NOUVELLE FORMULE - NOUVELLE FORMULE - NOUVELLE FORMULE

N°172 DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

GNU LINUX MAGAZINE / FRANCE

CODE / TESTS
Validez les fonctionnalités de vos applications grâce aux tests fonctionnels p.74

ACTU / PHP
Hack, le PHP made in Facebook, est-il le seul avenir de PHP ? p.06

PYTHON / WEB
Utilisez le protocole WebSocket avec Python 3.4 p.50

ANDROID / CAMERA
Étudiez le système de gestion photo et vidéo d'Android p.56

NETADMIN / SERVEUR
MAÎTRISEZ VOTRE SERVEUR HTTP NGINX
grâce à la souplesse du langage Lua et aux outils d'OpenResty ! p.32

ALGO / IA
Une pile, une file, une liste ? Disséquez les structures linéaires p.18

HUMEUR / FRANÇAIS
Enfin la traduction du jargon informatique présentée par le Dr KissCool ! p.10

L 19275 - 172 - F: 7,90 € - RD

France METRO : 7,90 € – CH : 13 CHF – BEL/PORT.CONT : 8,90 € – DOM TOM : 8,50 € – CAN : 14 $ cad – TUNISIE : 18 TND – MAR : 100 MAD

21 & 22 juin 2014 - LE CENTQUATRE-PARIS

Maker Faire® Paris

L'événement Maker Faire arrive en France

Astronomie
Inventions Cuisine
Impression 3D
Robotique Écologie
Chimie Sciences
Do It Yourself Drone
Compétition de robots Geek
Craft
Arduino Origami
Énergie Modèlisme
Génie Musique Internet
Objets connectés Bricolage
Fun Artisanat Ingénierie

Ateliers - Démonstrations - Conférences

Informations et billetterie
makerfaireparis.com

Un événement Make:

Co-produit par leFabShop

cent quatre paris l'incubateur

GNU/Linux Magazine France
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
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Réalisation graphique : Kathrin Scali & Jérémy Gall
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

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, **N° ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,90 €

FSC MIXTE Papier issu de sources responsables FSC® C015136

Membre April www.april.org

ÉDITORIAL

La vie nous réserve parfois des expériences fascinantes. J'ai dernièrement souhaité effectuer une commande de fournitures sur un site dont je suis habituellement client. Sans être une multinationale américaine, ce site n'est pas non plus celui d'un petit artisan de quartier. Cette société est internationalisée et peut vous livrer dans de nombreux pays du monde.

Ainsi, désirant remplir mon panier, je me connecte sur ce site et constate qu'il a été mis à jour : beaucoup plus clair, une ergonomie plus aboutie... Un bon point donc, si ce n'est la tentative de version mobile pour laquelle l'ergonomie est inexistante.

Je commence à parcourir le catalogue et, afin de sauvegarder mon début de commande, je tente de me connecter à mon compte... Impossible ! Peut-être me suis-je trompé de mot de passe ? Nouvel essai... Même résultat ! Après cinq tentatives, je demande la réinitialisation de mon mot de passe. On m'informe que je vais recevoir un mail... Je l'attends toujours... Je renouvelle ma demande... rien ! Bref, au bout de quelques essais j'abandonne, déçu et agacé.

La nuit portant conseil, je me dis le lendemain que la personne qui gère ce site n'est peut-être pas un lecteur assidu de notre magazine et que, de ce fait, il serait possible qu'après la mise à jour du site, tous les comptes aient été effacés (je sais, ça semble impensable, mais on a déjà vu pire...). Aussi, je m'autorise un nouvel essai : je retourne sur le site et, ô surprise (ou plutôt désolation), je parviens à créer un nouveau compte en utilisant la même adresse mail que celle de mon compte précédent. J'avoue avoir espéré un message du type « un compte existe déjà avec cette adresse »... Mais il ne viendra pas ! Je reçois donc un mail de bienvenue, où j'apprends que les anciens comptes ont été effacés ! Élémentaire mon cher Watson !!!

Légèrement énervé, je reprends mes achats abandonnés la veille. Je n'étais pas au bout de mes surprises : des boutons inactifs, un panier limité à cinq articles, des produits qui apparaissent, disparaissent et réapparaissent... Vous l'aurez compris : le site mis en production était en fait en développement... Si si, vous avez bien lu, « en développement », avec nous - clients - en guise de bêta testeurs ! Résultat des courses : pas de commande et une furieuse envie d'offrir un abonnement à nos magazines aux développeurs de l'agence web ayant remporté le contrat...

Heureusement il y a vous, lecteurs de *GNU/Linux Magazine*, vous qui ne vous comportez pas de la sorte. D'ailleurs, pour vous aider à tester vos sites/applications, vous trouverez dans ce numéro un article sur les tests fonctionnels. En restant dans le domaine du Web, vous pourrez en profiter pour optimiser votre serveur HTTP avec les articles sur Nginx. Enfin, nous vous avons préparé encore de nombreuses autres choses à découvrir dans nos différentes rubriques... en tournant la page.

Tristan Colombo

AGENDA

Hack in Paris

Date : du 23 au 27 juin

Lieu : Paris – France

Site officiel : https://www.hackinparis.com/home

Événement centré sur la présentation des techniques d'intrusion et de la lutte contre celles-ci. Pour rappel, les hackers se répartissent en plusieurs branches dont les *white hats* (professionnels de la sécurité informatique ayant pour seul objectif la connaissance) et les *black hats* (qui cherchent à nuire, voler des informations). Découvrez la réalité du hacking et ses conséquences : l'état de l'art de la sécurité informatique, l'espionnage industriel, les tests d'intrusion, la sécurité physique, les bonnes pratiques, l'analyse des malwares et les mesures à mettre en place pour contrer les attaques.

Forum Ter@tec 2014

Date : les 1er et 2 juillet

Lieu : Palaiseau – France

Site officiel : http://www.teratec.eu/forum/

Événement gratuit (réservé aux professionnels), inscription obligatoire sur le site Internet.

Le forum Ter@tec, axé sur la simulation numérique et le calcul intensif, propose des sessions plénières le premier jour et des ateliers orientés technologie ou usage le second jour. Parmi ces ateliers du mercredi 2 juillet, l'un d'entre eux sera consacré au calcul scientifique et open source avec des témoignages d'utilisateurs et des présentations de solutions permettant de dresser un vaste panorama des pratiques de l'open source dans des milieux industriels comme le spatial, l'aéronautique, l'énergie, le médical ou encore la micro-électronique.

15èmes Rencontres Mondiales du Logiciel Libre

Date : du 5 au 11 juillet

Lieu : Montpellier – France

Site officiel : http://2014.rmll.info

Plus connu sous son nom abrégé de RMLL, cet événement présente un cycle de conférences, tables rondes et ateliers pratiques autour du logiciel libre et de ses usages. À l'heure où ces lignes sont écrites, le programme n'est pas encore disponible, mais Richard Stallman aurait confirmé sa présence.

10th Netfilter Workshop

Date : du 7 au 11 juillet

Lieu : Montpellier – France

Site officiel : http://workshop.netfilter.org/2014

Après PyData en parallèle d'EuroPython, voici Netfilter Workshop en parallèle des RMLL. Il s'agit de l'événement annuel de la communauté des développeurs de Netfilter permettant de discuter des développements en cours et futurs. Pour les non initiés, Netfilter est un framework qui fournit des outils de pare-feu (voir http://www.netfilter.org).

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE
N° 172

Open Source Convention OSCON 2014

Date : du 20 au 24 juillet

Lieu : Portland – États-Unis

Site officiel : http://www.oscon.com/oscon2014

Événement organisé par l'éditeur O'Reilly et dont le nom complet est d'ailleurs O'Reilly Open Source Convention. Les sujets abordés sont très variés : de la théorie, des bases de données, du Web, Android, Perl, PHP... et peu de Python. D'un autre côté c'est normal : à cette date-là, les Pythonistes européens seront à EuroPython.

EuroPython 2014

Date : du 21 au 27 juillet

Lieu : Berlin – Allemagne

Site officiel : https://ep2014.europython.eu/en/

L'événement européen de l'année pour les développeurs Python. Tout sur Python : les bonnes pratiques, les tests, l'utilisation de Python en calcul scientifique, le Web et plus encore ! La conférence à ne pas manquer si vous êtes un développeur Python.

PyData 2014

Date : du 25 au 27 juillet

Lieu : Berlin – Allemagne

Site officiel : http://www.pydata.org/berlin2014

Événement organisé en parallèle de l'EuroPython et dédié à la gestion des gros volumes de données (big data) en Python : calcul, analyse et visualisation. Au programme : partage d'expérience et de techniques pour tous les niveaux avec des tutoriels pour les débutants et des workshops sur des sujets plus pointus.

EuroScipy Europe 2014

Date : du 27 au 31 août

Lieu : Cambridge – Royaume-Uni

Site officiel : http://www.euroscipy.org/2014

EuroScipy est le rendez-vous européen et annuel des utilisateurs et des développeurs des outils scientifiques écrits dans le langage de programmation Python. La 7ème édition de la conférence aura lieu à Cambridge (Royaume-Uni), du 27 au 31 août. Les deux premiers jours sont consacrés à des tutoriaux, avec deux salles en parallèle, l'une dédiée aux débutants, l'autre aux utilisateurs avancés. Suivent deux jours de conférence sur des sujets variés autour du langage Python, de ses usages, des bonnes pratiques, des retours d'expérience, etc.

La conférence s'achève avec des « code sprints », où débutants comme confirmés se rejoignent avec pour objectif de coder ensemble sur leur outil préféré.

La conférence réunit étudiants, enseignants, passionnés, chercheurs et industriels dans une ambiance très conviviale permettant de nombreux échanges entre tous les participants. ■

QUEL AVENIR POUR PHP ?

par ***Stéphane Mourey*** *[Taohacker]*

Depuis quelques temps, dans la sphère PHP, on parle avec beaucoup d'enthousiasme de Hack, l'implémentation à la sauce Facebook de PHP. Certains y voient une véritable révolution pour ce langage. Qu'en est-il vraiment ? Hack est-il l'avenir de PHP ? Y a-t-il d'autres possibilités ?

Hack est incontestablement un projet intéressant. L'idée d'étendre PHP de façon à lui permettre d'utiliser des types de variables statiques pour améliorer les performances est plus que bienvenue. Car l'une des faiblesses les plus importantes de PHP est bien là : les variables sont typées dynamiquement, et pour assumer cela, PHP doit constamment vérifier et corriger les types des variables pour que tout fonctionne. Cela ralentit considérablement le déroulement des scripts, et de nombreux bugs inattendus y trouvent leur source. Alors, sans rien enlever à l'agilité de PHP, pourquoi ne pas l'étendre pour permettre au programmeur d'utiliser des types statiques lorsqu'il le souhaite ? Charge à lui de le faire le plus systématiquement possible pour faire des scripts plus efficaces et plus sûrs. Cette idée-là me paraît à retenir.

1 Un enthousiasme à modérer

Une autre idée qui suscite beaucoup d'espoir est l'utilisation d'un compilateur JIT (*Just-In-Time*), compilant les scripts à la volée lors de leur premier appel et stockant le résultat quelque part pour un rappel plus rapide. Là aussi, un gain de performance est attendu. Pourtant, un PHP standard couplé avec un cache APC (*Alternative PHP Cache* : http://fr2.php.net/manual/fr/intro.apc.php) aura un fonctionnement assez similaire, même si j'ignore les détails éventuellement cruciaux des deux implémentations. APC sert justement à **ne pas** ré-interpréter un script déjà exécuté, son code compilé étant justement stocké en cache. Les deux souffrent du même handicap, dû à la nature faiblement typée de PHP : si un script a été compilé une première fois avec une variable d'un certain type, et si la seconde fois, elle est d'un autre type, la compilation est à refaire... Et il faut le détecter en amont, donc surveiller les types de variables avant d'exécuter les scripts compilés ou de procéder à une nouvelle compilation. Dans tous les cas, un tel fonctionnement, s'il est souvent plus rapide qu'une interprétation systématique, ne parvient pas à égaler celui des langages compilés.

Il est vrai que dès lors, il est encore plus intéressant de permettre au développeur d'utiliser des variables à type statique. Le compilateur JIT n'aura pas à re-compiler les scripts Hack, à moins qu'ils n'aient été modifiés. La compilation est donc plus rare, et la surveillance à exercer moindre. Mais pourquoi cela n'aurait-il pas été possible de l'implémenter en s'appuyant, quitte à les modifier, sur les outils standards de PHP ?

Et là, j'en arrive à ce qui représente à mes yeux le principal handicap technique de Hack : il ne fonctionne qu'avec HHVM (*HipHop Virtual Machine*), une machine virtuelle donc, qui fait serveur web. Hack et HHVM sont devenus des logiciels libres (licence BSD) à peu près en même temps, et vivent une histoire commune. Ils sont comme qui dirait, inséparables. Donc, si vous voulez utiliser Hack, vous devez adopter HHVM. Vous ne pourrez plus utiliser votre bien aimé Apache, votre svelte Lighttpd ou votre sportif Nginx (Qui a dit IIS ? Vous là-bas au fond ? Prenez la porte !). J'entends déjà mon administrateur réseau hurler. Bref, voilà un logiciel libre qui semble bien vouloir vous enchaîner. Y a-t-il une nécessité technique à cela, hors la fainéantise de tester plusieurs plateformes ? Les mauvaises langues y verront sans aucun doute la volonté d'imposer une solution technique globale, créant une dépendance technique... Pour ma part,

je crains surtout qu'une telle uniformité soit nuisible en termes de sécurité : une faille dans l'un ou l'autre produit toucherait de la même façon, et avec la même force, tous leurs utilisateurs dans le monde. Bref, il y a maintes raisons pour lesquelles une telle contrainte me paraît inacceptable.

Un autre élément incite encore à plus de prudence. J'ai écrit plus haut que l'idée de Hack était de permettre au développeur PHP d'utiliser des variables à type statique avec son joujou préféré. Hack affiche donc une volonté **d'étendre** PHP, mais la réalité est différente : Hack est d'abord **une ré-implémentation** de PHP. Or, la méthode de développement de PHP n'a pas été conçue pour permettre une ré-implémentation et du coup, des spécifications n'ont pas été produites, comme pour d'autres langages. Il n'est donc pas possible aux développeurs de Hack de s'y référer pour vérifier la conformité de leur code. Dans les faits, cela rend impossible la réalisation de tests unitaires comparant rigoureusement le comportement de Hack avec celui attendu de PHP. Le moyen le plus rigoureux qui reste est de comparer PHP et Hack à l'usage, de voir si pour des scripts donnés dans des contextes donnés, ils adoptent le même comportement. Et le meilleur moyen d'y parvenir est d'utiliser les tests écrits par les développeurs de frameworks PHP pour leur propre travail et d'examiner si les résultats de ces tests sont identiques selon qu'on utilise Hack ou PHP. Et là, l'argumentation de Hack me paraît quelque peu arrogante, car la méthode adoptée est légèrement différente : on utilise les tests des frameworks pour voir s'ils réussissent avec Hack. Comme si l'ambition de Hack n'était pas d'être conforme à PHP, mais d'être le mieux adapté possible à son usage, éventuellement mieux que lui. De là à ce qu'un jour les développeurs PHP soient amenés à faire des tests multiples pour déterminer quel est l'interpréteur utilisé, comme on le fait encore sous JavaScript, il n'y a qu'un pas. Par ailleurs, que valent ces tests pour couvrir l'ensemble de PHP ? N'y a-t-il pas pour chacun des présupposés, des situations récurrentes qui sont mieux testées que d'autres, le fonctionnement avec MySQL, par exemple, plutôt que le fonctionnement avec PostgreSQL ou d'autres bases de données propriétaires ? Quel est le risque pour le développeur PHP de se prendre les pieds dans le tapis dans les zones d'ombre laissées par ces tests ? Pour information, à l'heure où j'écris ces lignes, le taux de réussite des tests par Hack avec les principaux frameworks PHP est de 96,39 % (http://hhvm.com/frameworks). Nulle part n'est indiqué le taux de réussite avec un PHP standard. Mettons qu'il soit de 100 %...

Le choix de partir des sources existantes de PHP pour y ajouter des éléments et pour les améliorer aurait pu être fait. Il ne l'a pas été, sans doute y avait-il quelques raisons. Mais si PHP avait été simplement étendu et amélioré, certains risques auraient été évités : implémentation incomplète de l'état actuel de PHP ; difficultés à suivre les évolutions futures de PHP, toujours non-spécifiées, divergences inattendues des comportements (des bugs dans PHP, des bugs dans Hack, nécessairement différents) ; mise à mal de l'homogénéité des instances de PHP, qui fut toujours l'une de ses forces. Je reviens un instant sur l'idée de l'implémentation. L'API historique de connexion de PHP à MySQL est depuis longtemps maintenant dépréciée, même si elle est encore très fréquemment utilisée : il est recommandé d'utiliser PDO ou MySQLi. D'après le blog de HHVM (http://hhvm.com/blog/3689/implementing-mysqli), cette dernière n'a été implémentée dans Hack qu'en février dernier et devait être incluse dans la HHVM 2.5.0... qui a disparu de la roadmap (https://github.com/facebook/hhvm/wiki/Release-Schedule), la précédente étant la 2.4.0 et la suivante la 3.0.0 au moment où j'écris ces lignes. Je ne peux que m'interroger sur les autres bibliothèques.

Par ailleurs, à partir des sources de PHP, toute la communauté aurait pu y trouver des avantages : amélioration et extension du code profitant à tous les développeurs PHP, pas de problème de compatibilité avec la version officielle puisque Hack aurait alors la forme d'une contribution, pas de divergence de comportement des deux versions à craindre. Il est vrai que dans ce cas, il aurait fallu débattre avec la communauté et obtenir un consensus, ce qui peut paraître lourd. Facebook a préféré faire le choix d'un *breaking change*, d'un changement par rupture. Et pour lui donner de meilleures chances de succès, de l'appuyer avec sa force de communication.

Le tour de force est en train de réussir. À terme, quelles en seront les conséquences ? Il est sans doute impossible de les anticiper toutes. Mais il y en a au moins une qui paraît évidente : un *fork*, une séparation en deux groupes, de la communauté PHP. Et comme souvent dans ces cas-là, cela ne se fera pas sans rancune, ni regret.

2 Quel était le problème déjà ?

Pour bien comprendre à quel problème Hack se veut une réponse, revenons brièvement sur l'histoire de PHP. Ce langage a été créé en 1994 par Rasmus Lerdorf, qui

l'avait alors écrit en Perl pour recueillir des informations sur les visiteurs de son CV en ligne. Il l'a ensuite étendu, enrichi et réécrit en C, avant de le rendre public en 1995 sous le nom de PHP/FI acronyme pour *Personal Home Page / Form Interpreter*. À partir de là commença la grande aventure. Tout cela est très bien expliqué sur Wikipédia. La bonne idée est alors d'étendre PHP par l'intégration des bibliothèques C qui semblent les plus utiles : le langage évolue ainsi vite, offre une palette énorme d'instruments faciles à utiliser, même pour le néophyte. Ce qu'il faut retenir ici, c'est non pas le peu d'ambition initiale qui était celle de PHP et qui le rend inadapté aux projets de grande envergure, mais son aspect de boîte à outils universelle.

Ce péché originel se manifeste à nouveau à d'autres moments de l'histoire de PHP : par exemple, lorsque fut abordée la question de savoir s'il fallait implémenter les arguments nommés dans PHP, offrant ainsi la possibilité de passer des arguments par noms dans n'importe quel ordre plutôt que dans un ordre prédéterminé ; il fut décidé que les développeurs PHP n'en avaient pas besoin et qu'il était bien assez puissant comme ça. Quelques années plus tard, on s'aperçoit que, de plus en plus souvent, de nouvelles fonctions sont écrites en prenant comme argument... un tableau associatif dont les éléments constituent de véritables arguments nommés pour la fonction !

Dans un autre registre, le manque d'ambition de PHP se manifeste dans une certaine inconsistance des paramètres des fonctions standards. Comparons deux fonctions qui servent toutes deux à chercher un élément dans un ensemble et à donner sa position : `strpos` et `array_search`. `strpos()` cherche une chaîne de caractères dans une autre. `array_search()` cherche une valeur parmi les valeurs d'un tableau. Les deux fonctions attendent deux arguments : l'élément à rechercher et l'ensemble dans lequel s'effectue la recherche. Mais alors que `strpos()` attend en premier l'ensemble et en second l'élément, pour `array_search()`, c'est l'inverse ! Et on pourra trouver d'autres plaisanteries du même genre, qui font que la fonctionnalité d'affichage de la signature d'une fonction qu'on vient d'écrire, fonctionnalité offerte par tous les bons IDE, se révèle un outil indispensable pour le développeur PHP...

À ce manque d'ambition a succédé un mouvement contraire : les développeurs de PHP se sont montrés soudain trop ambitieux. Après la sortie de PHP 5 en 2004, PHP 6 fut lancé l'année suivante... Mais après plusieurs reports, PHP 6 fut abandonné en 2009 avec la sortie de PHP 5.3. L'un des grands objectifs de PHP 6 était d'intégrer profondément l'Unicode et en particulier, d'utiliser UTF-16 pour le moteur. La tâche s'est révélée trop compliquée. PHP 5.3 intègre les changements prévus pour PHP 6 les plus attendus par la communauté. Attente qui avait paru bien longue, puisqu'elle avait tout de même duré cinq ans !

Durant ces années, PHP a été incapable d'évoluer comme langage. Mais comme il avait déjà conquis le Web, il en fallait plus que cela pour que la communauté de ses utilisateurs, toujours plus large et enthousiaste, morde la poussière ! Et PHP a donc évolué dans son utilisation, et on a vu apparaître des frameworks de plus en plus complets et sophistiqués, rivalisant d'ingéniosité. Là, les choses semblaient vouloir bien évoluer, avec des cycles de développement et des projets mieux maîtrisés.

Mais à mesure qu'ils prenaient en volume, ces frameworks se trouvèrent et sont toujours tous confrontés à des difficultés liées à la nature interprétée du langage : à chaque consultation de page, il faut recharger la configuration, l'application et son framework. Le langage évolua pour permettre certaines astuces minimisant ce temps de rechargement : possibilité d'un cache APC stockant les scripts sous forme compilée, mais devant surveiller l'évolution des variables pour effectuer une recompilation si nécessaire ; chargement automatique des fichiers de classes au besoin... Mais reste toujours cette impossibilité de produire un code compilé une bonne fois. Contre-performance à comparer avec Python, langage partageant bien des points communs avec PHP, mais qui est fortement typé et, à partir de spécifications, permet des implémentations différentes : celles-ci se montrent capables de produire des fichiers compilés en bytecode pour la *Python Virtual Machine* (les fichiers `.pyc`), ou même pour la *Java Virtual Machine*. Bien que je n'en connaisse pas d'implémentation, Python pourrait donc être compilé au sens classique du terme et donc, se montrer performant. PHP en est incapable, à moins de revoir profondément son fonctionnement. Ce manque de performance, lorsqu'on s'appelle Zuckerberg, constitue un manque à gagner énorme.

3 Les alternatives à Hack

Mais il serait naïf de penser qu'un problème qui touche autant d'informaticiens ne connaisse que la solution proposée par Facebook, même si c'est celle qui fait le plus parler d'elle en ce moment.

Pour ma part, j'ai connaissance d'au moins deux autres solutions que je vais vous présenter brièvement (peut-être y reviendrai-je plus tard dans des articles détaillés). Elles ont des points communs, dont l'un en particulier me paraît séduisant : elles renouent avec l'esprit d'origine de PHP, ce côté « boîte à outils », où PHP est conçu comme un moyen rapide d'utiliser, de manipuler, de ré-agencer des éléments plus lourds écrits dans un langage compilé, plus efficace. Il s'agit donc de solutions qui visent à faciliter la création d'extensions à PHP, extensions compilées, chargées une seule fois au moment du démarrage du serveur Web et toujours disponibles en mémoire et donc, considérablement plus performantes que des frameworks écrits en PHP.

Le premier est **Zephir** (http://zephir-lang.com), qui se définit comme un nouveau langage rassemblant le meilleur de PHP et du C, destiné exclusivement à l'écriture d'extensions à PHP. Techniquement, il est fortement typé, se compile vers le C, puis de là, est compilé en langage machine. L'équipe à l'origine de ce projet s'est fait d'abord connaître avec Phalcon (http://phalconphp.com), le seul framework PHP écrit en C à ma connaissance.

Le second est **PHP-CPP** (http://www.php-cpp.com), qui est une librairie C++ destinée à faciliter le développement d'extensions pour PHP. Idéale pour un développeur C++ qui voudrait utiliser son langage préféré avec PHP : le développement d'extensions étant par ailleurs réputé compliqué et mal documenté. Idéale aussi pour le développeur PHP qui voudrait se mettre au C++, lui permettant d'aborder ce langage dans un contexte familier.

Conclusion

Hack a un intérêt technique incontestable, mais son lancement se fait avec une telle volonté de rupture avec l'équipe PHP que, s'il réussit, il en résultera une séparation de la communauté des utilisateurs PHP. Pourtant, en restant dans l'esprit de PHP, et en prolongeant ce qui a fait son succès, Zephir et PHP-CPP montrent qu'il est possible de répondre au problème des performances sans rupture. Il reste à explorer cette nouvelle direction, la plus prometteuse depuis que PHP est PHP. ■

PARLEZ ET ÉCRIVEZ EN FRANÇAIS !

par ***Dr KissCool*** *[Attention au deuxième effet...]*

Vous l'avez sans doute constaté en discutant avec des collègues : le jargon informatique est incompréhensible ! Et, pire que tout, il s'agit de termes en anglais ! Oui, vous avez bien lu : en anglais ! Heureusement qu'il existe des dictionnaires pour remettre les gens dans le droit chemin...

L'autre jour, alors que dans mon infinie bonté je tentais encore de transmettre quelques-unes de mes (très nombreuses) connaissances à notre rédacteur en chef, je me suis aperçu que Tristan Colombo – car souvenez-vous que Denis Bodor est tombé en disgrâce – donc disais-je, je me suis aperçu que Tristan ne comprenait pas un traître mot de ce que je pouvais dire... Et je dois vous avouer que son discours, bien que je le comprisse grâce à mes immenses facultés d'adaptation (notez l'emploi de l'imparfait du subjonctif, peut-être une première dans ce magazine !), était pour le moins assez obscur. Voici donc, pour vous lecteurs, des morceaux choisis de cette discussion.

1 Le noir parler

Que les âmes sensibles se préparent à lire ici un langage vulgaire, grossier, dont les intonations gutturales sont sans doute tombées dans l'oubli. Il s'agit du jargon employé par Celui-que-l'on-ne-nomme-pas (vous aurez compris de qui je parle). Le parallèle avec un certain livre **[1]** était tellement frappant que j'ai décidé de m'en inspirer dans cet article (oui, je sais, ma culture vous impressionne...). Les précautions d'usage ayant été énoncées, rentrons dans le vif du sujet avec cette rencontre ayant eu lieu il y a quelques semaines :

J'ai vu le listing de ton dernier script sur ton blog : c'est encore du PHP ! En plus, je ne sais pas si tu es le webmaster de ton site, mais certaines news ressemblent à des hoaxes et puis, l'URL www.le-grand-dr-kisscool.com tu ne trouves pas que ça fait un peu trop ?

Ben quoi ? Elle ne te plaît pas mon adresse réticulaire **[2]** ?

Ton adresse quoi ?!?!?

Réticulaire. R-É-T-I-C-U-L-A-I-R-E. Réticulaire.

Tu te sens bien ? Tu es malade ?... Enfin, je veux dire, plus que d'habitude ?

Ah, bien sûr, Môssieur le rédacteur en chef ne comprend pas ce qu'on lui dit et forcément ce sont les autres qui déraillent ! Je reconnais bien là les pratiques de Môssieur le rédacteur en chef !

Bon ok... calme toi... Et c'est quoi alors ton adresse articulaire ? L'adresse d'un kiné ? Moi je te parlais de ton URL dont le nom me semblait...

Mais c'est pas vrai ! L'adresse **réticulaire**... Ce que toi tu appelles... humm ça me fait mal de le dire... ce que tu appelles une URL.

Ah d'accord ! Et pourquoi tu changes le nom ? Tu trouves ça amusant ? Bon enfin, passons, sinon on va y passer la journée... Donc ton adresse polaire ne te choque pas ?

C'est toujours réticulaire, mais la réponse est non.

Soit... Je vois qu'on est encore dans un bon jour... Bien, bien. Et pour les news de ton blog qui ressemblent à des hoaxes ?

Les articles de forum **[3]** que je publie **[4]** dans mon bloc-notes **[5]** n'ont rien à voir avec des canulars **[6]** ! Un canular ça transite par courriel **[7]**.

Je ne te parle pas de ton bloc-notes ! Tu écris ce que tu veux dedans, ça ne me regarde pas. Et au passage, il est hors de question de le publier ton bloc-notes ! Je ne comprends rien à ce que tu dis !

Et c'est toi le rédacteur en chef ? Il va falloir que tu apprennes à parler français ! Un bloc-notes c'est ce que tu appelles un blog et on publie dedans des articles de forum...

Moi, dans un bloc-notes, comme son nom l'indique, je mets des notes... C'est juste la définition d'un bloc-notes...

Et en quoi le listage **[8]** de ma macrocommande **[9]** en PHP t'a-t-il choqué ?

?!?!? En rien... en rien...

Voilà une illustration parfaite (forcément, puisque c'est moi qui l'ai choisie) du fait que les adeptes du noir parler sont socialement inadaptés et ne peuvent communiquer qu'entre eux.

2 Le sindarin

Voici un deuxième exemple où cette fois-ci c'est mon propos de départ qui crée la confusion chez Celui-que-l'on-ne-nomme-pas :

Lors de mon dernier amorçage, je suis resté coincé sur la fenêtre d'attente. Il y avait visiblement un gros bogue et du coup, il m'a fallu utiliser mes fichiers de secours. Malgré une recherche dans mon logiciel de navigation et sur Twitter avec différents mots-dièses, je n'ai rien trouvé ! Auparavant, j'ai remarqué que je recevais beaucoup de courriels d'arroseurs et pas mal de tentatives de filoutage. Si tu pouvais lâcher ton manche à balai deux secondes pour m'aider...

C'est reparti... je ne comprends rien ! Et puis je ne suis pas en train de balayer, là je travaille.

Je croyais que tu testais un jeu ?

Ben oui.

Donc tu utilisais bien un manche à balai ?

Ben non. J'ai un bête joystick **[10]**.

Oui, c'est bien ce que je disais. Et donc, pour mon problème ?

Je ne sais pas. Visiblement tu as booté **[11]**, tu es resté coincé sur le splash screen [**12]** et maintenant, tu cherches des bogues de châtaignes ?

Un bogue ! Une erreur quoi !

Ah ! Tu veux parler d'un bug **[13]**. À l'origine ça s'appelle comme ça, car le premier problème rencontré en informatique était dû à un petit insecte mort dans les entrailles de la machine. Et un insecte, en anglais, ça se dit *bug*. Donc, si tu veux jouer à parler français, on n'a qu'à dire que l'on cherche un cafard.

Non. Dans le dictionnaire c'est bogue : B-O-G-U-E !

Oui... et il y a aussi « mot-dièse » dans ton dictionnaire... Autant un hashtag **[14]** on comprend tout de suite que ça vient de *hash* et de *tag, qu'il s'agit d'un mot-clé*, autant un « mot-dièse » c'est vraiment n'importe quoi. *Hash* en traduction littérale c'est un croisillon (le fameux dièse)... sauf que pour les américains, ce symbole sert aussi à désigner l'abréviation du terme « numéro », d'où l'idée de mot-clé que l'on perd totalement dans mot-dièse. Pourquoi pas un « mot-arobase-mot » pour une adresse de mail ?

Eh bien justement, puisque tu en parles, dans une adresse de courriel, on ne doit pas dire « tristan at gnulinuxmag.com » mais « tristan arrobe gnulinuxmag.com » **[15]**.

Ben je ne mets pas de robe...

Tu es irrécupérable ! Bon et pour mon problème alors ?

Alors là... je pense que c'est un coup des jardiniers malicieux.

Des jardiniers ?!?

Oui. Tu as bien dit que c'était des arroseurs, non ?

Ouh là... je... je pense que je vais traduire avant de m'énerver. Heureusement que je suis capable de comprendre ton langage horrible. Donc : j'ai reçu beaucoup de courriels de « spammers » **[16]** et pas mal de tentatives de « phishing » **[17]**.

Et c'est le seul problème que tu as rencontré ? Tu arrives à imprimer par exemple ?

Mon encre en poudre noire est vide...

Ah ben forcément... Et tu utilises une sarbacane pour la projeter ta poudre, ou tu as quelque chose de plus évolué ?

Mais c'est incroyable de ne rien comprendre à ce point ! L'encre en poudre c'est ce que tu appelles « toner » **[18]** ! C'est vraiment impossible de discuter avec toi ! Là, je sens que je suis en train de m'énerver...

Pushdug KissCool-glob u bagronk !

Hein ?!?! Ah ! Là, l'œil de Denis entouré de flammes ! Non, laissez-moi partir !

Notre discussion s'est arrêtée ici, préférant prendre mes jambes à mon cou pour résoudre mon problème seul plutôt que de m'emporter... Surtout que j'ai vraiment eu la confirmation que Linux Mag était aux mains d'une sorte de secte aux pratiques occultes et terriblement dangereuse. Je suis sûr qu'ils ont envoyé des émissaires, des reptiles gluants et rampants pour me retrouver, je les sens, ils sont proches.... ahhhh !

Conclusion

Ne sombrez pas vous aussi du côté obscur de la force du Milieu informatique. Le noir parler est un langage impur, bestial, qui ira jusqu'à corrompre vos réflexions. Utilisez plutôt le Sindarin ou l'un de ses dialectes tel que le français. Si vous avez des doutes sur le bon terme à employer, rendez-vous sur http://www.culture.fr/franceterme où la lumière vous touchera de sa grâce.

