucket/master
  remotes/origin/master
```

3.3 Clé publique/clé privée

Comme on peut le remarquer dans l'exemple précédent, à chaque **push**, il a fallu entrer le mot de passe, ce qui va s'avérer rapidement lassant. Ce qui est montré ici est adaptable pour GitHub,

il s'agit de la même procédure. L'idée est de remplacer l'authentification par mot de passe par une authentification par échange de clés. Voici les étapes essentielles. En premier lieu, on crée une clé publique et une clé privée :

```
$ ssh-keygen
```

On utilise ici un algorithme de chiffrement asymétrique, qui garantit que seule la clé publique correspond à la clé privée et vice versa, mais qu'il n'est pas possible de déterminer l'une à partir de l'autre. Il faut donc absolument ne **jamais** partager sa clé privée, mais on peut sans aucun risque donner sa clé publique à tout le monde, d'où leurs noms respectifs.

On peut ajouter cette clé privée à celles gérées sur son compte (sur l'ordinateur local) :

```
$ ssh-add id_rsa
```

Puis copier le contenu de sa clé publique dans le presse-papiers, afin d'aller le coller dans le formulaire de Bitbucket (ou GitHub) prévu à cet effet :

```
$ xclip -sel clip < ~/.ssh/id_rsa.pub
```

À noter que pour l'utilisation de Mercurial, il est important d'activer la compression SSH, ce qui se fait en modifiant le fichier de configuration **~/.hgrc** :

```
ssh = ssh -C
```

Il ne reste plus qu'à tester :

```
$ git pull bitbucket
Already up-to-date.
```

Comme vous pouvez le voir, le mot de passe n'est plus demandé !

3.4 Savoir utiliser les outils

Tout comme cela se fait pour GitHub, Bitbucket dispose d'un wiki et d'un outil de gestion de projet minimaliste, permettant de gérer tout à la fois : les évolutions d'anomalies et de suggestions. Les deux plateformes sont sur ces points très proches. Il n'y a donc pas grand-chose de plus à ajouter si ce n'est insister sur l'importance de la communication et sur le fait d'informer les contributeurs.

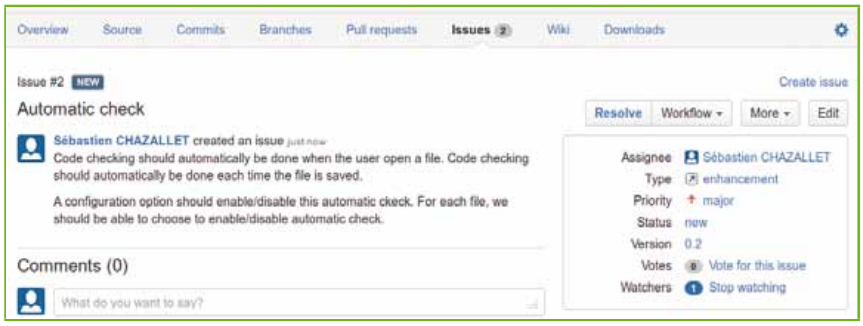


Figure 5

Pour faire du wiki une vitrine efficace, il faut apprendre à utiliser l'un de ces langages de balisage léger. Bitbucket propose quatre choix, ce qui est largement suffisant : Markdown, Creole, reStructuredText ou Textile ; ils sont tous relativement proches. Voici un pot-pourri vous permettant d'apprécier le langage Markdown :

```
# Ceci est un titre de niveau 1
## Ceci est un titre de niveau 2

Ceci est un paragraphe avec une
*accentuation* et une **accentuation
forte**.

Ceci est une liste à puces :
* point 1
* point 2
```

Il est à noter que, comme pour GitHub, le wiki est lui-même un dépôt et qu'il peut être cloné pour être modifié directement sur son poste de travail.

Le gestionnaire d'anomalies/évolutions/suggestions est très simple à utiliser et très efficace et il est lié aux commentaires présents dans les *commits*. L'outil est très orienté réseau social, avec la possibilité de laisser les anonymes poster des commentaires.

3.5 Comment organiser ses branches ?

La création d'une branche sur Bitbucket est aussi rapide et aussi simple à réaliser que ça l'est sur GitHub ; une page y est dédiée. Ce qui a été dit sur les branches dans le chapitre sur GitHub est également valable ici et de la même manière, ce qui suit est valable pour GitHub.

Voici comment informer le dépôt sur la machine locale qu'une nouvelle branche a été créée sur la plateforme distante :

```
$ git fetch
remote: Counting objects: 2, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (2/2), done.
From https://bitbucket.org/schazallet/flake8_
for_gedit
 * [new branch]      proposal   -> bitbucket/
proposal
```

On voit clairement une nouvelle branche apparaître. On peut le vérifier ainsi :

```
$ git branch -a
* master
  remotes/bitbucket/master
  remotes/bitbucket/proposal
  remotes/origin/master
```

On peut maintenant pousser dessus et on est libre ou non de créer une branche correspondante sur le dépôt local. En général, ce sera le cas, ou alors ce seront plusieurs branches locales qui pousseront sur une seule branche distante.

3.6 Comment proposer une reversion ?

Le profil utilisateur permet de visualiser l'ensemble des dépôts créés. On y voit la différence entre un dépôt initié et un dépôt dupliqué. La plateforme est également accompagnée d'outils orientés réseau social permettant de suivre/partager/être suivi. Cet aspect est primordial pour orienter sa communication et savoir évaluer l'adhésion qu'il suscite.

Au-delà de ceci, le fait de dupliquer un dépôt permet de modifier librement le code qu'il contient. Cependant, l'un

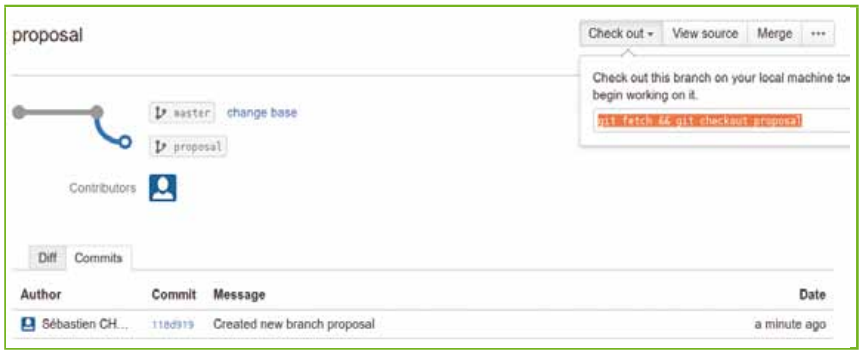


Figure 6

des aspects les plus importants consiste à **reverser** ses modifications. Proposer une reversion se fait exactement comme sur GitHub : il faut d'abord faire un *fork* du projet sur son compte Bitbucket, puis le cloner sur son poste de travail, créer le ou les commit(s) apportant la correction, pousser sur le dépôt de son compte et enfin, faire une demande de *merge*.

Ensuite, cela ne dépend plus de nous. Il faut que l'un des responsables du projet originel s'empare de votre travail et le valide ou le modifie.

4 Launchpad

4.1 Généralités

Launchpad est la plateforme mise en place par Canonical Ltd., l'éditeur d'Ubuntu, pour ses propres besoins. Elle est assez différente des deux précédentes. En effet, Launchpad n'est absolument pas conçue comme un outil orienté vers les réseaux sociaux, ni comme une vitrine pour un projet. Ces aspects-là ne sont donc (pour l'instant) pas particulièrement mis en avant.

Il s'agit plutôt d'un outil à destination des contributeurs et c'est comme cela qu'il faut le percevoir. Le contributeur est la cible des outils de la plateforme. Tout est fait pour lui assurer un haut niveau de service. En cela, il s'agit de la plateforme la plus professionnelle, mais elle paraît par la même occasion plus rigoureuse, moins facile d'accès.

Il y a une page (https://launchpad.net/+tour/index) qui résume assez bien la philosophie de Launchpad et l'esprit de la plateforme : celle-ci est un agrégat de différents outils complémentaires, tous parfaitement intégrés les uns aux autres et chacun ayant son rôle bien délimité, voire s'adressant à une qualité de contributeur particulière. On trouve par exemple un outil dédié aux traducteurs, un autre dédié aux simples utilisateurs qui ont des questions à poser, etc.

Là encore, si l'on décide d'utiliser cette plateforme, il faut avoir conscience que la gestion de la communauté est un vrai travail, très important, très visible et que cela consomme énormément de temps. Un projet qui ne répond pas à ses contributeurs est un projet qui a très peu de chances d'être adopté.

Pour résumer en caricaturant un peu, GitHub et Bitbucket s'adressent à des porteurs de projets souhaitant toucher une communauté, alors que Launchpad s'adresse à des équipes bien identifiées souhaitant toucher des contributeurs.

4.2 Notion d'équipe

La particularité de Launchpad est d'offrir un outil de gestion de projet qui est destiné à être utilisé par une équipe, laquelle est constituée de plusieurs membres identifiés et aux rôles bien définis. Les équipes sont de trois natures. Les équipes « ouvertes » acceptent tout le monde ; par contre, ce sont les membres des équipes « restreintes » qui ont toute latitude pour accepter ou non une candidature. Dans les équipes « modérées », les administrateurs ont ce rôle. Cela permet une structuration des communautés et la constitution d'équipes est importante pour des gros projets, lorsque les contributions sont nombreuses.

La plateforme a donc pour rôle de gérer les relations de travail entre membres de l'équipe, mais également entre l'équipe et les contributeurs, ou entre l'équipe et les utilisateurs. En ce qui nous concerne, dans le cadre de l'article, nous allons surtout faire un état des lieux des outils permettant de communiquer autour du projet.

4.3 Vue d'ensemble

Parmi les différentes applications composant Launchpad, on trouve :

- un service d'hébergement de code source (https://code.launchpad.net/) ;
- un gestionnaire d'anomalies (https://bugs.launchpad.net/) (Fig. 7, page suivante;
- un gestionnaire de spécifications et d'évolutions (https://blueprints.launchpad.net/) ;
- un gestionnaire de traductions (https://translations.launchpad.net/) ;
- un outil de support à destination de la communauté (https://answers.launchpad.net/) ;

Une des fonctionnalités importantes est la gestion des revues de code (https://help.launchpad.net/Code/Review). L'outil est en effet suffisamment bien calibré pour permettre de gérer de nombreux utilisateurs et sur une grande quantité de code (Fig. 8, page suivante).

On trouve également des outils permettant de gérer des listes de diffusion, par exemple, et donc d'organiser sa communication d'une manière efficace. Par contre, on ne trouve pas le fait de publier automatiquement un *tweet* lors de chaque commit, comme le proposent GitHub et Bitbucket.

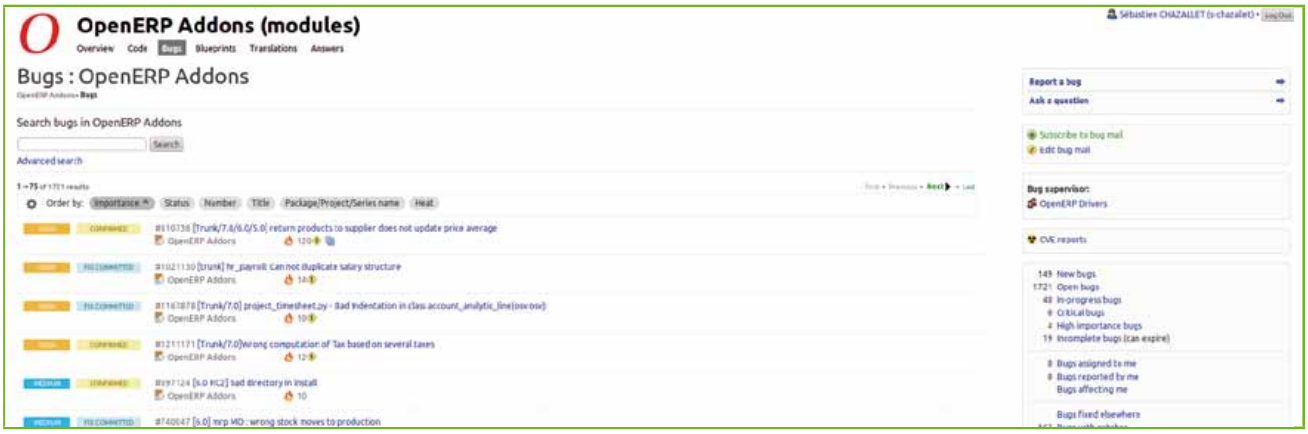


Figure 7

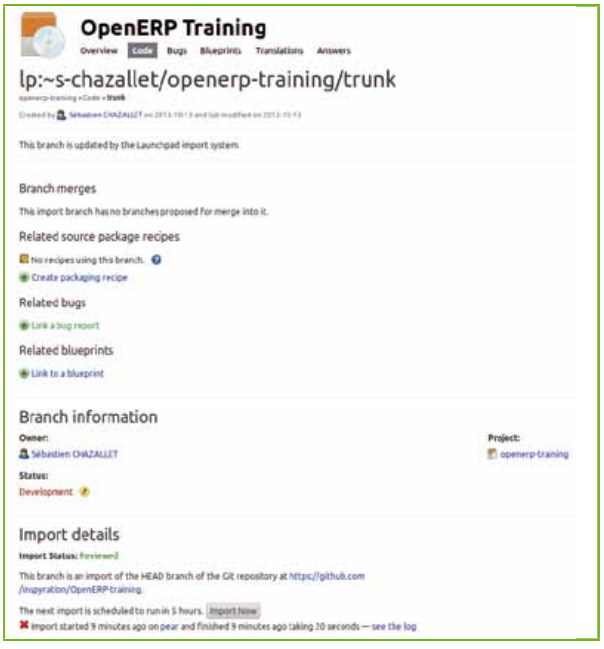


Figure 8

Conclusion

Il y a quelques années, c'était le règne des forges logicielles. Les projets étaient gérés de manière centralisée (indépendamment du gestionnaire de versions utilisé) par leurs concepteurs. Puis, les forges publiques sont apparues, permettant aux petits projets n'ayant pas les moyens de mettre une infrastructure en place d'être gérés plus facilement, et aux gros projets de se passer de l'infrastructure et de toucher une communauté encore plus large. C'est au fil du temps que la notion de communication autour du projet via les plateformes publiques a vu le jour, avec l'avènement des réseaux sociaux.

Pour bien choisir votre plateforme, il suffit simplement de se poser les questions suivantes :

- Quelle relation entretenir avec la communauté ?
- Quelle plateforme est susceptible de disposer de la plus grande communauté pour mon projet ?

Après, il reste des questions de second niveau (par rapport à l'orientation de l'article, qui porte plus sur les aspects de communication que sur les aspects techniques) :

- Quel gestionnaire de versions utiliser ?
- De quels outils ai-je besoin ?

On regrettera que GitHub ne supporte que Git et que Launchpad n'utilise que Bazaar, qui est assez différent des deux autres technologies (bien que l'on puisse importer un projet Git), les concepts de base étant vraiment différents. En effet, à ce niveau-là, le choix du gestionnaire de versions passe vraiment au premier plan et c'est l'une des raisons qui fait que Launchpad n'est pas aussi populaire que ce qu'il devrait être, malgré des outils très professionnels.

Pour le reste, si GitHub reste toujours la plateforme la plus connue, Bitbucket est clairement une alternative très séduisante, d'autant plus qu'elle est la seule des trois étudiées ici qui supporte deux gestionnaires de versions. On notera que leurs fonctionnalités sont très proches, GitHub ayant quelques petites facilités en plus (multiples formats pour le wiki, création d'un fichier **`README`**, intégration de la licence et d'un fichier **`gitignore`**).

Et puisque l'esprit de ce numéro est de parler des modèles économiques autour de l'open source, il convient de s'interroger sur la manière dont ces plateformes sont financées. Certes, il est possible de prendre des comptes payants, mais l'une des sources de revenus essentielles consiste à offrir aux chasseurs de tête un outil leur permettant de détecter des talents et de les évaluer pour pouvoir les recruter, voire plus généralement, de réaliser de l'intelligence économique, dans les limites que permettent les conditions d'utilisation, que vous avez bien entendu lues attentivement. ■

LA DOCUMENTATION DE VOTRE PROJET AVEC DOXYGEN

par Tristan Colombo

La documentation technique d'un code est essentielle pour sa maintenance et sa capacité à évoluer. Cela est vrai pour n'importe quel code, mais plus particulièrement pour les programmes distribués sous licence libre.

Lorsque vous écrivez un programme, vous savez précisément ce que vous faites, pourquoi vous employez tel ou tel algorithme, pourquoi vous avez découpé le code en *n* classes, etc. Mais toutes ces notions qui vous paraissent évidentes au moment où vous écrivez se révéleront incompréhensibles quelques semaines plus tard ! Et si vous, le concepteur du code, n'êtes même plus capable de comprendre ce que vous avez écrit, qu'en sera-t-il d'un autre développeur ?

Dans la philosophie générale du logiciel libre, vous mettez à disposition du code qui doit pouvoir être lu, corrigé, amélioré et éventuellement servir de « modèle » pour comprendre comment réaliser une opération particulière. Dans ma vision des choses, si vous ne documentez pas votre code, vous ne faites pas du logiciel libre, mais vous surfez sur la vague open source. Un peu comme si vous vendiez des légumes bio tout en consommant des produits transgéniques et en ne recyclant pas vos déchets. Vous utilisez le fait de rendre le code accessible comme argument marketing, tout en sachant pertinemment que les utilisateurs devront se tourner vers vous pour toute correction. Le code est certes disponible, mais pratiquement inutilisable à moins de consentir à un gros effort d'analyse et... de documentation !

Un générateur automatique de documentation de code ne peut pas inventer la documentation : il fonctionne en analysant les commentaires des fichiers et en y repérant des structures particulières. Il existe de nombreux logiciels effectuant cette tâche, certains étant plus orientés pour un usage sur un langage particulier alors que d'autres sont plus généralistes. Par exemple, **Sphinx** est adapté à Python et **Javadoc** cible bien sûr le langage Java. Je vais vous présenter ici un générateur généraliste, **Doxygen [1]**, qui est distribué sous licence GNU GPL.

Doxygen supporte les langages C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP, C#, et D. Bien sûr, comme dit précédemment, il existe d'autres logiciels mieux adaptés à certains langages. Il est par exemple absurde d'utiliser Doxygen pour documenter du code Python, même si c'est possible. Par contre, pour Java, même si Javadoc lui est dédié, Doxygen générera une documentation beaucoup plus élégante et dans de nombreux formats : HTML, LaTeX, PDF, PostScript, man, RTF et XML. Donc, pour résumer, si vous ne développez pas en Python, il y a de fortes chances pour que Doxygen réponde parfaitement à vos besoins.

Nous allons voir comment installer Doxygen, quels sont les outils mis à notre disposition et comment écrire les commentaires d'une application en vue de générer sa documentation. Pour finir, nous verrons comment configurer finement Doxygen et comment personnaliser la documentation générée.

1 Installation

Comme toujours, l'installation se fait de manière très simple sur les distributions basées sur Debian :

