



# **LINUX**

**MAGAZINE / FRANCE**

DÉVELOPPEMENT SUR SYSTÈMES UNIX, OPEN SOURCE & EMBARQUÉ

**N°202**

MARS  
2017

FRANCE  
MÉTRO.: 7,90 €  
DOM/TOM: 8,50 €  
BEL/LUX/PORT.  
CONT.: 8,90 €  
CH: 13 CHF  
CAN: 14 \$CAD

L 19275 - 202 - F: 7,90 € - RD



IA / Style transfer

Réseaux de neurones,  
GPU, traitement d'images,  
transfert de style...

## **DONNEZ UN CERVEAU À VOTRE PC !**

p.12

- Configurez votre réseau de neurones
- Entraînez-le et générez des images
- Optimisez le traitement et réduisez le temps de calcul

Apache / HAProxy

## **AJOUTEZ LE SUPPORT SSL/TLS DANS VOTRE CONFIGURATION LOAD BALANCING**

p.88

Bas niveau / Objet

## **APPRENEZ À CRÉER ET UTILISER LES ITÉRATEURS EN C++**

p.48

Embarqué / C++

## **METTEZ EN ŒUVRE UNE CAMÉRA 3D AVEC VOTRE RASPBERRY PI ET POINTCLOUD**

p.34



Python / Paquets

## **RENDEZ ENFIN VOS APPLICATIONS INSTALLABLES AVEC PIP3 !**

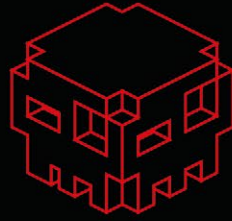
p.76

Dev / Annotation

## **ALLÉGEZ ET SIMPLIFIEZ VOTRE CODE JAVA AVEC LOMBOK**

p.62

\* WARGAME \* CONFERENCES \* CHALLENGES \* WORKSHOPS \*



# NUIT DU HACK XV

24->25 Juin 2017  
New York Hôtel Convention Center  
Disneyland Paris  
[www.nuitduhack.com](http://www.nuitduhack.com)  
@hackerzvoice



SHALL WE PLAY A GAME?



10, Place de la Cathédrale - 68000 Colmar - France  
Tél. : 03 67 10 00 20 - Fax : 03 67 10 00 21  
E-mail : [lecteurs@gnulinuxmag.com](mailto:lecteurs@gnulinuxmag.com)  
Service commercial : [abo@gnulinuxmag.com](mailto:abo@gnulinuxmag.com)  
Sites : [www.gnulinuxmag.com](http://www.gnulinuxmag.com) - [www.ed-diamond.com](http://www.ed-diamond.com)

**Directeur de publication :** Arnaud Metzler  
**Chef des rédactions :** Denis Bodor  
**Rédacteur en chef :** Tristan Colombo  
**Résponsable service infographie :** Kathrin Scali  
**Réalisation graphique :** Thomas Pichon  
**Résponsable publicité :** Valérie Frécharde,  
Tél. : 03 67 10 00 27 - [v.frechard@ed-diamond.com](mailto: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 €



La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans GNU/Linux Magazine France est interdite sans accord écrit de la société Les éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à GNU/Linux Magazine France, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire. Toutes les marques citées dans ce numéro sont déposées par leur propriétaire respectif. Tous les logos représentés dans le magazine sont la propriété de leur ayant droit respectif.

### SUIVEZ-NOUS SUR :



[https://www.facebook.com/  
editionsdiamond](https://www.facebook.com/editionsdiamond)



@gnulinuxmag

p.41/42

DÉCOUVREZ TOUS NOS  
ABONNEMENTS MULTI-SUPPORTS !

LES ABONNEMENTS ET LES ANCIENS  
NUMÉROS SONT DISPONIBLES !



EN VERSION PAPIER ET PDF :

[www.ed-diamond.com](http://www.ed-diamond.com)



Codes sources sur  
<https://github.com/glmf>

# ÉDITO



« J'ai vu tant de choses que vous, humains, ne pourriez pas croire. De grands navires en feu surgissant de l'épaule d'Orion. J'ai vu des rayons fabuleux, des rayons C, briller dans l'ombre de la porte de Tannhäuser. Tous ces moments se perdront dans l'oubli comme les larmes dans la pluie. »

Extrait de « Blade Runner » - H. Fancher et D. Peoples

De plus en plus, les avancées en intelligence artificielle permettent de résoudre des problèmes complexes, de tendre vers une intelligence « humaine » telle que celle que l'on peut voir associée aux répliquants du film « Blade Runner » de Ridley Scott (1982). Les machines sont ainsi capables de nous égaler voire de nous surpasser dans certains domaines : le jeu de dames, le jeu d'échecs et même dernièrement le jeu de Go. On peut voir aussi les énormes progrès réalisés par la société Boston Dynamics en ce qui concerne l'analyse et l'adaptation à un environnement. Ses robots bipèdes ou quadrupèdes sont capables de retrouver rapidement leur équilibre et d'adapter leur comportement en fonction des modifications de leur environnement (prendre un objet qui est déplacé par un être humain par exemple - <https://youtu.be/rVlhMGQgDkY>).

La créativité est toutefois un domaine réservé à l'homme et la conscience des robots de Westworld est encore loin. Toutefois, les techniques utilisées en IA réservent parfois des surprises. Ainsi, les réseaux de neurones représentant un formidable outil d'apprentissage, il est possible d'« apprendre » un style pour l'appliquer à une image. On peut ainsi obtenir de très belles images par « transfert de style », et celles-ci semblent être de véritables créations. On ne peut pas parler d'art, mais les résultats sont intéressants et vous pourrez expérimenter cette technique grâce à l'article que nous vous proposons ce mois-ci.

Outre l'IA, vous retrouverez bien entendu dans ce numéro toute la diversité qui fait la richesse de GNU/Linux Magazine avec :

- la distribution de vos paquets **Python** sur PyPi ;
- les itérateurs en **C++** ;
- l'annotation de code en **Java** avec **Lombok** ;
- le support SSL/TLS avec **HAProxy** ;
- l'utilisation d'une caméra 3D sur **Raspberry Pi** ;
- etc.

Je vous souhaite une bonne lecture et je vous retrouverai avec plaisir le mois prochain...

Tristan Colombo

## Donnez-nous votre avis sur le magazine !

Avec son numéro 200, GNU/Linux Magazine s'est transformé et a lancé sa nouvelle formule. N'hésitez pas à prendre 30 secondes pour nous donner votre avis sur cette nouvelle version du magazine ! Pour cela, vous pouvez :

- répondre à notre petit sondage disponible sur le blog du magazine (<http://www.gnulinuxmag.com/sondage-donnez-nous-votre-avis-sur-la-nouvelle-formule-du-magazine/>)
- nous écrire à [lecteurs@gnulinuxmag.com](mailto:lecteurs@gnulinuxmag.com)
- nous communiquer vos impressions sur le compte Twitter du magazine @gnulinuxmag

Merci à vous & bonne lecture !

# DISPONIBLE DÈS LE 10 MARS

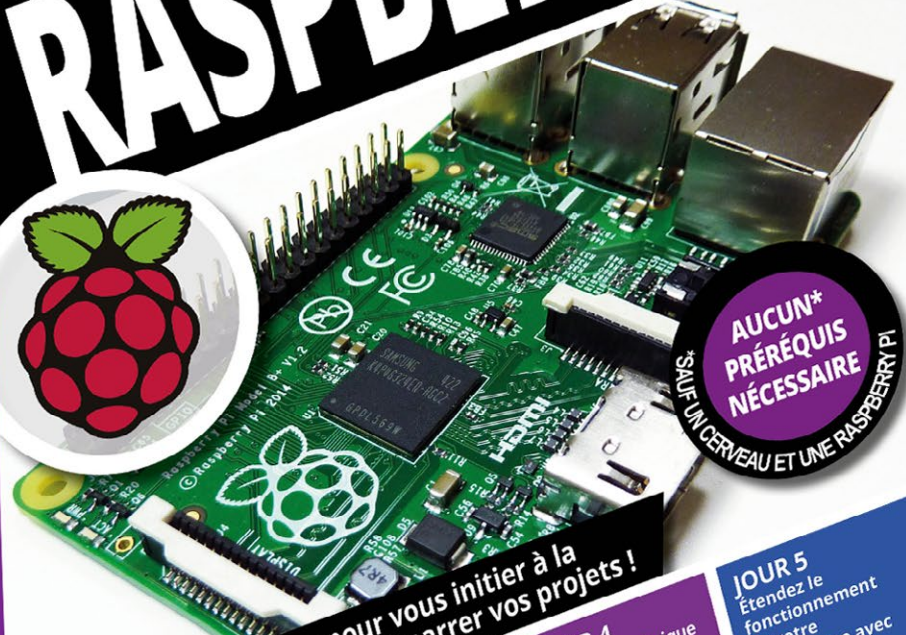
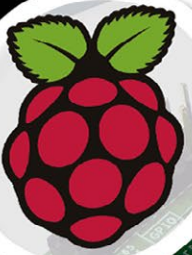
## HACKABLE HORS-SÉRIE N°2 !

LES GUIDES DE  
**HACKABLE**  
MAGAZINE

HORS-SÉRIE  
N°2

FRANCE METRO : 12,90 € — CH : 18,00 CHF — BELPORT.CONT : 13,90 € — DOM TOM : 13,90 € — CAN : 19,90 \$ CAD


# DÉBUTEZ EN PROGRAMMATION SUR RASPBERRY PI



**AUCUN\*  
PRÉREQUIS  
NÉCESSAIRE**  
\*SAUF UN CERVEAU ET UNE RASPBERRY PI

Le guide 100% pratique pour vous initier à la programmation Python et démarrer vos projets !

- JOUR 1** Créez un programme et apprenez à contrôler l'affichage
- JOUR 2** Donnez vie à votre code avec des houches et des fonctions
- JOUR 3** Faites interagir votre programme avec un utilisateur
- JOUR 4** Ajoutez une logique interne et organisez votre code en un tout
- JOUR 5** Étendez le fonctionnement de votre programme avec des fichiers

Édité par Les Éditions Diamond  
L 13630 - 2H - F: 12,90 € - RD  
  
www.ed-diamond.com

# DÉBUTEZ EN PROGRAMMATION SUR RASPBERRY PI !

# NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

# <http://www.ed-diamond.com>



# SOMMAIRE

## GNU/LINUX MAGAZINE FRANCE N° 202

### ACTUS & HUMEUR

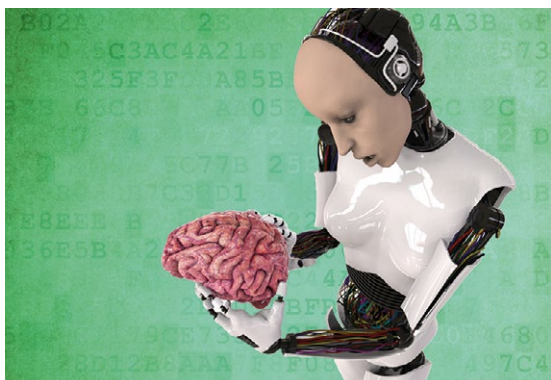
#### 06 FOSDEM 2017, BRUXELLES

Le FOSDEM est encore et toujours la principale réunion annuelle des développeurs open source européens, mondiaux voire intergalactiques.

### IA, ROBOTIQUE & SCIENCE

#### 12 TRANSFERT DE STYLE : ET SI VAN GOGH PEIGNAIT TUX ?

Qu'est-ce que le style transfer ? Comment cela fonctionne-t-il ? Comment en faire chez soi ?...



### SYSTÈME & RÉSEAU

#### 24 FACILITEZ-VOUS LA VIE AVEC LA LIGNE DE COMMANDES

Si vous utilisez GNU/Linux, alors vous utilisez forcément le shell. Que vous soyez développeur chevronné, administrateur gourou ou utilisateur éclairé, vous passez un temps incroyable devant des lignes de commandes...

### IoT & EMBARQUÉ

#### 34 CAMÉRA 3D ET NUAGE DE POINTS

Pour pouvoir se mouvoir dans l'espace, il est indispensable de pouvoir évaluer les distances avec les objets environnants. Les systèmes d'assistance à la conduite (freinage anticipé, détection de piéton, « radar » de recul) ou bien les drones en sont entre autres de bons exemples...

### KERNEL & BAS NIVEAU

#### 48 LES ITÉRATEURS EN C++

Les itérateurs sont un des piliers du langage C++. Ils permettent de réaliser énormément de tâches essentielles...

### HACK & BIDOUILLE

#### 54 LIVE-SYSTEM FROM SCRATCH

Un système vif (live-system) avec données persistantes permet de démarrer, avec une simple clé USB, n'importe quel ordinateur comme si c'était le sien...

### LIBS & MODULES

#### 62 CODE JAVA CONCIS AVEC LOMBOK

Si Java est un langage reconnu pour sa (relative) simplicité et clarté, il ne l'est clairement pas pour sa concision. À bien des égards, il est même parfois très (trop?) verbeux...

#### 68 CONCEPTION D'UN ÉMULATEUR DE LEDS WS2812

Les leds WS2812, encore appelées NeoPixels, sont des leds RGB programmables pour lesquelles il existe des bibliothèques (en C) et des modules (en Python)...

#### 76 RIEN NE VAUT LA PRATIQUE : CRÉATION D'UN PAQUETAGE POUR PIP

La documentation permettant de créer un packaging pour un projet Python est abondante et décrit une procédure relativement simple...

### MOBILE & WEB

#### 82 MOTEUR DE TEMPLATE TWIG : HÉRITAGE ET MACROS

Le moteur de template open source Twig facilite le développement, la sécurisation et la maintenance d'applications web PHP...

### SÉCURITÉ & VULNÉRABILITÉ

#### 88 CONFIGURATION TLS AVEC HAPROXY ET OPENSSL

Nous utilisons tous les jours des connexions chiffrées, le plus souvent à travers le protocole TLS. Que vous soyez un(e) internaute soucieux(se) de la sécurité...

### ABONNEMENTS

41/42 : abonnements multi-supports

# FOSDEM 2017, BRUXELLES

**PIERRE FICHEUX**

[Directeur technique Smile-ECS, Responsable Majeure GISTRE à l'EPITA]

**JEAN-MICHEL FRIEDT**

[FEMTO-ST Time & Frequency/SENSeOR]

**MOTS-CLÉS : CONFÉRENCE, OPEN SOURCE, DÉVELOPPEURS, ACTUALITÉ, COMPTE-RENDU**



**Le FOSDEM est encore et toujours la principale réunion annuelle des développeurs open source européens, mondiaux voire intergalactiques. Cet article est un bref compte-rendu (très partiel) de sujets évoqués dans quelques « developer rooms » fréquentées par les deux auteurs.**

L'ambiance du FOSDEM (*Free & Open source Software Developer's European Meeting*) est totalement unique puisque agnostique par rapport au côté commercial qui règne dans la majorité des autres conférences. Cela n'empêche pas des entreprises majeures (**Samsung, Bosh, Intel**, etc.) de présenter des travaux au travers de projets officiels ou non et donc de côtoyer le développeur indépendant.

L'accès est entièrement libre et aucune inscription n'est demandée. La manifestation dépasse désormais les frontières de quelques passionnés puisque le journal *Libération* lui a récemment consacré un article le 6 février 2017 [1].

Outre le contenu technique toujours très intéressant, l'organisation est assez remarquable quant à la qualité de l'accueil, du site web et des documentations fournies (programme, plan d'accès). Comme tous les ans nous avons pu profiter d'un ciel typiquement belge – *excessivement énervant* comme on dit là-bas :) - durant les deux jours de l'événement ! Le seul point négatif du FOSDEM est les redoutables « food trucks » qui permettent de se restaurer de manière approximative tout au long de la journée. Cependant, un

effort notable fut réalisé cette année puisque j'ai tout de même réussi à y trouver des frites/mayonnaise correctes.

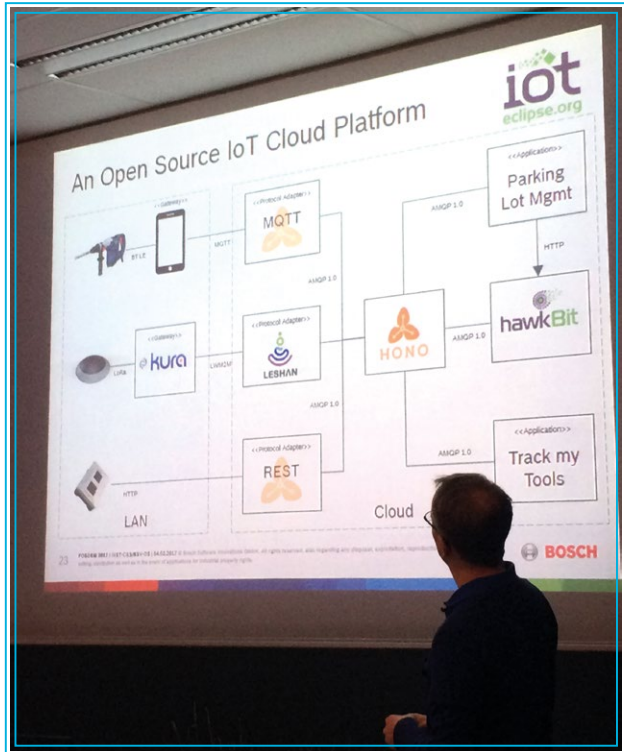
Comme pour les années passées, la stratégie était de se focaliser sur les sujets assez techniques (les *Developer rooms*) et proches de nos centres d'intérêt. J'ai pour ma part surtout fréquenté dans le bâtiment UD les salles UD2.120/UD2.218A hébergeant le sujet *Embedded, Mobile and Automotive* ainsi que le sujet *Internet of things* dans le bâtiment AW (salle AW1.126). Plusieurs conférences se terminaient par des démonstrations, ce qui est toujours agréable (mais périlleux pour le conférencier). C'est là aussi une des caractéristiques du FOSDEM. Jean-Michel Friedt a de son côté assisté aux conférences *Software Defined Radio, Electronic Design Automation* (salle AW1.120) et *Geospatial*.

Les liens vers les sujets évoqués sont disponibles sur le programme du FOSDEM [2].

## CONFÉRENCES

Après une première visite à l'amphithéâtre Janson (bondé pour le traditionnel « Welcome to FOSDEM », nous avons rapidement rejoint les salles correspondant à nos cibles, en premier lieu le sujet sur *Internet of things*.

La première conférence fut consacrée à un hommage à Pieter Hintjens [3], développeur et auteur belge décédé en octobre 2016. Il avait entre autres participé au FOSDEM 2016. Le dernier ouvrage publié par Pieter intitulé « *Confessions of a Necromancer* » nous a été chaudement recommandé. Il est disponible gratuitement sur la page personnelle de Pieter, mais l'achat de l'ouvrage en version papier [4] pourrait aider financièrement sa famille et surtout les enfants de Pieter, encore très jeunes.

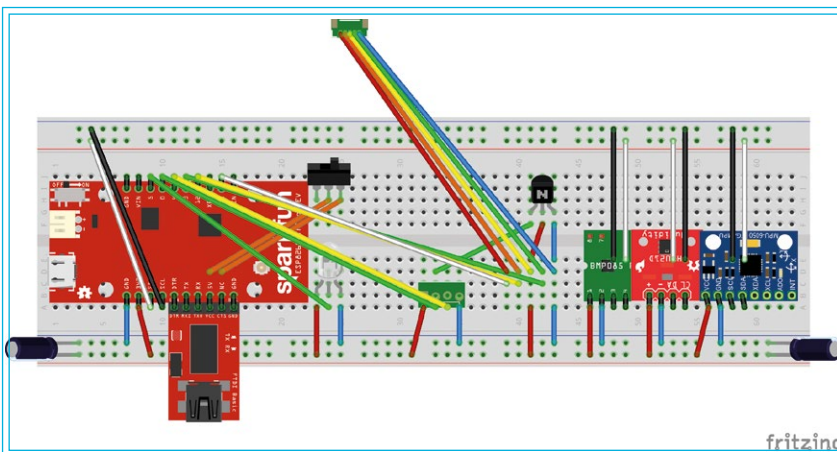


Composants IoT de la fondation Eclipse

Un des points forts de cette salle IoT fut la conférence de Dag Wieers qui présentait le projet **ADEM** [5] dont le but est de mesurer la pollution par les particules fines (sujet on ne peut plus d'actualité depuis quelques mois en France et particulièrement à Paris). Dag nous a présenté le prototype basé sur un module Sparkfun ESP8266 monté sur une plaque d'expérimentation. Ce module est programmable avec l'IDE Arduino.

Steffen Evers (responsable des activités open source chez *Bosch Software Innovations*) nous a ensuite offert une conférence très intéressante sur les possibilités offertes par le monde du libre pour construire une plateforme IoT complète en se basant sur des composants de la fondation **Eclipse**.

Le sujet est totalement d'actualité, car le développement de ce marché conduit à l'apparition de nombreuses plateformes propriétaires proposant aux entreprises des chaînes complètes depuis le matériel jusqu'à l'hébergement des données. C'est à première vue très alléchant, mais également très dangereux si l'on considère l'enfermement provoqué par ces solutions. Il était donc très satisfaisant



Prototype du capteur ADEM

de constater qu'une grande entreprise comme Bosch s'intéresse à une voie basée sur le logiciel libre.

Jelle DeVleeschouwer évoqua ensuite le développement d'une implémentation **6LoWPAN** dans le contexte du projet **picoTCP**. Plusieurs conférences du FOSDEM évoquèrent ce protocole, car c'est aujourd'hui un standard dérivé de l'IPv6, mais largement moins gourmand en bande passante (et donc bien adapté aux objets connectés de faible puissance). 6LoWPAN est de ce fait disponible sur les OS dédiés à l'IoT comme **Contiki** ou **RIOT** et bien entendu sous **Linux**. Ce dernier sujet fit l'objet d'une conférence le lendemain dans la salle *Embedded, Mobile and Automotive*.

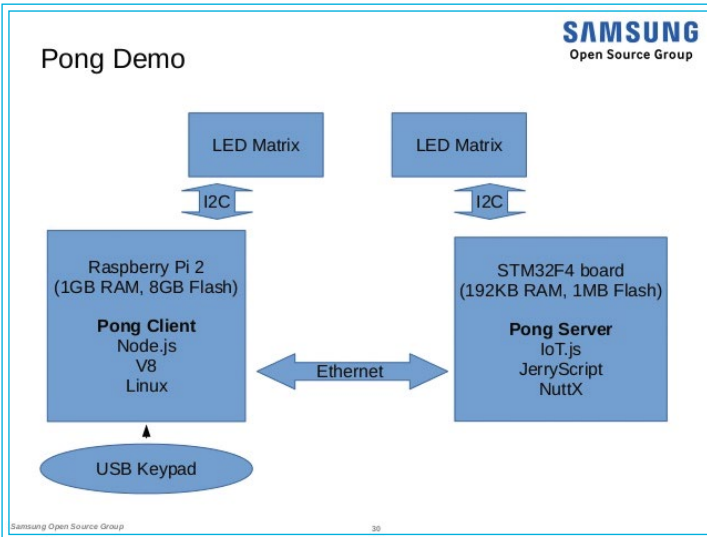
Afin de rester dans l'infiniment petit, Tilmann Scheller nous parla ensuite de **JerryScript**, une version de **JavaScript** adaptée aux microcontrôleurs et soutenue par Samsung. Le projet est fort intéressant, car ce langage largement répandu et très simple d'accès peut très bien avoir sa place dans des objets connectés. Durant la conférence, Tilmann nous présenta une comparaison de performances avec le concurrent **Duktape**. La démonstration, basée sur deux plateformes (une **Raspberry Pi** et une carte STM32), présentait un bon vieux jeu de « Pong » développé avec JerryScript et fonctionnant en mode client/serveur.

*metal* » (sans OS) dans 23 % des cas. Un premier exemple présentait un capteur de température basé sur une Raspberry Pi zero et un deuxième présentait une démonstration d'un réseau de capteurs de type **SensorTAG** (TI) connectés par le protocole 6LoWPAN. Dans ce dernier cas, une Raspberry Pi sous Yocto était utilisée comme « *border router* » grâce au logiciel libre **6LBR** [7].

À 15h il était plus que temps de quitter momentanément la salle (souvent bondée) et de tenter de trouver un peu de réconfort solide et liquide (une bière **Duvel** accompagnée de cubes de fromage et moutarde, *welcome to Belgium* !). Un bref passage sur les stands (au bâtiment K) démontra une fois de plus le succès de la manifestation, les stands étant quasiment inabornables.



La foule du bâtiment K



### Démonstration PONG avec JerryScript

Il était temps de m'atteler à la présentation de mon sujet consacré à l'utilisation de **Yocto** pour le prototypage IoT. Le sujet peut paraître scabreux, car Linux n'est pas forcément le système le mieux adapté à l'IoT de par son empreinte mémoire et les capacités matérielles nécessaires. Cependant, d'après la récente étude *IoT developer survey 2016* [6], Linux est utilisé dans 73 % des cas, suivi par l'approche « bare

metal » (sans OS) dans 23 % des cas. Un premier exemple présentait un capteur de température basé sur une Raspberry Pi zero et un deuxième présentait une démonstration d'un réseau de capteurs de type **SensorTAG** (TI) connectés par le protocole 6LoWPAN. Dans ce dernier cas, une Raspberry Pi sous Yocto était utilisée comme « *border router* » grâce au logiciel libre **6LBR** [7].

Un passage en salle UD2.120 nous permit d'assister aux trois dernières conférences *Embedded, Mobile and Automotive* en particulier sur les procédures de mise à jour des OS embarqués, mais surtout à l'excellente conférence consacrée à **AGL** (*Automotive Grade Linux*), alternative de la fondation Linux au projet **GENIVI** et bien entendu à **Android Auto**. La conférence de Dominig ar Foll (Intel) dura une heure et fut réellement passionnante. Dominig insista longuement sur la prise en compte des problèmes de sécurité dès le début du développement du projet AGL en isolant les applications de l'OS. Il évoqua également le futur d'AGL (AGL2++) qui



devrait utiliser un hyperviseur (restant à choisir) hébergeant dans des conteneurs des composants de niveaux de sécurité différents. Le concept le plus intéressant fut résumé dans la planche ci-dessous, i.e. « au niveau de la sécurité, surtout ne faites pas confiance à l'humain » ! Le problème est réel puisqu'il est clair que pour des millions de développeurs d'applications il existe très peu d'experts dédiés à la sécurité des systèmes embarqués.



AGL, *don't rely on human* !

Un stand AGL avec plusieurs démonstrations était également présent au bâtiment AW.

Une soirée était organisée autour de l'intérêt pour les systèmes embarqués pour l'automobile. Le lendemain fut consacré aux sujets sur l'embarqué sachant qu'il n'y avait plus de conférences IoT le dimanche. La première conférence par Luca Ceresoli intitulée « *How I survived to a SoC with a terrible Linux BSP* » était très instructive, car elle correspond à de (mauvaises) expériences réelles. Il est assez fréquent que des utilisateurs choisissent un matériel uniquement en fonction du prix sans se soucier suffisamment du support logiciel et donc de la qualité du BSP Linux. Luca évoqua l'exemple d'une carte à base d'un ancien CPU Samsung (S3C24xx) et utilisant un noyau 2.6 (sans aucun espoir d'évolution). La carte utilisait également un *bootloader* propriétaire (et non le célèbre **U-Boot**) ainsi qu'une procédure d'installation dédiée. On ne répétera jamais assez la nécessité d'utiliser du matériel « *mainline* » (ou qui s'en rapproche) sous peine d'être privé des évolutions développées par les communautés. Utiliser du logiciel libre sans s'appuyer sur les communautés est un non-sens, surtout pour les applications embarquées !

La conférence *Open Source Car Control* était présentée par Josh Hartung, conférencier très souriant et dynamique qui fit bien vivre son sujet. Le sujet présentait un système destiné à rendre un véhicule standard (l'exemple présenté était une **KIA**) en véhicule autonome en utilisant des modules **Arduino** (!) ce qui permet de réduire le coût du système à environ 10000 \$. Arnaud Taffanel présenta ensuite le projet **Loco Positioning** dédié au positionnement d'un robot (ici un drone de type **Crazyflie**) en intérieur. Le système présenté basé sur des capteurs (nommés *anchors* et utilisant un STM32) déployés sur le site permet de donner de bons résultats pour un coût largement inférieur à la technique de « motion capture ». La démonstration était impressionnante et fut très applaudie.

La conférence suivante par Lucas Bulwahn de **BMW** intitulée « *Success and Failure in Building an Open-Source Software Platform for Autonomous Driving Systems* » promettait beaucoup, mais fut un peu décevante. Le discours resta assez général peut-être à cause de problèmes de confidentialité internes à BMW Car IT. On sait effectivement qu'il y a parfois plusieurs possibilités dans le monde du libre et que la convergence des projets n'est pas toujours au rendez-vous, mais *quid* du cas concret des projets internes à BMW ? La fin de la conférence était cependant plutôt positive et ça fait toujours plaisir de voir évoquer **PREEMPT\_RT** sur des planches BMW !

Nous eûmes encore le temps d'assister à la conférence sur l'utilisation de 6LoWPAN (par Stefan Schmidt de Samsung) dans l'environnement Linux (soit le projet **Linux-wpan**). La conférence était très détaillée, un peu trop, car le conférencier ainsi que le maître de cérémonie oublièrent allègrement de regarder leur montre. Je fermis la marche juste après et il ne restait que 20 minutes sur les 30 accordées (d'autant que la majorité du public regagnait ses pénates), mais il n'en fallait pas plus pour faire un peu de publicité à **Xenomai** - outsider européen du projet temps-réel officiel de la fondation Linux (**PREEMPT\_RT**) - dans son utilisation pour le projet **openPOWERLINK**, bus industriel Ethernet open source désormais très répandu.

Deux sessions des *Developer rooms* portaient sur des thématiques proches du matériel : radio définie par logiciel (*Software Defined Radio*) et EDA (*Electronic Design Automation*) portant sur les outils de conception et simulation de circuits électroniques, puis configuration de FPGA.

La session sur la radio définie par logiciel était largement dominée par des présentations gravitant autour de GNU Radio, l'organisation par des membres de **Ettus Research** n'étant probablement pas étrangère aux choix des orateurs en ce sens. Bien que cette année cette thématique ait été abordée le samedi (et non le dimanche comme les années passées),

La salle était à nouveau comble avec nécessité de refuser des participants à chaque présentation, en particulier lors des présentations générales du matin. Parmi les faits marquants, le compte-rendu de Tom Rondeau sur le « hackfest » de la semaine passée portant sur la pollution du spectre électromagnétique par les chargeurs (de téléphone et d'ordinateurs portables), d'autant plus dramatique que le prix du périphérique baisse, et l'apothéose avec les chargeurs inductifs.



*La salle comble de la session radio logicielle, lors de la présentation introductive de GNU Radio de Marcus Muller, l'auteur de la plupart des réponses les plus techniques sur la liste de diffusion associée.*

Bien que conçus pour ne rayonner que localement, ces dispositifs de chargement sans fil propagent la puissance si inefficacement et les récepteurs radiofréquences sont si sensibles que leur effet pose question sur la disponibilité future des bandes radiofréquences associées pour les communications. L'utilisation des FPGA pour le traitement de signaux radiofréquences a eu la part belle, avec d'une part la présentation des pilotes et blocs FPGA de communication rapides (**JESD204B**), mais surtout l'introduction d'outils pour fournir le squelette de blocs de traitements déportant une partie du flux de traitement de **GNU Radio** dans le FPGA. En effet, l'outil **RFNOC**, largement plébiscité par Ettus Research, se cantonnait pour le moment à utiliser un *bitstream* fournissant un ensemble prédéfini de blocs de traitements dans lesquels les données étaient routées. Il est maintenant possible de synthétiser son propre *bitstream* avec les outils nécessaires à son application, mais surtout ajouter ses propres blocs – en **Verilog** – dans le *bitstream*.

Ainsi, même si le *bitstream* n'est pas généré dynamiquement en fonction du graphique proposant le flux de traitements dans **GNU Radio Companion** (ce qui est sûrement souhaitable compte tenu du temps de synthèse de Vivado), il est au moins possible de regrouper les outils nécessaires dans le *bitstream* transféré à la partie programmable du processeur. Une application ludique de Bastian Bloessl sur le décodage des transmissions radiofréquences entre feux de signalisation mobiles de

chantiers sert d'excellente introduction au novice désireux de découvrir le traitement numérique de signaux radiofréquences.



*Bastian Bloessl présente son étude du protocole de communication entre deux signalisations mobiles de chantier, un objectif simple pour s'initier au traitement des signaux numériques transmis sur porteuse radiofréquence.*

La discussion ouverte, occasion d'une pause déjeuner, a conclu sur la nécessité d'une documentation technique approfondie sur le traitement de signaux radiofréquences, et la radio définie par logiciel en particulier, s'appuyant sur GNU Radio pour engager le lecteur à expérimenter – écho à la présentation plénière de Tom Rondeau l'année dernière. En effet, la discussion a conclu que ce n'est pas le logiciel GNU Radio qui est complexe à appréhender, mais le domaine du traitement du signal échantillonné en temps discret en général. Prendre soin de propager les fréquences d'échantillonnage au fil des traitements et des décimations/interpolations, ou concevoir des filtres numériques respectant des contraintes de performance d'une part, tout en requérant des ressources de calcul raisonnables d'autre part (bande de transition de taille raisonnable) sont des problématiques indépendantes de l'implémentation de l'outil de traitement. Écrire un ouvrage autonome ou s'appuyant sur une des références du traitement numérique du signal (Proakis, Oppenheim, Lyons...) reste une question ouverte.

La fin de la session portait sur des projets plus spécialisés, par exemple des stations distribuées pour la réception de signaux satellitaires très haute fréquence (VHF), notamment issus des satellites étudiants ou universitaires. Le point traité par un des auteurs, la réception des signaux très basse fréquence émis par **DCF77** (Allemagne) pour la synchronisation d'horloges et la mesure des effets de l'ionosphère sur le temps de propagation, sera approfondi dans ces pages bientôt. La volonté des organisateurs pour l'année prochaine est de renouveler la liste des orateurs qui se répète significativement année après année. L'appel aux contributions est lancé.

La session EDA du dimanche a commencé sur une présentation de **GNUCap**, un moteur de résolution de circuits analogiques et numériques, suivi de **QUCS** (qu'on ne nommera pas la version libre de **Agilent ADS**) et évidemment **SPICE** (*Simulation Program with Integrated Circuit Emphasis*) et l'utilisation de la version sous forme de bibliothèque chargée dynamiquement de **ngspice** pour une simulation intégrée dans **KiCad**. Les perspectives sont tout à fait alléchantes, avec une simulation intégrée dans l'outil de dessin de schémas avec des perspectives éducatives intéressantes (capacité à dynamiquement modifier la valeur d'un composant et rafraîchir en temps réel la simulation), mais surtout la capacité d'exporter les schémas SPICE pour ensuite alimenter une simulation en ligne de commandes, fournissant une séquence cohérente partant de l'outil graphique intégré sans interdire à l'utilisateur expérimenté d'exploiter le simulateur efficacement en ligne de commandes. Ce point nous amène à KiCad, logiciel de conception de circuits électroniques qui a vu un développement accéléré depuis le soutien du **CERN** (organisation à laquelle l'organisateur de la session, Javier Serrano, appartient) [8]. Ce logiciel est devenu mature au bon moment, alors que le nouvel acquéreur de **Eagle (Autodesk)** a décidé de modifier la facturation de son logiciel propriétaire pour une licence à renouveler chaque année. Ce choix qui semble induire une fuite significative de ses utilisateurs vers Kicad a été noté lors de la session d'introduction à ce logiciel libre. Ici encore une salle comble, difficile d'entrer une fois la session quittée pour se restaurer.



*Les nouveautés de KiCad, avec notamment la simulation SPICE intégrée : les fonctionnalités de ce logiciel ont explosé récemment, en particulier avec le soutien du CERN pour son développement. L'organisateur, Javier Serrano, se tient debout à droite.*

L'organisation de la session portant sur la gestion de données géospatiales était quelque peu différente, avec par exemple une doctorante présentant les résultats de sa thèse sur l'assistance au choix d'un logement selon divers critères de distance de services (pour conclure que plus l'appartement est cher, meilleure est la probabilité de vivre dans un environnement considéré comme agréable). La gestion de données spatialisées porte

évidemment sur des domaines très vastes qui ne peuvent que difficilement tous intéresser un même auditeur, de la gestion de bases de données aux interfaces web et leur esthétique.

Toutes les présentations orales ont été enregistrées et validées par les équipes audio-vidéo des « *developer rooms* » ou par les auteurs eux-mêmes. Ces vidéos sont désormais, aux côtés des archives des années passées sur le site des vidéos du FOSDEM [9] indexées sur le numéro de salle dans laquelle s'était tenue chaque session.

## CONCLUSION

Une fois de plus le FOSDEM a su mettre en place une ambiance assez unique à la fois décontractée et d'un très bon niveau technique. Le geek solitaire peut côtoyer la multinationale durant les conférences ou sur les stands, ce qui est totalement dans l'esprit du logiciel libre. Une audience ouverte à discuter, avec de vrais échanges de fond, un modèle sur lequel les conférences habituellement qualifiées de scientifiques (et fort coûteuses) devraient s'inspirer. ■

## RÉFÉRENCES

- [1] Article *Libération* sur le FOSDEM : [http://www.liberation.fr/futurs/2017/02/06/logiciel-libre-a-la-conquete-du-grand-public\\_1546749](http://www.liberation.fr/futurs/2017/02/06/logiciel-libre-a-la-conquete-du-grand-public_1546749)
- [2] Le programme du FOSDEM : <https://fosdem.org/2017/schedule>
- [3] Pieter Hintjens : <http://hintjens.com>
- [4] Ouvrage « *Confessions of a Necromancer* » : <https://www.amazon.fr/Confessions-Necromancer-stories-Pieter-Hintjens/dp/1539178846>
- [5] Sparkfun ESP8266 : <https://www.sparkfun.com/products/13231>
- [6] IoT developer survey 2016 : <http://iot.ieee.org/images/files/pdf/iot-developer-survey-2016-report-final.pdf>
- [7] Projet 6LBR : <https://github.com/cetic/6lbr/wiki>
- [8] A. Del Rosso, KiCad software gets the CERN treatment (Fev. 2015) : <https://home.cern/about/updates/2015/02/kicad-software-gets-cern-treatment> et <http://kicad-pcb.org/>
- [9] Archive des vidéos du FOSDEM : <http://video.fosdem.org>

# TRANSFERT DE STYLE : ET SI VAN GOGH PEIGNAIT TUX ?

CÉLESTIN MATTE

[Doctorant INSA Lyon (financé par la région Rhône-Alpes)]

MOTS-CLÉS : NEURAL STYLE, STYLE TRANSFER, RÉSEAUX DE NEURONES, DEEP LEARNING



Le *neural style*, ou *style transfer*, a récemment fait son apparition, avec la publication d'un article en septembre 2015 [1]. Il émerge d'un contexte de fort développement des réseaux de neurones pour diverses applications, et notamment pour l'art. Quelques mois auparavant apparaissait le **deep dream**, programme faisant ressortir des *patterns* inexistantes dans des images, créant ce qui pourrait être considéré comme un style artistique à part entière.

Le neural-style permet de récupérer le « style » d'une image et de l'appliquer sur une autre. Cela permet, avec quasiment aucun effort, de copier le style d'un grand maître pour l'appliquer sur la photo de son chat. Perspective fort intéressante !

Cet article couvrira un peu de théorie, puis décrira pas à pas l'installation puis le transfert d'un style vers notre Tux chéri (figure 1).

## 1. UN PEU DE THÉORIE

Commençons par un peu de théorie : réseaux de neurones, *deep learning*, transfert de style et lien avec le *deep dreaming*.

Qu'est-ce que le style transfer ? Comment cela fonctionne-t-il ? Comment en faire chez soi ? Cet article est là pour répondre à toutes ces questions, et pour vous guider pas à pas sur un exemple.



Fig. 1 : Tux modifié en utilisant comme style une image glanée sur une banque d'images libre de droits [2].

## 1.1 Réseaux de neurones

Les réseaux de neurones sont une structure algorithmique copiant très schématiquement le fonctionnement de neurones dans le cerveau. Chaque neurone réalise une opération mathématique donnée, produisant ou non une sortie en fonction d'une valeur seuil qui lui est propre. Les réseaux peuvent avoir une immensité de topologies différentes, et les liens entre leurs neurones peuvent avoir des poids différents (appelés « poids synaptiques »).