Et n'oubliez pas :

« *Il n'y a pas de mauvais langage... Il n'y a que de mauvais développeurs* ». ■

Références

[1] Tolkien J. R. R., « Le seigneur des anneaux », éd. Christian Bourgeois, 2003 (éd. originale 1954-1955).

[2] JO du 16 mars 1999 : Adresse réticulaire = URL
Dénomination unique à caractère universel qui permet de localiser une ressource ou un document sur l'Internet, et qui indique la méthode pour y accéder, le nom du serveur et le chemin à l'intérieur du serveur.

[3] JO du 16 mars 1999 : Article de forum = News
Document similaire à un message électronique, destiné à alimenter un ou plusieurs forums.

[4] JO du 7 juin 2007 : Publier = Post
Introduire un article ou une contribution sur un forum ou dans un groupe de discussion.

[5] JO du 20 mai 2005 : Bloc-notes = Blog
Site sur la Toile, souvent personnel, présentant en ordre chronologique de courts articles ou notes, généralement accompagnés de liens vers d'autres sites.

[6] JO du 20 mai 2005 : Canular = Hoax
Information fausse transmise par messagerie électronique et incitant les destinataires abusés à effectuer des opérations ou à prendre des initiatives inutiles, voire dommageables.

[7] JO du 20 juin 2003 : Courriel = E-mail
Document informatisé qu'un utilisateur saisit, envoie ou consulte en différé par l'intermédiaire d'un réseau.

[8] JO du 22 septembre 2000 : Listage = Listing
Document produit en continu par une imprimante d'ordinateur.

[9] JO du 20 avril 2007 : Macrocommande = Script
Programme constitué d'une suite de commandes dispensant l'utilisateur de les saisir, et permettant d'effectuer une fonction particulière ou de contribuer à l'exécution d'un autre programme.

[10] JO du 22 septembre 200 : Manche à balai = Joystick
Dispositif de commande à plusieurs degrés de liberté servant à déplacer le curseur d'un écran de visualisation.

[11] JO du 10 octobre 1998 : Amorcer = Booter
Mettre en marche un ordinateur en provoquant l'exécution de l'amorce.

[12] JO du 20 mai 2005 : Fenêtre d'attente = Splash screen
Fenêtre qui s'affiche provisoirement sur un écran pendant l'installation d'un logiciel.

[13] JO du 22 septembre 2000 : Bogue = Bug
Défaut de conception ou de réalisation se manifestant par des anomalies de fonctionnement.

[14] JO du 23 janvier 2013 : Mot-dièse = Hashtag
Suite signifiante de caractères sans espace commençant par le signe # (dièse), qui signale un sujet d'intérêt et est insérée dans un message par son rédacteur afin d'en faciliter le repérage.

[15] JO du 8 décembre 2012
Caractère @ fréquemment employé dans les adresses de courrier électronique pour séparer le nom identifiant l'utilisateur de celui du gestionnaire de la messagerie.

Note :

1. @ est à l'origine le symbole de l'arroba (de l'arabe ar-roub, le « quart »), ancienne unité de capacité et de poids espagnole et portugaise.

   Ce sigle est également utilisé dans les langues anglo-saxonnes, dans des formules telles que « tant de tel article @ tant l'unité ». Dans ces emplois, il est appelé « a commercial », et son tracé, identique à celui de l'arroba, résulterait de la ligature de l'accent grave avec le « a » de la préposition française « à », autrefois d'usage courant dans le commerce international.

2. Lorsqu'une adresse est fournie oralement, @ se dit « arrobe » alors qu'il se dit at en anglais.

[16] JO du 5 mai 2005 : Arroseur = Spammer

[17] JO du 12 février 2006 : Filoutage = Phishing
Technique de fraude visant à obtenir des informations confidentielles, telles que des mots de passe ou des numéros de cartes de crédit, au moyen de messages ou de sites usurpant l'identité d'institutions financières ou d'entreprises commerciales.

[18] JO du 27 décembre 2009 : Encre en poudre = Toner

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 50
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 50
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

QU'EST-CE QU'UN LOGICIEL LIBRE ?
1. HISTOIRE ET DÉFINITION

par ***Nicolas Jullien*** *[Maître de conférences en économie à l'Institut Mines-Télécom, Télécom Bretagne, iSchool]*

Les logiciels libres sont souvent gratuits. Mais les logiciels gratuits ne sont pas toujours libres. Et malgré la gratuité, tout le monde n'utilise pas ces logiciels et certaines entreprises vivent de la vente de ces logiciels et des services associés. Cet article, sur l'histoire et la définition du libre, est le premier d'une série qui se propose de répondre aux questions non techniques sur ce qu'est le libre.

J'enseigne à Télécom Bretagne, une école d'ingénieurs tournée vers les technologies de l'information. La plupart de mes étudiants utilisent Firefox, certains une distribution Linux et une majorité a un smartphone sous Android dans sa poche.

Pourtant, ces adeptes des logiciels libres restent perplexes quand on leur demande d'expliquer ce qu'est un logiciel libre, la différence avec un logiciel gratuit, le pourquoi de la participation à un projet de logiciel libre, etc.

Il est évident que le lecteur de GLMF est sans doute plus au fait de ces différences et peut même être surpris que des étudiants d'une des plus grandes écoles d'ingénieurs de France en sachent si peu sur des logiciels qui dominent leur marché (faut-il rappeler ici que Linux est le noyau du système d'exploitation leader sur les smartphones, Android ?). Mais c'est la même perplexité dans mon autre travail, à l'institut de recherche technologique B<>COM, où il s'agit d'inventer les technologies numériques du futur.

Les discussions enflammées entre partisans de l'« open source » et ceux du « logiciel libre » montrent que ce n'est pas simple. L'histoire nous apprend aussi que les développements collectifs, coopératifs, existent depuis le début de l'informatique, ou plutôt du logiciel, dans les années 1950. La nouveauté du libre ne serait donc pas, seulement, dans la production coopérative. Dans cet article et dans les suivants, je propose de discuter des caractéristiques du logiciel ouvert et libre : de l'organisation et des participants aux communautés de développement ; de la place des entreprises et de leur modèle économique (comment elles gagnent de l'argent, pourquoi elles s'investissent dans ces communautés) ; mais avant tout, de ce qu'est un logiciel libre (ou de code ouvert).

Un petit détour par l'histoire de l'informatique, puis par ce qu'est un logiciel, nous amènera à mieux préciser les caractéristiques d'un logiciel libre et du mouvement du libre.

Au début de l'informatique, dans les années 50, la production informatique était pratiquement exclusivement une production américaine. L'activité informatique était très liée aux projets militaires, aux financements militaires et aux projets de recherche : l'état américain a financé la recherche et le développement de l'informatique à ses débuts et la plupart des machines produites à cette époque l'ont été pour ce « client ». Les machines étaient livrées « nues » par le fournisseur, c'est-à-dire sans programmes. Le fournisseur aidait son client à mettre en œuvre ses machines et à développer les programmes qui lui permettaient de les utiliser. Le logiciel était alors considéré comme un outil et un objet de recherche. Sa production reposait sur des relations de collaboration poussées entre centres de recherche publics et industrie et les principaux utilisateurs de logiciel étaient alors les principaux producteurs (les universités, les grandes entreprises, surtout celles liées à l'industrie de défense et, bien sûr, le principal producteur de machines informatiques, IBM).

L'ensemble des logiciels produits était disponible, libre de droits de propriété ; la notion même de protection intellectuelle du logiciel n'existait pas. La seule façon de protéger un logiciel était de le garder secret, ce qui n'était

pas la stratégie soutenue par les institutions publiques et par les entreprises informatiques émergentes. En effet, la diffusion de logiciels dans le domaine public était directement bénéfique aux producteurs : plus la quantité de logiciels disponibles était grande, plus l'étaient les incitations à acheter un ordinateur. De plus, le département américain de la défense (DoD) a encouragé ces pratiques en finançant des programmes de recherche et de développement coopératifs universités-entreprises. Il a aussi favorisé la diffusion des « bonnes pratiques » de programmation. Enfin, le milieu de la recherche est traditionnellement un milieu favorable à la coopération et à la diffusion d'informations sur les connaissances produites. Finalement, si l'on se place du point de vue des premiers utilisateurs des ordinateurs, ce qui étonne est que cette production coopérative se soit marginalisée alors qu'elle semblait satisfaire l'ensemble des acteurs, producteurs et utilisateurs de logiciels. C'est pourtant ce qui s'est passé.

La raison est assez simple à expliquer : des évolutions techniques, aussi bien sur le logiciel (compilateurs, langages de programmation), que sur les machines (notion de série, notion de système d'exploitation), le développement des usages, ont fait que les logiciels développés dans les entreprises devenaient de plus en plus importants, qu'ils duraient de plus en plus longtemps, mais aussi qu'ils devenaient de plus en plus indépendants de la partie matérielle de la machine.

Si le système de production coopératif est toujours à la base du développement de nombreux logiciels, le lien entre la demande (les utilisateurs) et les producteurs n'est plus direct, mais passe par l'intermédiaire des entreprises. Il y a d'un côté la recherche, coopérative, productrice de prototypes et financée par des subventions ou des commandes publiques et de l'autre l'industrie, qui s'approprie ces prototypes et qui les distribue.

Le corollaire de l'industrialisation de la production du logiciel est que les fournisseurs ont cherché à être rémunérés pour leur travail, et les clients ont souhaité connaître précisément, dans ce qui leur était facturé, ce qui relevait de la machine, du logiciel existant et revendu, et des développements spécifiques, ou du service. Des logiciels standards, ou progiciels sont apparus. Dans tous les cas, la question du qui fait quoi, de la propriété des logiciels s'est posée.

Sans rentrer dans les détails de l'histoire, cela s'est résolu par la création d'un droit de propriété intellectuelle sur le logiciel, adapté du droit d'auteur (ou copyright, aux États-Unis), en 1969. Ce droit permet au producteur de fixer les conditions d'utilisation de sa production, au moyen d'un contrat d'utilisation, la licence : prix pour l'utilisation, durée de cette utilisation (un an, deux ans, la durée de « vie » du logiciel ou de la machine qui le fait fonctionner...), accès ou non au code source du logiciel, droit de le modifier ou non.

Cette tendance s'est affirmée dans les années 90 : ces années correspondent à un relatif désengagement des institutions fédérales américaines du financement des projets logiciels et donc, à une plus grande difficulté pour initier et financer de tels projets. De plus, la recherche prend une orientation plus théorique (algorithmique, tests de stabilité des programmes, etc.). Considérant l'orientation de plus en plus abstraite des projets de recherche et la baisse des commandes publiques, il apparaît normal que les industriels abandonnent la production coopérative, qui n'est plus que l'apanage de la production universitaire. Il semble que le système de production du logiciel se soit transformé pour, finalement, aboutir à un schéma classique : les centres de recherche publics inventent des nouveaux outils et de nouveaux concepts et les industriels les transforment en objets utilisables, en se les appropriant.

Les entreprises se sont spécialisées et se spécialisent toujours, selon un schéma classique dans les industries du service, selon que les utilisateurs (et les clients) sont plus ou moins sensibles au prix, ont des besoins spécifiques, ou sont variés en termes de besoin, et qui est résumé dans la figure 1 (le lecteur pourra appliquer le même schéma pour choisir sa prochaine destination de vacances, par exemple). Certaines cherchent à couvrir l'ensemble du spectre des utilisations (personnalisation), en proposant des offres relativement standards : cela permet de diminuer le coût par utilisateur (car le coût de développement du progiciel est « partagé » entre tous les utilisateurs), même si l'offre ne s'adapte pas exactement aux besoins. D'autres se spécialisent dans une niche (souvent autour de besoins métiers), ce qui permet de mieux adapter le logiciel aux besoins de cette niche, à un coût qui reste raisonnable (si la population d'utilisateurs est moins grande, elle reste suffisante pour partager les coûts de production du logiciel), avec éventuellement une partie d'adaptation du logiciel aux besoins. Enfin, héritières des entreprises qui ont développé les premiers systèmes informatiques pour les banques, les compagnies aériennes, les grandes entreprises, des entreprises se spécialisent dans le développement à façon, l'intégration de briques logicielles existantes (parfois appelées SSII, ici nommées « architectes » car elles construisent une solution, plus ou moins spécifique, pour chaque client) (Fig. 1).

Si on limite le libre à la production coopérative, ou à la gratuité, on ne comprend pas comment il aurait sa place dans ce processus historique.

En fait, ce détour par l'histoire m'a permis de souligner les facteurs expliquant et définissant le logiciel libre : c'est un logiciel, c'est-à-dire qu'il répond à un besoin d'un utilisateur, dont la production a été organisée pour répondre plus efficacement à ce besoin, dans le cadre d'une relation contractuelle permise par le système du droit d'auteur, c'est-à-dire régie par une licence. Parce que les technologies ont évolué (notamment avec la mise en réseau permise par Internet), les besoins ont évolué, et le système de production et la relation contractuelle aussi.

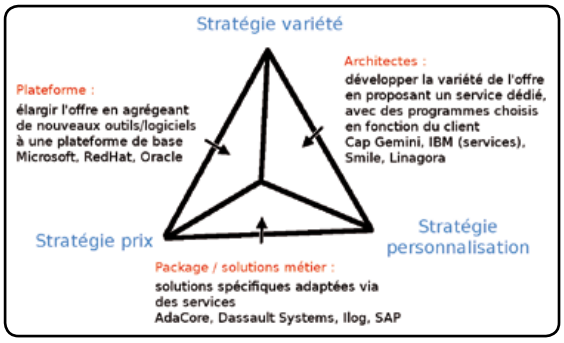


Fig. 1 : Les stratégies de production de logiciel (adapté de DangNguyen et al. [1] et Cusumano [2])

La mise en réseau des machines et des utilisateurs dans et hors des organisations fait que les échanges de données transitent par de nombreux sous-systèmes, inconnus a priori, et que les données produites doivent être lisibles par des progiciels différents (on peut faire l'exercice en imaginant un courriel contenant une pièce attachée ou un lien vers une page HTML échangé entre un rédacteur en chef et un de ses auteurs). De plus, la diffusion des machines vers l'individu, mais aussi la multiplication des terminaux (hier ordinateurs de bureau, portables, aujourd'hui smartphones et tablettes), en plus de renforcer ce phénomène, implique que les logiciels (et plus particulièrement les progiciels) doivent être adaptés aux plateformes, mais aussi aux besoins et aux connaissances de chaque individu, sans perdre l'avantage des économies d'échelle, donc la normalisation des programmes sur lesquels la solution est fondée.

En parallèle de cette évolution, les technologies de programmation ont évolué, elles aussi : l'arrivée des langages de programmation objet (C ++, Java) a permis à des composants logiciels déjà développés d'être réutilisés. Cela a conduit à la notion de « logiciels modulaires » : l'idée est de développer un ensemble de petits programmes logiciels (modules ou de composants logiciels), qui ont chacun une fonction spécifique. Ils peuvent être associés à et utilisés sur n'importe quelle machine, grâce à leurs interfaces de communication normalisées.

Ce qui caractérise l'évolution technologique des logiciels est donc l'interdépendance croissante entre les logiciels, tandis que les composants logiciels qui sont ré-utilisés sont de plus en plus raffinés et spécialisés. Il devient alors logique que certains utilisateurs recherchent des solutions plus ouvertes, qui leur garantissent un plus grand contrôle sur les interfaces d'échange, mais aussi une plus grande souplesse d'adaptation. Par exemple, l'atout d'Internet n'est pas tant d'offrir un « protocole » pour permettre la transmission de données, car cela existait déjà, mais d'en offrir un suffisamment simple et souple, ce qui lui a permis de s'imposer comme un standard pour l'échange.

L'ouverture du code source du logiciel, permet de vérifier plus facilement son fonctionnement, ses interfaces, pour ceux qui en ont les capacités, même si, de nos jours, on utilise le plus souvent les interfaces de programmation (API) comme intermédiaires pour construire le dialogue entre deux logiciels.

Cette ouverture du code, mais surtout l'organisation modulaire et coopérative qui a été mise en place autour des grands projets de développement de logiciel libre, est aussi un avantage dans ce monde où des utilisateurs hétérogènes échangent sans arrêt des données et des informations. En effet, les grands projets de logiciels libres comme Apache, Linux, Mozilla, LibreOffice pour ne citer que les plus populaires, sont en fait constitués de plusieurs sous-projets, spécialisés dans une sous-partie (et une sous-tâche) du logiciel, et organisés par une architecture globale. L'interface avec ces logiciels est aussi facilitée car l'organisation répartie oblige à documenter de façon précise le fonctionnement de chaque module. C'est grosso-modo la même organisation que celle qu'utilise Microsoft. La différence réside dans le fait que n'importe qui peut accéder à cette organisation, facilitant pour l'utilisateur externe l'identification du module qui la concerne et diminuant son investissement pour comprendre le fonctionnement du logiciel, ou pour participer à son évolution. Évidemment, de telles organisations coopératives, virtuelles et mondiales, n'ont été rendues possibles qu'avec la diffusion d'Internet, en même temps qu'elle a changé les besoins. Je reviendrai dans un prochain article sur le fonctionnement de cette organisation, du pourquoi les participants, et notamment les entreprises, s'impliquent.

Enfin, et c'est la dernière innovation du logiciel libre, cette ouverture et cette autorisation de modification sont garanties contractuellement : comme pour tous les logiciels, les logiciels libres viennent avec un contrat, une licence d'utilisation. Il y en a de nombreuses, les plus utilisées étant la GPL (Linux, entre autres), la licence BSD (OpenBSD, FreeBSD) et, pour la France, la licence CeCILL. Comme pour tous les contrats, et toutes les licences, certaines conditions

d'utilisation varient suivant les licences libres. Toutes vous garantissent la libre utilisation, l'accès aux sources et la libre modification du logiciel protégé. Certaines obligent que les utilisateurs qui souhaitent redistribuer le logiciel, éventuellement modifié, le fassent sous les mêmes termes et conditions (la GPL), d'autres non (la BSD). Il existe des guides permettant d'analyser les licences, comme celui proposé par l'INRIA **[3]**, ou le livre de Benjamin Jean **[4]**.

Pour conclure, un logiciel libre est d'abord un logiciel, qui répond à des besoins. Comme tout logiciel, il est protégé par le droit d'auteur et l'utilisateur a des droits et des obligations qui sont résumés dans la licence. Sont appelés « logiciels libres » des logiciels dont la licence garantit à l'utilisateur le fait d'obtenir les sources du logiciel, de pouvoir modifier ces sources et de pouvoir les redistribuer. Ces logiciels se sont diffusés grâce à Internet, qui a généré de nouveaux besoins, pour lesquels des logiciels libres existaient déjà (car souvent issus, comme Internet, de la recherche universitaire). Et l'organisation coopérative de production des logiciels libres les plus utilisés, et les plus complexes, a été rendue possible grâce à la mise en réseau des développeurs. Cette organisation fera l'objet de l'article du mois prochain, avant de parler des modèles économiques des entreprises du logiciel libre. ■

Références

[1] Dang Nguyen G., Leray Y., et Mével O., « Enseignements et prospective des pratiques des entreprises de services », 10e rencontre sur la prospective des métiers : quel management demain ?, 2010.

[2] Cusumano M., « The Business of software: What every manager, programmer, and entrepreneur must know to thrive and survive in good times and bad », Free Press, NY, 2004.

[3] Steer S. et Fitzgibbon M., « Guide d'approche et d'analyse des licences de logiciels libres », INRIA, www.inria.fr/content/download/5892/48431/version/2/file/INRIA_guide_analyse_licences_libres_vf.pdf

[4] Jean B., « Du bon usage des licences libres », Option Libre, Framabook, 2011.

D'autres références pour aller plus loin

[5] Horn F., « L'économie des logiciels », Coll. repères, La Découverte, 2004.

[6] Paloque-Berges C. et Masutti C (éditeurs), « Des logiciels partagés aux licences échangées », Histoires et cultures du Libre, Framabook, 2013.

LES STRUCTURES LINÉAIRES

par ***Tristan Colombo***

Nous stockons toujours des données dans des variables. Celles-ci sont plus ou moins complexes et peuvent prendre la forme de tableaux, listes ou piles. Il n'est pas inintéressant de revoir les structures disponibles, comment les construire et les utiliser.

Lorsqu'il faut mémoriser des données dans un programme, il faut bien sûr les stocker dans des variables. Tous les langages fournissent des types simples tels que les entiers, les caractères ou encore éventuellement les booléens. Viennent ensuite des types dits « complexes », qui sont en fait des structures de données bâties à partir des types simples comme un tableau d'entiers par exemple. Tous les langages vont permettre au développeur de créer ses propres types structurés, qui lui permettront de mieux s'adapter au problème qu'il souhaite traiter. Nous n'utiliserons pas la même structure que nous voulions modéliser une suite de processus dépendant d'une certaine priorité, un jeu de cartes ou la liste des coups possibles dans un jeu de dames.

Parmi les structures possibles, on distingue les structures linéaires des structures de graphe. Nous nous attacherons dans cet article seulement aux structures linéaires, qui sont des structures dans lesquelles chaque élément possède un successeur (sauf bien entendu le dernier).

1 Les tableaux

Un tableau est une suite d'espaces mémoire adjacents pouvant stocker des données d'un même type ou de types différents suivant les langages. L'accès aux éléments se fait à l'aide d'un indice qui indique à quelle adresse mémoire aller lire la donnée. Un tableau a une taille fixe et ne peut donc stocker plus d'éléments que ce pourquoi il a été défini au départ. Si l'on souhaite absolument ajouter plus d'éléments dans le tableau, il faut alors le recopier intégralement dans un tableau plus grand.

Pour un tableau `T` on note `T[i]` le `i + 1` ème élément. Ainsi, un tableau `T` de taille `n` contiendra les éléments `T[0]`, `T[1]`, ..., `T[n - 1]`. La définition d'un tableau se fera de manière fort diverse suivant les langages. En C, on devra ainsi indiquer précisément la quantité de mémoire que l'on souhaite affecter au tableau :

```
int *tab;
int n = 100;
tab = malloc(n * sizeof(int))
```

Normalement, il faut même aller encore plus loin en vérifiant la valeur retournée par `malloc` qui indique si l'allocation a pu avoir lieu ou non.

En Python, les choses se passent différemment, de manière plus élégante... mais aussi plus dangereuse quand on ne sait pas ce que l'on fait. Voici donc un exemple d'ajout d'un élément dans un tableau (appelé « liste » en Python) : la première version déclare un tableau de `n` éléments et en ajoute un `n + 1` ème et la deuxième version déclare directement une liste de `n + 1` éléments. Pour chaque version, nous estimerons le temps de calcul à l'aide de la fonction magique `%timeit` dans IPython :

```
In [1]: def version_1():
   ...:     l = [1, 2, 3, 4]
   ...:     l += [5]
   ...:

In [2]: %timeit version_1()
1000000 loops, best of 3: 295 ns per loop

In [3]: def version_2():
   ....:     l = [1, 2, 3, 4, None]
   ....:     l[4] = 5
   ....:

In [4]: %timeit version_2()
10000000 loops, best of 3: 176 ns per loop
```

La version 1 est 1,6 fois plus lente que la version 2. Petite parenthèse pour satisfaire les Pythonistes curieux, même en utilisant la méthode **append** on perd du temps :

```
In [5]: def version_3():
   ....:     l = [1, 2, 3, 4]
   ....:     l.append(5)
   ....:

In [6]: %timeit version_3()
1000000 loops, best of 3: 243 ns per loop
```

Cette version n'est certes plus que 1,4 fois plus lente, mais plus lente quand même qu'une affectation directe de la liste à sa taille finale.

Les opérations fondamentales dans un tableau sont l'insertion, la lecture, la suppression et le test de vacuité (le tableau est-il vide ou pas ?). Dans les parties suivantes, nous supposerons que les tableaux sont remplis progressivement du début à la fin sans laisser de case vide, sinon les cas seraient trop simples.

1.1 Insertion

L'ajout d'un élément à la suite des éléments déjà présents se fait en temps constant (O(1)) en indiquant simplement l'indice où insérer le nouvel élément (voir figure 1).

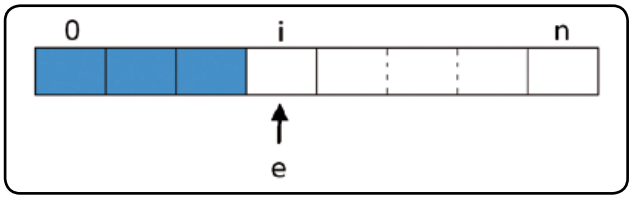


Fig. 1 : Ajout d'un élément en fin de tableau

Si vous souhaitez ajouter un élément **e** à une position donnée **i**, entre d'autres éléments, l'opération sera cette fois plus longue, car il faudra décaler d'une case tous les éléments se trouvant entre la position **i** et la fin des données. La figure 2 illustre ce cas qui s'exécute cette fois en temps linéaire dans le pire des cas (O(n)).

Supposons qu'un tableau **T** de taille **n** contienne **j + 1** éléments stockés dans les cases **0** à **j** (comme sur la figure 2). Voici l'algorithme permettant d'insérer un élément **e** en position **i**, où **0 <= i <= j + 1** :

```
Pour k variant de j à i - 1 Faire
    T[k] <- T[k + 1]
T[i] <- e
```

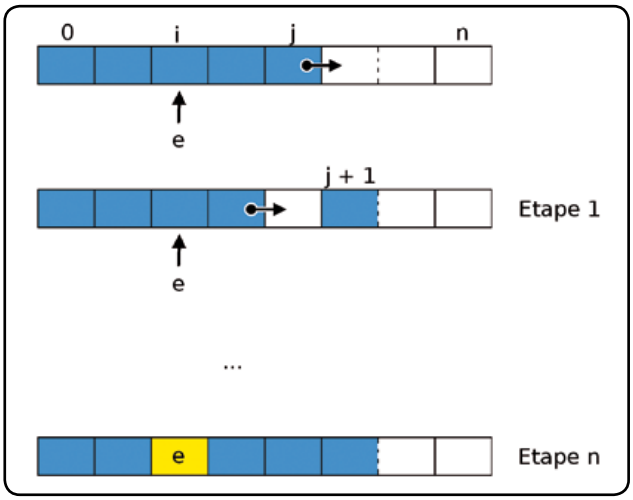


Fig. 2 : Insertion d'un élément à une position donnée dans un tableau

Attention, la boucle s'effectue ici en sens descendant de **j** à **i - 1**. Ainsi, lorsque **k = i - 1**, nous finissons le décalage de tous les éléments se trouvant à droite de la position **i** avec **T[i - 1] <- T[i]**.

À propos de la notation grand O

La notation grand O employée pour indiquer le temps d'exécution d'un algorithme a été introduite en 1894 par Paul Bachmann **[1]**. Cette écriture permet de simplifier la présentation d'une approximation en supprimant des informations détaillées qui parasiteraient l'information principale. Par exemple, avec cette notation on peut écrire **1/2 n2 + n = O(n2)**. Le signe d'égalité remplace l'approximation, l'information pertinente est bien le **O(n2)**.

Cette notation est utilisée pour indiquer la complexité d'un algorithme où O(n) signifie « au pire n opérations ».

1.2 Lecture

La lecture se fait naturellement en temps constant (O(1)), puisqu'il suffit d'indiquer l'indice correspondant à l'élément souhaité pour l'obtenir. Par exemple, l'obtention du **i**ème élément du tableau **T** se fait par **T[i]**.

1.3 Suppression

La suppression du dernier élément se fait en O(1). Par contre, si nous souhaitons supprimer un élément positionné à l'indice **i** pour un tableau contenant **j** éléments (**j > i**),

alors il faudra effectuer l'opération inverse de l'insertion et décaler toutes les cases de `i + 1` à `j` vers la gauche, tout en effaçant le contenu de la case `j`. Cette opération sera elle aussi en O(n).

```
Pour k variant de i à j - 1 Faire
    T[k] <- T[k + 1]
T[j] <- void
```

1.4 Test de vacuité

Un tableau `T` est vide s'il ne contient aucun élément... Il suffit donc de tester la valeur de `T[0]` (opération en O(1)) :

```
Fonction EstVide(T) -> Booléen
    Si T[0] est vide alors
        Retourner Vrai
    Sinon
        Retourner Faux
```

1.5 Résumé

Voici un tableau résumant les coûts de chacune des opérations sur les tableaux :

Opération	Coût en temps
Insertion en fin	O(1)
Insertion à une position donnée	O(n)
Lecture	O(1)
Suppression en fin	O(1)
Suppression à une position donnée	O(n)
Test de vacuité	O(1)

2 Les piles et les files

Les piles et les files sont des structures linéaires qui se distinguent par la façon dont les éléments seront insérés et supprimés :

- dans une pile (*stack* en anglais), les insertions et suppressions sont réalisées du même côté de la liste (soit au début, soit à la fin). Une pile fonctionne suivant le modèle LIFO (*Last In First Out* ou dernier entré premier sorti). Une pile d'assiettes est un bon exemple de pile. Lorsque vous posez une assiette, ce sera toujours sur le dessus de la pile et lorsque vous en retirez une ce sera également sur le dessus de la pile et donc, elle correspondra à la dernière que vous ayez posée.
- dans une file (*queue* en anglais), toutes les insertions sont faites d'un côté de la liste et toutes les suppressions de l'autre côté. La file fonctionne suivant le modèle FIFO (*First In First Out* ou premier entré premier sorti). Prenons l'exemple de voitures arrivant à un péage : le premier arrivant sera le premier à pouvoir payer et partir ; il s'agit donc d'une file. Notez qu'il existe un cas particulier de file nommé file à double fin (*deque* pour *double-ended queue*). Cette file peut fonctionner indifféremment depuis la gauche ou la droite.

La figure 3 illustre la structure d'une pile et des différents types de files. Ces structures peuvent être très simplement implémentées en utilisant un tableau.

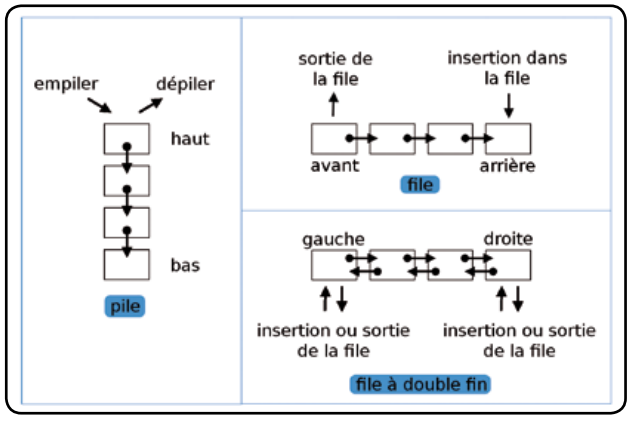


Fig. 3 : Structures de piles et de files

2.1 Les piles

Il suffit de trois opérations pour pouvoir manipuler une pile :

- le test de vacuité ;
- une fonction permettant d'empiler un élément ;
- une fonction permettant de dépiler un élément (récupération de la valeur et suppression de la pile).

2.1.1 Les opérations

2.1.1.1 Test de vacuité

Soit une pile (tableau) `P` de taille `n`, si `P[0]` est vide alors la pile est vide.

```
Fonction EstVide(P) -> Booléen
    Si P[0] est vide alors
        Retourner Vrai
    Sinon
        Retourner Faux
```

L'opération se déroule bien sûr en temps constant (O(1)).

2.1.1.2 Empilement

Pour que les opérations d'empilement et de dépilement soient optimales, il faut considérer que l'indice 0 du tableau représente le bas de la pile et que nous conservons un indice indiquant où se situe l'élément sur le haut de la pile. Soit une pile **P** de taille **n** et **i** l'indice désignant le haut de la pile. Pour empiler un élément **e**, il faut effectuer les opérations suivantes :

```
i <- i + 1
P[i] <- e
```

Là encore, la complexité en temps est en O(1).

2.1.1.3 Dépilement

Pour dépiler un élément dans une variable **e** en suivant les mêmes notations que pour l'empilement, il s'agira simplement des opérations inverses (complexité en O(1)) :

```
e <- P[i]
i <- i - 1
```

2.1.2 Un exemple pratique de pile

Nous ne pourrons pas utiliser Python pour cet exemple... tout est déjà défini dans les listes ! Pour empiler, il suffit d'utiliser la méthode **append** et pour dépiler, la méthode **pop** :

```
>>> pile = [1, 2, 3, 4]
>>> pile.append(5)
>>> pile
[1, 2, 3, 4, 5]
>>> pile.pop()
5
>>> pile
[1, 2, 3, 4]
```

Passons donc à une implémentation en C utilisant les tableaux. Pour commencer, il faut définir la structure :

```
01: #include <stdio.h>
02:
03: #define MAX 100
04:
05: typedef struct
06: {
07:     int courant;
08:     int elements[MAX];
09: } pile;
```

Ici, il va falloir créer une fonction supplémentaire pour initialiser la position de l'élément courant à -1 et indiquer ainsi qu'il n'y a aucun élément dans la pile.

```
11: void initialise(pile *P)
12: {
13:     P->courant = -1;
14: }
```

Ensuite, il suffit de suivre les algorithmes précédents pour écrire les fonctions de test de vacuité, d'empilement et de dépilement.

```
16: int estVide(pile P)
17: {
18:     if (P.courant == -1)
19:     {
20:         return 1;
21:     }
22:     else
23:     {
24:         return 0;
25:     }
26: }
27:
28: int empile(pile *P, int elt)
29: {
30:     if (P->courant == MAX - 1)
31:     {
32:         return 0;
33:     }
34:     else
35:     {
36:     P->courant = P->courant + 1;
37:         P->elements[P->courant] = elt;
38:     return 1;
39:     }
40: }
41:
42: int depile(pile *P)
43: {
44:     int elt;
45:
46:     if (estVide(*P))
47:     {
48:         return 0;
49:     }
50:     else
51:     {
52:         elt = P->elements[P->courant];
53:         P->courant = P->courant - 1;
54:     return elt;
55:     }
56: }
```

Ces lignes de code ont été écrites de manière à être le plus lisible possible en accord avec les algorithmes précédents. Bien entendu, le code peut être optimisé et les lignes 52 à 54 pourraient par exemple être écrites de la manière suivante :

```
return P->elements[P->courant--]
```

Il ne reste plus qu'à tester notre code :

```
58: int main(void)
59: {
60:     pile P;
61:     int elt;
62:
63:     initialise(&P)
64:
65:     if (estVide(P))
66:     {
67:         printf("La pile est vide !\n");
68:     }
69:
70:     empile(&P, 1);
71:     empile(&P, 2);
72:     empile(&P, 3);
73:     elt = depile(&P);
74:     printf("Element : %d\n", elt);
75:
76:     return 0;
75: }
```

2.2 Les files

Comme pour les piles, il suffit de trois opérations pour pouvoir manipuler une file :

- le test de vacuité ;
- une fonction permettant d'intégrer un élément dans la file ;

- une fonction permettant de sortir un élément de la file (récupération de la valeur et suppression de la file).