```
$ sudo aptitude install doxygen doxygen-gui
graphviz
```

Comme vous pouvez le constater, nous n'installons pas que le paquetage **doxygen** ; nous installons également **doxygen-gui** qui permet de disposer de l'outil **doxywizard**, une application graphique de configuration de Doxygen qui simplifie considérablement son utilisation, et **graphviz** qui permettra à Doxygen de générer des graphes des relations d'héritage entre les classes.

Les utilisateurs d'autres distributions devront soit récupérer les sources en suivant les liens de la section téléchargement **[2]** et les compiler en suivant une procédure classique (assurez-vous d'avoir installé les logiciels suivants : flex, bison, make, strip, perl, Qt, LaTeX, graphviz, ghostscript et python), soit utiliser **git** (et compiler également) :

```
$ git clone https://github.com/doxygen/doxygen.git
$ cd doxygen
$ ./configure
$ make
$ make install
```

Vous devez maintenant avoir une version fonctionnelle de Doxygen. Avant de l'utiliser, je vous propose d'analyser son fonctionnement.

2 Fonctionnement de Doxygen

Doxygen est en fait le regroupement de trois outils :

- L'application principale, nommée **doxygen**, qui permet d'analyser les codes sources et de générer la documentation dans le format pré-sélectionné ;
- Une interface graphique, nommée **doxywizard**, qui permet de configurer et d'exécuter Doxygen ;
- Une application d'extraction d'informations, **doxytag**, fonctionnant en mode console. Cet outil permet d'inclure dans votre documentation la documentation d'autres fichiers, pour lesquels vous n'avez pas accès au code source mais seulement aux pages HTML générées par Doxygen.

Le schéma de la figure 1 illustre le fonctionnement simplifié de Doxygen : **doxywizard** permet de générer le fichier de configuration et de lancer **doxygen** (cette opération peut être réalisée en mode console), **doxytag** crée un fichier de tags issus d'une documentation et enfin, **doxygen** lit le code source et les différents fichiers de configuration et de tags pour générer la documentation dans le format choisi par l'utilisateur. Pour un schéma complet, voir la documentation du site officiel **[3]**.

Voyons maintenant comment commenter un code source de manière à ce que Doxygen soit capable de générer la documentation technique.

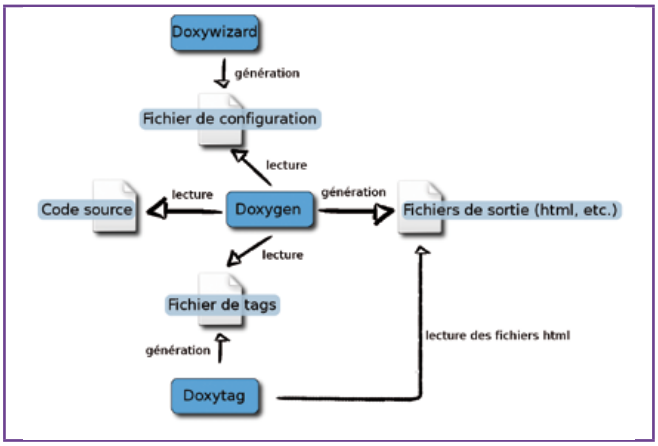


Fig. 1 : Principe de fonctionnement simplifié de Doxygen

3 Commenter une application et la documenter

Dans cette partie, nous allons avoir besoin d'un petit code d'exemple et il va donc falloir choisir un langage. J'ai choisi d'écrire ce code en Java, mais je vous rappelle que vous pouvez faire strictement la même chose avec d'autres langages (sauf Python si vous voulez respecter les conventions de codage du PEP257).

Avant d'écrire le code, il faut savoir comment écrire les commentaires pour que Doxygen soit capable de les interpréter. Ces commentaires doivent être compatibles avec les tags Javadoc **[4]** (auxquels ont été ajoutés quelques tags spécifiques à Doxygen). Les tags Javadoc sont des sortes de commandes permettant de classifier les informations contenues dans les commentaires du code. Ces tags doivent être utilisés dans des blocs de commentaires débutant généralement par **/**** et finissant par ***/** (bien entendu, si le langage cible utilise d'autres marqueurs de commentaires, il faudra utiliser ceux-là).

Doxygen va détecter les blocs de commentaires, les analyser et inclure leurs informations dans la documentation. Les blocs de commentaires qui comportent plusieurs lignes sont généralement écrits en utilisant le caractère * en début de ligne. Voici un exemple de bloc de code qui sera analysé par Doxygen :

```
/**
 * Bloc de commentaires
 * sur plusieurs lignes.
 */
```

Dans ces blocs, nous pourrons utiliser les tags définis par Javadoc. Les blocs seront toujours analysés de la même manière :

- Les phrases de commentaires placées au début du bloc représentent la description de la méthode, classe, fonction, etc.
- Dans la suite, les tags Javadoc seront utilisés en les faisant précéder du caractère **@** (un commentaire Javadoc s'écrit sous la forme **@tag_javadoc commentaire**). Doxygen admet une autre écriture où les tags sont précédés du caractère \. Vous pouvez donc utiliser indifféremment l'un ou l'autre, mais comme Javadoc était le précurseur, autant se conformer à la notation de départ.

Voici un exemple illustrant la structure d'un bloc de commentaire :

```
/**
 * Description de la méthode
 * sur plusieurs lignes.
 *
 * @tag_javadoc_1 commentaire
 * ...
 * @tag_javadoc_n commentaire
 */
```

Tag Doxygen	Informations associées	Nature
`@brief`	description	Description brève du comportement d'une méthode, classe, etc. (à ne pas confondre avec la description longue décrite précédemment). Cette description s'arrête à la première ligne de commentaire vide et elle apparaîtra dans la partie de résumé et dans la partie de description détaillée de la documentation. `@brief Description rapide de la méthode.`
`@author`	nom	Nom de l'auteur de la méthode, classe, etc. (une ligne par auteur). `@author <a href="mail@mail.fr">Tristan Colombo</a>`
`@version`	numéro	Version de l'application (peu utilisé pour les méthodes). `@version 1.0.2`
`@since`	numéro	Indique le numéro de la version où le code est apparu. `@since 1.0.1`
`@date`	date	Spécifie une date qui peut être la date de création ou de modification par exemple. `@date 15/01/2010`
`@param`	type nom description	Indique le type, le nom et la description d'un paramètre d'une méthode. `@param int rayon Rayon du cercle en mm`
`@return`	type description	Indique le type et la description de la valeur de retour d'une méthode `@return double Aire du disque en mm`
`@include`	nom de fichier	Insère le code d'un fichier externe contenant un exemple d'exécution du code. Attention : le répertoire dans lequel sont stockés les fichiers doit être défini dans le fichier de configuration de Doxygen (voir les explications dans la suite de l'article dans la section « Configuration avancée »). `@include display.php`

`@code`
`@endcode`
	code d'exemple	Insère quelques lignes de code d'exemple.
		`@code`
		`  Cercle r = new Cercle(5);`
		`  System.out.println(r.getRayon());`
		`@encode`

`@exception`	type description	Description du traitement d'une exception. `@exception ArithmeticException Sortie du programme avec le code -1.`
`@package`	nom	Indique un nom de paquetage. `@package tools`
`@attention`	message	Message important à mettre en relief. `@attention Ceci est une fonction de test.`
`@bug`	description	Indique la présence d'un bug. `@bug Impossible de saisir un entier.`
`@todo`	description	Liste d'actions restant à réaliser. `@todo Ajouter contrôle des entrées.`

Notez qu'il est possible d'utiliser des tags HTML dans vos commentaires. Ces derniers seront pris en compte lors de la génération de la documentation au format HTML.

Voyons maintenant quelques tags disponibles parmi les plus utilisés et leur signification. Pour une liste complète des tags disponibles dans Doxygen, vous pourrez consulter la page qui leur est dédiée sur le site officiel **[5]**. Le tableau, page précédente, contient un ensemble de tags, pour chacun d'eux les informations qui doivent lui être associées, la nature de ces informations et un exemple d'utilisation dans un commentaire.

Bien sûr, si vous avez la curiosité de vous rendre sur le site de Doxygen, vous constaterez qu'il existe beaucoup plus de tags et qu'il est même possible de créer des structures conditionnelles ! Cette structure étant un peu plus complexe, j'ai préféré l'isoler du tableau précédent.

Les instructions utilisées dans les structures conditionnelles sont **@if**, **@ifnot**, **@else**, **@elseif** et **@endif**. La condition qui sera testée sera la présence ou l'absence d'une variable définie dans le fichier de configuration (nous verrons cela dans la partie suivante). Pour créer des conditions complexes, vous pourrez utiliser les opérateurs logiques **&&** (et), **||** (ou) et **!** (non), tout en prenant garde de bien encadrer votre condition par des parenthèses.

Voici un exemple de commentaire conditionnel où une partie de la documentation ne sera générée que si l'on se trouve en mode **debug** (variable du fichier de configuration) :