Étant donnée la quantité de variables en jeu, la principale difficulté dans l'utilisation d'un réseau de neurones consiste en le choix de la valeur de ces variables. Différentes méthodes ont été testées pour apprendre automatiquement ces valeurs depuis l'apparition des réseaux de neurones dans les années 50. C'est l'efficacité des réseaux multi-couches (*deep learning* ou apprentissage profond) qui font le succès des réseaux de neurones depuis 2010.

## 1.2 Deep learning

L'idée du *deep learning* (apprentissage profond) est de structurer les tâches en couches reliées les uns aux autres, réalisant des opérations de niveaux d'abstraction différents. Par exemple, un réseau de reconnaissances d'images pourra être constitué d'une couche travaillant sur les pixels, reliée à une couche reconnaissant des bordures simples, elle-même reliée à une couche reconnaissant des motifs, puis des parties d'objets, puis des objets, etc.

Si le *deep learning* est aussi populaire en ce moment, c'est pour une raison simple : alors que l'on pensait jusqu'alors que l'avenir de l'intelligence artificielle passerait par l'enseignement d'heuristiques aux systèmes d'intelligence artificielle, on

s'est rendu compte qu'avec une structure appropriée et une puissance de calcul suffisante, de telles structures étaient capables de découvrir les heuristiques par elles-mêmes. Mieux, elles pouvaient le faire parfois mieux que les humains ! On a ainsi vu un tel système battre un des plus grands maîtres du jeu de go, alors qu'une telle perspective semblait éloignée d'encore au moins quelques dizaines d'années.

En tant que système multi-couches, le *deep learning* est particulièrement sujet à la diversité des topologies possibles. Pour le transfert de style, le principal réseau utilisé se nomme VGG (*Visual Geometry Group*). Il s'agit d'un réseau de 16 couches de neurones (ou 19, selon la version), connu pour obtenir de bons résultats en reconnaissance d'image.

## 1.3 Transfert de style

Dans un article publié en septembre 2015, des chercheurs de Tübingen et de Houston ont introduit un algorithme utilisant du *deep learning* pour créer des « images artistiques de haute qualité perceptuelle ». Leur article introduit l'idée que la représentation du style et du contenu peuvent être séparés dans un certain type de réseau de neurones. Cela a lancé la voie du transfert de style, rapidement étendu et amélioré par d'autres articles : gain de vitesse, application au son ou à la vidéo, etc.

Pour parvenir à cette prouesse, les auteurs de l'article construisent un réseau capturant les informations de texture d'une image, mais pas l'organisation des éléments de celle-ci. Une fois ces informations de texture mémorisées dans un réseau, il est possible de les appliquer sur une image différente.

Le transfert de style est un problème d'optimisation : on cherche à appliquer un modèle précalculé (le style) sur une

image. Pour cela, on définit une fonction objectif (*loss function*) qu'on cherche à minimiser. Il s'agit d'une somme pondérée de l'erreur (*loss*) entre l'image originale et l'image produite et de l'erreur entre le style original et celui appliqué. En jouant sur les paramètres de la pondération, on peut donc donner plus d'importance à l'image originale ou au style utilisé (cf. section 5).

## 1.4 Différences avec le deep dream

Le *deep dream* a été présenté dans un article de juillet 2015 par des chercheurs de Google. Antérieur à l'article précédent, il introduit l'idée de la génération d'images à but artistique par des réseaux de neurones profonds.

L'idée est ici plus simple : il s'agit de lancer « à l'envers » un réseau de neurones entraîné à reconnaître des images spécifiques. Le but est de produire en sortie l'image originale modifiée de telle sorte que l'on voie les endroits où le réseau « pense » reconnaître lesdites images. Cela produit des sortes d'hallucinations visuelles comparables à celles obtenues par un cerveau humain sous drogue psychédélique. On obtient ainsi une image étrange, où des formes inexistantes se dessinent sur les structures de l'image originale.

En faisant tourner l'algorithme plusieurs fois à partir de bruit, on obtient une représentation stylisée et combinée des objets que le réseau a été entraîné à reconnaître.

Le réseau de neurones utilisé pour l'article susnommé ayant été entraîné sur des images de chiens, de fractales colorées et de pagodes, on retrouve souvent des images hallucinantes contenant des chiens et des pagodes de façon colorée. Il est évidemment possible d'entraîner des réseaux sur de nouvelles images pour halluciner des choses différentes (pourquoi pas des tux ?).

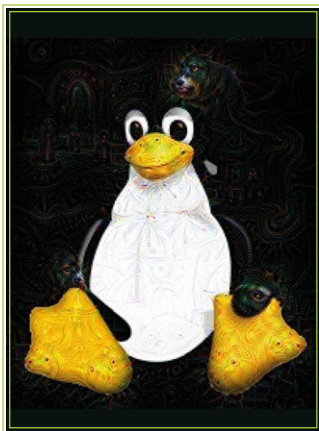


Fig. 2 : Tux modifié avec le *Deep Dream*.

La figure 2 montre le résultat d'un tux passé au filtre du *Deep Dreaming*.

Nous n'étudierons pas plus en détail le *deep dream* dans cet article, car il y a tellement de choses à faire avec (notamment en jouant avec les différentes couches) que cela pourrait constituer un article à part entière !

Notez que le transfert de style est parfois aussi appelé *deep dreaming*.

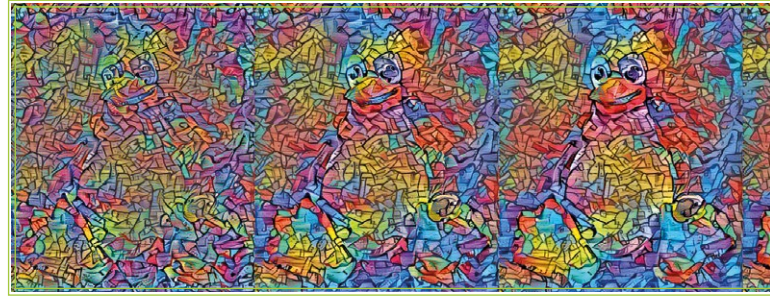


Fig. 3 : Les étapes (toutes les 100 itérations) de la génération de l'image finale.

## 2. OPTIONS D'UTILISATION

Plusieurs méthodes, plus ou moins simples et rapides, permettent de générer des images avec transfert de style. Passons-les en revue.

### 2.1 Demande à Claude

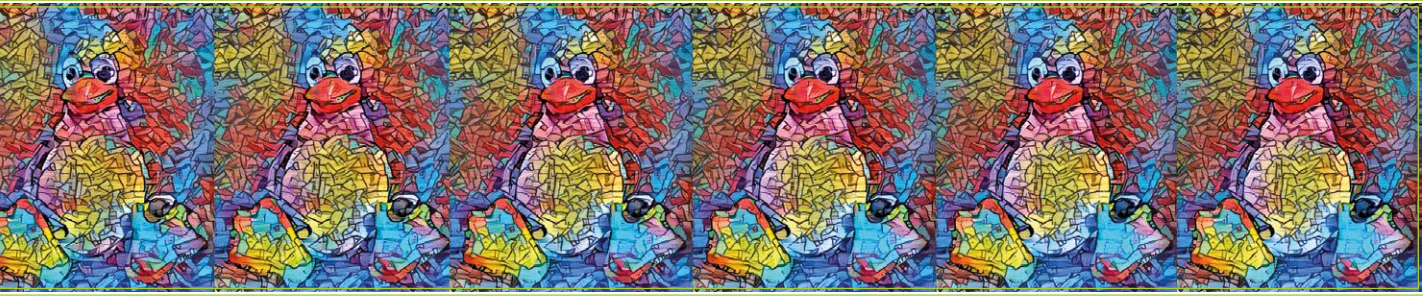
Passons rapidement sur la méthode la plus simple : utiliser le *cloud* (aussi appelé « l'ordinateur de quelqu'un d'autre »). Il existe de nombreuses pages web et applications pour smartphone permettant de générer des images en *neural-style*. L'avantage évident de tels systèmes est qu'on évite toute la partie installation (et éventuel achat de matériel nécessaire). L'inconvénient, c'est qu'on ne contrôle pas les paramètres, et que l'on n'apprend rien ! Dans cet article, nous allons donc nous attacher à tout installer nous-mêmes, mais sachez que des solutions directes existent.

Après quelques tests, je constate que leur qualité est très limitée : certains ne fonctionnent pas, beaucoup demandent une inscription, d'autres ne permettent pas de choisir le fichier de style, d'autres encore se veulent être des réseaux sociaux à part entière. À vrai dire, ceux qui fonctionnent correctement [3] sont basés sur la version rapide de l'algorithme (dont je discute les limites plus bas) et ne permettent pas de choisir le style.

### 2.2 Chez soi c'est quand même mieux

Voyons donc comment installer un des outils permettant de faire du style transfer : **neural-style**. Les explications qui suivent (ainsi que toutes les installations de l'article) sont prévues pour **Debian Stretch**.

Il faut commencer par installer **torch**, un *framework* en **lua** utilisé entre autres pour manipuler des réseaux de neurones. Il faut télécharger la version prévue pour l'installation (et non pas la version de développement) :



```
$ git clone https://github.com/torch/distro.git --recursive
$ ./install-deps # Utilise apt-get, vous pouvez aussi lire
le script et installer les paquets
# nécessaires a la main
$ ./install.sh
$ mkdir build && cd build
$ cmake ..
$ make
$ sudo make install
```

Le script d'installation vous demandera si vous souhaitez qu'il ajoute un script à votre `.bashrc`. Acceptez, il sert à exporter des variables d'environnement pour que torch puisse s'exécuter correctement.

Pour vérifier que l'installation de torch est correcte, il suffit d'utiliser la commande `th` et de voir si cela ouvre un shell interactif.

Maintenant que torch est activé, on peut installer le reste des dépendances de `neural-style` :

```
$ sudo aptitude install libprotobuf-dev protobuf-compiler
$ luarocks install loadcaffe
```

Pour la dernière commande, un bug vis-à-vis des versions de `protobuf` peut être contourné en l'exécutant en root.

Ensuite, on clone le dépôt et on télécharge le modèle VGG, c'est-à-dire le schéma des réseaux de neurones. Celui-ci a le format des réseaux `Caffe`, un autre *framework* de *deep learning*.

```
$ git clone https://github.com/jcjohnson/neural-style.git
$ sh models/download_models.sh
```

Il ne manque qu'une chose : les images à utiliser ! Pour notre exemple, j'ai pris les images [4] et [5]. Voilà, normalement tout est bon pour lancer un calcul :

```
$ th neural_style.lua -style_image PB_20140912201212155.
jpg -content_image tux.png -output_image tux_robots.jpg
-print_iter 1 -gpu -1
```

Les options utilisées parlent d'elles-mêmes :

- `-style_image` : le chemin vers l'image dont on veut copier le style ;
- `-content_image` : le chemin vers l'image sur laquelle appliquer le style ;

- `-output_image` : le nom du fichier de sortie. Notez que l'extension que vous indiquez sera utilisée pour produire l'image dans le format correspondant ;
- `-print_iter` : pour afficher une sortie à chaque itération (il y en a **1000** par défaut) ;
- `-gpu -1` : pour ne pas utiliser de GPU (nous allons voir cela dans les sections suivantes).

Au cours du calcul, le script enregistre les images intermédiaires toutes les 100 itérations, ce qui permet de se rendre compte du fonctionnement du système (figure 3). On peut aussi les utiliser pour créer un GIF animé !

## 3. ACCÉLÉRATION

Bon voilà, c'est bien joli tout cela, mais notre calcul précédent prend plus ou moins... 10 heures ! Il va falloir étudier les moyens disponibles pour l'accélérer.

### 3.1 GPU

Les GPU sont très utiles pour accélérer des calculs de réseaux de neurones, ceux-ci bénéficiant particulièrement du calcul massivement parallèle.

#### 3.1.1 Choix de la carte

Pour le choix de la carte, notez que si les cartes **Nvidia** ont un meilleur rapport qualité/prix, les pilotes libres ne permettent

pas d'utiliser **CUDA** (mais d'autres technologies équivalentes moins performantes). Privilégiez une carte particulièrement performante pour les calculs de basse précision, ceux-ci étant massivement utilisés dans les réseaux de neurones.

Il faut de plus installer un *backend* permettant le calcul parallèle : **CUDA** ou **OpenCL**. Le backend CUDA étant plus utilisé par la communauté, il est plus abouti et plus efficace en terme de calcul. Le *backend* OpenCL, quant à lui, a l'avantage d'être en licence libre (et donc plus facile à installer). Pour les cartes gérant CUDA (Nvidia), l'utilisation additionnelle du *backend* **cuDNN** permet un léger gain de performance, et surtout une meilleure gestion de la mémoire (ce qui permet de générer des images plus grandes).

Une carte graphique milieu de gamme (environ 300€) est adaptée pour générer des images de taille 512x512, mais sera quelque peu limitée pour aller au-delà (jusqu'à 800-900 pour une carte avec 6 Gio de mémoire, en fonction des images). Le facteur limitant est ici la mémoire de la carte, sachant que le swap n'est généralement pas possible en cas d'utilisation totale de cette mémoire (peu de cartes le gèrent). La consommation mémoire de l'outil est assez importante : 3.5 Gio avec les paramètres par défaut, 1 Gio en utilisant un *backend* moins consommateur de mémoire. La carte graphique utilisée doit en posséder au moins autant, puisqu'une insuffisance en mémoire rendra l'exécution impossible.

### 3.1.2 Installation

Pour faire du calcul sur GPU, quelques installations supplémentaires sont nécessaires, et dépendent de la carte choisie et du *backend* désiré.

Pour OpenCL, l'installation est a priori simple, mais je ne l'ai pas testée. Il faut d'abord installer les drivers

correspondant à la carte graphique (qui dépendent du constructeur, cf. [6]), puis installer **cltorch** et **clnn** :

```
$ git clone --recursive https://github.com/hughperkins/distro -b distro-cl ~/torch-cl
$ cd ~/torch-cl
$ ./install-deps
$ ./install.sh
```

Dans le cas d'une carte Nvidia, installer CUDA et cuDNN est plus compliqué. Le module du noyau nécessite que les *headers* du noyau utilisé soient installés. On commence donc par les installer, avant de faire de même pour tous les paquets nécessaires. Les paquets lua **cutorch** et **cunn** ne compilent qu'avec des versions de gcc inférieures à 5. Il faut donc passer des *flags* pour que les bonnes versions soient utilisées (vérifiez vos versions de gcc et g++ avant cela). En résumé :

```
$ sudo aptitude install linux-headers-$(uname -r)
$ sudo aptitude install nvidia-driver nvidia-kernel-dkms
nvidia-cuda-toolkit nvidia-cuda-dev
$ CC=gcc-4.8 CXX=g++-4.8 luarocks install cutorch
$ CC=gcc-4.8 CXX=g++-4.8 luarocks install cunn
```

Toujours dans le cas d'une carte Nvidia, on peut optionnellement rajouter cuDNN, qui permet un léger gain de performances, et surtout une meilleure utilisation mémoire. cuDNN n'est pas *packagé* sur Debian, il faut le télécharger sur le site de Nvidia [7], après enregistrement. On l'installe ensuite comme un paquet normal, puis on installe la bibliothèque lua associée :

```
$ sudo dpkg -i 'libcudnn5_5.1.5-1+cuda8.0_amd64.deb'
$ CC=gcc-4.8 CXX=g++-4.8 luarocks install cudnn
```

Si tout est bon après un redémarrage, on peut désormais lancer *neural-style* avec l'option **gpu** (0 signifie d'utiliser le GPU numéro 0, pas de la désactiver), avec cuDNN si on l'a installé :

```
$ th neural_style.lua -style_image PB_20140912201212155.jpg
-content_image tux.png -output_image \ tux_robots.jpg
-print_iter 1 -gpu 0 -backend cudnn
```

Ou en utilisant OpenCL :

```
$ th neural_style.lua -style_image PB_20140912201212155.jpg
-content_image tux.png -output_image \ tux_robots.jpg
-print_iter 1 -gpu 0 -backend clnn
```

Et voilà, le calcul ne prend plus que quelques minutes !

### 3.2 TPU

Les réseaux de neurones utilisés dans le *style transfer* ont la particularité de nécessiter uniquement du calcul sur petits entiers (8 bits). Si les GPU possèdent de nombreux cœurs, ceux-ci sont inutilement puissants pour une telle application. Pour cette raison, Google a développé des puces spécialement dédiées à ce genre de calcul, appelées TPU (*Tensor Processing Unit*), et les utilise depuis un an dans ses *datacenters*. Ces puces permettraient un énorme gain de vitesse et de performance par watt.



# Professionnels, Collectivités, R & D...



M'abonner ?

Choisir le papier,  
le PDF, la base  
documentaire,  
ou les trois ?

Me réabonner ?

Permettre à mes équipes  
de lire les magazines en  
PDF, consulter la base  
documentaire ?

*C'est possible ! Rendez-vous sur :*

**<http://proboutique.ed-diamond.com>**

*pour consulter les offres !*

N'hésitez pas à nous contacter pour un devis personnalisé par e-mail :

**[abopro@ed-diamond.com](mailto:abopro@ed-diamond.com)** ou par téléphone : **+33 (0)3 67 10 00 20**



Et là vous vous dites : « Mais c'est super ! Ça se trouve où ? ». Malheureusement, ces puces ne sont pas disponibles dans le commerce, et Google n'a pas annoncé vouloir les commercialiser.

Pour être exhaustif, notons qu'il existe d'autres processeurs destinés à accélérer les outils de *machine learning* en se spécialisant dans les opérations mathématiques de basse précision : FPGA, d'autres types d'ASIC... [8]

### 3.3 Fast neural network

Il existe une implémentation plus récente du *style transfer* permettant un calcul bien plus rapide. Celle-ci est basée sur un article d'octobre 2016 [9]. L'idée est de remplacer l'erreur (*loss*) « par pixel » par l'erreur « perceptuelle ». En d'autres termes, le système ne cherche plus à faire coller chaque pixel à l'image originale, mais calcule la perte à un niveau plus élevé, ce qui permet de mesurer plus finement la similitude « perçue » entre deux images. De plus, l'implémentation utilise une modification dans l'architecture du réseau qui permet également un gain de performance. Celle-ci se base sur un précalcul du modèle, qui peut alors être utilisé à faible coût pour être appliqué sur de nouvelles images.

Au niveau pratique, les modèles doivent donc être précalculés afin de pouvoir être utilisés. Une fois le projet précédent installé, il n'y a rien de nouveau à installer pour créer de nouvelles images en utilisant les modèles préinstallés (en fait, il y a même moins de dépendances). Il n'y a donc qu'à cloner le dépôt :

```
$ git clone https://github.com/jcjohnson/fast-neural-style
```

Vous pouvez à présent créer de nouvelles images avec les modèles existants :

```
$ th fast_neural_style.lua -model models/eccv16/starry_night.t7 -input_image tux.png -output_image tux-starry_night.png
```

Notez qu'avec cette implémentation, la création d'images utilisant des modèles existants ne prend plus qu'une minute sur CPU (et quelques secondes sur GPU !). L'entraînement de nouveaux modèles reste cependant extrêmement coûteux : 4 à 6 heures sur une carte graphique haut de gamme (on passe de 6 à 4,5h avec cuDNN). Il n'est donc pas vraiment envisageable d'entraîner de nouveaux modèles sur CPU...

Justement, voyons maintenant l'entraînement de nouveaux modèles. Pour cela, il y a encore 2-3 choses à installer : **torch-hdf5** et **python-h5py**.

```
$ git clone https://github.com/deepmind/torch-hdf5
$ cd torch-hdf5
$ luarocks install hdf5-0-0.rockspec
$ sudo aptitude install python-h5py
```

Pour entraîner un réseau, il faut de plus le faire sur une banque d'images. Il faut donc un tel jeu d'images, et le préparer pour créer un fichier **.hdf5**, qui contient des informations sur le contenu des images. On va utiliser le jeu de données **COCO**, une grosse base de données d'images variées. Celui-ci est composé d'un jeu d'entraînement de 13 Gio ainsi que d'un jeu de vérification de 6 Gio. Ce dernier sert à effectuer des passes de vérification du bon fonctionnement de l'entraînement du réseau sur des images différentes de celles du jeu d'entraînement.

```
$ wget http://msvocds.blob.core.windows.net/coco2014/train2014.zip
$ wget http://msvocds.blob.core.windows.net/coco2014/val2014.zip
$ unzip train2014
$ unzip val2014
$ python scripts/make_style_dataset.py
--train_dir train2014 --val_dir val2014
--output_file file.h5
```

Notez que l'exécution du script prend quelques heures. Il est a priori possible d'obtenir des résultats très similaires avec un sous-ensemble très réduit du jeu de données, mais cette étape n'est à exécuter qu'une seule fois de toute manière, et l'étape d'entraînement n'est pas accélérée par l'utilisation d'un jeu d'images plus petit.

Comme pour *neural-style*, on a besoin du réseau VGG pour entraîner notre modèle :

```
$ ./models/download_vgg16.sh
```

Si tout est bon, on peut commencer à entraîner notre nouveau modèle :

```
$ th train.lua -h5 file normal.hdf5
-style_image PB_20140912201212155.jpg
-gpu 0 -use_cudnn 1
```

Si, comme c'est le cas à l'heure où j'écris ces lignes, vous obtenez cette erreur :

```
Error: unable to locate HDF5 header file
at hdf5.h
```

Voici une manière de la corriger :

- installez le paquet **libhdf5-serial-dev** ;
- modifiez le fichier **hdf5-0-0.rockspec** pour ajouter le dossier de localisation de **hdf5.h** à la commande **cmake**. Concrètement, remplacez la ligne :



Fig. 4 : Résultat obtenu avec fast-neural-style.



Fig. 5 : Une autre image avec fast-neural-style.



Fig. 6 : L'équivalent de l'image de la figure 5 avec le neural-style original.

```
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_PREFIX_PATH='$(LUA_BINDIR)/..' -DCMAKE_INSTALL_PREFIX="$(PREFIX)';
```

par :

```
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_PREFIX_PATH='$(LUA_BINDIR)/..' -DCMAKE_INSTALL_PREFIX="$(PREFIX)'" -DHDF5_INCLUDE_DIR='/usr/include/hdf5/serial/';
```

Puis relancez la commande d'installation :

```
$ luarock install hdf5-0-0.rockspec
```

On obtient le résultat visible en figure 4.

Comme je le disais plus haut, le résultat est bien moins intéressant qu'avec la version lente de l'algorithme. Cependant, l'image Tux est très « lisse » et « simple ». On obtient des résultats bien plus intéressants pour d'autres images : cf. figure 5 par exemple. Ce dernier est tout de même à mon goût moins intéressant que son équivalent neural-style (figure 6).

## 4. QUELQUES CONSEILS ARTISTIQUES

Après quelques utilisations sur diverses images, voici les conseils que je peux donner pour obtenir des résultats intéressants :

- Parfois, les fichiers d'étapes (cf. figure 3) sont plus intéressants que le résultat final (notamment les 200 premières itérations). Pensez aussi à essayer

d'augmenter le nombre d'itérations si l'image finale n'est pas « terminée » (c'est-à-dire si les images d'étapes changent encore beaucoup dans les dernières centaines d'itérations).

- Pour les styles, choisir des images de styles avec de nombreux détails, ou un style bien défini. Prendre un simple logo donne des résultats inintéressants (utiliser Tux ou le logo Debian comme image de style ne donne rien de bien).
- Par défaut, les images sont générées avec une taille de 512 pixels pour son côté le plus grand. Cela donne des images assez petites, et surtout pixelisées. Agrandir l'image de résultat (avec l'option **-image\_size**) augmente le niveau de détails en même temps que la netteté (pensez à l'option **style\_scale** pour remédier à cela). Comme indiqué plus haut, il est cependant difficile d'augmenter beaucoup les



Fig. 7 : Résultats obtenus en faisant varier l'option **style\_scale** avec les valeurs suivantes : **0.1, 0.2, 0.5, 0.8, 1.5, 2.0** (valeur par défaut : **1.0**). Les valeurs supérieures demandent trop de mémoire pour ma carte graphique de 6 Gio.

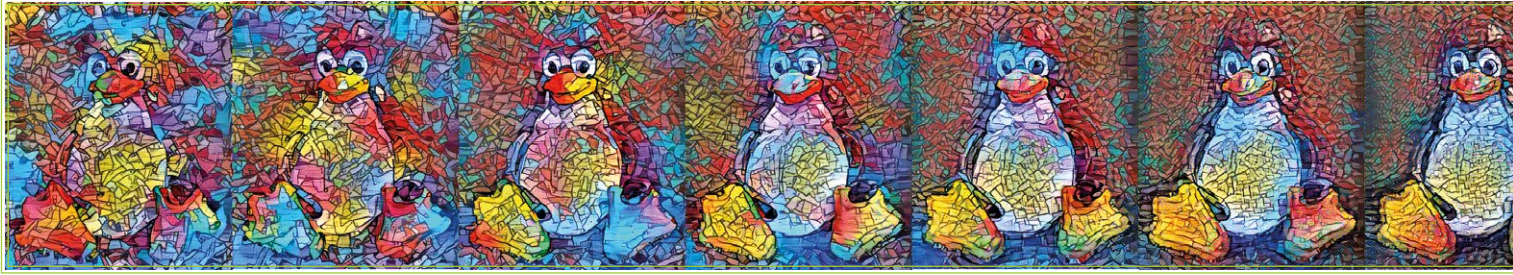


Fig. 8 : Résultats obtenus en faisant varier l'option **content\_weight** avec les valeurs suivantes : 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000 (valeur par défaut : 5).

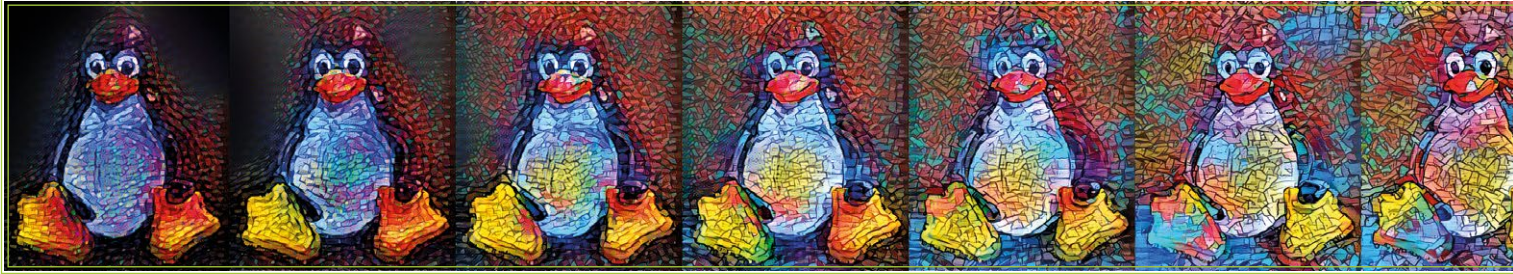


Fig. 9 : Résultats obtenus en faisant varier l'option **style\_weight** avec les valeurs suivantes : 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000 (valeur par défaut : 100).

tailles des images générées, puisque l'utilisation mémoire augmente de façon quadratique. La mémoire de la carte graphique est donc rapidement pleine, à moins d'en avoir plusieurs... Quant à l'option du calcul sur CPU (pour bénéficier d'une mémoire plus grande et de la possibilité de swaper), il est extrêmement lent : il m'a fallu 46h de calcul pour générer une image de 1024x682 pixels !

- Notez qu'il est possible d'utiliser plusieurs styles à la fois avec **neural-style**, ce qui peut donner des résultats intéressants. Avec **fast-neural-style**, ce n'est pas possible.
- L'outil **fast-neural-style**, s'il est plus rapide, donne des résultats moins intéressants. Comme le notent les auteurs de l'article, cette solution a tendance à produire des motifs répétitifs. À mon sens, le résultat est vraiment moins pertinent.



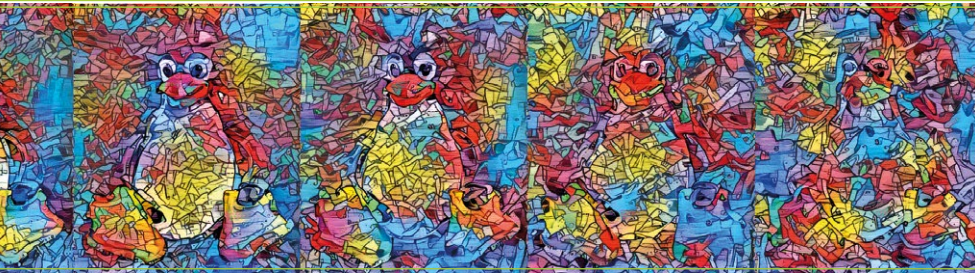
Fig. 10 : Résultats obtenus en faisant varier l'option **-init image** (à gauche) au lieu de **-init random** (à droite).

Il est possible de jouer avec les paramètres des outils. Étudions-en quelques-uns :

- L'option **style\_scale** permet d'appliquer le style avec une échelle plus ou moins grande. Varier ce paramètre peut donner une meilleure intégration du style dans le résultat final (voir figure 7, page précédente).
- Les options **content\_weight** et **style\_weight** permettent de donner plus ou moins de poids au style ou à l'image originale dans la construction de l'image finale. Leurs valeurs par défaut sont respectivement 5 et 100. Les figures 8 et 9 montrent les résultats obtenus en faisant varier ces paramètres.
- Ne pas hésiter à jouer avec tous les autres paramètres ! Les outils présentés en possèdent de nombreux, qu'il serait long de passer en détail. Par exemple, l'option **-init image** permet de démarrer le calcul à partir de l'image d'origine, plutôt que d'un bruit aléatoire. Cela donne un résultat moins déconstruit, plus proche de l'image originale (cf. figure 10).

## 5. POUR ALLER PLUS LOIN : LA VIDÉO

Les images c'est bien. Quand elles bougent, c'est mieux ! Il existe plusieurs techniques différentes pour passer de l'image à la vidéo. Voyons-en une rapide, et une autre orientée vers un meilleur résultat.



calculent en plus le flux optique, c'est-à-dire le mouvement apparent des objets prenant en compte le mouvement relatif de la caméra. Cela donne un résultat stable, bien qu'encore imparfait (on trouve entre autres des artefacts étranges dans le résultat et il reste des soucis de cohérence). Notez que cet algorithme utilise **neural-style** et non **fast-neural-style**, et est donc lent : j'ai mis environ 7 heures pour générer une vidéo de 10 secondes en 320x200.

Si vous avez déjà installé les dépendances des outils précédents (torch7, loadcaffe et CUDA), il n'y a plus qu'à cloner le dépôt et à télécharger deux outils, disponibles directement sous la forme de binaires. Ceux-ci servent à calculer le flux optique entre deux images.

## 5.1 En temps réel

L'outil précédent (**fast-neural-style**) est tellement rapide qu'il permet une exécution en temps réel sur une entrée vidéo. Et justement, il comprend un script permettant de faire cela. Pour l'utiliser, il faut installer quelques dépendances supplémentaires :

```
$ sudo aptitude install libopencv_highgui
$ luarocks install camera
$ luarocks install qtlua
```

Pour lancer le script, il faut utiliser **qlua** et non plus **th** :

```
$ qlua webcam_demo.lua -gpu 0 -models
models/model.t7
```

Le script s'occupe de chercher une entrée vidéo (webcam) et d'y appliquer le transfert de style en temps réel. Et voilà !

## 5.2 Stabilisée

Vous remarquerez cependant rapidement un défaut avec le script précédent : le style n'est pas appliqué identiquement à chaque image, ce qui donne une sorte de clignotement assez désagréable. Cela est dû au fait qu'étant initialisée avec des valeurs aléatoires, chaque image va converger vers un maximum local différent. Un article d'octobre 2016 [10] par des chercheurs de l'université de Freiburg a étudié la question pour corriger ce problème. Pour simplifier, ses auteurs proposent d'initialiser chaque image avec les valeurs de l'image précédente afin de conserver une certaine uniformité, et

```
$ git clone https://github.com/manuelrueder/
artistic-videos
$ cd artistic-videos
$ wget lear.inrialpes.fr/src/deepmatching/code/
deepmatching_1.2.2.zip
$ unzip deepmatching_1.2.2.zip
$ mv deepmatching_1.2.2_c++/deepmatching-static .
$ wget pascal.inrialpes.fr/data2/deepmatching/
files/DeepFlow_release2.0.tar.gz
$ tar xzf DeepFlow_release2.0.tar.gz
$ mv DeepFlow_release2.0/deepflow2-static .
```

Un script existe pour lancer l'outil avec les options par défaut :

```
$ ./stylizeVideo.sh <video> <image_style>
```

Si on veut utiliser d'autres options pour le transfert de style, il faut décomposer le script : celui-ci découpe une vidéo en images, calcule le flux optique, lance le transfert de style sur chaque image, puis reconstruit la vidéo à partir des images générées. En bref :

```
$ mkdir tmp
$ ffmpeg -i <video> tmp/frame_%04d.ppm
$ makeOptFlow.sh ./tmp/frame_%04d.ppm ./tmp/
flow $resolution
$ th artistic_video.lua -content_pattern
tmp/%03d.ppm -flow_pattern \
tmp/flow_default/backward_[%d]_{%d}.flo
-flowWeight_pattern \
tmp/flow_default/reliable_[%d]_{%d}.pgm
-output_folder tmp -style image <style> -gpu 0 \
-backend cudnn -number_format %03d
$ ffmpeg -i tmp/out-%04d.png video-stylized.mp4
```

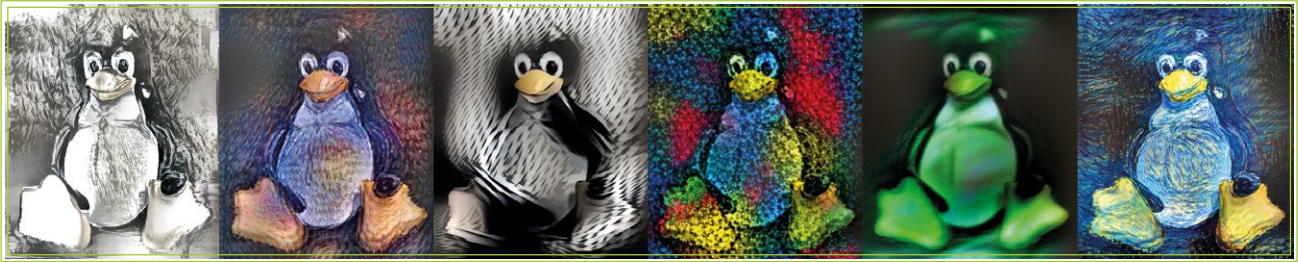


Fig. 11 : D'autres résultats avec des images de style différentes.

Attention : comme auparavant, on se retrouve rapidement à court de mémoire si on utilise des images trop grandes. Il sera certainement nécessaire de réduire la taille des images lors de la conversion de la vidéo en images avec l'option **-vf scale=<résolution>** de **ffmpeg**.

## 6. FILE AU ZOO, FI !

Le développement du *machine learning* à des fins artistiques pose de nouveaux questionnements philosophiques. S'il y a quelques années, il était encore courant de penser que l'art resterait pour longtemps du domaine de l'humain, inaccessible par la machine, la publication du *deep dream* par des chercheurs de Google a balayé cette supposition d'un revers de main : une intelligence artificielle aurait créé un nouveau style d'art ! Évidemment, cette affirmation est à relativiser largement puisque c'est avant tout les chercheurs et non la machine qui ont créé cet art. Le *style transfer* va encore plus loin : une machine est désormais capable de copier le style d'un humain sur une nouvelle image. Incroyable !

La barrière est donc repoussée. On peut se reconforter en se disant que l'art consiste avant tout à créer quelque chose de nouveau en fonction du contexte, qui sache perturber, questionner et émerveiller le spectateur. Et ça, la machine en est toujours incapable, puisqu'elle doit nécessairement être guidée par un être humain. Pour l'instant. Mais pour combien de temps ? Le défi est lancé !

## L'IMAGE DE LA FIN

En note finale, remarquons que si le transfert de style fonctionne pour les images, rien ne l'empêche d'être possible pour d'autres arts, notamment la musique [11] et la danse [12]. À vos claviers !

Et pour finir, voici quelques images de tux modifié en utilisant quelques images et photos glanées sur divers sites proposant des images libres de droits. Imaginez ce qu'il est possible de faire en ne se limitant pas pour des questions de droit ! ■

## RÉFÉRENCES

- [1] GATYS L., ECKER A. et BETHGE M., « A neural algorithm of Artistic style », 2015 : <https://arxiv.org/pdf/1508.06576v2.pdf>
- [2] Photo de Jeffrey Betts : <http://finda.photo/image/11418>
- [3] Application Aristo : <https://artisto.my.com/>
- [4] Photo de Jeffrey Betts : <http://finda.photo/image/11418>
- [5] Image de Tux : <https://commons.wikimedia.org/wiki/File:Tux.png>
- [6] Installation des drivers des cartes graphiques sous Debian : <https://wiki.debian.org/fr/GraphicsCard>
- [7] cuDNN : <https://developer.nvidia.com/rdp/cudnn-download>
- [8] Accélération des outils de machine learning : [https://en.wikipedia.org/wiki/AI\\_accelerator](https://en.wikipedia.org/wiki/AI_accelerator)
- [9] JOHNSON J., ALAHI A. et LI F.-F., « Perceptual losses for real-time style transfer and super-resolution », ECCV 2016 : <http://cs.stanford.edu/people/jcjohns/eccv16/>
- [10] RUDER M., DOSOVITSKIY A. et BROX T., « Artistic style transfer for videos », 2016 : <https://arxiv.org/pdf/1604.08610.pdf>
- [11] *Algorithmic Music Generation with Recurrent Neural Networks* : <https://www.youtube.com/watch?v=0VTI1BBLyDE>
- [12] *Generative Choreography using Deep Learning* : <http://peltarion.com/creative-ai>

# NE MANQUEZ PAS LA NOUVELLE FORMULE!

## LINUX PRATIQUE N°100

**NOUVELLE FORMULE : NOUVELLES RUBRIQUES, ENCORE + PRATIQUE !**

**GNU LINUX PRATIQUE**  
SUR PC, MAC ET RASPBERRY PI

**MARS AVRIL 2017**

**SOLUTION PRO**  
Renforcez la sécurité de vos données : utilisez Vault pour gérer vos secrets p. 66

**DÉBUTANTS RASPBERRY PI & LINUX**  
Réalisez vos premiers développements sur Raspberry Pi avec l'éditeur Geany p. 92

**CONTRÔLEZ L'ACCÈS AU WEB !**  
CONTRÔLE PARENTAL, CHARTE INFORMATIQUE, BORNE PUBLIQUE, POINT D'ACCÈS WIFI... p. 50

**GRAPHISME**  
Impressionnez votre auditoire avec des présentations animées grâce à Sozi p. 14

**SYSTÈME**  
Personnalisez votre clavier multimédia à l'aide de XBindKeys p. 34

**CONCEPTION**  
Réalisez vos schémas électroniques et vos montages avec Fritzing p. 22

**PROGRAMMATION**  
Programmez où que vous soyez avec l'environnement Codiad p. 44

**WEB**  
8 conseils pour optimiser votre landing page et améliorer vos conversions p. 58

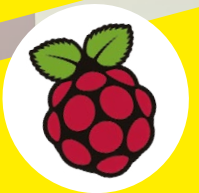
**TUTORIELS**

**NOUVEAU : UN CAHIER RASPBERRY PI DANS CHAQUE NUMÉRO !**



**NOUVELLES RUBRIQUES, ENCORE + PRATIQUE !**

**NOUVEAU : UN CAHIER RASPBERRY PI DANS CHAQUE NUMÉRO !**



# CONTRÔLEZ L'ACCÈS AU WEB !

**ACTUELLEMENT DISPONIBLE CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :**  
<http://www.ed-diamond.com>



# FACILITEZ-VOUS LA VIE AVEC LA LIGNE DE COMMANDES

KEVIN DENIS

[Open source and security enthusiast]

MOTS-CLÉS : SHELL, BASH, CLI, RACCOURCIS CLAVIER, KISS



**Si vous utilisez GNU/Linux, alors vous utilisez forcément le shell. Que vous soyez développeur chevronné, administrateur gourou ou utilisateur éclairé, vous passez un temps incroyable devant des lignes de commandes. Cet article donne quelques trucs et astuces pour exploiter encore plus efficacement ce shell.**

**N**ous avons tous de nombreuses raisons de préférer un système d'exploitation à un autre, un environnement graphique à un autre. Mais quelle que soit votre distribution, votre manière de travailler, vous finirez invariablement par manipuler la ligne de commandes, autrement dit le shell.