2.2.1 Les opérations

2.2.1.1 Test de vacuité

Le test est identique à celui de la pile, puisque la représentation se fait sous forme de tableau et l'opération se déroule en O(1). Soit une file **F** de taille **n**, si **F[0]** est vide alors la file est vide.

```
Fonction EstVide(F) -> Booléen
    Si F[0] est vide alors
        Retourner Vrai
    Sinon
        Retourner Faux
```

2.2.1.2 Insertion dans la file

Pour que les opérations d'insertion et de sortie de la file soient optimales, il faut considérer que l'indice 0 du tableau représente l'arrière de la file et que nous conservons un indice indiquant où se situe l'élément le plus à l'arrière de la file (indice d'insertion) et le plus à l'avant de la file (indice de sortie de la file). Dans notre représentation, au fur et à mesure de l'utilisation de la file nous perdrons de la capacité de stockage... C'est l'éternel dilemme entre optimisation en temps ou en mémoire.

Soit une file **F** de taille **n** , **av** l'indice désignant l'avant de la file et **ar** l'indice désignant l'arrière de la file. Pour insérer un élément **e** dans la file, il faut effectuer les opérations suivantes :

```
ar <- ar + 1
F[ar] <- e
```

La complexité en temps est en O(1).

2.2.1.3 Sortie de la file

Pour sortir un élément de la file dans une variable **e** en suivant les mêmes notations que pour l'insertion, il s'agira simplement des opérations inverses basées cette fois-ci sur l'avant de la file (complexité en O(1)) :

```
e <- P[av]
av <- av + 1
```

2.2.2 Un exemple pratique de file

Là encore, en Python il suffira d'utiliser une liste en appliquant certaines contraintes pour l'insertion et la sortie de la file :

```
>>> file = [1, 2, 3, 4]
>>> file.append(5)
>>> file
[1, 2, 3, 4, 5]
>>> file.pop(0)
1
>>> file
[2, 3, 4, 5]
```

En C, il faut un peu plus d'instructions pour mettre en place une file. Vous verrez que le code est très similaire à celui définissant une pile.

```
01: #include <stdio.h>
02:
03: #define MAX 100
04:
05: typedef struct
06: {
07:     int avant;
08:     int arriere;
09:     int elements[MAX];
10: } file;
```

Par rapport à l'implémentation de la pile, il a fallu ajouter une variable permettant de stocker un indice supplémentaire. Dans la phase d'initialisation, nous avons donc une opération supplémentaire :

```
12: void initialise(file *F)
13: {
14:     F->avant = 0;
15:     F->arriere = -1;
16: }
```

Pour savoir si la file est vide, il faut vérifier si l'indice **avant** désignant la position de l'élément à retirer de la file est supérieur à l'indice **arriere** désignant la position du dernier élément inséré. En effet, si tel est le cas, la donnée se trouvant à l'indice **avant** n'a pas été insérée par notre programme (elle correspond donc à une valeur quelconque) :

```
18: int estVide(file F)
19: {
20:     if (F.avant > F.arriere)
21:     {
22:         return 1;
23:     }
24:     else
25:     {
26:         return 0;
27:     }
28: }
```

L'insertion et la sortie de la file fonctionnent de la même manière que l'empilement et le dépilement dans une pile, à la nuance près que nous utilisons ici une fois l'indice **arriere** et l'autre fois l'indice **avant** :

```
30: int insere(file *F, int elt)
31: {
32:     if (F->arriere == MAX - 1)
33:     {
34:         return 0;
35:     }
36:     else
37:     {
38:     F->arriere = F->arriere + 1;
39:         F->elements[F->arriere] =
elt;
40:  return 1;
41:     }
42: }
43:
44: int sort(file *F)
45: {
46:     int elt;
47:
48:     if (estVide(*F))
49:     {
```

ABONNEZ-VOUS !

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

NOUVEAU ! Abonnez-vous (réabonnez-vous) en ligne sur :

boutique.ed-diamond.com

Vous pouvez ainsi :
- ➧ Avoir accès à votre suivi personnalisé d'abonnement
- ➧ Profiter des promos réservées à nos abonnés
- ➧ Vous réabonner facilement sans interruption d'abonnement

Pour plus d'informations, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

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 newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

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 : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

ABONNEMENT GLMF

➡ **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 **GUIDES !**

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)

NOUVEAU ! Abonnez-vous (réabonnez-vous) en ligne sur :

boutique.ed-diamond.com

Vous pouvez ainsi : ➡ Avoir accès à votre suivi personnalisé d'abonnement ➡ Profiter des promos réservées à nos abonnés ➡ Vous réabonner facilement sans interruption d'abonnement

Pour plus d'informations, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

➡ 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 françaises

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-séries/Guides et je vis en Belgique.
Je coche donc l'offre **T+** (la totale avec tous les Hors-Séries/Guides), puis ma zone (E), le montant sera donc de 415 euros.

Je choisis de régler par :

☐ Chèque bancaire ou postal à l'ordre des Éditions Diamond (uniquement France et DOM TOM)

☐ Pour les règlements par virements, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

Date et signature obligatoires

N°172

```
50:         return 0;
51:     }
52:     else
53:     {
54:         elt = F->elements[F->avant];
55:         F->avant = F->avant + 1;
56:    return elt;
57:     }
58: }
```

Pour tester le code, il faut bien sûr le programme principal :

```
60: int main(void)
61: {
62:     file F;
63:     int elt;
64:
65:     initialise(&F);
66:
67:     if (estVide(F))
68:     {
69:         printf("La file est vide !\n");
70:     }
71:
72:     insere(&F, 1);
73:     insere(&F, 2);
74:     insere(&F, 3);
75:     elt = sort(&F);
76:     printf("Element : %d\n", elt);
77:
78:     return 0;
79: }
```

2.3 Résumé

Voici un tableau résumant les coûts de chacune des opérations sur les tableaux : voir ci-dessous.

3 Les listes chaînées

Plutôt que d'utiliser une structure monobloc telle qu'un tableau et gaspiller de la mémoire pour représenter une liste, nous pouvons définir une structure plus souple appelée liste chaînée. Une liste chaînée est composée de maillons liés les uns aux autres par un champ (en général nommé **suivant**) et qui contiennent un champ de données qui peut être une donnée simple ou une structure. La figure 4 montre la composition d'une liste chaînée.

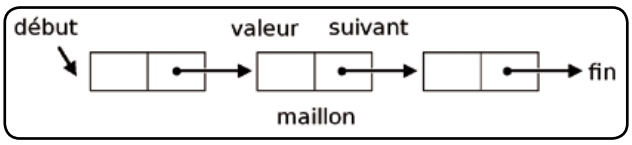


Fig. 4 : Structure d'une liste chaînée

Les opérations fondamentales sur une liste chaînée seront le test de vacuité, l'insertion, la suppression et l'accès à un élément en fonction de sa position.

3.1 Les opérations

3.1.1 Test de vacuité

Soit une liste **L** et une absence de valeur notée **nil**, pour savoir si une liste est vide un test suffit :

```
Fonction EstVide(L) -> Booléen
    Si L == nil alors
        Retourner Vrai
    Sinon
        Retourner Faux
```

Cette opération est évidemment en O(1).

3.1.2 Insertion

L'insertion est un peu plus délicate. Elle peut se produire en début de liste, en fin de liste ou à une position donnée.

3.1.2.1 Insertion en début de liste

L'insertion en début de liste est le cas le plus simple. Il faut créer un nouveau maillon contenant l'élément **e** à ajouter, indiquer que la liste commence par ce maillon et que le maillon suivant est l'ancien début de liste. Avec le schéma de la figure 5 ce sera sans doute plus clair...

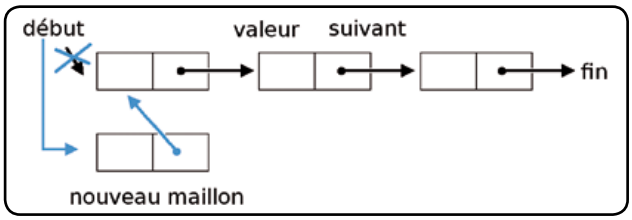


Fig. 5 : Insertion d'un élément en tête d'une liste chaînée

Opération	Coût en temps pour une pile	Coût en temps pour une file
Test de vacuité	O(1)	O(1)
Insertion	O(1)	O(1)
Récupération de la valeur suivante (pas de choix)	O(1)	O(1)

Supposons donc que l'on veuille insérer un élément **e** dans une liste **L** et que la structure d'un maillon comprend un champ **valeur** et un champ **suivant**.

```
m <- nouveau maillon
m.valeur <- e
m.suivant <- L
L.suivant <- m
```

On commence par lier le maillon à la liste avant de changer la tête de la liste de manière... à ne pas perdre la tête (de la liste). En effet, c'est l'erreur la plus courante : la perte de la tête de la liste implique la perte de toutes les informations qu'elle contient puisque nous n'avons plus de point d'entrée.

L'insertion en début de liste est bien entendu constante en temps (O(1)).

3.1.2.2 Insertion en fin de liste

Le mécanisme de l'insertion en fin de liste sera similaire à l'insertion en tête, à la différence près qu'il faudra parcourir l'intégralité de la liste avant de procéder à l'insertion. Notez également qu'il faudra utiliser une tête de lecture temporaire pour parcourir la liste et ne pas perdre la véritable tête de la liste (voir figure 6).

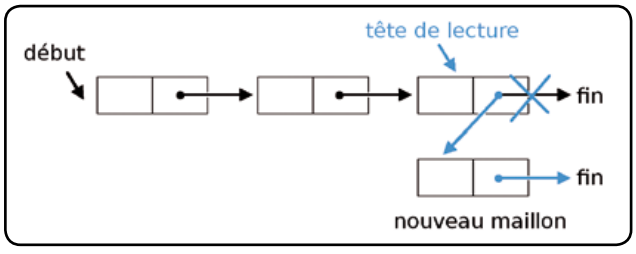


Fig. 6 : Insertion d'un élément en fin d'une liste chaînée

Nous prendrons ici les mêmes notations que précédemment et nous utiliserons en plus une tête de lecture **tete** :

```
m <- nouveau maillon
m.valeur <- e
m.suivant <- nil
Si non EstVide(L) Alors
    tete <- L
    Tant que tete.suivant <> nil Faire
        tete <- tete.suivant
    tete.suivant <- m
Sinon
    L <- m
```

Cette fois, il faut parcourir les **n** éléments que contient la liste avant de pouvoir réaliser l'insertion : la complexité est en O(n).

3.1.2.3 Insertion à une position donnée

Le dernier type d'insertion consiste à donner un indice ou une valeur après laquelle insérer un nouveau maillon. Nous considérerons ici que nous souhaitons effectuer l'insertion en position **i**, soit après les **i - 1** premiers maillons (voir figure 7).

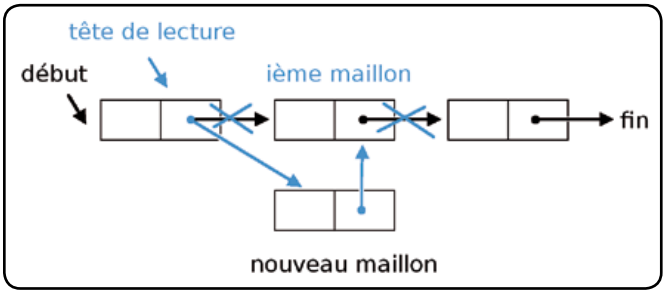


Fig. 7 : Insertion d'un élément dans une liste chaînée à une position donnée

Voici l'algorithme correspondant à l'insertion d'un élément **e** en position **i**. Nous utiliserons ici une variable **cpt** permettant de connaître l'indice de l'élément sur lequel nous nous trouvons.

```
m <- nouveau maillon
m.valeur <- e
tete <- L
cpt <- Ø
Si i == Ø Alors
    m.suivant <- tete
    tete.suivant <- m
Sinon Si non EstVide(L) Alors
    Tant que cpt <> i - 1 Faire
        tete <- tete.suivant
        cpt <- cpt + 1
        Si tete == nil Alors
            Erreur
    Si tete.suivant == nil Alors
        m.suivant <- nil
    Sinon
        m.suivant <- tete.suivant
    tete.suivant <- m
```

Là encore, dans le pire des cas, il faut parcourir **n** éléments et la complexité est donc en O(n).

3.1.3 Suppression

La suppression est relativement simple à comprendre puisque l'on court-circuite un maillon. Là encore, le maillon à supprimer peut être indiqué en fonction de sa valeur ou de son indice et nous utiliserons encore l'indice. La figure 8 illustre le court-circuit d'un maillon.

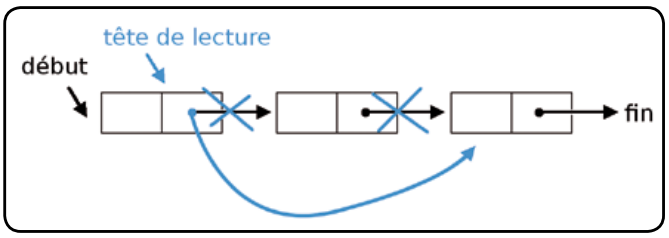


Fig. 8 : Suppression d'un élément dans une liste chaînée d'après une position donnée

Comme vous avez pu le constater sur la figure 8, nous aurons une fois de plus besoin de la tête de lecture de manière à ne pas perdre l'élément que nous supprimons et, suivant les langages, pouvoir l'éliminer de la mémoire. Nous cherchons ici à supprimer le **i** ème élément de la liste :

```
tete <- L
cpt <- Ø
Si EstVide(L) Alors
    Erreur
Sinon Si i == Ø Alors
    L <- L.suivant
    tete.suivant <- nil
    Libérer mémoire tete
Sinon
    Tant que cpt <> i - 1 Faire
        tete <- tete.suivant
        cpt <- cpt + 1
        Si tete.suivant == nil Alors
            Erreur
    poubelle <- tete.suivant
    tete.suivant <- poubelle.suivant
    poubelle.suivant <- nil
    Libérer mémoire poubelle
```

Classiquement, comme il faut parcourir les éléments de la liste, cet algorithme est en O(n).

3.1.4 Accès à un élément en fonction de sa position

Maintenant que nous avons vu l'insertion à une position donnée et la suppression, accéder à un élément en fonction d'un indice est une simplification des méthodes déjà employées, puisqu'il suffira de parcourir la liste à l'aide d'une tête de lecture (voir figure 9).

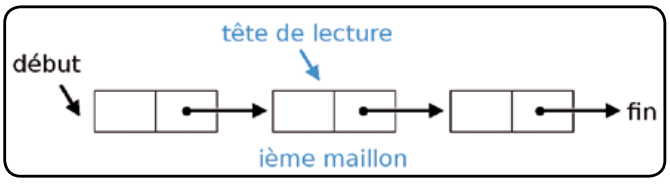


Fig. 9 : Accès à un élément dans une liste chaînée d'après une position donnée

```
tete <- L
cpt <- Ø
Si EstVide(L) Alors
    Erreur
Sinon
    Tant que cpt <> i Faire
        tete <- tete.suivant
        cpt <- cpt + 1
        Si tete == nil Alors
            Erreur
    Utiliser tete.valeur
```

Encore une fois, cette opération s'exécute en temps linéaire (O(n)).

3.2 Un exemple pratique de liste chaînée

Les listes chaînées se prêtent particulièrement bien à une modélisation objet : nous avons l'objet **Maillon** et l'objet **Liste**. Cette fois-ci, nous pourrons utiliser Python. La classe **Maillon** sera vraiment très simple puisqu'elle ne comportera qu'un constructeur dont la mission est de définir les deux attributs **valeur** et **suivant**. Cette classe se trouve dans un fichier **Maillon.py** :

```
01: class Maillon:
02:
03:     def __init__(self, elt):
04:         self.valeur = elt
05:         self.suivant = None
```

La classe **Liste** du fichier **Liste.py** contiendra toute l' « intelligence » de la liste :

```
01: from Maillon import Maillon
02:
03:
04: class Liste:
05:
06:     def __init__(self):
07:         self.debut = None
```

Le constructeur définit seulement un attribut qui représente le début de la liste. Pour savoir si la liste est vide, il suffit de regarder la valeur de cet attribut :

```
09:     def estVide(self):
10:         return self.debut is None
```

Les méthodes d'insertion suivent scrupuleusement les algorithmes énoncés précédemment :

```
12:     def insere_debut(self, elt):
13:         m = Maillon(elt)
14:         m.suivant = self.debut
15:         self.debut = m
16:
17:     def insere_fin(self, elt):
18:         m = Maillon(elt)
19:         if not self.estVide():
20:             tete = self.debut
21:             while tete.suivant is not None:
22:                 tete = tete.suivant
23:             tete.suivant = m
24:         else:
25:             self.debut = m
26:
27:     def insere_indice(self, elt, i):
28:         m = Maillon(elt)
29:         tete = self.debut
30:         cpt = 0
31:         if i == 0:
32:             m.suivant = tete
33:             self.debut = m
34:         elif not self.estVide():
35:             while cpt != i - 1:
36:                 tete = tete.suivant
37:                 cpt = cpt + 1
38:                 if tete is None:
39:                     print("ERREUR")
40:                     return False
41:             if tete.suivant is not None:
42:                 m.suivant = tete.suivant
43:             tete.suivant = m
```

La suppression est, elle aussi, conforme à l'algorithme précédent. Notez l'absence de libération de la mémoire : Python dispose d'un ramasse-miettes ou *garbage collector*, qui se chargera de récupérer l'espace mémoire non utilisé. C'est pour cela qu'il est très important de penser à supprimer tout lien entre le maillon à supprimer et la liste (voir notamment la ligne 63).

```
45:     def supprimer(self, i):
46:         tete = self.debut
47:         cpt = 0
48:         if self.estVide():
49:             print("ERREUR")
50:             return False
51:         elif i == 0:
52:             self.debut = self.debut.suivant
53:             tete.suivant = None
54:         else:
55:             while cpt != i - 1:
56:                 tete = tete.suivant
57:                 cpt = cpt + 1
58:                 if tete.suivant is None:
59:                     print("ERREUR")
60:                     return False
61:             poubelle = tete.suivant
62:             tete.suivant = poubelle.suivant
63:             poubelle.suivant = None
```

L'obtention d'une valeur de la liste située à un indice donné est en fait une simplification de la fonction de suppression :

```
65:     def valeur(self, i):
66:         tete = self.debut
67:         cpt = 0
68:         if self.estVide():
69:             print("ERREUR")
70:             return False
71:         else:
72:             while cpt != i:
73:                 tete = tete.suivant
74:                 cpt = cpt + 1
75:                 if tete is None:
76:                     print("ERREUR")
77:                     return False
78:             return tete.valeur
```

Petit bonus : une fonction d'affichage qui fonctionne sur le principe de la tête de lecture, que l'on déplace sur la liste :

```
80:     def affiche(self):
81:         tete = self.debut
82:         while tete is not None:
83:             print(tete.valeur)
84:             tete = tete.suivant
```

Il est enfin possible de tester toutes ces fonctions :

```
86: if __name__ == "__main__":
87:     L = Liste()
88:
89:     if L.estVide():
90:         print("La liste est vide!")
91:
92:     L.insere_indice(1, 0)
93:     L.insere_indice(2, 0)
94:     L.supprimer(1)
95:     L.affiche()
96:
97:     print()
98:     print(L.valeur(1))
```

3.3 Résumé

Voici un tableau résumant les coûts de chacune des opérations sur les listes :

Opération	Coût en temps
Test de vacuité	O(1)
Insertion en début	O(1)
Insertion en fin	O(n)
Insertion à un indice donné	O(n)
Suppression à un indice donné	O(n)
Accès à un élément d'un indice donné	O(n)

3.4 Les autres types de listes chaînées

Il existe d'autres types de listes chaînées dont je n'ai pas parlé ici :

- les listes circulaires (voir figure 10).

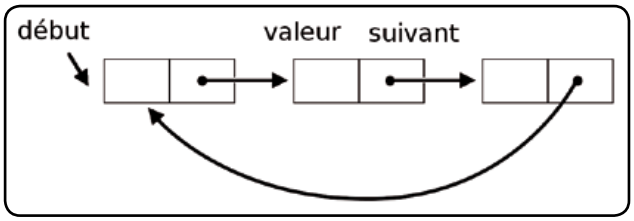


Fig. 10 : Liste chaînée circulaire

- les listes doublement chaînées (voir figure 11). Cette structure entraîne forcément une modification de la structure des maillons qui devront contenir un champ supplémentaire pointant vers le maillon précédent. Dans le cas de l'implémentation en Python, cela consisterait simplement en l'ajout d'un attribut :

```
01: class Maillon:
02:
03:     def __init__(self, elt):
04:         self.valeur = elt
05:         self.suivant = None
06:         self.precedent = None
```

Pour les différentes méthodes, il faudra ensuite penser à effectuer un chaînage correct non seulement avec les maillons suivants, mais également les précédents.

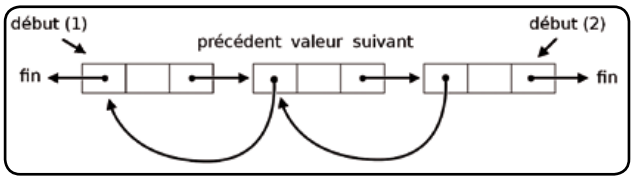


Fig. 11 : Liste doublement chaînée

On peut bien sûr imaginer une infinité d'autres structures basées sur des chaînages de maillons en fonction des cas à traiter, il ne s'agit jamais que de zones mémoires liées entre elles par des pointeurs.

4 Les tables de hachage

Une table de hachage (ou tableau associatif, ou encore dictionnaire suivant les langages) est une structure de données composée de couples clé/valeur où chaque clé fait référence au plus à une valeur. Il s'agit d'un tableau non ordonné où l'accès à un élément se fait grâce à la clé qui lui est associée et qui est transformée en indice grâce à une fonction de hachage comme le montre la figure 12.

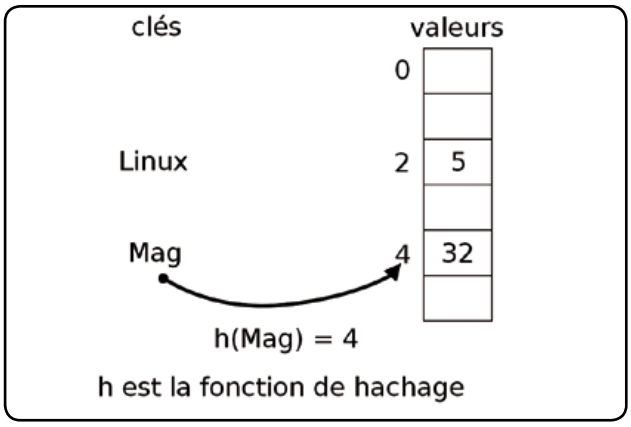


Fig. 12 : Fonctionnement d'une table de hachage

Le choix de la fonction de hachage est essentiel, car c'est elle qui permet de retrouver une valeur en fonction de sa clé et qu'il faut minimiser le plus possible les risques de collisions (deux clés distinctes pointent vers le même indice). En d'autres termes, si nous considérons que **h** est la fonction de hachage et que **(cle_1, valeur_1)** et **(cle_2, valeur_2)** sont deux couples de clé/valeur, une collision apparaît lorsque **h(cle_1) = h(cle_2)**. En général, il est malheureusement impossible d'éviter les collisions... Il faut donc être capable dans un premier temps de les minimiser puis, dans un second temps, de les traiter.

Le paradoxe de la date anniversaire

Ce paradoxe dit que si au moins 23 personnes sont présentes dans une pièce, il y a de très fortes chances pour que deux d'entre elles aient le même jour et le même mois d'anniversaire. Si nous rapportons cela à une fonction de hachage, si nous avons 23 clés et une table de 365 cases, la probabilité pour que deux clés ne soient pas associées à une même case est de ... 0,4927. Les collisions ne peuvent pas être évitées, voilà pourquoi il faut les gérer !

4.1 Résolution des collisions

4.1.1 Résolution par chaînage

La solution la plus simple consiste à stocker les clés et les valeurs et, en cas de collision, de créer une liste chaînée (voir figure 13). Si nous notons **T** le tableau contenant

les données, **h** la fonction de hachage, l'algorithme permettant de rechercher un élément associé à une clé **cle_1** est le suivant :

```
i <- h(cle_1)
Si T[i] == nil Alors
    Erreur
Sinon
    tete <- T[i]
    Tant que tete <> nil Faire
        Si tete.cle == cle_1 Alors
            Valeur trouvée !
            Sortir
        Sinon
            tete = tete.suivant
    Erreur
```

Normalement, la majorité des éléments devraient être accessibles en temps constant... Mais c'est le pire des cas qu'il faut considérer et, dans ce cas, tous les éléments du dictionnaire occupent la même position et nous avons donc une complexité en O(n) pour accéder à un élément.

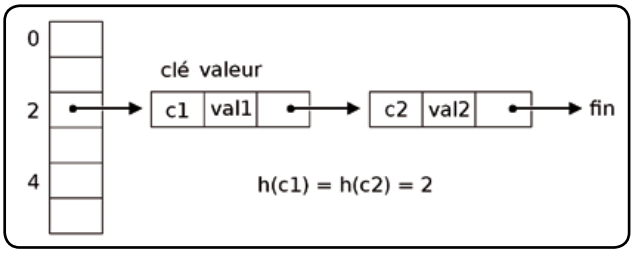


Fig. 13 : Résolution des collusions dans une table de hachage par chaînage

4.1.2 Résolution par adressage ouvert

Une autre solution permettant de résoudre les collisions est l'adressage ouvert : les informations associées à une même clé sont stockées de manière contiguë. L'idée est en fait d'avoir une fonction de hachage qui ne va non plus déterminer la position d'une valeur, mais plutôt d'une zone dans laquelle la référence doit pouvoir être trouvée comme le montre la figure 14. Cette fois, l'algorithme de recherche est le suivant :

```
i <- h(cle_1)
Tant que T[i] <> nil Alors
    Si T[i].cle == cle_1 Alors
            Valeur trouvée !
            Sortir
    Sinon
        i <- i + 1
Erreur
```

La recherche se fait encore une fois en temps linéaire (O(n)).

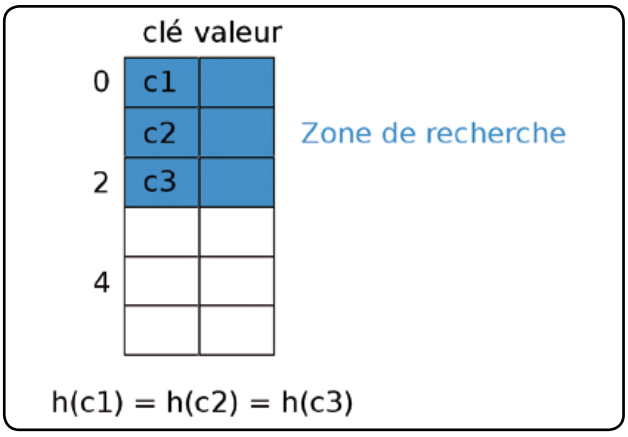


Fig. 14 : Résolution des collusions dans une table de hachage par adressage ouvert

4.2 La fonction de hachage

La fonction de hachage est au cœur du mécanisme des tables de hachage, elle est fondamentale et il faut donc la choisir avec rigueur. Sa mission est, en fonction d'une clé numérique ou alpha-numérique, de retourner un indice numérique avec le minimum de collisions possible. Voici les deux exemples les plus simples de fonctions de hachage s'appliquant à une résolution des collisions par chaînage.

4.2.1 Méthode par division

Soit **c** une clé et **m** la taille du tableau **T** permettant de stocker les données. **h** est une fonction de hachage telle que **0 <= h(c) < m** (de manière à ce que l'indice puisse être associé à **T**).

La méthode par division est particulièrement simple ; elle consiste à définir **h** telle que **h(c) = c mod m**.

Par exemple, pour une clé numérique valant 12 et un tableau de 5 éléments, **h(12) = 12 mod 5 = 2** : la valeur associée à la clé 12 sera stockée dans T[2].

4.2.2 Méthode par multiplication

La méthode par multiplication est plus difficile à appréhender, car il faut travailler avec des fractions plutôt qu'avec des entiers (mais nous ramenons bien sûr le résultat à une valeur entière). En posant **A** un nombre réel tel que **0 < A < 1**, la fonction de hachage est **h(c) = [m(Ac - [Ac])]** où **[ ]** désigne la partie entière... Le problème de ne pas pouvoir utiliser LaTeX pour écrire

les articles fait que certains symboles mathématiques ne sont pas accessibles, comme par exemple la notation anglo-saxonne de la partie entière où la fermeture des crochets n'est présente qu'en haut ou en bas et qui permet de préciser l'arrondi par défaut ou par excès (ici c'est un arrondi par défaut). Cette écriture peut être simplifiée en `h(c) = [ m x { Ac } ]` où `{ }` représente la partie fractionnaire.

Donald Knuth **[2]**, professeur émérite en informatique de l'université de Standford, pionnier de l'algorithmique, créateur de TeX et auteur de la série d'ouvrages « The Art of Computer Programming », recommande de prendre pour valeur de A ce qu'il appelle le ratio d'or : `A = (√5 - 1) / 2 ≈ 0.618`.

En reprenant l'exemple précédent, pour une clé numérique valant 12 et un tableau de 5 éléments, `h(12) = [ 5 x { 0.6 x 12 } ] = [ 5 x { 7.2 } ] = [ 5 x 0.2 ] = 1` : la valeur associée à la clé 12 sera stockée dans T[1].

Conclusion

Nous sommes revenus dans cet article sur des structures que l'on utilise tous les jours, mais dont on ignore parfois le fonctionnement lorsque l'on utilise des bibliothèques toutes prêtes. Si à un moment donné vous souhaitez optimiser votre code, il est essentiel de comprendre leur fonctionnement interne... Parfois, une structure non adaptée peut faire perdre beaucoup de temps de calcul...

Si cet article vous a ouvert l'appétit et que vous souhaitez obtenir plus d'informations sur le sujet, je vous recommande la lecture (même si elle est ardue) des ouvrages de Donald Knuth **[2][3]** dans lesquels vous trouverez de nombreuses explications. ■

Références

[1] Bachmann P., « Die analytische Zahlentheorie », Leipzig B. G. Teubner, 1894. (peut être consulté sur https://archive.org/details/dieanalytischeza00bachuoft)

[2] Knuth D., « The Art of Computer Programming - Volume 1 Fundamental algorithms », Addison Wesley, 1998.

[3] Knuth D., « The Art of Computer Programming - Volume 3 Sorting and searching », Addison Wesley, 1998.

DEMANDEZ LA LUNE À NGINX !

*par **Emile 'iMil' Heitor** [pour GCU-Squad! canal historique et la secte des serviettes de bain oranges]*

Nginx n'est plus un logiciel à la marge, avec près de 20% de parts de marché du top un million des sites les plus fréquentés au monde, il représente même un acteur sur lequel on peut (et doit !) compter.

La souplesse de nginx, sa configuration aux antipodes de l'ancestral `apache.conf`, mais aussi et surtout ses incroyables performances en font un des « chouchous » des acteurs du Web aujourd'hui.

Nginx dispose d'une palette de modules officiels des plus utiles, mais il est souvent intéressant de s'enquérir des contributions tierces, car certaines d'entre elles amènent parfois de véritables révolutions. Nous avions déjà parcouru dans ces colonnes les capacités du pare-feu applicatif NAXSI **[1]**, nous allons aujourd'hui plonger dans les fabuleuses possibilités amenées par le module `lua`.

1 Programmer le protocole

Imaginez une seconde : qui, dans la chaîne HTTP, est le mieux placé pour tout voir passer, interagir, modifier ? Le serveur lui-même, voire le proxy inverse **[2]**, rôle que joue avec brio nginx. Aussi, à l'aide de directives nginx et une API Lua **[3]** très accessible, on pourra, à la volée :

- Transformer les URI,
- Manipuler les en-têtes,
- Générer du contenu,
- Interagir avec d'autres composants,
- Intervenir dans le corps des objets transférés.

Le tout avec un véritable langage de programmation dont l'empreinte est minuscule et les performances « proches du C natif », en particulier grâce au compilateur LuaJIT **[4]**.

En effet, afin d'obtenir les meilleures performances, il est fortement recommandé par le site du projet d'utiliser LuaJIT, le compilateur *Just In Time*, plutôt que l'interpréteur Lua standard dès que cela est possible. Il est intéressant de noter que l'interpréteur de LuaJIT, à l'occasion de sa version 2.0, a été réécrit en **assembleur**. On est loin d'autres univers moins regardants...

2 Installation

Si vous êtes l'heureux possesseur d'un système d'exploitation utilisant `pkgsrc` **[5]** comme système de paquets, la tâche sera relativement aisée puisque votre serviteur a inclus le support LuaJIT dans les paquets `www/nginx` et `www/nginx-devel` en mars 2014. Aussi, il vous suffira d'ajouter au fichier `/etc/mk.conf` la directive de compilation suivante :

```
PKG_OPTIONS.nginx+= luajit
```

Ou encore d'installer le paquet binaire proposé par NetBSDfr **[6]**, qui l'inclura au moment où vous lirez ces lignes.

Utilisateurs de Debian GNU/Linux, si le paquet `nginx-extras` **[7]** inclut bien le support Lua, il ne s'agit malheureusement que de l'interpréteur standard et non de la version Jit. Cela permettra tout de même de s'essayer aux exemples que nous allons dérouler dans cet article, mais en production, cela vous priverait de performances optimales.