```
/**
 * Commentaire standard valable tout le temps
 *
 * @if debug
 *   Ne s'affiche que si debug est défini
 * @endif
 */
```

À mon avis, il est possible de correctement documenter un projet en se limitant aux tags vus précédemment, de manière à conserver un bon compromis entre le temps passé à coder et le temps passé à documenter le code.

Je vous propose maintenant d'appliquer quelques-uns des tags que nous avons vus à une classe Java contenue dans un fichier **A.java** (les lignes de commentaires apparaissent en rouge) :

```
01: /**
02:  * @brief Classe de test A.
03:  * Cette classe permet de tester la génération de doc avec Doxygen.
04:  *
05:  * @author <a href="mailto:toto@toto.com">Toto</a>
06:  * @author <a href="mailto:titi@toto.com">Titi</a>
07:  *
08:  * @date 22/08/2013
09:  * @version 0.0.2
10:  */
11: public class A
12: {
13:     /**
14:      * @brief Attribut <b>var</b> permettant de stocker un entier.
15:      */
16:     private int var;
17:
18:    /**
19:     * @brief Accesseur de l'attribut var.
20:     *
21:     * Affecte une valeur à l'attribut.
22:     *
23:     * @author <a href="mailto:toto@toto.com">Toto</a>
24:     * @date   21/08/2013
25:     *
26:     * @since 0.0.1
27:     *
28:     * Exemple d'utilisation :
29:     * @code
30:     *   A v =new A(5);
31:     *   System.out.println(v.getValeur());
32:     * @endcode
33:     *
34:     * @return  int  La valeur de l'attribut var.
35:     */
36:     public int getVar()
37:     {
38:       return this.var;
39:     }
40:
41:     /**
42:      * @brief Modifieur de l'attribut var.
43:      *
44:      * Modifie la valeur de l'attribut.
45:      *
46:      * @author <a href="mailto:titi@toto.com">Titi</a>
47:      * @date   21/08/2013
48:      *
49:      * @param  int   <b>value</b>   La valeur à affecter à var.
50:      *
51:      * @return void
52:      */
53:     public void setVar(int value)
54:     {
55:       this.var = value;
56:     }
57:
58:     /**
59:      * @brief Constructeur de la classe A.
60:      *
61:      * @author <a href="mailto:toto@toto.com">Toto</a>
62:      * @date   21/08/2013
63:      *
64:      * @param  int   <b>value</b>   La valeur à affecter à l'attribut var.
65:      *
66:      * @if debug
67:      *   @todo Penser à ajouter un attribut valeur2 et à l'initialiser.
68:      *
69:      *   @bug  L'absence de contrôle de la valeur provoque un bug.
70:      * @endif
71:      */
72:     public A(value)
73:     {
74:       this.setVar(value);
75:     }
76:
77:   }
```

Pour cet exemple, nous allons utiliser l'interface graphique Doxywizard. Pour cela, ouvrez une console et tapez :

```
$ doxywizard
```

Une nouvelle fenêtre va apparaître (voir figure 2). Vous devrez commencer par indiquer le répertoire à partir duquel vous souhaitez exécuter Doxygen (zone 1 sur la figure). Ce répertoire sera le répertoire contenant le code source de votre projet.

Vous pouvez distinguer ensuite une zone contenant trois onglets (zone 2) : ***Wizard***, ***Expert*** et ***Run***. L'onglet ***Wizard*** est sélectionné par défaut et permet une configuration allégée de Doxygen. L'onglet ***Expert*** permet de modifier manuellement toutes les valeurs du fichier de configuration et l'onglet ***Run***, comme son nom l'indique, permet de lancer la génération de la documentation. Chaque onglet ***Wizard*** et ***Expert*** comporte un certain nombre de pages de configuration qui sont signalées dans la zone ***Topics***.

Nous allons ici utiliser la configuration simplifiée (***Wizard***) et nous allons donc avoir accès aux pages ***Project***, ***Mode***, ***Output*** et ***Diagrams***. La première page, la page ***Project***, est déjà affichée. Elle permet de configurer le nom, la version et le logo du projet (zone 3), le répertoire contenant le code des sources et s'il doit être scanné récursivement (zone 4) et enfin, le répertoire dans lequel sera placée la documentation générée (zone 5). Pour ce dernier point, je vous conseille de créer un répertoire **doc** dans le répertoire contenant le code source.

Une fois ces informations renseignées, pour passer à la page suivante, cliquez sur le bouton ***Next*** en bas à droite de la fenêtre, ou cliquez sur le label ***Mode*** dans la zone ***Topics***. Cette nouvelle page permet d'indiquer si l'on souhaite générer la documentation uniquement pour les éléments commentés ou pour tous les éléments. Nous choisirons ici tous les éléments, mais cela n'aura aucune incidence puisque notre seul fichier est commenté. Cette page permet aussi de sélectionner le langage du code source pour un meilleur résultat. Notre code étant en Java, nous sélectionnerons ***Optimize for Java or C# output***.

La page suivante est la page ***Output*** et elle permet de sélectionner le format de sortie de la documentation générée. Comme dit précédemment, vous pouvez générer des fichiers HTML avec ou sans menu de navigation, des fichiers PDF, etc. Nous laisserons les valeurs par défaut, en ajoutant seulement le menu de navigation.

Enfin, la dernière page, la page ***Diagrams*** permet de paramétrer la génération des diagrammes. Vous pouvez choisir de désactiver les diagrammes, d'utiliser le générateur intégré à Doxygen ou enfin, d'utiliser GraphViz. Je vous recommande fortement la dernière option qui permet de générer plus de diagrammes et de les configurer plus finement (vous pouvez notamment générer le graphe des appels de chaque méthode avec les options ***Call graphs*** et ***Called by graphs***).

L'étape de configuration est maintenant achevée. Vous avez la possibilité de sauvegarder le fichier de configuration ainsi réalisé en vous rendant dans le menu ***File > Save***. Ce fichier, nommé par défaut **Doxyfile**, pourra ainsi être ré-utilisé après modification du code de ce projet pour mettre à jour la documentation (en relançant **doxyfile** et en chargeant le fichier de configuration dans ***File > Open***) .

Pour générer la documentation, il suffit maintenant d'aller sur l'onglet ***Run*** et de cliquer sur le bouton ***Run doxygen***. Les fichiers constituant la documentation se trouvent alors dans le répertoire indiqué précédemment et ils sont classés par format (un répertoire **html** pour la documentation HTML, etc.). En cliquant sur le bouton ***Show HTML output***, votre navigateur s'ouvrira sur la documentation générée. Dans un premier temps, vous serez peut-être déçu en ouvrant la page d'index de la documentation dans votre navigateur. En effet, cette dernière est assez... vide. Mais en naviguant, vous vous rendrez vite compte que de précieuses informations sont disponibles sur la structuration du code. Notamment, si vous consultez l'onglet ***Classes*** puis la classe ***A*** dans le menu de gauche, vous obtiendrez une page identique à celle présentée en figure 3.

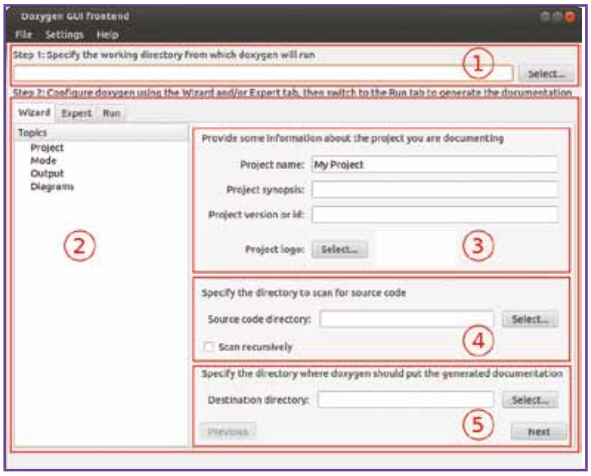


Fig. 2 : L'interface graphique de configuration Doxywizard

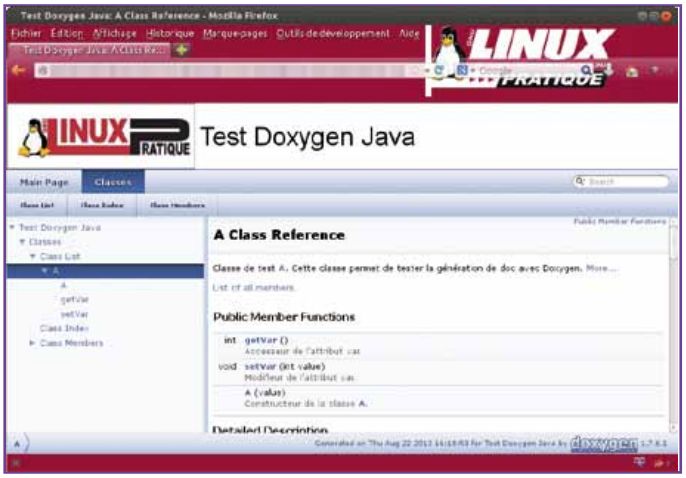


Fig. 3 : Documentation générée pour la classe A

En imaginant qu'une classe B hérite de A, nous aurions vu apparaître le diagramme des relations entre ces classes comme le montre la figure 4. Notez également qu'il est possible de cliquer sur tous les diagrammes pour accéder directement aux objets qui y sont représentés.

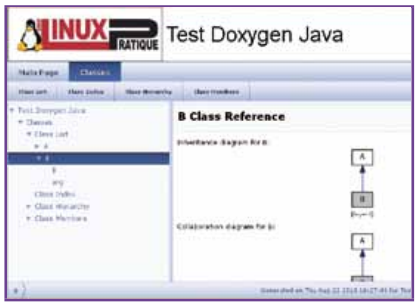


Fig. 4 : Diagramme généré pour les classes A et B

Le commentaire conditionnel contenant les tags **@todo** et **@bug** n'est pas visible... Et c'est bien normal, puisque la variable **debug** n'a pas été définie ! Pour la définir, il va falloir modifier le fichier de configuration.

4 Configuration avancée

Nous avons vu comment générer simplement la documentation technique d'un projet, mais certains éléments comme la définition de variables ne sont accessibles que dans la configuration avancée. Le fichier de configuration (**Doxyfile** par défaut) contient toutes les informations sur la manière de générer la documentation technique d'un projet donné. Il s'agit d'un fichier texte qui initialise des variables de configuration comme le montre l'extrait suivant :

```
# Doxyfile 1.7.6.1
...
# This tag specifies the encoding used for all
characters in the config file that follow.
DOXYFILE_ENCODING      = UTF-8

# The PROJECT_NAME tag is a single word (or a
sequence of words surrounded
# by quotes) that should identify the project.

PROJECT_NAME           = "Test Doxygen Java"
# The PROJECT_NUMBER tag can be used to enter a
project or revision number.
# This could be handy for archiving the
generated documentation or
# if some version control system is used.