Cet article va parler de plusieurs petits trucs et astuces qui rendent la vie plus aisée sous la ligne de commandes. Tout le monde a ses « trucs », certains sont appris dès la première utilisation du shell, comme le rappel de commande ou l'édition. Cet article va donner quelques raccourcis clavier, méthodes ou options un peu moins connues, mais très pratiques.

La première partie décrit quelques raccourcis clavier sous bash, suivis en deuxième partie de quelques *builtins* bien utiles qui peuvent faire gagner énormément de temps et de concision dans l'appel de commandes. La troisième partie évoquera quelques petits outils généralement installés par défaut avec vos distributions qui gagnent à être connus. La dernière partie donne des options un peu moins classiques de commandes que tout le monde connaît.



# 1. LES RACCOURCIS CLAVIER DE BASH

bash est le shell (ou interpréteur de commandes) qui est installé par défaut sur la très grande majorité des distributions GNU/Linux. Vous utilisez certainement déjà des raccourcis clavier, comme **<Flèche haut>** pour rappeler la commande précédente ou **<Ctrl> + <c>** pour arrêter la commande en cours (*break*). Voici une petite sélection de commandes sans doute moins connues, mais très pratiques.

Les raccourcis sont définis par l'appui sur une touche, un signe plus et la seconde touche sur laquelle appuyer. Ainsi, **<Ctrl> + <c>** demande d'appuyer sur **<Ctrl>**, puis en laissant **<Ctrl>** enfoncé, d'appuyer sur la touche **<c>** du clavier. Si des enchaînements sont à faire, ils sont clairement indiqués.

## 1.1 Navigation dans la ligne de commandes

bash permet de naviguer simplement entre le début et la fin de la ligne de commandes avec **<Ctrl> + <a>** (début de ligne) et **<Ctrl> + <e>** (fin de ligne, 'e' pour *end*).

## 1.2 Utilisation des arguments de la commande précédente

**<Alt> + <.>** a un comportement intéressant : ce raccourci réécrit le dernier argument de la commande précédente. Je donne un exemple :

```
$ mkdir dossier
$ cd (appui sur <Alt> + <.>)
```

La dernière ligne devient alors :

```
kevin@slackware:/tmp$ cd dossier
```

Un second appui sur **<Alt> + <.>** donne le dernier argument de la deuxième commande précédant la ligne en cours, etc.

## 1.3 Recherche dans l'historique des commandes

Tout le monde connaît le rappel de commandes avec la flèche haut. Mais il existe une version plus puissante avec **<Ctrl> + <r>** qui cherche dynamiquement dans l'historique de commandes parmi les caractères entrés au clavier. Par exemple, supposons que nous ayons dans notre historique de commandes des appels à **tcpdump** :

```
$ history | grep tcpdump
1518 sudo tcpdump -n -i vboxnet0 tcp port 80
1519 sudo tcpdump -n -i vboxnet0 ! tcp port 3389
1690 sudo tcpdump -n -i wlan0
2006 history | grep tcpdump
```

Les retrouver avec **<Flèche haut>** demanderait un nombre d'appuis relativement élevé, mais avec **<Ctrl> + <r>** nous pouvons directement chercher la commande :

```
$ # appui sur <Ctrl> + <r>
(reverse-i-search) '': # apparition du prompt de recherche
(reverse-i-search) 'tcp': sudo tcpdump -n -i wlan0 # après avoir tapé tcp
(reverse-i-search) 'tcp': sudo tcpdump -n -i vboxnet0 ! tcp port 3389 # appui sur <Ctrl> + <r> de nouveau pour remonter dans l'historique à la précédente occurrence de tcp
```

## 1.4 Effacement arrière

**<Alt> + <Backspace>** permet d'effacer depuis la position du curseur vers l'arrière jusqu'à la précédente césure de mot. Un exemple étant plus parlant :

```
$ ma_commande monoption monarg<curseur>ument #
appui sur <Alt> + <Backspace>
$ ma_commande monoption <curseur>ument
```

## 1.5 L'auto-complétion contextuelle

La touche **<Tab>** est également un raccourci clavier très connu qui fait la complétion de commandes. Débutez une commande, un chemin, appuyez sur **<Tab>** et bash tentera d'auto-compléter la commande ou le chemin s'il n'y a pas d'ambiguïtés. S'il y en a, reapez sur **<Tab>** une seconde fois et bash vous affichera la liste des choix disponibles.

Je cite ce raccourci, car il peut devenir encore plus puissant avec l'ajout du paquet **bash-completion**. Ce paquet permet à bash d'auto-compléter les options des commandes, les noms d'hôtes ssh, etc. Il faut ajouter les lignes suivantes à votre fichier de configuration **.bashrc** :

```
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi
```

## 1.6 Expansion d'alias et de globs

Le shell permet la création d'alias, et l'utilisation de *globs* (les étoiles ou points d'interrogation). C'est extrêmement puissant, mais il arrive qu'entre les alias ou les expansions de *globbing*, il soit difficile de savoir quelle commande est réellement lancée. Par exemple, lorsqu'on lance `ls *txt`, qu'est-ce que le shell va interpréter et quelle commande sera lancée? `<Alt> + <Ctrl> + <e>` étend les alias et `<Ctrl> + <x>` puis `<*>` étend les `*` :

```
$ ls *txt # appui sur <Alt> + <Ctrl> + <e>
$ ls --color=auto *txt
$ ls --color=auto *txt # appui sur <Ctrl> + <x> puis <*>
$ ls --color=auto a.txt secret.txt todo.txt
```

C'est extrêmement utile dans le cas d'un `rm *` par exemple.

## 1.7 Édition de la dernière commande utilisée

`fc`, ou son équivalent `<Ctrl> + <x>` puis `<Ctrl> + <e>` permet d'ouvrir un éditeur de texte, et d'éditer la dernière commande utilisée. Ceci est très utile lorsqu'une commande comporte de nombreuses options, de nombreux arguments. L'édition est bien plus puissante dans un éditeur de texte que par la ligne de commandes. Lorsque l'éditeur de texte ferme, la commande est exécutée.

### NOTE

L'éditeur lancé est celui qui correspond à la variable d'environnement `$EDITOR`.

## 1.8 Et tous les autres :-)

La liste des raccourcis clavier de bash pourrait remplir un *GNU/Linux Magazine* entier : `bind -P` donne la liste de tous les raccourcis et il y en a beaucoup. Si vous souhaitez aller plus loin, il suffit de chercher sur internet « bash shortcuts ». Les utilisateurs d'autres shells possèdent la même richesse de raccourcis aussi je les invite à consulter également leur manuel.

# 2. LES BASH BUILTINS ET VARIABLES QUI RENDENT LA VIE PLUS FACILE

Après les raccourcis clavier qui permettent de simplifier la vie de l'utilisateur, nous allons voir les petites fonctionnalités de bash qui permettent de gagner du temps.

## 2.1 Enchaînement de commandes

`&&` se place entre deux commandes shell : `cmd1 && cmd2` signifie que si `cmd1` réussit, alors on exécute `cmd2` sinon ne fait rien.

```
$ true && echo 'ça marche'
ça marche
$ false && echo 'ça marche'
$
```

C'est une manière très rapide d'enchaîner les commandes avec condition. Voici un exemple un peu plus parlant :

```
$ make build && git push
```

Inutile de rester devant la console en attendant la fin du `build`, et le `push` ne sera fait que si le `build` réussit.

Le `||` fait exactement l'inverse, `cmd1 || cmd2` signifie que `cmd2` est lancé seulement si `cmd1` échoue :

```
$ true || echo 'ça échoue'
$ false || echo 'ça échoue'
ça échoue
$ make build || mail moi@corp -s 'build cassé!!'
```

## 2.2 Les tests simples

Les tests simples se font via les crochets `[` et `]`. La liste des options est disponible dans le shell bash, mais les plus simples et utiles sont :

- `-f` : est-ce que le fichier existe (fichier ordinaire) ;
- `-s` : est-ce que le fichier existe et est non nul ;
- `-z` : est-ce que la variable d'environnement existe.

Combiné avec le `||` ou le `&&`, cela permet d'écrire des scripts ou *one-liner* très compacts :

```
$ [ -f file.log ] && rm file.log
$ # pas de message d'erreur si
file.log n'existe pas...
```

Ou dans des scripts :

```
$ cat a
#! /bin/bash
[ -z $1 ] && (echo "donnez un argument" ; exit)

[ -f file.lock ] && (echo "fichier lock
présent, exit"; exit)
```

# VOUS L'AVEZ RATÉ ? VOUS AVEZ UNE SECONDE CHANCE !

GNU/LINUX MAGAZINE HORS-SÉRIE n°86

```
touch file.lock
#doing stuff
rm file.lock
$ ./a
donnez un argument
$ ./a a
$ touch file.lock
$ ./a a
fichier lock présent, exit
$
```

## 2.3 \$PS1 et suivants

La variable **PS1** correspond à la ligne du prompt :

```
$ echo $PS1
\u@\h:\w/$
```

**\u** est remplacé par le nom d'utilisateur, **\h** par l'hôte, **\w** par le *working directory* et **\\$** affiche un **\$** (ou un **#** si l'uid vaut root). Mais cette variable peut être remplacée par une multitude d'options et peut se configurer extrêmement finement. Voici quelques exemples (**\A** affiche l'heure) :

```
kevin@slackware:~$ export PS1='> '
> echo "pas un super prompt"
pas un super prompt
> export PS1="Un texte statique > "
Un texte statique > echo "ok"
ok
Un texte statique > export PS1='\A -- \u@\h
is working in \w $ '
15:09 -- kevin@dell is working in ~ $ cd /tmp
15:09 -- kevin@dell is working in /tmp $
```

Il est possible d'appeler des fonctions lors de la construction de la variable d'environnement **PS1**. Par exemple :

```
kevin@slackware:~$ last_cmd() { [ $? -eq
0 ] || echo "last command failed - " ; }
kevin@slackware:~$ export PS1="\last_
cmd\ \u@dojo:\w/$ "
kevin@slackware:~$ true # la
commande réussit, le prompt ne change pas
kevin@slackware:~$ false # la
commande fail, le prompt l'affiche
last command failed - kevin@slackware:~$
true
kevin@slackware:~$
```

La customisation de **PS1** pourrait elle aussi remplir un *GNU/Linux Magazine* entier. Il est possible d'appeler des codes ANSI (changer la couleur des caractères), d'afficher la branche git en cours, de changer de couleur si on travaille à distance en ssh, etc. Des sites web proposent même de customiser en



# 75 RECETTES POUR ACCÉLÉRER VOS DÉVELOPPEMENTS !

À NOUVEAU  
DISPONIBLE  
CHEZ VOTRE MARCHAND  
DE JOURNAUX ET SUR :  
<http://www.ed-diamond.com>



ligne le **PS1** de manière graphique et fournissent un bloc de code à copier/coller dans son `~/.bashrc`.

Les variables **PS2**, **PS3** et **PS4** servent à personnaliser certains prompts secondaires, et sont moins utiles.

## 2.4 Le mode debug du bash

bash est un interpréteur de commandes, comme nous le voyons, mais c'est aussi un langage de script. Et bash dispose d'un mode debug qui peut s'appeler avec l'option **-x** :

```
$ cat test.sh
#!/bin/bash

echo hop
[ -z "$1" ] && echo "pas de variable $1"
ls *plop
$ bash test.sh
hop
pas de variable
ls: impossible d'accéder à *plop: Aucun fichier
ou dossier de ce type
$ bash -x test.sh
+ echo hop
hop
+ '[' -z '' ']'
+ echo 'pas de variable '
pas de variable
+ ls '*plop'
ls: impossible d'accéder à *plop: Aucun fichier
ou dossier de ce type
```

bash affiche chacune des commandes qu'il exécute par un symbole **+** avant son exécution. S'il s'agit d'un script appelé par un autre programme, il est possible de remplacer le shebang `#!/bin/bash` par `#!/bin/bash -x` dans la première ligne.

## 2.5 Gérer les données sauvegardées dans l'historique

bash conserve la liste des commandes tapées. Cela permet à l'utilisateur de retrouver les anciennes commandes utilisées via **<Flèche haut>** ou **<Ctrl> + <cr>**. Dans certaines situations, l'enregistrement de la commande n'est pas forcément voulu ou souhaitable. Si le contenu de cette variable contient le caractère espace, alors toute commande qui commence par un espace sur la ligne de commandes n'est pas enregistrée dans l'historique :

```
$ export HISTIGNORE=' '
$ echo "this is not secret"
this is not secret
$ echo "this is secret" # un espace précède le echo
```

```
this is secret # la commande est
exécutée
$ # rappel de la commande précédente avec
<Flèche haut>
$ echo "this is not secret"
```

La commande précédée d'un espace n'est pas stockée dans l'historique.

## 2.6 Correction automatique de chemins

L'option **shopt -s cdspell** est pratique, car bash va tenter de corriger de lui-même les chemins avec faute de frappe :

```
$ shopt -s cdspell
$ cd /usr/local/bin # il manque un o
/usr/local/bin # bash indique la correction
/usr/local/bin$ # nous arrivons au bon
endroit
```

## 2.7 Quelques alias sympathiques

### 2.7.1 Please

```
alias please='sudo $(history -p \!\!)
```

Cet alias est souvent cité en exemple, et son utilité n'est plus à démontrer (cet alias doit être placé dans le `.bashrc`) :

```
$ /etc/init.d/exim4 start
[...] Starting exim4 (via systemctl):
exim4.serviceFailed to start exim4.service:
Access denied
failed!
$ please
sudo /etc/init.d/exim4 start
[ ok ] Starting exim4 (via systemctl):
exim4.service.
```

### 2.7.2 Des titres dans les terminaux

```
alias cdbuild=cd /chemin/long/vers/build ;
printf '\!\!e]0;IN_BUILD\a\!\!\'
```

Lorsque nous travaillons sous X, les shells sont disponibles dans des terminaux (**xterm**, **Terminal**, etc.). Ces terminaux ont un titre, qui reprend généralement le prompt **PS1** du shell. Mais ce titre est configurable via une séquence ANSI : `\!\!e]0`. Il est ainsi possible d'ajouter une information visuelle à l'environnement. Lorsque je travaille, je renomme automatiquement mes terminaux afin de pouvoir distinguer d'un coup d'œil où je me trouve.

### 2.7.3 Ne pas utiliser d'alias

De nombreuses commandes sont *aliasées* sur elles-mêmes, comme :

```
alias ls=ls --color
```

Si un utilisateur souhaite utiliser la commande sans alias, il suffit de la préfixer par un `\` :

```
$ \ls
file1
```

## 3. LES PETITES COMMANDES INJUSTEMENT MÉCONNUES

Comme nous venons de le voir, les options du shell sont extrêmement pratiques. Mais il faut savoir utiliser aussi tout un ensemble de petites commandes, elles aussi extrêmement utiles pour effectuer divers types de tâches.

### 3.1 Ouverture d'un fichier avec l'application par défaut

**xdg-open** est une commande réservée à ceux qui travaillent dans un terminal sous environnement graphique : elle permet d'ouvrir un fichier ou une URL avec l'application par défaut du gestionnaire de fenêtres.

```
$ xdg-open image.jpg
$ xdg-open http://site.intra.corp/docs/rapport.odt
```

Cette commande évite de devoir enchaîner plusieurs commandes, ou d'utiliser la souris pour ouvrir un explorateur de fichiers.

### 3.2 Surveiller l'état d'une commande

**watch** est une commande qui permet de surveiller l'état d'une autre commande. Elle exécute périodiquement la commande souhaitée et l'affiche à l'écran. Par exemple, pour visualiser l'espace qui se libère pendant un **rm -f** :

```
$ rm -f backup/* && watch du -hs backup
# le prompt disparaît
Every 2,0s: du -hs backup/          Mon Nov 28 15:03:14 2016

743M      backup/
```

Et toutes les deux secondes, la commande est rafraîchie à l'écran.

### 3.3 Lire à l'envers : tac et rev

**tac** est une commande qui fait exactement l'inverse de **cat**, c'est-à-dire que les fichiers sont concaténés en partant de la dernière ligne et en remontant à la première :

```
$ cat numbers
one
two
three
$ tac numbers
three
two
one
```

**rev** lit les lignes à l'envers :

```
$ tac numbers | rev
eerht
owt
eno
```

### 3.4 Concaténation ligne par ligne

**paste** est une commande intéressante, car elle sait concaténer deux fichiers ligne par ligne :

```
$ cat numbers
one
two
three
$ cat num
1
2
3
$ paste num numbers
1one
2two
3three
$ paste num numbers -d ";"
1;one
2;two
3;three
```

L'option **-d** permet de changer le délimiteur de séparation. La commande **paste** m'a souvent servi pour ajouter des colonnes à des fichiers csv.

### 3.5 Manipuler des fichiers json

**jq** est un outil que tout utilisateur manipulant du json devrait connaître. C'est un grep très amélioré pour le json, il offre énormément d'avantages. Tout d'abord, il sait afficher proprement du json :

```
$ echo '{"foo": 42, "bar":{"sub": "barssub", "another": 1 }}' | jq .
{
  "foo": 42,
  "bar": {
    "sub": "barssub",
    "another": 1
  }
}
$ # on ne le voit pas ici, mais il fait la coloration aussi :)
```

Il sait aussi extraire des valeurs du json :

```
$ echo '{"foo": 42, "bar":{"sub": "barssub", "another": 1 }}' | jq .foo
42
```

Je ne peux pas faire de tutoriel sur **jq** en quelques lignes, mais je vous invite vraiment à tester cet outil.

### 3.6 Timeout

**timeout** est le genre de commande qui fait toujours regretter de ne pas l'avoir connue plus tôt. Comme son nom l'indique, elle implémente un *timeout*. Il suffit de donner un temps limite en secondes (ou minutes, ou jours) et la commande souhaitée. La gestion des codes d'erreurs est prévue :

```
$ timeout 1 sleep 25
$ echo $?
124 # timeout a coupé la commande sleep
$ timeout 25 sleep 1
$ echo $?
0 #la commande sleep a fini avec succès)
```

Je m'en suis beaucoup servi pour des connexions à des serveurs réseau qui pouvaient être en échec pendant trop de temps, mais elle est utile pour beaucoup d'autres cas.

### 3.7 Duplication de sortie standard

**tee** n'est pas seulement qu'un accessoire de golf. C'est aussi un duplicateur de la sortie standard. **tee** lit ce qui lui arrive depuis l'entrée standard et l'écrit dans la sortie standard et un fichier. Par exemple, vous pouvez vouloir consulter la sortie d'un traitement en temps réel, mais aussi l'enregistrer pour conservation. **tee** résout très facilement ce problème :

```
$ ./do_job | tee out.log
[+]First part of job
OK
[+]Second part of job
OK
$ cat out.log
[+]First part of job
OK
[+]Second part of job
OK
```

La sortie de la commande **do\_job** est dupliquée dans le fichier **out.log**.

### 3.8 Mieux que grep

**grep** est un outil absolument indispensable pour tout utilisateur du shell. Mais **grep** a quelquefois des limites pénibles. Deux outils sont souvent appelés « *a grep better than grep* » :

- **ack** est un outil écrit en **Perl** qui est spécialement prévu pour le programmeur. Par défaut, **ack** ne cherche pas dans les dossiers **.git**, **.svn**, etc. Il permet aussi de ne chercher que dans un type de langage et pas les autres :

```
$ ack --ruby my_function
```

- **ripgrep** (ou **rg**) est un autre clone de **grep** qui est écrit en **Rust** et qui est rapide (en plus d'offrir beaucoup de fonctionnalités). **ripgrep** évite de chercher dans des sous-dossiers de gestions de sources, sait faire de la coloration syntaxique, a un moteur de *regex* puissant, etc.

Si vous utilisez **grep** fréquemment, je vous invite à utiliser un de ces deux programmes qui vous fera gagner du temps.

### 3.9 Gestion du presse-papier

**xclip** est un outil en ligne de commandes permettant de gérer le presse-papier de **X-Window**. Le presse-papier sous X-window permet de sélectionner à la souris (clic gauche) et coller avec clic milieu. **xclip** permet de récupérer la sortie standard (ou un fichier) et copier le résultat dans le presse-papier. C'est extrêmement pratique lors de multiples sessions ssh pour échanger des fichiers (il faut lancer **ssh** avec l'option **-X** ou **-Y** pour accéder au *clipboard* sur la session distante). Il est même possible d'échanger des binaires via l'aide de **base64** :

```
$ ssh -X server
Last login: Mon Nov 28 14:03:42 2016 from gateway
$ cat binary | base64 | xclip
$ exit
$ echo ' # appui sur bouton du milieu de la souris
# affichage du base64
' | base64 -d > binary
```

### 3.10 Et tout le reste

Les distributions GNU/Linux contiennent une multitude de petits outils qui peuvent rendre la vie beaucoup plus facile aux utilisateurs. N'hésitez pas à explorer un dossier comme `/usr/bin`, ou tapez « *better command than command* » dans un moteur de recherche.

## 4. LES PETITES OPTIONS MÉCONNUES DES COMMANDES CONNUES

Certaines commandes classiques ont des options insoupçonnées qui rendent la vie plus facile.

### 4.1 cd -

`cd -` permet de revenir dans le précédent dossier où se trouvait l'utilisateur. C'est particulièrement pratique lorsqu'il est nécessaire de basculer entre deux dossiers :

```
/tmp/src$ cd /tmp/build/
/tmp/build$ #vérification de trucs dans dossier build
/tmp/build$ cd -
/tmp/src          #de retour dans src/
/tmp/src$        #modification de src, ou autres actions
/tmp/src$ cd -
/tmp/build        #de retour dans build/
/tmp/build$ #etc.
```

### 4.2 make sans Makefile

`make` est l'ami du développeur. Mais `make` dispose d'une cible par défaut, sans `Makefile` qui consiste à compiler un fichier source en exécutable à partir de son nom :

```
$ cat hello.c
#include <stdio.h>

void main(void) {
    puts("Hello, world\n");
}
$ make hello
cc    hello.c  -o hello
kevin@slackware:/tmp/src$ ./hello
Hello, world
```

C'est particulièrement utile lors de l'écriture de petits programmes.

### 4.3 Exécuter une commande après un find

La commande `find` permet de trouver tout type de fichiers sur un système GNU/Linux. Cette commande offre une option particulièrement intéressante `-exec` qui permet de lancer une commande pour chacun des fichiers trouvés par `find`. La syntaxe est simple, il suffit de remplacer le nom de fichier par `{}` et finir la ligne de commande par `\;` :

```
$ find . -name "backup_2015*"
-exec rm {} \;
```

De la même manière, il est possible de renommer les fichiers, les déplacer, faire un grep, etc.

### 4.4 Lecture de plusieurs fichiers en parallèle

`tail` est une commande qui affiche la fin d'un fichier. Elle est surtout connue pour son option de flux `-f` qui affiche au fur et à mesure de leur arrivée les lignes ajoutées au fichier. Tout le monde a dû lancer au moins une fois un `tail -f file.log`. Mais `tail -f` peut aussi ouvrir plusieurs fichiers à la fois :

```
$ tail -f build.log crash.log
==> build.log <==
==> crash.log <==
==> build.log <==
[+] Building stuff
make all targets
gcc -o hello hello.c
==> crash.log <==
[ ] gcc get killed by a SIG USR1
You should investigate
==> build.log <==
[+] Cleaning stuff
```

Ce qui permet de lire plusieurs fichiers de log en parallèle sans risque d'ambiguïté sur la provenance du message.

## 4.5 -taupe de netstat

La commande **netstat** affiche la liste des connexions réseau en cours. La combinaison d'options **-taupe** est simple à retenir (il suffit de penser au petit animal qui creuse des tunnels et fait des connexions d'un point à un autre). Dans l'ordre, nous avons toutes (**-a**) les connexions tcp et udp (**-t -u**), le pid responsable de la connexion (**-p** et si vous êtes root vous obtenez en plus le nom complet) avec des informations étendues (**-e**) :

```
$ sudo netstat -taupe
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat User Inode PID/Program name
tcp 0 0 *:ssh *:LISTEN root 26829 826/sshd
tcp 0 0 localhost:smtp *:LISTEN root 16293 1205/exim4
tcp 0 0 localhost:6010 *:LISTEN kevin 24920 2667/0
tcp 0 0 localhost:6011 *:LISTEN kevin 171405 6392/2
tcp 0 0 *:55838 *:LISTEN statd 16401 795/rpc.statd
tcp 0 0 *:sunrpc *:LISTEN root 15613 783/rpcbind
tcp 0 0 *:http *:LISTEN root 15321 1292/apache2
tcp 0 0 192.168.99.2:ssh gateway:58796 ESTABLISHED root 171845 6912/sshd: kevin
...
```

## 4.6 -thor de ls

**thor** n'est pas que le Dieu du tonnerre dans la mythologie nordique. C'est aussi un ensemble d'options pour **ls** qui permet de trier les fichiers selon la date de modification (**-t**) en forme *human-readable* (**-h** qui affiche **K** ou **M** lorsque les fichiers font plusieurs kibi ou mébi octets) en affichant uniquement le possesseur du fichier (**-o**) et l'affichage se fait à l'envers (**-r**), donc le plus vieux en premier et le plus récent en dernier :

```
$ ls -thor
-rw-r--r-- 1 kevin 14 nov. 28 13:08 numbers
-rw-r--r-- 1 kevin 6 nov. 28 13:10 num
-rw-r--r-- 1 kevin 23K nov. 28 16:27 glmf.txt
```

## 4.7 -R de sort

**sort** permet de trier un fichier. L'option **-R** fait précisément l'inverse en renvoyant de façon aléatoire le fichier d'entrée :

```
$ cat numbers
one
two
three
$ sort -R numbers
two
three
one
$ sort -R numbers
one
three
two
kevin@slackware:~$
```

Ce n'est sans doute pas un aléa de niveau cryptographique, mais c'est largement suffisant pour casser l'ordonnement d'une *playlist* par exemple.

## 4.8 tar xvf

Cette dernière astuce concerne l'absence d'option à utiliser. **tar** est une commande réputée peu amicale avec l'utilisateur, dont les options ne suivent pas toujours les mêmes conventions que les autres commandes (pas de tirets à utiliser par exemple). Le format **tar** compressé reste toutefois un moyen incontournable pour échanger des arborescences de fichiers. Le problème se situe au niveau du compresseur utilisé et de son option à employer. Comment désarchiver un **pack.tar.bz2** ou un **pack.tar.xz**, ou encore un ancien **pack.tar.Z** ou **pack.tgz**. À chaque compresseur son option de décompression, ce dont il n'est pas toujours simple de se souvenir. Il est peu connu que **tar** sait de lui-même trouver le bon décompresseur sans option à ajouter :

```
$ ls pack.t*
pack.tar.bz2 pack.tar.xz
pack.tgz
$ tar xvf pack.tar.bz2
file3
$ tar xvf pack.tar.xz
file2
$ tar xvf pack.tgz
file1
```

Un drame dans un strip de **xkcd** aurait pu être évité si Rob avait lu cet article : <https://xkcd.com/1168/> ;)

## CONCLUSION

Nous voici arrivés au terme de cet article. Ce genre d'énumérations est forcément limité, et je ne pouvais pas lister l'ensemble des outils.

J'espère vous avoir intéressé, et n'hésitez pas à ouvrir un man ou *googler* des descriptions de commande : j'ai souvent été surpris de voir que la petite commande que je cherchais à implémenter existait déjà. ■



# À PARTIR DU 1<sup>ER</sup> MARS, CONNECT ÉVOLUE !

# LISEZ CE NUMÉRO ET PLUS DE 150 AUTRES EN LIGNE !



## ACTUELLEMENT SUR CONNECT :

- **CE NUMÉRO**
- **et + de 150 autres numéros de GNU/Linux Magazine**



- **71 numéros Hors-Séries de GNU/Linux Magazine**

# TOUT CELA À PARTIR DE 199 € TTC\*/AN !

\* Tarif France Métropolitaine

Rendez-vous sur [connect.ed-diamond.com](http://connect.ed-diamond.com) pour découvrir Connect !

Pour tous renseignements complémentaires, contactez-nous :

• via notre site internet : [www.ed-diamond.com](http://www.ed-diamond.com)

• par téléphone : **03 67 10 00 20**

ou envoyez-nous un mail à [connect@ed-diamond.com](mailto:connect@ed-diamond.com) !

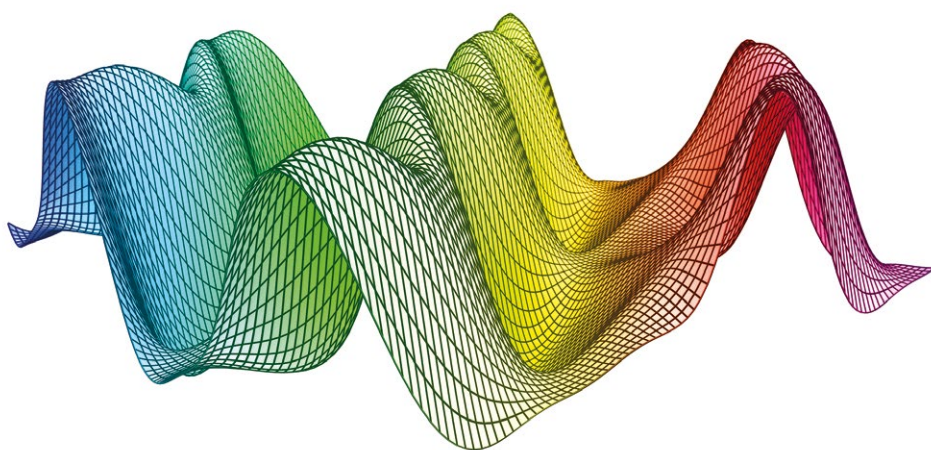


# CAMÉRA 3D ET NUAGE DE POINTS

**LAURENT DELMAS**

[Ingénieur généraliste. Utilisateur GNU/Linux depuis 2001]

**MOTS-CLÉS : RASPBERRY PI, CAMERA 3D, NUAGE DE POINTS**



Pour pouvoir se mouvoir dans l'espace, il est indispensable de pouvoir évaluer les distances avec les objets environnants. Les systèmes d'assistance à la conduite (freinage anticipé, détection de piéton, « radar » de recul) ou bien les drones en sont entre autres de bons exemples. L'évolution technologique de ces dernières années a permis de sortir des laboratoires les capteurs constituant les caméras temps de vol communément appelées caméra 3D. Les applications permises sont innombrables allant de l'automobile à la robotique industrielle en passant par l'agriculture. Cet article va vous présenter le principe de fonctionnement d'une caméra temps de vol. En vous appuyant sur une caméra du commerce [1], vous apprendrez à faire l'acquisition d'un nuage de points puis effectuer divers traitements via l'API PointClouds [2] afin de reconstruire la scène 3D.

La mesure de distance ne date pas d'aujourd'hui, quel que soit le domaine. Depuis de nombreuses années sont utilisés de nombreux capteurs à cet effet tels que les capteurs ultrasons dont le fonctionnement est basé sur la mesure du temps de vol d'une onde acoustique. On retrouve des applications d'estimation de distance aussi bien dans la nature avec la chauve-souris et le dauphin qui utilisent les ultrasons et leurs échos pour se localiser et ainsi éviter d'entrer en collision avec divers obstacles, que dans le domaine médical avec l'échographie ou le domaine maritime avec le sonar en passant par l'incorruptible secteur de l'automobile avec l'aide au stationnement qui emploie également les ultrasons pour déterminer la distance d'un obstacle. L'utilisation de capteurs plus évolués est également mise en œuvre notamment dans l'automobile, l'aéronautique et les applications militaires comme les radars. Le principe de fonctionnement reste similaire à la mesure temps de vol des ultrasons. La différence principale repose sur l'utilisation d'ondes électromagnétiques dont les fréquences sont beaucoup plus élevées que les ondes acoustiques, allant de quelques gigahertz pour les radars à courtes portées (24GHz) à plusieurs dizaines de gigahertz pour les radars longues portées (77GHz). L'avantage d'utiliser des hautes fréquences est

d'accroître d'une part la résolution et d'autre part le rafraîchissement permettant de pouvoir déterminer la vitesse de déplacement des objets environnants en utilisant l'effet Doppler. Bien que l'effet Doppler soit également utilisé avec les ultrasons en échographie notamment. Les scrutateurs laser appelés couramment Lidar utilisent aussi ce principe de mesure de temps de vol pour des faisceaux lumineux cohérents (Laser), majoritairement rouges. L'inconvénient principal des Lidars est la partie mécanique mobile les rendant ainsi sensibles aux chocs et vibrations.

Bien que l'utilisation de caméras stéréoscopiques dont le fonctionnement est identique à celui de nos yeux ou l'association de capteurs infrarouges judicieusement couplés à une caméra classique comme la **Kinect** de **Microsoft** peuvent être utilisés pour obtenir une image tridimensionnelle de l'environnement, ces caméras tout en n'étant pas vraiment compactes, ne réalisent pas directement l'acquisition de la profondeur des éléments des prises de vue et requièrent un traitement pouvant être complexe.

L'évolution et l'amélioration technologiques toujours croissantes ont permis l'arrivée de caméras à temps de vol. En effet, ces caméras utilisent un capteur d'images ayant la particularité de mesurer le temps de vol pour chaque pixel et requièrent donc une miniaturisation des composants électroniques pour obtenir une résolution décente. Les données obtenues avec ces caméras correspondent directement aux dimensions des objets et peuvent être exploitées en tant que telles par l'application.

## 1. PRINCIPE DE FONCTIONNEMENT

Une caméra 3D est constituée d'une source lumineuse modulée (i.e. LED infrarouge), d'une optique dont le but est de focaliser les rayons lumineux réfléchis sur la cellule du capteur d'images. L'ensemble est synchronisé par une électronique dédiée comprenant un microcontrôleur ou microprocesseur suivant les constructeurs et applications. La distance est obtenue en mesurant pour chaque pixel la différence de phase entre la lumière émise et la phase de lumière réfléchi sur l'objet. La figure 1 représente l'architecture simplifiée d'une caméra temps de vol.

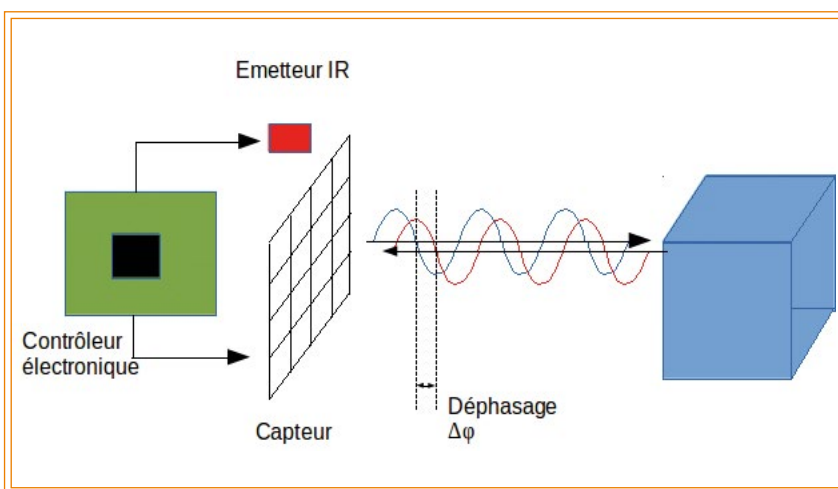


Fig. 1 : Architecture simplifiée d'une caméra temps de vol.

La distance **d** est calculée avec la traditionnelle formule : distance égale vitesse fois le temps. Le retard **t** correspondant au déphasage **Δj** entre l'onde émise et l'onde reçue est déterminé ainsi  $t = \frac{\Delta\phi}{2\pi f}$ . Ce qui permet, en intégrant la fréquence de modulation de l'onde lumineuse émise **f**, d'obtenir la formule suivante pour le calcul de la distance  $d = \frac{c}{2} \frac{\Delta\phi}{2\pi f}$  où **c** représente la célérité de la lumière, soit **3.108** m/s.

Par exemple, pour une source lumineuse modulée à une fréquence de 10MHz, la longueur d'onde correspondante est de 30 mètres, soit une distance maximale de mesure de 15 mètres (la distance maximale étant la demi-longueur d'onde du fait de l'aller-retour de cette dernière).

Bien que de nombreuses caméras 3D soient présentes sur le marché (**Basler**, **IFM**, **PMDtec**, **Fotonic**, **DepthSense**, etc.), seuls quelques capteurs temps de vol existent : la famille des capteurs **REAL3** développée conjointement par **Infineon** et **Pmdtechnologie gmbh** [3], intégrée dans la caméra **Pico Flexx** de **PMDtec** [4] ainsi que dans la caméra **o3d3xx** d'**IFM** [1] ; il y a également la gamme de capteurs CMOS temps de vol **DephSense** de **SoftKinetic** que l'on retrouve chez **Texas Instruments** dans le capteur **OPT8241** [5] ainsi que dans le capteur **MLX75023** de **Melexis** [6] utilisé dans l'automobile.

Cet article a pour but de vous présenter l'utilisation d'une caméra 3D ainsi que l'exploitation des points enregistrés sous forme de nuage en utilisant l'API **PointCloud** [2]. Pour cela, la caméra **o3d303** de **IFM** sera utilisée [1]. Elle est très bien documentée et a le bon goût d'utiliser des protocoles ouverts XML-RPC pour la configuration et TCP/IP pour le transfert des données. De plus, une implémentation GPL des pilotes est faite par **Loveparkrobotics** [7][8].

## 2. MISE EN PLACE DE L'ENVIRONNEMENT DE COMPILATION CROISÉE

Les pilotes développés par Loveparkrobotics ont été développés et validés pour les distributions **Ubuntu**. Toutefois il est possible de les utiliser sur d'autres distributions. Afin d'avoir un intérêt pour les applications embarquées, un **Raspberry Pi 3** avec la distribution **Raspbian** de septembre 2016 sera utilisé. Certes, développer directement sur un Raspberry Pi n'est pas le plus adapté. Il est plus judicieux de mettre en place un environnement de compilation croisée à cet effet [9]. Pour pouvoir réaliser la compilation croisée des pilotes, il faut s'assurer que les dépendances soient disponibles sur l'architecture cible, en l'occurrence une architecture ARM. Le plus simple étant de partir d'une image de la racine du système de fichiers cibles où les dépendances sont installées.

Pour cela, si vous n'avez pas déjà une carte mémoire microSD avec la dernière version de Raspbian, vous allez commencer par la créer. Tout d'abord, rendez-vous sur le site officiel de Raspberry Pi et téléchargez la dernière version de Raspbian [10]. Insérez ensuite la carte microSD sur laquelle vous allez installer Raspbian. Attention, vérifiez à ce qu'il n'y ait pas de données importantes, car elles seraient perdues. Ouvrez un terminal et saisissez la commande suivante :

```
laurent@ASUS:~$ dd bs=4M if=2016-09-23-raspbian-jessie.img of=/dev/sdc
```

Remplacez **/dev/sdc** par le nom correspondant à votre carte microSD.

Une fois terminé, placez votre microSD fraîchement flashée dans votre Raspberry Pi et démarrez-le pour installer tous les paquets nécessaires à la compilation des pilotes. Ouvrez un terminal et saisissez la commande suivante pour l'installation de l'ensemble des paquets nécessaires à la construction des pilotes de la caméra :

```
laurent@ASUS:~$ sudo apt-get install libboost1.55-dev libgtest-dev libgoogle-glog-dev libxmlrpc-core-c3-dev libxmlrpc-c++8-dev libopencv-dev libeigen3-dev libflann-dev libqhull-dev libpcl-dev
```

Voilà votre système est prêt, vous pouvez l'arrêter. Revenez à votre ordinateur, sur lequel va être mis en place l'environnement de compilation croisée : ouvrez un terminal, créez un répertoire de travail et faites une image de la microSD contenant le système de votre Raspberry Pi mis à jour.

```
laurent@ASUS:~$ mkdir workspace
laurent@ASUS:~$ cd workspace
```

```
laurent@ASUS:~workspace$ dd if=/dev/sdc of=/home/laurent/workspace/rasp_base.img
```

À partir de l'image ainsi réalisée, vous allez monter la racine du système de fichiers du Raspberry Pi. Listez les partitions présentes sur la carte à l'aide de la commande **kpartx** :

```
base.img
loop0p1 : 0 129024 /dev/loop0 8192
loop0p2 : 0 15507456 /dev/loop0 137216
```

Vous retrouvez deux partitions, ce qui est tout à fait normal puisque la carte microSD est composée de deux partitions :

- la première, au format VFAT contient le noyau statique et les fichiers de configuration ;
- la seconde, au format ext3 contient la racine du système.