La compilation manuelle n'est pas beaucoup plus fastidieuse grâce au projet OpenResty **[8]** :

```
$ VERSION=1.5.8.1 # à remplacer par la version courante d'OpenResty
$ wget http://openresty.org/download/ngx_openresty-${VERSION}.tar.gz
$ tar zxvf ngx_openresty-${VERSION}.tar.gz
$ cd ngx_openresty-${VERSION}/
$ ./configure --with-luajit
$ make
$ make install
```

Le préfixe utilisé sera **/usr/local/openresty/nginx**, ce qui évitera de semer le chaos dans votre installation. À noter que les utilisateurs de Debian GNU/Linux et Ubuntu devront s'acquitter de l'ajout préalable des paquets suivants :

```
# apt-get install libreadline-dev libncurses5-dev libpcre3-dev
libssl-dev perl make
```

Et les utilisateurs de Fedora / Red Hat :

```
# yum install readline-devel pcre-devel openssl-devel
```

3 Bonjour, AK47

Nginx disposant désormais des super-pouvoirs que lui octroie LuaJIT, nous allons pouvoir exposer notre premier -minuscule- bout de code à l'aide du langage Lua. Nous allons pour cela déclarer une location nginx particulière, que nous agrémenterons au fil de nos expériences. Pour le moment, nous déclarons le type MIME **text/plain**. Voici l'extrait concerné d'un fichier **nginx.conf** :

```
location /luatest {
  default_type 'text/plain';

  content_by_lua 'ngx.say("Bonjour, Camarade.")';
}
```

Notre premier morceau de Lua / nginx. Nous utilisons ici la directive **content_by_lua**, qui exécutera le code placé entre simple ou double quotes. On pourrait également appeler un fichier contenant du code Lua grâce à la directive **content_by_lua_file** à qui on passerait en paramètre le chemin vers le script.

On utilise au sein de ce « programme » la fonction **say** de l'API nginx de Lua. Cette fonction affiche une chaîne de caractères et la suffixe d'un retour chariot (**\n**).

Nous avons bien évidemment accès, depuis un morceau de code Lua, aux différentes variables de nginx, par exemple :

```
content_by_lua 'ngx.print("Document root: [" .. ngx.var["document_root"] .. "]\\n")'
```

Ou encore :

```
content_by_lua 'ngx.print("Document root: [" .. ngx.var.document_root .. "]\\n")'
```

Résultat :

```
imil@tatooine:~$ curl -o- -s http://coruscant/luatest
Document root: [/home/imil/www]
```

Et puisque nous avons accès à toutes les variables internes de nginx, nous pouvons conditionner des comportements utiles à l'administrateur système en quête d'informations :

```
if tonumber(ngx.var.upstream_response_time) > 1 then
    ngx.log(ngx.WARN, "[LENT!] Reponse de l'upstream <" .. ngx.var.upstream_addr ..
">: " .. ngx.var.upstream_response_time);
end
```

À ce propos, en cas d'erreur de programmation, cette dernière se retrouvera naturellement dans l'**error.log** défini dans la configuration du serveur.

Bien entendu, on a également accès aux variables que sait manipuler le serveur :

```
content_by_lua 'ngx.print("glmf vaut: <" .. ngx.var["arg_glmf"] .. ">\\n")'
```

Résultat :

```
imil@tatooine:~$ curl -o- -s http://coruscant/luatest/?glmf=OVER9000
glmf vaut: <OVER9000>
```

Rentrons un peu plus dans les arcanes du protocole ; parmi les informations dont dispose le serveur web, on retrouve la requête émanant du client, que l'on peut afficher sans plus de soin :

```
content_by_lua 'ngx.print(ngx.req.raw_header())'
```

Résultat :

```
imil@tatooine:~$ curl -o- -s http://coruscant/luatest
GET /luatest HTTP/1.1
User-Agent: curl/7.26.0
Host: coruscant
Accept: */*
```

Ou encore récupérer dans un tableau associatif bien plus aisé à manipuler :

```
content_by_lua '
  local head = ngx.req.get_headers()
  for k, v in pairs(head) do
      ngx.say("header disponible: " .. k)
  end
  ngx.say("Host: ", head["Host"])
';
```

Résultat :

```
imil@tatooine:~$ curl -o- -s http://coruscant/luatest
header disponible: user-agent
header disponible: host
header disponible: accept
Host: coruscant
```

Avant de servir la ressource demandée, l'API lua-nginx permet également de s'interposer dans l'échange :

```
local res = ngx.location.capture("/foo")
if res.status ~= 200 then
  ngx.print("GOULAG! (reponse: " .. res.status .. ")\\n")
else
  ngx.print(res.body)
end
```

Résultat :

```
imil@tatooine:~$ curl -o- -s http://coruscant/luatest
GOULAG! (reponse: 404)
```

Quelques explications s'imposent ici. L'appel à **ngx.location.capture**, qui est non-bloquant, comme nginx lui-même, effectue une requête **interne** vers la location **foo** ; comprendre qu'il n'y a aucune transaction TCP, HTTP ou encore d'IPC. Cela permet une efficience optimale, puisque s'articulant sur les mécanismes internes à nginx. **ngx.location.capture** retourne dans la variable locale **res** le tableau suivant :

- **res.status** contient le code de retour de la réponse HTTP ;
- **res.header** est un tableau contenant l'ensemble des en-têtes de réponse ;
- **res.body** contient le corps du message, par exemple le HTML ;
- **res.truncated** est un drapeau booléen permettant de savoir si le contenu de **res.body** est tronqué.

Et puisque l'on dispose du contenu retourné dans **res.body**, on peut a priori le transformer comme bon nous semble :

```
location /luatest {
  default_type 'text/plain';
  content_by_lua '
      local res = ngx.location.capture("/")
      if res.status ~= 200 then
          ngx.say("GOULAG!! (reponse: " .. res.status .. ")")
      else
          local c, n, err = ngx.re.sub(res.body, "[uU][pP]", "down")
          if c then
              ngx.print(c)
          else
              ngx.log(ngx.ERR, "erreur: ", err)
              return
          end
      end
  ';
}
```

Oui, pour ne rien gâcher, l'API lua-nginx supporte les expressions régulières.

Avec cette portion de code, lorsqu'on interroge la racine de **coruscant**, on obtient :

```
imil@tatooine:~$ curl -o- -s http://coruscant/
up.
```

Et lorsqu'on interroge la location **/luatest** :

```
imil@tatooine:~$ curl -o- -s http://coruscant/luatest
down.
```

Effrayant. À utiliser avec discernement, évidemment, mais les possibilités offertes par les capacités de *matching* des fonctions **ngx.re.*** ouvrent la porte à de multiples possibilités de contrôle de validité ou d'interaction avec des modules Lua tiers, tels que **resty.redis** **[9]** ou **resty.memcached** **[10]**.

4 In mother Russia, Lua controls you

Nous nous sommes jetés tête la première sur la directive **content_by_lua**, mais ce n'est pas, loin s'en faut, la seule section manipulable en Lua. Voyons ensemble les possibilités offertes par quelques-unes d'entre elles.

- **set_by_lua** nous permettra de réaliser des opérations complexes en Lua, afin de retourner une valeur dans une variable à la disposition de nginx. Ceux d'entre vous qui ont buté sur les limitations de la manipulation

des variables au sein du serveur HTTP verront ici une façon simple d'additionner, soustraire, concaténer ou encore importer des valeurs dépendant d'une infinité de paramètres.

L'utilisation est similaire à **content_by_lua** si ce n'est la présence d'une variable de retour.

```
set $tagada '';

set_by_lua $ret '
  ngx.var.tagada = "tsointsoin"

  return "et hop"
';
```

Ici, la variable nginx **$tagada** vaut « tsointsoin » et **$ret** « et hop ».

- **rewrite_by_lua** agit dans la phase de réécriture ; on pourra par exemple utiliser cette fonctionnalité pour rediriger immédiatement vers une location interne à l'aide de la fonction **ngx.exec** :

```
rewrite_by_lua '
  local m, err = ngx.re.match(ngx.var.remote_addr, "192\.168\.1\.")
  if m then
      ngx.exec("/")
  end
';
```

- **access_by_lua** donnera une immense souplesse dans la gestion des autorisations d'accès à une ressource. Voyons un exemple basique d'authentification dépendant d'une ressource tierce :

```
access_by_lua '
  local res = ngx.location.capture("/auth")

  if res.status == ngx.HTTP_OK then
      return -- tout s'est bien déroulé, on poursuit
  end

  if res.status == ngx.HTTP_FORBIDDEN then
      ngx.exit(res.status) -- le backend a renvoyé un code 403,
on le relaye et on sort
  end
';
```

- Terminons cette liste d'exemples par la directive **header_filter_by_lua** grâce à laquelle on pourra réécrire les contenus des en-têtes de façon triviale :

```
ngx.header["X-Powered-By"] = "nginx baby!"
```

On imagine facilement ici la surcharge d'en-têtes malencontreusement laissée par un administrateur peu soucieux de la sécurité de ses serveurs.

5 Bon bah j'deviendrais bien maître du monde moi ce soir

Ce n'est là qu'une poignée des quarante (à ce jour) directives à votre disposition pour ajuster à la volée le comportement souhaité entre l'utilisateur et la ressource visitée. À l'aide des nombreux modules tiers disponibles, d'autres horizons ingénieux s'offrent à vous ; on remarquera par exemple le site http://devblog.mixlr.com/2012/09/01/nginx-lua/ qui donne l'exemple d'une directive **access_by_lua** qui fait communiquer nginx avec Redis **[11]** pour lire rapidement si un visiteur fait partie des adresses IP bannies sur le serveur.

Du fait de la simplicité du langage Lua et des innombrables possibilités du combo avec le serveur nginx, gageons que les contributions et nouveautés vont pulluler d'ici peu. ■

Références

[1] Site du pare-feu NAXSI : https://github.com/nbs-system/naxsi

[2] Définition d'un proxy inverse : http://fr.wikipedia.org/wiki/Proxy_inverse

[3] Site officiel de Lua : http://www.lua.org/

[4] Comparaison des performances de Lua et LuaJIT sur différentes architectures : http://luajit.org/performance.html

[5] Site officiel du projet pkgsrc (portable package build system) : http://pkgsrc.org/

[6] Paquets NetBSDfr : http://packages.netbsdfr.org/

[7] Paquet nginx-extra GNU/Debian : https://packages.debian.org/fr/wheezy/nginx-extras

[8] Site officiel du projet OpenResty : http://openresty.org/

[9] Module Lua Redis : https://github.com/agentzh/lua-resty-redis

[10] Module Lua Memcached : https://github.com/agentzh/lua-resty-memcached

[11] Site officiel de Redis : http://redis.io/

JOUONS UN PEU (PLUS) AVEC NGINX

*par **Emile 'iMil' Heitor** [pour GCU-Squad! canal historique et la secte des serviettes de bain oranges]*

Il existe pour le serveur web / proxy inverse nginx une foule de modules tierce partie, à ajouter au besoin, pas encore intégrés upstream mais pourtant de très bonne facture. Parmi les auteurs de ces modules, il en est un qui sort du lot par la quantité mais surtout la qualité de ses contributions, il s'agit de Yichun Zhang, plus connu sous son pseudo : agentzh.

Ce qui donne aux contributions de ce développeur une saveur particulière, c'est que ce dernier n'est autre qu'un employé de la société CloudFlare, qui utilise massivement le serveur nginx. Au vu du trafic considérable que cette société spécialisée dans le CDN **[1]** réalise, on peut présumer d'une certaine qualité et stabilité dans le code produit par ce furieux développeur.

Outre la beauté de ses réalisations, toutes disponibles sur son compte GitHub **[2]**, le lecteur programmeur pourra lire avec une certaine admiration le code de ses modules, non seulement produit avec le plus grand soin, mais également remarquablement documenté et truffé d'exemples. On est loin des Lennarteries. Le lecteur assidu n'en est pas à sa première rencontre avec agentzh, puisque c'est ce dernier qui a produit le fabuleux nginx-lua dont nous vantions les mérites dans l'article précédent.

Je vous propose de parcourir une sélection de quelques modules sortis des doigts magiques d'agentzh et d'y appliquer quelques exemples pratiques d'utilisation qui remplaceront avantageusement d'inutiles cascades phpesques.

Les modules que nous utiliserons dans la suite de cet article font tous partie de la suite OpenResty **[3]**, évidemment maintenue par... agentzh. OpenResty n'est pas un fork de nginx, il s'agit simplement d'une méthode de distribution d'un paquet nginx sous forme de source contenant toutes les extensions tierce partie jugées intéressantes pour transformer le serveur web en serveur d'applications.

Les heureux utilisateurs de **pkgsrc** **[4]** pourront utiliser la variable **PKG_OPTIONS.nginx** dans le fichier **/etc/mk.conf** afin d'y ajouter les modules souhaités, ces derniers ayant été inclus en mars 2014 par votre serviteur.

1 ngx_echo

Le premier module que nous allons manipuler est le très pratique et versatile **ngx_echo** **[5]**. Comme on peut l'imaginer à son nom, la première utilité du module **echo** est de renvoyer du texte, à la façon de la commande UNIX **echo(1)** :

```
$ cat nginx-test.conf
location /modtest {
    echo "foo.";
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
foo.
```

Tout comme la commande **echo(1)**, on peut ajouter le paramètre **-n** afin d'éviter un retour chariot au prochain affichage :

```
$ cat nginx-test.conf
location /modtest {
    set $a "foo... ";
    set $b "bar";

    echo -n $a;
    echo "$b from $remote_addr";
}
```

Car oui, bien évidemment, la commande **echo** permet d'afficher toutes les variables disponibles dans nginx :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
foo... bar from 192.168.1.1
```

Au-delà de ses simples capacités d'affichage, le module **echo** dispose d'un arsenal de fonctions aux applications multiples. Découvrons par exemple le fonctionnement de **echo_before_body** et **echo_after_body** :

```
location /modtest {
    echo_before_body "je passe avant le corps";
    echo_after_body "et moi je passe après";

    proxy_pass http://localhost/lecorps;
}

location /lecorps {
    echo "je suis le corps.";
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
je passe avant le corps
je suis le corps.
et moi je passe après
```

Le module **echo** ne propose pas seulement des fonctions, il met également en place des variables accessibles par nginx :

```
location /modtest {
    echo_reset_timer;
    echo $echo_client_request_headers;
    echo "Temps passé: $echo_timer_elapsed";
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
GET /modtest HTTP/1.1
User-Agent: curl/7.26.0
Host: coruscant
Accept: */*

Temps passé: 0.000
```

Ici, on a utilisé :

- **echo_reset_timer** qui met à **0** un compteur de temps ;
- la variable **$echo_client_request_headers** qui contient les en-têtes émis par le client ;
- la variable **$echo_timer_elapsed** dans laquelle on retrouve le temps écoulé depuis le dernier **echo_reset_timer**.

Encore plus fort, le module **ngx_echo** permet de réaliser des boucles basiques et même du traitement simplifié de chaînes !

```
location /modtest {
    echo_foreach_split '\n' $echo_client_request_headers;
            echo "En-tete: $echo_it";
    echo_end;
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
En-tete: GET /modtest HTTP/1.1
En-tete: User-Agent: curl/7.26.0
En-tete: Host: coruscant
En-tete: Accept: */*
En-tete:
```

ngx_echo dispose également d'une fonction **echo_exec** qui, au lieu de réaliser une redirection classique à travers le protocole HTTP, utilise les capacités internes du serveur nginx, gagnant ainsi quelques précieuses microsecondes :

```
location /modtest {
    echo_exec /ailleurs page=casimir;
}

location /ailleurs {
    echo "La vérité est ici: $arg_page";
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
La vérité est ici: casimir
```

On remarque qu'il est possible de passer des paramètres à **echo_exec** qu'on récupère de façon tout à fait classique dans nginx.

Un dernier exemple de fonctions utiles fournies par le module **ngx_echo** :

```
location /modtest {
    echo_sleep 2;
    echo_duplicate 100 ".";
    echo "\nFausse erreur !";
    echo_status 502;
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
..........
Fausse erreur...
imil@tatooine:~$ curl -I -s http://coruscant/modtest
HTTP/1.1 502 Bad Gateway
Server: nginx/1.5.12
Date: Sun, 23 Mar 2014 10:33:36 GMT
Content-Type: application/octet-stream
Connection: keep-alive
```

Ici, on a fait croire au client qu'il a fait réaliser au serveur un traitement lourd qui a échoué :

- On attend deux secondes grâce à la fonction **echo_sleep** ;
- On répète 100 fois une chaîne de caractères via **echo_duplicate** ;
- On renvoie un message quelconque ;
- On renvoie un code de retour **HTTP 502**.

Je vous invite à visiter la page du module **[5]**, afin de découvrir les multiples autres capacités de **ngx_echo** qui mériterait un article à lui tout seul.

2 set-misc

Cette petite boîte à outils étend les possibilités d'affectation de variables. En outre, **set-misc** ajoute des capacités d'échappement, « déséchappement », encodage et décodage hexadécimal, MD5, SHA1, base32, Base64, ainsi que bien d'autres facilités.

Voyons par exemple comment échapper une chaîne de caractères passée en paramètre **get** :

```
location /modtest {
    set_escape_uri $requete $arg_q;

    echo $requete;
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest?q="un paramètre non échappé"
un%20param%e8tre%20non%20%e9chapp%e9
```

La fonction **set_if_empty** fournit un moyen rapide de s'assurer qu'une variable est non vide :

```
location /modtest {
    set $mavar $arg_meilleur;

    set_if_empty $mavar "GLMF !";
    echo $mavar;
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest?meilleur=Bisounours
Bisounours
imil@tatooine:~$ curl -s http://coruscant/modtest
GLMF !
```

set_quote_sql_str, **set_quote_pgsql_str**, **set_quote_json_str** s'assurent respectivement qu'une chaîne de caractères est mise entre quotes selon les règles de MySQL, PostgreSQL et JSON.

La série de fonctions **set_encode_*** peut se révéler très utile pour encoder ou décoder simplement des valeurs, par exemple :

```
location /modtest {
    set_encode_base64 $encode $arg_id;

    echo $encode;
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest?id=login
bG9naW4=
```

On peut ainsi encoder et décoder des valeurs hexadécimales, base32 ou base64, ou de la même façon, générer des signatures SHA1 ou MD5 :

```
location /modtest {
    set_sha1 $sig $arg_id;

    echo $sig;
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest?id=login
2736fab291f04e69b62d490c3c09361f5b82461a
```

Quelques autres fonctions sont destinées à la génération aléatoire, ou pseudo-aléatoire de chaînes de caractères. Par exemple, pour produire une suite de 64 caractères aléatoires, on utilise la fonction **set_secure_random_alphanum** comme ceci :

```
location /modtest {
    set_secure_random_alphanum $rnd 64;

    echo $rnd;
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
ojrxwQ1hdH9yJNElUnimCBDOgDvKCKPQyMZff4T2k6MVolG8UuEBmuNzpdG3ZZLc
```

On remarquera également dans la liste des possibilités de **set-misc** des fonctions comme **set_formatted_gmt_time** et **set_formatted_local_time**, qui renseigneront une variable avec une chaîne de caractères représentant une date suivant le format standard de la fonction C **strftime**.

3 array-var

Le module **array-var** ajoute à nginx la capacité de traiter des tableaux de variables. Les fonctions sont au nombre de quatre :

- **array_split** découpe une variable en fonction d'un séparateur et place le résultat dans une variable représentant un tableau ;
- **array_join** réalise exactement l'inverse, on construira une chaîne de caractères constituée de plusieurs autres qu'on rejoindra avec un caractère ;
- **array_map** va décorer les résultats présents dans le tableau d'une chaîne de notre choix ;
- **array_map_op** quant à lui, décorera le tableau à l'aide de fonctions tierces.

Voyons tout cela dans un exemple :

```
location /modtest {
    array_split ',' $arg_vals to=$vals;
    array_map_op set_sha1 $vals;
    array_map "<$array_it>" $vals;
    array_join '\n' $vals;

    echo $vals;
}
```

Résultat :

```
imil@tatooine:~$ curl -s http://coruscant/
modtest?vals=GLMF,ca,roulaize
<3ca77e0bcb9bc9346058204c7528e04f4e57427f>
<1c42c72cf95aa1b76609b585b34baf6b501d713e>
<ee6efb18d32b7e610ae0d4243566a6b88c6229c7>
```

Ici, on découpe les valeurs passées dans la variable get **vals** suivant le séparateur **,** puis on applique la fonction **set_sha1** du module **set-misc** vu plus haut ; on décore la chaîne avec les caractères **<>**, on rejoint les éléments du tableau avec un retour chariot et on affiche le tout.

4 headers-more

Le module **headers-more** est une amélioration notable du module standard **headers** fourni avec nginx. **headers-more** permet d'ajouter, effacer ou placer un en-tête, il bénéficie également des capacités suivantes :

- Mise à zéro des en-têtes *builtin* tels que **Content-Type**, **Content-Length** ou **Server** ;
- Spécification d'un code HTTP grâce à une option **-s** ;
- Modification du type de contenu à l'aide de l'option **-t**.

Fonctionnalité non négligeable, **headers-more** a aussi la possibilité de modifier les en-têtes entrants, la preuve en images en commençant sur le serveur :

```
location /modtest {
    more_set_input_headers 'User-Agent: Kurt/10.8' 'X-Fake-Me: blebleble';

    proxy_pass http://exar/;
}
```

Un client :

```
imil@tatooine:~$ curl -s http://coruscant/modtest
```

Capture du trafic vers le serveur web cible du **proxy_pass** :

```
Host: exar
Connection: close
User-Agent: Kurt/10.8
Accept: */*
X-Fake-Me: blebleble
```

Dans l'autre sens, voici les en-têtes renvoyés par défaut par une location ne renvoyant qu'une chaîne de caractères à l'aide de la fonction **echo** :

```
imil@tatooine:~$ curl -I -s http://coruscant/modtest
HTTP/1.1 200 OK
Server: nginx/1.5.12
Date: Sun, 23 Mar 2014 13:49:03 GMT
Content-Type: application/octet-stream
Connection: keep-alive
```

Modifions les en-têtes à l'aide de **more-headers** :

```
location /modtest {
    more_set_headers 'Content-Type: text/plain' 'X-Powered-By: Goldorak';
    echo "foo";
}
```

Et de constater :

```
imil@tatooine:~$ curl -I -s http://coruscant/modtest
HTTP/1.1 200 OK
Server: nginx/1.5.12
Date: Sun, 23 Mar 2014 14:06:55 GMT
Content-Type: text/plain
Connection: keep-alive
X-Powered-By: Goldorak
```

À l'aide de la fonction **more_clear_headers**, on pourra également faire disparaître certains en-têtes :

```
more_clear_headers 'Server';
```

Résultat :

```
imil@tatooine:~$ curl -I -s http://coruscant/modtest
HTTP/1.1 200 OK
Date: Sun, 23 Mar 2014 14:09:40 GMT
Content-Type: text/plain
Connection: keep-alive
X-Powered-By: Goldorak
```

Nous n'avons plus d'en-tête **Server**.

5 encrypted-session

Le module **encrypted-session** fournit des capacités de chiffrement / déchiffrement pour les variables nginx. Son utilisation est classique :

- On définit une clé de chiffrement,
- On définit éventuellement un vecteur d'initialisation (*IV*) ; par défaut il vaut **deadbeefdeadbeef**,
- On définit éventuellement un temps d'expiration de la session ; par défaut elle vaut un jour,
- On chiffre la donnée.

En clair (oh-oh), cela donne :

```
location /modtest {
    encrypted_session_key "laclesuperbalaisepourglmfblableh";

    set_secure_random_lcalpha $rnd 32;
    set_encrypt_session $session $rnd;
    set_encode_base32 $session;
    add_header Set-Cookie 'token=$session';

    echo "foo";
}
```

On note dans cet exemple l'utilisation de la fonction **set_secure_random_lcalpha** issue du module **set-misc**

et permettant de générer une suite aléatoire de 32 caractères alphabétiques, ainsi que la fonction **set_encode_base32** qui créera une chaîne au format base32 avec la valeur de la variable chiffrée **$session**. Enfin, on ajoute un en-tête **Set-Cookie** contenant un jeton de valeur **$session**. Le résultat est le suivant :

```
HTTP/1.1 200 OK
Server: nginx/1.5.12
Date: Sun, 23 Mar 2014 14:42:44 GMT
Content-Type: application/octet-stream
Connection: keep-alive
Set-Cookie: token=bf99rjkobnho3o943eel1ram6hfj5edt8736q3d5ak1aub4l
b6tr1cka4sieqmi7v7cknuak7gqk6tc2khdlp8fekr8vhb0fia8apj8=
```

Le fonctionnement du déchiffrement est exactement inverse ; dans le cas présent, on convertirait la chaîne initialement en base32 à l'aide de **set_decode_base32** et on récupérerait la valeur déchiffrée grâce à la commande **set_decrypt_session**.

6 form-input

Nous achevons cette mise en bouche avec le module **form-input**. Ce dernier lit les requêtes **post** et **put** afin de transformer les arguments passés dans le corps du message en variables nginx. On peut aisément imaginer utiliser ce type de fonctionnalité pour filtrer des tentatives d'intrusion.

Exemple d'utilisation :

```
location /modtest {
    set_form_input $val;
    set_if_empty $val "rien !";
    echo "Input: $val";
}
```

On teste :

```
imil@tatooine:~$ curl --data "val=plop" -s http://coruscant/
modtest
Input: plop
imil@tatooine:~$ curl -s http://coruscant/modtest
Input: rien !
```

À noter qu'on peut également utiliser la syntaxe suivante :

```
set_form_input $variable val;
```

Dans celle-ci, on déclare que le champ concerné dans l'*input form* est **val** et la variable servant à stocker la valeur du champ est **$variable**.

Le module met à notre disposition une seconde fonction, **set_form_input_multi**, qui enregistrera dans un tableau toutes les valeurs d'un champ, par exemple :

```
location /modtest {
    set_form_input_multi $val;
    array_join ' ' $val;
    echo "Input: $val";
}
```

Résultat :