PROJECT_NUMBER         =
# ...
```

Vous retrouvez dans ces lignes les éléments que vous avez configurés dans l'interface graphique. Par exemple, le nom du projet est stocké dans la variable **PROJECT_NAME**. Si vous parcourez entièrement le fichier **Doxyfile** (quelques 1800 lignes), vous vous rendrez compte qu'il existe un très grand nombre de variables de configuration.

Pour renseigner ces variables, le plus simple est encore d'utiliser l'application **Doxywizard** et de sélectionner l'onglet ***Expert***. Cet onglet contient toutes les variables organisées en thèmes (***Topics*** en anglais). En passant la souris sur le nom d'une variable, vous verrez apparaître une explication détaillée sur son utilité. Devant la quantité de variables disponibles, je vous propose un petit récapitulatif des variables qui me paraissent les plus importantes : voir tableau ci-dessous.

Topic	Variable	Définition
Project	**OUTPUT_LANGUAGE**	Permet de définir la langue utilisée pour générer la documentation (les noms d'onglets, les titres, etc., seront alors traduits).
Project	**ALIASES**	Permet de définir des commandes Javadoc sous la forme **variable=valeur**. Ces commandes sont comprises par Doxygen et créent une nouvelle entrée dans la documentation. Par exemple, pour créer une entrée décrivant des effets de bord (à éviter !) : **@sideeffects=\par Effets de bord :\n**
Build	**EXTRACT_PRIVATE**	Inclut les attributs privés dans la documentation (sinon ils sont omis !).
Build	**EXTRACT_STATIC**	Inclut les attributs statiques dans la documentation.
Build	**ENABLED_SECTIONS**	Permet de définir les variables conditionnelles. Ajoutez simplement **debug** dans cette partie et les commentaires **@todo** et **@bug** apparaîtront et vous pourrez noter la présence dans le menu (section ***Pages associées***) des liens ***Listes des choses à faire*** et ***Liste des bugs associés*** (si vous avez choisi « french » pour la variable **OUTPUT_LANGUAGE**, sinon ce sera bien sûr en anglais).
Input	**EXAMPLE_PATH**	Définit le ou les chemin(s) de recherche pour l'inclusion de code depuis des fichiers externes.
Source Browser	**SOURCE_BROWSER**	Pour chaque méthode, affiche un lien vers le code source (affichable dans la documentation). Exemple de documentation (les liens sont soulignés) : **Définition à la ligne 75 du fichier A.java.**
Html	**HTML_HEADER**	Définit le chemin d'un fichier HTML pour modifier l'en-tête de la documentation (voir dans la suite pour plus d'explications).
Html	**HTML_FOOTER**	Définit le chemin d'un fichier HTML pour modifier le pied de page de la documentation.
Html	**HTML_STYLESHEET**	Définit le chemin d'un fichier CSS pour modifier le style de la documentation.

Sans connaissance de ces options, vous pourriez par exemple vous demander pourquoi vos attributs ayant une visibilité privée n'apparaissent pas dans la documentation... Pourtant, il s'agit d'un comportement tout à fait normal et prévisible.

Pour modifier l'apparence de la documentation générée, il faut utiliser les trois options du thème « Html » présentées ci-dessus. Pour changer l'en-tête (option **HTML_HEADER**), il va falloir créer un fichier HTML comportant la déclaration du type de document, la balise ouvrante **<html>** suivie du bloc d'en-tête **<header>...</header>** et enfin, la balise ouvrante du corps du document **<body>**.

Dans ce fichier, vous pourrez utiliser des noms de variables qui seront remplacés par Doxygen lors de la génération du document. Ces variables sont :

- **$title** : titre de la page ;
- **$datetime** : date et heure de génération de la documentation ;
- **$date** : date de génération de la documentation ;
- **$year** : année de génération de la documentation ;
- **$doxygenversion** : version de Doxygen utilisée pour générer la documentation ;
- **$projectname** : nom du projet (configuré dans Doxywizard ou dans la variable **PROJECT_NAME**) ;
- **$projectnumber** : numéro du projet (configuré dans Doxywizard ou dans la variable **PROJECT_NUMBER**).

Voici un exemple de fichier d'en-tête :

```
01: <!doctype html>
02:
03: <html lang="fr">
04:
05:   <head>
06:     <title>Documentation du projet
$projectname ($projectnumber)</title>
07:     <meta charset="utf-8" />
08:     <link rel="stylesheet" href="doxygen.
css" />
09:     <link rel="stylesheet" href="tabs.css" />
10:   </head>
11:
12:   <body>
```

L'utilisation des variables Doxygen en ligne 6 a été mise en évidence. Notez également les lignes 8 et 9 qui chargent les feuilles de styles **doxygen.css** et **tabs.css** : comme nous redéfinissons l'en-tête des pages, il faut spécifier les feuilles de styles de Doxygen pour ne pas perdre la présentation par défaut. La feuille **tabs.css** permet de définir le style des onglets de navigation et la feuille **doxygen.css** permet de définir le style général de la documentation (présentation des entités, coloration du code source, etc.).

La variable de configuration **HTML_STYLESHEET** permet de spécifier le chemin vers une feuille de styles qui sera utilisée à la place de la feuille **doxygen.css** (attention, si vous avez redéfini l'en-tête, cette feuille ne sera pas chargée). En utilisant **HTML_STYLESHEET**, vous ne pourrez donc pas modifier le style des onglets... Si vous souhaitez vraiment redéfinir le style de la documentation générée, la meilleure solution reste de définir deux nouvelles feuilles de styles et de les charger en utilisant un fichier d'en-tête personnalisé.

La modification du pied de page (variable **HTML_FOOTER**) se fera de la même manière à partir d'un fichier HTML. Ce fichier devra contenir au minimum les balises fermantes **</body>** et **</html>**. Les variables Doxygen disponibles sont les mêmes que celles utilisées pour l'en-tête, comme le montre l'exemple suivant :

```
01:     <div style="float : bottom; font-style :
italic">
02:       $datetime : ma documentation générée
avec Doxygen $doxygenversion
03:     </div>
04:
05:   </body>
06:
07: </html>
```

Nous avons défini ici un pied de page, qui apparaîtra vraiment en bas de la page comme l'indiquent les directives de style de la ligne 1 (il serait toutefois préférable d'utiliser une classe dans un fichier de styles).

Pour achever cette partie sur la configuration avancée de Doxygen, je tiens à attirer votre attention sur l'intérêt du fichier **Doxyfile**. Il est bien sûr très important de l'enregistrer pour ne pas avoir à redéfinir toute la configuration à chaque fois que vous voudrez générer la documentation de votre projet... Mais ce fichier peut vous permettre d'automatiser la génération de la documentation ! En effet, si vous utilisez un gestionnaire de versions concurrentes tel que **Git**, vous pouvez définir un *hook post-commit* qui déclenchera la génération de la documentation à chaque modification du code ! Perdre du temps pour en gagner...

Conclusion

Nous avons vu comment, grâce à Doxygen, générer la documentation technique d'un projet et améliorer la présentation de cette documentation.

Commenter un code n'est pas une activité inutile, cela permet d'aider les autres développeurs à atteindre un objectif commun : la correction ou l'amélioration du code du projet. La documentation du code participe à la qualité d'un projet et à la création d'une communauté qui pourra se fédérer autour de lui. Tout comme pour le code, la documentation peut être reprise et améliorée, mais a minima, il faut qu'elle existe et qu'elle ne soit pas de trop mauvaise qualité (si c'est pour dire que la ligne **a + b** réalise l'addition de deux variables, il vaut mieux ne rien écrire...).

Ne considérez donc pas que la mise en place, la configuration d'un logiciel de documentation et l'écriture des commentaires est une perte de temps : c'est un gain de temps pour le futur ! ■

Références

[1] Site officiel de Doxygen : http://www.doxygen.org/

[2] Page de téléchargement des paquetages RPM de Doxygen : http://www.stack.nl/~dimitri/doxygen/download.html

[3] Schéma de fonctionnement de Doxygen : http://www.stack.nl/~dimitri/doxygen/manual/infoflow.png

[4] Site officiel de Javadoc : http://www.oracle.com/technetwork/java/javase/documentation

[5] Liste des tags Doxygen : http://www.stack.nl/~dimitri/doxygen/manual/commands.html

DOKUWIKI : UN WIKI SIMPLE ET ÉVOLUTIF

par Yann Morère

Un wiki c'est très pratique. Pour un groupe de travail, cela permet de construire de manière collaborative une base de connaissances accessible à tous. Mais un wiki, cela peut aussi servir individuellement. C'est très utile et rapide comme bloc-notes par exemple. Découvrons ensemble DokuWiki, un wiki simple et polyvalent.

1 Introduction

Mais avant de découvrir DokuWiki, rappelons ce qu'est un wiki et comment il fonctionne. Voici sa définition sur Wikipédia **[1]** : « Un wiki est un site web dont les pages sont modifiables par les visiteurs, ce qui permet l'écriture et l'illustration collaboratives des documents numériques qu'il contient. Il utilise un langage de balisage et son contenu est modifiable au moyen d'un navigateur web ». Le premier wiki date de 1995 et a été créé par Ward Cunningham. Le plus connu des wikis est bien sûr Wikipédia, le projet d'encyclopédie universelle et multilingue. Wiki vient du mot hawaïen signifiant rapide ; le redoublement du mot signifie « très vite / très rapide ». On remarquera que l'acronyme peut aussi signifier « What I Know Is » (« voici ce que je sais »).

Les wikis (et ils sont très nombreux) font partie d'une catégorie de logiciels destinés au Web : les CMS (*Content Management System*) ou SGC en français (Système de Gestion de Contenu). La plupart d'entre eux permettent la conception et la mise à jour dynamique de sites web ou d'applications multimédias. Ils possèdent des fonctionnalités communes : travail collaboratif, chaîne de publication, structuration de contenu, séparation de la forme et du contenu (par l'intermédiaire de thèmes/canevas et le stockage des informations dans une base de données pour la plupart d'entre eux), hiérarchisation des utilisateurs, gestion de versions des documents.

Mais il existe plusieurs types de CMS. Le choix dépendra de la finalité d'exploitation et de vos compétences : blog, base de connaissances, journal en ligne, site vitrine, site personnel, e-commerce, multilingue, etc. Les plus connus des CMS actuels sont Joomla, Drupal, eZ Publish, WordPress, Dotclear **[2]** et pour les wikis, Wikimedia.

DokuWiki est donc un système de gestion de contenu, qui va nous permettre de mettre à jour rapidement nos pages, d'en créer simplement de nouvelles. Il a été créé en 2004 par Andreas Gohr et utilise le langage PHP ; il est publié sous la licence GNU GPL v. 2.01.

C'est un outil très adapté si vous désirez créer une base de connaissances collective, l'utiliser comme un carnet de notes personnelles, réaliser la documentation d'un projet ou d'un logiciel, ou encore en faire le CMS de votre Intranet.

Il existe de nombreux wikis (plus de 100 !). Le site WikiMatrix **[3]**, vous permet de comparer les fonctionnalités de Mediawiki et DokuWiki par exemple. Vous constaterez ainsi qu'il fait partie des wikis les plus utilisés et qu'il possède des fonctionnalités supplémentaires utiles comme les ACL (*Access Control List*) permettant la gestion fine des utilisateurs et de leurs rôles, ainsi que leurs droits d'accès. Nous expliquerons ceci en détail dans la suite de l'article.

2 Fonctionnalités de DokuWiki

Un de ses principaux avantages, mise à part sa facilité de maintenance, de sauvegarde et d'intégration, est qu'il n'utilise pas de base de données pour le stockage du contenu (contrairement à Mediawiki par exemple), mais de simples fichiers textes stockés dans une arborescence de répertoires reflétant la structure arborescente mise en place dans votre wiki. Ceci le rend plus rapide que ses homologues utilisant une base de données. Cela évite aussi d'installer une serveur de base de données sur la machine. On peut alors se contenter d'un serveur LAP à la place de LAMP (Linux Apache MySQL PHP). Les fonctionnalités de base sont les suivantes :

- une syntaxe simple ;
- une gestion de révisions de pages illimitées ;
- la visualisation des derniers changements sur les pages ;
- un analyseur de différences entre les révisions de pages ;

- le téléversement et l'insertion d'images et de fichiers (avec la gestion des types de fichiers autorisés) ;
- l'organisation possible du contenu en espaces de noms (répertoires), que l'on peut ensuite parcourir via un index automatique.

Dans son ergonomie d'utilisation, il possède des fonctionnalités bien utiles :

- On peut éditer les sections de page pour modifier seulement une petite partie d'une page. Ceci est très pratique quand on possède de longues pages contenant du code source par exemple ;
- Il possède une navigation de type « fil d'Ariane » ;
- Une table des matières est générée automatiquement ;
- Les pages sont verrouillées lors de l'édition pour éviter les conflits.

On retrouvera dans ce wiki la possibilité de contrôle anti-spam, le support multilingue, de l'UTF-8. La fonction de recherche est accélérée par l'utilisation de recherche *plain text* (rendu possible par l'utilisation de fichiers textes pour le stockage).

Comme pour ses homologues, les fonctionnalités peuvent être étendues par l'intermédiaire de greffons réalisés par la communauté d'utilisateurs. Il faudra cependant faire attention aux problèmes de sécurité potentiels apportés par l'utilisation de greffons anciens ou non mis à jour. L'utilisation des thèmes vous permettra de modifier l'apparence de votre wiki très simplement.

On retrouvera bien sûr la syndication par l'intermédiaire de flux RSS et Atom.

En ce qui concerne l'authentification, il est possible de coupler DokuWiki à un LDAP, Active Directory ou encore une bases de données MySQL, PostgreSQL, etc.

Passons maintenant à la phase d'installation.

3 Installation

La page **[4]** présente les prérequis nécessaires pour l'utilisation/utilisation de DokuWiki. Dans notre cas, nous utiliserons le serveur web Apache avec le support de PHP, mais il est possible d'utiliser d'autres serveurs comme Lighttpd ou Nginx. Pour la gestion des images, nous aurons besoin de l'extension GD, ou encore d'ImageMagick.

L'installation d'un serveur LAP est très simple avec Ubuntu :

```
$ sudo apt-get install apache2 php5 libapache2-mod-php5 php-gd imagemagick
```

Sous Ubuntu, le fonctionnement est immédiat, l'intégration et la configuration de PHP pour Apache est automatique. Avec Arch Linux, cela se déroule en 2 étapes. D'abord l'installation :

```
$ sudo pacman -S apache php php-apache php-gd imagemagick
```

Puis la phase de configuration qui consiste à configurer le serveur web Apache pour l'utilisation de PHP. Pour cela, dans le fichier **/etc/httpd/conf/httpd.conf**, il faut ajouter dans la section « LoadModule » les lignes

```
# Load php5 module
LoadModule php5_module modules/libphp5.so
```

puis dans la section des « Include » (vers la fin du fichier) les lignes :

```
# PHP settings
Include conf/extra/php5_module.conf
```

Ensuite, on démarre le serveur Apache pour la session courante et pour les prochains démarrages :

```
$ sudo systemctl start httpd
$ sudo systemctl enable httpd
```

Vérifions le bon fonctionnement à l'aide de la simple page **test.php** que l'on déposera dans le répertoire **/srv/http/** dans le cas d'Arch Linux et **/var/www** dans le cas d'Ubuntu.