Préparez les deux partitions à être montées :

```
laurent@ASUS:~workspace$ sudo kpartx -l rasp_base.img
add map loop0p1 (252:0) : 0 129024 linear 7:0 8192
add map loop0p2 (252:1) : 0 15507456 linear 7:0 137216
laurent@ASUS:~workspace$ ls -l /dev/mapper
total 0
crw----- 1 root root 10, 236 nov. 13 09:22 control
lrwxrwxrwx 1 root root 7 nov. 13 18:24 loop0p1
-> ../dm-0
lrwxrwxrwx 1 root root 7 nov. 13 18:24 loop0p2
-> ../dm-1
```

Créez un répertoire puis montez dans ce même répertoire la racine du système de la cible (i.e. racine du Raspberry Pi) :

```
laurent@ASUS:~workspace$ mkdir rootfs
laurent@ASUS:~workspace$ sudo mount /dev/mapper/loop0p2 /home/laurent/workspace/rootfs
laurent@ASUS:~workspace$ cd rootfs
laurent@ASUS:~workspace/rootfs$ ls
bin dev home lost+found mnt proc run srv tmp var
boot etc lib media opt root sbin sys usr
```

Vous retrouvez le système de fichiers de votre Raspberry Pi.

Il est temps maintenant d'installer les outils de développement croisés spécifiques à l'architecture ARM sur votre ordinateur. Il existe plusieurs chaînes de compilation ayant chacune ses avantages et inconvénients [11][12][13]. Dans la suite de cet article, va être utilisée la version 4.9 du compilateur **Linaro** [11]. Téléchargez la version correspondant à votre système, puis décompressez l'archive dans le répertoire **/opt** :

```
laurent@ASUS:~workspace$ tar Jxvf gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabi.tar.xz /opt
laurent@ASUS:~workspace$ export PATH=/opt/gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabi/bin:$PATH
```

Voilà, vous avez un environnement de compilation permettant d'une part de construire les pilotes de la caméra et d'autre part de compiler les programmes pour le Raspberry Pi.

Quelle qu'en soit la raison, pour ceux qui souhaitent utiliser la caméra sur ordinateur et non le Raspberry Pi, vous pouvez également installer les dépendances requises par les pilotes sur votre ordinateur. Dans l'exemple ci-dessous le nom de la machine est **ASUS** :

```
laurent@ASUS:~$ sudo apt-get install
libboost1.55-dev libgtest-dev libgoogle-
glog-dev libxmlrpc-core-c3-dev libxmlrpc-
c++8-dev libopencv-dev libeigen3-dev
libflann-dev libqhull-dev libpcl-dev
```

### 3. COMPILATION DES PILOTES

Maintenant que tout est prêt, téléchargez les sources des pilotes depuis le site officiel :

```
laurent@ASUS:~workspace$ git clone
https://github.com/lovepark/libo3d3xx.git
```

Les pilotes sont constitués de trois modules. Un premier module **camera** qui implémente le protocole XML-RPC utilisé pour initialiser et configurer la ou les caméras lorsque plusieurs sont utilisées simultanément dans une même application. Un second module **framegrabber** dont le but est l'acquisition des données provenant de la caméra dans une structure **ByteBuffer** également implémentée dans le module **framegrabber**. Cette structure **ByteBuffer** permet de réaliser l'interface avec des structures d'images et de nuages de points. Le troisième et dernier module image est une passerelle entre le **ByteBuffer** et **OpenCV** et **PCL**.

La compilation des pilotes s'effectue de façon séquentielle. Le module image dépend du module **framegrabber** qui dépend lui-même du module **camera**. Par conséquent, le module **camera** doit être compilé puis installé avant de pouvoir compiler et installer le module **framegrabber** à son tour. Il en est de même pour le module **image** qui nécessite que le module **framegrabber** soit installé.

Pas de difficultés particulières pour effectuer une compilation depuis votre ordinateur. Placez-vous dans le répertoire du module à compiler, puis utilisez la commande **cmake** suivie de la commande **make** :

Expert  
Infrastructure  
& DevOps

Cloud, PaaS & SaaS  
Intégration continue  
Infrastructure hybride



Au cœur de votre  
Transformation  
digitale

Automatisation & DevOps  
Virtualisation & Docker  
Middleware & Big Data

Partenaire de vos projets **Open Source**  
[www.sflx.ca/infra](http://www.sflx.ca/infra)



Savoir-faire  
**LINUX**

Toronto  
+1 647 556-2598

Québec  
+1 418 525-7354

Montréal  
+1 514 276-5468

Lyon & Paris  
+33 9 72 46 89 80



Contribuez à bâtir un monde plus libre  
[carrieres.savoirfairelinux.com](http://carrieres.savoirfairelinux.com)

```

laurent@ASUS: ~workspace$ ls
libo3d3xx          rasp base.img          rootfs
laurent@ASUS: ~workspace$ cd libo3d3xx/modules
laurent@ASUS: ~workspace/libo3d3xx/modules$ ls
laurent@ASUS: ~workspace/libo3d3xx/modules$ cd camera
laurent@ASUS: ~workspace/libo3d3xx/modules/camera$
mkdir build
laurent@ASUS: ~workspace/libo3d3xx/modules/camera$ cd
build
laurent@ASUS: ~workspace/libo3d3xx/modules/camera/
build$ cmake -DCMAKE_INSTALL_PREFIX=/usr -DUBUNTU_
VERSION:STRING=8.6.0 ..
laurent@ASUS: ~workspace/libo3d3xx/modules/camera/
build$ make

```

Il est aussi possible de créer un paquet **Debian** avec la commande **make package** pour une installation ultérieure :

```

laurent@ASUS: ~workspace/libo3d3xx/modules/
camera/build$ make package

```

Pour installer le module, utilisez la commande suivante :

```

laurent@ASUS: ~workspace/libo3d3xx/
modules/camera/build$ sudo make install

```

Procédez de même avec les modules **framegrabber** et **image**.

Concernant la compilation croisée pour le Raspberry Pi, il suffit de passer en argument de la commande **-DCMAKE\_TOOLCHAIN\_FILE** avec le fichier **arm-linux-gnueabi.cmake** contenant la description du compilateur croisé. Plusieurs exemples de fichiers **toolchains** sont fournis dans le répertoire **cmake/toolchains** du projet. Cependant, il vous faudra éditer l'un de ces fichiers avec les répertoires propres à votre installation.

Afin que la gestion de l'environnement croisé soit bien prise en compte, les fichiers **CmakeLists.txt** de chaque module doivent être modifiés. Voici le code du patch appliqué :

```

7c7
< set(CMAKE_INSTALL_PREFIX "/usr")# CACHE STRING
"CMake install prefix")
> set(CMAKE_INSTALL_PREFIX "/usr" CACHE STRING
"CMake install prefix")
9,10c9,11
< set(CMAKE_INSTALL_PREFIX "${CMAKE_FIND_ROOT_
PATH}/usr") #CACHE STRING "CMake install prefix")
< set(CPACK_INSTALL_PREFIX "/usr")# CACHE STRING
"CPack install prefix")
---
> set(CMAKE_INSTALL_PREFIX "${CMAKE_SYSROOT}/usr"
> CACHE STRING "CMake install prefix")
> set(CPACK_INSTALL_PREFIX "/usr" CACHE
STRING "CPack install prefix")

```

Et la traditionnelle ligne de commandes pour appliquer le patch :

```

laurent@ASUS: ~workspace/libo3d3xx/
modules/camera$ patch CMakeLists.txt
CMakeLists.txt.patch

```

Voici l'exemple de fichier **toolchain** utilisé en option de **cmake** afin de compiler les pilotes pour Raspberry Pi sur l'environnement croisé précédemment mis en place.

```

set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSROOT /home/laurent/workspace/rootfs)

set(CROSSSTOOL_ROOT /opt/gcc-linaro-4.9-
2014.11-x86_64_arm-linux-gnueabi)
set(CMAKE_C_COMPILER "${CROSSSTOOL_ROOT}/bin/arm-
linux-gnueabi-gcc")
set(CMAKE_CXX_COMPILER "${CROSSSTOOL_ROOT}/bin/
arm-linux-gnueabi-g++")

set(CMAKE_FIND_ROOT_PATH ${CMAKE_SYSROOT})
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)

set(CROSSSTOOL_EXE_LINKER_FLAGS
"-Wl,-rpath-link,${CMAKE_SYSROOT}/
lib:${CMAKE_SYSROOT}/lib/arm-linux-
gnueabi:${CMAKE_SYSROOT}/usr/lib:${CMAKE_
SYSROOT}/usr/lib/arm-linux-gnueabi")

set(CPACK_DEBIAN_PACKAGE_ARCHITECTURE "armhf")

```

Ainsi que la ligne de commandes **cmake** associée :

```

laurent@ASUS: ~workspace/libo3d3xx/
modules/camera/build$ cmake -DCMAKE_
TOOLCHAIN_FILE=../../cmake/toolchains/
arm-linux-gnueabi.cmake -DUBUNTU_
VERSION:STRING=8.6.0 ..

```

La compilation du module **image** dans l'environnement croisé pose quelques problèmes de par ses dépendances. Il faut modifier le fichier **PCLConfig.cmake** pour que ce dernier prenne en compte l'environnement croisé. Ce fichier se trouve dans le répertoire **rootfs/usr/lib/arm-linux-gnueabi/cmake/pcl**.

Voici le code du patch pour le fichier **PCLConfig.cmake** :

```

550,563c550,556
< #LD
< if(NOT CMAKE_CROSSCOMPILING)
< if(WIN32 AND NOT MINGW)
< # PCLConfig.cmake is installed to
PCL_ROOT/cmake
< get_filename_component(PCL_ROOT
"${PCL_DIR}" PATH)
< else(WIN32 AND NOT MINGW)
< # PCLConfig.cmake is installed to PCL_
ROOT/share/pcl-x.y
< set(PCL_ROOT "/usr")
< endif(WIN32 AND NOT MINGW)

```

```

< else ()
<   set(PCL_ROOT "${CMAKE_SYSROOT}/usr")
< endif ()
<
< message(STATUS "pcl:${PCL_ROOT}")
---
> if(WIN32 AND NOT MINGW)
> # PCLConfig.cmake is installed to PCL_ROOT/cmake
>   get_filename_component(PCL_ROOT "${PCL_DIR}" PATH)
> else(WIN32 AND NOT MINGW)
> # PCLConfig.cmake is installed to PCL_ROOT/share/pcl-x.y
>   set(PCL_ROOT "/usr")
> endif(WIN32 AND NOT MINGW)

```

Appliquez-le en tant que root :

```

laurent@ASUS: ~/workspace/libo3d3xx/rootfs/usr/lib/arm-
linux-gnueabi/hf/cmake/pcl# patch -b PCLConfig.cmake
PCLConfig.cmake.patch

```

Vous pouvez poursuivre la compilation et l'installation du module **image** comme précédemment.

À partir de maintenant, les pilotes sont disponibles et fonctionnels aussi bien sur votre ordinateur que dans votre environnement de compilation croisée. De plus, vous pouvez installer directement les pilotes sur le Raspberry Pi à partir des paquets au format Debian :

```

laurent@raspberrypi:~$ sudo dpkg -i libo3d3xx-camera-0.4.9_armhf.deb
laurent@raspberrypi:~$ sudo dpkg -i libo3d3xx-framegrabber-0.4.9_armhf.deb
laurent@raspberrypi:~$ sudo dpkg -i libo3d3xx-image-0.4.9_armhf.deb

```

## NOTE

Il est possible qu'un message d'erreur apparaisse au moment de l'édition des liens. Si cela se produit, vous devez modifier les fichiers **libc.so** et **libpthread.so** se trouvant dans le répertoire **rootfs/usr/lib/arm-linux-gnueabi/hf/** du système de fichiers monté à partir de l'image de la carte microSD. Ne modifiez rien sur la carte mémoire microSD elle-même. Dans ce but, utilisez les correctifs présents dans le répertoire **libo3d3xx/cmake/toolchains/patches-arm-linux-gnueabi/hf**.

```

laurent@ASUS:~/workspace/$ cd libo3d3xx/cmake/
toolchains/patches-arm-linux-gnueabi/hf
laurent@ASUS:~/workspace/libo3d3xx/modules/camera/
build$ cp lib* /home/laurent/workspace/rootfs/usr/lib/
arm-linux-gnueabi/hf
laurent@ASUS:~/workspace/libo3d3xx/modules/camera/
build$ sudo patch -b libc.so libc.so.patch
laurent@ASUS:~/workspace/libo3d3xx/modules/camera/
build$ sudo patch -b libpthread.so libpthread.so.patch

```

De même, sous la version 16.04 d'Ubuntu, il apparaît une erreur au moment de l'appel de **cmake** due à un bug avec la librairie **libproj.so** [14][15][16]. Pour corriger ce dernier, créez un lien symbolique vers votre version :

```

laurent@ASUS:~/workspace/$ sudo ln -s /usr/lib/x86_64-linux-
gnu/libproj.so.<version> /usr/lib/x86_64-linux-gnu/libproj.so

```

## 4. ACQUISITION ET CONFIGURATION DE LA CAMÉRA

### 4.1 Capture d'un nuage de points

Pour réaliser une acquisition d'images ou plutôt un nuage de points correspondant aux coordonnées des objets vus par la caméra ainsi que l'amplitude du signal de chaque point, il faut initialiser puis configurer la caméra. La communication et la configuration s'effectuent via le protocole XML-RPC [17]. Les différentes fonctions de la caméra sont structurées suivant différents objets XML-RPC comme représentés sur la figure 2 (voir page suivante).

L'utilisation de la caméra passe donc par l'écriture de programmes **C++**. Pour vous familiariser, vous allez initialiser une caméra et faire une acquisition d'image que vous enregistrerez dans un fichier au format PCD. Tout d'abord, pour utiliser les pilotes, il faut bien évidemment inclure les entêtes suivantes :

```

#include "o3d3xx_camera.h"
#include "o3d3xx_framegrabber.h"
#include "o3d3xx_image.h"

```

Puis créer les trois objets indispensables : la caméra, le **framegrabber** et le buffer **Image** correspondant respectivement aux trois modules du pilote.

L'objet **camera** permet d'initialiser la caméra et de la configurer (taux de rafraîchissement, temps d'exposition, adresse IP, etc.)

L'acquisition est réalisée par l'objet **framegrabber** via plusieurs fonctions qui spécifient le mode d'acquisition : à intervalle de temps régulier, déclenché par interruption logicielle, ou bien en continu.

Quant au buffer **Image**, il permet la conversion et l'extraction des données

brutes au format PCD ou image respectivement via les API PCL et OpenCV.

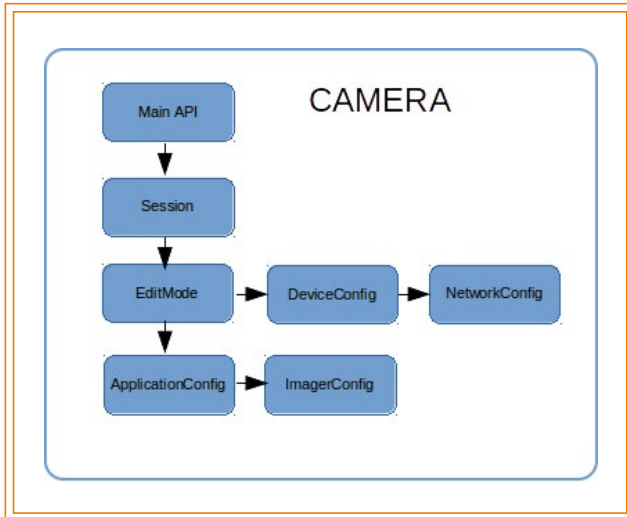


Fig. 2 : Objets XML-RPC constituant la caméra.

Le programme suivant initialise les différents objets nécessaires puis réalise la capture et enregistre le nuage de points dans le fichier **nuage01.pcd** :

```

int main(int argc, const char **argv)
{
    o3d3xx::Logging::Init();

    o3d3xx::Camera::Ptr camera = std::make_
shared<o3d3xx::Camera>();
    o3d3xx::ImageBuffer::Ptr img = std::make_
shared<o3d3xx::ImageBuffer>();
    o3d3xx::FrameGrabber::Ptr fg = std::make_share
d<o3d3xx::FrameGrabber>(camera);

    if (! fg->WaitForFrame(img.get(), 500))
    {
        std::cerr << "Timeout !!!" << std::endl;
        return -1;
    }

    pcl::io::savePCDFFileASCII("nuage01.pcd",
*(img->Cloud()));
    return 0;
}
  
```

Pour compiler l'ensemble, le plus simple est de créer un fichier **CmakeLists.txt** dont voici le contenu :

```

project(Cam3D)
cmake_minimum_required(VERSION 2.8)

set(CMAKE_BUILD_TYPE Release)
  
```

```

set(CMAKE_MODULE_PATH
  ${Cam3D_SOURCE_DIR}/cmake
  ${CMAKE_MODULE_PATH}
)

include(ubuntu_version)
find_package(o3d3xx_camera 0.4.6 EXACT
REQUIRED)
find_package(o3d3xx_framegrabber 0.4.6
EXACT REQUIRED)
find_package(o3d3xx_image 0.4.6 EXACT
REQUIRED)
find_package(PCL 1.7 REQUIRED)

if(NOT (UBUNTU_VERSION VERSION_LESS
"16.04"))
    list(REMOVE_ITEM PCL_IO_LIBRARIES
"vtkproj4")
endif()

SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}
-std=c++11")

include_directories(
  ${O3D3XX_IMAGE_INCLUDE_DIRS}
  ${O3D3XX_CAMERA_INCLUDE_DIRS}
  ${O3D3XX_FRAMEGRABBER_INCLUDE_DIRS}
  ${PCL_INCLUDE_DIRS}
)

link_directories(
  ${O3D3XX_IMAGE_LIBRARY_DIR}
  ${O3D3XX_CAMERA_LIBRARY_DIR}
  ${O3D3XX_FRAMEGRABBER_LIBRARY_DIR}
  ${PCL_LIBRARY_DIRS}
)

add_definitions(
  ${PCL_DEFINITIONS}
)

add_executable(CAM3D src/mainGNU.cpp)
target_link_libraries(CAM3D
  ${O3D3XX_CAMERA_LIBRARIES}
  ${O3D3XX_FRAMEGRABBER_LIBRARIES}
  ${O3D3XX_IMAGE_LIBRARIES}
  ${PCL_LIBRARIES}
)
  
```

Compilez le programme. Avant de l'exécuter, assurez-vous que votre ordinateur ou le Raspberry Pi est bien configuré sur le même réseau que la caméra. Par défaut, la caméra utilise l'adresse IP **192.168.0.69**. Lorsque tout est bien configuré, démarrez le programme. Apparaît alors dans votre répertoire un nouveau fichier nommé **nuage01.pcd** contenant les points de la scène. Si vous n'avez pas encore installé le visualiseur fourni avec l'API PCL, il est temps de le faire avec la commande suivante :

```

laurent@ASUS:~/workspace/programmes$ sudo
apt-get install pcl-tools
  
```



# Abonnez-vous !



M'abonner ?

Me réabonner ?

Compléter ma collection en papier ou en PDF ?

Pouvoir consulter la base documentaire de mon magazine préféré ?

C'est simple... c'est possible sur :

<http://www.ed-diamond.com>



... OU SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :

Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	



Les Éditions Diamond  
Service des Abonnements  
10, Place de la Cathédrale  
68000 Colmar – France  
Tél. : + 33 (0) 3 67 10 00 20  
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

Ce document est la propriété exclusive de Johann Locatelli (johann.locatelli@businessdecision.com)

Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

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

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : <http://boutique.ed-diamond.com/content/3-conditions-generales-de-ventes> et reconnais que ces conditions de vente me sont opposables.

# Bon d'abonnement

## CHOISISSEZ VOTRE OFFRE !

SUPPORT		PAPIER		PAPIER + BASE DOCUMENTAIRE	
Prix TTC en Euros / France Métropolitaine*				1 connexion BD	
Offre	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC
LM	11 <sup>n°</sup> GNU/Linux Magazine France	<input type="checkbox"/> LM1	69,-	<input type="checkbox"/> LM13	245,-
LM+	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> Hors-Série	<input type="checkbox"/> LM+1	125,-	<input type="checkbox"/> LM+13	299,-
<b>LES COUPLAGES AVEC NOS AUTRES MAGAZINES</b>					
A	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> Linux Pratique	<input type="checkbox"/> A1	105,-	<input type="checkbox"/> A13	399,-
A+	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> Hors-Série + 6 <sup>n°</sup> Linux Pratique + 3 <sup>n°</sup> Hors-Série	<input type="checkbox"/> A+1	189,-	<input type="checkbox"/> A+13	489,-
B	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> MISC	<input type="checkbox"/> B1	109,-	<input type="checkbox"/> B13	499,-
B+	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> Hors-Série + 6 <sup>n°</sup> MISC + 2 <sup>n°</sup> Hors-Série	<input type="checkbox"/> B+1	185,-	<input type="checkbox"/> B+13	629,-
C	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> Linux Pratique + 6 <sup>n°</sup> MISC	<input type="checkbox"/> C1	149,-	<input type="checkbox"/> C13	669,-
C+	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> Hors-Série + 6 <sup>n°</sup> Linux Pratique + 3 <sup>n°</sup> Hors-Série + 6 <sup>n°</sup> MISC + 2 <sup>n°</sup> Hors-Série	<input type="checkbox"/> C+1	249,-	<input type="checkbox"/> C+13	769,-
J	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> Hackable	<input type="checkbox"/> J1	105,-	<input type="checkbox"/> J13	399,-
J+	11 <sup>n°</sup> GNU/Linux Magazine France + 6 <sup>n°</sup> Hors-Série + 6 <sup>n°</sup> Hackable	<input type="checkbox"/> J+1	159,-	<input type="checkbox"/> J+13	459,-
<b>LA TOTALE DIAMOND !</b>					
L	11 <sup>n°</sup> GLMF + 6 <sup>n°</sup> HK* + 6 <sup>n°</sup> LP + 6 <sup>n°</sup> MISC	<input type="checkbox"/> L1	189,-	<input type="checkbox"/> L13	839,-
L+	11 <sup>n°</sup> GLMF + 6 <sup>n°</sup> HS + 6 <sup>n°</sup> HK* + 6 <sup>n°</sup> LP + 3 <sup>n°</sup> HS + 6 <sup>n°</sup> MISC + 2 <sup>n°</sup> HS	<input type="checkbox"/> L+1	289,-	<input type="checkbox"/> L+13	939,-

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | HK = Hackable

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)



**Particuliers = CONNECTEZ-VOUS SUR :**  
<http://www.ed-diamond.com>  
 pour consulter toutes les offres !

\*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

**Professionnels = CONNECTEZ-VOUS SUR :**  
<http://proboutique.ed-diamond.com>  
 pour consulter toutes les offres !

\*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !



Vous pouvez dès lors ouvrir le fichier **nuage01.pcd** comme ceci :

```
laurent@ASUS:~workspace/programmes$ pcl_viewer nuage01.pcd
```

La figure 3 montre une photo classique de la scène et la figure 4 représente le nuage de points obtenu avec la caméra 3D.



Fig. 3 : Photo de la scène.

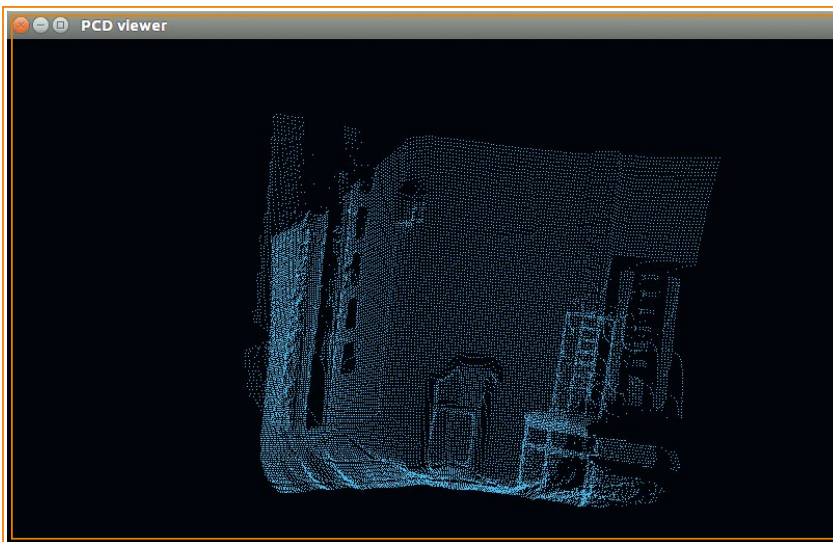


Fig. 4 : Nuage de points de la scène.

Dans l'exemple de cet article, vous retrouvez les deux chaises de la photo, une de taille normale et une seconde de taille d'enfant. De plus, vous constatez que la caméra voit au travers des vitres, ce qui est normal puisque la caméra mesure des temps de vol de rayons infrarouges qui eux-mêmes passent au travers des vitres.

En ouvrant le fichier **nuage01.pcd** avec un éditeur de texte, vous voyez en début de fichier les caractéristiques principales de la prise de vue suivies d'une liste de nombres.

```
VERSION .7
FIELDS x y z intensity
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 176
HEIGHT 132
VIEWPOINT 0 0 0 1 0 0 0
POINTS 23232
DATA ascii
-1.062 -0.753 1.658 444
-1.053 -0.756 1.669 431
-1.049 -0.761 1.688 424
-1.042 -0.765 1.702 412
-1.036 -0.769 1.717 404
```

Les informations importantes et intéressantes de l'en-tête sont la ligne contenant le champ (**FIELDS**), ce qui permet de savoir à quoi correspond la liste des nombres. Ici, il s'agit des coordonnées des points (**x, y, z**) et de l'intensité du signal (**intensity**). Les valeurs **WIDTH** et **HEIGHT** correspondent à la taille de l'image, en l'occurrence 176x132.

## 4.2 Modification des paramètres de la caméra

Jusqu'à présent, vous avez réalisé une capture avec les paramètres par défaut de la caméra. La modification des paramètres peut être faite de deux façons différentes. Soit directement en implémentant dans le programme les méthodes proposées dans le module caméra. Soit en utilisant une chaîne de caractères ou un fichier au format JSON contenant la configuration souhaitée.

Ci-dessous est représenté un programme qui modifie le taux de rafraîchissement en utilisant une chaîne de caractères au format JSON et la méthode **FromJSON** de l'objet **Camera** pour transmettre la configuration à la caméra.

```
int main(int argc, char** argv)
{
    o3d3xx::Logging::Init();

    std::string json =
    R" (
```

```

    {
        "o3d3xx":
        {
            "Device":
            {
                "ActiveApplication": "1"
            },
            "Apps":
            [
                {
                    "TriggerMode": "1",
                    "Index": "1",
                    "Imager":
                    {
                        "FrameRate": "10",
                        "Type": "under5m_moderate"
                    }
                }
            ]
        }
    }
);

//creation de l'objet caméra
o3d3xx::Camera::Ptr CAM=std::make_
shared<o3d3xx::Camera>();
std::cout << "Setting camera configuration: "
<< std::endl
        << json << std::endl;
CAM->FromJSON(json);

return 0;
}

```

Il vous est ainsi possible d'écrire un programme qui édite un fichier au format JSON avec les paramètres de la caméra.

L'utilisation du programme **o3d3xx-dump** affiche toutes les informations de la caméra allant de la version du software aux paramètres d'acquisition en passant par la configuration réseau.

En regardant de plus près le fichier JSON généré par **o3d3xx-dump**, dont voici un extrait, vous remarquerez l'imbrication des éléments comme présentée sur la figure 2.

```

"o3d3xx":
{
  "libo3d3xx": "409",
  "Date": "Sun Nov 20 10:06:29 2016",
  "HWInfo":
  {
    ...
  },
  "SWVersion":
  {
    ...
  },
  "Device":
  {

```

```

...
},
"Net":
{
  ...
},
"Apps":
{
  ...
  "Imager":
  {
    ...
    "ExposureTime": "3000",
    "FrameRate": "1",
    ...
  }
}
}

```

## 5. TRAITEMENT DU NUAGE DE POINTS

Les captures qui ont pu être faites jusqu'à maintenant ne bénéficient d'aucun filtrage ou traitement. Pourtant dans un environnement embarqué, il peut être judicieux, suivant l'application concernée de diminuer le nombre de points à analyser afin de réduire le temps de calcul. L'API PCL offre un ensemble d'outils pour effectuer différents traitements. Quelques-uns vont être présentés dans la suite de ce chapitre.

### 5.1 Filtrage passe-bas

Dans le cas où l'analyse porte sur une zone spécifique, il est intéressant de ne garder que les points de cette zone. Par exemple, une distance comprise entre 2 et 3 mètres. Pour cela, PCL propose une fonction dédiée **PassTrough** (passe-bas) qui applique un filtrage suivant un critère, par exemple l'axe X, Y ou Z. Elle est très facile d'utilisation. Tout d'abord, il faut créer un objet associé au filtre puis définir le nuage de points qui va être filtré (i.e. **nuage01.pcd** ou directement issu de la caméra). Ensuite les paramètres sont choisis et définis. Voici l'exemple de code qui réalise un filtrage suivant l'axe Z pour le nuage de points **nuage01.pcd** et extrait ceux situés entre 2 et 3 mètres dans un nouveau nuage.

```

#include <pcl/io/io.h>
#include <pcl/io/pcd_io.h>

#include <pcl/filters/passthrough.h>

```

```

int main(int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZI>::Ptr nuage
(new pcl::PointCloud<pcl::PointXYZI>);
    pcl::PointCloud<pcl::PointXYZI>::Ptr nuage_
filtre (new pcl::PointCloud<pcl::PointXYZI>);
    pcl::PCDWriter writer;

    if (pcl::io::loadPCDFile<pcl::PointXYZI>
("nuage01.pcd", *nuage) == -1)
    {
        PCL_ERROR ("Impossible de charger nuage.
pcd \n");
        return (-1);
    }
    pcl::PassThrough<pcl::PointXYZI> filtre;
    filtre.setInputCloud (nuage);
    filtre.setFilterFieldName ("z");
    filtre.setFilterLimits (2.0, 3.0);
    pass.filtre (*nuage_filtre);
    writer.write<pcl::PointXYZI> ("nuage_filtre.
pcd", *nuage_filtre, false);
    return 0;
}

```

Visualisez les différences entre le nuage de points d'origine et le nuage de points filtré avec le programme **pcl\_viewer**. Vous remarquez sur la figure 5 que seules les chaises sont présentes dans le nuage filtré. Il est bien sûr possible de mettre en cascade les filtres pour réduire encore plus la zone.

## 5.2 Filtrage par VoxelGrid

Un autre type de filtrage utile est le filtrage VoxelGrid. Il consiste à remplacer un ensemble de points présent dans une « boîte » par un point ayant la position moyenne de l'ensemble des points contenus dans ladite boîte. De même que pour le filtrage passe-bas, il faut instancier l'objet associé au filtre VoxelGrid, définir le nuage de points d'origine (**setInputCloud**)

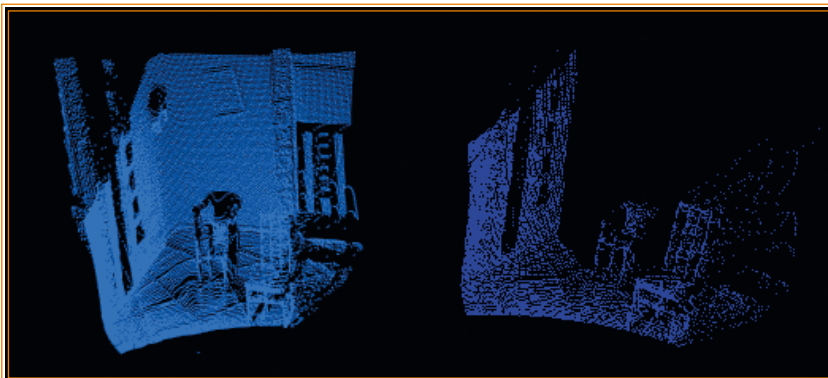


Fig. 5 : Nuage de points avant et après un filtrage *PassThrough*.

puis la dimension de la « boîte » (**setLeafSize**) dans laquelle l'ensemble des points sera remplacé par le point moyen puis appliquer le filtre en précisant le nuage de points dans lequel seront stockés les points après filtrage. Dans l'exemple ci-dessous, la boîte correspond à un cube de 1cm de côté.

```

#include <pcl/io/io.h>
#include <pcl/io/pcd_io.h>
#include <pcl/filters/voxel_grid.h>

int main(int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZI>::Ptr nuage (new
pcl::PointCloud<pcl::PointXYZI>);
    pcl::PointCloud<pcl::PointXYZI>::Ptr nuage_
filtre (new pcl::PointCloud<pcl::PointXYZI>);
    pcl::PCDWriter writer;

    if (pcl::io::loadPCDFile<pcl::PointXYZI>
("nuage01.pcd", *nuage) == -1)
    {
        PCL_ERROR ("Impossible de charger nuage.pcd
\n");
        return (-1);
    }
    pcl::VoxelGrid<pcl::PCLPointXYZI> filtre;
    filtre.setInputCloud (nuage);
    filtre.setLeafSize (0.01f, 0.01f, 0.01f);
    filtre.filter (*nuage_filtre);
    writer.write<pcl::PointXYZI> ("nuage_filtre.
pcd", *nuage_filtre, false);
    return 0;
}

```

La figure 6 (voir page suivante) représente les nuages de points avant et après filtrage.

## 5.3 Extraction de plan

Afin de modéliser une scène avec un logiciel de dessin tridimensionnel tel que **Blender** ou **FreeCAD**, il peut s'avérer utile d'effectuer une extraction de formes cylindre, plan ou autres avec les fonctions **SACSegmentation** et **ExtractIndices**. Comme pour les fonctions de filtrages précédentes, il faut instancier l'objet **SACSegmentation** dans lequel seront définis les paramètres propres à l'extraction tels que le type de **model** (**SACMODEL\_PLANE** pour un plan), la méthode d'approximation (i.e. **SAC\_RANSAC** [14]) et surtout l'écart entre les points acceptés pour l'estimation du plan. Le code suivant est un exemple d'extraction de plan, il calcule également les coefficients de l'équation du plan  $aX+bY+cZ+d=0$  obtenu.

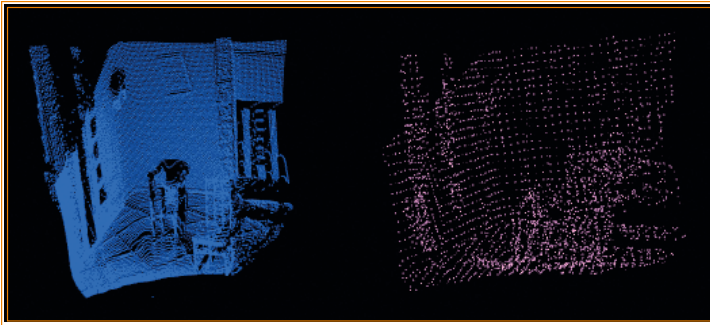


Fig. 6 : Nuage de points avant et après un filtrage VoxelGrid.

```
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/filters/extract_indices.h>
int main(int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZI>::Ptr cloud (new
    pcl::PointCloud<pcl::PointXYZI>);
    pcl::PointCloud<pcl::PointXYZI>::Ptr cloud
    filtered (new pcl::PointCloud<pcl::PointXYZI>);
    pcl::PCDWriter writer;

    if (pcl::io::loadPCDFile("nuage01.
    pcd", *cloud) == -1)
    {
        PCL_ERROR("Couldn't read file\n");
        return -1;
    }

    pcl::SACSegmentation<pcl::PointXYZI> seg;
    pcl::PointIndices::Ptr inliers (new
    pcl::PointIndices);
    pcl::ModelCoefficients::Ptr coef (new
    pcl::ModelCoefficients);
    seg.setOptimizeCoefficients(true);
    seg.setModelType(pcl::SACMODEL_PLANE);
    seg.setMethodType(pcl::SAC_RANSAC);
    seg.setMaxIterations(100);
    seg.setDistanceThreshold(0.03);

    seg.setInputCloud(cloud);
    seg.segment(*inliers, *coef);
    if (inliers->indices.size() == 0)
    {
        std::cout<<"Extimination de plan impossible\
n"<<std::endl;
        return -1;
    }

    pcl::ExtractIndices<pcl::PointXYZI> extract;
    extract.setInputCloud(cloud tmp);
    extract.setIndices(inliers);
    extract.setNegative(false);
    extract.filter(*cloud_filtered);

    std::cout<<"Equation du plan :
aX+bY+cZ+d=0"<<std::endl;
    std::cout<<"Coefficients du plan:"<<coef->
values[0]<<" "
    <<coef->values[1]<<" "
    <<coef->values[2]<<" "
    <<coef->values[3]<<" "<<std::endl;

    writer.write<pcl::PointXYZI>("Plan01.
    pcd", *cloud_filtered, false);
    return 0;
}
```

La figure 7 représente le plan extrait correspondant au plan arrière de la scène.

La console suivante affiche les coefficients de l'équation du plan  $aX+bY+cZ+d=0$ . Vous constatez que le plan est face à la caméra et légèrement incliné à une distance de 3,96 mètres.

```
laurent@raspberrypi:~/workspace/tests/
build $ ./CAM3D
Equation du plan : aX+bY+cZ+d=0
Coefficients du plan:0.321947 0.057562
0.945006 -3.96144
```

## 5.4 Projection sur un plan

Comme pour un scanner, si vous souhaitez extraire des tranches 2D de la scène, vous pouvez utiliser la fonction **ProjectInliers** qui s'utilise comme les filtres précédents. Tout d'abord, il faut définir le type de support (**setModelType**) sur lequel vous souhaitez faire la projection du nuage de points. Il est en effet possible de projeter sur un plan comme sur une sphère. Dans l'exemple suivant, c'est le plan X-Z de normale Y qui est utilisé comme support de projection. Ses coefficients sont donc  $a=c=d=0$  et  $b=1$ .

```
#include <pcl/filters/project_inliers.h>
#include <pcl/ModelCoefficients.h>
int main(int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZI>::Ptr cloud
    (new pcl::PointCloud<pcl::PointXYZI>);
    pcl::PointCloud<pcl::PointXYZI>::Ptr cloud_
    tmp (new pcl::PointCloud<pcl::PointXYZI>);
    pcl::PointCloud<pcl::PointXYZI>::Ptr cloud
    filtered (new pcl::PointCloud<pcl::PointXYZI>);
    pcl::PCDWriter writer;

    if (pcl::io::loadPCDFile("nuage01.
    pcd", *cloud) == -1)
    {
        PCL_ERROR("Couldn't read file\n");
        return -1;
    }
    //plan aX+bY+cZ+d=0
    //plan sur le plan X-Z en Y=0
    pcl::ModelCoefficients::Ptr coef (new
    pcl::ModelCoefficients());
    coef->values.resize(4);
    coef->values[0]=0.0; //a
    coef->values[1]=1.0; //b
    coef->values[2]=0.0; //c
    coef->values[3]=0.0; //d

    pcl::ProjectInliers<pcl::PointXYZI> proj;
    proj.setModelType(pcl::SACMODEL_PLANE);
    proj.setInputCloud(cloud);
    proj.setModelCoefficients(coef);
    proj.filter(*cloud_filtered);
    writer.write<pcl::PointXYZI>("Proj01.
    pcd", *cloud_filtered, false);
    return 0;
}
```

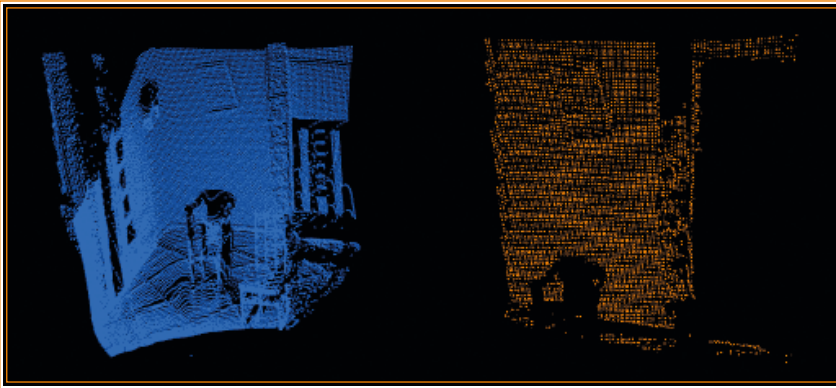


Fig. 7 : Nuage de points du plan extrait.