```
imil@tatooine:~$ curl --data "val=plop&val=bar&val=foo" -s http://coruscant/modtest
Input: plop bar foo
```

On reconnaît la fonction **array_join** présentée plus haut dans la section **array-var**.

7 Plus de musique, moins de bruit

On le comprend à la lecture des possibilités offertes à nginx par ces multiples extensions, on atteint un degré de contrôle a priori sans précédent au sein d'un serveur web et, de surcroît, d'un proxy inverse. En interagissant de la sorte, et aussi facilement, avec l'intégralité des composants du protocole HTTP, on s'imagine probablement factoriser un grand nombre de scripts et applicatifs précédemment écrits pour contrôler l'interaction client / serveur.

Le travail formidable d'agentzh ne constitue cependant qu'une partie des innombrables ajouts **[6]** disponibles pour le serveur nginx, et de véritables perles peuplent la page du site du serveur dédiée aux modules tierce partie. ■

Références

[1] Définition de Content Delivery Network : http://fr.wikipedia.org/wiki/Content_delivery_network

[2] Compte GitHub de Yichun Zhang : https://github.com/agentzh/

[3] Site officiel de OpenResty : http://openresty.org/

[4] Site officiel du projet pkgsrc (portable package build system) : http://pkgsrc.org/

[5] Page de HttpEchoModule : http://wiki.nginx.org/HttpEchoModule

[6] Liste de modules non officiellement supportés par nginx : http://wiki.nginx.org/3rdPartyModules

ADMINISTRER SA MESSAGERIE AVEC MODOBOA

par **Antoine Nguyen**

Mon précédent article vous a permis de préparer un serveur de messagerie muni de Modoboa. La dernière version sortie en février améliore grandement ses possibilités d'administration, vous allez les découvrir en détails dans cet article.

En suivant pas à pas le didacticiel présenté dans mon précédent article, vous avez installé et configuré un serveur de messagerie. Bien qu'étant fonctionnel, vous ne pouvez pas encore l'exploiter faute d'outil adapté (à moins d'être un administrateur chevronné et familier des systèmes mail).

Vous allez maintenant découvrir en détails les fonctionnalités d'administration offertes par Modoboa et qui vont vous permettre de maîtriser votre messagerie. Vous découvrirez aussi que ce logiciel s'adapte à différents contextes d'utilisation : serveur principal ou relais de filtrage, il répond à tous vos besoins ! (Qui a dit que j'exagérais ?!)

Note

Certains exemples fournis dans cet article font référence à un domaine fictif `domain.tld` qu'il faudra remplacer par le vôtre.

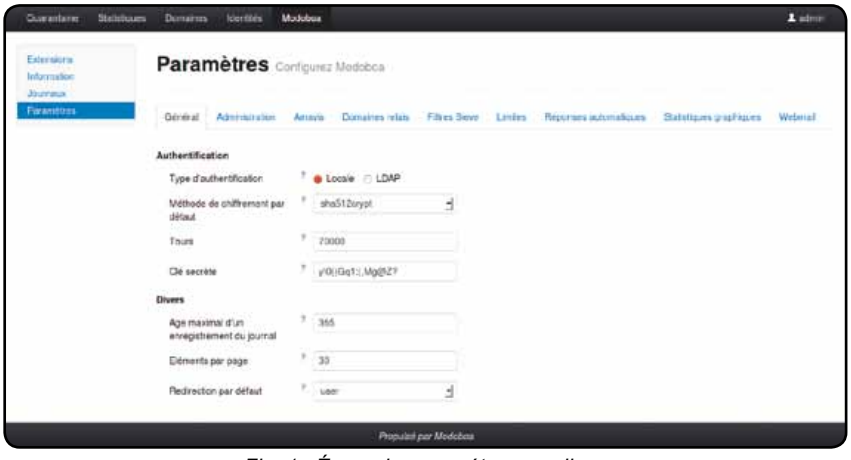


Fig. 1 : Écran de paramétrage en ligne

1 Configuration et utilisation de Modoboa

1.1 Console d'administration

L'interface d'administration est l'outil principal fourni par Modoboa. Ce dernier a d'ailleurs été développé à l'origine pour des besoins d'administration.

1.1.1 Paramétrage du système

La grande majorité des fonctionnalités de Modoboa sont paramétrables en ligne. Cliquez sur le menu ***Modoboa*** situé dans la barre noire en haut de page pour y accéder (Fig. 1).

Outre les paramètres généraux, chaque extension a la possibilité de définir les siens. Les onglets que vous apercevez dans la capture d'écran regroupent les paramètres par application (ou extension).

1.1.2 Analyse et surveillance

Modoboa dispose de quelques outils permettant de faciliter son exploitation. Tout d'abord, quelques statistiques concernant le trafic mail sont fournies nativement. Des courbes basées sur RRDtool permettent de visualiser :

- le trafic normal : messages envoyés et reçus par minute ;
- la taille en octets par minute du trafic normal ;
- le mauvais trafic : messages rejetés et renvoyés par minute.

Les périodes de consolidation par défaut sont : jour, semaine, mois, année et peuvent être sélectionnées depuis l'interface. « Consolider » signifie que les données stockées par RRDtool seront agrégées au fil du temps, afin de limiter la taille occupée sur le disque.

Note

N'oubliez pas de configurer correctement les différents chemins d'accès nécessaires à la génération de ces graphiques via l'écran de paramétrage, onglet ***Statistiques graphiques***. Indiquez `/srv/www/modoboa_instance/modoboa_instance/media/stats` comme répertoire de stockage des fichiers PNG.

De plus, un journal des événements garde une trace des actions importantes effectuées depuis Modoboa par les utilisateurs, en leur associant un niveau de criticité :

- Connexion/déconnexion des utilisateurs (info) ;
- Ajout d'objet (info) ;
- Modification d'objet (avertissement) ;
- Suppression d'objet (critique).

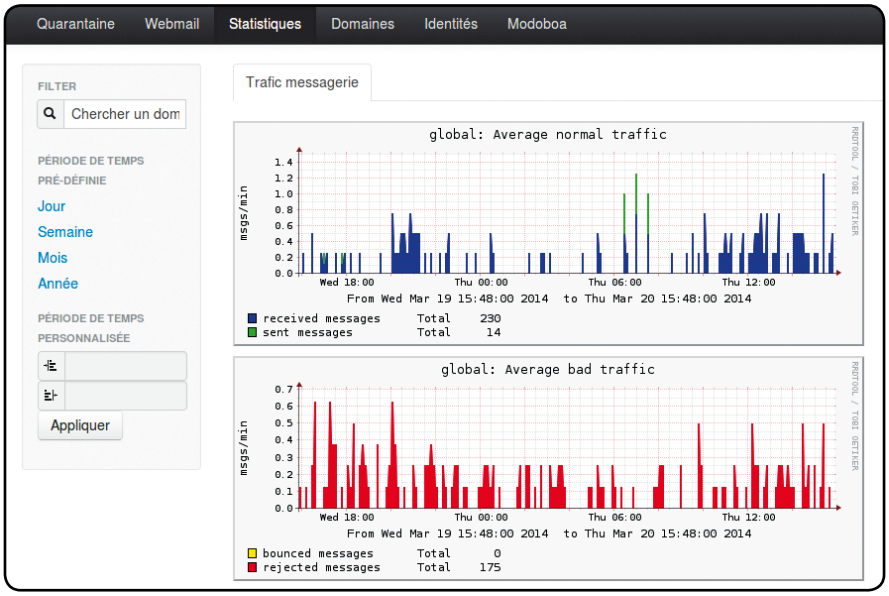


Fig. 2 : Graphiques RRDtool

Le temps de conservation des entrées de l'historique est modifiable depuis l'écran de paramétrage, dans l'onglet ***Administration*** (Fig. 3).

1.1.3 Peuplement

Pour la gestion quotidienne, les opérations d'ajout, modification et suppression pour tous les objets sont disponibles depuis l'interface. La page ***Domaines*** regroupe la gestion :

- des domaines et alias de domaines ;
- des domaines « relais » et alias de domaines « relais ».

Une des nouveautés de la version 1.1.0 est la définition depuis la page ***Domaines*** des domaines « relais » (ou de type « relais »). Il s'agit en fait de domaines dont la destination finale n'est pas le MTA géré par Modoboa. Celui-ci agit alors comme une passerelle.

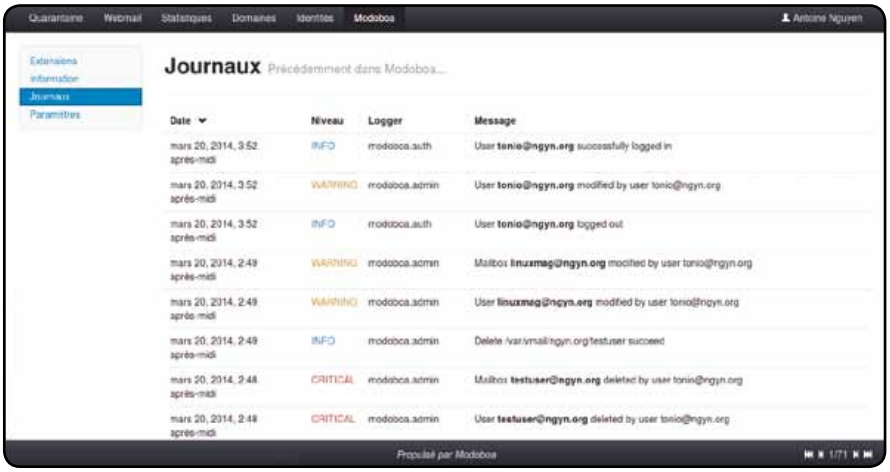


Fig. 3 : Journal des événements

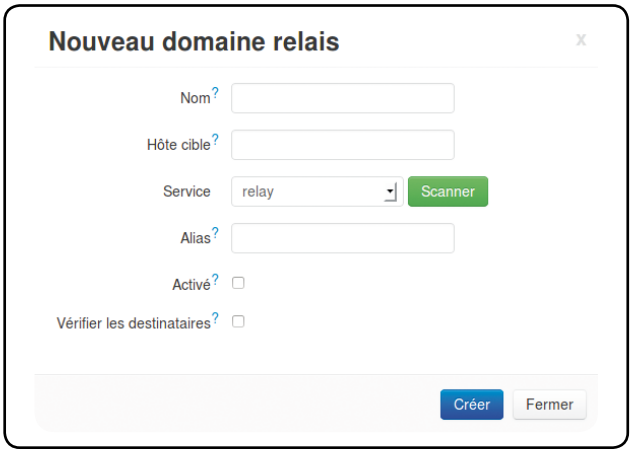


Fig. 4 : Création d'un domaine relais

La page ***Identités*** regroupe quant à elle la gestion :

- des comptes utilisateurs ;
- des alias, transferts et listes de distribution.

De plus, une fonctionnalité d'import massif est disponible depuis les deux pages (menu de gauche).

Fig. 5 : Menus d'import et d'export

À partir d'un fichier au format CSV, il est possible de créer plusieurs objets en une seule opération. Le format attendu est décrit directement dans les formulaires d'import.

Fig. 6 : Formulaire d'import d'identités

Note

L'import est un tantinet sensible, veillez à déclarer vos objets dans le bon ordre (compte, puis alias ou domaine, puis alias de domaine).

À l'inverse, vous pouvez exporter tout ou partie de vos objets vers un fichier CSV respectant le même format. Il devient alors possible de sauvegarder/restaurer votre installation (de manière basique et très manuelle, je vous l'accorde).

Note

L'export s'adapte à la sélection en cours. Si vous avez filtré le *listing* (pour rechercher un domaine en particulier par exemple), l'export interprétera ce filtre et le fichier résultant ne contiendra que les objets concernés.

1.1.4 Délégation des permissions

Modoboa étant une solution **multi-domaine**, il est nécessaire de pouvoir déléguer l'administration. Différents rôles prédéfinis sont disponibles, afin de rendre possible ce fonctionnement :

- *Utilisateur simple* : rôle le moins permissif. Il permet uniquement d'accéder aux fonctionnalités orientées « utilisateur » telles que le webmail ;
- *Administrateur de domaine(s)* : peut gérer un ou plusieurs domaine(s) (création, modification et suppression d'objets dans le domaine, personnalisation du paramétrage Amavis par domaine) ;
- *Revendeur* (si extension *Limits* activée) : un administrateur de domaines qui, de plus, peut affecter des limites (du type nombre maximal d'objets) aux administrateurs de domaines ;
- *Super administrateur* : peut tout faire depuis l'interface.

L'affectation de ces rôles se fait lors de la création/modification d'un compte utilisateur.

Fig. 7 : Sélection du rôle d'un nouveau compte

Complétez votre collection d'anciens numéros !

boutique.ed-diamond.com

numerique.ed-diamond.com

1.1.5 Gestion des quotas

Modoboa permet la définition rapide de quotas. Un quota consiste à limiter l'espace disque utilisé par une boîte aux lettres. Dovecot permet de limiter la taille d'une boîte et/ou le nombre de messages qu'elle contient. Seule la limitation de la taille est possible via Modoboa. Les quotas se définissent :

- au niveau d'un domaine : tous les comptes créés par la suite utiliseront par défaut la valeur du domaine ;
- au niveau d'une boîte : permet d'ignorer la valeur du domaine au cas où celle-ci ne conviendrait pas.

> **Note**
>
> Seuls les super administrateurs peuvent assigner, au niveau d'une boîte, un quota supérieur à la valeur du domaine.

Le suivi des quotas, quant à lui, fait l'objet d'une page spécifique dans la section ***Identités*** (Fig. 8).

1.1.6 Limiter les créations

Il est possible d'affecter des limites de création aux administrateurs de domaines. Cette fonctionnalité peut s'avérer utile pour contrôler l'utilisation des ressources système (afin de ne pas surcharger la plateforme), ou pour mettre en place une grille tarifaire (ISP). Ces limites peuvent concerner :

- le nombre de domaines et alias de domaines ;
- le nombre de domaines « relais » et alias de domaines « relais » ;
- le nombre de comptes et alias ;
- le nombre d'administrateurs de domaines.

La définition de ces valeurs se fait depuis l'onglet ***Ressources***, disponible dans le formulaire de modification d'un compte. Des valeurs par défaut utilisées lors de la création d'un nouveau compte peuvent être définies depuis l'écran de paramétrage (onglet ***Limites***).

Un *widget* de suivi de la consommation est affiché aux administrateurs de domaines depuis la page ***Identités*** (Fig. 9).

Fig. 9 : Widget de suivi des ressources

1.2 Gestionnaire Amavis

Amavis est un outil puissant, pratique, mais difficile à exploiter pour le commun des mortels. Premièrement, il est écrit en Perl (troll activé) et toute blague mise à part, il impose presque la connaissance de ce langage à ses utilisateurs... Modoboa tente de rendre plus digeste son exploitation grâce à une extension dédiée qui permet :

- une gestion des messages mis en quarantaine ;
- un paramétrage personnalisé par domaine.

1.2.1 Paramétrage par domaine

Les administrateurs de domaines (et rôles supérieurs) peuvent personnaliser le comportement d'Amavis pour un domaine donné.

Fig. 10 : Personnalisation Amavis par domaine

Toutes les possibilités de personnalisation ne sont pas encore disponibles, mais celles présentes sont déjà intéressantes :

- Activation/désactivation du filtrage des spams/virus/messages à bannir ;
- Seuils de détection du spam ;

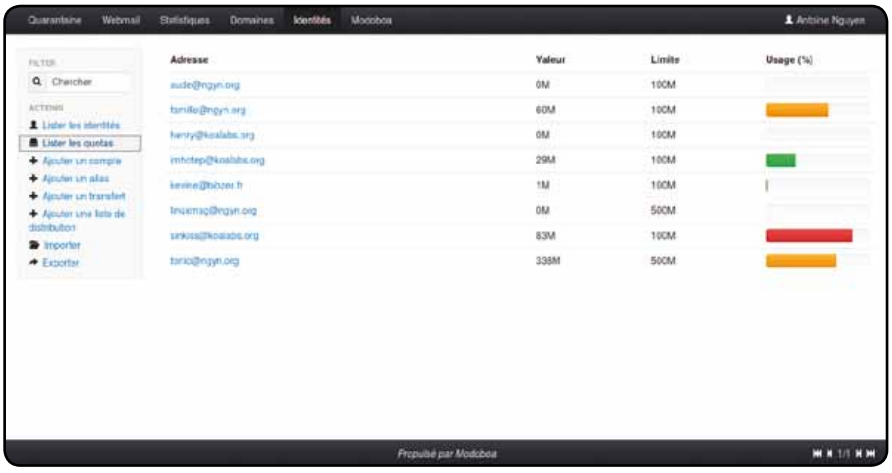


Fig. 8 : Liste des quotas avec leur consommation actuelle

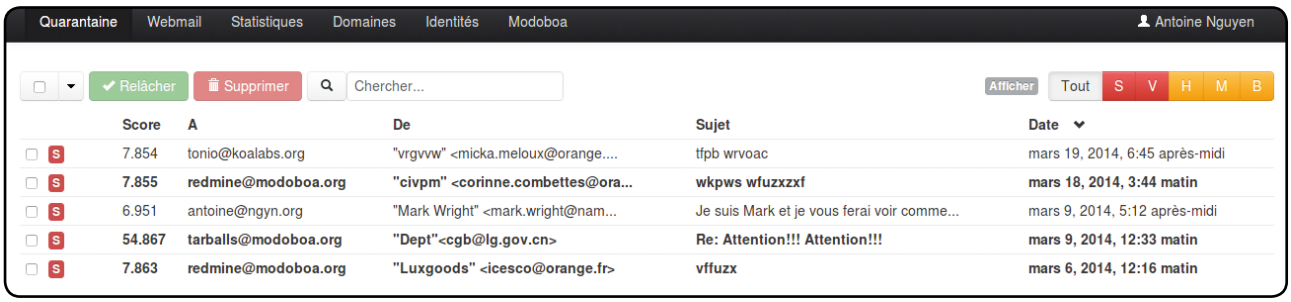


Fig. 11 : Affichage du contenu de la quarantaine

- Insertion d'un message personnalisé (type ****SPAM****) dans le sujet des messages concernés.

1.2.2 Quarantaine

Le gestionnaire de quarantaine (menu ***Quarantaine*** dans la barre noire en haut de page) permet aux utilisateurs d'accéder aux messages mis en quarantaine par Amavis.

Les possibilités d'exploitation de la quarantaine s'adaptent en fonction du rôle d'un utilisateur :

- Les utilisateurs simples n'ont accès qu'aux messages dont ils sont le destinataire (alias compris) ;
- Les administrateurs de domaines ont accès à l'ensemble du trafic concernant leurs domaines (alias compris).

Note

La libération d'un message étant une opération sensible, les utilisateurs simples n'y ont pas accès par défaut. À la place, ils peuvent envoyer une requête de libération à leur administrateur, qui décidera de la suite. Ce comportement est modifiable depuis l'écran de paramétrage.

Pour que la libération fonctionne correctement, rendez-vous dans l'écran de paramétrage, onglet ***Amavis***, et changez la valeur du paramètre ***Mode de connexion à Amavis*** de `unix` à `inet`.

Tout comme un webmail, le gestionnaire de quarantaine permet de consulter un message pour en vérifier le contenu (texte seulement) (Fig. 12).

Pour inciter les utilisateurs à gérer leurs messages, Modoboa propose un système de notification par e-mail, qui les prévient automatiquement dès qu'un message leur étant destiné est mis en quarantaine. Ce message contient des instructions permettant d'agir sur le(s) message(s) sans s'authentifier.

Note

Cette fonctionnalité n'est disponible qu'avec Amavis 2.8.0 et supérieur. Consultez la documentation de Modoboa pour plus de détails (http://modoboa.readthedocs.org/en/1.1.0/getting_started/plugins.html#self-service-mode).

2 Tests de bon fonctionnement

Modoboa ne fournit pas (encore ?) de mécanisme automatique pour valider qu'un serveur mail fonctionne bien. Vous l'aurez remarqué dans cet article, beaucoup de paramètres sont nécessaires afin d'obtenir un résultat, le nombre potentiel de fautes de frappe étant proportionnel. Beaucoup de tests peuvent être effectués lors d'une validation. Nous allons nous concentrer sur les plus évidents :

- Connexion d'un utilisateur à sa boîte via IMAP ;
- Envoi d'un message ;
- Réception d'un message ;
- Réception d'un spam.

Fig. 12 : Affichage d'un spam

2.1 Accès utilisateur via IMAP

Nous allons utiliser Thunderbird, le client mail de Mozilla, pour valider cette partie. Cela nous permettra de vérifier le bon fonctionnement d'automx par la même occasion.

> **Note**
>
> Valider le fonctionnement d'automx implique que la configuration de votre zone DNS est correcte et qu'elle a été propagée.

Avant d'ouvrir votre client de messagerie, n'oubliez pas de créer votre domaine, ainsi qu'un compte utilisateur dans Modoboa ;-)

Depuis Thunderbird, cliquez sur ***Édition > Gestion des comptes,*** puis dans la liste déroulante ***Gestion des comptes*** (en bas à gauche), sélectionnez ***Ajouter un compte de messagerie...*** Renseignez les informations demandées, cliquez sur ***Continuer*** et Thunderbird s'occupe du reste ! Si tout se déroule comme prévu, vous devriez apercevoir une fenêtre ressemblant à celle-ci :

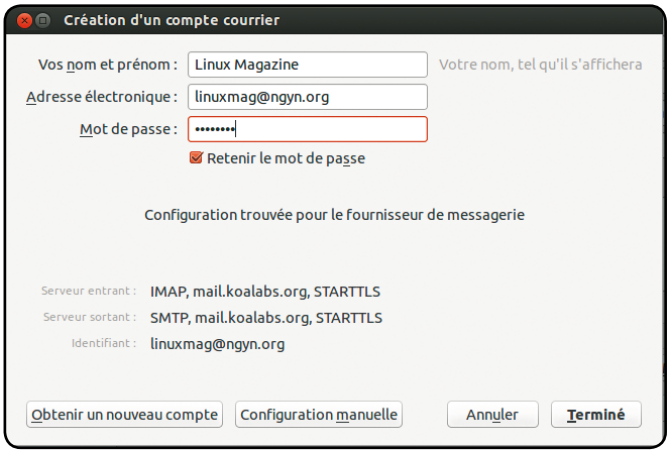


Fig. 13 : Ajout d'un compte de messagerie

Le message ***Configuration trouvée pour le fournisseur de messagerie*** vous indique qu'automx fonctionne parfaitement. Cliquez sur ***Terminé*** pour finaliser l'opération.

Fig. 14 : Liste des boîtes aux lettres du compte créé

Retournez sur l'écran principal de Thunderbird. Dans la liste de gauche (boîtes aux lettres), sélectionnez le dossier ***Courrier entrant*** situé sous le compte que vous venez de créer. Si Dovecot est configuré correctement, vous verrez apparaître une liste de dossiers en plus de ***Courrier entrant*** (ceux configurés plus tôt dans la section « Dovecot / Boîtes aux lettres »).

2.2 Envoi d'un message

Muni de votre compte fraîchement configuré dans Thunderbird, tentez d'envoyer un message vers une adresse mail extérieure à votre serveur (vous en avez bien une qui traîne quelque part). Le résultat attendu est la bonne réception du message de l'autre côté. S'il s'avérait que cela ne fonctionne pas, je vous invite à consulter de très près le contenu du fichier `/var/log/mail.log` de votre serveur. Vous y trouverez plein d'informations utiles :-)

2.3 Réception d'un message

Toujours muni de votre merveilleux compte, faites cette fois-ci l'opération inverse : envoyez-vous un message depuis l'adresse extérieure au serveur. Le résultat attendu est la réception de ce message par votre serveur et une livraison dans la boîte aux lettres qui va bien (celle créée un poil plus tôt).

En cas de souci, n'hésitez pas à inspecter le fichier `/var/log/mail.log` pour tenter de comprendre ce qui se passe mal.

> **Note**
>
> Attention à votre configuration DNS ! Si elle n'est pas encore propagée, le monde extérieur ne pourra pas vous envoyer de message.

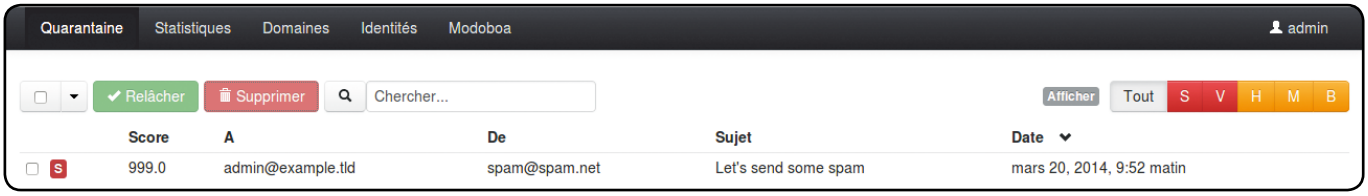


Fig. 15 : Spam mis en quarantaine

2.4 Réception d'un spam

Valider le bon fonctionnement d'Amavis peut sembler compliqué. Comment simuler l'envoi d'un spam ? La communauté SpamAssassin s'est déjà posé cette question et la réponse est GTUBE (*Generic Test for Unsolicited Bulk Email*). Il s'agit d'un contenu de test dont la signature est reconnue par les moteurs comme SpamAssassin. Plus d'informations sur http://spamassassin.apache.org/gtube/.

Vous trouverez sur le site de Modoboa un script prêt à l'emploi vous permettant d'utiliser ce test. Connectez-vous sur une machine différente de votre serveur de messagerie (c'est important) et exécutez les commandes suivantes :

```
$ wget http://modoboa.org/resources/send_spam.py
$ python send_spam.py --server-address <IP> spam@spam.net user@example.tld
```

Remplacez **<IP>** par l'adresse IP de votre serveur et assurez-vous que la dernière adresse (ici **user@example.tld**) existe bien chez vous. Si Amavis est correctement configuré, vous apercevrez une ligne comme celle-ci dans le fichier **/var/log/mail.log** :

```
Mar 20 15:35:33 modoboa amavis[17653]: (17653-01) Blocked SPAM
{BouncedInternal,Quarantined}, LOCAL [192.168.1.40]:39170 [192.168.1.40]
<spam@spam.net> -> <user@example.tld>, quarantine: Qqf-_twfE8MM, Message-ID:
<20140320143532.7380.22319@picasso>, mail_id: Qqf-_twfE8MM, Hits: 999, size:
1180, 887 ms
```

De plus, le message envoyé sera consultable depuis le gestionnaire de quarantaine (Fig. 15).

Conclusion

J'espère que cet article vous a convaincu que la gestion de sa propre plateforme de messagerie est plus digeste grâce à l'utilisation de Modoboa. Je me suis concentré sur le côté administratif, mais sachez que des outils orientés « utilisateur » sont aussi disponibles : webmail, création de filtres et autres. Ils feront peut-être l'objet d'un futur article.

Quoiqu'il en soit, Modoboa est un logiciel qui évolue tous les jours et qui séduit de plus en plus d'utilisateurs (particuliers et professionnels). Malgré des lacunes dues à son jeune âge, il représente une véritable alternative libre et intègre aux solutions fermées et/ou commerciales telles que Gmail ou autre ! ■

WEBSOCKET, LE WEB CONNECTÉ

par **Sébastien Chazallet** *[Ingénieur logiciels libres]*

Le protocole WebSocket vise à permettre d'établir un canal de communication bidirectionnel entre le client (navigateur) et le serveur et de la maintenir. Il est en cours de standardisation par le W3C, mais est déjà utilisable. Cet article présente un exemple d'implémentation utilisant le tout nouveau module asyncio de Python 3.4 et en créant un petit client JavaScript pur, dont l'objectif est de transmettre une quantité importante de données. En procédant ainsi, la page s'affiche rapidement et l'utilisateur ne se retrouve pas devant une page figée, mais devant une page déjà bien structurée qui se remplit au fur et à mesure.

Dans le monde du Web, la performance est un aspect extrêmement important. Les intégrateurs de site de e-commerce savent bien qu'une page doit arriver en 0,4 seconde au maximum et qu'au-delà, cela coûte 10 % d'audience par dixième de seconde supplémentaire.

Usuellement, lorsque le client demande une page, elle est construire par le serveur et envoyée ensuite au client qui n'a plus qu'à la lire, la déchiffrer et l'afficher. Or, dans le cas d'une page contenant beaucoup de données, la page prend du temps pour être créée côté serveur, prend du temps à être transférée et du temps à être affichée. Pendant tout ce temps, le navigateur donne l'impression d'être figé, alors qu'il est soit en attente, soit en train de travailler.

Toujours est-il que pour l'utilisateur, cette situation n'est pas idéale. Dans le cas d'un tableau de données, la solution classique consiste à utiliser la pagination et à faire des requêtes Ajax pour aller chercher les données d'une page précise. En limitant ainsi la quantité de données à transmettre, on améliore la performance. Les architectures Ajax sont plébiscitées à juste titre, car elles permettent de répondre parfaitement à ce besoin.

Cependant, si l'on tient absolument à afficher toutes les données dans une page, cette solution est limitée par le fait que l'on est synchrone (on envoie la requête, on attend qu'elle soit traitée, on récupère la réponse qui contient toutes les données et on les traite). Le temps de traitement peut être long et tant que le processus n'est pas terminé, la page est figée. Dans ce cas-là, la solution consiste alors à transmettre une page contenant tout sauf la donnée et un bout de code JavaScript qui va utiliser un websocket pour remplir les données de façon asynchrone. On voit ainsi les données arriver peu à peu et on peut même être capable d'afficher un avancement, de manière à informer l'utilisateur que les données sont en train d'être acheminées. C'est une solution idéale à la fois lorsqu'il y a trop de données, mais également lorsqu'il y a peu de données, mais qu'elles sont longues à recouvrer. Mais ce n'est pas la seule utilité du protocole WebSocket... En effet, il permet également au serveur de notifier le client, ce qui pourrait permettre, par exemple, l'implémentation du protocole MVC. Mais on est ici hors du spectre de l'article.

1 Installer Python 3.4

Python 3.4 est la toute dernière version de Python qui porte avec elle le module `asyncio` permettant

de gérer superbement bien l'asynchrone. Il la faut donc. Cependant, comme vous avez peut-être une version packagée de Python 3, ce n'est peut-être pas une bonne idée de la remplacer. Il faut donc installer Python 3.4 à côté d'un éventuel Python 3 packagé pour votre distribution.

Pour commencer, il faut mettre le système à jour et s'assurer d'avoir deux paquets importants :

```
# aptitude update
# aptitude full-upgrade
# aptitude install libssl-dev openssl
```

Ensuite, il faut utiliser le traditionnel (l'utilisation du préfixe n'est pas obligatoire) :

```
# ./configure && make && make altinstall
```

Normalement, si tout s'est bien passé, ces commandes devraient fonctionner et vous donner les emplacements des exécutables utiles. Dans notre cas :

```
# which python3.4
/usr/local/bin/python3.4
# which pip3.4
/usr/local/bin/pip3.4
```

1.1 Installer les modules utiles

L'exemple présenté va utiliser les modules **csv** et **json**, qui font partie de la bibliothèque standard, mais également le nouveau module **asyncio**, également dans la bibliothèque standard. Il n'y a donc rien de particulier à faire pour cela.

Le module **csv** va nous permettre de lire un fichier CSV qui contiendra les données à transmettre, tandis que le module **json** permettra de convertir ces données dans un langage que JavaScript comprendra.

On va également utiliser le module **websocket**, lequel porte le nom du protocole qu'il permet de mettre en place. C'est lui qui nous permettra de réaliser toute la partie serveur.

```
# pip3.4 install websockets
```

Une fois ceci prêt, on peut commencer à réaliser la partie serveur.

2 Partie serveur

2.1 Lire le fichier CSV

Quitte à faire un article sur Python, autant réviser les fondamentaux. En Python, la lecture d'un fichier CSV est une formalité :

```
import csv

BUFFSIZE = 1 << 12

def csvfile_line_generator(filename, *, encoding='utf8'):
    """Read a CSV file and yield each line"""
    with open(filename, encoding=encoding) as csvfile:
        dialect = csv.Sniffer().sniff(csvfile.read(BUFFSIZE))
        csvfile.seek(0)
        reader = csv.DictReader(csvfile, dialect=dialect)
        yield from reader
```

L'élément clé est de lire un peu le fichier pour détecter le dialecte, puis de le relire en entier.

Ensuite, on utilise l'expression **yield from** qui a été introduite en Python 3, ce qui nous permet d'avoir ici un générateur et nous permet de commencer à envoyer les données alors même que l'on n'a pas terminé de lire le fichier.

2.2 Convertir en JSON

Il faut créer un générateur qui va utiliser le générateur précédent. La solution la plus simple est d'utiliser un décorateur, afin de décorer la fonction précédente. D'autre part, au lieu de réaliser une itération et d'utiliser un autre mot-clé **yield**, on va utiliser la fonction **map** qui est également un générateur et dont l'utilisation est plus élégante :

```
import json

def json_converter(gen):
    def wrapped(*args, **kwargs):
        return map(json.dumps, gen(*args, **kwargs))
    return wrapped
```

Il faut maintenant décorer la fonction précédente de la section 2.1 :

```
@json_converter
def csvfile_line_generator(filename, *, encoding='utf8'):
```

L'avantage de procéder ainsi est que la fonction de conversion vers JSON est réutilisable sur n'importe quel générateur ou itérateur.

2.3 Créer une partie serveur

De manière à envoyer les données, nous allons utiliser ici un motif de conception de type producteur – consommateur. Le producteur va se contenter d'envoyer toutes les données, une par une, tant que le websocket est ouvert, en itérant simplement sur la fonction précédente :

```
import asyncio

@asyncio.coroutine
def producer(websocket, uri):
    gen = csvfile_line_generator('comsimp2014.txt', encoding='latin1')
    for data in gen:
        if websocket.open:
            yield from websocket.send(data)
```

On peut noter l'utilisation de **yield from** qui permet de résoudre élégamment la problématique qui consiste à envoyer la donnée par morceaux (en fonction de sa taille) en respectant le protocole réseau, le tout réalisé à haut niveau. Le fichier utilisé ici (**comsimp2014.txt**) est issu de http://www.insee.fr/fr/methodes/nomenclatures/cog/telechargement/2014/txt/comsimp2014.zip. Ce dernier contient la liste des communes françaises en 2014.

Il faut maintenant lancer ce serveur :

```
import websockets

WS_SERVER, WS_PORT = 'localhost', 8765

if __name__ == '__main__':
    start_server = websockets.serve(producer, WS_SERVER, WS_PORT)
    asyncio.get_event_loop().run_until_complete(start_server)
    asyncio.get_event_loop().run_forever()
```

Il est intéressant de noter les responsabilités relatives des deux modules **websocket** et **asyncio**.

2.4 Créer une partie cliente

Notre travail est normalement terminé, cependant, il est totalement inutile d'aller commencer une partie cliente côté JavaScript si l'on n'a pas testé plus simplement notre partie serveur.

Pour cela, le plus simple est de réaliser le consommateur correspondant au producteur écrit plus haut :

```
@asyncio.coroutine
def consumer():
    websocket = yield from websockets.connect(WS_URI)
    while websocket.open:
        yield from asyncio.sleep(1)
        data = yield from websocket.recv()
        print(">>> %s" % data)
```

On utilise toujours les mêmes principes, à savoir les mots-clés **yield from** et l'itération, tant que le socket est ouvert. On notera également que pour égayer un peu le processus, on ajoute une pause d'une seconde côté client, histoire de voir les lignes qui défilent tranquillement.

Il ne reste plus qu'à utiliser ce consommateur en ajoutant temporairement une petite ligne :

```
if __name__ == '__main__':
    start_server = websockets.serve(producer, WS_SERVER, WS_PORT)
    asyncio.get_event_loop().run_until_complete(start_server)
    asyncio.get_event_loop().run_until_complete(consumer())
    asyncio.get_event_loop().run_forever()
```

En lançant le script, vous devriez voir les données des différentes communes s'afficher, une toutes les secondes.

Dès que cela fonctionne, il ne reste plus qu'à réaliser la partie cliente.

3 Partie cliente

3.1 Structure de base

Pour cette partie, on décide de ne pas recourir à une bibliothèque externe, mais de tout faire à la main, en utilisant simplement le JavaScript lui-même. La raison est simplement que l'on veut bien montrer que le websocket n'est pas quelque chose qui est lié à une bibliothèque, mais quelque chose qui est lié au langage. On ne se préoccupe pas du tout de la disposition des blocs dans la page, ou plus généralement des feuilles de styles, de manière à ne pas alourdir le code.

Pour la structure de base, on souhaite avoir une page web contenant trois espaces permettant respectivement d'afficher d'une part le nombre de communes et un bouton pour interrompre le chargement, d'autre part les messages d'information et enfin, les messages d'erreur. Ensuite, on affiche un tableau qui va lister les villes, leur numéro de département et de région.

Le code HTML est donc minimaliste :

```
<!DOCTYPE html>
<html>
    <head>
        <title>WebSocket Test</title>
        <meta charset="UTF-8" />
        <script>
            // Pour une meilleure lisibilité le code JavaScript
            // a été extrait du fichier (voir dans la suite)
        </script>
    </head>

    <body>
        <div>
            <h1>WebSocket Test</h1>
            <span id="number"></span> Villes ajoutées.
            <input type="button" value="Close"  onclick="_WS.close();"/>
        </div>
        <div id="infos">
            <h2>Infos</h2>
        </div>
        <div id="errors">
            <h2>Errors</h2>
        </div>
        <div id="output">
            <table>
                <thead>
                    <th>N°</th>
                    <th>Ville</th>
                    <th>Departement</th>
                    <th>Région</th>
                </thead>
                <tbody></tbody>
            </table>
        </div>
    </body>
</html>
```

Il suffit de veiller à donner des identifiants corrects aux éléments qui nous intéressent. On notera le bouton qui permet de terminer le websocket et le fait qu'il appelle une méthode particulière que nous allons détailler.

La première étape consiste à initialiser un objet **WebSocket** lors du chargement de la page, ce qui se fait en écrivant une méthode nommée **init** et en liant l'appel de cette méthode à l'événement **load** de la fenêtre.

L'objet **WebSocket** a quatre méthodes à renseigner ou *hooks*. Il s'agit des méthodes **onopen**, **onclose**, **onmessage** et **onerror**. Il faut donc créer ces méthodes et faire en sorte qu'elles soient liées lors de l'initialisation.

On doit ensuite ajouter la méthode **close** liée au bouton vu plus haut.

Le tout donne ceci :

```
var _WS = {
    uri: 'ws://localhost:8765/',
    ws: null,
    init : function (e) {
        _WS.s = new WebSocket(_WS.uri);
        _WS.s.onopen = function (e) { _WS.onOpen(e); };
        _WS.s.onclose = function (e) { _WS.onClose(e); };
        _WS.s.onmessage = function (e) { _WS.onMessage(e); };
        _WS.s.onerror = function (e) { _WS.onError(e); };
    },
    onOpen: function () {
    },
    onClose: function () {
    },
    onMessage: function (e) {
    },
    onError: function (e) {
    },
    close: function () {
    },
};

window.addEventListener('load', _WS.init, false);
```

Maintenant que le squelette du client est fait, nous pouvons commencer à jouer un peu.

3.2 Connexion, déconnexion

Lorsque la page est chargée, la connexion est faite et immédiatement, le serveur envoie son premier message. Une fois que tous les messages sont envoyés, la connexion est fermée.

On souhaite écrire des messages dans la bonne **div** à chaque fois qu'un événement se passe. Pour cela, on va créer une fonction supplémentaire, nommée **write** :

```
write: function (message, where, balise) {
var e = document.createElement(balise);
    e.innerHTML = message.toString();
    document.getElementById(where).appendChild(e);
},
```

Les trois arguments permettent de préciser le message (qui peut être du texte ou du code HTML), l'identifiant de la **div** à laquelle il faudra ajouter le message et le type de balise qui doit être utilisé (**div**, **span**, **p**, …).

On peut en profiter pour mesurer le temps entre l'ouverture et la fermeture du socket pour savoir combien de temps dure le processus. Pour cela, il faut déclarer deux variables (comme on a déclaré l'**uri** plus haut) :

```
startTime:null,
elapsedTime:0,
```

Puis, il suffit de décider de l'action à réaliser lorsque le socket est ouvert (ici, écrire un message et stocker l'instant courant) et lorsqu'il est fermé (calculer le temps écoulé et écrire un message) :

```
onOpen: function () {
    _WS.startTime = new Date().getTime();
    _WS.elapsedTime = 0;
    _WS.write('CONNECTED', 'infos', 'p');
},
onClose: function () {
    _WS.refresh();
    if (_WS.startTime != null) {
        _WS.elapsedTime = new Date().getTime() - _WS.startTime;
    }
    _WS.write('DISCONNECTED: ' + _WS.elapsedTime + ' ms', 'infos', 'p');
},
```

L'explication sur l'appel de la fonction **refresh** est donnée dans la section suivante.

On peut également en profiter pour gérer aussi l'interruption volontaire du websocket :

```
close: function () {
    _WS.write('GOODBYE !', 'infos', 'p');
    _WS.s.close();
}
```

Ainsi que la gestion des erreurs :

```
onError: function (e)
    _WS.write('ERROR: ' + e.data, 'errors', 'p');
},
```

À cet instant, utiliser le client devrait fonctionner, les messages devraient apparaître, mais les données ne seront pas traitées. Il s'agit de la dernière étape.

3.3 Utilisation des données

Nous souhaitons deux choses : afficher les données dans un tableau et afficher le nombre de données traitées, en temps réel.

Pour la première partie, nous créons une méthode **append**, destinée à ajouter une donnée au tableau (donnée brute au format JavaScript, ici un tableau associatif ou dictionnaire ou *mapping*) :

```
table_content: document.getElementById('output').children[0].children[1]
append: function (data) {
    var tr = document.createElement('tr');
    var td1 = document.createElement('td');
```