```
<?php
  phpinfo ();
?>
```

Puis, dans votre navigateur, chargez le lien http://localhost/test.php ; si tout s'est bien passé, vous obtenez la page détaillant les options de PHP utilisées.

Avant de poursuivre, pensons à la sécurité de notre futur wiki web. La page **[5]** nous donne les informations nécessaires à la sécurisation de notre installation de DokuWiki. Dans un premier temps, il faudra configurer Apache afin qu'il utilise les fichiers **.htaccess** contenus dans l'arborescence de DokuWiki. Ces derniers ordonnent au serveur Apache de ne pas autoriser l'accès à certains répertoires. En effet, comme DokuWiki sauvegarde sa configuration et ses pages dans des fichiers, ces derniers ne doivent pas être accessibles directement depuis le Web.

Pour cela, il faut éditer le fichier **/etc/httpd/conf/httpd.conf** comme suit pour activer la directive **AllowOverride** :

```
   # AllowOverride controls what directives may be placed in
.htaccess files.
    # It can be "All", "None", or any combination of the keywords:
    #   Options FileInfo AuthConfig Limit
    #
    AllowOverride All
```

Puis, interdisons la possibilité de lecture du contenu des répertoires. On poursuit l'édition du fichier **httpd.conf** et l'on commente la ligne suivante dans la directive **<Directory "/srv/http">** :

```
# Options Indexes FollowSymLinks
```

On redémarre ensuite le serveur Apache. Ces modifications éviteront que la page d'installation de DokuWiki affiche un problème de sécurité.

Dans la suite, nous utiliserons principalement la page **[6]** qui décrit les différentes étapes. On commence par télécharger la dernière version stable, **dokuwiki-2013-05-10a.tgz** « Weatherwax » à l'heure de l'écriture de ces lignes, à l'adresse **[7]**.

On décompresse l'archive dans le répertoire **/srv/http** (dans le cas d'Arch Linux). On renomme ensuite le répertoire en **archwiki** (la machine qui héberge possède Arch Linux). Par défaut, le propriétaire des fichiers que l'on vient de décompresser est **root**. Cela peut poser des problèmes si le serveur Apache doit créer des fichiers dans l'arborescence du wiki (et ce sera la cas dès la première création de page). On va rendre l'utilisateur **httpd** propriétaire de l'arborescence de fichiers de notre wiki :

```
# chown -R http:http archwiki/
```

Remarque

Vous pouvez trouver le nom de l'utilisateur propriétaire du serveur web dans le fichier de configuration **httpd.conf** aux lignes suivantes :

```
# User/Group: The name (or #number) of
the user/group to run httpd as.
# It is usually good practice to
create a dedicated user and group for
# running httpd, as with most system
services.
#
User http
Group http
```

On peut maintenant lancer l'installation proprement dite par l'intermédiaire de l'adresse http://localhost/archwiki/install.php dans votre navigateur préféré. Après avoir choisi la langue par défaut de notre wiki, on remplit le formulaire illustré en figure 1.

Dans notre cas, on choisira un wiki « fermé » ; nous allons l'utiliser au sein

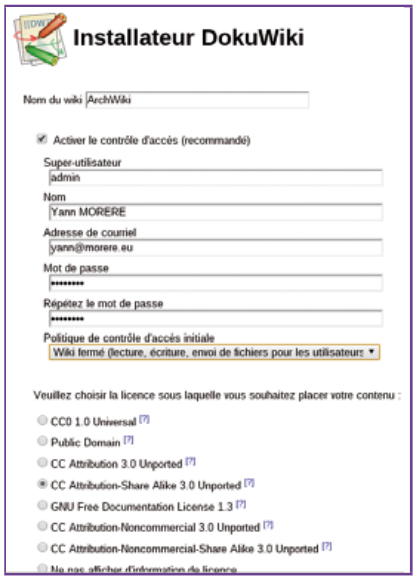


Figure 1

d'une ou plusieurs équipe(s) de développement. La plupart des pages seront privées et associées à des groupes/utilisateurs. Il sera bien sûr possible d'afficher une partie publique dans notre wiki grâce aux ACL. Finalement, on efface le fichier d'installation :

```
# rm install.php
```

Voici à quoi ressemble notre tout nouveau wiki (Fig. 2).

Figure 2

Afin d'éviter les auto-inscriptions (bouton « s'enregistrer » en haut à gauche), on se connecte et on désactive l'enregistrement de nouveaux utilisateurs ; nous les ajouterons manuellement par la suite. Pour cela, après s'être connecté en « administrateur », on se rend à la page ***Administrer***. Ensuite, dans la section ***Paramètres d'authentification > disableactions***, on coche l'action ***S'enregistrer***. On n'oublie pas de valider les modifications en bas de la page d'administration. Après déconnexion, on s'aperçoit que le bouton a disparu.

Dans cette page ***Administrer***, vous trouverez tous les paramètres de configuration de DokuWiki : paramètres de base, d'affichage, authentification, d'anti-spam, d'édition, de syndication, etc. C'est dans cette page aussi que vous trouverez le paramétrage de l'authentification via LDAP, ou encore base de données SQL.

Pour une configuration plus avancée des paramètres de votre wiki, je vous renvoie à l'adresse **[21]** qui décrit toutes les sections et leurs paramètres.

4 Configuration

4.1 Installation d'un nouveau thème

L'apparence de DokuWiki peut être complètement modifiée grâce aux *templates* (également appelés modèles, skins, gabarits ou encore thèmes). Il n'y a pas de gestionnaire d'installation de template comme on peut en trouver sur des CMS tel que WordPress. Ici, c'est « old school » ; on décompresse une archive dans le répertoire **lib/tpl** de l'installation de DokuWiki. On commence par choisir un thème qui nous convienne parmi ceux disponibles sur la page **[8]**. J'ai choisi le template « arctic » **[9]**.

Remarque

On fera attention de vérifier que le thème est bien compatible avec la version de DokuWiki installée. Le cas échéant, il est possible que certains bugs apparaissent.

Pour l'installation, on télécharge puis on décompresse l'archive **arctic-stable.tar.gz** dans le répertoire **/srv/http/archwiki/lib/tpl**. Il vous faudra bien sûr les droits d'administrateur. Ensuite, on change le propriétaire du répertoire **arctic** afin de ne pas avoir de soucis avec les droits d'accès. Les commandes suivantes résument ces actions :

```
# cd /srv/http/archwiki/lib/tpl
# wget http://r.notomorrow.de/dokuwiki-template-arctic/pkg/arctic-stable.tgz
# tar xzf arctic-stable.tgz
# ls
arctic  arctic-stable.tgz  default  dokuwiki  index.php
# chown -R http:http arctic
# rm arctic-stable.tgz
```

Afin d'activer ce nouveau thème, on se rend dans la partie administration, dans la section ***Paramètres de DokuWiki > template*** afin de le sélectionner. Normalement, l'interface a complètement changé et ressemble désormais à celle de la figure 3.

Figure 3

On peut alors configurer notre nouveau thème dans la partie administration de DokuWiki : définir l'agencement des colonnes droite et gauche, ainsi que leurs contenus. À titre d'exemple, voici la configuration utilisée sur le wiki de la figure 3 :

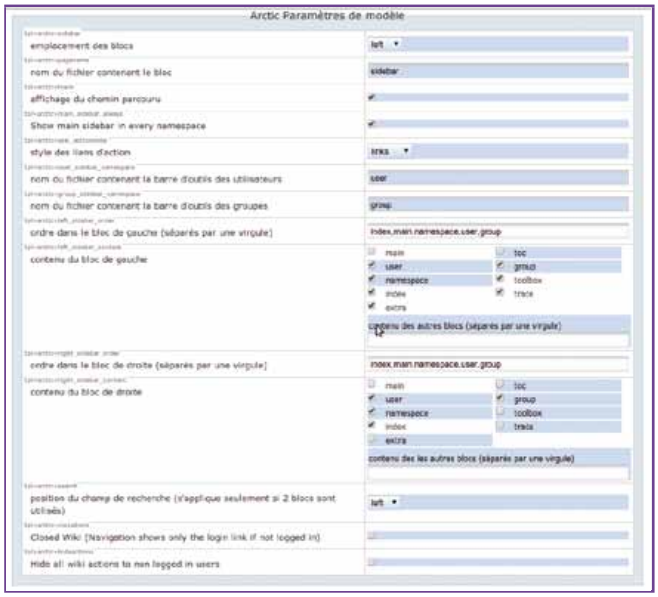


Figure 4

Voyons maintenant comment installer des plugins qui nous permettront d'étendre les fonctionnalités de notre wiki.

4.2 Installation d'extensions/plugins

L'installation d'une extension (ou plugin) peut se faire manuellement ; on télécharge l'archive puis on la décompresse dans le répertoire **`lib/plugins/`**. On peut aussi utiliser le gestionnaire de plugins disponible dans la partie administration.

Lors du choix des plugins, on prêtera une attention toute particulière à la compatibilité avec la version de DokuWiki installée et la présence de bugs signalés sur la page de téléchargement. Les plugins sont tous répertoriés à l'adresse **[10]**. Ils sont de différents types :

- les extensions « Syntax » qui étendent la syntaxe basique de DokuWiki ;
- les extensions « Action » qui remplacent ou étendent les fonctionnalités du noyau de DokuWiki ;
- les extensions « Admin » qui fournissent des outils d'administration supplémentaires ;
- les extensions « Helper » qui fournissent des fonctionnalités partagées par d'autres modules ;
- les extensions « Render » qui ajoutent de nouveaux modes d'exportation ou remplacent le moteur de rendu XHTML de base ;
- les extensions « Remote » qui ajoutent des web services ;
- les extensions « Auth » qui ajoutent des extensions d'authentification.

Dans notre cas, j'ai choisi d'ajouter la liste d'extensions suivante :

- **`box`** qui permet de mettre en valeur des parties de votre page à l'aide de boîtes colorées ;
- **`code`** qui permet l'intégration de code source dans la page avec la coloration syntaxique ;
- **`flowplayer`** qui permet d'intégrer des vidéos dans la page ;
- **`gallery`** qui permet de réaliser une galerie photo ;
- **`note`** qui permet de mettre en valeur une partie de texte à l'aide de boîtes ;
- **`odt`** qui permet d'exporter au format LibreOffice toute une page en un seul clic ;
- **`odt2dw`** qui permet de créer une page DokuWiki à partir d'un fichier LibreOffice ;
- **`pagemove`** qui permet de déplacer ou renommer une page facilement ;
- **`uparrow`** qui affiche une flèche qui permet de remonter directement en haut de la page ;

- **addnewpage** qui ajoute un mini formulaire pour la création de nouvelles pages ;
- **translation** qui permet de gérer la traduction d'une même page dans des langues différentes.

Vous pouvez en ajouter bien d'autres en fonction de vos besoins : le plugin LaTeX vous permettra de réaliser le rendu d'équation mathématique dans votre page, captcha pour se défendre des robots, Doodle vous permettra de fixer la date d'une réunion, etc.

L'installation d'extensions à l'aide du gestionnaire est simplissime (Fig. 5) ; il suffit de copier/coller l'adresse de téléchargement du plugin dans le champ idoine et de cliquer sur le bouton ***Télécharger***. DokuWiki s'occupe du reste.

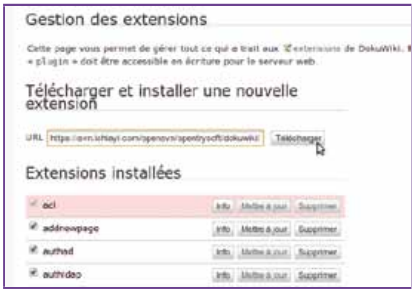


Figure 5

Cependant, lors de mes tests, j'ai rencontré des erreurs de téléchargement. En effet, de nombreux plugins utilisent dorénavant un dépôt GitHub. Ce dernier utilise le protocole HTTPS pour le téléchargement. Si comme moi vous obtenez des erreurs, il vous faudra activer l'extension « openssl » pour PHP, afin de pouvoir utiliser le protocole HTTPS. Il suffit de dé-commenter la ligne suivante :