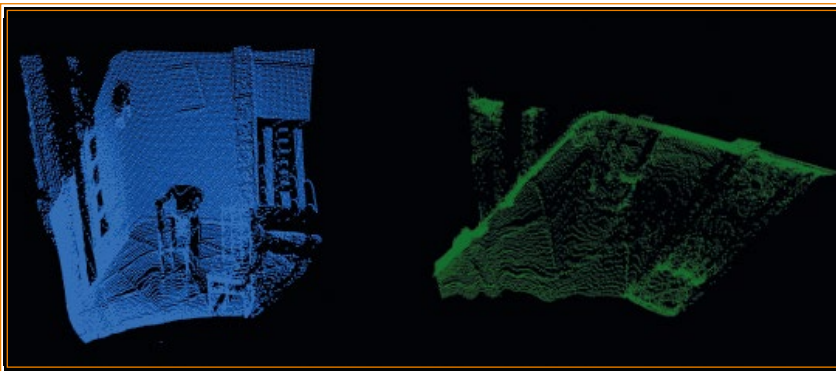


Fig. 8 : Nuage de points original et projection de l'ensemble des points sur le plan X-Z.

La figure 8 montre le nuage de points d'origine et sa projection sur le plan X-Z.

Dans cette section, vous avez vu uniquement quelques éléments de filtrage proposés par l'API PCL ainsi que d'extraction de plan. Il existe bien d'autres fonctions de filtrage, mais pas seulement. PCL fournit des fonctions d'extractions de caractéristiques comme l'histogramme, les points clés (SWIFT), des fonctions de reconnaissances d'objets... En vous référant à la documentation du site PointClouds, vous obtiendrez l'ensemble des possibilités, mais aussi bien d'autres exemples d'utilisation [19].

## CONCLUSION

Lors de cet article, vous avez découvert l'utilisation d'une caméra 3D du commerce avec un Raspberry Pi. De plus, vous avez mis en place un environnement de compilation croisée [9] Raspberry Pi. Afin de compiler les pilotes et ensuite développer vos propres applications embarquées. Vous avez également pris en main la capture d'images 3D avec une caméra du commerce [1]. À partir du nuage de points acquis, vous avez appris à utiliser quelques fonctions basiques de l'API PCL (PointClouds) afin d'appliquer différents filtres, d'extraction de plan et autres. Vous voyez le potentiel que cela apporte aussi bien pour cartographier un environnement que pour son utilisation en réalité virtuelle. ■

## RÉFÉRENCES

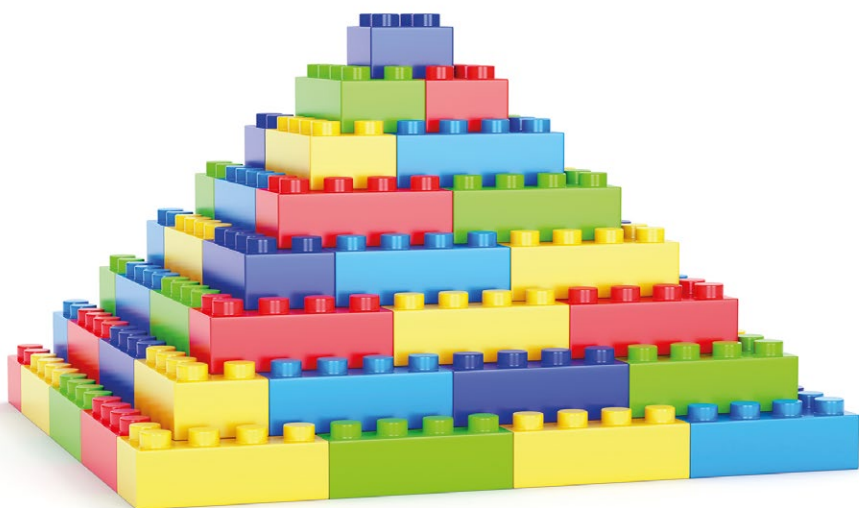
- [1] <http://www.ifm.com/products/fr/ds/O3D303.htm>
- [2] <http://pointclouds.org/>
- [3] [http://www.infineon.com/dgdl/Infineon-REAL3+Image+Sensor+Family-PB-v01\\_00-EN.PDF?fileId=5546d462518fd850151a0afc2302a58](http://www.infineon.com/dgdl/Infineon-REAL3+Image+Sensor+Family-PB-v01_00-EN.PDF?fileId=5546d462518fd850151a0afc2302a58)
- [4] [http://pmdtec.com/picoflexx/downloads/PMD\\_RD\\_Brief\\_CB\\_pico\\_flexx\\_V0201.pdf](http://pmdtec.com/picoflexx/downloads/PMD_RD_Brief_CB_pico_flexx_V0201.pdf)
- [5] <http://www.ti.com/product/opt8241>
- [6] <https://www.melexis.com/en/product/mlx75023/automotive-qvga-time-of-flight-sensor>
- [7] <http://loveparkrobotics.com/>
- [8] <https://github.com/lovepark/libo3d3xx>
- [9] <https://boutique.ed-diamond.com/gnulinix-magazine-hors-series/789-gnulinix-magazine-hs-75.html>
- [10] <https://www.raspberrypi.org/downloads/raspbian/>
- [11] <https://releases.linaro.org/14.11/components/toolchain/binaries/arm-linux-gnueabi/>
- [12] [http://sourcery.mentor.com/public/gnu\\_toolchain/arm-none-linux-gnueabi/](http://sourcery.mentor.com/public/gnu_toolchain/arm-none-linux-gnueabi/)
- [13] <https://wiki.debian.org/ToolChain/Cross>
- [14] <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=819741>
- [15] <https://bugs.launchpad.net/ubuntu/+source/pcl/+bug/1573174>
- [16] <https://bugs.launchpad.net/ubuntu/+source/vtk6/+bug/1573234>
- [17] <http://xmlrpc.scripting.com/>
- [18] [https://en.wikipedia.org/wiki/Random\\_sample\\_consensus](https://en.wikipedia.org/wiki/Random_sample_consensus)
- [19] <http://pointclouds.org/documentation/>

# LES ITÉRATEURS EN C++

SÉBASTIEN CHAZALLET

[Ingénieur Logiciels libres]

MOTS-CLÉS : C++, ITÉRATEUR, ALGORITHMIQUE



**Les itérateurs sont un des piliers du langage C++. Ils permettent de réaliser énormément de tâches essentielles et sont abondamment utilisés dans la bibliothèque standard.**

C++ est un langage de programmation de référence, disposant d'une bibliothèque standard assez conséquente. Cette bibliothèque contient entre autres des objets conteneurs (**vector**, **array**, **deque**, **list**, **map**, etc.). Chacun de ces conteneurs est destiné à stocker des objets en répondant à des exigences particulières. Toujours est-il qu'ils ont quelque chose en commun : pour pouvoir les parcourir, il faut faire appel à un itérateur.

Un itérateur est un concept qui n'est pas exclusif à C++. Il s'agit d'un patron de conception de comportement. Un itérateur est un objet qui permet de parcourir un autre objet en garantissant son intégrité : l'utilisation d'un itérateur en elle-même ne doit pas changer l'état de l'objet parcouru.

Un itérateur définit essentiellement deux méthodes : une permettant d'accéder à l'élément courant et une autre permettant de faire de l'élément suivant

le nouvel élément courant. Ce qui définit la performance de l'itérateur est la complexité de ces deux opérations.

## 1. CRÉATION D'UN ITÉRATEUR

### 1.1 Présentation détaillée

Pour illustrer l'utilisation d'un itérateur, nous allons créer un vecteur :

```
std::vector<int> v {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Un itérateur se crée ainsi :

```
std::vector<int>::iterator it = v.begin();
```

Plus précisément, on vient de créer un itérateur qui pointe vers le premier élément du vecteur : pour faire simple, en C++, l'itérateur est donc un pointeur (pour les détails techniques [1]). On notera aussi que l'itérateur est typé d'une manière particulière : on voit qu'il s'agit d'un itérateur qui agit sur un objet de type vecteur d'entiers. Si on travaille sur un vecteur qui contient autre chose que des entiers, ou encore sur un conteneur autre qu'un vecteur, on utilisera un type d'itérateur différent.



Pour accéder à l'élément courant, il faut faire ceci :

```
int e = *it;
```

Et pour accéder à l'élément suivant :

```
++it;
```

Reste à déterminer à quel moment l'itérateur a atteint la fin du conteneur. Pour cela, il faut créer un autre conteneur :

```
std::vector<int>::iterator it_end = v.end();
```

Pour tester le fait que l'on n'ait pas terminé l'itération, il faut procéder ainsi :

```
it != it_end;
```

Munis de tous ces éléments, nous pouvons écrire notre premier algorithme de parcours d'un vecteur :

```
#include <iostream>
#include <vector>

int main ()
{
    std::vector<int> v {0, 1, 2, 3, 4, 5,
6, 7, 8, 9};

    std::cout << "Le vecteur contient :";
    for (std::vector<int>::iterator it =
v.begin() ; it != v.end(); ++it)
    {
        std::cout << ' ' << *it;
    }

    return 0;
}
```

Le résultat est le suivant :

```
Le vecteur contient : 0 1 2 3 4 5 6 7 8 9
```

On peut constater que la syntaxe utilisée est relativement directe.

## 1.2 Itérer à l'envers

On pourrait imaginer que pour itérer à l'envers, il suffirait d'inverser l'utilisation de **begin** et de **end**. Et bien pas du tout. Il faut utiliser un autre type d'itérateur, conçu spécifiquement pour itérer à l'envers. Voici l'exemple précédent repris pour afficher la liste à l'envers :

```
#include <iostream>
#include <vector>

int main ()
{
```

```
    std::vector<int> v {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    std::cout << "Le vecteur contient :";
    for (std::vector<int>::reverse_iterator it =
v.rbegin() ; it != v.rend(); ++it)
    {
        std::cout << ' ' << *it;
    }

    return 0;
}
```

Le résultat est le suivant :

```
Le vecteur contient : 9 8 7 6 5 4 3 2 1 0
```

On voit que la différence entre les deux codes est minime.

La question qui reste est : quelle différence entre **rbegin** et **end** ? En fait, c'est relativement simple et lié à la façon dont un itérateur est implémenté : lorsque l'on itère à l'en-droit, **begin** est le pointeur vers le premier élément, mais **end** n'est pas le pointeur vers le dernier élément ; c'est un pointeur vers l'élément virtuel qui serait après le dernier élément (donc un élément hors du conteneur, donc un élément qui signifie que l'itération est terminée et que l'itérateur n'est plus valide).

De la même manière, **rbegin** est un pointeur vers le dernier élément du vecteur tandis que **rend** est un pointeur vers un élément virtuel qui se situerait avant le premier élément du vecteur (donc lui aussi en dehors du conteneur).

## 1.3 Itérateurs constants

Un itérateur constant est un itérateur qui ne peut pas être utilisé pour modifier la donnée qu'il pointe. Il est donc un moyen de sécuriser nos itérations lorsqu'elles ne sont censées ne faire que de la lecture.

À vrai dire, nos deux premiers exemples auraient dû être réalisés avec de tels itérateurs, puisque l'on ne fait que de la lecture :

```
#include <iostream>
#include <vector>

int main ()
{
    std::vector<int> v {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    std::cout << "Le vecteur contient :";
    for (std::vector<int>::const_iterator it =
v.cbegin() ; it != v.cend(); ++it)
    {
```

```

        std::cout << ' ' << *it;
    }
    std::cout << "\nLe vecteur contient :";
    for (std::vector<int>::const_reverse_
iterator it = v.crbegin() ; it != v.crend();
++it)
    {
        std::cout << ' ' << *it;
    }

    return 0;
}

```

```

Le vecteur contient : 0 1 2 3 4 5 6 7 8 9
Le vecteur contient : 9 8 7 6 5 4 3 2 1 0

```

Leur bonne utilisation permet d'améliorer la sécurité de programmation : le développeur le choisit spécifiquement pour s'empêcher d'utiliser par inadvertance cet itérateur pour faire des modifications, lorsqu'il sait que cela ne doit pas être fait.

Il faut noter un élément important : lorsque l'on manipule des vecteurs **const**, les méthodes **begin** et **end** renvoient des **const\_iterator** et les méthodes **rbegin** et **rend** renvoient des **const\_reverse\_iterator**.

### 1.4 Itérer et modifier

Par opposition à ce que l'on vient de voir, il est possible d'utiliser un itérateur pour modifier la donnée. Voici un exemple :

```

#include <iostream>
#include <vector>

int main ()
{
    std::vector<int> v(10);
    int i=0;

    for (std::vector<int>::iterator it =
v.begin() ; it != v.end(); ++it)
    {
        *it = i++;
    }

    return 0;
}

```

Comme tout à l'heure, on itère du début à la fin de l'itérateur. Cependant, cela n'est seulement possible que parce que l'on a précisé une taille au vecteur, ici un itérateur de 10 nombres entiers. Sans cela, le programme ne saurait pas sur combien d'éléments il doit itérer.

Pour réaliser la modification de tous les objets parcourus, on va simplement utiliser **\*it** et faire une affectation.

### 1.5 Syntaxe simplifiée (C++11)

À partir de la norme C++ 11, il est possible d'utiliser une syntaxe simplifiée pour parcourir une liste d'objets. Cette syntaxe évite la déclaration du type de l'itérateur (qui est souvent assez longue), et l'appel aux fonctions **begin/end** devient implicite. Voici un exemple de code :

```

for (std::vector<int>::iterator it =
v.begin() ; it != v.end(); ++it)
{
    *it = i++;
}

```

Le code précédent peut s'écrire d'une manière plus légère en procédant ainsi :

```

for (int &it : v)
{
    it = i++;
}

```

## 2. ALGORITHMIQUE

### 2.1 Recherche d'un élément

Rechercher un élément dans un conteneur est l'une des opérations les plus courantes. Voici le vecteur que nous allons utiliser :

```

vector<int> v {0, 1, 2, 3, 4, 5, 6, 7, 8,
9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

```

Pour effectuer une recherche, nous avons besoin de faire appel à un itérateur. Voici comment rechercher, par exemple, le nombre **4** :

```

vector<int>::iterator it = find(v.begin(),
v.end(), 4);

```

Une fois que l'on a effectué cette recherche, on peut regarder le résultat. Si **it** est égal à **v.end()**, cela signifie que la recherche a échoué – rappelez-vous, **end** pointe vers l'élément théorique après le dernier élément et non le dernier élément lui-même, sans quoi il y aurait confusion.

Si l'itérateur a une autre valeur que **end()**, cela signifie que **it** pointe vers l'élément trouvé. On se rappelle également que **\*it** est cet élément. Le point important consiste à savoir à quelle **position** se trouve cet élément. Pour cela,

il faut faire `it-v.begin()`. Ce simple calcul va renvoyer le nombre attendu.

Au final, il est possible de produire un code tel que celui-ci :

```
01: if (it != v.end())
02: {
03:     cout << "élément trouvé: " << *it <<
" à la position " << (it-v.begin()) << endl;
04: }
05: else
06: {
07:     cout << "élément non trouvé" << endl;
08: }
```

Là où les choses se corsent un peu, c'est si l'on souhaite retrouver non pas la première occurrence, mais la dernière. Non pas par rapport à la technique d'itération, mais par rapport à la méthode de calcul de la **position**, alors on fait :

```
01: vector<int>::reverse_iterator rit =
find(v.rbegin(), v.rend(), 4);
02: if (rit != v.rend())
03: {
04:     cout << "élément trouvé: " << *rit <<
" à la position " << (v.size() - 1 - (rit-v.
rbegin())) << endl;
05: }
06: else
07: {
08:     cout << "élément non trouvé" << endl;
09: }
```

On retrouve bien le nombre **4** à la position **14**. Cette méthode de calcul fait intervenir la taille du conteneur et le calcul précédent.

Enfin, on peut aussi souhaiter retrouver toutes les occurrences :

```
01: cout << "éléments trouvés aux positions:";
02: it = v.begin();
03: while (true)
04: {
05:     it = find(it, v.end(), 4);
06:     if (it != v.end())
07:     {
08:         cout << '\ ' << (it-v.begin());
09:         it++;
10:     }
11:     else
12:     {
13:         cout << '\.' << endl;
14:         break;
15:     }
16: }
```

L'astuce dans cet algorithme consiste à faire la recherche (ligne 5) autant de fois que nécessaire, jusqu'à ce que l'on ait

parcouru tout l'objet, mais en la recommençant au rang suivant à chaque boucle. Ce dernier point est réalisé par la ligne 9.

Sachez également qu'il est possible de conduire plusieurs recherches à la fois, c'est-à-dire de rechercher la première occurrence de plusieurs éléments :

```
01: int numbers[] = {4, 2};
02:
03: it5 = find_first_of(v5.begin(), v5.end(),
numbers, numbers+2);
04:
05: if (it5 != v5.end())
06: {
07:     cout << "élément 3 trouvé: " << *it5 <<
" à la position " << (it5-v5.begin()) << endl;
08: }
09: else
10: {
11:     cout << "élément 3 non trouvé" << endl;
12: }
```

Pour terminer, voici comment faire cette même recherche en partant de la fin, à l'aide d'une fonction dédiée :

```
01: it5 = find_end(v5.begin(), v5.end(),
numbers, numbers+2);
02:
03: if (it5 != v5.end())
04: {
05:     cout << "élément 4 trouvé: " << *it5 <<
" à la position " << (it5-v5.begin()) << endl;
06: }
07: else
08: {
09:     cout << "élément 4 non trouvé" << endl;
10: }
```

## 2.2 Fonctions utilisant des itérateurs

Comme on vous le disait, les itérateurs sont des outils indispensables, utilisés de manière très fréquente et en particulier de manière abondante par la bibliothèque standard. Nous allons présenter ici très rapidement l'une d'entre elles, **algorithm**[1].

Cette bibliothèque propose énormément de fonctionnalités qui réalisent des tâches très diverses. Voici pour commencer la fonction **random\_shuffle** :

```
vector<int> v {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
random_shuffle(v.begin(), v.end());
```

Cette fonction va nous permettre de mélanger totalement les valeurs de notre vecteur, le tout en une seule ligne. Nous

n'avons pas besoin d'écrire l'algorithme de bas niveau pour faire une telle tâche, ce qui est un soulagement. Je vous invite à découvrir par vous-même cette bibliothèque particulière, car elle recèle de fonctionnalités particulièrement pratiques et qui deviennent vite indispensables.

Ceci étant dit, il est des cas où l'on se retrouve avec plusieurs manières de faire une même chose. Voici par exemple comment inverser l'ordre des valeurs d'un vecteur :

```
reverse(v.begin(), v.end());
```

Dans ce cas-là, le changement est dit « en place ». En effet, les valeurs à l'intérieur du vecteur **v** ont été inversées. Il est aussi possible de vouloir conserver notre vecteur **v** en l'état et avoir le résultat de l'opération dans une autre variable. Pour ceci, il faut procéder ainsi :

```
vector<int> v2;
v2.resize(v.size());
reverse_copy(v.begin(), v.end(), v2.begin());
```

On voit donc l'importance de la maîtrise des itérateurs : on les retrouve partout ! Ici, on dit que l'on copie les valeurs du vecteur **v** de la fin au début et que l'on commence à écrire les objets copiés à partir du début du vecteur **v2**. On a pris soin au préalable de donner à ce dernier la bonne taille.

Dans ce cas-là, la variable d'origine **v** est conservée et une nouvelle variable est créée pour le résultat : c'est la raison d'être du suffixe **\_copy** dans le nom de la fonction. Notons qu'il existe aussi des fonctions dont le nom simple permet de réaliser une copie alors que le même nom de fonction avec le suffixe **\_inplace** permet de travailler sur le même objet, il s'agit d'un changement « en place ». C'est la différence, par exemple, entre les fonctions **merge** et **merge\_inplace** lesquelles permettent de rassembler tous les éléments de deux conteneurs dans un seul.

Précisons encore que, pour ce cas précis, on peut créer notre vecteur **v2** beaucoup plus rapidement :

```
vector<int> v2 (v.rbegin(), v.rend());
```

Précisons un cas où l'utilisation de la méthode précédente est tout de même un passage obligé :

```
int t[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
std::vector<int> v;
myvector.resize(10);
std::reverse_copy(t, t+10, v.begin());
```

Et oui ! Il est possible de convertir un tableau **C** en un vecteur à l'aide de cette technique. Simple, efficace, on peut même décider de ne garder qu'une partie ou rajouter une partie du contenu d'un tableau dans un vecteur :

```
myvector.resize(14);
std::reverse_copy(t+3, t+9, v.begin()+8);
```

Précisons au passage que la plupart des fonctionnalités de la bibliothèque **algorithm** fonctionnent aussi avec des types standard :

```
int odd[] = {1,3,5,7,9};
int even[] = {0,2,4,6,8};
reverse(odd, odd+5);
random_shuffle(even, even+5);
```

Pour trier à nouveau ces éléments, nous pouvons utiliser ceci :

```
std::sort(odd, odd+5);
std::sort(even, even+5);
```

Nouvelle fonction qui, bien entendu, siérait tout aussi bien aux vecteurs, par exemple.

Voici pour terminer sur ces sujets de conversion et manipulation un dernier exemple permettant de mettre le contenu de deux tableaux d'entiers dans un vecteur d'entiers, tout en le triant :

```
std::vector<int> v4(10);
std::vector<int>::iterator it = v4.begin();

it = std::copy(odd, odd+5, it);
// Résultat : 1 3 5 7 9 0 0 0 0 0
it = std::copy(even, even+5, it);
// Résultat : 1 3 5 7 9 0 2 4 6 8

std::inplace_merge(v4.begin(), v4.begin()+5, v4.end());
```

La fonction **inplace\_merge** fonctionne lorsque les données sont déjà dans le vecteur : il faut les avoir copiées auparavant (ce que l'on a fait avec la fonction **copy**). Elle prend en paramètre le début du vecteur, le milieu (la jointure entre les deux données copiées, ce qui correspond en fait au début de la seconde donnée) et la fin du vecteur.

### 2.3 Différents types d'itérateurs

Dans nos exemples précédents, nous avons utilisé un itérateur sur un vecteur d'entier, ce qui est un cas particulier d'opérateur, car il possède beaucoup de propriétés. On différenciera les types d'itérateurs suivants :

- entrée : itérateur utilisé pour lire des données. Des opérations de comparaisons d'itérateur sont disponibles ;
- sortie : itérateur utilisé pour écrire des données. On peut affecter une valeur au contenu de l'itérateur ;

- avancement : dispose des fonctions **begin** / **end** permettant de se déplacer vers l'avant ;
- bidirectionnel : dispose des fonctions **begin** et **rbegin** / **end** et **rend**. Peut être incrémenté et décrémenté. Autrement, un itérateur qui ne possède pas cette propriété ne peut être qu'incrémenté ;
- accès aléatoire : l'itérateur peut être utilisé pour lire le conteneur de manière non linéaire. L'itérateur dispose de méthodes pour supporter les opérateurs **+** et **-**, l'accès indexé (syntaxe de type tableau avec **[ ]**) et les opérateurs de comparaison **<**, **>**, **<=**, **>=**.

## 3. CRÉER SON PROPRE ITÉRATEUR

### 3.1 Réutilisation de l'itérateur standard

Lorsqu'une classe représente un conteneur d'objet, il est tout à fait possible et recommandé d'implémenter un itérateur sur la liste d'objets contenus.

Cela est d'autant plus simple à réaliser lorsque les objets contenus sont eux-mêmes stockés dans un conteneur standard, possédant déjà un itérateur. Il suffit alors d'implémenter les fonctions *proxy* vers l'itérateur du conteneur (le mot *proxy* est utilisé en référence au patron de conception).

Un tel exemple est réalisé dans le hors-série n°83 de *GNU/Linux Magazine*, traitant de la réalisation d'un casse-brique. En effet, dans ce projet, il est nécessaire de construire des niveaux contenant des listes de briques à casser. Les briques (ainsi que d'autres propriétés) sont contenues dans la classe **Niveau**. L'écriture de deux simples méthodes **begin** / **end** permettent l'itération sur les briques du niveau :

```
01: class Niveau {
02:
03:     public:
04:         /* itérateur sur les element */
05:         std::list<Element*>::iterator
begin() { return m_element.begin(); }
06:         std::list<Element*>::iterator
end() { return m_element.end(); }
07:
08:     private:
09:         std::list<Element*> m_element;
10:
11:         ...
12: }
```

Les pointeurs vers les briques sont contenus dans une liste classique de type **std::list**. L'écriture des méthodes mandataires permet d'accéder directement au contenu de la liste de l'objet et de manipuler un itérateur.

L'utilisation se fait ensuite aussi facilement que nous l'avons vu précédemment :

```
01: for (Element *it : niveau) {
02:     ...
03: }
```

### 3.2 Écriture d'un itérateur spécifique

Il peut se présenter des cas où le besoin d'écrire son propre itérateur se fait sentir, par exemple pour modifier le comportement des méthodes de l'itérateur standard. Il existe alors plusieurs méthodes pour implémenter une telle classe.

Chacun préférera utiliser la programmation générique (*templates*) pour réaliser une implémentation générique, utilisable sur le plus d'objets possible. Ici encore, il sera recommandé de construire sa propre classe d'itérateur à partir d'une classe d'itérateur existante, par exemple en héritant de celle-ci.

Nous pouvons naturellement évoquer la classe de la librairie standard : **std::iterator**, qui est la première solution à envisager. Il est aussi possible d'utiliser des bibliothèques alternatives, telles que **Boost**, qui propose une classe **boost::iterator\_adaptor** pour construire ses propres itérateurs.

## CONCLUSION

Les itérateurs sont un outil très puissant, puisque très souples, génériques et offrant un moyen sécurisé et relativement simple pour parcourir un conteneur de données, quel que soit le type de ce conteneur.

Comme on a pu le voir, il y a un certain nombre de connaissances qu'il faut engranger et de techniques qu'il faut savoir utiliser, mais en fin de compte, cet outil n'est pas si complexe que ça à maîtriser.

Les itérateurs, de par leur omniprésence, sont, pour le développeur C++, un passage obligé. ■

## RÉFÉRENCE

- [1] Documentation technique sur les Random-Access iterator : <http://www.cplusplus.com/reference/iterator/RandomAccessIterator/>.



# LIVE-SYSTEM FROM SCRATCH

**FRANK ENDRES**

[Développeur de Slackware-Live – une solution de construction de système vif]

**MOTS-CLÉS : LIVE-SYSTEM, UEFI, BIOS, INITRAMFS, SQUASHFS, OVERLAY**



De nombreuses distributions Linux proposent des « Live-DVD » afin de permettre aux utilisateurs de les essayer avant de les installer. L'objectif de cet article est de construire son propre système vif, mais avec une finalité un peu différente : disposer d'un système nomade, sur clé USB, avec des données persistantes.

De plus, afin de permettre son utilisation à la fois sur des ordinateurs anciens et récents, le système vif sera amorçable à la fois en mode BIOS et UEFI.

Pour la démonstration, la distribution utilisée est **Arch Linux** (64 bits), avec le bureau léger **LXDE**, mais le principe est le même pour les autres distributions (y compris 32 bits) et environnements de bureau. Les étapes sont les suivantes :

- installation et paramétrage du système,
- partitionnement et formatage de la clé USB,
- transfert du système sur la clé USB,
- création d'un système de fichiers initial,
- configuration de l'amorçage.

Pour réaliser l'opération, une clé USB (ou un disque dur externe) d'au moins 4 Gio (de préférence rapide pour l'écriture)

Un système vif (live-system) avec données persistantes permet de démarrer, avec une simple clé USB, n'importe quel ordinateur comme si c'était le sien. Cet article présente les étapes de la construction d'un tel système, à partir de zéro.

est nécessaire ; attention : il faut effectuer une sauvegarde de son contenu au préalable, car il sera complètement effacé ; dans cet article, cette clé est dénommée **sdz**.

## NOTE

La plupart des commandes sont systèmes et requièrent les privilèges de l'administrateur (root).

# 1. INSTALLATION ET PARAMÉTRAGE DU SYSTÈME

Avant de pouvoir construire un système vif, il faut au préalable installer les paquetages dont il sera constitué. Deux approches, qui ne nécessitent ni partition dédiée, ni machine virtuelle et non intrusive, sont proposées :

- à partir d'un système Arch Linux déjà installé,
- depuis un autre système ; dans ce cas, il faudra aussi un DVD/R ou une deuxième clé USB dénommée **sdx** dans cet article.

Dans les deux cas, les fichiers du système sont installés dans un dossier (donc séparés des autres fichiers du périphérique de stockage) ; l'emplacement d'installation est mémorisé dans la variable d'environnement **ROOT**.

## 1.1 Depuis un système Arch Linux déjà installé

Pour celles ou ceux qui disposent déjà d'un système Arch Linux fonctionnel, l'installation peut se faire dans un simple dossier avec les commandes suivantes (qui ont le même effet que la commande **pacstrap**) :

```
$ su
# ROOT=/mnt/system
# mkdir $ROOT
# mkdir -p $ROOT/var/{cache/pacman/pkg,lib/pacman}
# mkdir -p $ROOT/{dev,proc,sys,run,tmp,etc,boot,root}
# chmod 0750 $ROOT/root
# chmod 1777 $ROOT/tmp
# chmod 0555 $ROOT/proc $ROOT/sys
# for d in dev proc sys; mount --bind /$d $ROOT/$d; done
# pacman --root $ROOT -Sy base xorg lxde midori \
net-tools wireless tools wpa supplicant rfkill \
ipw2100-fw ipw2200-fw b43-fwcutter iw \
network-manager-applet networkmanager
# for d in dev proc sys; umount $ROOT/$d; done
```

## 1.2 Depuis un autre système

Pour celles ou ceux qui utilisent un autre Linux, la solution la plus simple consiste à utiliser le média d'installation d'Arch Linux.

### 1.2.1 Préparer le média d'installation

Télécharger l'image ISO(9660) d'Arch Linux : <https://www.archlinux.org/download>, puis la graver, par exemple avec la commande (**sr0** est généralement le nom associé au graveur de disques optiques) :

```
# cdrecord dev=/dev/sr0 /
chemin/vers/archlinux.iso
```

Ou la transférer sur la deuxième clé USB après en avoir si besoin sauvegardé le contenu :

- lister les périphériques :

```
# fdisk -l
```

- effectuer la copie en remplaçant **sdx** par le périphérique (sans numéro de partition) correspondant à la clé USB ; attention, cette opération est destructive pour les données de celle-ci :

```
# dd bs=4M if=/chemin/vers/
archlinux.iso of=/dev/sdx
```

Enfin, redémarrer l'ordinateur et l'amorcer avec le média d'installation précédemment créé ; si besoin, désactiver le mode « secure boot » dans le programme de configuration de la carte mère.

À l'invite, charger la disposition de clavier française :

```
# loadkeys fr
```

### 1.2.2 Sélectionner l'emplacement d'installation

Lister les partitions du disque dur et repérer celle à utiliser : elle doit avoir un système de fichiers Linux et est dénommée **sd<sup>i</sup>** (i<sup>ème</sup> partition du disque **sd<sup>y</sup>**) dans cet article :

```
# fdisk -l
```

Monter la partition qui contiendra les fichiers du système installé en remplaçant **sd<sup>yi</sup>** par celle choisie :

```
# mount /dev/sdyi /mnt
```

Vérifier l'espace disponible (2 Gio suffisent pour une installation d'un système de base avec un environnement de bureau léger) :

```
# df | grep /dev/sdyi
```

Créer le dossier qui contiendra les fichiers d'installation :

```
# mkdir /mnt/system
# ROOT=/mnt/system
```

### 1.2.3 Installer le système

Installer les groupes de paquets (système de base, environnement graphique et bureau) ainsi que d'éventuels paquets additionnels (ici des outils de configuration réseau, des microprogrammes de cartes WiFi et le navigateur web **Midori**) :

```
# pacstrap $ROOT base xorg lxde midori \
net-tools wireless_tools wpa_supplicant rfkill \
ipw2100-fw ipw2200-fw b43-fwcutter iw \
network-manager-applet networkmanager
```

## 1.3 Paramétrage du système

Le paramétrage consiste essentiellement à paramétrer le système pour la France :

- fuseau horaire :

```
# ln -sf /usr/share/zoneinfo/Europe/Paris
$ROOT/etc/localtime
```

- locales (traductions des textes, unités de mesure, formats de date) :

```
# sed -i 's/^#fr_FR.UTF-8/fr_FR.UTF-8/' $ROOT/
etc/locale.gen
# chroot $ROOT locale-gen
# echo 'LANG=fr_FR.UTF-8' > $ROOT/etc/locale.conf
```

- clavier (en mode console et graphique) :

```
# echo 'KEYMAP=fr' > $ROOT/etc/vconsole.conf
# cat > $ROOT/etc/X11/xorg.conf.d/00-keyboard.
conf << EOF
Section "InputClass"
    Identifiant "system-keyboard"
    MatchIsKeyboard "on"
    Option "XkbLayout" "fr"
EndSection
EOF
```

Ajouter ensuite un utilisateur (recommandé pour un environnement graphique) :

```
# chroot $ROOT useradd -m -g users -G wheel votreLogin
```

Et définir les mots de passe pour cet utilisateur et l'administrateur (root) :

```
# echo -e "root:mdpRoot\nvotreLogin:votreMDP"
| chroot $ROOT chpasswd
```

Enfin, activer le gestionnaire de session pour un démarrage en mode graphique :

```
# chroot $ROOT systemctl enable lxdm
```

Puis le démon du gestionnaire de configuration réseau :

```
# chroot $ROOT systemctl enable NetworkManager
```

## 2. PRÉPARATION DE LA CLÉ USB

La clé USB destinée à contenir le système vif doit avoir un partitionnement particulier afin qu'elle soit amorçable à la fois en mode BIOS et en mode UEFI : il doit être « hybride » (à la fois MsDOS / Intel et GPT), et requiert ces quatre partitions :

- une partition « système EFI », nécessaire pour permettre le démarrage en mode UEFI,
- une partition pour le système vif,
- une partition pour les données persistantes,
- une partition (il s'agit en fait d'une « réservation d'espace ») « boot BIOS », nécessaire à GRUB pour permettre le démarrage en mode BIOS.

Calculer la taille nécessaire au système vif : elle est d'au moins 27 % (meilleure compression) à 37 % (meilleure vitesse) de celle du système installé (prévoir 5 % de marge) :

```
# taille='du -s -BM $ROOT | sed s/M.*$/ /'
# echo "Taille du système installé :
$taille Mio"
# let taille=taille*42/100
# echo "Taille estimée du système vif :
$taille Mio"
```

Connecter la clé USB destinée à contenir le système vif et la partitionner en remplaçant **sdz** par le périphérique correspondant à la clé :

```
# gdisk /dev/sdz
```

Dans la session interactive **GDisk** :

- créer une nouvelle table de partitions : **o**
- créer une première partition : **n**, de type **ef00** (EFI System) ; elle doit commencer au secteur **2048** (valeur par



défaut) et avoir une taille d'au moins 50 Mio (mais il peut être intéressant qu'elle soit plus grande, car ce sera la seule utilisable depuis un système Windows) ;

- créer une deuxième partition : **n**, de type **8300** (Linux) ; elle doit avoir au moins la taille estimée pour le système vif ;
- créer une troisième partition (celles pour les données) occupant le reste de l'espace disponible, et de type **8300** (Linux) ;
- créer une dernière partition, numérotée **128** (par exemple – pour se rappeler qu'elle est à part) entre les secteurs **34** et **2047** de type **ef02** (*BIOS boot partition*) ;
- accéder aux commandes de transformation et récupération : **r** pour créer un MBR hybride : **h** ; saisir les partitions : **1 2 3**, puis accepter les valeurs par défaut (types des partitions : EF et 83) et répondre non aux autres questions ;
- enfin, écrire la nouvelle table : **w**.

Vérifier le partitionnement avec la commande **gdisk -l /dev/sdz** ; pour une clé USB de 8 Gio avec une partition EFI de 500 Mio et 2 Gio pour le système vif, le résultat devrait ressembler au tableau ci-dessous.

Une fois la clé USB partitionnée, procéder au formatage des partitions :

- la partition EFI doit avoir un système de fichiers FAT (le label est libre) :

```
# mkfs.vfat -n "EFI" /dev/sdz1
```

- le label (ici « LiveSys ») du système de fichiers de la partition contenant le système vif est libre, à condition d'utiliser le même pour la détection du média :

```
# mkfs.ext4 -L "LiveSys" /dev/sdz2
```

- pour la partition contenant les données, il faut un système de fichiers Linux ; de plus sur un périphérique de stockage à base de mémoire Flash, il est recommandé de désactiver la journalisation afin de limiter les écritures (pour des raisons de performances et d'usure du média) :

```
# mkfs.ext4 -L "Data" -O "^has_journal" /dev/sdz3
```

Number	Start (sector)	End (sector)	Size	Code	Name
1	2048	1026047	500.0 MiB	EF00	EFI System
2	1026048	5220351	2.0 GiB	8300	Linux filesystem (système vif)
3	5220352	15630302	5.0 GiB	8300	Linux filesystem (données)
128	34	2047	1007.0 KiB	EF02	BIOS boot partition

## Boot BIOS/UEFI et tables des partitions

Pendant de nombreuses années, les cartes mères des ordinateurs compatibles PC étaient équipées d'un microprogramme, le BIOS (*Basic Input Output System*) ; ce dernier utilisait le MBR (*Master Boot Record*, premier secteur d'un périphérique de stockage) qui contenait :

- un unique code pour amorcer le système (celui de Microsoft, de LiLo, ou de Grub...), l'installation de Windows après Linux empêchant le démarrage de ce dernier ;
- une table des partitions permettant d'en définir un maximum de quatre (dont une étendue pouvant elle-même être subdivisée en lecteurs logiques) avec une taille limitée de disque de 2 Tio ; c'est le partitionnement dit *MsDOS / Intel*.

Cette organisation étant limitée, une nouvelle norme a vu le jour ; le microprogramme UEFI (*Unified Extended Firmware Interface*) remplace aujourd'hui le BIOS :

- les programmes d'amorces sont désormais enregistrés chacun dans leur propre dossier (ce qui permet d'en avoir plusieurs simultanément) sur une partition de type ef00 (c'est le code hexadécimal correspondant à « *EFI System* »), formatée en FAT (*File Allocation Table*) ;
- le partitionnement, de type GPT (*GUID Partition Table*), est défini entre les secteurs 2 à 33 du disque dur.

Le partitionnement GPT et Intel ne se chevauchant pas, il est possible (dans les limites du partitionnement Intel) de le définir deux fois, dans chacune des deux normes (c'est le partitionnement hybride).

De même, il est possible de placer un programme d'amorce dans la partition EFI, et dans le MBR ; dans ce dernier cas, les 442 octets réservés ne suffisent pas à GRUB (*GRand Unified Bootloader*) qui requiert une partition supplémentaire de type ef02 (*BIOS boot Partition*) entre les secteurs 34 et 2047.

Les microprogrammes UEFI disposent le plus souvent d'un mode de compatibilité parfois appelé CSM (*Compatibility Support Module*) ou *legacy* permettant de prendre en charge le démarrage en mode BIOS.

### 3. TRANSFERT DU SYSTÈME SUR LA CLÉ USB

Cette étape consiste à transférer le système installé vers la clé USB sous la forme d'un fichier « SquashFS », un système de fichiers compressé en lecture seule [1].

La partition du système vif aura le contenu suivant ; son point de montage est mémorisé dans la variable d'environnement **LIVE** :

- dossier **grub** : fichiers et configuration de GRUB (créé dans l'étape 5 de cet article),
- fichier **initrd.gz** : système de fichiers initial (créé dans l'étape 4),
- fichier **system.sfs** : fichier SquashFS contenant le système vif,
- fichier **vmlinuz** : noyau Linux.

Installer l'utilitaire permettant de créer un système de fichiers SquashFS (en configurant au préalable le WiFi si nécessaire) :

```
# wifi-menu
# pacman -Sy squashfs-tools
```

Monter la clé USB, puis créer le système de fichiers SquashFS :

```
# LIVE=/mnt/live
# mkdir $LIVE
# mount /dev/sdz2 $LIVE
# mksquashfs $ROOT $LIVE/system.sfs -comp
lzo -b 256K
```

#### NOTE

L'algorithme LZO et la taille de bloc de 256 Kio ont été choisis pour la vitesse de décompression (plus de quatre fois plus rapide que XZ), au détriment du ratio de compression ; pour favoriser ce dernier facteur, remplacer `-comp lzo -b 256K` par `-comp xz -b 1M` pour économiser environ 30 % d'espace disque (cf. comparatif [2]).

Copier ensuite le noyau Linux :

```
# cp $ROOT/boot/vmlinuz-linux $LIVE/
vmlinuz
```

Enfin, copier le contenu du dossier **home** (du système installé) dans la partition dédiée aux données :