```
            td1.innerHTML = _WS.number;
            tr.appendChild(td1);
            var td2 = document.createElement('td');
            td2.innerHTML = data['NCCENR'];
            tr.appendChild(td2);
            var td3 = document.createElement('td');
            td3.innerHTML = data['DEP'];
            tr.appendChild(td3);
            var td4 = document.createElement('td');
            td4.innerHTML = data['REG'];
            tr.appendChild(td4);
            _WS.table_content.appendChild(tr);
        },
```

On stocke préalablement l'élément **tbody** du tableau, de manière à ne pas avoir à le récupérer à chaque ajout, puis on crée à l'aide des fonctionnalités DOM de JavaScript une ligne de 4 colonnes.

Pour la seconde partie, nous avons créé un endroit spécialement pour cela dans le code HTML. Il suffit d'aller y inscrire le bon chiffre, ce qui se fait à l'aide d'une fonction **refresh** :

```
        refresh: function() {
            document.getElementById('number').innerHTML = _WS.number;
        },
```

On décide d'appeler cette fonction une fois toutes les 100 données traitées :

```
        number: 0,
        onMessage: function (e) {
            _WS.number++;
            if (_WS.number % 1000 == 0) {
                _WS.refresh();
            }
            _WS.append(JSON.parse(e.data));
        },
```

On note que la donnée JavaScript transmise à la méthode **append** est récupérée à l'aide de **JSON.parse** ; il s'agit de la donnée envoyée par le serveur Python asynchrone.

Conclusion

Le WebSocket en est à ses balbutiements, il s'agit cependant d'une technologie qui devrait connaître un bel avenir au fur et à mesure que ses spécifications avanceront.

En l'état actuel des choses, il y a déjà pas mal de choses à faire pour s'amuser un peu et dans ce cadre-là, Python se positionne clairement comme une très bonne solution, permettant de développer très rapidement côté serveur.

Pour télécharger le code, rendez-vous sur inspyration.org et n'hésitez pas à faire des retours ! ■

À LA DÉCOUVERTE D'ANDROID : SOURIEZ, VOUS ÊTES FILMÉ...

par ***Benjamin Zores*** *[architecte Android @ Alcatel-Lucent]*

Allons toujours plus loin dans la découverte du système Android, avec un aperçu d'un sous-système complexe et en pleine évolution : la caméra.

La course aux fonctionnalités continue dans le petit monde des smartphones et cela fait maintenant bien longtemps qu'il n'est plus possible pour les constructeurs de se restreindre aux simples fonctions téléphoniques. Le domaine de la photographie et de la vidéo est en plein essor et Android, comme tout système mobile, se doit de répondre à cette demande croissante. Avec la course incessante aux capteurs à dizaines de méga-pixels et aux fonctionnalités les plus diverses, voyons comment l'OS met en œuvre ces demandes.

1 Architecture logicielle

Pour les plus fidèles lecteurs de cette série, vous serez ravis d'apprendre que le sous-système de gestion de la caméra ne s'avère pas si différent des autres sous-systèmes d'Android, comme le représente la figure 1.

L'ensemble repose donc sur l'enchaînement des couches applicatives suivantes :

- **Framework applicatif Java**. Ce dernier implémente l'API `android.hardware.camera`, disponible via le SDK pour tout développeur applicatif. Ce canevas ne fait finalement qu'exposer le code natif au travers de la couche JNI.
- **Interface JNI**. Comme pour de nombreuses autres ressources matérielles, Android repose son interface sur une implémentation C++, pour des raisons de performance. L'interface avec le monde Java se fait alors au sein de la couche JNI, au travers du fichier `frameworks/base/core/jni/android_hardware_Camera.cpp`.
- **Framework natif C++**. Ce dernier trouve ses sources dans le fichier `frameworks/av/camera/Camera.cpp` et correspond à l'implémentation native (i.e. C++) de notre API. Les appels au framework sont ensuite redirigés au *Camera Service* au travers du proxy Binder.
- **Proxy IPC Binder**. Ce dernier fournit trois interfaces d'IPC :
  - `ICameraService`, une interface vers le service du même nom ;
  - `ICamera`, une interface vers un périphérique matériel donné ;
  - `ICameraClient`, une interface de retour entre le périphérique matériel et le framework applicatif.
- **Camera Service**. Comme de nombreux autres services Android, ce dernier tourne en tâche de fond (sous la forme d'un *daemon*) et

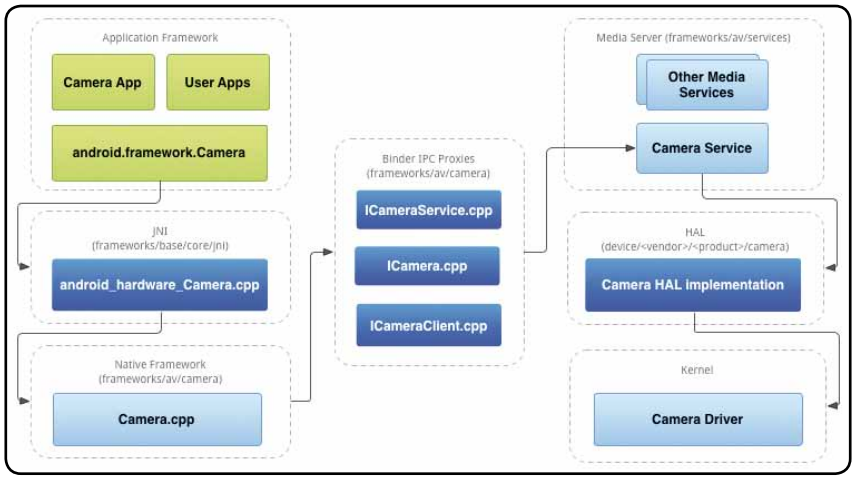


Fig. 1 : Architecture logicielle du sous-système de gestion de la caméra

est responsable de l'interfaçage avec le matériel, au travers de la HAL. Vous trouverez son implémentation au sein du fichier `frameworks/av/services/Camera/libcameraservice/CameraService.cpp` des sources de l'AOSP.

- **Couche d'abstraction matérielle HAL**. C'est, comme bien souvent, ici que les choses se compliquent. Cette (ou plutôt ces) dernière(s) s'occupe(nt) de s'interfacer avec votre matériel. C'est ici que se déroule toute la logique et l'intelligence du processing de votre périphérique. Cette brique est très souvent propriétaire.
- **Pilote noyau**. Il s'agit là d'un « classique » pilote de caméra pour Linux, utilisant le plus souvent l'API V4L2 (Video4Linux2). La seule contrainte pour ce dernier est d'être capable de supporter et d'exposer les formats d'images YV12 et NV21, de manière à offrir une capacité de pré-visualisation à l'écran et d'enregistrement vidéo (dit autrement, vous ne pouvez vous satisfaire d'un capteur qui prendrait une photo JPEG).

Voilà donc pour les bases. Mais il serait hélas trop simple de s'arrêter en si bon chemin. Android a en effet connu une petite révolution au cours des deux dernières années relatives à la gestion de la caméra.

2 État de l'art et limitations

Jusqu'à Android 4.0 (*Ice Cream Sandwich*), le système de gestion de la caméra (ou du moins de l'appareil photo) a été conçu selon une approche simple de type « viser et déclencher ». Comme pour un appareil photo classique, on vise, on procède à la mise au point et on prend la photo. Mais les photos prises manquent souvent de piqué, de couleurs, de contraste, etc. Force est de constater que le professionnel ou l'amateur confirmé procédera de toute manière à un post-traitement sur son ordinateur.

Mais pour le commun des mortels, ça ne sera pas le cas. La plupart des gens souhaitent prendre une photo et que cette dernière soit directement parfaite et utilisable, sans s'encombrer des détails. On rentre alors dans un nouveau paradigme : celui de la photographie numérique assistée (ou automatiquement post-produite), également appelée « Computational Photography » en anglais. L'ensemble des tâches de post-traitement doivent donc être conférées directement à la source, à l'appareil qui a pris la photo. Ceci permet la génération d'images complexes comme des panoramas (où un ensemble de clichés sont pris et associés l'un à l'autre pour former une image continue), de photos HDR (pour *High Dynamic Range*) où une même image est prise plusieurs fois à différentes expositions qui seront ensuite fusionnées pour générer une image sans qu'aucune partie ne soit sur- ou sous-exposée, etc.

D'autres fonctionnalités sont imaginées et possibles, comme la capacité de prendre plusieurs clichés à des profondeurs de champ différentes et de les ré-assembler ensuite pour créer une image où toutes les parties sont nettes, ou encore de pouvoir choisir lors de la visée sur l'écran d'appliquer des traitements (floutages, grain, amélioration…) de tout ou partie de l'image à prendre. Il est également possible d'effectuer des pré-traitements logiciels (donc avant le déclenchement), pour de la reconnaissance de visages ou de gestes, ou encore des travaux de réalité augmentée. Vous l'aurez compris, un travail de pré- ou post-traitement algorithmiquement complexe est donc nécessaire. Cela tombe bien, votre terminal Android dispose aujourd'hui de toute la puissance nécessaire pour réaliser ce genre d'opérations coûteuses !

Mais un problème se pose. Ces fonctionnalités n'ont pas été envisagées à la conception d'Android. Concrètement, cela signifie que l'API et surtout la HAL d'Android ne permettent de faire (dans leur version 1.0) qu'une pré-visualisation, une prise de photo, ou encore un enregistrement vidéo. Le traitement de photos en rafale, nécessaire pour du post-traitement HDR ou encore pour prendre des panoramiques, est extrêmement difficile à mettre en œuvre et en termes de métadonnées remontées (ou remontables) par la HAL, il est difficile de récupérer davantage que celles permettant la détection de visages.

Pour répondre à ces besoins, la HAL a dû être étendue et de nouvelles API doivent être offertes aux développeurs. Jelly Bean (4.1 à 4.3) et Kit Kat (4.4) ont donc introduit des changements majeurs en termes de HAL et d'API, comme le présente le tableau suivant (voir page suivante).

Notez que la dénomination est des plus complexes, mais que pour être compatible avec l'API v2 du module de caméra, votre périphérique doit implémenter la HAL v2 ou v3+.

3 Pipeline opérationnel

Dans la pratique, la stack Android liée à la gestion de la caméra se retrouvera responsable de l'ensemble d'opérations représentées au sein de la figure 2. La HAL aura bien évidemment la responsabilité de contrôle du matériel, qu'il s'agisse du capteur en lui-même, de l'optique ou encore du flash.

Version d'Android	Version du CAMERA_MODULE	Version du CAMERA_DEVICE (HAL)	Version du SDK	Fonctionnalités applicatives
4.0	1.0	1.0	14	Reconnaissance de visages
4.0.4	1.0	1.0	15	Stabilisation vidéo
4.1	1.0	1.0	16	Contrôle de l'autofocus
4.2	2	2	17	HDR, bruit de déclenchement
4.3	2.1	3	18	N/A
4.4	2.2	3.1	19	N/A

Relation entre les versions d'Android et celles des API applicatives et de la HAL

L'utilisation correcte de cette dernière produira une image (ou un flux vidéo brut YV12) à laquelle seront appliqués un ou plusieurs des algorithmes dits « 3A » :

- **AF** (*Auto Focus*) : contrôle automatique de la mise au point et de la distance focale ;
- **AWB** (*Auto White Balance*) : contrôle automatique de la balance des blancs ;
- **AE** (*Auto Exposure*) : contrôle automatique de l'exposition (entrée lumineuse).

L'emploi de ces algorithmes constitue généralement la « recette secrète » (i.e. comprendre « propriétaire ») de l'ISP (ou *Image Signal Processor*), module matériel associé au SoC ou au capteur (les deux options sont possibles) et permettant le traitement du signal. Le contrôle de ces trois fonctionnalités est du ressort complet de la HAL (encore une fois, très souvent secrètement conservée). Différents autres algorithmes peuvent ensuite être chaînés, tels que la réduction de bruit, l'ajustement de couleurs, la correction géométrique, le renforcement des contours, etc. L'image ainsi générée sera ensuite agrandie ou réduite à la dimension désirée, puis envoyée sous la forme d'un buffer avec les métadonnées qui lui sont associées. Voilà pour le parcours du pipeline ! Comprenez bien que ce pipeline est une vue logique du matériel. La HAL ne fait qu'ordonnancer différents sous-ensembles matériels, afin d'extraire une image brute (RAW) du capteur et de là, fournir une image YUV exploitable. Notez qu'en usage vidéo, et contrairement aux images graphiques à représentation informatique qui sont enregistrées en niveaux de couleurs RVB (ou RGB), les flux vidéos sont représentés en niveaux de chrominance et de luminance (YUV), l'œil humain étant plus sensible aux variations d'intensité lumineuse qu'aux variations de couleurs.

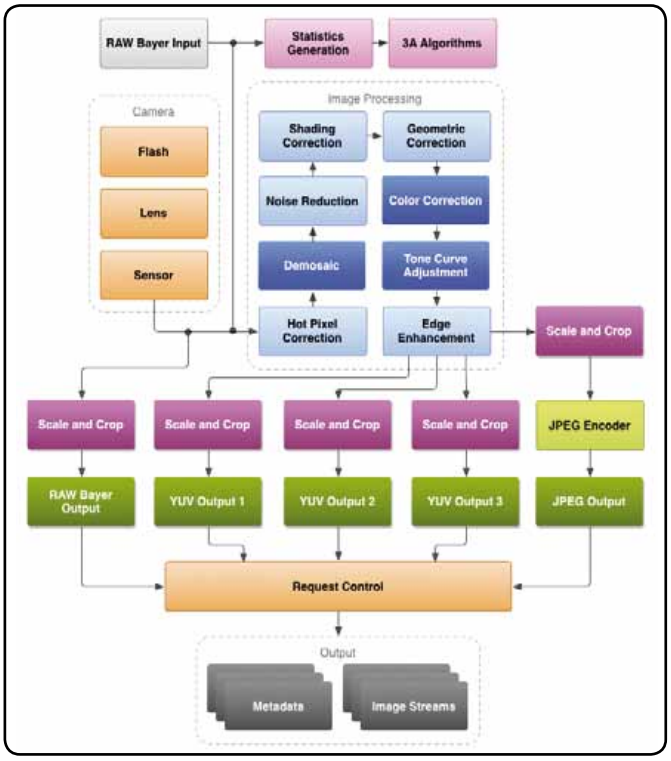


Fig. 2 : Pipeline opérationnel de traitement des données

4 HAL v3

Comme nous venons de le voir plus tôt, de grands changements sont intervenus depuis ICS et la HAL a connu trois révisions majeures. En pratique, cette (ou plutôt ces) dernière(s) se retrouve(nt) au sein du répertoire **`hardware/libhardware/include/hardware`** des sources de l'AOSP. Vous y trouverez donc respectivement les fichiers **`camera_common.h`**, **`camera.h`**, **`camera2.h`** et **`camera3.h`**. Vous l'aurez compris, en tant que constructeur, vous pouvez donc (avec Kit Kat), choisir d'implémenter l'une des trois versions possibles de HAL pour votre caméra.

Si votre périphérique supporte aujourd'hui la v1 et que vous souhaitez faire le saut, je ne saurai que trop vous recommander de passer directement à la v3.1. La version 2.0 de la HAL n'a en effet que peu vécu et est déjà marquée comme obsolète. Plus vicieux même, plusieurs HAL peuvent tout à fait cohabiter. Vous pouvez disposer d'une

caméra frontale compatible v1 et d'une caméra arrière compatible v3. Et de même, votre caméra arrière pourrait même (et c'est l'usage) être compatible v1 et v3, car implémentée par deux HAL. Et c'est d'ailleurs l'usage qui en est fait pour le peu de terminaux aujourd'hui compatibles avec la HAL v3 (à savoir le Nexus 7 et le Nexus 5). Pour la simple et bonne raison qu'aujourd'hui, en termes de SDK, seule l'API `android.hardware.camera` est exposée.

Aussi, si la HAL v3 est considérée comme stable (et donc prête à être implémentée par les constructeurs), l'API pour développeurs ne l'est pas. Cette dernière n'est d'ailleurs même pas exposée. Le code existe dans le framework application, mais les symboles sont tagués `{@hide}` et donc non exploitables. Le package applicatif Java s'appelle `android.hardware.camera2` **[1]**, mais la dénomination « ProCamera » se retrouve également dans les sources de l'AOSP. Si vous cherchez néanmoins à utiliser cette dernière pour commencer à développer vos applications, cela peut se faire via quelques astuces. Commencez par récupérer un terminal tournant sous Kit Kat et récupérez-en le framework Java :

```
adb pull /system/framework/core.jar .
adb pull /system/framework/framework.jar .
```

Puis, convertissez-le de DEX vers JAR pour pouvoir l'utiliser sous Eclipse :

```
dex2jar core.jar
dex2jar framework.jar
```

Enfin, sous Eclipse, ajoutez les JAR générés (`core-dex2jar.jar` et `framework-dex2jar.jar`) au sein du menu ***Projet > Propriétés > Chemin Java > Librairies > Ajouter des JAR externes***. Et le tour est joué ;-) Les plus intéressés trouveront un exemple d'application utilisant cette nouvelle API au sein de la ressource donnée en référence **[2]**.

Mais revenons plus en détails sur les changements introduits par nos différentes HAL :

- **1.0** (`camera.h`) : il s'agit là de la première HAL relative à la gestion de la caméra. Elle fut introduite avec Android 4.0 (ICS) comme une conversion 1:1 de l'ancienne couche d'abstraction C++ `CameraHardwareInterface`. Elle offre une compatibilité complète avec l'API `android.hardware.camera` du SDK.
- **2.0** (`camera2.h`) : cette dernière fut introduite avec Jelly Bean 4.2 et supporte également l'API du SDK. Elle étend les fonctionnalités de la HAL v1 avec davantage de possibilités de contrôles manuels et la fonctionnalité ZSL (*Zero Shutter Lag*) qui permet les déclenchements multiples de prises de vue sans délai. Cette dernière nécessite des capteurs à capacités étendues.
- **3.0** (`camera3.h`) : cette dernière fut introduite avec Jelly Bean 4.3 et présente une rupture complète de l'API de la HAL (contrairement à la v2), mais conserve les prérequis matériels de la v2. Cette dernière ré-invente les mécanismes de synchronisation et la gestion des actions. Les événements déclencheurs (*triggers*) deviennent des requêtes (*requests*) et les notifications (*notifications*) résultantes deviennent des résultats (*results*).
- **3.1** (`camera3.h`) : cette dernière fut introduite avec Kit Kat 4.4 et ajoute simplement la commande `flush()`, permettant d'annuler toutes les requêtes en queue (et les buffers associés), aussi vite que possible.

À noter que les intéressés pourront trouver (chose suffisamment rare pour le préciser), l'implémentation open source de la HAL v3 du Nexus 5 pour SoC Qualcomm Snapdragon 800 **[3]**.

5 Séquencement opérationnel

Tâchons maintenant de comprendre le séquencement opérationnel au travers de la très complexe figure 3 (voir page suivante). Je souhaite d'ailleurs remercier Google pour la mise en place de ce travail de documentation.

La première chose à faire pour une application désireuse d'utiliser la caméra est de l'identifier. Les appels successifs aux fonctions `getDeviceIdList()` et `getCameraProperties()` du `CameraManager` permettent ainsi de récupérer un `CameraProperties` décrivant les caractéristiques physiques de la caméra (matérielle) à utiliser, ainsi que sa configuration. Il suffit ensuite d'ouvrir l'accès au périphérique via l'appel à `openCamera()` du `CameraManager`, pour se retrouver avec une instance de `CameraDevice`. L'application a alors en charge d'y attacher un ou plusieurs objet(s) de type `Listener`. Ceci permet par exemple à l'application d'être notifiée dès que le matériel est disponible ou ne l'est plus.

Il est ensuite nécessaire de demander à la caméra l'acquisition d'une ou plusieurs image(s) par le biais de requêtes. À chaque requête de l'application est associé un objet `CaptureRequest` qui sera mis en queue et envoyé à la caméra. Les requêtes sont mises en queue et sont toutes opérées dans l'ordre d'appel. Si l'application souhaite capturer une vidéo à 30 images

par seconde, c'est à elle d'envoyer 30 `CaptureRequest` successives à la caméra via la HAL. Notez qu'avec Kit Kat, l'ajout de la fonction `flush()` dans la HAL permet de vider complètement la file de traitement.

À chaque `CaptureRequest` en entrée se voient associés un objet `CaptureResult` (paquet contenant les métadonnées et qui retourne à l'application via un `CaptureListener` positionné au préalable), ainsi que différents buffers d'images de sortie. Ces derniers sont propres à la demande de capture qui aura été faite. C'est en effet à l'application d'associer une `TargetSurface` (surface cible) à la `CaptureRequest` et de configurer le format de sortie via l'appel à la fonction `configureOutputs()` du `CameraDevice`.

Ces formats de destination peuvent être de plusieurs types : `SurfaceView`, `SurfaceTexture`, `MediaCodec`, `MediaRecorder`, `Renderscript Allocation` ou encore `ImageReader` (depuis Kit Kat uniquement). Vous trouverez la raison d'être de chacun au sein de la documentation du SDK **[4]**. Le but étant, dans tous les cas, que ces buffers soient préparés par l'application (via des appels à la HAL Gralloc pour allouer des buffers d'un certain type) et transférés (en termes de pointeur) à la HAL (et donc à la caméra), pour une écriture directe. Ceci évite toute recopie de buffer (approche « zero-copy ») pour passer d'un niveau à l'autre, minimisant d'autant la bande passante mémoire nécessaire et la consommation CPU. Dit autrement, votre SoC ne pourrait pas traiter des flux vidéo 1080p, voire 4K à 30 ou 60 images par seconde, s'il devait perdre ses ressources matérielles à recopier inutilement des buffers.

Le format de destination inclut également la résolution de sortie. Ainsi, plutôt que de recevoir une image en haute définition du capteur et devoir la convertir ensuite, c'est directement l'ISP qui réalisera l'opération en hardware. De même, dans le cas d'un enregistrement vidéo, l'image en provenance du capteur et après traitement par l'ISP sera directement transmise (sans recopie) au VPU (*Video Processing Unit*) pour encodage matériellement assisté de la surface, sans passage par le CPU et la mémoire centrale.

À chaque `CaptureRequest` émanant de la queue la caméra associe donc une image en retour, qui sera stockée dans ce buffer. De la même façon, les métadonnées seront associées au sein d'un `CaptureResult`. Ces dernières sont

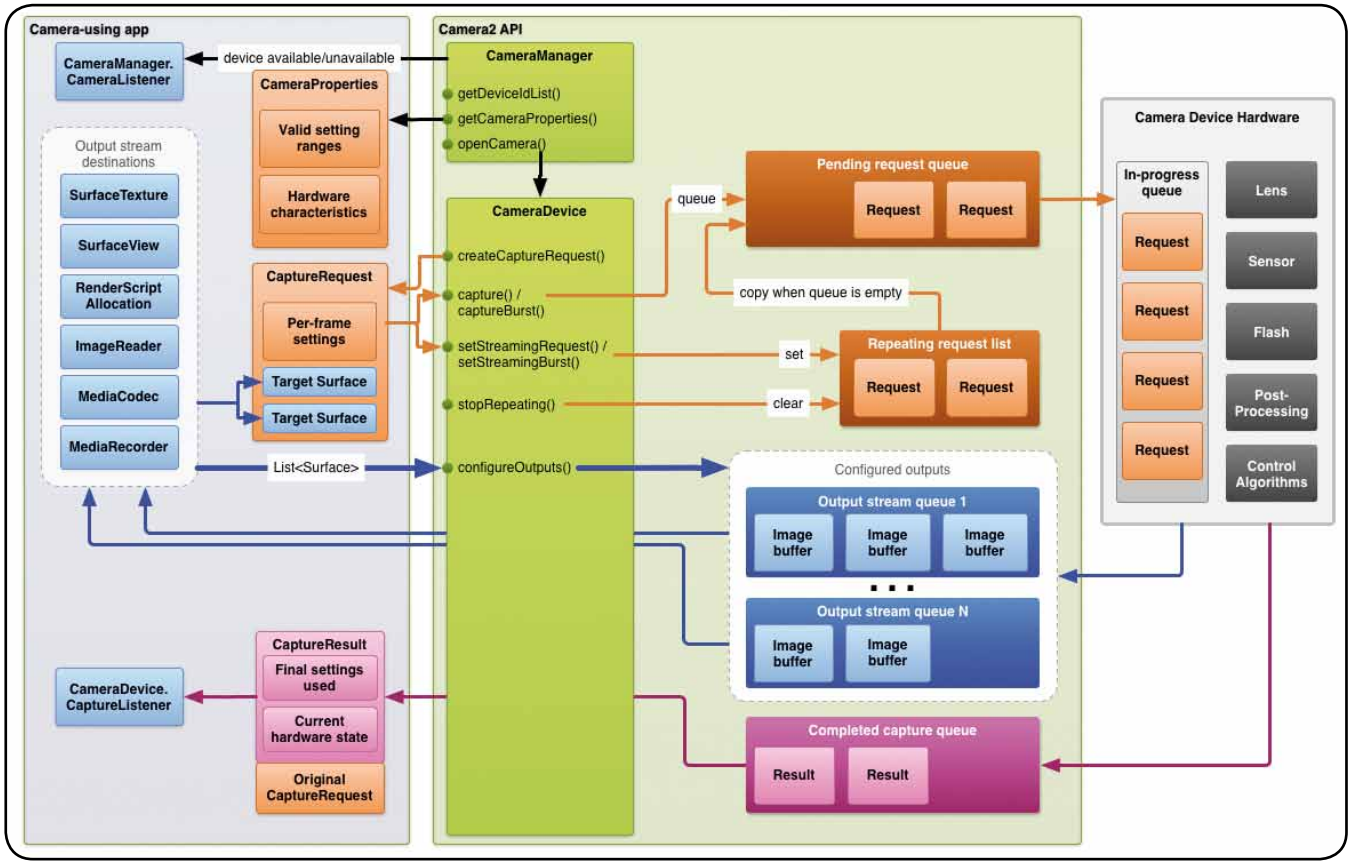


Fig. 3 : Séquencement opérationnel de capture d'image par la caméra

variées et contiennent des informations telles que la résolution et le format des pixels, la configuration du capteur, de la lentille, ou encore du flash, les opérations 3A réalisées, le procédé de conversion RAW vers YUV utilisé, ou encore les statistiques générées.

À noter qu'une `CaptureRequest` peut être explicite et unique (e.g. prise d'une photo) via un appel à la fonction `capture()` du `CameraDevice`, ou multiple (e.g. enregistrement vidéo ou pré-visualisation vidéo temps-réel) via un appel à la fonction `setRepeatingRequest()` du `CameraDevice`. Bien entendu, un appel à `capture()` sera prioritaire sur les appels à `setRepeatingRequest()`, garantissant la prise en compte du déclenchement photographique.

Pour résumer, au niveau de la HAL, il faut bien comprendre que les requêtes arrivent de manière asynchrones depuis le framework applicatif et que ces dernières sont placées dans une FIFO et devront donc toutes êtres traitées de manière séquentielle. Si plusieurs buffers de sortie sont déclarés, les horodatages associés à ces derniers devront tous être identiques pour permettre ensuite au framework de les traiter avec cohérence temporelle.

Conclusion

Voici donc qui conclut notre aperçu des facultés de capture vidéo et photo d'Android et de l'évolution fonctionnelle qui se profile dans nos téléphones. Les constructeurs ne vont pas tarder à rendre leur matériel compatible avec la nouvelle HAL v3 et l'API Camera v2 d'Android pointant le bout de son nez, j'ai grande hâte de découvrir comment les développeurs applicatifs vont pouvoir utiliser leurs nouveaux jouets. À bientôt pour découvrir davantage encore les entrailles du premier système mobile mondial ! ■

Références

[1] https://android.googlesource.com/platform/frameworks/base/+/kitkat-release/core/java/android/hardware/camera2/

[2] https://github.com/lbk003/Cam3HiddenAPIs

[3] https://android.googlesource.com/platform/hardware/qcom/camera/+/kitkat-release/QCamera2/HAL3/

[4] http://developer.android.com/reference/packages.html

IL Y A TOUJOURS DES CHOSES À APPRENDRE AVEC VIM

par ***Tristan Colombo***

Depuis quelques années déjà, je vous propose périodiquement des articles sur la façon de configurer Vim, de créer des macros, des fichiers de coloration syntaxique, etc. Voici revenu le temps de notre rendez-vous annuel pour mettre à jour notre configuration et éventuellement apprendre de nouvelles commandes : il y a toujours des choses à apprendre avec Vim.

Nous allons considérer que nous partons d'une configuration vierge et la nouvelle configuration sera axée sur mes préoccupations du moment : Python, Web (HTML, CSS et JavaScript), et C. Les touches de raccourcis seront notées **<touche>** et les combinaisons seront signalées par des signes **+**.

1 Les extensions

1.1 Vundle : l'indispensable

Vundle est un gestionnaire d'extensions pour Vim : il permet leur installation et leur mise à jour. Pour l'installer, il faut commencer par récupérer le code de Vundle :

```
# git clone https://github.com/gmarik/
vundle.git ~/.vim/bundle/vundle
```

Il faut ensuite indiquer dans votre fichier **.vimrc** quelles sont les extensions que vous allez utiliser. Pour commencer, nous allons bien sûr référencer Vundle lui-même de manière à ce qu'il puisse être mis à jour. Voici le contenu de notre fichier, basé sur la documentation de l'extension Vundle **[1]** :

```
01: " Vimrc configuration file
02: " Version : 2014-01
03:
04: " Vundle configuration
05: "-------------------------------------
06: set nocompatible
07: filetype off
08:
09: set rtp+=~/.vim/bundle/vundle/
10: call vundle#rc()
11:
12: " Vundle management (for updates)
13: Bundle 'gmarik/vundle'
14:
15: " List of bundles (or extensions)
16: "-------------------------------------
17:
18: filetype plugin indent on
```

À partir de la ligne 17, nous insérerons la liste des extensions que nous souhaitons utiliser.

Note

J'ai pris l'habitude de ne plus utiliser directement de fichier **.vimrc** mais de créer un lien vers un fichier **.vim/vimrc**, ce qui me permet de « transporter » simplement et rapidement toute ma configuration.

Désormais, vous aurez accès principalement aux commandes suivantes :

- **:BundleList** : liste les *bundles* (extensions) installés ;
- **:BundleInstall!** : installe ou met à jour les bundles listés dans le fichier de configuration.

Pour l'instant nous n'avons installé que Vundle, donc ces commandes ne nous servent pas à grand-chose (et même la mise à jour est inutile,

puisque nous venons à peine de cloner le dépôt de Vundle). Par contre, il faut savoir que vous pouvez lancer les installations et les mises à jour directement depuis la console en tapant :

```
# vim +BundleInstall +qall
```

On pourrait donc imaginer une tâche planifiée (cron) ou un script permettant de simplifier et d'automatiser les mises à jour.

1.2 Airline, pour une barre de statut améliorée

Vim-powerline permet d'afficher une barre de statut en couleurs contenant des symboles et beaucoup plus agréable à utiliser que la barre standard. Cette extension est un peu lourde, dépend de Python et n'est plus maintenue... Mais une nouvelle extension, plus légère, a vu le jour : **vim-airline** (l'auteur a écrit la première version de cette extension dans un avion, d'où le nom). Petit plus de cette version : elle supporte de nombreuses extensions permettant d'accroître encore son efficacité **[2]**.

1.2.1 Installation

Ajoutez la ligne suivante à votre fichier **vimrc** et lancez l'installation depuis vim à l'aide de **:BundleInstall** :

```
17: Bundle 'bling/vim-airline'
```

Relancez vim et... vous ne verrez aucune différence avec la barre de statut précédente. C'est normal ! Nous n'avons pas fini la configuration. À la fin du **vimrc**, ajoutez les lignes suivantes pour forcer l'affichage de Powerline même si vous n'avez qu'une seule fenêtre (pas de *hsplit* ou de *vsplit*) :

```
21: " Airline setup
22: "--------------------------------------------------------------------------
23: set laststatus=2
```

En relançant à nouveau vim, vous obtiendrez un affichage plus informatif mais guère esthétique, comme le montre la figure 1.

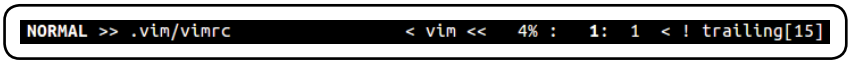


Fig. 1 : Barre de statut Airline utilisant les caractères « spéciaux » par défaut.

Pour activer l'affichage des véritables caractères spéciaux (diverses flèches, etc.), il faut ajouter une ligne de configuration.