```
extension=openssl.so
```

dans le fichier **/etc/php/php.ini** puis de redémarrer Apache.

Arrêtons-nous quelques instants sur le module « addnewpage ». Il permet d'ajouter un petit formulaire de création de page, ce qui simplifie la tâche à l'utilisateur. En effet, il n'est plus nécessaire de créer un lien vers cette nouvelle page comme de coutume dans les wikis. L'ajout du formulaire dans une page se fait à l'aide de la syntaxe :

```
{{NEWPAGE}}
```

et cela crée le formulaire suivant :

Figure 6

Cependant, il peut être intéressant d'avoir accès à la création de page directement depuis le menu de la colonne de gauche. Pour cela, on va ajouter le code de création du formulaire dans la page qui régit l'affichage de la colonne gauche. Dans le répertoire **/srv/http/archwiki/lib/tpl/arctic/**, on édite/crée le fichier **left_sidebar.html** et on y ajoute le code suivant comme indiqué dans **[11]** :

```
<!-- Extra contest for left sidebar goes
there -->
<?php echo p_render('xhtml',p_get_instructi
ons('{{NEWPAGE}}'),$info) ?>
```

Dans le cas d'une création, on n'oublie pas de modifier le propriétaire de la page :

```
# chown http:http left_sidebar.html
```

Voilà, normalement, le formulaire de création de page se trouve dans la colonne de gauche (Fig. 7).

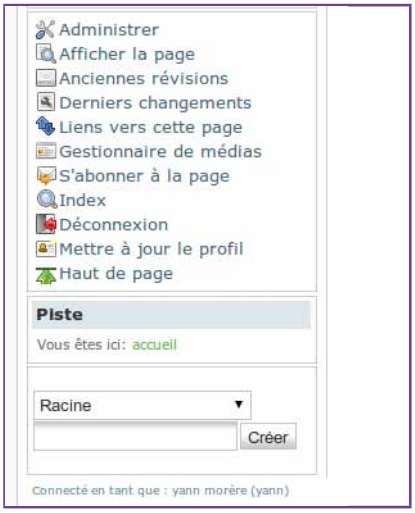


Figure 7

Passons maintenant à la création de vos pages wiki.

5 Syntaxe et création de pages

5.1 Syntaxe

DokuWiki utilise un langage de balisage simple. Les plugins ajoutent aussi leur propre syntaxe. Vous trouverez à l'adresse **[12]** un récapitulatif de la syntaxe des principales balises de DokuWiki, ainsi que celles fournies par quelques plugins (notamment ceux que nous avons installés). Vous trouverez aussi sur cette page la création de tableaux, de listes, de liens, l'insertion d'images, de caractères spéciaux, de boîtes, de notes et d'émoticônes.

Lors de la création des pages, un éditeur WYSIWYG vous assiste dans votre rédaction (Fig. 8).

5.2 Pages et espaces de noms

Lors de la création d'une page, son nom est converti en lettres minuscules et les seuls caractères spéciaux autorisés sont **.,-_**.

DokuWiki possède le concept des espaces de noms (*namespace*) ou catégories. C'est en fait un dossier dans lequel DokuWiki stockera les fichiers textes correspondant aux pages que vous générerez. Les deux points **:** sont utilisés pour créer des espaces de noms.

Il n'est pas nécessaire de créer explicitement les catégories. Pour créer une page dans une nouvelle catégorie, il suffit de saisir le nom de la catégorie et le nom de la page séparés par **:**. Ainsi, la saisie de la chaîne de caractères **test:exemple** dans notre formulaire de création de page créera la page « exemple » dans le nouvel espace de noms « test ».

Remarque

On peut aussi créer une page en entrant directement son URL ; si elle n'existe pas et que vous avez les droits, elle sera créée.

Figure 8

Il est bien sûr possible de spécifier une arborescence complète en enchaînant les espaces de noms séparés par des **:**. Par exemple, **blender:tutoriels:plugins:matwire** générera l'arborescence suivante :

```
├─ blender
│  ├─ tutoriels
│  │  ├─ plugins
│  │  │  ├─ matwire.txt
```

Pour supprimer une page, il suffit de l'éditer et d'en effacer son contenu. Lorsque vous avez enlevé toutes les pages d'une catégorie, la catégorie n'existe plus.

En ce qui concerne le renommage, nous utiliserons le plugin prévu à cet effet « pagemove », dont l'accès se fait par l'intermédiaire de la partie administration (Fig. 9).

Concernant la création des pages, afin de rendre votre wiki attractif et lisible, il est fortement conseillé de diviser vos textes longs en sections, en utilisant les entêtes de différents niveaux. Cela permet, lorsque cette option est activée, d'afficher une table des matières en haut de page qui permet l'accès rapide à une section particulière (Fig. 10).

Figure 9

Figure 10

Les conseils de « Bon Style » d'une page wiki sont donnés en page **[13]**.

5.3 Traduction de pages

Dans le cadre d'un wiki d'équipe internationale, il peut être intéressant de posséder des traductions dans différentes langues de chacun des documents. Pour cela, nous allons utiliser le plugin « Translation » précédemment installé. Dans un premier temps, nous allons configurer l'extension pour qu'elle prenne en charge une série de langues de traduction. Nous baserons les noms des langues sur les codes ISO : dans mon cas, j'ai choisi pour le test « fr, en, br » (français, anglais et portugais du Brésil).

L'astuce consiste à utiliser des espaces de noms différents pour chaque langue et d'y déposer les fichiers traduits. Pour la configuration du plugin, on remplit la section correspondante dans la partie administration (Fig. 11, page suivante).

Ensuite, on va ajouter un morceau de code à notre template afin d'afficher un sélecteur des traductions disponibles de chaque page. Pour cela, on ajoute le code suivant dans le fichier **pageheader.html** que l'on créera :

```
<?php
$translation = &plugin_
load('helper','translation');
if ($translation) echo $translation-
>showTranslations();
?>
```

Ensuite, on n'oublie pas de modifier les droits d'accès :

```
# chown http:http pageheader.html
```

Pour l'instant, le menu de sélection de langue n'est pas affiché. Il nous faut d'abord créer les pages **fr:test**, **en:test** et **br:test** et les remplir avec chacune leur traduction. Maintenant, vous avez accès aux différentes traductions de la page (Fig. 12, page suivante).

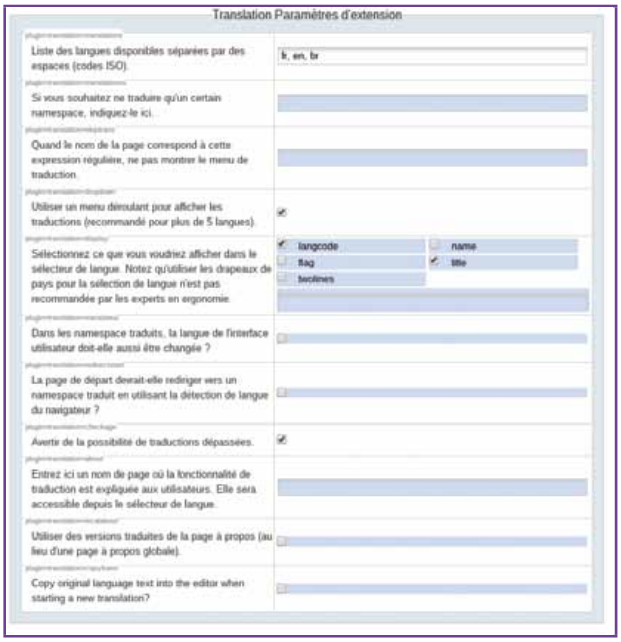


Figure 11

On remarquera que si la traduction d'une page n'existe pas et que vous tentez de l'afficher, on vous proposera gentiment de la créer et donc de traduire :-).

Vous trouverez d'autres options de configuration dans la documentation du plugin à l'adresse **[14]**. Passons maintenant à une partie très importante dans DokuWiki, la gestion des ACL.

6 Les ACL

La caractéristique première d'un wiki est d'être ouvert à tout le monde, et modifiable facilement par toutes les personnes qui désirent contribuer. Mais ce genre de concept ne vaut que dans un monde parfait où il n'y a pas d'éléments nocifs... Et comme chacun sait, nous sommes très loin d'un monde parfait.

Cela nous fait une bonne raison de limiter l'accès au wiki. Dans notre cas, il sera même fermé. Seulement un certain nombre d'utilisateurs identifiés et authentifiés (à vous de choisir le moyen : manuel LDAP, SQL, ...), que l'on pourra même diviser en groupes, auront un accès. L'accès externe au wiki se fera par exemple en lecture seule, et certains groupes auront des droits d'écriture sur certaines parties du wiki et non sur d'autres... Tout cela pour dire que DokuWiki fournit un contrôle d'accès très puissant.

Afin de pouvoir utiliser les ACL, il faut que DokuWiki soit configuré pour les utiliser. Pour cela, dans la partie administration, section ***Authentification***, il faut cocher ***Utiliser les listes de contrôle d'accès (ACLs)*** et enregistrer la configuration.

Figure 12

6.1 Utilisateurs et groupes

Des utilisateurs peuvent être ajoutés, retirés grâce au gestionnaire d'utilisateurs (Fig. 13). Lors de cette inscription (nous avons désactivé l'auto-inscription), il est possible de spécifier un ou plusieurs groupe(s) d'appartenance. Par défaut, deux groupes sont présents :

- ***ALL*** qui représente tout le monde. Ce groupe comprend aussi les utilisateurs non connectés. C'est par l'intermédiaire de ce groupe que l'on peut gérer l'affichage public de certaines pages par exemple, dans le cas de notre wiki privé ;
- ***user***, le groupe par défaut des utilisateurs enregistrés.

Lors de la création des utilisateurs, vous pourrez spécifier de nouveaux groupes (séparés par des virgules). Dans l'exemple de la figure 13, l'utilisateur Pierre fera partie du groupe ***user*** (par défaut) et aussi du groupe des développeurs. On pourrait imaginer aussi un découpage par projet, afin de bien séparer les utilisateurs.

Figure 13

6.2 Les droits d'accès / restrictions d'accès

Il faut ici raisonner en termes de règles de droits d'accès. Elles peuvent être liées aux pages, mais aussi aux espaces de noms et sont aussi liées aux utilisateurs et aux groupes. Ceci permet notamment de bloquer facilement toute une partie du wiki à l'aide d'une seule règle.

Il existe 5 types de permissions : lire (***read***), éditer (***edit***), créer (***create***), télécharger vers le serveur (***upload***) et effacer (***delete***). Les permissions de niveau plus élevé contiennent celles des niveaux inférieurs, ***read*** étant le niveau le plus bas. Les permissions ***Création***, ***Envoyer*** et ***Effacer*** ne s'appliquent qu'aux espaces de noms et non aux pages.

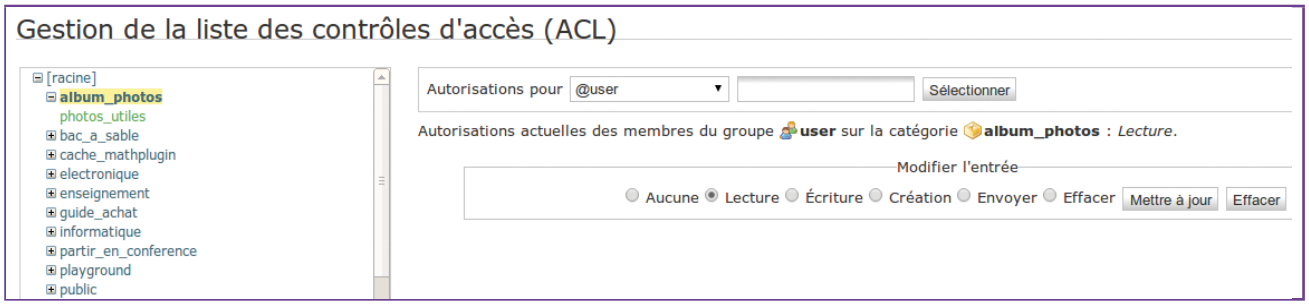


Figure 14

Il faut bien comprendre que lorsque DokuWiki contrôle les droits qu'il doit attribuer à un utilisateur, il utilise toutes les règles contenant le nom d'utilisateur ou de groupe. La règle qui donne la permission la plus élevée est utilisée. Les permissions concernant la page sont d'abord examinées, ensuite tous les espaces de noms supérieurs sont contrôlés jusqu'à ce qu'une règle soit trouvée.

Ainsi, si une page est interdite en lecture pour le groupe ***user***, mais autorisée pour le second groupe dont vous faites partie, vous aurez le droit de lire cette page.

Pour ajouter/enlever des règles, le plus simple est d'utiliser l'interface de gestion des ACL (***Gestion de la liste des contrôles d'accès***) dans la partie administration. La figure 14, présente le formulaire d'ajout de règles d'accès.

On commence par sélectionner dans le menu déroulant le groupe auquel on veut attribuer les règles. Si vous désirez appliquer des règles à un utilisateur spécifique, il faudra renseigner son login. Ensuite, dans la partie arborescente à gauche, on va pouvoir sélectionner l'espace de noms ou le fichier sur lequel appliquer cette règle. Une règle appliquée sur un espace de noms s'appliquera sur les espaces de noms enfants et les pages qu'ils contiennent. Finalement, on choisit le type de permissions à attribuer parmi les boutons radio. Il ne reste plus qu'à appliquer la règle.

La figure 15 présente une partie des ACL appliquées sur le wiki d'un laboratoire (qui n'existe plus :-)) **[15]**. Cette gestion permet de laisser une partie du wiki publique, et pour les utilisateurs enregistrés, de séparer les droits d'accès pour le groupe générique ***user*** et le groupe ***lasc*** représentant une partie de ces utilisateurs enregistrés.

7 Téléversement et type MIME

Lors de la rédaction, vous avez la possibilité de téléverser des images ou des documents pour améliorer votre article. Pour cela, DokuWiki dispose d'un gestionnaire de téléversement qui permet l'ajout multiple de fichiers (Fig. 16).

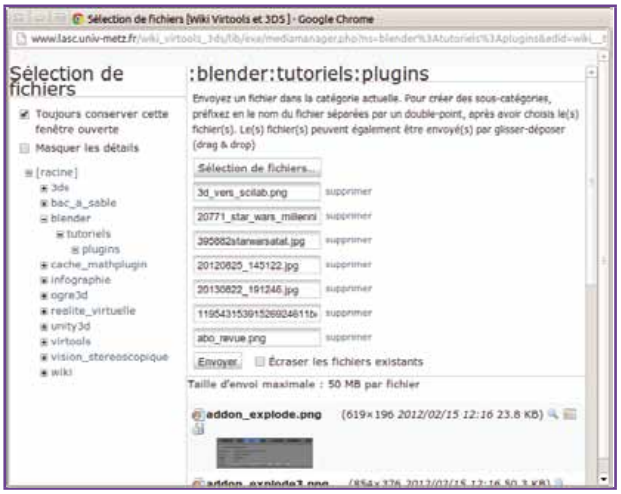


Figure 16

Figure 15

Cependant, si vous tentez de téléverser un simple fichier texte, DokuWiki refusera. En effet, une série de types de fichiers sont désactivés pour le téléversement, car les spammeurs les utilisent dans les wikis publics **[16]**. C'est notamment le cas des fichiers HTML, .conf, TXT, ou encore XML. De même, tous les types de fichiers ne sont pas autorisés au téléversement, toujours pour des problèmes de sécurité. La liste complète des types valides est définie en page **[16]**.

Si vous désirez néanmoins autoriser les fichiers textes, il faudra éditer le fichier **conf/mime.conf** de DokuWiki et dé-commenter la ligne :