```
# mkdir /mnt/data
# mount /dev/sdz3 /mnt/data
# cp -dpR $ROOT/home/* /mnt/data
# umount /mnt/data
# rmdir /mnt/data
```

### 4. CRÉATION DU SYSTÈME DE FICHIERS INITIAL

Le système de fichiers initial (*InitRamFS*) est un système de fichiers chargé en mémoire, dont le rôle consiste à charger les pilotes nécessaires et à monter le véritable système de fichiers.

#### 4.1 Squelette

La distribution Arch Linux dispose de la commande **mkinitcpio** pour générer un système de fichiers initial ; ce type d'utilitaire effectue les opérations suivantes :

- création d'une arborescence de système Linux minimaliste contenant **Busybox**, un logiciel qui implémente les utilitaires standards Linux, mais reprogrammés, simplifiés et combinés dans un seul exécutable pour économiser de la place ;
- ajout de pilotes (contrôleurs USB, systèmes de fichiers, etc.) ;
- paramétrage du script d'initialisation ;
- création d'une archive CPIO compressée du système.

Dans Arch Linux, cet utilitaire peut être paramétré en utilisant des crochets (*hooks*) prédéfinis pour ajouter le support du gestionnaire de périphériques Udev ou l'inclusion des pilotes de périphériques de stockage (de type bloc) par exemple ; pour un paramétrage plus fin, il est également possible de demander l'inclusion de pilotes (modules) spécifiques comme les systèmes de fichiers SquashFS, Overlay et Extended 4 (cf. type de système de fichiers de la partition contenant le système vif) :

```
# cat $ROOT/etc/mkinitcpio.conf | \
sed -e 's/^HOOKS=.*$/HOOKS=\"base udev
modconf block keyboard\"/' \
-e 's/^MODULES=.*$/MODULES=\"squashfs
overlay ext4\"/' \
> $ROOT/etc/mkinitcpio-live.conf
```

Générer un squelette de système de fichiers initial (cf. étapes a et b) en utilisant le fichier de configuration précédemment créé :

```
# mount --bind /proc $ROOT/proc
# mount --bind /dev $ROOT/dev
# chroot $ROOT mkinitcpio -s -k /boot/
vmlinuz-linux -c /etc/mkinitcpio-live.conf
# umount $ROOT/dev
# umount $ROOT/proc
```

Le squelette est généré dans le dossier **root** d'un dossier temporaire (au nom aléatoire), renommé ici **initrd-tree** :

```
# mv $ROOT/tmp/mkinitcpio.* /root $ROOT/
initrd-tree
# rm -rf $ROOT/tmp/mkinitcpio.*
# rm -f $ROOT/etc/mkinitcpio-live.conf
```

## 4.2 Script d'initialisation

Le script d'initialisation doit mettre en place une arborescence particulière avant de basculer sur le système vif :

```
/live # système de fichiers en RAM
|- media # partition contenant le système vif
|- squashfs # fichier SquashFS
|- changes # changements apportés au système
vif (en RAM)
/mnt # union de /live/changes(rw) & /live/
squashfs(ro)
```

Écraser le script d'initialisation **\$ROOT/initrd-tree/init** par défaut avec le contenu indiqué ci-après (cf. étape c) ; remarque : ce script est également disponible sur le dépôt GitHub du magazine.

Les premières lignes du script d'initialisation servent à monter les pseudos systèmes de fichiers et à démarrer le service **UDev** dont le rôle est ici de charger automatiquement les pilotes en fonction du matériel détecté puis des commandes de montage :

```
#!/bin/ash
mount -t proc proc /proc -o
nosuid,noexec,nodev
mount -t sysfs sys /sys -o nosuid,noexec,nodev
mount -t devtmpfs dev /dev -o mode=0755,nosuid
mount -t tmpfs run /run -o
nosuid,nodev,mode=0755
kmod static-nodes --format=tmpfiles --output=/
run/tmpfiles.d/kmod.conf
systemd-tmpfiles --prefix=/dev --create --boot
/usr/lib/systemd/systemd-udev --daemon
--resolve-names=never
udevadm trigger --action=add --type=subsystems
udevadm trigger --action=add --type=devices
udevadm settle
```

Les lignes qui suivent permettent de créer un système de fichiers **TmpFS** (en mémoire volatile) qui contiendra les changements apportés au système vif :

```
mkdir /live
mount -t tmpfs tmpfs /live
mkdir /live/media
```

L'étape suivante consiste à rechercher (un délai peut être nécessaire avant qu'elle ne soit visible par le système) et à monter (en lecture seule) la partition de la clé USB contenant le système vif ; attention à utiliser le même label que lors du formatage pour la recherche avec **blkid** :

```
while [ -z "$livesys" ]; do #tant que la clé
n'est pas détectée
sleep 1 #attendre 1s
livesys='blkid | grep "LABEL=\"LiveSys\""' | sed
-n lp | cut -f1 -d:'
done
mount -o ro $livesys /live/media
livedata='echo $livesys | cut -c1-8'"3"
```

Une fois la clé USB montée, il faut monter le système de fichiers SquashFS :

```
mkdir /live/squashfs
mount -o loop -t squashfs /live/media/
system.sfs /live/squashfs
```

Puis effectuer l'unification entre le système de fichiers en mémoire et SquashFS (cf. encadré) :

```
mkdir /live/changes
mkdir /live/.workdir
mkdir /mnt
mount -t overlay -o workdir=/live/.workdir,\
upperdir=/live/changes,\
lowerdir=/live/squashfs overlay /mnt
```

À ce stade, le système vif est monté (en lecture-écriture) dans le dossier **/mnt** ; les instructions suivantes permettent d'initialiser ses fichiers **/etc/mtab** (montages déjà actifs) et **/etc/fstab** ; ce dernier contient une directive pour monter la partition de données de la clé USB dans le dossier **/home** ; remarque : l'option **noatime** indique de ne pas mettre à jour l'heure de dernier accès aux fichiers, afin de limiter les écritures :

```
cat /etc/mtab | sed 's@overlay /mnt@overlay /@' >
/mnt/etc/mtab
cat > /mnt/etc/fstab << EOF
proc /proc proc defaults 0 0
sysfs /sys sysfs defaults 0 0
tmpfs /dev/shm tmpfs defaults 0 0
none / tmpfs defaults 0 0
$livedata /home ext4 noatime 0 0
EOF
```

Rendre « visible » les montages depuis le système vif :

```
mkdir /mnt/live
mount -o rbind /live /mnt/live
```

Arrêter le service UDev et basculer sur le système vif :

```
udevadm info --cleanup-db
udevadm control --exit
exec switch_root /mnt /sbin/init
```

Une fois ce script d'initialisation écrit, il faut le rendre exécutable :

```
# chmod +x $ROOT/initrd-tree/init
```

## LE SYSTÈME DE FICHIERS OVERLAY

Apparu dans la version du 3.18 du noyau Linux, le système de fichiers Overlay [3] permet à l'instar de ses aînés (notamment UnionFS ou AUFS) de réunir le contenu de plusieurs dossiers (« branches ») en un seul. Les branches peuvent être des points de montage ; la branche haute est en lecture-écriture, tandis que les branches basses sont en lecture seule :

- Lors de l'accès en lecture, un fichier est recherché en priorité dans la branche haute, puis dans les branches basses.
- Lors d'une modification d'un fichier présent dans une branche basse, le fichier est réécrit (cf. COW – *Copy On Write*) dans la branche haute ; lors de la suppression d'un fichier présent dans une des branches basses, un fichier spécial indiquant son effacement est créé dans la branche haute : il s'agit d'un fichier de même nom, de type caractère, avec des numéros majeurs et mineurs nuls, et aucun droit.

Dans le cadre des systèmes vifs (notamment lorsqu'ils sont sur disque optique), ce type de système de fichiers, associé à SquashFS et TmpFS permet de simuler une racine (*RootFS*) en lecture-écriture alors que le système est en lecture seule ; pour fonctionner, le système Linux a en effet besoin de pouvoir créer ou modifier certains fichiers.

## 4.3 Archive CPIO compressée

Enfin, créer l'archive CPIO compressée contenant le système de fichiers initial (cf. étape d) puis effacer éventuellement le dossier **initrd-tree** (désormais inutile) :

```
# ( cd $ROOT/initrd-tree; find . | bsdcpio
-o -H newc \
| gzip -9c > $LIVE/initrd.gz; )
# rm -rf $ROOT/initrd-tree
```

## 5. CONFIGURATION DE L'AMORÇAGE

La dernière étape consiste à rendre la clé USB amorçable. De nombreux systèmes vifs utilisent SysLinux [4], mais la solution retenue présentée ici est basée sur GRUB [5] (moins documentée pour cet usage) ; pour l'installer :

```
# pacman -Sy grub
```

### 5.1 Installation de GRUB pour le démarrage en mode UEFI

Monter la partition EFI de la clé USB, et y créer le dossier **/EFI/BOOT** :

```
# mkdir /mnt/efi
# mount /dev/sdz1 /mnt/efi
# mkdir -p /mnt/efi/EFI/BOOT/
```

Créer ensuite un fichier de configuration qui pointe sur celui de la partition contenant le système vif (ainsi, la même configuration pourra être utilisée pour les démarrages en modes BIOS et UEFI) :

```
# cat > /mnt/efi/EFI/BOOT/grub.cfg << EOF
search --label --set=root "LiveSys"
set prefix=(\${root})/grub
EOF
```

Créer enfin l'image EFI de GRUB (fichier **/EFI/BOOT/bootx64.efi**) ; elle embarque le fichier de configuration et un minimum de modules : ceux de « base » ainsi que ceux permettant le support du partitionnement GPT et des systèmes de fichiers « Extended » (pour pouvoir accéder aux fichiers de la partition contenant le système vif) ; attention à ne **pas** utiliser la commande **grub-install** (pour le mode EFI) ici :

```
# grub-mkimage -p "" -o /mnt/efi/EFI/BOOT/
bootx64.efi \
  -O x86_64-efi -c /mnt/efi/EFI/BOOT/grub.cfg \
  search normal boot linux multiboot configfile
all_video part_gpt ext2
# rm -f /mnt/efi/EFI/BOOT/grub.cfg #embarqué
dans l'image EFI
# umount /mnt/efi
# rmdir /mnt/efi
```

Remarque : pour un système 32 bits, il faut remplacer **x86\_64-efi** par **i386-efi** dans la commande **grub-mkimage**.

## 5.2 Installation de GRUB pour le démarrage en mode BIOS

L'installation en mode BIOS est plus simple ; la commande suivante copie une partie de GRUB dans les 442 premiers octets du MBR et le reste dans la partition de démarrage dédiée au BIOS :

```
# grub-install --force --target i386-pc
--boot-directory $LIVE /dev/sdz
```

Il ne reste plus qu'à créer le fichier de configuration basique (sans menu) pour charger le noyau, le système de fichier initial et amorcer le système :

```
# cat > $LIVE/grub/grub.cfg << EOF
linux /vmlinuz
initrd /initrd.gz
boot
EOF
umount $LIVE
rmdir $LIVE
```

## 6. TESTER LE SYSTÈME VIF

Redémarrer à présent sur la clé USB contenant le système vif, et « croiser les doigts », car plusieurs problèmes peuvent survenir (la liste du tableau ci-dessous n'est pas exhaustive) :

Symptôme	Cause	Résolution
clé non amorçable (en modes BIOS et / ou EFI)	mauvaise installation de GRUB	refaire l'étape 5 (penser à monter la partition contenant le système vif)
plantage du script d'initialisation	erreur dans le script	refaire l'étape 4 en ajoutant dans le script d'initialisation la commande <b>sh</b> juste avant la commande <b>switch_root</b> pour pouvoir diagnostiquer le problème

## CONCLUSION

Cet article aura guidé le lecteur ou la lectrice étape par étape dans la réalisation de son propre système vif, à partir de zéro, dans un objectif double :

- pédagogique : pour comprendre le principe de fonctionnement d'un système vif ;
- de « production » : afin de disposer de son propre système Linux nomade (avec persistance des données).

Les opérations requièrent de la rigueur et de la persévérance, mais y être parvenu soi-même (selon la « philosophie » DIY – *Do It Yourself*) apporte beaucoup de satisfaction...

J'espère que vous aurez eu autant de plaisir à suivre cet article que j'en ai eu à vous faire partager mon expérience dans ce domaine ; cet article s'inspire en effet de la solution **Slackware-Live [6]** que j'ai développée à l'origine pour **Slackware** et qui supporte maintenant aussi Arch Linux. ■

## RÉFÉRENCES

- [1] Site officiel de *SquashFS* : <http://squashfs.sourceforge.net>
- [2] Comparatif des algorithmes de compression : <https://frama.link/quashfs-performance-testing>
- [3] Site officiel du système de fichiers *Overlay* (URL raccourcie) : <https://frama.link/OverlayFS>
- [4] Documentation *SysLinux* : <https://wiki.archlinux.org/index.php/Syslinux>
- [5] Documentation *GRUB* : <https://www.gnu.org/software/grub/manual/grub.html>
- [6] Solution Slackware-Live : <https://slackware-live.tuxfamily.org>

# CODE JAVA CONCIS AVEC LOMBOK

**ROMAIN PELISSE**

[Sustain Developer @ Red Hat]

Relecture par Pauline Durand-Mabire

**MOTS-CLÉS : JAVA, PROGRAMMATION OBJECT, IDE, ANNOTATION, SUCRE SYNTAXIQUE**



**Si Java est un langage reconnu pour sa (relative) simplicité et clarté, il ne l'est clairement pas pour sa concision. À bien des égards, il est même parfois très (trop?) verbeux. D'ailleurs, de nombreux langages alternatifs, qu'il s'agisse d'autres plateformes comme Ruby ou de langages construits sur la JVM comme Closure ou Scala, se vantent sans cesse, et avec raison, d'être plus concis. Néanmoins, cette verbosité n'est pas une fatalité, comme l'illustre l'excellent projet Lombok.**

Le projet **Lombok** est une initiative très intéressante apparue il y a maintenant plusieurs années de cela, visant à offrir une série d'annotations permettant d'alléger la syntaxe du langage Java. L'idée qui se cache derrière est très simple : lors de la précompilation, les annotations sont remplacées par du code généré. Ainsi, lorsque le code est compilé, les instructions remplacées par les annotations sont présentes.

Le cas d'utilisation le plus simple à imaginer est la génération d'accesseurs – et bien évidemment, ce fut là le premier problème que le projet Lombok ait abordé. En effet, il est très aisé de générer les méthodes nécessaires à partir de la simple signature de l'attribut de la classe, et de nombreux éditeurs de code avancés permettent déjà de le faire, qu'il s'agisse d'**Eclipse**, de **Netbeans** ou d'**IDEA**. Mais malheureusement, le code généré dans ce cas encombre votre code source, tandis que l'approche de Lombok, plus élégante, libère ce dernier.

Bref, comme nous allons le voir dans cet article, Lombok permet d'alléger et de simplifier votre code, mais aussi de le rendre plus clair, plus conforme et même plus robuste !

## 1. INSTALLATION DE LOMBOK

Lombok est publié sous la forme d'une simple archive Java JAR. Mais il est important de bien comprendre dès maintenant qu'il **n'est pas nécessaire de l'embarquer** au sein de l'applicatif, et ce quel que soit son format de déploiement, puisqu'il a fini son travail lors de la phase de compilation. C'est un point essentiel, car de nombreux *frameworks*, visant à enrichir le langage Java, comme les célèbres **Guava [1]** de Google ou le **Apache Lang [2]**, nécessitent d'embarquer des archives supplémentaires dans l'applicatif en tant que tel. Ce n'est pas le cas ici, et c'est non seulement rafraîchissant, mais c'est aussi un frein en moins à l'adoption du *framework*.

Néanmoins, pour éviter que votre éditeur de code ne signale sans cesse des problèmes de compilation, il est nécessaire d'installer Lombok au sein de ce dernier. Mais nul besoin de s'inquiéter, les développeurs du projet Lombok ont rendu cette opération aussi simple que possible en fournissant une petite interface graphique pour régler ce point rapidement :

```
$ java -jar /chemin/vers/lombok.jar
```

### IntelliJ

Pendant plusieurs années, le très apprécié - mais malheureusement propriétaire IDE - IntelliJ n'était pas réellement supporté par Lombok. Heureusement, un *plugin* spécifique a fait son apparition et il est désormais très facile à installer à partir de l'IDE lui-même.

## 2 L'ANNOTATION @VALUE

Cette première annotation est probablement la plus simple à comprendre – et la plus « rentable », car elle permet de supprimer beaucoup de code de « plomberie ». En effet, cette annotation permet de générer les méthodes accesseurs nécessaires pour lire ou modifier les champs d'un objet, mais elle fournit également une implémentation adaptée des méthodes usuelles `toString()`, `hashCode()` et `equals()`.

Mais avant d'étudier l'utilisation de cette fonctionnalité de Lombok, faisons un court rappel de l'état de l'art de la conception de classe, et particulièrement celle qui est destinée à jouer le rôle de conteneur de données. Ces derniers sont souvent appelés en anglais « *Plain Old Java Object* » (en abrégé : acronyme POJO), et ne représentent généralement qu'un élément du modèle de données de l'applicatif. Leur fonction principale, dont est issu leur nom, est juste de transporter les données modélisées au sein de l'application.

Prenons un exemple simple de POJO. Assumons que votre application gère, entre autres, des utilisateurs authentifiés. Il est donc vraisemblable qu'elle dispose d'une classe « utilisateur » destinée à contenir toutes les données (et métadonnées) relatives à un utilisateur connecté à l'application.

Ainsi, les différents champs de cette classe stockent ces dernières sous la forme d'une série d'attributs. Chacun de ces attributs sera associé à des méthodes accesseurs (*getters* et *setters*).

Les bonnes pratiques de l'industrie requièrent que tous les attributs de ce type de classe soient privés, et que l'on ne puisse donc pas y accéder et surtout les modifier directement. Pour ces opérations, il doit être nécessaire de passer par l'intermédiaire des méthodes accesseurs. Le code de ces méthodes est plus trivial et peut donc être souvent généré par l'IDE, comme signalé plus haut.

Au-delà de ces considérations, une autre bonne pratique a récemment émergé dans la conception de classes de données. En effet, avec l'augmentation du nombre de cœurs dans le système physique, les applications sont de plus en plus concurrentes. C'est-à-dire que leur code s'exécute en parallèle, et donc, souvent, modifie **en même temps**, les instances de classes de données. Pour éviter ce genre de situation, il est donc recommandé d'utiliser des instances **immuables**.

Cette approche réduit les chances d'accès concurrents à une même donnée lors de l'exécution, puisque les objets créés à partir de ces valeurs immuables ne pourront simplement pas être modifiés, forçant, de facto le développeur à utiliser une copie de ces données – s'il lui est nécessaire de les modifier. Sans aller plus loin sur ce sujet, retenons juste pour cet article qu'il est recommandé d'avoir des champs immuables pour les POJO.

Toutes ces considérations préliminaires effectuées, voici ce à quoi pourrait ressembler notre classe **User** si elle respecte toutes ces bonnes pratiques :

```
package org.jboss.set.aphrodite;

public class User {

    private final long id;
    private final String name;
    private final String password;

    public User(long id, String name,
String password) {
        this.id = id;
        this.name = name;
        this.password = password;
    }
}
```

```

public long getId() {
    return id;
}

public String getName() {
    return name;
}

public String getPassword() {
    return password;
}
}

```

À la lecture de ce code, nous pouvons rapidement faire deux constatations. Tout d'abord, le code est déjà très long, alors que l'information qu'il véhicule est au final minuscule – il s'agit juste des noms et types des trois champs de la classe. Et on note immédiatement aussi que ces trois champs forment en fait une estimation très optimiste de la modélisation nécessaire.

En effet, il est plus que vraisemblable que le code d'une telle classe au sein d'une « vraie » application contienne beaucoup plus de données. Pour s'en convaincre, il suffit de jeter un œil à la classe `UserModel` [3] du projet `Keycloak` par exemple. Et pourtant, de manière surprenante, elle est relativement concise, mais tout de même plus élaborée que notre exemple.

Pour rendre la situation encore plus déplorable, nous n'avons pas du tout fini d'implémenter cette classe. Toujours dans une logique de respecter les bonnes pratiques en usage dans le monde Java (et de la programmation orientée objet au sens large), il nous reste encore à ajouter (ou plutôt à surcharger) les méthodes `toString()`, `hashCode()` et `equals()` que nous avons évoquées plus haut.

Là encore, ce genre de code est trivial, et la plupart des IDE savent le générer pour vous, si vous leur demandez gentiment :

```

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + (int) (id ^ (id
>>> 32));
    result = prime * result + ((name == null) ?
0 : name.hashCode());
    result = prime * result + ((password ==
null) ? 0 : password.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    User other = (User) obj;

```

```

if (id != other.id)
    return false;
if (name == null) {
    if (other.name != null)
        return false;
} else if (!name.equals(other.name))
    return false;
if (password == null) {
    if (other.password != null)
        return false;
} else if (!password.equals(other.
password))
    return false;
return true;
}

@Override
public String toString() {
    return "User [id=" + id + ", name="
+ name + ", password=" + password + "];"
}

```

Comme on peut le voir dans l'extrait de code ci-dessus, et à l'exception près de la méthode `hashCode()`, ces méthodes sont juste du code de « plomberie », trivial et très verbeux, mais nécessaire. Tout ceci ayant été détaillé, voyons ce qu'apporte Lombok sur ce sujet.

Si nous utilisons l'annotation `@Value` proposée par Lombok, voici à quoi ressemble la même classe :

```

import lombok.Value;

@Value
public class User {

    long id;
    String name;
    String password;
}

```

En effet, grâce à Lombok, les attributs de la classe `User` ci-dessus sont automatiquement modifiés pour être qualifiés de « privé et de final ». Les accesseurs sont ajoutés, avec un constructeur par défaut, acceptant tous les paramètres requis pour construire une instance valide, et les méthodes `toString()`, `hashCode()` et `equals()` sont, elles aussi, ajoutées et implémentées.

Ce qu'il est essentiel de comprendre ici, c'est que le code binaire (`bytecode java`) généré par la compilation de la classe ci-dessus est **strictement identique** à celui de la classe complète que nous avons documentée plus haut. Dans les faits, si l'on a réduit de beaucoup la quantité de code à écrire, nous n'avons rien perdu en terme de fonctionnalités ! La classe annotée par Lombok est, à l'exécution, identique à la classe « complète ».

L'exécution du code suivant permet de s'en convaincre si nécessaire :



```
public static void main(String[] args) {
    User u = new User(0L, "romain", "pelisse");
    System.out.println(u.toString());
    User o = new User(1L, "romain", "rpelisse");
    if ( ! u.equals(o) ) System.out.println(o);
}
```

```
public class User {
    long id;
    String name;
    String password;
}
```

### 3. LA MÉTHODE « BUILDER »

L'utilisation de méthodes « builder » est très en vogue en programmation orientée objet depuis quelques années. Il faut bien reconnaître que c'est pour de bonnes raisons, car elles améliorent en effet grandement la lisibilité du code. Au cas où le lecteur n'est pas familier de leur fonctionnement, en voici une rapide explication.

L'idée est d'ajouter une **méthode statique** à la classe (nommée « builder »), qui offre une méthode alternative à l'utilisation d'un constructeur, pour allouer une instance à la classe. Cette méthode retourne donc un objet « builder », spécifique à la classe, qui dispose d'autant de méthodes que la classe dispose de champs (chaque méthode nommée comme le champ auquel elle est associée).

Chacune de ces méthodes retourne elle aussi l'instance du « builder » :

```
public UserBuilder id(long id) {
    ...
    return this ;
}
```

Enfin, l'objet « builder » dispose aussi d'une méthode « build », destinée à être invoquée en dernier, et retournant une instance complète de l'objet **User**.

Comme souvent, expliquer du code n'est pas très simple, un exemple rapide est donc sans doute plus parlant :

```
User builderExample = User.builder().id(3L)
    .name("Romain").password("pelisse").build();
```

Là encore, du point de vue de l'implémentation, le code de la méthode **builder()** et des méthodes associées est trivial, mais demande l'ajout de nombreuses lignes de code. Code que l'on qualifiera là encore de « code de plomberie ».

Et une fois encore, Lombok vient à notre rescousse en permettant d'implémenter cette fonctionnalité – et tous les bouts de code nécessaires, à l'aide d'une seule annotation :

```
import lombok.Builder;
import lombok.Value;

@Value @Builder
```

### 4. INSTANCE IMMUTABLE... VRAIMENT ?

Comme évoqué plus haut, il est recommandé de ne créer que des **instances immutables**. Néanmoins, ce n'est malheureusement pas toujours aussi aisé qu'il y paraît.

Ajoutons, par exemple, un attribut à notre classe **User** :

```
import lombok.Builder;
import lombok.Value;

import java.util.Date;
import java.util.Map;

@Value
@Builder
public class User {

    long id;
    String name;
    Map<Long, Date> accessLog;
    String password;
}
```

Si Lombok garantit bien que le champ **accessLog** est immutable (en ajoutant le mot-clé **final** devant), rien n'interdit malheureusement d'en modifier son contenu :

```
user.getAccessLog().put(0L, new Date());
```

Et presque sans nous en rendre compte, nous venons de rendre notre instance supposée immutable, au contraire très mutable. Protéger ce champ contre toute modification involontaire est bien sûr possible et facile, mais là encore, au prix d'un lourd « code de plomberie » à produire.

Mais heureusement, là aussi, ce dernier peut être intégralement généré par Lombok, avec l'ajout de l'annotation suivante :

```
import lombok.Builder;
import lombok.Singular;
import lombok.Value;

import java.util.Date;
import java.util.Map;

@Value
@Builder
```

```
public class User {
    long id;
    String name;
    @Singular("accessLog") Map<Long, Date>
    accessLogs;
    String password;
}
```

Pratique, n'est-ce pas ? Avec une seule annotation, nous venons non seulement de nous assurer que le champ est bien immuable, mais aussi que l'API cliente sera correctement nommée :

```
User builderExample = User.builder().
id(3L).name("Romain").password("pelisse").
accessLog("one access log").accessLog("and an
other").accessLog("and other...").build();
```

Mais Lombok va encore plus loin. Si le nom de l'attribut est un mot anglais dans sa forme plurielle, il peut deviner son singulier ! Nul besoin donc de renseigner l'annotation systématiquement :

```
import lombok.Builder;
import lombok.Singular;
import lombok.Value;

import java.util.List;

@Value
@Builder
public class Parent {
    @Singular List<String> children;
}
```

Ce qui nous donne le code suivant à l'appel :

```
Parent.builder().child("alice").child("bob");
```

## 5. ALLEZ ENCORE PLUS LOIN DANS L'IMMUTABILITÉ

À l'aide de `@Value` et de `@Singular`, l'on s'est déjà assuré qu'aucune instance construite à partir de notre classe ne soit immuable. C'est déjà beaucoup, mais rien n'interdit de modifier la référence vers l'instance, comme la notion de valeur, dans des langages comme Scala, qui permettent de le faire :

```
scala> val s = "Une chaîne de caractère"
scala> s = "Autre Chaîne"
<console>:8: error: reassignment to val
```

On peut évidemment empêcher ce genre d'opération en Java, en ajoutant systématiquement le préfixe `final`, juste devant le type de l'instance :

```
final User = new User();
```

Mais la syntaxe de Scala présentée ci-dessus et qui permet de déterminer automatiquement le type de la valeur - en plus de garantir son immutabilité, est de loin, beaucoup plus concise.

Ainsi, Lombok a donc introduit non pas une annotation, mais un nouveau mot-clé au langage Java :

```
val user = User.builder().id(3L).
name("Romain").password("pelisse").build();
```

## 6. SYNCHRONISATION ET PROGRAMMATION CONCURRENTÉ

La conception de programme utilisant des traitements concurrents est de loin l'une des stratégies de programmation les plus complexes. À bien des égards, les serveurs d'applications JEE développés il y a vingt ans avaient comme objectif avoué (et ambitieux) de libérer autant que possible leurs utilisateurs de ce genre de considérations. La norme recommande même de ne pas utiliser de processus (« thread ») au sein d'une application JEE.

Ceci étant dit, l'évolution du matériel, multipliant le nombre de cœurs sans augmenter leur performance individuelle, force quelque peu les développeurs à revenir à la programmation concurrente. Alors, pas de publicité mensongère, Lombok n'est certainement pas un *framework* de programmation concurrente (comme le célèbre **Akka**, par exemple), mais il offre néanmoins une nouvelle annotation assez appréciable pour l'utilisation, souvent verbeuse, du mot-clé **synchronized** en Java.

Pour rappel, ce mot-clé permet d'indiquer à la machine virtuelle Java que la méthode (ou le bloc) ne peut fonctionner en parallèle – et il assure donc qu'un autre processus (« thread ») entrant dans cet extrait de code sera « mis en pause », tant que le premier processus n'aura pas fini de s'exécuter. Ceci dépasse le cadre de cet article, mais il est important de souligner que ce mot-clé doit être utilisé avec la plus grande parcimonie. Il est très aisé de créer de gros problèmes de performance à l'aide d'un seul **synchronize** mal placé.

Pour l'anecdote, une application bancaire en Allemagne, a posé de très gros problèmes de déploiement, mettant en retard le projet, et créé un certain « vent de panique ». Après analyse, la source du problème s'est révélée être l'utilisation du mot-clé **synchronized** par un des composants du *framework* « web », autour d'une méthode très utilisée pendant la génération des pages HTML.

Une fois le mot-clé retiré, le déploiement de l'application est devenu possible – sans le moindre effet de bord. Est-ce

que l'utilisation de **synchronized** autour de cette méthode était vraiment justifiée ?

Pour être implémentée proprement, une méthode (ou un bloc) synchronisée exige non seulement l'utilisation du mot-clé **synchronized**, mais aussi la définition d'une variable locale, utilisée comme verrou (« lock »). En soit, rien de compliqué, mais ceci est relativement verbeux :

```
private final java.lang.Object $lock = new
java.lang.Object[0];
private DateFormat format = new
SimpleDateFormat("MM-dd-YYYY");

public String synchronizedFormat(Date date) {
    synchronized ($lock) {
        return format.format(date);
    }
}
```

Là encore, Lombok propose une simple annotation, incluant le nom du verrou, si nécessaire, pour réduire la verbosité du code, tout en conservant toutes les informations pertinentes :

```
@Synchronized(" readLock ")
public String synchronizedFormat(Date date) {
    return format.format(date);
}
```

À la compilation, une variable interne, nommée **\_readLock** va être générée. Comme elle n'apparaît pas dans le code, celui-ci est moins verbeux, et la tentation pour le développeur trop « astucieux » d'utiliser cette variable à d'autres fins est également moins forte...

## 7. EXCEPTION

De toutes les constructions verbeuses – et honnêtement peu élégantes – du langage Java, la palme revient assurément à la gestion des exceptions – avec leur encombrante clause **try/catch**. Pour se rafraîchir la mémoire, voici un parfait exemple, très classique, d'ouverture d'un simple fichier :

```
try(BufferedReader br = new BufferedReader(new
FileReader("file.txt"))) {
    StringBuilder sb = new StringBuilder();
    String line = br.readLine();

    while (line != null) {
        sb.append(line);
        sb.append(System.lineSeparator());
        line = br.readLine();
    }
    String everything = sb.toString();
} catch (Exception e) {
    throw new IllegalStateException("File
can't be open:" + );
}
```

À noter que la version ci-dessus n'est pas la plus verbeuse, car nous avons choisi d'intercepter toutes les exceptions – et non pas seulement les quelques-unes qui peuvent être remontées lors de l'exécution des méthodes invoquées.

Là encore, Lombok permet de se débarrasser de l'instruction **try/catch** - et de la remplacer par une simple annotation :

```
public static void main(String[] args) {
    System.out.println(readFile("/tmp/mailcheck.log"));
}

@sneakyThrows
public static void readFile(String filename) {
    BufferedReader br = new BufferedReader(new
FileReader(filename));
    StringBuilder sb = new StringBuilder();
    String line = br.readLine();
    while (line != null) {
        sb.append(line);
        sb.append(System.lineSeparator());
        line = br.readLine();
    }
    System.out.println(sb.toString());
}
```

On notera que, malheureusement, Lombok ne permet pas de réduire le code de l'ouverture du fichier en tant que tel.

## CONCLUSION

Voilà un rapide tour des fonctionnalités de Lombok [4], et des motivations derrière leur implémentation. Son utilisation est très simple, et facilite la lecture du code, sans vraiment induire de risques (rappelez-vous : tout se passe pendant la phase de compilation). Et si le projet ou l'organisation n'en accepte pas l'utilisation, vous pouvez toujours utiliser **delombok**, une extension Java, qui ajoute le code généré aux fichiers sources ! En quelques mots, il n'y a pas de réels points techniques ou de difficultés pouvant empêcher l'adoption du *framework*, donc si ce dernier vous plaît, n'hésitez pas à l'essayer. ■

## RÉFÉRENCES

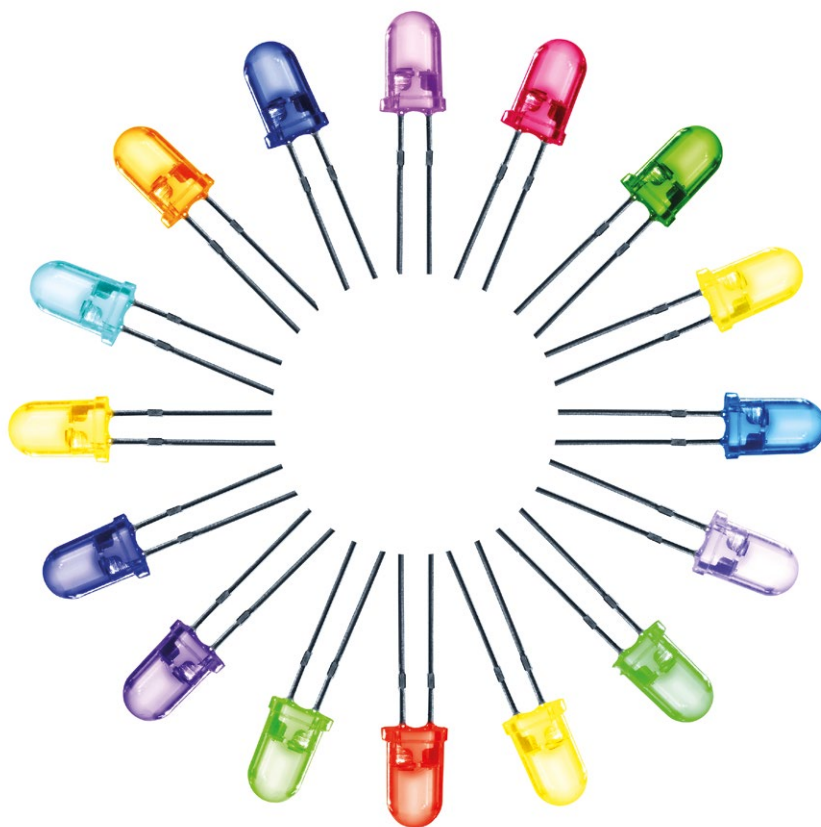
- [1] Guava : <https://github.com/google/guava>
- [2] Apache Commons Lang : <https://commons.apache.org/proper/commons-lang/>
- [3] La classe **UserModel** issue du projet Keycloak : <https://github.com/keycloak/keycloak/blob/master/server-spi/src/main/java/org/keycloak/models/UserModel.java>
- [4] Le site du projet Lombok : <https://projectlombok.org/index.html>



# CONCEPTION D'UN ÉMULATEUR DE LEDS WS2812

TRISTAN COLOMBO

MOTS-CLÉS : WS2812, NEOPIXEL, PYTHON, ÉMULATEUR, MATPLOTLIB, PYGAME



Les leds WS2812, encore appelées NeoPixels, sont des leds RGB programmables pour lesquelles il existe des bibliothèques (en C) et des modules (en Python). Mais comment faire lorsque l'on a commandé une matrice de leds et que l'on voudrait commencer à programmer ? Créer un émulateur bien sûr !

Pour un hors-série du magazine **Hackable** devant paraître sous peu, j'ai travaillé sur un écran de leds sous la forme d'une matrice de 8 x 8 leds WS2812. Mais pour pouvoir publier ce hors-série sur le développement en Python sous Raspberry Pi, il fallait que les lecteurs ne disposant pas encore de tout le matériel nécessaire puissent tout de même programmer. J'ai donc développé un émulateur permettant de remplacer le module **neopixel** [1] et de simuler le comportement de l'écran de leds. Je vous propose ici un retour d'expérience.

## 1. LE MODULE NEOPIXEL

Je me suis donc basé sur le module **neopixel** (voir encadré) pour communiquer avec l'écran et, pour pouvoir émuler l'affichage, il me faut reproduire le fonctionnement de ce module. La première des choses à faire est de savoir ce que fait le module !

```
$ python 3
...
>>> import neopixel
>>> help(neopixel)
Help on module neopixel:
```

```

NAME
    neonpixel

DESCRIPTION
    # Adafruit NeoPixel library port to the rpi_ws281x library.
    # Author: Tony DiCola (tony@tonydicola.com), Jeremy Garff (jer@jers.net)

CLASSES
    builtins.object
        Adafruit_NeoPixel

class Adafruit_NeoPixel(builtins.object)
    | Methods defined here:
    |
    |     __del__(self)
    |
    |     __init__(self, num, pin, freq_hz=800000, dma=5, invert=False, brightness=255,
channel=0, strip_type=1050624)
    |         Class to represent a NeoPixel/WS281x LED display. Num should be the
    |         number of pixels in the display, and pin should be the GPIO pin connected
    |         to the display signal line (must be a PWM pin like 18!). Optional
    |         parameters are freq, the frequency of the display signal in hertz (default
    |         800khz), dma, the DMA channel to use (default 5), and invert, a boolean
    |         specifying if the signal line should be inverted (default False), and
    |         channel, the PWM channel to use (defaults to 0).
    |
    |     begin(self)
    |         Initialize library, must be called once before other functions are
    |         called.
    |
    |     getPixelColor(self, n)
    |         Get the 24-bit RGB color value for the LED at position n.
    |
    |     getPixels(self)
    |         Return an object which allows access to the LED display data as if
    |         it were a sequence of 24-bit RGB values.
    |
    |     numPixels(self)
    |         Return the number of pixels in the display.
    |
    |     setBrightness(self, brightness)
    |         Scale each LED in the buffer by the provided brightness. A brightness
    |         of 0 is the darkest and 255 is the brightest.
    |
    |     setPixelColor(self, n, color)
    |         Set LED at position n to the provided 24-bit color value (in RGB order).
    |
    |     setPixelColorRGB(self, n, red, green, blue)
    |         Set LED at position n to the provided red, green, and blue color.
    |         Each color component should be a value from 0 to 255 (where 0 is the
    |         lowest intensity and 255 is the highest intensity).
    |
    |     show(self)
    |         Update the display with the data from the LED buffer.
    |
FUNCTIONS
    Color(red, green, blue, white=0)
        Convert the provided red, green, blue color to a 24-bit color value.
        Each color component should be a value 0-255 where 0 is the lowest intensity
        and 255 is the highest intensity.

```

Cette simple commande nous donne le squelette de notre programme. Nous savons que nous devons créer un fichier **neopixel.py** (pour que les deux modules soient parfaitement interchangeables), et que ce fichier contiendra une classe **Adafruit\_NeoPixel** composée de dix méthodes et d'une fonction **Color()**.

#### NOTE

Suivant le matériel acheté, il est possible que l'on vous ait vendu des leds PL9823 pour des WS2812B. Ces leds fonctionnent exactement de la même manière... mais elles n'attendent pas les bits de couleur dans le même ordre (GRB au lieu de RGB) [3]. Lors de l'appel de la fonction **Color()** vous devrez donc spécifier le type de leds employé lors de l'appel du constructeur de **Adafruit\_NeoPixel**.

Voici donc le squelette de notre fichier **neopixel.py** :

```
01: class Adafruit_NeoPixel:
02:     def __del__(self):
03:         pass
04:
05:     def __init__(self, num, pin, freq_
hz=800000, dma=5, invert=False, brightness=255,
channel=0, strip_type=1050624):
06:         pass
07:
08:     def begin(self):
09:         pass
10:
11:     def getPixelColor(self, n):
12:         pass
13:
14:     def getPixels(self):
15:         pass
16:
17:     def numPixels(self):
18:         pass
19:
20:     def setBrightness(self, brightness):
21:         pass
22:
23:     def setPixelColor(self, n, color):
24:         pass
25:
26:     def setPixelColorRGB(self, n, red,
green, blue):
27:         pass
28:
29:     def show(self):
30:         pass
31:
32: def Color(red, green, blue, white=0):
33:     pass
```

Nous voyons qu'il y a des paramètres que nous conserverons en mémoire à titre informatif, mais qui ne seront pas employés (par exemple la fréquence **freq\_hz** ne sera pas utile pour l'émulateur).

## Le module neopixel

Pour pouvoir manipuler ce module, il a bien entendu fallu l'installer. Il est compris dans le projet **rpi\_ws281x** disponible sur GitHub. Voici les commandes permettant de l'installer :

```
$ sudo apt install build-essential git
scons python-dev swig
```

Il faut ensuite cloner le projet et le compiler :

```
$ git clone https://github.com/jgarff/
rpi_ws281x
$ cd rpi_ws281x
rpi_ws281x$ sudo scons
```

Pour finir, il faut installer le module Python :

```
rpi_ws281x$ cd python
rpi_ws281x/python$ sudo python3 setup.py
install
```

## 2. CONSTRUCTION DE L'ÉMULATEUR

Pour le code de nos fonctions, nous nous inspirerons bien entendu du fichier **neopixel.py** originel [2] et de son fonctionnement. Cela implique des allers-retours entre les tests sur écran de leds et sur l'émulateur de manière à obtenir le même comportement que ce soit pour éclairer les leds ou pour afficher des messages d'erreur. Notre objectif sera de faire fonctionner le programme d'exemple **strandtest.py** fourni avec le projet **rpi\_ws281x**.

Nous allons avoir une autre problématique au niveau de l'émulateur : sur l'écran de leds on peut éclairer ou éteindre des leds et il va falloir les représenter sur le moniteur. Pour cela, deux solutions : soit employer une interface graphique du type **Qt** nécessitant l'emploi de **Qthreads** et de signaux pour communiquer (l'interface est bloquante), soit utiliser quelque chose de plus simple comme **matplotlib**. J'ai choisi la solution de facilité...

Maintenant se pose une question essentielle : par où commencer ? Comme je l'ai dit précédemment, nous allons

faire des allers-retours entre l'utilisation du module original et l'écriture de l'émulateur. Par exemple, que se passe-t-il si nous créons une instance de **Adafruit\_NeoPixel** et que nous ne faisons pas appel à la méthode **begin()** ?