```
24: let g:airline_powerline_fonts=1
```

Mais vous vous apercevrez que le résultat n'est pas encore probant (voir figure 2).

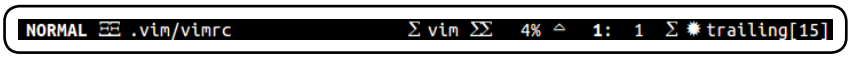


Fig. 2 : Barre de statut Airline utilisant une police de caractères non compatible.

Pour que l'affichage soit correct, il faut une police de caractères contenant les caractères spéciaux utilisés par Airline (les flèches par exemple). Il faut donc installer une telle police, ce qui se fait assez simplement sous Linux :

1. Télécharger la police PowerlineSymbols :

```
# wget https://github.com/Lokaltog/powerline/raw/develop/font/PowerlineSymbols.otf
# mv PowerlineSymbols.otf ~/.fonts
```

2. Télécharger le fichier de configuration pour **fontconfig** :

```
# wget https://github.com/Lokaltog/powerline/raw/develop/font/10-powerline-symbols.conf
# mv 10-powerline-symbols.conf ~/.fonts.conf.d
```

3. Exécuter la commande suivante :

```
# fc-cache -vf ~/.fonts
```

Si les répertoires **~/.fonts** ou **~/.fonts.conf.d** n'existent pas, il faudra bien sûr les créer pour pouvoir y placer les fichiers des étapes 1 et 2. En relançant vim, vous pourrez voir que les caractères étranges ont été remplacés par quelque chose de beaucoup plus lisible (voir figure 3).

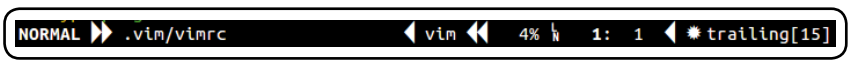


Fig. 3 : Barre de statut Airline utilisant une police de caractères compatible.

C'est mieux, mais ce n'est pas encore la barre annoncée : il nous manque les couleurs ! Pour les activer, il faut ajouter une nouvelle ligne au fichier de configuration de vim :

```
25: set t_Co=256
```

Fig. 4 : Barre de statut Airline utilisant une police de caractères compatible et un affichage en couleurs.

Fig. 5 : Barre d'onglets Airline en ayant activé l'extension « Smarter tab line »

La figure 4 illustre le résultat obtenu. Si le thème par défaut ne vous convient pas, sachez qu'une dizaine de thèmes différents sont disponibles **[3]** et qu'il suffit d'indiquer le nom du thème choisi dans une variable de configuration :

```
26: let g:airline_theme='murmur'
```

Si vous souhaitez seulement connaître le nom du thème actuellement utilisé sur votre machine, ou en changer seulement pour la session en cours, vous pourrez utiliser la commande **:AirlineTheme** (sans paramètre celle-ci affiche le thème utilisé et suivie du nom d'un thème, elle modifie le thème actif).

Il est possible de redéfinir le contenu de la barre de statut en utilisant la syntaxe courante de vim (voir les variables de la famille **airline_section_**).

1.2.2 Les extensions

Je l'ai dit en préambule, Airline supporte de nombreuses extensions. Pour la plupart d'entre elles, vous n'aurez rien à faire pour les activer : elles dépendent d'une extension Vim et si celle-ci est présente, alors l'extension Airline se déclenchera automatiquement.

Il existe une exception pour la configuration de l'extension « Smarter tab line » permettant d'avoir le même type d'affichage que celui de la barre de statut pour les onglets. Une ligne devra être ajoutée pour l'activer :

```
27: let g:airline#extensions#tabline#enabled = 1
```

1.3 VimWiki, parce que personne ne se souvient de toutes les commandes

Vim, et c'est toute sa force, dispose d'un nombre pléthorique de commandes. On peut découvrir pratiquement tous les jours la commande ultime qui nous aurait fait gagner beaucoup de temps si on l'avait connue... Mais ces commandes sont utiles dans des contextes particuliers et dès que l'on en sort, on les oublie. Comment faire pour retrouver rapidement l'information en cas de besoin ? Plusieurs solutions existent. Jusqu'à présent, j'utilisais un simple fichier texte affiché dans vim à l'aide d'un raccourci clavier. On peut pousser le système un peu plus loin en bâtissant un fichier d'aide vim, dans lequel des liens permettront de parcourir rapidement le document. Enfin, l'aboutissement ultime sera une sorte de wiki intégré à vim. Pour ce dernier, une extension existe, il s'agit de VimWiki, que l'on installe comme les autres extensions (n'oubliez pas d'exécuter **:BundleInstall**) :

```
18: Bundle "vimwiki/vimwiki"
```

L'accès au Wiki se fait par **<\> + <w> + <w>**. Lors du premier lancement, vous devrez confirmer la création du répertoire qui contiendra le wiki :

```
Vimwiki: Make new directory: /home/tristan/vimwiki
 [Y]es/[n]o?
```

Par défaut, on nous propose de créer le répertoire dans notre répertoire utilisateur... Je préférerais que ce soit dans **~/.vim** de manière à pouvoir manipuler simplement tous les fichiers de configuration relatifs à vim. Nous répondrons donc « n » à la question et nous configurerons le chemin vers notre répertoire dans le **.vimrc** :

```
32: " Vimwiki setup

33: "-------------------------------------------------------------
34: let g:vimwiki_list=[{'path':'~/.vim/vimwiki'}]
```

Il faut ensuite créer le répertoire où seront hébergés les fichiers du wiki :

```
$ mkdir ~/.vim/vimwiki
```

Pour tester si la modification est effective, dans vim, en mode commande, tapez **<\> + <w> + <w>**. Un nouveau fichier **~/.vim/vimwiki/index.wiki** doit s'ouvrir : il s'agit

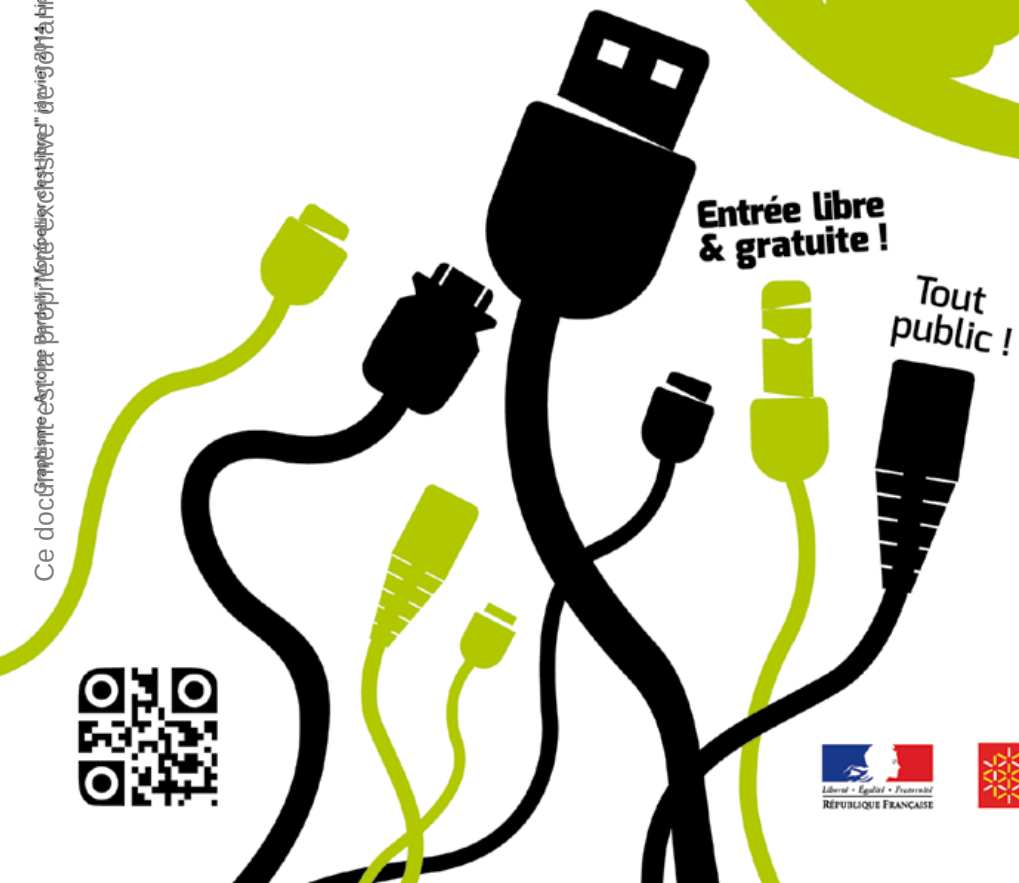
rmll.info

Montpellier

Du 5 au 11 juillet 2014

Esplanade Charles-de-Gaulle
et Université Montpellier 2 (UM2)

15èmes Rencontres Mondiales du Logiciel Libre



Entrée libre & gratuite !

Tout public !

RMLL
MONTPELLIER 2014
Le Libre et vous

Fig. 6 : Barre de statut Powerline après ajout de l'extension Fugitive.

de la page d'entrée de votre wiki. Il vous suffit d'éditer ce fichier dans le style wiki (encadrer par des étoiles pour écrire en gras, entre **[[** et **]]** pour déclarer un lien, etc.). Pour accéder à la page ciblée par un lien, placez-vous sur le lien et appuyez simplement sur **<Return>** en mode commande (si la page n'existe pas, on vous proposera de la créer). Pour revenir à la page initiale, il faudra appuyer sur **<Backspace>**. Voici quelques lignes d'exemple d'un fichier d'index :

```
01: = Vim memo =
02:
03:         * [[Copier / Coller / Déplacer]]
04:         * [[Sélectionner]]
05:         * [[Gestion des buffers]]
06:         * [[Macros]]
```

En organisant correctement votre fichier, vous n'aurez plus d'excuse pour ne pas retrouver LA commande dont vous aviez besoin et s'il s'agit d'une nouvelle commande, le temps passé en recherches ne sera pas perdu puisque vous pourrez mettre à jour rapidement votre wiki !

1.4 Interfacer Vim avec Git

L'extension qui gère le mieux Git est Fugitive (pensez toujours au **:BundleInstall**) :

```
19: Bundle 'tpope/vim-fugitive'
```

Les commandes Git peuvent être appelées directement depuis Vim grâce à des commandes qui commencent toutes par la lettre **G** :

- **:Gwrite** : équivalent à **git add fichier_courant**. Le fichier courant est stocké dans l'index.
- **:Gread** : équivalent à **git checkout fichier_courant**. Recharge le fichier courant tel qu'il était lors du dernier commit.
- **:Gremove** : équivalent à **git rm fichier_courant**. Supprime le fichier courant (buffer et index).
- **:Gmove** : équivalent à **git mv fichier_courant**. Renomme le fichier courant.
- **:Gcommit** permet de transférer (« commiter ») le code après avoir saisi un commentaire.

Notez qu'il n'y a rien à faire pour mettre à jour la barre de statut de Powerline. Ouvrez un fichier d'un dépôt Git et vous verrez apparaître l'information dans la barre de statut (voir figure 6).

1.5 Naviguer dans l'arborescence de fichiers avec NerdTree

L'installation se fait via Vundle :

```
20: Bundle 'scrooloose/nerdtree'
```

Cette fenêtre offre une commande permettant d'ouvrir une fenêtre de navigation parmi les fichiers. Taper la commande entière n'étant pas pratique, nous allons ajouter un raccourci clavier. Pour une meilleure lisibilité, j'ai pris l'habitude de séparer le fichier contenant mes raccourcis clavier du reste de la configuration de Vim. Il faudra lire ce fichier **.vim/shortkeys.vim** depuis le fichier **.vimrc** :

```
36: " Shortkeys loading
37: "------------------------------------------------------------
38: execute 'source ' . $HOME . '/.vim/shortkeys.vim'
```

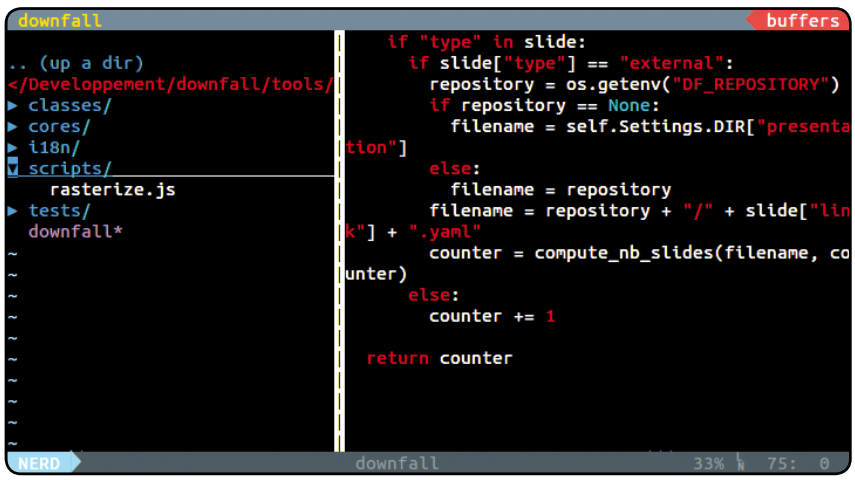


Fig. 7 : Navigation dans l'arborescence des fichiers avec NerdTree.

L'OPEN SOURCE AU SERVICE DES COLLECTIVITÉS AU **COTER CLUB**

17 - 18 juin

au Palais des Congrès de

Caen

avec SoLibre

Objectif libre
Infrastructure & Formations Linux

ametys
Web CMS - Intranet

BlueMind
Messagerie & espaces collaboratifs

Avencall
Communiquez librement

Merethis
Centreon editor

MAKINA CORPUS
Développons de nouveaux horizons

LINUX MAGAZINE / FRANCE
Administration et développement sur systèmes open source et embarqués

SUDOKEYS
libérez votre système d'information

CoTer Club http://www.coter-club.org

SoLibre est membre du CNLL

La création du lien dans le fichier **.vim/shortkeys.vim** se fait par :

```
01: " NerdTree activation
02: "------------------------------------------------------------
03: map <F2> <Esc>:NERDTreeToggle<CR>
```

Vous n'avez plus qu'à appuyer sur la touche **<F2>** pour faire apparaître et disparaître la fenêtre de navigation présentée en figure 7.

1.6 Mise en place de l'auto-complétion

Nous allons avoir besoin de l'extension YouCompleteMe et de paquetages spécifiques. L'auto-complétion sera proposée de manière automatique, sans avoir à appuyer sur une quelconque combinaison de touches. Elle fonctionnera pour du C/C++, du Python, du PHP, etc.

L'installation de cette extension est un peu plus complexe que pour les autres... Il faut d'abord que vous vous assuriez que votre version de Vim est supérieure à 7.3.584 :

```
:version
VIM - Vi IMproved 7.3 (2010 Aug 15, compiled May  4 2012 04:22:36)
Rustines incluses : 1-429
```

Ici, nous sommes malheureusement en version 7.3.429. Il va falloir compiler Vim depuis les sources. Commençons par nous assurer que nous disposons de tous les outils nécessaires :

```
$ sudo aptitude install libncurses5-dev libgnome2-dev libgnomeui-dev \
libgtk2.0-dev libatk1.0-dev libbonoboui2-dev \
libcairo2-dev libx11-dev libxpm-dev libxt-dev python-dev ruby-dev mercurial \
build-essential cmake
```

Il faut ensuite supprimer tous les paquets relatifs à Vim (pas de panique, le système ne restera pas très longtemps sans votre éditeur favori) :

```
$ sudo aptitude remove vim vim-runtime gvim vim-common \
vim-gnome vim-gui-common vim-tiny
```

Vous n'aurez pas forcément tous ces paquets installés, mais au moins la liste devrait être complète. Sous Ubuntu, le système vous proposera de supprimer le méta-paquet **ubuntu-minimal** : faites-le sans crainte, vous n'effacerez rien compromettant votre système.

L'étape suivante consiste à récupérer les sources :

```
$ hg clone https://code.google.com/p/vim
```

Vous pouvez faire une pause de deux ou trois minutes le temps de tout télécharger. Il faut ensuite compiler le code en s'assurant de spécifier le bon chemin pour **python2.7-config** :

```
$ cd vim
$ which python2.7-config
/usr/bin/python2.7-config
$ ./configure --with-features=huge \
--enable-rubyinterp \
--enable-pythoninterp \
--with-python-config-dir=/usr/bin/python2.7-config \
--enable-perlinterp \
--enable-gui=gtk2 --enable-cscope -prefix=/usr
$ make VIMRUNTIMEDIR=/usr/share/vim/vim74
$ sudo make install
$ cd ..
$ rm -R vim
```

Pour finir, il faut définir Vim en tant qu'éditeur par défaut :

```
$ sudo update-alternatives --install /usr/bin/editor editor /usr/bin/vim 1
$ sudo update-alternatives --set editor /usr/bin/vim
$ sudo update-alternatives --install /usr/bin/vi vi /usr/bin/vim 1
$ sudo update-alternatives --set vi /usr/bin/vim
```

Effectuez à nouveau le test de version dans Vim pour vous assurer d'une part que Vim fonctionne et que d'autre part, vous êtes bien en version 7.4.

Il faut passer maintenant à l'installation de l'extension YouCompleteMe proprement dite (toujours par Vundle) :

```
21: Bundle 'Valloric/YouCompleteMe'
```

L'installation est un peu longue et vous obtiendrez un message d'avertissement à la fin de celle-ci :

```
ycm_client_support.[so|pyd|dll] and ycm_core.[so|pyd|dll] not detected;
you need to compile YCM before using it. Read the docs!
```

Nous allons donc suivre la recommandation et procéder à la compilation de YCM (YouCompleteMe) et de la dernière version de **clang**, qui est fortement conseillée dans la documentation. Pour commencer, il faut ajouter les dépôts relatifs à votre distribution (liste disponible sur http://llvm.org/apt/). Il faut donc ouvrir le fichier **/etc/apt/sources.list** et y insérer (exemple pour Ubuntu 12.04 LTS) :

```
deb http://llvm.org/apt/precise/ llvm-toolchain-precise main
deb-src http://llvm.org/apt/precise/ llvm-toolchain-precise main
deb http://ppa.launchpad.net/ubuntu-toolchain-r/test/ubuntu precise main
```

L'exemple de cette distribution est intéressant, car il requiert en plus le dépôt PPA pour obtenir la dernière version de **gcc**. N'oubliez pas de récupérer les clés :

```
$ wget -O - http://llvm.org/apt/llvm-snapshot.gpg.key | sudo apt-key add -
$ sudo add-apt-repository ppa:ubuntu-toolchain-r/ppa
```

Et procédez enfin à l'installation :

```
$ aptitude update
$ aptitude install clang-3.5 lldb-3.5 libclang-3.5-dev lldb-3.5-dev
$ cd ~/.vim/bundle/YouCompleteMe
$ ./install.sh --clang-completer
```

Je dois reconnaître que l'installation n'a pas été très aisée en suivant la documentation (pourtant très fournie) de l'extension. J'espère que les indications données ici vous éviteront de perdre trop de temps en installation (mais ce ne sera pas du temps perdu, vous ne le regretterez pas !).

Si vous testez les capacités d'auto-complétion immédiatement, vous risquez d'être déçus... Sur du C++ ça marche plutôt bien, par contre sur du Python, le lien avec l'extension Jedi qui est censée proposer les choix de complétion n'a pas l'air de trop fonctionner. Ce n'est pas trop grave, pour Python il existe une extension couteau suisse qui marche très bien...

1.7 Python-mode : le couteau suisse du développeur Python

Python-mode contient tout un tas d'outils permettant de prendre en charge les environnements virtuels, de vérifier les conventions PEP8, d'ajouter et supprimer simplement des points d'arrêt, de vérifier la syntaxe à l'aide de pylint et pyflakes, de plier/déplier les docstrings, de proposer une complétion de code (qui fonctionnera donc avec YouCompleteMe), etc. L'installation n'en sera pas plus compliquée que pour les autres modules :

```
22: Bundle 'klen/python-mode'
```

Sans aucune modification, vous aurez accès à une complétion plus intéressante, comme le montre la figure 8.

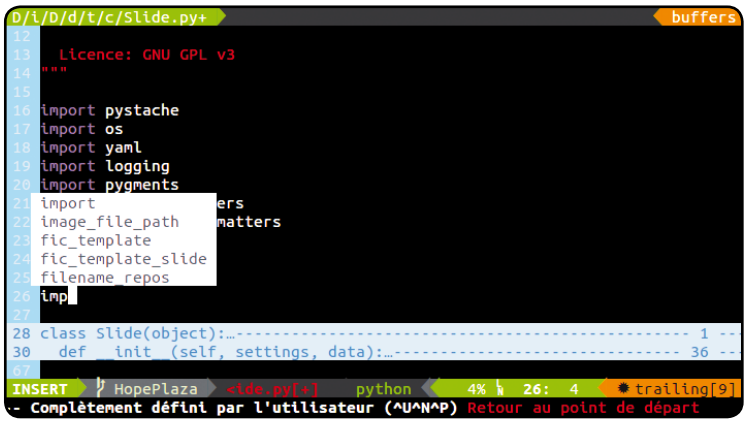


Fig. 8 : Auto-complétion dans un fichier Python.

Voici quelques fonctionnalités en vrac (pour plus de détails, tapez **:help pymode**) :

- Pour afficher la documentation d'une commande, il faut se placer sur l'instruction et taper sur **<K>** (en majuscule) en mode commande.
- Pour ajouter / retirer des points d'arrêt : **<\>** puis **<b>**. Une ligne **import ipdb; ipdb.set_trace()** apparaîtra ou disparaîtra.
- **<\>** puis **<r>** pour exécuter le code à l'aide de l'interpréteur de votre environnement virtuel.
- **<Ctrl> + <c>** puis **<g>** : retrouve la définition de l'élément sous le curseur.
- **<espace>** sur une docstring pour la déplier.

La vérification de la syntaxe du code se fait de manière automatique à la sauvegarde.

1.8 Surround

Installation par la méthode habituelle (et toujours **:BundleInstall**) :

```
23: Bundle 'tpope/vim-surround'
```

S'il vous arrive de remplacer des termes encadrant des mots comme **"ma phrase"** par **'ma phrase'** ou de manipuler du code HTML et de changer des noms de tags, cette extension vous rendra de grands services. Positionnez votre curseur entre les caractères à remplacer, puis tapez la commande **cs<ancien><nouveau>**. Par exemple, sur **"ma phrase"**, **cs" ** produira ** ma phrase **.

Pour encadrer plusieurs lignes, passez en mode visuel, sélectionnez vos lignes, puis tapez **S<encadrement>**.

La commande **ds<encadrement>** permet de supprimer un encadrement.

Lorsque vous utilisez des tags HTML comme encadrement, ceux-ci sont automatiquement fermés et vous pouvez même utiliser des attributs... Vous n'avez - presque - plus rien à faire !

1.9 Aligner simplement des colonnes avec Align

Installation :

```
24: Bundle 'vim-scripts/Align'
```

Certains développeurs aiment que leur code soit aligné et indenté sur certains caractères (comme les signes **=** dans les affectations). L'extension Align permet de réaliser cette indentation, en indiquant un caractère sur lequel aligner d'autres lignes. Par exemple, considérons le code suivant :

```
a = 1
maVariable = 'GNU/Linux magazine'
autreVariable = 2.543
```

Si vous souhaitez aligner ces lignes sur le caractère **=**, il suffit de passer en mode visuel, de sélectionner ces lignes, puis de taper la commande :

```
:'<,'>Align =
```

Vous obtiendrez alors un alignement parfait :

```
a             = 1
maVariable    = 'GNU/Linux magazine'
autreVariable = 2.543
```

1.10 Zen-coding : restez zen en codant du HTML

Zen-coding... ne s'appelle plus zen-coding, mais emmet-vim. Il s'agit d'une extension qui vous permet d'écrire de manière très rapide du code HTML en vous servant d'abréviations. L'installation utilise toujours Vundle :

```
25: Bundle 'mattn/emmet-vim'
```

Vous pourrez alors utiliser les différentes abréviations pour accélérer l'écriture de vos fichiers : tapez l'abréviation, puis appuyez sur **<Ctrl>** + **<y> + <,>**. Voici deux exemples d'utilisation :

1. Nous commençons un fichier HTML et nous souhaitons avoir un squelette de fichier. Il suffit de taper :

```
html:5
```

Appuyez ensuite sur la combinaison magique **<Ctrl> + <y> +<,>**, vous obtiendrez :

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title></title>
</head>
<body>

</body>
</html>
```

2. Vous voulez créer une liste où chaque item aura un identifiant numéroté et une classe de style spécifique ? C'est encore très simple en utilisant la notation JQuery :

```
ul>li.mon_style#mon_id$*5
```

Un petit passage par emmet-vim et on obtient :

```
<ul>
   <li id="mon_id1" class="mon_style"></li>
   <li id="mon_id2" class="mon_style"></li>
   <li id="mon_id3" class="mon_style"></li>
   <li id="mon_id4" class="mon_style"></li>
   <li id="mon_id5" class="mon_style"></li>
</ul>
```

Pour plus d'exemples, je vous invite à consulter la page https://raw.githubusercontent.com/mattn/emmet-vim/master/TUTORIAL.

2 Du Vim « pur » pour la configuration

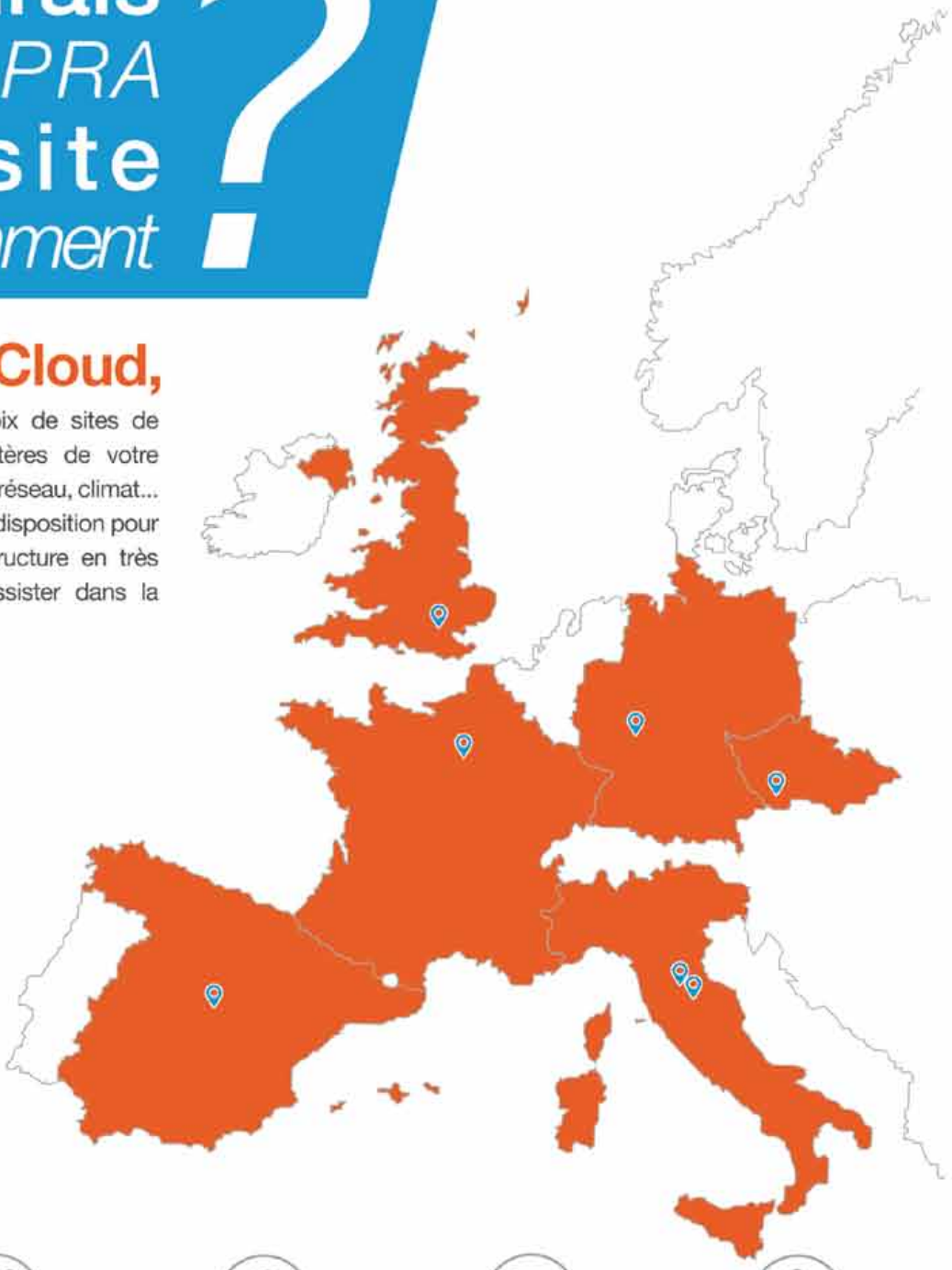
Les extensions sont vraiment très pratiques... Mais il ne faut pas oublier que l'on peut faire déjà beaucoup de choses seulement avec Vim ! Pour commencer, il est évident qu'il faut activer la coloration syntaxique à chaque lancement. L'affichage des numéros de ligne dans la marge pourrait également être très intéressant :

```
45: " General
46: "-------------------------------------------------------
47: syntax on
48:
49: " Affichage des numéros de ligne
50: set number
51: highlight LineNr ctermbg=blue ctermfg=gray
```

Je voudrais *bâtir un PRA* **multi-site** *Je fais comment* ?

Avec Aruba Cloud,

vous disposez d'un large choix de sites de secours, en fonction des critères de votre stratégie de sécurité: proximité, réseau, climat... Nos équipes sont aussi à votre disposition pour vous aider à bâtir une infrastructure en très haute disponibilité et vous assister dans la définition de votre stratégie.



- 3 hyperviseurs
- 6 datacenters en Europe
- APIs et connecteurs
- 70+ templates
- Contrôle des coûts

Nous avons choisi Aruba Cloud car nous bénéficions d'un haut niveau de performance, à des coûts contrôlés et surtout car ils sont à dimension humaine, comme nous. Xavier Dufour - Directeur R&D - ITMP

Contactez-nous! 0810 710 300 www.arubacloud.fr

aruba CLOUD

MY COUNTRY. MY CLOUD.*

Cloud Public | Cloud Privé | Cloud Hybride | Cloud Storage | Infogérance

On peut aussi afficher une bande rouge indiquant le nombre maximum de caractères à ne pas dépasser par ligne. Si vous souhaitez respecter la règle des 80 caractères par ligne, il est toujours mieux d'appréhender visuellement la longueur des lignes que l'on tape :

```
53: " Longueur maximale des lignes
54: set textwidth=80
55: " Surligne la colonne du dernier caractère autorisé par textwidth
56: set cc=+1
```

Autre fonction très utile, l'affichage des caractères invisibles. Ne vous est-il jamais arrivé de coder en Python ou en LaTeX et de ne pas comprendre d'où venait une erreur... alors qu'il s'agissait d'un espace insécable ? Vim est capable d'afficher des symboles correspondant à chaque caractère invisible si on les définit auparavant :

```
58: " Définition de l'affichage des caractères invisibles avec 'set list'
59: set listchars=nbsp:¤,tab:>-,trail:¤,extends:>,precedes:<,eol:¶,trail:·
```

Désormais, en tapant **`:set list`** en mode commande vous ferez apparaître les caractères invisibles (et **`:set nolist`** pour les faire disparaître). Si vous utilisez fréquemment cette fonctionnalité, il pourra être intéressant de créer un raccourci.

3 Quelques rappels

Je vous propose maintenant quelques rappels « en vrac », des commandes Vim pratiques que l'on n'utilise pas forcément tous les jours et qui trouveraient leur place dans notre VimWiki.

3.1 Opérations sur un texte encadré

Supposons que vous ayez la ligne suivante :

```
<div class="ma_classe"></div>
```

Vous pouvez effectuer simplement des opérations sur du texte encadré par un caractère particulier (l'inverse de l'extension Surround) :

- Changer le texte entre guillemets : positionnez le curseur entre les guillemets, passez en mode commande et tapez **`ci"`**. Cette opération supprimera le texte et passera en mode insertion. Si vous souhaitez seulement supprimer le texte, tapez **`di"`**.
- Sélectionner le texte entre guillemets : positionnez le curseur entre les guillemets, passez en mode commande et tapez **`vi"`**.

3.2. Déplacer des lignes

Pour déplacer la ligne numéro **`n1`** en position **`n2`** :

- Positionnez-vous sur la ligne **`n1`** et tapez **`:mn1`**,
- ou alors tapez **`:n1mn2`**.

La commande **`t`** réalise la même opération en conservant la ligne originale (duplication).

3.3 Les registres

Lorsque vous réalisez des copies de lignes, n'oubliez pas les registres ! Ceux-ci représentent 26 zones de mémoires de **`a`** à **`z`** dans lesquelles stocker des données pour les appeler par la suite. Pour copier une ligne dans le registre **`a`**, placez-vous sur la ligne souhaitée et, en mode commande, tapez : **`"ayy`** (**`"a`** pour choisir le registre **`a`** et **`yy`** pour copier la ligne). Lorsque vous voudrez accéder à cette valeur, il vous suffira de refaire appel à ce registre et de coller la valeur qu'il contient (par **`"ap`**).

Attention à quelques registres particuliers :

- **`"_`** est la poubelle : on ne peut rien récupérer de ce registre ;
- **`"%`** est le nom du fichier courant ;
- **`".`** correspond à votre dernière insertion de texte (en mode insertion) ;
- **`":`** est la dernière commande que vous ayez exécutée ;
- **`"/`** est le dernier motif de recherche employé.

4 Et si je n'ai pas le temps de tout configurer ?

Tout ce que l'on vient de voir est bien alléchant... Mais l'on ne dispose pas toujours du temps nécessaire pour réaliser entièrement sa configuration rêvée de Vim. Le projet spf13-vim **[4]** est là pour vous. Il s'installe en une seule ligne de commandes en sauvegardant votre ancienne configuration de Vim et il contient déjà nombre d'extensions utiles. Cerise sur le gâteau : vous pourrez bien sûr par la suite ajouter votre propre configuration pour l'améliorer encore !

Pour l'installation, ouvrez un terminal et tapez simplement :

```
$ curl http://j.mp/spf13-vim3 -L -o - | sh
```

Le processus est un peu long, car il faut aller chercher toutes les extensions. Bien sûr, comme pour toute solution toute prête, il y a des extensions qui ne vous serviront jamais... Vous pourrez faire le ménage par la suite.

Parmi les extensions installées se trouve Solarized, qui améliore l'affichage du code (choix des polices et des couleurs). Si le choix par défaut ne vous convient pas, vous pouvez le changer par **:colorscheme <Tab>** qui vous affichera la liste des choix possibles. Pour ma part, je préfère le thème **ir_black**. Mais une fois ce choix fait, lorsque vous fermerez Vim il sera perdu... Pour l'enregistrer, créez un fichier **~/.vimrc.local**. C'est dans ce fichier que vous pourrez ajouter vos propres lignes de configuration (par exemple, des éléments que nous aurons pu voir précédemment). Pour le thème **ir_black**, notre fichier **~/.vimrc_local** contiendra :

```
01: colorscheme ir_black
```

Et en relançant Vim... ça ne marchera pas ! Il s'agit du bug #384 qui n'est pas résolu depuis fort longtemps... Donc, on ne va pas attendre une éventuelle mise à jour et on va trouver une petite astuce avec une commande automatique qui se lance à chaque ouverture de fichier :

```
01: autocmd FileType * colorscheme ir_black
```

Et voilà, ça marche ! À vous maintenant de découvrir toutes les possibilités offertes par les extensions installées, supprimer celles qui ne vous servent pas, ajouter vos propres commandes de configuration dans le fichier **~/.vimrc.local** et vos extensions dans le fichier **~/.vimrc.bundles.local**. Par exemple, pour ajouter VimWiki, il faudra insérer dans **~/vimrc.bundles.local** la ligne :

```
01: Bundle "vimwiki/vimwiki"
```

Vous devrez ensuite suivre la procédure décrite précédemment. Vous pouvez faire de même avec les extensions présentées en début d'article mais absentes de spf13-vim (pensez à exécuter **:BundleInstall**...)