```
txt text/plain
```

8 Sauvegarde du wiki

Si votre wiki est hébergé et que vous avez un accès FTP, il sera alors très simple de réaliser une sauvegarde complète (et même faire un miroir local au cas où le service distant serait défaillant) via un client FTP. Eh oui, il n'y a pas de base de données, la copie de l'arborescence suffit ! On réalise cela simplement à l'aide de **lftp** et son mode **mirror** par exemple :

```
# lftp ftp://user:"motdepasse"@serveur -e "mirror -e html/dokuwiki
./mirror_dokuwiki ; quit"
```

Lors de mes tests, j'ai remarqué que **lftp** n'aimait pas les dossiers vides et générait des erreurs dans le cas où le serveur FTP n'accepte pas les connexions passives ; on ajoute alors l'option **set ftp:passive-mode 0** :

```
# lftp ftp://user:"motdepasse"@serveur -e "set ftp:passive-mode 0;
mirror -e html/dokuwiki ./mirror_dokuwiki ; quit"
```

9 Ferme de wikis

Si la gestion des groupes et ACL ne vous suffit pas pour séparer proprement les contenus de vos équipes de travail, vous pouvez passer à un niveau d'abstraction supérieur avec la ferme de wikis. Il s'agit d'une collection de wikis qui utilise le même serveur web et partage la même instance du moteur de wikis. Le moteur wiki « parent » se nomme « la ferme » et les sites wikis « enfants » sont les animaux.

Les animaux partagent un jeu unique de plugins et templates, mais chacun d'eux peut activer/utiliser ceux qui lui semblent nécessaires. Il s'agit donc d'un wiki multisite.

Il existe plusieurs méthodes pour créer ce type de structure. Une approche ancienne était basée sur l'utilisation des liens symboliques (*Symlink Farm* **[17]**). La page **[18]** indique la manière la plus récente et rapide de réaliser une ferme de wikis. On trouvera aussi un plugin « farm » **[19]** qui automatise la méthode du « redirect » **[20]** pour la gestion de la ferme.

Conclusion

Nous voici au terme de cet article. Après plus de 5 ans d'utilisation journalière de DokuWiki, cet outil modulaire et adaptable me rend de très nombreux services et je ne l'échangerais pour aucun autre wiki. Son installation est très simple, ses plugins sont nombreux. Il est mis à jour régulièrement et fait partie des wikis les plus utilisés. Et que dire de l'utilisation des fichiers textes pour le stockage... C'est tout simplement l'interopérabilité ultime ;-) Comme dit l'autre : « l'essayer, c'est l'adopter ». ■

Liens

[1] http://fr.wikipedia.org/wiki/Wiki

[2] http://www.framasoft.net/rubrique168.html

[3] http://www.wikimatrix.org/

[4] https://www.dokuwiki.org/start?id=fr:requirements

[5] https://www.dokuwiki.org/fr:security

[6] https://www.dokuwiki.org/start?id=fr:install

[7] http://www.splitbrain.org/projects/dokuwiki

[8] https://www.dokuwiki.org/fr:template

[9] https://www.dokuwiki.org/template:arctic

[10] https://www.dokuwiki.org/start?id=fr:plugins

[11] https://www.dokuwiki.org/plugin:addnewpage

[12] http://www.lasc.univ-metz.fr/dokuwiki/doku.php?id=wiki:dokuwiki_en_10_minutes

[13] https://www.dokuwiki.org/fr:tips:good_style

[14] https://www.dokuwiki.org/plugin:translation

[15] http://www.lasc.univ-metz.fr/dokuwiki/

[16] https://www.dokuwiki.org/start?id=fr:mime

[17] https://www.dokuwiki.org/tips:symlink_farm

[18] https://www.dokuwiki.org/farms

[19] https://www.dokuwiki.org/plugin:farm

[20] https://www.dokuwiki.org/tips:redirect_farm

[21] https://www.dokuwiki.org/start?id=config

ASCII.IO : LA SOLUTION POUR VOS TUTORIELS VIDÉO EN MODE CONSOLE

par Benoît Benedetti

Vous avez développé un nouvel outil en ligne de commandes qui va révolutionner le monde ! Mais voilà, sans une bonne démonstration pour montrer ses fonctionnalités, celui-ci va complètement passer inaperçu. Pourquoi ne pas faire un screencast ? Parce que vous n'avez pas le temps de faire le montage d'un screencast complet, ni l'argent d'investir dans un micro ? Mais si une simple capture de votre terminal suffit pour présenter votre création, le service Ascii.io répond à votre problématique !

1 Présentation

Ascii.io est un service en ligne qui vous propose de partager des captures de votre terminal. Ce service se compose d'un outil Python **[1]**, qui permet de procéder aux enregistrements et de les uploader sur le site en ligne ascii.io. Si vous ne désirez pas passer par le site officiel pour stocker vos données, le code source du service web est disponible **[2]** et offre la possibilité d'héberger votre propre instance d'Ascii.io.

2 Enregistrer vos captures à l'aide du client

2.1 Installation

L'utilitaire Python est disponible en version 1.0 à l'écriture de cet article, que nous allons installer sous Debian Wheezy. Commencez par installer Curl :

```
$ sudo aptitude -y install curl
```

Pour pouvoir procéder à l'installation de l'utilitaire :

```
$ curl -sL get.ascii.io | bash
```

Après avoir demandé votre mot de passe pour installer le script Python à l'aide de **sudo**, la commande **asciiio** est disponible :

```
$ asciiio --version
asciiio 1.0
```

2.2 Capture

Pour capturer, rien de plus simple : lancez simplement la commande **asciiio** sans argument, la capture va démarrer ; terminez celle-ci par la commande **exit** :

```
$ asciiio
~ Asciicast recording started. Hit ^D (that's Ctrl+D) or type "exit" to finish.

user@debian:~$ asciiio --help
usage: asciiio [-h] [-i] [-y] [-c <command>] [-t <title>] [action]

Asciicast recorder+uploader.

Actions:
 rec           record asciicast (this is the default when no action given)
 upload        upload recorded (but not uploaded) asciicasts
 auth          authenticate and/or claim recorded asciicasts

Optional arguments:
 -c command    run specified command instead of shell ($SHELL)
 -t title      specify title of recorded asciicast
 -y            don't prompt for confirmation
 -h, --help    show this help message and exit
 --version     show version information
$ exit
```

Après être sorti de la capture, **asciiio** vous demande si vous désirez uploader celle-ci vers le service en ligne, ou la supprimer :

```
~ Asciicast recording finished.
~ Do you want to upload it? [Y/n]
```

Si vous ouvrez un autre shell, vous pouvez voir les fichiers de la capture, créés dans un sous-dossier du dossier **queue** de votre répertoire **~/.asciio** :

```
$ ls -R ~/.ascii.io/queue/
...
/home/user/.ascii.io/queue/1377077030:
meta.json  stdout  stdout.time
```

Ces différents fichiers ne sont pas au format vidéo et inexploitables en tant que tels, sauf par le service web. Validez donc l'upload pour pouvoir visualiser vos captures :

```
~ Asciicast recording finished.
~ Do you want to upload it? [Y/n] Y
~ Uploading...
http://ascii.io/a/5069
```

L'upload terminé, les fichiers temporaires sont supprimés et votre capture disponible à l'adresse indiquée (Fig. 1).

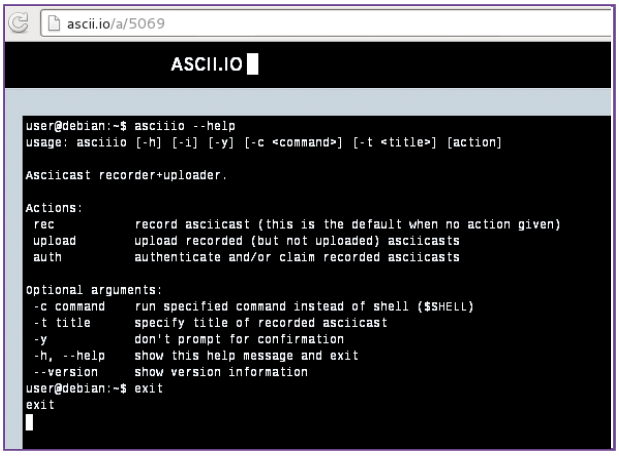


Fig. 1 : Visualisation du screencast à l'URL indiquée

3 Gérer vos captures

Si vous voulez pouvoir gérer, rassembler, supprimer ou modifier les titres et descriptions de vos captures, il faut créer un compte sur Ascii.io, puis réclamer les captures à partir de ce compte.

Malheureusement, l'identification se fait via les services d'authentification propriétaires de GitHub ou Twitter. Si vous possédez déjà un compte chez eux, ou que cela ne vous effraie pas, connectez-vous à l'aide de l'un ou l'autre. Puis, affichez l'URL ascii.io de votre jeton client pour pouvoir réclamer vos captures. Pour cela, lancez **asciiio** avec l'action **auth** :

```
$ asciiio auth
Open following URL in your browser to authenticate and/or claim
recorded asciicasts:

http://ascii.io/connect/7ec949ec-0a42-11e3-bcaf-08002778576f
```

Ouvrez l'URL indiquée, vous devriez arriver sur votre compte utilisateur avec tous les screencasts que vous avez enregistrés auparavant. Dorénavant, toute nouvelle capture viendra s'ajouter automatiquement à cette liste.

Si vous cliquez sur une capture vous appartenant tout en étant connecté, en bas de la page de lecture, en plus des informations générales sur la capture, sont disponibles des actions supplémentaires (Fig. 2) : un lien pour suppression et un autre pour mettre à jour titre et description (Fig. 3).

Fig. 2 : Si vous êtes connecté, plusieurs actions sont disponibles.

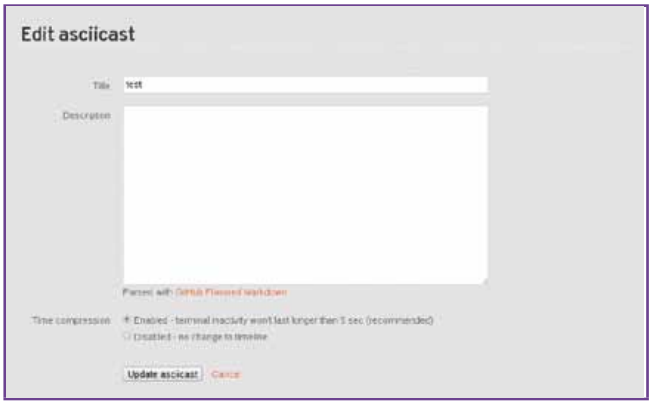


Fig. 3 : Édition des caractéristiques du screencast

Vous remarquerez le code JavaScript dans la section ***Share***, en bas de cette page. Il vous permet d'inclure facilement votre capture sur votre site web :