```
01: from neopixel import Adafruit_NeoPixel,
Color
02:
03: LED_COUNT = 64
...
11: if __name__ == '__main__':
12:     strip = Adafruit_NeoPixel(LED_COUNT,
LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT, LED_
BRIGHTNESS)
13:     strip.setPixelColor(0, Color(255, 0, 0))
14:     strip.show()
```

Nous obtenons simplement un code d'erreur :

```
$ sudo python3 neopixel_test.py
le shell a retourné 139
```

Le résultat est le même en commentant la ligne 13 ou la ligne 14. donc si la méthode **begin()** n'a pas été invoquée, un appel à **setPixelColor()** ou à **show()** provoque une erreur **139**. De manière plus générale en fait, l'absence d'appel à **begin()** en ayant créé une instance de **Adafruit\_NeoPixel** provoque une erreur, et un appel à **begin()** après avoir utilisé une autre méthode également. Nous pouvons donc en déduire le code suivant :

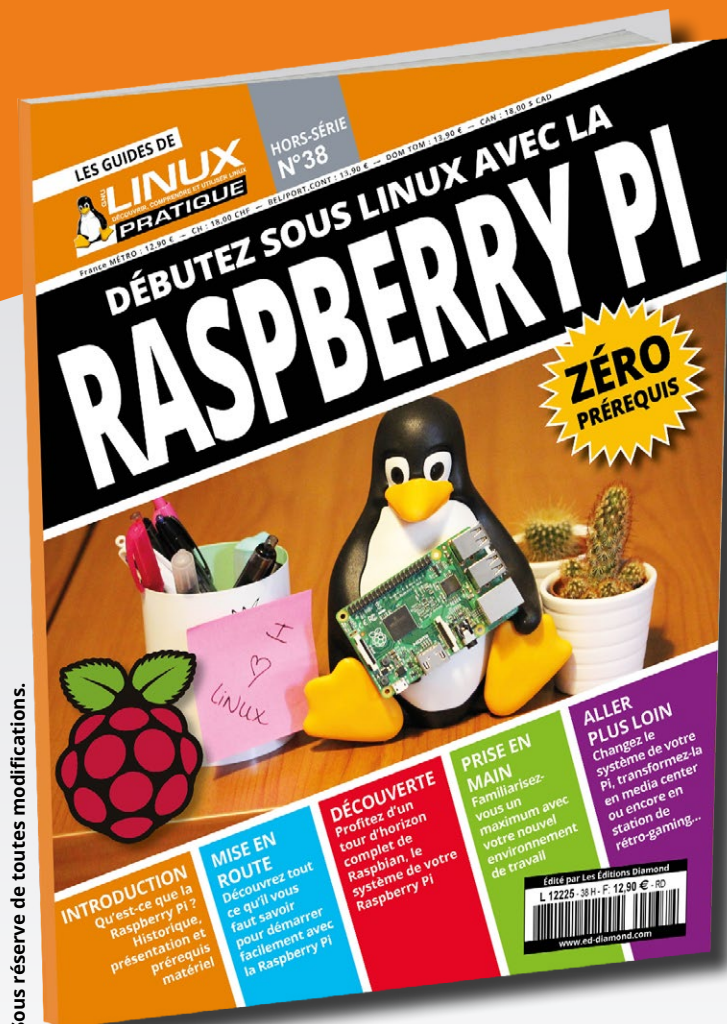
```
01: class Adafruit_NeoPixel:
02:     def __del__(self):
03:         self._beginTestError()
04:
05:     def __init__(self, num, pin, freq hz=
800000, dma=5, invert=False, brightness=255,
channel=0, strip_type=1050624):
06:         self._begin = False
07:
08:     def _beginTestError(self):
09:         if not self._begin:
10:             exit(139)
11:
12:     def begin(self):
13:         self._begin = True
...
```

J'ai défini une méthode privée **\_beginTestError()** qui se charge de vérifier que l'attribut **\_begin** est passé à **True** (que la méthode **begin()** a été appelée puisque l'on modifie justement **\_begin** en ligne 13). J'appelle ensuite cette méthode dans toutes les autres méthodes, interdisant leur exécution si **begin()** n'a pas été appelé.

Nous pouvons maintenant définir le constructeur. Ceci ne représente pas de grandes difficultés puisque nous n'utilisons pas la majorité des paramètres ; nous utiliserons simplement le nombre de leds pour initialiser une liste. Du coup

# DISPONIBLE DÈS LE 24 MARS !

## LINUX PRATIQUE HORS-SÉRIE n°38



Sous réserve de toutes modifications.

# DÉBUTEZ SOUS LINUX AVEC LA RASPBERRY PI

NE LE MANQUEZ PAS  
CHEZ VOTRE MARCHAND  
DE JOURNAUX ET SUR :



<http://www.ed-diamond.com>

nous pouvons écrire le code de toutes les méthodes en lien avec cette liste :

```

01: class Adafruit_NeoPixel:
...
05:     def __init__(self, num, pin,
freq_hz=800000, dma=5, invert=False,
brightness=255, channel=0, strip_
type=1050624):
06:         self._begin = False
07:         self._leds = [0] * num
08:         self._pin = pin
09:         self._freq_hz = freq_hz
10:         self._dma = dma
11:         self._invert = invert
12:         self._brightness = brightness
13:         self._channel = channel
14:         self._strip_type = strip_type
...
23:     def getPixelColor(self, n):
24:         self._beginTestError()
25:         return self._leds[n]
26:
27:     def getPixels(self):
28:         self._beginTestError()
29:         return self._leds
30:
31:     def numPixels(self):
32:         self._beginTestError()
33:         return len(self._leds)
34:
35:     def setBrightness(self, brightness):
36:         self._beginTestError()
37:         self._brightness = brightness
38:
39:     def setPixelColor(self, n, color):
40:         self._beginTestError()
41:         if n < self.numPixels():
42:             self._leds[n] = color
43:
44:     def setPixelColorRGB(self, n, red,
green, blue):
45:         self._beginTestError()
46:         if n < self.numPixels():
47:             self._leds[n] = Color(red,
green, blue)
...

```

Dans `_leds` nous stockerons une liste des couleurs de chaque led du ruban (ligne 7). La définition des autres méthodes se fait alors très simplement en travaillant sur cette liste (notez toutefois les tests des lignes 41 et 46 permettant de s'assurer que l'on ne tente pas d'indiquer une couleur pour une led hors de l'écran).

Notre émulateur est un peu plus spécialisé que le module initial, car il s'attache à émuler des écrans de leds ayant une taille de  $n \times n$  leds. Nous pouvons donc ajouter un test dans le constructeur pour nous assurer que la taille spécifiée par l'utilisateur est correcte :

```

...
05:     def __init__(self, num, pin, freq_
hz=800000, dma=5, invert=False, brightness=255,
channel=0, strip_type=1050624):
06:         self._begin = False
07:         sqrtNum = int(math.sqrt(num))
08:         if sqrtNum ** 2 != num:
09:             exit(1)
...

```

Certes, il est un peu « barbare » de sortir du programme comme ça, simplement avec un code d'erreur, mais j'ai adopté la même politique que dans le module original...

Pour la fonction `Color()`, nous garderons le code initial sans modification :

```

def Color(red, green, blue, white=0):
    return (white << 24) | (red << 16) |
        (green << 8) | blue

```

Par contre, nous allons avoir besoin de la fonction inverse, capable de recevoir une couleur en 24 bits et de retourner les couleurs sous la forme d'un tuple de valeurs RGB normalisées (comprises entre 0 et 1) :

```

def _invColor(color):
    blue = color & 255
    green = (color >> 8) & 255
    red = (color >> 16) & 255
    return (red / 255, green / 255, blue / 255)

```

Pour l'aspect graphique de l'émulateur, nous allons déclencher l'affichage de la représentation de l'écran lors de l'appel à `begin()` :

```

...
04: import matplotlib.pyplot as plt
05:
06:
07: class Adafruit_NeoPixel:
...
32:     def begin(self):
33:         self._begin = True
34:         self._fig = plt.gcf()
35:         self._fig.canvas.set_window_
title('WS2812B Matrix Emulator')
36:
37:         plt.axis([-1, self._cols, -1,
self._rows])
38:         plt.axis('off')
39:         plt.ion()
40:         for row in range(self._rows):
41:             for col in range(self._cols):
42:                 plt.plot(row, col,
marker='s', markersize=180//self._cols,
color='black')
43:         plt.show()

```



# ACTUELLEMENT DISPONIBLE! MISC n°90

Il faut bien entendu importer **matplotlib** (ligne 4) avant de pouvoir l'utiliser dans la méthode **begin()**. En ligne 34, nous stockons une référence à la figure courante dans l'attribut **self.\_fig** avant de donner un titre à la fenêtre (ligne 35). Nous définissons ensuite les axes en indiquant leurs indices minimum et maximum sur les abscisses et les ordonnées (ligne 37) puis nous les cachons (ligne 39). Pour centrer la figure lors de l'affichage d'un écran de  $n \times n$  leds, nous avons besoin d'axes partant de **-1** jusqu'à **n** sur les deux axes (puisque l'écran est carré...).

L'appel à **plt.ion()** en ligne 39 est très important puisque c'est lui qui place matplotlib en mode interactif nous permettant de conserver une fenêtre graphique non bloquante.

## NOTE

Les interfaces graphiques sont bloquantes vis-à-vis du programme qui les lance, car elles utilisent une boucle infinie permettant de détecter les événements (déplacement souris, clic, etc.).

La boucle des lignes 40 à 42 va afficher les  $n \times n$  points représentant les leds (les **self.\_rows** x **self.\_cols** points pour être plus précis). Ces points seront de forme carrée (**marker='s'** où **s** signifie *square*), de couleur noire (**color='black'**) puisque représentant des leds éteintes et d'une taille adaptée (arbitrairement) au nombre de leds de l'écran (**markersize=180//self.\_cols**). Enfin, en ligne 43 l'appel à **plt.show()** permet d'afficher la fenêtre graphique (non bloquante du fait de l'appel précédent à **plt.ion()**).

Une fois que la représentation de l'écran de leds est affichée, c'est la méthode **show()** - de l'émulateur, à ne pas confondre avec celle de matplotlib - qui effectuera sa mise à jour :

```
...
76:     def show(self):
77:         self._beginTestError()
78:         plt.clf()
79:         plt.axis([-1, self._cols, -1, self._
rows])
80:         plt.axis('off')
81:         for pixel in range(len(self._leds)):
82:             row, col = self._
lineToMatrix(pixel)
83:             plt.plot(row, col, marker='s',
markersize=180//self._cols, color=
invColor(self._leds[pixel]))
84:             plt.pause(0.0001)
...
```

La ligne 78, ô combien importante, efface la figure. Pourquoi est-elle si importante ? Essayez de la commenter et vous comprendrez immédiatement : l'affichage est de plus en plus lent ! Ce problème m'a fait longuement rechercher



# TELEGRAM, SIGNAL, WHATSAPP QUELLE CONFIANCE LEUR ACCORDER ?

NE LE MANQUEZ PAS  
CHEZ VOTRE MARCHAND  
DE JOURNAUX ET SUR :

<http://www.ed-diamond.com>



son origine, pensant que c'était l'affichage qui ralentissait le processus alors que l'on sature « simplement » le CPU qui doit afficher à chaque étape un graphe de plus qu'à l'étape précédente (et comme matplotlib n'est pas non plus connu pour sa rapidité...).

Les lignes 79 à 83 ont déjà été vues dans `begin()` à la différence près qu'ici nous spécifions un code couleur en RGB normalisé grâce à la fonction `_invColor()` écrite précédemment. La ligne 84 effectue une micro-pause qui déclenche la mise à jour de l'affichage.

En testant l'émulateur sur le programme `strandtest.py`, on obtient bien une simulation de l'affichage des leds comme le montre la figure 1 (ici le paramètre `LED_COUNT` a été modifié avec une valeur de `64`).

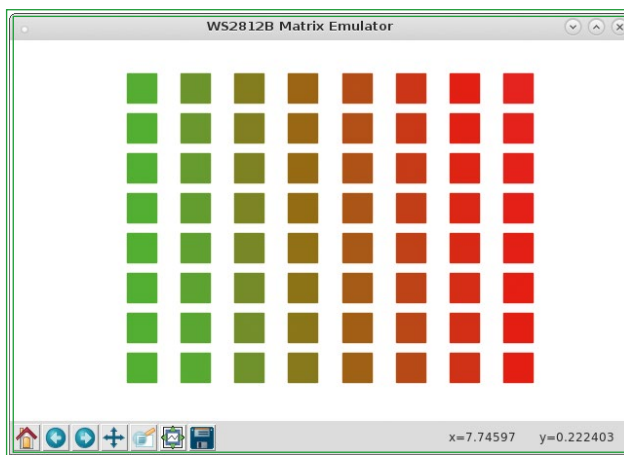


Fig. 1 : L'émulateur exécutant le programme de test `strandtest.py`.

### 3. AMÉLIORATION

Cette première version est fonctionnelle... mais elle n'est pas vraiment rapide. Étant bloqué dans ma logique de boucle événementielle je n'ai pas pensé à regarder ce que l'on pourrait faire avec **PyGame**. Je remercie donc Denis Bodor qui m'a fait découvrir la méthode magique `display.flip()` permettant de rafraîchir la fenêtre graphique. La logique reste la même, il ne faut effectuer que quelques modifications sur le code :

```
001: import pygame
...
006: class Adafruit_NeoPixel:
...
010:     def __init__(self, num, pin, freq_
hz=800000, dma=5, invert=False,
011:                 brightness=255, channel=0,
strip_type=1050624, led_size=30):
```

```
...
033:     # Graphic parameters
034:     # pixel size
035:     self._psize = led_size
036:     # gap between pixel
037:     self._pgap = int(self._psize / 4)
038:     # margins
039:     self._marg = self._psize
040:     # pixels
041:     self._pixel = []
042:     # screen
043:     self._screen = None
044:     # draw surface
045:     self._surface = None
046:     self._width = self._marg +
(self._cols * (self._psize + self._pgap)) + \
047:         (self._marg - self._pgap)
048:     self._height = self._marg +
(self._rows * (self._psize + self._pgap)) + \
049:         (self._marg - self._pgap)
...
053:     def begin(self):
054:         self._begin = True
055:         pygame.init()
056:         pygame.display.set_caption('WS2812
Matrix Emulator')
057:         self._screen = pygame.display.
set_mode((self._width, self._height))
058:         self._screen.fill((255, 255, 255))
059:         self._surface = pygame.display.
get_surface()
060:
061:         for row in range(self._rows):
062:             for col in range(self._cols):
063:                 self._pixel.append(pygame.
Rect(self._marg + (col * (self._psize + self._
pgap)), self._marg + (row * (self._psize +
self._pgap)), self._psize, self._psize))
064:                 self._surface.fill((0, 0,
0), self._pixel[(row * self._rows) + col])
065:                 pygame.display.flip()
...
099:     def show(self):
100:         self._beginTestError()
101:         pygame.display.flip()
102:         for pix in range(len(self._
leds)):
103:             self._surface.fill(_
invColor(self._leds[pix], self._pixel[pix])
104:         pygame.display.flip()
105:         for event in pygame.event.
get(pygame.QUIT):
106:             exit(0)
...
113:     def _invColor(color):
114:         blue = color & 255
115:         green = (color >> 8) & 255
116:         red = (color >> 16) & 255
117:         return (red, green, blue)
```

En dehors des nouveaux paramètres graphiques des lignes 46 à 49 (taille de pixel, espacement, marge et liste de pixels), les attributs `self._width` et `self._height` sont calculés pour déterminer la taille de la fenêtre graphique.

Dans la méthode `begin()`, il faut initialiser le mode graphique (ligne 55), donner un titre à la fenêtre (ligne 56) et définir la fenêtre (taille et couleur de fond dans les lignes 57 et 58). En ligne 59, nous définissons un nouvel attribut qui permettra d'accéder à l'élément graphique. Nous ajoutons ensuite des objets de type `Rect` dans la liste `_pixel` (ligne 63). Ces objets représentent les leds qui sont affichées en noir dans un premier temps (ligne 64). En ligne 65, se trouve la fameuse instruction de rafraîchissement du contenu de la fenêtre graphique.

Dans la méthode `show()`, entre deux appels à `display.flip()` (lignes 101 et 104), nous affichons les leds dans la couleur stockée par `self._leds` (ligne 103). Notez que la sortie de `_invColor()` n'a plus besoin d'être normalisée ici (voir ligne 117). Enfin, les lignes 105 et 106 permettent d'autoriser la sortie du programme en cliquant sur le bouton `close` de la fenêtre.

La figure 2 ne montre pas le gain de rapidité, vous devez donc me faire confiance (à moins de tester par vous-même).

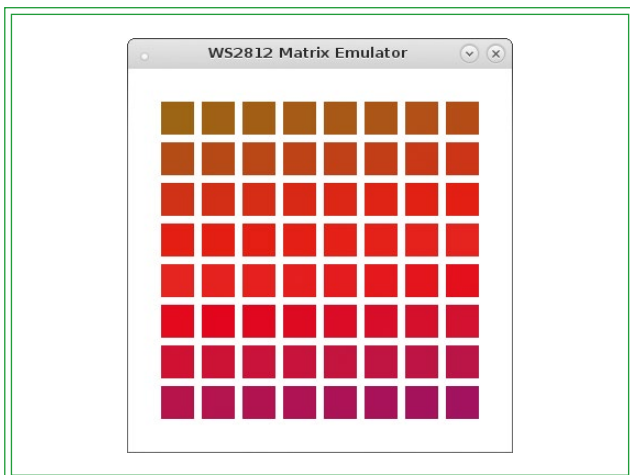


Fig. 2 : L'émulateur (PyGame) exécutant le programme de `test_strandtest.py`.

## CONCLUSION

L'objectif est atteint : en partant du code du module `neopixel.py` nous avons pu réaliser un module du même nom capable de se substituer de manière transparente au module original. Il suffit alors de placer ou non le fichier `neopixel.py` de l'émulateur dans le répertoire d'un projet pour obtenir un affichage à l'écran ou sur matériel. La difficulté principale concernait la possibilité de disposer d'un affichage graphique non bloquant, ce qui a été réalisé grâce au mode interactif de matplotlib puis grâce à PyGame. Cet émulateur peut bien entendu être amélioré et n'attend que vos contributions... ■

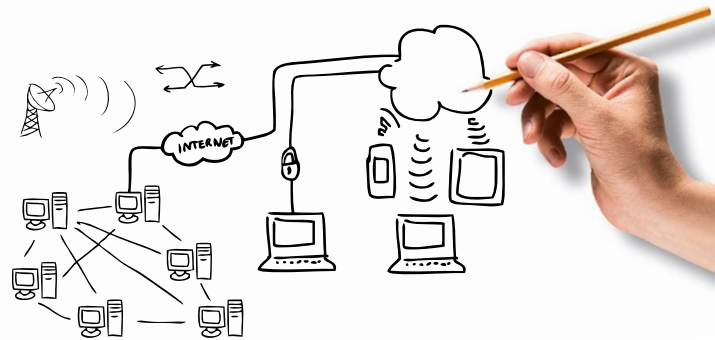
## Code source sur GitHub

Ce projet devant normalement continuer à évoluer dans le futur, il dispose de son propre dépôt GitHub que vous trouverez sur <https://github.com/Hackable-magazine/vrtneopixel>.

## RÉFÉRENCES

- [1] Module neopixel : [https://github.com/jgarff/rpi\\_ws281x](https://github.com/jgarff/rpi_ws281x)
- [2] Code source du module neopixel : [https://github.com/jgarff/rpi\\_ws281x/blob/master/python/neopixel.py](https://github.com/jgarff/rpi_ws281x/blob/master/python/neopixel.py)
- [3] BODOR D., « WS2812 : la led intelligente », *Hackable Magazine n°6*, mai-juin 2015, p. 8 à 19.

# LICENCE PRO RÉSEAUX ET TÉLÉCOMS

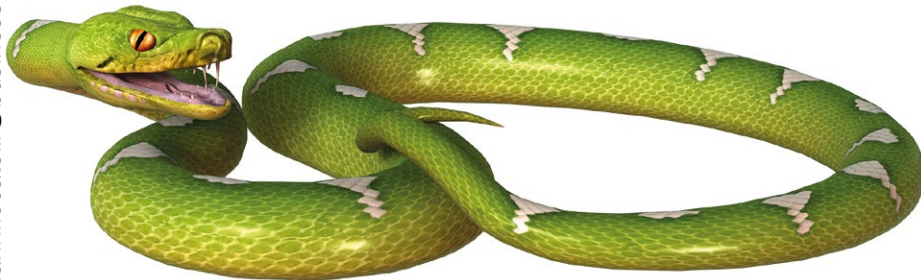


UN BAC+3 QUI  
**AIME**  
L'OPEN-SOURCE  
À L'IUT DE BÉZIERS

# RIEN NE VAUT LA PRATIQUE : CRÉATION D'UN PAQUETAGE POUR PIP

TRISTAN COLOMBO

MOTS-CLÉS : PYTHON, PAQUETAGE, PIP, DISTRIBUTION, PYPI



La documentation permettant de créer un paquetage pour un projet Python est abondante et décrit une procédure relativement simple. Pourtant, lorsque l'on passe à la pratique on en arrive à se demander si les différents auteurs ont réellement créé et mis à disposition un paquetage...

Un paquetage **Python** permet de distribuer son projet et de toucher un maximum d'utilisateurs. En effet, tester un module en l'installant grâce à **pip** (à l'intérieur d'un *virtualenv*) ne prendra que quelques secondes alors que s'il faut aller chercher les sources et les compiler,

il y a de forts risques qu'à la première erreur nous stoppions tous nos efforts.

Il est donc important de pouvoir empaqueter un projet et le distribuer sur **PyPi** et le point de départ de cette « aventure » sera la page <https://packaging.python.org/distributing>. Car ne vous y trompez pas,

il s'agit bien d'une aventure, d'une chasse au trésor où disposant d'une carte précise vous pensiez pouvoir rafler le magot en suivant précisément les instructions... mais personne ne vous avait parlé de l'hydre à deux têtes gardant la salle des coffres ! J'ai récemment mis en ligne un projet d'émulateur d'écran de leds [1] nommé initialement **vrt\_neopixel**. Voici le récit des opérations permettant de distribuer un projet très simple...

## 1. OÙ TOUT COMMENCE PLUTÔT BIEN

Au départ il y a bien entendu un projet. Ici il s'agit d'un tout petit projet comportant deux fichiers : le module principal **vrt\_neopixel.py** et un fichier d'exemple **strandtest.py**. En suivant la documentation officielle, on nous indique que nous allons avoir besoin de **twine**, une suite d'outils permettant d'interagir avec PyPi. Nous installons donc sereinement ledit paquetage :

```
$ sudo pip3 install twine
```

Il faut ensuite utiliser **setuptools** et pour cela définir un certain nombre de fichiers :

- **setup.py** qui permet de configurer le projet. C'est aussi le point d'entrée du système en ligne de commandes permettant d'effectuer différentes tâches ;
- **setup.cfg** qui est un fichier de configuration ini contenant les valeurs par défaut qui seront utilisées par **setup.py** ;
- **MANIFEST.in** qui permet d'ajouter manuellement des fichiers qui ne seraient pas automatiquement intégrés par le processus standard (les fichiers non-Python) ;
- **LICENSE.txt** qui contient le texte relatif à la licence choisie ;
- **README.rst** qui est le fichier de description du projet (celui affiché par PyPi).

Qui dit augmentation du nombre de fichiers, dit structuration du répertoire du projet. Pour suivre la convention préconisée dans la documentation, nous allons créer un répertoire pour notre projet, y inclure les fichiers de configuration de setuptools et créer un autre répertoire du même nom que le premier contenant les fichiers sources du projet. Voilà à quoi nous aboutissons pour le projet `vrt_neopixel` :

```
$ tree vrt_neopixel
vrt_neopixel
├── LICENSE.txt
├── MANIFEST.in
├── README.rst
├── setup.py
└── vrt_neopixel
    ├── sample
    │   └── strandtest.py
    └── vrt_neopixel.py
```

Maintenant il faut remplir ces fichiers...

## 1.1 Le fichier setup.py

Pour écrire le **setup.py**, un bon point de départ est le fichier **setup.py** fourni dans le projet d'exemple : <https://github.com/pypa/sampleproject>. Voici les éléments essentiels à indiquer :

```
01: from setuptools import setup, find_packages
02: from codecs import open
03: from os import path
04:
05: here = path.abspath(path.dirname(__file__))
06:
07: with open(path.join(here, 'README.rst'),
08:           encoding='utf-8') as f:
09:     long_description = f.read()
```

Nous récupérons la description longue du projet directement depuis le fichier **README.rst** et nous la stockons dans la variable **long\_description**.

La configuration du projet se fait ensuite lors de l'appel de la fonction **setup()** :

```
10: setup(
11:     name='vrt_neopixel',
```

Il faut bien entendu indiquer le nom du projet (dans **name**).

```
13:     packages=['vrtneopixel'],
```

La liste des répertoires contenant des éléments relatifs au projet est donnée dans **packages**. On peut lister ces répertoires manuellement (comme c'est le cas ici en ligne 13) ou bien utiliser la fonction **find\_packages()** de **setuptools** (importée en ligne 1, mais inutilisée ici). Cette fonction trouve automatiquement les répertoires contenant des fichiers Python et accepte un paramètre **exclude** pour indiquer les répertoires à ne pas scanner (exemple : **packages=find\_packages(exclude=['doc','test\*'])**).

```
15:     version='1.0.11',
```

Il faut également indiquer un numéro de version en se conformant au format défini dans le **PEP440** [2].

```
17:     description='A WS2812 matrix emulator',
18:     long_description=long_description,
```

Nous donnons ensuite une description courte (ligne 17) et longue (ligne 18) à notre projet. La description longue est issue de notre lecture du **README.rst** des lignes 5 à 8.

```
20:     url='https://github.com/Hackable-Magazine/
21:         vrt_neopixel',
```

Le paramètre **url** permet d'indiquer l'adresse où se trouvent les sources.

```
22:     author='Tristan Colombo',
23:     author_email='tristan@gnulinuxmag.com',
```

Nous précisons l'auteur du projet et son adresse de contact dans les paramètres **author** et **author\_email**.

```
25:     license='GPLv3+',
```

Il faut bien entendu choisir une licence.

```
27:     classifiers=[
28:         'Development Status :: 5 - Production/
29:         Stable',
30:         'Intended Audience :: Developers',
31:         'Topic :: System :: Emulators',
32:         'License :: OSI Approved :: GNU
33:         General Public License v3 or later (GPLv3+)',
34:         'Programming Language :: Python :: 3',
35:     ],
36:     1,
```

Le paramètre **classifiers** est le plus long à compléter, car les chaînes de caractères utilisables sont listées sur la page [https://pypi.python.org/pypi?%3Aaction=list\\_classifiers](https://pypi.python.org/pypi?%3Aaction=list_classifiers). On va indiquer ici quel est le stade de développement du projet (*Development Status*), à quel public on s'adresse (*Intended Audience*), dans quelle catégorie on classe le projet (*Topic*), à nouveau quelle est la licence utilisée (*License*), et avec quelle(s) version(s) de Python le projet est compatible (*Programming Language*).

```
38:     keywords='emulator ws2812 neopixel rpi_
ws281x RaspberryPi',
```

Nous indiquons ensuite une liste de mots-clés permettant de retrouver notre projet.

```
40:     install_requires=['matplotlib'],
41: )
```

Pour finir, nous listons les dépendances de notre projet. Ici nous n'avons besoin que de **matplotlib** sans nécessité d'une version minimale (sinon il était possible de l'indiquer par '**paquetage>=x.y.z**').

## 1.2 Le fichier LICENSE.txt

Pour le fichier **LICENSE.txt**, il n'y a rien de bien compliqué puisqu'il suffit de rechercher sur Internet le fichier texte correspondant à la licence choisie. Comme ici il s'agit de la GPLv3, on peut la récupérer sur <https://www.gnu.org/licenses/gpl-3.0.txt>.

## 1.3 Le fichier README.rst

Le fichier **README.rst** de présentation du projet est, comme son extension l'indique, un fichier au format *ReStructuredText*. Il contient la description du projet dont voici un extrait :

```
01: Émulateur d'écran WS2812
02: =====
03:
04: Ce module peut être utilisé pour émuler un
05: écran de leds WS2812 Matrix en
06: remplacement du module 'neopixel.py' pour
07: Raspberry Pi fourni par le projet
08: **rpi_ws281x** (<https://github.com/jgarff/rpi\_ws281x>).
09: ...
```

## 1.4 Le fichier MANIFEST.in

Le fichier **MANIFEST.in** n'est en fait qu'une liste de fichiers non Python à intégrer au projet. Il faut faire précéder le nom des fichiers du mot-clé **include** :

```
01: include LICENSE.txt
02: include README.rst
```

## 2. OÙ ÇA COMMENCE À DÉRAPER

Normalement nous avons fait le plus dur du travail. Il ne reste plus qu'à générer les sources de la distribution et à tout transférer sur PyPi. Dans un excès de confiance devant la relative facilité de mise en place du processus j'ai décidé de mettre directement mon projet sur PyPi... erreur ! Voici donc comment faire les choses correctement.

Tout d'abord, il existe deux sites PyPi : le « classique » et la version de test sur <https://testpypi.python.org/pypi>. Pour nos premiers essais, nous utiliserons donc fort logiquement testpypi.

### 2.1 Création d'un compte

Pour pouvoir utiliser testpypi, il faut disposer d'un compte utilisateur. Rendez-vous sur [https://pypi.python.org/pypi?%3Aaction=register\\_form](https://pypi.python.org/pypi?%3Aaction=register_form) et remplissez le formulaire. À la suite de la validation de ce dernier, vous recevrez un mail permettant de confirmer la création du compte. Notez bien votre identifiant et votre mot de passe (dans la suite de l'article, nous considérerons qu'il s'agit de **mon\_login/motP4sse**).

### 2.2 Le fichier de configuration pour l'accès à testpypi

Créez un fichier **~/.pypirc** contenant les lignes suivantes (en remplaçant bien entendu les identifiants de connexion par les vôtres) :

```
01: [distutils]
02: index-servers=
03:     testpypi
04:
05: [testpypi]
06: repository = https://testpypi.python.
07: org/pypi
08: username = mon login
09: password = motP4sse
```

**NOTE** : Plus tard, si on utilise twine avec l'option **-p motP4sse**, il est possible de supprimer la ligne du champ **password** de **~/.pypirc**. Mais comme vous le verrez dans la suite, ce n'est pas forcément une solution opérationnelle...

### 2.3 On teste sur testpypi

Depuis le répertoire du projet (celui qui contient le **setup.py**), on lance :

```
$ python3 setup.py register -r testpypi
running register
```

```

running egg info
creating vrtneopixel.egg-info
writing top-level names to vrt_neopixel.egg-info/top_level.txt
writing requirements to vrt_neopixel.egg-info/requirements.txt
writing dependency_links to vrt_neopixel.egg-info/dependency_links.txt
writing vrt_neopixel.egg-info/PKG-INFO
writing manifest file 'vrt_neopixel.egg-info/SOURCES.txt'
reading manifest file 'vrt_neopixel.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'vrt_neopixel.egg-info/SOURCES.txt'
running check
Registering vrt_neopixel to https://testpypi.python.org/pypi
Server response (200): OK

```

Comme vous le voyez, la commande a enregistré le projet sur testpypi et a créé un certain nombre de fichiers dans le nouveau répertoire **vrt\_neopixel.egg-info**. On y trouve notamment un fichier **PKG-INFO** qui résume les informations sur le paquetage (souvenez-vous de ce fichier, nous allons en avoir besoin).

Si le résultat de la commande précédente est **OK**, c'est que vos fichiers de configuration sont corrects et vous pouvez poursuivre en lançant la commande qui va construire la distribution source et qui va la transférer sur testpypi :

```

$ python3 setup.py sdist upload -r testpypi
...
running upload
Submitting dist/vrt_neopixel-1.0.11.tar.gz
to https://testpypi.python.org/pypi
Server response (200): OK

```

Une réponse **OK** indique que votre paquet a été correctement transféré sur testpypi.

**NOTE** : Si vous obtenez un message d'erreur du type **HTTP Error 400: A file named "vrt\_neopixel-1.0.11.tar.gz" already exists for vrt\_neopixel-1.0.11. To fix problems with that file you should create a new release.**, c'est que vous avez déjà transféré votre projet sur testpypi (ici en version 1.0.11). La solution consiste à incrémenter la version dans le fichier **setup.py** puis à relancer la commande.

## 2.4 On essaye l'installation via pip3

Notre paquet étant disponible sur testpypi, nous allons essayer de l'installer comme tout utilisateur grâce à **pip3** :

```

$ sudo pip3 install -i https://testpypi.python.org/pypi vrt_neopixel
...
Successfully built vrt_neopixel
Installing collected packages: vrt_neopixel
Successfully installed vrt-neopixel-1.0.11

```

**NOTE** : Pensez à créer un virtualenv pour effectuer vos tests...

Et voilà ! Normalement, il suffit de tester notre paquet depuis un script shell et à le publier sur PyPi (pas celui de test).

## 3. OÙ LES PROBLÈMES S'ACCUMULENT

Lançons un shell Python pour tester notre module :

```

$ python3
>>> import vrt_neopixel
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'vrt_neopixel'

```

Il y a visiblement un problème...

### 3.1 Le choix du nom du paquetage

Si l'on se remémore le message de fin d'installation via **pip3**, nous pouvons avoir une piste :

```

...
Successfully installed vrt-neopixel-1.0.11

```

Ainsi notre paquet aurait été renommé en **vrt-neopixel** au lieu de **vrt\_neopixel**... Cela demande vérification ! Ouvrons le fichier **vrt\_neopixel.egg-info/PKG-INFO** :

```

01: Metadata-Version: 1.1
02: Name: vrt-neopixel
03: Version: 1.0.11
...

```

Eh oui ! Apparemment setuptools n'aime pas les caractères underscore ! Il s'agit d'un problème de convention de nommage entre setuptools, distutils et PyPi. setuptools renomme donc les noms contenant des underscores ; nous n'avons pas besoin de lancer plus d'investigations sur le sujet : il faut changer de nom de paquet. Pour cela il faut :

- renommer tous les fichiers et répertoires **vrt\_neopixel** en **vrtneopixel** ;
- modifier le paramètre **name** du **setup()** de **setup.py** en **'vrtneopixel'** et incrémenter la version.

La mise à jour du projet se fait par :

```

$ python3 setup.py sdist upload -r testpypi

```

Ensuite il ne faut pas oublier de désinstaller puis réinstaller le paquet :

```
$ sudo pip3 uninstall vrt neopixel
$ sudo pip3 install -i https://testpypi.
python.org/pypi vrtneopixel
```

**NOTE :** Lorsque le nom du paquet ne change pas en cours de développement (ce qui est normalement le cas la plupart du temps), il est inutile de procéder à une désinstallation, une simple mise à jour suffit :

```
$ sudo pip3 install --upgrade -i https://
testpypi.python.org/pypi vrtneopixel
```

## 3.2 L'accès aux fichiers Python

Refaisons un test :

```
$ python3
>>> import vrtneopixel
>>>
```

Ça y est, ça à l'air de fonctionner ! Le module **vrtneopixel** contient une fonction **Color()**, voyons si nous pouvons l'appeler :

```
>>> vrtneopixel.Color(255, 56, 23)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'vrtneopixel' has no
attribute 'Color'
```

Aïe ! Encore un problème ! Mais que contient donc ce module ?

```
>>> dir(vrtneopixel)
['_doc_', '_loader_', '_name_', '_
package_', '_path_', '_spec_']
```

Comme ça la réponse est claire : rien ! Nous n'avons visiblement pas eu accès au fichier **vrtneopixel.py** du répertoire **vrtneopixel**. D'un autre côté, en Python ne faut-il pas un fichier **\_\_init\_\_.py** pour accéder à des fichiers Python dans un sous-répertoire ? Ajoutons donc ce fichier (vide) pour obtenir l'arborescence suivante :

```
$ tree vrtneopixel
vrtneopixel/
├── LICENSE.txt
├── MANIFEST.in
├── README.rst
├── setup.py
├── vrtneopixel
│   ├── __init__.py
│   ├── sample
│   └── strandtest.py
└── vrtneopixel.py
```

Modifions le numéro de version dans **setup.py** et transférons le nouveau code sur testpypi avant de mettre à jour notre installation :

```
$ python3 setup.py sdist upload -r testpypi
$ sudo pip3 install --upgrade -i https://
testpypi.python.org/pypi vrtneopixel
```

Testons à nouveau :

```
>>> import vrtneopixel
>>> vrtneopixel.Color(22,22,22)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'vrtneopixel' has
no attribute 'Color'
```

Ah, non ! Ça recommence ! Pourtant nous avons bien accès au sous-répertoire... Et si justement la solution était là ? Pour accéder à des sous-répertoires lors de l'import, on utilise des points pour « descendre » dans l'arborescence. Ici nous devons rentrer dans un répertoire **vrtneopixel**. Testons notre hypothèse :

```
>>> vrtneopixel.vrtneopixel.
Color(22,22,22)
1447446
```

C'est ça ! Il s'agit d'un problème d'accès à l'information ; notre paquetage est fonctionnel !

## 4. OÙ L'ON PARVIENT ENFIN À NOTRE OBJECTIF

Pour pouvoir accéder directement aux éléments du module **vrtneopixel.py** (en l'occurrence nous voulons pouvoir utiliser la fonction **Color()** et la classe **Adafruit\_NeoPixel**), il faut l'indiquer dans le fichier **\_\_init\_\_.py** :

```
01: from .vrtneopixel import Adafruit_
NeoPixel, Color
```

Notez la présence du point dans **.vrtneopixel** qui indique que nous travaillons sur le fichier **vrtneopixel.py** du répertoire courant.