Conclusion

Que vous souhaitiez partir d'un configuration vierge ou vous aider de la configuration du projet spf13-vim, il y a toujours moyen avec Vim de définir précisément ce que vous attendez de votre éditeur de code. Vous avez déjà « perdu » beaucoup de temps à apprendre à utiliser Vim... Vous pourrez continuer en le configurant, mais il faut reconnaître que l'on ne regrette jamais le temps passé à paramétrer ce petit bijou ! ■

Références

[1] Extension Vundle : https://github.com/gmarik/vundle

[2] Extension Vim-airline : https://github.com/bling/vim-airline

[3] Liste des thèmes disponibles pour Vim-airline : https://github.com/bling/vim-airline/wiki/Screenshots

[4] Site officiel de spf13-vim : http://vim.spf13.com/

LE MONDE MERVEILLEUX DES TESTS FONCTIONNELS

par **Frédéric Le Roy**

Beaucoup de développeurs seront d'accord avec moi : le développeur n'aime pas rédiger des tests et exécuter des campagnes de tests. Il codera toutefois des tests unitaires pour valider son code. Mais rares sont ceux qui de gaieté de cœur vont rédiger des tests fonctionnels ou en automatiser... Cet article va essayer de faire prendre conscience aux contributeurs des petits projets qu'il est important de consacrer du temps aux tests.

Ah les tests... Ce mot magique qui permet de rendre silencieux un canal IRC dès que l'on cherche des volontaires... Cette phase d'un projet que beaucoup de développeurs esquivent avec plus ou moins d'adresse.... Cette activité ô combien chronophage (au début du projet en tout cas, car après, elle vous fait gagner beaucoup de temps), qui retarde la sortie d'une version et empêche le développeur ambitieux et inspiré d'attaquer l'implémentation d'une nouvelle fonctionnalité qui va révolutionner le monde binaire !

1 Les différents types de tests

En tout premier lieu, il faut savoir qu'il existe différents types de tests. Chaque type de tests a un objectif précis. En fonction du projet, certaines catégories de tests ne sont pas nécessaires.

Dans cet article, nous aborderons uniquement les tests fonctionnels qui représentent déjà à eux seuls un sacré travail !

1.1 Les différents tests

Il existe différents types de tests :

- Les **tests unitaires** permettent de tester des petites portions de code (fonctions, classes, modules) indépendamment du reste de l'application. Ils sont codés par les développeurs afin de vérifier qu'une évolution n'introduit pas de régression dans le code existant.
- Les **tests fonctionnels** permettent de valider des aspects fonctionnels d'une application.
- Les **tests de vulnérabilité** permettent de vérifier la sécurité de l'application et de son écosystème (versions des serveurs d'applications ou de la base de données utilisée, ...).
- Les **tests de performances** qui sont constitués de tout un ensemble de tests liés à la performance de l'application :
  - **tests de charge** : on simule l'activité d'un nombre important d'utilisateurs sur l'application. En général, on simule une activité égale ou légèrement supérieure à ce que subit l'application en conditions réelles. Ceci permet de vérifier que, suite à une évolution, l'application est toujours capable de tenir la charge.
  - **tests de performance** : ils sont très proches des tests de charge. Pour différents volumes d'utilisateurs, on va mesurer les performances de l'application (car une application peut tenir fonctionnellement en charge, mais avoir des temps de réponse déplorables passé un certain niveau de charge).
  - **tests aux limites** : il s'agit ici de simuler une activité bien supérieure à l'activité attendue, afin de voir comment se comporte l'application dans ce contexte.
  - etc.

2 Introduction aux tests fonctionnels

2.1 Les tests fonctionnels sont-ils utiles ?

Les tests fonctionnels ont une mission principale qui est de permettre la validation fonctionnelle d'une application. Là où le test unitaire permet de valider des fonctions ou portions de code, le test fonctionnel permet de valider les fonctionnalités d'une application d'un point de vue utilisateur.

On peut schématiser les tests fonctionnels en deux sous-ensembles :

- les **tests de non régression** qui permettent de vérifier que les fonctionnalités existantes n'ont pas été impactées par les derniers développements ;
- les **tests liés aux nouvelles fonctionnalités** qui permettent de vérifier que les développements ont bien été réalisés. Notez que ces tests deviennent à leur tour des tests de non régression pour les versions futures.

Vous êtes toujours sceptique ? Imaginez que tout votre code soit couvert par des tests unitaires et que ces tests unitaires soient d'une rare qualité. Tout est testé, tout est couvert, il ne sert à rien de réaliser des tests fonctionnels donc ! Erreur !!!

En effet, une fonctionnalité d'une application va probablement utiliser plusieurs fonctions de votre code. Ces fonctions seront liées entre elles via des appels, des *callbacks* ou tout autre mécanisme permis par votre langage de prédilection. Il suffit donc d'un mauvais enchaînement, autrement dit d'un grain de sable pour que vos fonctions si parfaites (tels de rutilants engrenages bien huilés) produisent au final un résultat erroné.... C'est ici que les tests fonctionnels ont leur utilité : ils permettent de valider la merveilleuse mécanique que vous avez mise en place et de traquer le grain de sable (voire le sac de sable dans certains cas...).

2.2 OK, c'est utile, mais ça va me prendre combien de temps ? Et c'est rentable à partir de quand ?

Il ne faut pas se faire d'illusion, créer des scénarios de test prend du temps, beaucoup de temps parfois. Passer les tests manuellement prend aussi du temps. Et pour finir, les maintenir prend du temps.

Le temps que vous prendront vos tests est très variable en fonction de votre projet, de la manière dont sont rédigés vos tests (tests manuels, tests automatiques), de leur qualité (couvrent-ils toutes les fonctionnalités, tous les cas d'usage ou seulement un sous-ensemble représentatif ?), des outils utilisés et de la manière dont vous allez gérer le *versioning* de ces tests.

Mais, et je me permets d'insister, MAIS ces tests vous feront gagner du temps. Ce temps est lui aussi très variable et dépend de plusieurs facteurs :

- La qualité de vos développements (plus vous ferez de bugs, plus les tests vous permettront de les détecter avant qu'il ne soit trop tard).
- La taille de votre communauté active : si votre communauté est active sur l'utilisation et les rapports de bugs, vous aurez potentiellement un certain nombre de tickets à traiter suite à la découverte de bugs. Qui dit tickets dit analyse de chaque ticket, correction des bugs, tests unitaires des corrections, fermeture des tickets, etc. Souvent les tickets sont créés un certain moment après que la fonctionnalité soit codée (et donc, le code n'est plus forcément très frais dans la tête du développeur : plus le temps passe, plus un bug est long à corriger).
- Le fait que vous ayez éventuellement automatisé les tests. Dans ce cas, ils pourront être lancés de manière régulière et permettront de mettre rapidement en évidence des bugs ou régressions.

3 Organismes

Il existe des organismes dont le rôle est d'expliquer les normes, méthodes et bonnes pratiques pour gérer vos tests :

- CFTL : Comité Français des Tests Logiciels (qui représente en France l'ISTQB) **[1]**.
- ISTQB : *International Software Testing Qualification Board*.

Ces organismes peuvent proposer des formations et conférences.

4 Définitions

Mettons maintenant en place le vocabulaire qui sera utilisé dans cet article.

Note

Certaines des définitions suivantes sont reprises du CFTL.

- **ISO 9000** : il s'agit de l'ensemble des normes concernant la gestion de la qualité (les tests rentrent dans ce contexte, car ils permettent de vérifier la qualité d'une application).

- **Exigence** (*requirement*) **[1]** : une condition ou capacité requise par un utilisateur pour résoudre un problème ou atteindre un objectif qui doit être tenu ou possédé par un système ou composant pour satisfaire à un contrat, standard, spécification ou autre document imposé formellement.
- **Exigence fonctionnelle [1]** : une exigence qui spécifie une fonction qu'un composant ou un système doit remplir.
- **Exigence non fonctionnelle [1]** : une exigence qui ne se rapporte pas aux fonctionnalités, mais à des attributs tels la fiabilité, le rendement, l'utilisabilité, la maintenabilité et la portabilité.
- **Non conformité** : non réalisation d'une exigence spécifique. Ce terme a été défini dans la norme ISO 9000.
- **Anomalie** : c'est le défaut de réalisation d'une exigence spécifique. Ce défaut peut être lié à un dysfonctionnement de l'application, à un non respect des spécifications ou des standards, ou à un non sens dans l'expérience utilisateur.
- **Risque** : le risque est la prise en compte d'une exposition à un danger, un préjudice ou autre événement dommageable, inhérent à une situation ou une activité. Le risque est défini par la probabilité de survenue de cet événement et par l'ampleur de ses conséquences.
- **Suite de tests** (*test suite*) **[1]** : un ensemble de plusieurs cas de tests pour un composant ou une fonctionnalité, où les post-conditions d'un test sont souvent utilisées comme pré-conditions du test suivant.
- **Cas de test** ou **scénario de test** (*test case*) **[1]** : un ensemble de valeurs d'entrée, de pré-conditions d'exécution, de résultats attendus et de post-conditions d'exécution, développées pour un objectif ou une condition de test particulier, tel qu'exécuter un chemin particulier d'un programme ou vérifier le respect d'une exigence spécifique [d'après IEEE 610].

5 La préparation des tests : les exigences

Avant de rédiger des scénarios de tests, il faut tout d'abord s'occuper des exigences : la notion d'exigence est un concept qu'il faut absolument comprendre pour organiser correctement les campagnes de tests. Les exigences doivent idéalement être définies lors de la phase de spécification d'un projet. Si le projet est déjà en cours, il faudra alors s'appuyer sur les spécifications, ou à défaut, toute documentation décrivant le projet pour rédiger ces exigences. **Dans le cas où aucune documentation n'existe, il faut commencer par rédiger ces documentations !** Une exigence est donc forcément liée à des documents quels qu'ils soient.

5.1 Exemples de quelques exigences

Voici quelques exigences tirées en vrac des besoins qu'on pourrait avoir sur un projet domotique.

E1	Le plugin qui gère la technologie 1 wire permet de relever de manière périodique la température des sondes de type DS18B20.
E2	Il est possible d'allumer ou d'éteindre la lumière.
E3	Il est possible de faire varier l'intensité d'une lumière.
E4	Le serveur doit supporter un grand nombre de connexions simultanées.

Ces exigences vous paraissent toutes correctes ? Eh bien ce n'est pas le cas.

5.2 Validité d'une exigence

5.2.1 Définition

La validité d'une exigence dépend de certains facteurs. La norme IEEE 830-1998 **[2]** définit huit critères de qualité qui permettent de dire si une exigence est valide ou non :

- **Complète** : la spécification de l'exigence est complète.
- **Cohérente** : il n'y a aucune exigence qui contredit notre exigence et elle n'est pas redondante avec une autre exigence. Il n'y a pas de non sens dans l'exigence.
- **Délimitée** : elle est dans le périmètre du projet ou du composant concerné et n'est pas redondante avec une autre exigence déjà définie pour le projet/composant.
- **Hiérarchisable** : nous sommes capables de classer l'exigence en fonction des risques liés ou de son importance, sa priorité.

- **Modifiable** : toute modification de l'exigence entraîne peu ou pas d'impact sur les autres exigences de même niveau dans la hiérarchie.
- **Non ambiguë** : il n'y a qu'une manière d'interpréter l'exigence.
- **Traçable** : nous sommes capables de retrouver cette exigence dans les spécifications et le lien avec les versions applicatives.
- **Vérifiable** : nous sommes capables de tester cette exigence.

Il n'est pas facile de valider une exigence et donc, de choisir les exigences. En effet, certains de ces critères peuvent être complexes à évaluer suivant les cas. Toutefois, il faut faire en sorte de respecter au mieux ces critères.

Lorsque les exigences sont définies avant la phase de développement, si une exigence n'est pas valide, il faut réaliser une analyse afin de rendre l'exigence valide (soit en la modifiant, soit en l'éclatant en plus petites exigences ou via tout autre moyen de la rendre valide).

5.2.2 Cas pratique

Revenons à nos quatre exigences...

E1	Le plugin qui gère la technologie 1 wire permet de relever de manière périodique la température des sondes de type DS18B20.

Cette exigence est valide : elle est suffisamment précise (technologie, type de sonde, besoin bien défini) pour être à la fois cohérente, délimitée, hiérarchisable, modifiable, non ambiguë et vérifiable. L'aspect traçable et complet est lié à la documentation et le versioning ; pour l'exemple nous supposerons que ces deux points sont OK.

E2	Il est possible d'allumer ou d'éteindre la lumière.
E3	Il est possible de faire varier l'intensité d'une lumière.

Ces deux exigences ne sont pas valides : elles ne sont pas délimitées. En effet, plusieurs technologies domotiques permettent de réaliser ces actions : X10, PLCBUS, Z-Wave, etc. Chaque technologie étant différente, il y aura plusieurs implémentations différentes de la fonctionnalité. Il faudrait donc ici définir chacune de ces 2 exigences pour chaque type de technologie à supporter.

E4	Le serveur doit supporter un grand nombre de connexions simultanées.

Ici non plus l'exigence n'est pas valide. Elle est ambiguë : on ne précise pas le nombre de connexions, le type de requêtes réalisées par les connexions, etc. Le titre de l'exigence peut rester vague (et donc, rester tel quel), mais alors le document associé à l'exigence doit contenir toutes ces informations.

5.3 Hiérarchiser les exigences

Dans le cas de tests fonctionnels manuels, il peut être intéressant de ne pas passer toutes les suites de tests afin de gagner du temps. Si nous reprenons l'exemple d'un projet domotique qui gère plusieurs technologies, lors d'évolutions sur une technologie, il n'est pas nécessaire (si le projet est modulaire) de passer les suites de tests liées aux autres technologies.

De la même manière, dans le cas d'un petit *patch*, il peut être judicieux de ne pas passer toutes les suites de tests, mais seulement les suites de tests les plus représentatives.

Afin de pouvoir réaliser ceci, il vous faut tout d'abord hiérarchiser vos exigences : il s'agit tout simplement de les classer dans une arborescence. Un nœud pourra être lié à une technologie domotique et toutes les exigences de ce nœud ne concerneront que cette technologie.

5.4 Le facteur de risque

Idéalement, chaque exigence doit être associée à un facteur de risque, ce facteur permettra ensuite de déterminer les exigences qui sont importantes de celles qui le sont moins (et donc, d'orienter correctement les campagnes de tests). Comment le calculer ?

Remémorons-nous la définition du risque : « Le risque est la prise en compte d'une exposition à un danger, un préjudice ou autre événement dommageable, inhérent à une situation ou une activité. Le risque est défini par la probabilité de survenue de cet événement et par l'ampleur de ses conséquences. ». Telle quelle, cette définition ne nous sert à rien. En effet, comment calculer le facteur de risque d'une exigence avec cette définition ?

Pour calculer les facteurs de risque, dans les sociétés de services, les approches de type RRBT (*Risk and Requirement Based Testing*) ou 2RBT ou R2BT sont souvent utilisées. Ces approches, toutes brevetées (beurk) par de grandes sociétés (Logica, etc.) sont relativement efficaces pour déterminer les facteurs de risque. Elles peuvent par contre être complexes et lourdes à mettre en place sur de petits projets (d'autant plus si ce n'est

pas votre métier !). Nous allons donc plutôt voir ici comment déterminer ce facteur avec du bon sens.

5.4.1 Liste des types de priorités

Tout d'abord, il nous faut déterminer une liste de priorités. Il faut éviter que cette liste contienne plus de 4 éléments, sinon vous allez passer votre temps à calculer des facteurs (ce n'est pas le but du jeu non plus...). Classiquement, on va retrouver cette liste de priorités qui correspondra à 90 % des projets :

- **mineure** : un défaut sur une exigence de cette priorité représentera juste une petite gêne pour un utilisateur (léger problème de mise en page, traduction peu précise mais non impactante).
- **normale** : ici, un défaut entraînera une vraie gêne de l'utilisateur.
- **importante** : le défaut d'une telle exigence sera critique, mais il existe une méthode de contournement.
- **indispensable** : un défaut entraîne l'impossibilité pour l'utilisateur de finir une action ou de compléter un processus.

En fonction des projets, une exigence peut ne pas avoir le même impact. Par exemple, une exigence d'impression a peu de criticité dans une application lambda, alors que cette même exigence sera indispensable au guichet de la SNCF (imaginez-vous payer et ne pas recevoir le fameux titre de transport car l'impression ne fonctionne plus ?).

Chaque exigence sera donc liée à une priorité.

5.4.2 Points sensibles

Maintenant, vous pouvez lister les points sensibles de votre applicatif. Cela peut être :

- la pérennité des données de l'application (exemple : une purge d'une table de statistiques),
- la fiabilité des données (exemple : prévisions météo),
- le nombre d'utilisateurs concernés et l'importance de ces utilisateurs,
- les impacts légaux (respect de la vie privée : divulgation accidentelle des listes des membres avec leurs e-mails pour un forum),
- la sécurité.

Limitez-vous à quelques points et classez-les par ordre d'importance.

5.4.3 Taux d'utilisation d'une fonctionnalité

Si une fonctionnalité de votre application est utilisée 9 fois sur 10, c'est qu'elle est très importante ! Donnez un taux d'utilisation à vos exigences en fonction de son ratio d'utilisation (que ça soit une valeur théorique ou mesurée).

5.4.4 Complexité d'une fonctionnalité

Plus une fonctionnalité est complexe, plus elle a de chance d'être buggée suite à une évolution. Donnez une note de complexité aux exigences associées.

5.4.5 On mixe tout et on obtient le niveau de risque lié à chaque exigence

Une fois que vous avez analysé ces différents points pour chaque exigence (un tableau devrait vous aider), il vous sera ensuite facile de déterminer le risque d'une exigence par rapport à l'ensemble des autres exigences. Cette méthode manuelle n'est pas très précise, mais reste quand même de manière globale plutôt efficace.

Voici un exemple de tableau (mais sans le résultat, car ce sont des données fictives) :

Exigence	Priorité (1... 4)	Point sensible (1... 5)	Taux d'utilisation (0... 100%)	Complexité (1... 10)
Exigence n°1	Indispensable (4)	Sécurité (4)	30 %	2
Exigence n°2	Normale (2)	Pérennité des données (5)	50 %	8
Exigence n°3	Mineure (1)	N/A	5 %	6

On voit bien ici que l'exigence n°3, malgré une complexité plus importante que l'exigence n°1, n'aura pas un facteur de risque important : elle est peu utilisée, ne concerne aucun point sensible et est de priorité mineure. Les exigences n°1 et n°2 auront par contre un facteur de risque très proche malgré des différences importantes.

Dans le monde de l'entreprise, il existe des formules très précises pour déterminer le facteur de risque en fonction de toutes ces données. Pour un petit projet libre (voire même un moyen), cela s'apparente à sortir le missile nucléaire face à un Pinscher nain **[3]** !

PROFESSIONNELS DES TICE, COLLECTIVITÉS, ÉCOLES D'INGÉNIEURS, UNIVERSITÉS, R & D, ENSEIGNANTS, ...

GNU LINUX MAGAZINE / FRANCE

VOUS PROPOSE 2 NOUVEAUX SERVICES !

1 VOUS SOUHAITEZ LIRE GNU/LINUX MAGAZINE EN VERSION PDF ?

PDF

LINUX MAGAZINE / FRANCE + HORS-SÉRIES

VOICI LES ABONNEMENTS PDF COLLECTIFS !

Ce service vous permet d'abonner votre structure (écoles, collectivités, entreprises, etc.) à l'édition PDF de nos magazines afin d'en profiter dès leur parution chez les marchands de journaux.

Sans DRM, téléchargez simplement, lisez et annotez vos eBooks sur votre PC, smartphone ou liseuse électronique.

2 VOUS SOUHAITEZ RETROUVER ET CONSULTER LES ARTICLES DE GLMF ?

BASE DOCUMENTAIRE LINUX MAGAZINE / FRANCE + HORS-SÉRIES

VOICI LA BASE DOCUMENTAIRE !

L'accès à la base documentaire en ligne de GNU/Linux Magazine et de ses Guides vous permettra d'effectuer des recherches dans la majorité des articles parus, qui seront disponibles 6 mois après leur parution en magazine. Vous pourrez ainsi effectuer des recherches sur les articles indexés, copier les codes, etc.

La consultation s'effectue sur notre nouveau service **connect.ed-diamond.com** qui est en place depuis janvier 2014 (n'hésitez pas à le visiter !)...

connect.ed-diamond.com

N'hésitez pas à consulter notre offre sur **boutique.ed-diamond.com**,

« **Base Documentaire TOTALE** » à **399 € HT/an**

(5 connexions comprises) pour profiter de la base documentaire de l'ensemble des parutions des Éditions Diamond !

BESOIN DE RENSEIGNEMENTS SUPPLÉMENTAIRES OU D'UN DEVIS SUR MESURE ?

N'hésitez pas à envoyer un e-mail à **aboprof@ed-diamond.com** ou à téléphoner au **+33 (0)3 67 10 00 27**

6 Préparation des scénarios de test

6.1 Avant toute chose, l'outillage !

Ici, pas de recette miracle, tout est question de goût, du nombre de scénarios de tests à réaliser, des possibilités techniques (automatisation possible ou pas), éventuellement des habitudes des personnes qui vont s'occuper de créer les tests (si vous faites appel à une équipe externe), etc.

Voici quelques questions qui pourront vous aider à vous orienter :

- Est-ce que l'application peut être testée de manière automatique ? Et est-ce que vous le souhaitez ?
- Est-ce que l'application est importante et nécessite d'être couverte par beaucoup de tests ?
- Quand est-ce que vous souhaitez pouvoir tester l'application ? À chaque commit ou seulement en fin de version ?

La manière de gérer les scénarios de tests peut varier d'un projet à un autre. Mais je pense qu'on peut raccourcir ces manières de faire en 3 catégories :

- L'utilisation d'un référentiel de tests,
- L'automatisation totale des tests (le code des tests faisant office de référentiel),
- Un habile mix entre les deux ; c'est sûrement la méthode la plus fiable et la plus « esthétique », mais c'est aussi la plus chronophage.

6.1.1 Le référentiel des tests

Il existe plusieurs solutions allant du trop classique classeur (Excel, LibreOffice, etc.) à la solution propriétaire et ultra complète qui a un coût non négligeable, en passant bien évidemment par des solutions libres, peut-être moins sexy et moins complètes, mais tout aussi efficaces pour un usage classique.

Le cas du classeur tient plus de l'artisanat qu'autre chose et je vous le déconseille pour un tas de raisons :

- La complexité d'utilisation avec plusieurs personnes différentes simultanément,
- Les limites liées au format classeur,
- Le risque de corruption du fichier (et cela arrive plus souvent qu'on ne le pense, notamment avec une utilisation partagée sur un réseau).

Les solutions dédiées se présentent pour la plupart sous forme d'applications web, dans lesquelles il est possible de :

- gérer les exigences,
- créer des suites de tests et les associer à des exigences,
- créer des scénarios de test dans les suites de tests,
- gérer les versions de la solution et y affecter des suites de tests,
- exécuter les tests sur une version,
- obtenir des statistiques sur la couverture des tests (nombre de tests passés, échoués, bloqués, …),
- créer facilement des tickets de bug dans votre tracker préféré (voire directement dans la solution de test pour les plus complètes),
- etc.

Ces solutions nécessiteraient un article dédié. Voici quelques noms de solutions libres : TestLink, Squash, RTMR, Salomé TMF.

Pour illustration, voici à quoi peut ressembler un scénario de test :

Étape	Action	Résultat attendu
1	Se connecter à l'URL du serveur d'administration http://ip:port/login	Une page s'affiche avec 3 éléments : ▪ zone de texte pour le login ▪ zone de texte pour le mot de passe ▪ bouton « Login »
2	Saisir le login : « admin1 »	Le login est affiché en clair.
3	Saisir un mauvais mot de passe : « xxxx »	Le mot de passe est affiché avec des « * » à la place des caractères.
4	Cliquer sur « Valider »	Une notification apparaît : « Login ou mot de passe incorrect ». L'utilisateur peut modifier les champs login et mot de passe.

Il peut être très intéressant dans la description du test de mentionner une estimation du temps qui sera nécessaire pour passer le test (ceci permettra par la suite de calculer le temps nécessaire pour passer toute une campagne de tests).

6.1.2 L'automatisation des tests

Dans l'hypothèse d'une automatisation des tests, je suppose que nous n'allons pas utiliser de référentiel de tests.

Automatiser les tests a un coût : ceci nécessite de réaliser des développements. Ces mêmes développements peuvent eux-mêmes être buggés ! Toutefois, il faut bien garder en tête que le coût de passage des tests est nul en temps humain. Si vous avez beaucoup de tests à réaliser (et surtout que le fonctionnement de l'application ne change pas radicalement à chaque version), cela signifie aussi qu'il y aura beaucoup de tests à passer. L'automatisation vous permettra ici de réaliser de nombreux tests (voire tous les tests) de manière très régulière !

Il y a en général 2 écoles sur le lancement des tests automatisés :

- Lancement à chaque commit sur le dépôt principal : ici, chaque commit sera testé individuellement et donc, il sera très aisé d'identifier la modification qui a généré une erreur.
- Lancement quotidien durant la nuit. Cette solution est utilisée principalement en entreprise sur des environnements de développement : quand il y a beaucoup de développeurs sur un projet, il y a beaucoup de commits (et SVN restant beaucoup utilisé, un commit SVN signifie donc un commit sur le serveur a contrario de gestionnaires de versions comme Git ou Mercurial qui permettent de réaliser des commits en local avant de les pousser) et donc, il peut y avoir une forte charge générée sur le serveur de tests. De même, suivant votre infrastructure, vous ne pourrez lancer qu'une campagne de tests à la fois.

Attention !

Avoir seulement des tests automatisés implique plusieurs choses :

- Il faudra hiérarchiser de manière claire ces tests dans votre dépôt sous peine de vous y perdre.
- Il faudra bien documenter le code de vos tests avec des exemples dès que possible dans les commentaires. Ceci est très important pour assurer la maintenance des tests !
- Si vous désirez ne pouvoir lancer qu'un sous-ensemble des tests, il peut être nécessaire de vous outiller un peu, afin d'avoir une liste des tests (avec par exemple des niveaux de criticité) et un lanceur.

Un autre avantage de l'automatisation des tests est la simplification du versioning : si vous incluez le code des tests dans les branches de votre gestionnaire de versions, il devient très facile de gérer ces tests, tout comme vous le faites déjà pour le code.

L'automatisation des tests peut être faite via différents outils, que ça soit une solution codée « maison » (à base de scripts, de Python ou tout autre langage), ou grâce à des frameworks ou outils de tests dédiés. Chaque type d'application aura l'outil qui lui correspond et ici, tout moteur de recherche sera votre ami ! Vous pourrez trouver une liste d'outils sur le portail anglais de Wikipédia sur les tests logiciels **[4]**.

6.1.3 Le mix !

Pour finir, le mix entre les 2 méthodes est à mon avis le Graal du test, mais il est aussi bien plus complexe à gérer !

Des solutions propriétaires permettent de réaliser les deux de manière très intégrée (en admettant que les capacités d'automatisation de la solution correspondent à vos besoins, ce qui n'est pas toujours le cas), mais ces solutions ne sont pas forcément adaptées à de petits projets.

La principale difficulté de cette méthode est de garder en phase à la fois le référentiel et les tests automatisés. Si vous y arrivez, il faut quand même avouer que pour les personnes qui vont gérer les tests, ça sera assez sympathique, car ils bénéficieront de tout le confort d'une solution dédiée (avec les rapports, les graphiques, …) et de l'automatisation (créer une campagne et la lancer suffira pour lancer tous les tests sélectionnés pour la campagne).

6.1.4 Le mix, mais en mieux !

Il existe une dernière méthode qui permet de mélanger allègrement tests automatisés et référentiel de tests lisible humainement : le *Behavior Driven Development* **[5]**. C'est une méthode qui est axée sur le langage naturel et va permettre de rédiger des tests humainement lisibles, par exemple :

```
Feature: se connecter

  Scenario: connexion avec un compte valide
     Given l'application est démarrée
      when je me connecte en tant que John avec le mot de passe tutu
      then la page d'accueil s'affiche
  Scenario: connexion avec un compte invalide
     Given l'application est démarrée
      when je me connecte en tant que John avec le mot de passe xxxx
      then la page d'erreur mauvais utilisateur s'affiche
```

Cette méthode est vraiment très séduisante et a un aspect un peu magique.

Petite digression technique pour l'explication : pour que ce type de tests fonctionne, chaque phrase est (pour faire très court) liée à une fonction. Par exemple, la phrase « je me connecte en tant que John avec le mot de passe tutu » va être liée à une fonction qui, via des expressions régulières, va récupérer l'identifiant et le mot de passe, puis tenter une connexion sur la page de login avec les données fournies dans la phrase. Une fois toutes les fonctions prêtes et les phrases documentées, des personnes qui ne sont pas des développeurs peuvent créer de manière très intuitive de nouveaux tests.

6.2. La rédaction des scénarios de tests

La rédaction des scénarios de tests est plus une affaire d'habitude qu'autre chose : les premiers scénarios sont toujours les plus délicats à écrire et souvent le béotien va réécrire plusieurs fois ses scénarios. Je vous invite, avant de commencer, à lire des scénarios de tests existant sur d'autres applications que la vôtre.

Pendant la rédaction des scénarios, ne perdez jamais de vue ces questions :

- Est-ce que vous avez donné toutes les informations nécessaires pour le test ? Par exemple, un template d'URL (http://ip:port/MonAppli/login), l'identifiant à utiliser (admin, guest, etc.), les informations à saisir dans un formulaire ou à défaut, une explication de comment trouver ces informations, etc.
- Est-ce que le scénario est précis et ne donne pas lieu à plusieurs interprétations ?
- Est-ce que le scénario « B » nécessite qu'un scénario « A » ait été validé afin de pouvoir le passer ? Si oui, il faut donner cette information.

7 La campagne de tests

Vous avez maintenant :

- des exigences hiérarchisées et avec un niveau de risque,
- chaque exigence comprend une ou plusieurs suite(s) de tests.

Il reste donc à préparer la campagne de tests !

7.1 Cas des tests automatisés

Ici, peu de réflexion : si les tests sont automatisés, au mieux ils se lancent déjà tout seuls de manière régulière, au pire, il fait les lancer... Sauf cas particulier (des tests qui durent quelques jours), vous pouvez lancer la totalité des tests pour plus de sécurité.

Une fois les tests lancés, vous récupérerez un rapport (avec un format qui sera lié à l'outillage de test choisi), qu'il vous faudra lire pour voir s'il faut créer des rapports de bugs. Dans le cas où les tests sont lancés de manière récurrente, c'est un travail à réaliser au fil de l'eau. Dans le cas d'un lancement manuel, il vous faudra le faire à la fin de l'exécution des tests.

7.2 Cas des tests manuels

Dans le cas de tests manuels, il vous faut préparer la campagne de tests.

Tout d'abord, il vous faut savoir si vous désirez tester toutes les exigences ou seulement une partie. S'il s'agit de la première phase de tests de la nouvelle version de l'application et que la plupart des fonctionnalités ont été impactées par les développements, il vous faudra valider toutes les exigences. Dans le cas où seules certaines fonctionnalités ont été impactées, il vous faudra valider toutes les exigences liées et une partie des autres exigences (liste à définir en fonction du risque et du temps que vous pouvez consacrer aux tests !).

Si toutes les fonctionnalités sont impactées, que vous êtes en version candidate et que toutes les exigences ont déjà été testées pendant les versions bêta, vous pouvez réduire les exigences à tester en fonction du temps que vous pouvez allouer aux tests (et bien entendu, toujours en prenant en compte le risque).

Conclusion

Nous arrivons au bout de cet article qui n'est finalement qu'une petite mise en bouche face à ce sujet qu'est le test logiciel. Il resterait tant de choses à dire...

J'espère vous avoir convaincu de la nécessité de réaliser des tests fonctionnels et vous avoir donné quelques pistes pour vous lancer à votre tour ! ■

Références

[1] Comité français des tests fonctionnels : http://www.cftl.net/

[2] Description de la norme IEEE 830-1998 : http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html

[3] Définition du Pinscher nain : http://fr.wikipedia.org/wiki/Pinscher_nain

[4] Liste d'outils d'automatisation des tests : http://en.wikipedia.org/wiki/Portal:Software_Testing

[5] Définition de la méthode agile de BDD : http://fr.wikipedia.org/wiki/Behavior_Driven_Development

28 06 2014

NUIT DU HACK

DISNEYLAND PARIS

12TH EDITION

CHALLENGES
CONFERENCES
WORKSHOPS
BUG BOUNTY

TAKE THE CHALLENGE

Schedule + Booking:
www.nuitduhack.com

Community:
www.hackerzvoice.net

Social:
@hackerzvoice

Un évènement organisé par **HACKERZVOICE** en partenariat avec **SYSDREAM**

LA VRAIE ALTERNATIVE OPEN SOURCE POUR CONSTRUIRE VOTRE BOX D'ENTREPRISE

CONSTRUISEZ VOTRE PROPRE SYSTÈME DE PARTAGE DE FICHIERS ET DE COFFRE-FORT ÉLECTRONIQUE DANS VOTRE CLOUD PRIVÉ !

LinShare.org est la solution choisie par : INSERM, Conseil d'État, Ministère de l'Intérieur, de l'Outre-mer, des Collectivités territoriales et de l'Immigration, Ministère de l'agriculture, de l'agro-alimentaire et de la forêt, RSF, Groupe CASINO, Brake France, la Cours des Comptes …

LINAGORA

www.linagora.com www.linshare.org