```
<html>
 <body>
  <h1>Ma super capture</h1>
  <p> Cliquez sur la capture suivante pour lancer la lecture
   <script type="text/javascript" src="http://ascii.io/a/5069.js"
id="asciicast-5069" async></script>
  </p>
 </body>
</html>
```

La capture apparaîtra tel un GIF animé, mais lu une seule fois. La lecture est lancée en cliquant sur la capture (Fig. 4).

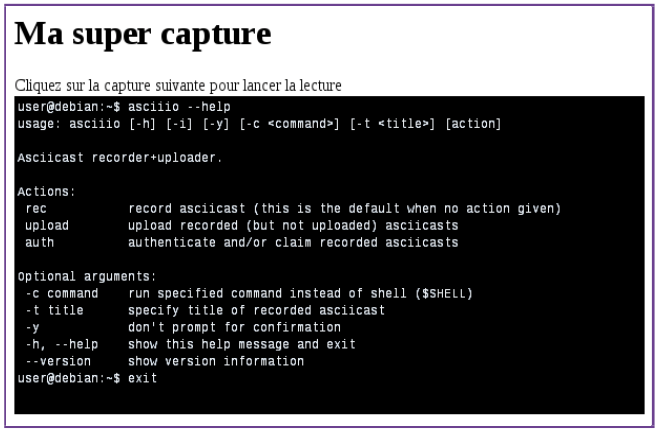


Fig. 4 : Notre capture intégrée dans une page web. Non encore lue, elle est grisée et nécessite de cliquer dessus pour lancer sa lecture.

4 Auto-hébergement

Si héberger vos captures sur un serveur tiers vous ennuie, vous pouvez auto-héberger le service sur votre propre machine. Nous allons voir comment auto-héberger votre propre instance web de Ascii.io. Attention, dans la suite, toutes les manipulations sont à effectuer en tant que l'utilisateur root.

4.1 Rbenv

Le service Ascii.io est développé avec le framework Ruby on Rails dans sa toute dernière version 4 et conseille d'utiliser Ruby 2. Or, Debian utilise Ruby 1.9.3 par défaut. On va donc installer **rbenv**, un utilitaire qui permet d'installer facilement n'importe quelle version de Ruby :

```
$ aptitude -y install git
$ git clone https://github.com/sstephenson/rbenv.git ~/.rbenv
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bash_profile
$ echo 'eval "$(rbenv init -)"' >> ~/.bash_profile
$ exec $SHELL -l
$ git clone https://github.com/sstephenson/ruby-build.git ~/.rbenv/
plugins/ruby-build
$ aptitude -y install build-essential libssl-dev libreadline-dev
$ CONFIGURE_OPTS="--with-readline-dir=/usr/include/readline" rbenv
install 2.0.0-p247
$ rbenv global 2.0.0-p247
$ rbenv rehash
```

Vous devriez avoir Ruby en version 2 de disponible :

```
$ ruby -v
ruby 2.0.0p247 (2013-06-27 revision 41674) [x86_64-linux]
```

4.2 PostgreSQL

L'application nécessite une base de données PostgreSQL. On installe ce serveur de base de données :

```
$ aptitude -y install postgresql
```

Puis, on crée le rôle **root** pour pouvoir utiliser la base de données :

```
$ echo "local all root trust" >>  /etc/postgresql/9.1/main/pg_hba.conf
$ su - postgres -c "createuser -s -r -d root"
$ service postgresql restart
```

4.3 Redis

Ascii.io utilise également la gemme Sidekiq, qui permet, en arrière-plan, de générer les images miniatures des screencasts, pour être affichées sur les pages du site. Cette gemme sera installée plus loin, nous allons pour l'instant installer le moteur de stockage Redis dont cette gemme a besoin :

```
$ wget http://download.redis.io/releases/redis-2.6.16.tar.gz -O /
tmp/redis-2.6.16.tar.gz
$ cd /tmp
$ tar xzf redis-2.6.16.tar.gz
$ cd redis-2.6.16
$ make
$ make install
```

4.4 Installation

On a encore besoin d'installer des dépendances :

```
$ aptitude -y install libxml2-dev libxslt1-dev libpq-dev tmux bsdutils
```

Pour pouvoir gérer les captures, l'application utilise la librairie externe **libtsm**. Celle-ci n'est pas disponible dans les dépôts Debian (et sûrement non disponible sur de nombreuses autres distributions), et est automatiquement compilée et installée par le script d'installation de Ascii.io. Il faut néanmoins récupérer manuellement les paquets nécessaires à sa compilation :

```
$ aptitude -y install libtool libbsd-dev pkg-config autotools-dev
python-dev automake autoconf
```

On peut maintenant récupérer les sources de l'application dans le dossier **~/ascii.io** :

```
$ git clone git://github.com/sickill/ascii.io.git ~/ascii.io
$ cd ~/ascii.io/
```

Ensuite, on crée un fichier de configuration de connexion à la base de données à partir du fichier exemple fourni avec les sources :

```
$ cp config/database.yml.example config/database.yml
```

Modifiez ce fichier **config/database.yml** comme suit pour refléter notre configuration :

```
#config/database.yml
# Enable passwordless Postgres access with following commands:
#
# echo "local all $USER trust" | sudo tee -a /etc/postgresql/9.1/
main/pg_hba.conf
# sudo su - postgres -c "createuser -s -r -d $USER"
```

```
development:
  adapter: postgresql
  encoding: unicode
  database: ascii_io_development
  pool: 25
  username: root
  password:
  min_messages: WARNING

# Warning: The database defined as "test"
will be erased and
# re-generated from your development
database when you run "rake".
# Do not set this db to the same as
development or production.
test:
  adapter: postgresql
  encoding: unicode
  database: ascii_io_test
  pool: 5
  username: root
  password:
  min_messages: WARNING

production:
  adapter: postgresql
  encoding: unicode
  database: ascii_io_production
  pool: 25
  username: root
  password:
```

Il nous faut aussi installer la gemme Bundler :

```
$ gem install bundler
$ rbenv rehash
```

Bundler, à partir du fichier **Gemfile** du répertoire des sources, va automatiquement installer toutes les gemmes nécessaires indiquées dans ce fichier. L'application nécessite d'ailleurs un moteur de rendu JavaScript, qui doit être indiqué manuellement dans ce fichier **Gemfile**. Nous utiliserons la gemme **therubyracer** comme moteur JS. Ouvrez donc le fichier **Gemfile** et ajoutez cette dépendance :

```
#fichier Gemfile
...
gem 'tsm', :git => 'git://github.com/
sickill/tsm.git'
gem 'therubyracer'
...
```

Puis, lancez le script d'installation, qui va compiler la librairie **libtsm**, appeler **bundler** pour installer les différentes gemmes et **rake** pour initialiser la base de données :

```
$ ./script/setup
```

4.5 Démarrage

Dans un premier terminal, on démarre Redis :

```
$ redis-server
```

Dans un deuxième, on peut démarrer Sidekiq, qui va permettre de générer les miniatures :

```
$ bundle exec sidekiq
```

Si vous avez suivi toutes les étapes précédentes, vous pouvez (enfin !) démarrer votre application rails :

```
$ bundle exec rails server
...
>> Listening on 0.0.0.0:3000, CTRL+C to stop
```

L'application est démarrée sur le port 3000 du serveur. Vous pouvez donc ouvrir votre navigateur à l'URL http://adresse_ip_du_serveur:3000 et profiter d'un site web identique au site officiel, vide de toute capture pour le moment.

4.6 Enregistrer des captures

Le client utilise par défaut htttp://ascii.io comme URL d'upload. Pour utiliser une autre URL, il faut modifier votre fichier de configuration **~/.ascii.io/config**. Si vous ouvrez ce fichier, celui-ci comporte trois sections : **user**, **api** et **record**. Par défaut, seule la section **user** possède un paramètre de configuration, à savoir votre **token** d'identification utilisateur.

Pour modifier l'URL de l'API à utiliser, renseignez le paramètre **url** de la section **api** dans **~/.ascii.io/config** :

```
[api]
url = 'http://localhost:3000'
```

Mettez l'adresse IP adéquate, ou **localhost** si, comme moi, client et serveur sont sur la même machine.

Vous pouvez désormais utiliser le client comme nous l'avons fait dans la première partie de l'article. Lorsque vous confirmerez l'upload, la capture sera cette fois uploadée vers votre instance auto-hébergée. Vous aurez l'ultime satisfaction d'avoir uploadé la capture avec pour identifiant le numéro 1 (Fig. 5).

Vous pourrez également continuer à intégrer une capture en récupérant le code JavaScript affiché sur sa page d'information, comme nous l'avons vu en première partie.

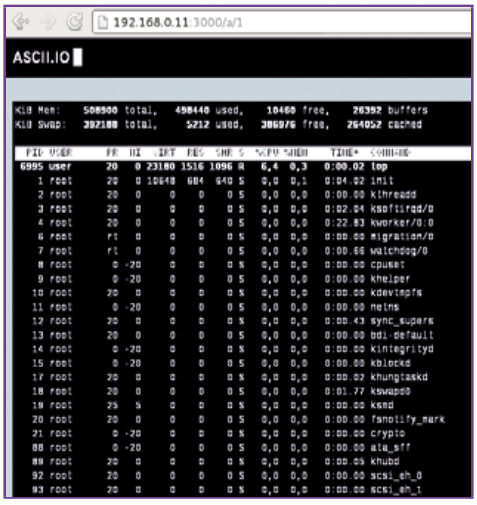


Fig. 5 : Notre premier screencast auto-hébergé !

Conclusion

Ascii.io est un petit service bien pratique pour vos captures, en plus de pouvoir être auto-hébergé. Malheureusement, même en étant auto-hébergé, le seul système d'authentification disponible se fait par Twitter ou GitHub. Si vous avez beaucoup d'utilisateurs qui désirent gérer leurs captures, vous pouvez opter pour la solution simple de donner une adresse IP publique à votre serveur, et laisser l'authentification par ces services tiers. Vous pouvez également opter pour la solution longue en contribuant au projet et en offrant d'autres moyens d'authentification ;-).

Ou sinon, si vous êtes seul utilisateur du service, vous pouvez simplement ignorer l'authentification et utiliser votre service auto-hébergé tel quel, puis supprimer au besoin des captures en tapant directement dans la base de données.

Capturez bien ! ■

Références

[1] https://github.com/sickill/ascii.io-cli

[2] https://github.com/sickill/ascii.io

IMAGINER ET CONCEVOIR UN MONDE SANS CONTRAINTE

GREAT PLACE TO WORK® Best Workplaces 2013 France

UNIVERSUM TOP 100 IDEAL EMPLOYER 2013 STUDENT SURVEY

PARROT RECRUTE !

En plein cœur de Paris, Parrot développe des solutions innovantes en partenariat avec des marques prestigieuses du monde de l'automobile. Technologies de pointe, challenges complexes en systèmes embarqués et ambitions internationales, Parrot fait le pari de l'innovation dans l'univers des périphériques sans fil. Pour rester le leader des systèmes embarqués, nous renforçons nos équipes de développement spécialisées sur Linux et Android . Vous serez ainsi garant de la bonne mise en œuvre des logiciels produits chez nos clients.

International par essence, nous évoluons dans un environnement anglophone et cherchons avant tout des collaborateurs passionnés, autonomes et experts sur leurs métiers.

MATTHIEU, Expert logiciel

NOUS RACONTE

«Faire partie de l'équipe R&D de Parrot, tant sur le secteur produits grand public ou automobile, c'est l'opportunité de participer à l'ensemble du processus de développement.

De la conception et validation d'ASIC Parrot jusqu'à la fourniture de services natifs (reconnaissance vocale, lecteur multimédia, connectivité réseau, ...) en passant par le développement dans le noyau linux, nous nous appuyons sur de nombreux outils et briques open source.

Nous sommes amenés à interagir directement avec la communauté par un échange sur les mailing list, la proposition de patch, des conférences, ... Cet écosystème enrichissant est formateur car il permet d'apprendre des autres, de confronter ses idées à celles de nombreux développeurs, de partager sa passion et de communaliser nos efforts.

Une des forces de Parrot est qu'elle permet de travailler sur une diversité de domaines avec une grande liberté.»

BOUSCULEZ TOUS LES CODES SUR
WWW.RECRUTE.PARROT.COM

UN MONDE D'INNOVATIONS AU CŒUR DE PARIS

Parrot

ontheMOON

Le

CNLL

Conseil National du Logiciel Libre

et ses représentants régionaux

PRÉSENTENT

les RENCONTRES REGIONALES du LOGICIEL LIBRE & du SECTEUR PUBLIC

Du 04 Octobre 2013 au 17 Avril 2014

Paris
Lille
Metz
Strasbourg
Lyon
Rennes
Brest
Nantes
Bordeaux
Toulouse

Venez rencontrer vos pairs
avec des problématiques communes !

Venez rencontrer
les prestataires locaux de l'OpenSource !

En savoir plus sur **WWW.rrll.fr**

Evénement soutenu par **ADOSIS**, *Agence Digitale*, spécialiste OpenSource
www.adosis.com

adosis