```
$ python 3
>>> import vrtneopixel
>>> vrtneopixel.Color(22,22,22)
1447446
```

Enfin ! Tout fonctionne ! Nous pouvons maintenant transférer notre projet sur PyPi.

**NOTE :** Si vous souhaitez faire place nette sur testpypi, rendez-vous sur <https://testpypi.python.org> et connectez-vous. En haut et à droite de la page se trouve un cadre avec la liste de vos paquetages. Cliquez sur celui que vous souhaitez supprimer et sur la page suivante, tout en bas, cliquez sur le bouton **Remove this project completely**.

N'oubliez pas également de supprimer votre installation locale :

```
$ sudo pip3 uninstall vrtneopixel
```



Pour passer à l'installation sur PyPi il va falloir créer un nouveau compte, cette fois sur <https://pypi.python.org> (le compte sur testpypi est indépendant). La procédure est exactement la même que précédemment. Nous supposons que cette fois nos identifiants sont **logpypi/pwd**.

Il faut ensuite ajouter le site de PyPi dans notre fichier `~/.pypirc` :

```
01: [distutils]
02: index-servers=
03:     pypi
04:     testpypi
05:
06: [pypi]
07: repository = https://upload.pypi.org/legacy/
08: username = logpypi
09: password = pwd
10:
11: [testpypi]
12: repository = https://testpypi.python.org/
13: username = mon_login
14: password = motP4sseE
```

Il faut ensuite générer les sources de la distribution (pensez à changer le numéro de version dans `setup.py`) :

```
$ python3 setup.py sdist
...
creating dist
Creating tar archive
removing 'vrtneopixel-1.0.0' (and everything
under it)
```

Vous pouvez également créer un *build package* (appelé également *wheel*) pour une installation simplifiée pour l'utilisateur final : il s'agit d'un paquetage ne nécessitant pas de compilation depuis les sources à l'installation. Ce n'est pas très utile ici, mais comme on peut le réaliser simplement, faisons-le :

```
$ python3 setup.py bdist_wheel
...
creating build/bdist.linux-x86_64/wheel/
vrtneopixel-1.0.0.dist-info/WHEEL
```

Il ne reste plus qu'à enregistrer le projet sur PyPi et à le transférer. Pour l'enregistrement, la documentation indique une procédure simplifiée à l'aide de twine :

```
$ twine register dist/vrtneopixel-1.0.0-py3-
none-any.whl
Registering package to https://upload.pypi.org/
legacy/
Registering vrtneopixel-1.0.0-py3-none-any.whl
HTTPError: 410 Client Error: This API is no
longer supported, instead simply upload the
file. for url: https://upload.pypi.org/legacy/
```

L'API n'est plus supportée... il serait temps de mettre à jour la documentation officielle !

Nous passerons donc par un enregistrement sur le site en allant sur la page [https://pypi.python.org/pypi?%3Aaction=submit\\_form](https://pypi.python.org/pypi?%3Aaction=submit_form). Notez que le premier choix proposé sur cette page est également twine (qui ne fonctionne pas)... Nous prendrons le troisième choix en fournissant le fichier **PKG-INFO** se trouvant dans `vrtneopixel.egg-info/PKG-INFO` et en cliquant ensuite sur le bouton **Add package info**.

Il ne reste plus que le transfert... et cette fois twine fonctionne :

```
$ twine upload dist/*
Uploading distributions to https://upload.pypi.org/legacy/
Uploading vrtneopixel-1.0.0-py3-none-any.whl
[=====] 10219/10219 - 00:00:02
Uploading vrtneopixel-1.0.0.tar.gz
[=====] 21418/21418 - 00:00:01
```

**NOTE** : Si vous rencontrez des problèmes de conflits lors du transfert sur PyPi (**This filename has previously been used, you should use a different version**), vous pouvez réinitialiser simplement la construction de votre paquet en supprimant trois répertoires :

```
$ rm -R build/ dist/ vrtneopixel.egg-info/
```

Pensez bien entendu à changer aussi le numéro de version.

Votre projet est maintenant accessible par le monde entier :

```
$ sudo pip3 install --upgrade vrtneopixel
```

## CONCLUSION

Nous arrivons au terme de notre parcours censé être une simple balade. Nous avons pu voir que de nombreuses embûches parsemaient la piste et il aurait été bien agréable d'avoir une véritable documentation à jour. En espérant que mon expérience puisse vous servir et vous éviter de perdre du temps... ■

## RÉFÉRENCES

- [1] Code source de vrtneopixel : <https://github.com/Hackable-magazine/vrtneopixel>
- [2] Format des numéros de version (PEP440) : <https://www.python.org/dev/peps/pep-0440/#public-version-identifiers>
- [3] Le format rst : <https://fr.wikipedia.org/wiki/ReStructuredText>

# MOTEUR DE TEMPLATE TWIG : HÉRITAGE ET MACROS



**MARIANA ANDUJAR**

[Ingénieur en informatique à Aix-Marseille-Université]

**MAGALI CONTENSIN**

[Ingénieur en informatique au CNRS]

**MOTS-CLÉS : TWIG, PROGRAMMATION WEB, HÉRITAGE, INCLUSION, MACRO, CSV**



**Le moteur de template open source Twig facilite le développement, la sécurisation et la maintenance d'applications web PHP.**

**L**e moteur de template **Twig** permet de séparer l'affichage, c'est-à-dire la partie qui génère le code **HTML** présenté à l'utilisateur, du code métier **PHP**. L'article précédent (« *Moteur de template Twig : prise en main* », *GNU/Linux Magazine* n°199, pages 44 à 49) a

présenté l'installation et la prise en main de Twig. Cet article permet d'apprendre à utiliser les fonctionnalités avancées du moteur de *template* Twig : l'héritage et les macros. L'objectif est de créer une page web présentant, dans des formats différents, les livres d'une bibliothèque.

## 1. ARCHITECTURE DE L'APPLICATION

Le répertoire **appli** de l'application comportera trois dossiers : **vendor**, **vue** et **modele**. Il est placé à la racine du serveur web.

Pour tester le code, il faut installer le moteur de *template* Twig dans le répertoire **vendor** : <http://twig.sensiolabs.org/>. La version de Twig testée dans ce document est la 1.26.1 du 5 octobre 2016.

Pour exécuter le moteur de *template*, il faut un serveur web **Apache** avec PHP qui ne doit pas être antérieur à la version 5.2.7.

## 2. CRÉATION DES TEMPLATES

Les *templates* sont utilisés pour générer l'interface utilisateur, ils contiennent du code HTML statique ainsi que du code dynamique Twig. Ils sont placés dans le répertoire **vue** de l'application.

Les pages web sont composées de plusieurs parties : un en-tête, un menu, un contenu spécifique et un pied de page. Pour faciliter la lecture des *templates* et leur maintenance, il est possible de séparer les différentes parties dans des *templates* qui seront inclus par le *template* principal.

Dans cette phase, nous allons créer l'en-tête et le menu dans deux fichiers : **header.twig** et **menu.twig**. La figure 1 montre l'emplacement de ces deux blocs dans la page web. L'en-tête affichera un titre HTML de niveau un, dont le contenu sera fixé par les données reçues par le *template*. Les parties dynamiques du code entre délimiteur double accolade seront remplacées par les valeurs des variables **titre\_doc** et **date**. Dans la figure 1, **titre\_doc** est remplacé par le mot bibliothèque et **date** par la date du jour.

```
{# header.twig #}
<header>
  <h1>{{ titre_doc }} - {{ date }}</h1>
</header>
```

Le menu comporte les liens de navigation pour accéder aux différentes parties du site web.

```
{# menu.twig #}
<a href="accueil.php">Accueil</a> |
<a href="liste.php">Liste</a> |
<a href="ajout.php">Ajouter</a> |
<a href="modif.php">Modifier</a> |
<a href="supprimer.php">Supprimer</a>
```

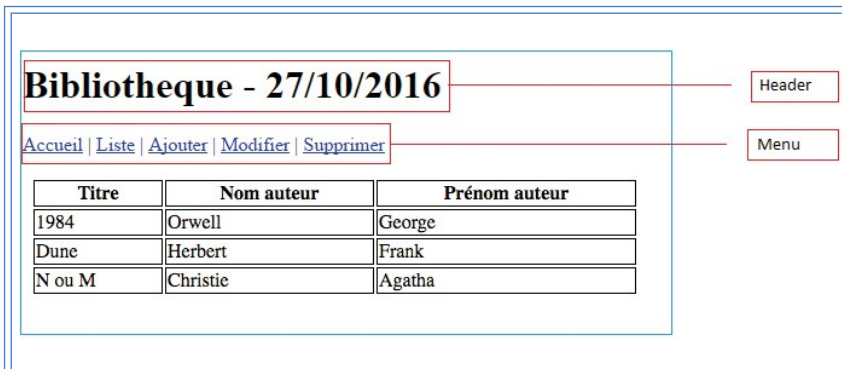


Fig. 1 : Liste des livres.

### 3. MISE EN PAGE GÉNÉRALE

Toutes les pages du site web utilisent la même mise en page générale. Plutôt que de la dupliquer, il est possible de la définir une seule fois, puis de l'utiliser dans tous les *templates*. Le fichier **layout.twig** définit le document HTML : en-tête, corps, sections, barre de navigation. Le titre du document est fixé par la variable **titre\_page**. Les fichiers **header.twig** et **menu.twig** sont inclus grâce à l'instruction **include** placée entre les délimiteurs **{% %}**. Le code contenu dans le fichier **header.twig** sera inclus dans le *layout* après la balise ouvrante de l'élément **body**. Celui du menu sera placé dans l'élément **nav**.

Le *template* comporte des blocs qui pourront être redéfinis dans les *templates* spécifiques à chaque page. Ils sont délimités par **{% block nom\_block %} ... {% endblock %}**. Les blocs présents sont **menu** et **body**.

```
{# layout.twig #}
<!doctype html>
<html>
  <head>
    <title>{{ titre_page }}
  </title>
    <meta charset='UTF-8'>
    <link rel="stylesheet"
href="style/style.css">
  </head>
  <body>
    {% include 'header.twig' %}
    <nav>
      {% block menu %}
        {% include 'menu.
twig' %}
      {% endblock %}
    </nav>
    {# afficher les titres et
noms des auteurs des livres #}
    <section>{% block body %}
    {% endblock %}</section>
  </body>
</html>
```

## 4. CRÉATION DU SCRIPT PHP

Pour simplifier l'exemple, nous allons créer un script **data.php** dont la fonction **getData()** retourne des données statiques plutôt que d'interroger une base de données. Ce fichier, placé dans le répertoire **modele**, est inclus dans **biblio.php**. Twig travaille avec des tableaux de données. Dans le *template* Twig, **{{ titre\_doc }}** fait référence à la valeur de la clé **titre\_doc** du tableau associatif des données.

```
<?php
// data.php
function getData() {
  // tableau associatif des
donnees
```

```

$data = array(
    'titre_doc' => 'Bibliotheque',
    'titre_page' => 'Liste des livres',
    'date' => date("d/m/Y"),
    // pour simplifier l'exemple, les
    données sont définies
    // statiquement (généralement elles
    sont extraites d'une BD)
    'biblio' => array(
array('titre'=>'1984', 'nom'=>'Orwell',
'prenom'=>'George'),
        array('titre'=>'Dune', 'nom'=>
'Herbert', 'prenom'=>'Frank'),
        array('titre'=>'N ou M', 'nom'=>
'Christie', 'prenom'=>'Agatha')
    )
);
return $data;
}

```

Une fois les données récupérées et stockées dans la variable `$donnees` dans `biblio.php`, le script inclut l'autoloader, puis définit le mode de chargement du *template* et son emplacement, en créant une instance de la classe `Twig_Loader_Filesystem`. Le *template* sera chargé depuis le chemin passé en paramètre, c'est-à-dire le répertoire `vue` placé au même niveau de l'arborescence que `biblio.php`. Nous avons défini deux tableaux d'options pour les phases de développement et de production. Pendant le développement, le système de cache est désactivé en attribuant la valeur `false` à l'option `cache`. Lorsque l'application sera en production, Twig stockera dans le répertoire `cache` les fichiers PHP générés. L'option `autoescape` permet de protéger automatiquement les sorties contre les attaques de type XSS. Le *template* qui est utilisé dépend du format reçu dans l'URL. Si l'URL contient la variable `format` avec une valeur `csv`, `html` ou `resume`, le nom du *template* prendra cette valeur. Par exemple, pour l'URL : `http://localhost/appli/biblio.php?format=html`, le *template* chargé sera `biblio.html.twig`. Par défaut, si le format n'est pas renseigné, c'est `html` qui sera utilisé.

```

<?php
// biblio.php
// récupérer le tableau des données
require 'modele/data.php';
$donnees = getData();

// inclure l'autoloader
require_once 'vendor/autoload.php';
// templates chargés à partir du système de
fichier (répertoire vue)
$loader = new Twig_Loader_Filesystem('vue');
// options : prod = cache dans le répertoire
cache, dev = pas de cache

```

```

$options_prod = array('cache' => 'cache',
'autoescape' => true);
$options_dev = array('cache' => false,
'autoescape' => true);
// stocker la configuration
$twig = new Twig_Environment($loader,
$options_dev);

$listeformat = ['csv', 'html', 'resume'];
$format = (isset($_GET['format']) && in_
array($_GET['format'], $listeformat)) ?
$_GET['format'] : 'html';

// charger+compiler le template, exécuter,
envoyer le résultat au navigateur
echo $twig->render("biblio.$format.twig",
$donnees);

```

## 5. HÉRITAGE DE TEMPLATE

Dans cette phase, nous allons créer deux vues qui héritent de `layout.twig`, elles auront donc la même mise en page générale. La première, `biblio.html.twig`, affichera toutes les informations de la bibliothèque dans un tableau HTML (figure 1). La seconde, `biblio.resume.twig`, affichera la liste des titres des livres (figure 4).

Le *template* fils `biblio.html.twig` débute par une instruction `extends` pour hériter du père (`layout.twig`) tout le code HTML, ainsi que le contenu de tout bloc qu'il n'a pas redéfini. Puis, il redéfinit le contenu du bloc `body` dans lequel il affiche les titres et les auteurs de chaque livre.

```

{# biblio.html.twig #}
{% extends "layout.twig" %}
{% block body %}
    {% if biblio is not empty %}
    <table>
        <thead>
            <tr><th>Titre</th><th>Nom auteur</
th><th>Prénom auteur</th></tr>
        </thead>
        <tbody>
            {# afficher les titres et noms des
auteurs des livres #}
            {% for item in biblio %}
                <tr>
                    <td>{{ item.titre }}</td>
                    <td>{{ item.nom }}</td>
                    <td>{{ item.prenom }}</td>
                </tr>
            </tr>
            {% endfor %}
        </tbody>
    </table>

```

# DISPONIBLE DÈS LE 10 MARS

## GNU/LINUX MAGAZINE HORS-SÉRIE N°89 !



# MAÎTRISEZ LA PROGRAMMATION DE SCRIPTS SHELL

# NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

# <http://www.ed-diamond.com>



```
{% else %}
    Aucun livre dans la bibliothèque.
{% endif %}
{% endblock %}
```

Pour tester l'exemple, il suffit d'utiliser l'URL suivante dans le navigateur : <http://localhost/appli/biblio.php>. Lorsque le navigateur demande le script `biblio.php`, les données et la vue `biblio.html.twig` sont compilées dans un fichier PHP, c'est ce fichier qui est exécuté et qui génère le code HTML présentant les données.

Le fils `biblio.html.twig` hérite du contenu du bloc `menu` défini dans la `layout`. Mais il est possible de changer le contenu de ce bloc pour ajouter des items au menu, par exemple un lien auteurs (figure 2). Pour cela, il faut redéfinir le bloc `menu`. Pour conserver le contenu du bloc dans la `layout`, il faut utiliser `{{ parent() }}` dans le bloc du fils comme illustré dans le listing suivant :

```
{% extends "layout.twig" %}
{% block menu %}
    {{ parent() }}
    | <a href="auteurs.php">Auteurs</a>
{% endblock %}
{% block body %}
    ...
{% endblock %}
```

## Bibliothèque - 27/10/2016

[Accueil](#) | [Liste](#) | [Ajouter](#) | [Modifier](#) | [Supprimer](#) | [Auteurs](#)

Titre	Nom auteur	Prénom auteur
1984	Orwell	George
Dune	Herbert	Frank
N ou M	Christie	Agatha

Fig. 2 : Menu redéfini avec héritage.

## Bibliothèque - 27/10/2016

[Auteurs](#)

Titre	Nom auteur	Prénom auteur
1984	Orwell	George
Dune	Herbert	Frank
N ou M	Christie	Agatha

Fig. 3 : Menu redéfini sans héritage.

Sans l'instruction `parent()`, le menu défini dans le père ne sera pas hérité, le menu ne contiendra que le lien auteurs (figure 3).

Dans le fils `biblio.resume.twig`, le bloc `body` est redéfini pour afficher la liste de tous les titres (figure 4). Pour tester l'exemple, il faut utiliser l'URL : <http://localhost/appli/biblio.php?format=resume>.

```
{# biblio.resume.twig #}
{% extends "layout.twig" %}
{% block body %}
    <h3>Titre</h3>
    {% for item in biblio %}
        <p>{{ item.titre }}</p>
    {% endfor %}
{% endblock %}
```

## Bibliothèque - 27/10/2016

[Accueil](#) | [Liste](#) | [Ajouter](#) | [Modifier](#) | [Supprimer](#)

Titre

1984

Dune

N ou M

Fig. 4 : Exécution de `biblio.resume.twig`.

## 6. AJOUT DE MACRO

Les macros évitent de répéter du code HTML. Elles sont similaires à des fonctions, elles acceptent des arguments optionnels. Par exemple, le fichier `macros.twig` crée une macro `lien` générant un lien hypertexte à partir des variables `src`, `text` et `class` qu'elle reçoit en paramètre.

```
{# macros.twig #}
{% macro lien( src, text, class='lien') %}
    <a href="{{ src }}" class="{{ class }}">{{ text }}</a>
{% endmacro %}
```

Pour utiliser la macro, il faut l'importer. Dans l'exemple, elle est importée dans le fichier `menu.twig` sous le nom `li`. Lors de l'exécution, la macro génère les liens hypertextes du menu. Ils ont tous la classe `lien` sauf le lien `supprimer` dont la classe est `inactif`.

```
{% from 'macros.twig' import lien as li %}

{{ li('accueil.php','Accueil') }} |
{{ li('liste.php ','Liste') }} |
{{ li('ajout.php ','Ajouter') }} |
{{ li('modif.php ','Modifier') }} |
{{ li('supprimer.php ','Supprimer', 'inactif') }}
```

Il est possible d'importer toutes les macros d'un fichier en une seule fois :

```
{% import 'macros.twig' as macro %}

{{ macro.lien('accueil.php','Accueil') }} |
...
```

## 7. PROTECTION CONTRE LE XSS

Twig permet de protéger les sorties vers le navigateur de manière automatique en utilisant l'option **autoescape** ou avec le filtre **escape (raccourci e)**. Le mode de protection automatique est activé par défaut avec la stratégie de protection HTML. C'est-à-dire que les données sont automatiquement protégées du XSS et de l'usurpation de contenu. La protection automatique est effectuée juste avant l'envoi au navigateur, une fois que la variable a été traitée par tous les filtres.

En l'absence de protection automatique, l'instruction **{{ titre|e }}** applique la protection HTML à la valeur de la variable **titre** avant son envoi au navigateur.

Lorsqu'un XSS est présent dans les données, la protection remplace les caractères **<** et **>** par des entités HTML. Le navigateur affiche le texte du XSS au lieu de l'exécuter (figure 5). Sans cette protection, le code **JavaScript** est exécuté (figure 6).

Bibliothèque - 27/10/2016		
<a href="#">Accueil</a>	<a href="#">Liste</a>	<a href="#">Ajouter</a>
<a href="#">Modifier</a>	<a href="#">Supprimer</a>	
Titre	Nom auteur	Prénom auteur
1984<script>alert(5)</script>	Orwell	George
Dune	Herbert	Frank
N ou M	Christie	Agatha

Fig. 5 : Protection XSS.

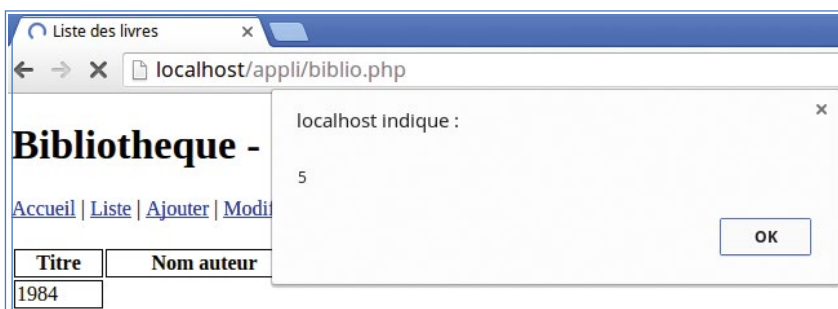


Fig. 6 : Sans protection XSS.

## 8. AUTRE FORMAT DE SORTIE

Twig peut être utilisé pour générer d'autres formats de sortie que le HTML. Par exemple, le fichier **biblio.csv.twig** retourne les données au format CSV. Pour tester l'exemple, il faut utiliser l'URL : <http://localhost/appli/biblio.php?format=csv>.

```
{# biblio.csv.twig #}
Titre, Auteur
{% for item in biblio%}
{{ item.titre }}, {{ item.nom }}
{% endfor %}
```

Afin que le navigateur ouvre le fichier CSV avec un tableur (figure 7), il faut envoyer un champ d'en-tête HTTP **Content-Disposition**. Ceci est réalisé dans le fichier **biblio.php** avec la fonction **header**.

```
// biblio.php
...
if ($format == 'csv') {
    header('Content-Disposition:attachment;filename=test.csv');
}
// charger+compiler le template,
// exécuter, envoyer le résultat au
// navigateur
echo $twig->render("biblio.$format.twig", $donnees);
```

	A	B
1	Titre	Auteur
2	1984	Orwell
3	Dune	Herbert
4	N ou M	Christie

Fig. 7 : Format CSV

## CONCLUSION

Une fois toutes les étapes réalisées, le script **biblio.php** produit trois résultats différents en fonction du paramètre **format** passé dans l'URL : la liste des livres au format HTML (figure 1) ou CSV (figure 7), ainsi qu'un résumé au format HTML (figure 4). ■

# CONFIGURATION TLS AVEC HAPROXY ET OPENSSL

ERWAN LOAËC

[Ingénieur systèmes et réseaux]

**MOTS-CLÉS : TLS, SSL, PERFECT FORWARD SECRECY, LOAD BALANCER, HAPROXY, OPENSSL**



**Nous utilisons tous les jours des connexions chiffrées, le plus souvent à travers le protocole TLS. Que vous soyez un(e) internaute soucieux(se) de la sécurité de vos échanges ou administrateur d'un serveur web, nous allons essayer de lever une partie des mystères qui se cachent derrière ce protocole...**

**L**e chiffrement des données devient de nos jours un standard. Cette sécurité, souvent basée sur TLS (ex « SSL »), est utilisée sur de nombreux protocoles. Le plus populaire est probablement le HTTP. Aujourd'hui, même les plus néophytes savent qu'il faut être vigilant à la présence du cadenas sur le navigateur. Pour autant, ce protocole qui paraît simple, cache en réalité beaucoup de subtilités qu'il est important de connaître.

## 1. LE PROTOCOLE TLS

Avant d'aborder un cas concret de mise en œuvre, il est bon d'aborder quelques principes généraux du chiffrement TLS.

### 1.1 La poignée de main (TLS Handshake)

La mise en œuvre d'une connexion TLS nécessite une poignée de main entre le client et le serveur. C'est lors de ces échanges que sont notamment définis les algorithmes qui seront utilisés pour le chiffrement.



No.	Time	Source	Destination	Protocol	Length	IRTT	Info
27	2017-01-06 21:39:45.163734	192.168...	89.38.1...	TLSv1.2	571	0.0530...	Client Hello
34	2017-01-06 21:39:45.220823	89.38.1...	192.168...	TLSv1.2	1514	0.0530...	Server Hello
39	2017-01-06 21:39:45.483836	89.38.1...	192.168...	TLSv1.2	719	0.0530...	CertificateServer Key Exchange, Server Hello Done
40	2017-01-06 21:39:45.488005	192.168...	89.38.1...	TLSv1.2	180	0.0530...	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
44	2017-01-06 21:39:45.540727	89.38.1...	192.168...	TLSv1.2	105	0.0530...	Change Cipher Spec, Encrypted Handshake Message

Fig. 1 : Exemple d'une capture wireshark d'une poignée de main TLS.

La première étape, comme visible sur la capture présentée en figure 1, s'appelle le « Client Hello ». Le client initie la connexion en spécifiant notamment la liste des suites de chiffrement (ciphers suite) qu'il supporte, dans un ordre de préférence. Nous détaillerons plus loin la signification d'une suite de chiffrement. C'est également dans cet échange que le client présente l'ensemble des extensions qu'il prend en charge, par exemple le SNI (*Server Name Indication*), qui permet au serveur de présenter un certificat différent en fonction du domaine appelé par le client.

Lors de l'étape suivante, le « Server Hello », le serveur indique la suite de chiffrement qu'il a choisi en s'appuyant sur les souhaits du client. Cependant, et c'est un point important, c'est le serveur qui décide des algorithmes qui seront utilisés, parmi la liste des possibilités annoncée par le client.

Dans l'échange qui suit, le serveur transmet un ou plusieurs certificats qui composent la chaîne de certification. Il s'agit d'un élément fondamental de la sécurité du TLS qui ne sera pas traité dans cet article.

Une fois que le client peut confirmer l'identité du serveur grâce à la validité de la chaîne de certification, il reste à s'accorder sur une clé symétrique, donc commune au client et au serveur. Cette clé ne doit pas transiter en clair. L'étape qui permet aux deux parties de s'accorder sur la clé symétrique s'appelle le « Key Exchange ». Nous reviendrons plus longuement sur cette étape dans la section 1.3.

Une fois toutes ces étapes accomplies, il est possible d'avoir des échanges de données sécurisés bidirectionnels entre le client et le serveur.

## 1.2 Suite de chiffrement

Nous avons abordé la notion de suite de chiffrement (ou « cipher suite » en anglais) dans la section précédente. Il s'agit d'un ensemble d'algorithmes utilisés conjointement pour sécuriser un échange TLS.

La suite cryptographique détaille notamment les algorithmes utilisés pour l'échange de clé, le chiffrement symétrique et l'authentification des messages. Il est facile de l'observer dans les navigateurs courants lorsque l'on navigue en utilisant le TLS. La figure 2 montre la suite de chiffrement utilisée avec Firefox pour le site <https://www.amazon.fr>, en se rendant dans les informations de la page (onglet **Sécurité**).

Examinons l'exemple suivant : **TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256**. Cette succession de termes barbares se découpe en plusieurs parties :

- **EC****DHE** **RS****A** : algorithmes utilisés pour l'échange de clé lors de la poignée de main initiale. L'**EC****DHE** correspond au mécanisme pour échanger la clé partagée, et **RS****A** pour le chiffrement asymétrique des paramètres transmis lors de l'utilisation de Diffie-Hellman.
- **AES\_128\_GCM** : algorithmes utilisés pour le chiffrement symétrique des données. Dans cet exemple, le chiffrement symétrique utilise l'**AES** avec une clé de **128** bits, avec un mode d'enchaînement des blocs de type **GCM**.
- **SHA256** : fonction de *hashage* utilisée pour authentifier les messages.

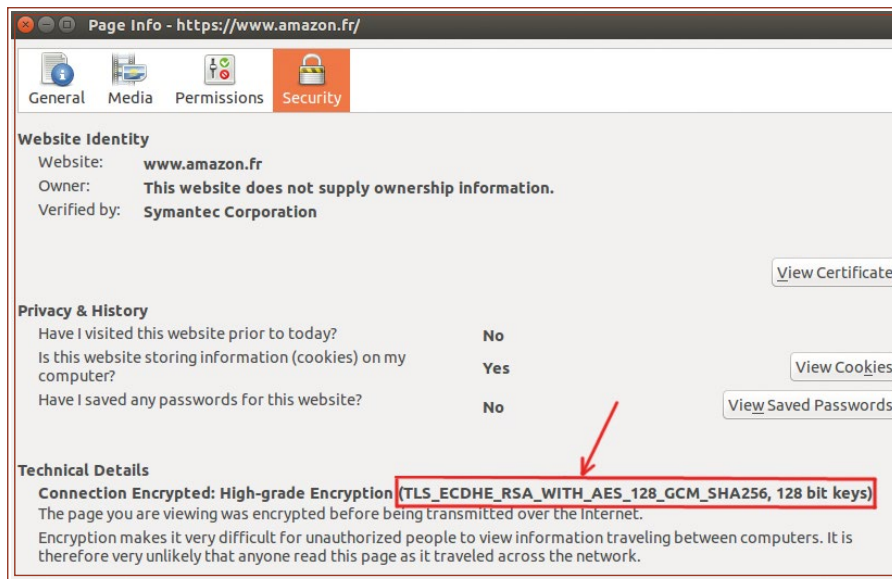


Fig. 2 : Suite de chiffrement utilisée avec Firefox pour le site <https://www.amazon.fr>.

Les noms de ces suites cryptographiques sont définis dans plusieurs RFC. Un identifiant unique, codé sur 2 octets, est associé à chaque suite cryptographique. Cet identifiant est inscrit auprès de l'IANA [1]. La suite que nous avons prise comme exemple a pour identifiant **0xc02f**. Cette valeur pourra par exemple être observée lors d'une capture réseau.

## 1.3 Échange de clé

Nous avons rapidement évoqué cette étape dans les paragraphes précédents. C'est l'une des étapes les plus importantes. Son but est de faire en sorte que le client et le serveur s'entendent sur une clé, qui ne sera connue que par les deux parties, et qui va servir à chiffrer les données transportées.

On retrouve actuellement deux grands principes pour le mécanisme d'échange de clé.

### 1.3.1 RSA « brut »

Pendant longtemps, les échanges de clé se sont basés sur l'algorithme de chiffrement asymétrique RSA. Le principe de fonctionnement est relativement simple, le certificat TLS transmis au client par le serveur contient une clé publique.

Grossièrement, le client génère une clé symétrique, puis la chiffre grâce à la clé publique présente dans le certificat en s'appuyant sur le RSA. Le serveur est le seul à pouvoir décoder la clé symétrique en utilisant la clé privée qu'il est le seul à connaître.

Ce mécanisme fonctionne bien, mais présente un problème de sécurité. En effet, imaginons que des échanges cryptés soient capturés et enregistrés, il serait possible de tout décoder dès lors que l'on réussirait à obtenir la clé privée, même longtemps après.

Cette contrainte amène le concept de « Perfect Forward Secrecy » (ou PFS).

### 1.3.2 Perfect Forward Secrecy

Cette propriété indique qu'il ne doit pas être possible de déchiffrer un message précédemment enregistré, même en cas de compromission de la clé privée dans le futur. On peut imaginer que celle-ci puisse être obtenue par compromission du serveur, récupération du disque dur, social engineering, etc.

Afin d'obtenir la propriété de « Perfect Forward Secrecy », l'échange de clé s'appuie généralement sur l'algorithme Diffie-Hellman. Nous n'allons pas rentrer dans les détails mathématiques. Pour les plus curieux, la page Wikipédia présente très bien les choses [2].

De cette manière, la clé partagée ne transite jamais à travers le réseau. La clé publique présente dans le certificat est également utilisée, mais cette fois-ci ce n'est plus pour protéger la clé partagée, mais pour authentifier le serveur. En effet, en chiffrant les paramètres Diffie-Hellman avec la clé privée, le client peut authentifier le serveur en déchiffrant le message avec la clé publique.

Pour garantir le *Perfect Forward Secrecy*, les paramètres Diffie-Hellman doivent être différents à chaque session, et doivent être générés à la volée (sans stockage persistant, pour éviter la compromission). Ces paramètres sont dits « éphémères ». Cette propriété correspond à la lettre « E » dans les acronymes **DHE**, et **ECDHE** (*Diffie-Hellman Ephemeral*, et *Elliptic Curve Diffie-Hellman Ephemeral*).

### 1.3.3 Optimisation de la poignée de main

L'échange de clé est une opération lourde en termes de calcul. Il existe des mécanismes qui permettent d'optimiser la poignée de main TLS afin d'éviter l'étape d'échange de clé à chaque connexion. En plus du gain en calcul, il est possible de gagner en bande passante. Les certificats, transmis par le serveur, peuvent représenter un certain volume de la bande passante, directement lié au nombre d'échanges de clé réalisés.

#### 1.3.3.1 Les tickets de session

Ce premier mécanisme est défini dans la RFC 5077 [3]. Les tickets de session permettent de conserver un état de la session TLS dans une structure qui n'est déchiffrable que par le serveur.

Cette extension TLS est « server stateless ». En effet, le serveur n'a pas besoin de conserver d'information relative à la session. La clé partagée, la suite de chiffrement, ainsi qu'une date maximale d'utilisation sont chiffrées et transmises au client.

Le client doit conserver le ticket de session ainsi que le secret partagé. Pour reprendre une session sans échange de clé, il transmet le ticket au serveur. Celui-ci le décode, et s'il confirme sa validité, reprend la session grâce à la clé partagée qu'il peut maintenant réutiliser.

L'utilisation des tickets de session est pratique lorsque le trafic est traité par une ferme de plusieurs serveurs. Il n'est pas nécessaire de maintenir des informations d'état des sessions communes aux serveurs. En revanche, il est nécessaire d'utiliser la même clé sur l'ensemble des serveurs afin de permettre une reprise des sessions, quel que soit le serveur qui prend en charge la requête.

Cette extension peut remettre en cause le mécanisme de « Perfect Forward Secrecy ». En effet, la capacité à déchiffrer la connexion tient au fait de posséder la clé qui a servi à chiffrer le ticket de session sur le serveur. Il faudrait donc que la clé soit changée fréquemment, et qu'elle ne soit jamais stockée sur disque. On distingue bien les contraintes si on doit gérer une ferme de serveurs.

### 1.3.3.2 Les ID de session

La seconde solution permettant de reprendre une session précédemment établie se base sur les ID de session. Il ne s'agit pas d'une extension du protocole comme les tickets.

Cette fois-ci, toutes les informations relatives à la session TLS sont stockées à la fois sur le client et sur le serveur. Lorsque le client établit une nouvelle connexion sur le serveur, il indique l'identifiant de la session précédente. Dans le cas où le serveur a encore le contexte dans son cache, la session peut être reprise. Dans le cas contraire, le serveur initie une poignée de main complète.

Lorsque le cache est volatile, c'est-à-dire uniquement en mémoire sur le serveur, cette solution s'avère plus sécurisée. Contrairement aux tickets de session, la prise en charge d'un cache des sessions du côté serveur demande des ressources en mémoire qui peuvent être significatives en fonction du trafic.

## 2. PROXY TLS

Dans une architecture web classique, on retrouve généralement en frontal des clients, un *load balancer* qui s'occupe de distribuer les requêtes vers un ensemble de serveurs web.

C'est cet équipement frontal qui prend généralement en charge la surcouche de chiffrement TLS. En effet, la communication entre le *load balancer* et les serveurs web se fait généralement à travers un réseau interne, qui ne nécessite pas forcément un chiffrement.

Une autre raison, qui est rarement évoquée, est que certains fournisseurs de certificat TLS fixent le prix en fonction du nombre d'équipements sur lesquels il est installé. Le prix peut vite grimper si le certificat est configuré sur toute une ferme de serveurs au lieu d'un seul équipement frontal.

Dès qu'on cherche des solutions hautes performances, beaucoup d'entreprises se tournent vers les leaders du marché (F5, Citrix...). Ces solutions intègrent des cartes de chiffrement dédiées. Pour la plupart il s'agit des cartes Cavium Nitrox. Le problème c'est que ces cartes ne sont performantes pour les calculs à base de courbes elliptiques que dans la

dernière génération Nitrox V, et les principaux constructeurs semblent toujours utiliser les processeurs Nitrox III, qui n'apportent pas de performance particulière sur les algorithmes à base de courbe elliptique, principalement utilisés dans la mise en œuvre du « Perfect Forward Secrecy ».

Il est donc intéressant de regarder les possibilités offertes dans le monde *open source*, pouvant être mises en œuvre dans un environnement serveur classique.

La solution la plus naturelle est probablement OpenSSL. La société Intel a publié une étude sur les performances des principaux algorithmes utilisés pour l'échange de clé TLS en utilisant les processeurs Intel Xeon E5 v3 [4], puis v4 (sorties en 2016) [5]. L'étude indique que la branche OpenSSL 1.0.2 peut profiter de certaines instructions disponibles sur ces processeurs et qui permettent d'obtenir d'excellentes performances, notamment dans les calculs basés sur les courbes elliptiques. Ces processeurs sont aujourd'hui massivement utilisés sur les serveurs.

Nous allons expliquer comment mettre en œuvre une solution d'*offloading* TLS pour un environnement web avec **OpenSSL** et **HAProxy**.

## 2.1 Installation de OpenSSL 1.0.2

En partant d'une installation **Debian Jessie** sur laquelle est installé le *package* **build-essential**, nous allons compiler la dernière version de la branche **OpenSSL** 1.0.2, c'est-à-dire la 1.0.2j au moment où ces lignes sont écrites :

```
$ cd /usr/src
$ wget https://www.openssl.org/source/openssl-1.0.2j.tar.gz
$ tar -xvzf openssl-1.0.2j.tar.gz
$ cd openssl-1.0.2j
$ ./config --prefix=/opt/openssl-1.0.2j no-shared
$ make depend
$ make
$ make install
```

À titre de comparaison, on peut faire les mêmes opérations avec la version 1.0.1u.

En utilisant la commande **openssl speed**, la comparaison des performances entre la version 1.0.1u et 1.0.2j est flagrante :

```
$ /opt/openssl-1.0.2j/bin/openssl speed ecdhp256
Doing 256 bit ecdh's for 10s: 95594 256-bit
ECDH ops in 10.00s

                op          op/s
256 bit ecdh (nistp256)  0.0001s  9559.4
```

```
$ /opt/openssl-1.0.1u/bin/openssl speed
ecdh256
Doing 256 bit ecdh's for 10s: 24862 256-
bit ECDH ops in 10.00s

      op      op/s
256 bit ecdh (nistp256) 0.0004s 2486.2
```

Sur l'environnement utilisé pour exécuter ces deux versions, les performances de la version 1.0.2j sont presque quatre fois supérieures à la version 1.0.1u pour l'algorithme ECDH.

## 2.2 Génération des clés et certificats

Pour notre environnement de test, nous allons créer notre propre autorité de certification en générant nous-mêmes un certificat autosigné :

```
$ openssl req -x509 -newkey rsa:2048 -keyout
CAkey.pem -out CA.pem -days 3650
```

Puis nous générerons deux certificats qui seront installés sur le serveur. Le premier utilisera l'algorithme de clé publique RSA, avec une clé privée de 2048 bits, puis le second utilisera ECDSA avec une clé de 256 bits.

Dans l'introduction de la RFC 4492 [6], il est indiqué que le niveau de sécurité d'un chiffrement avec une clé RSA 2048 bits est équivalent à un chiffrement basé sur les courbes elliptiques avec une taille de clé de 233 bits. Avec une taille de 256 bits, le certificat qui utilise ECDSA offre donc un niveau de sécurité plus élevé que le certificat basé sur RSA.

Pour un site web destiné au public, il sera nécessaire de faire l'acquisition des certificats auprès des autorités de certification de confiance installés dans les navigateurs.

### 2.2.1 Création du certificat RSA

Il faut tout d'abord générer une clé privée et un **CSR** (*Certificate Signing Request*). Afin de faciliter l'étude, on indique le type de certificat dans un des champs du **DN** (*Distinguished Name*), par exemple **O=Cert RSA Key**.

```
$ openssl req -newkey rsa:2048 -nodes -keyout
pkey.rsa.pem -out rsa.csr
```

Ensuite, depuis le certificat racine (autosigné), nous générerons un certificat à partir du CSR.

```
$ openssl x509 -req -days 365 -in rsa.csr
-CA CA.pem -CAkey CAkey.pem -out cert.
rsa.pem -CAcreateserial
```

Attention à spécifier un **CN** (*Common Name*) qui correspond au domaine qui sera utilisé. Dans notre exemple, il s'agira de **tls.loaec.fr**.

À l'issue de cette étape, nous aurons besoin des fichiers suivants :

- **pkey.rsa.pem** : la clé privée RSA ;
- **cert.rsa.pem** : le certificat RSA.

### 2.2.2 Création du certificat EC

Sur le même principe que l'étape précédente, nous créons la clé privée puis le CSR, en utilisant cette fois deux commandes distinctes, et enfin le certificat.

```
$ openssl ecpkeygen -name prime256v1 -genkey
-noout -out pkey.ecdsa.pem
$ openssl req -new -sha256 -key pkey.ecdsa.
pem -out ecdsa.csr
$ openssl x509 -req -days 365 -in ecdsa.
csr -CA CA.pem -CAkey CAkey.pem -out cert.
ecdsa.pem -CAcreateserial
```

Le choix de la courbe **prime256v1**, également connue sous le nom **NIST P-256**, est basé sur sa forte compatibilité avec les navigateurs standards.

Notons cette fois-ci la présence de ces deux fichiers :

- **pkey.ecdsa.pem** : la clé privée ECC ;
- **cert.ecdsa.pem** : le certificat ECDSA.

## 2.3 Installation de HAProxy

HAProxy est l'une des solutions *open source* de *loadbalancing* capables de prendre en charge le chiffrement TLS. La solution sur laquelle nous allons nous appuyer se base sur OpenSSL. Il y aurait beaucoup de similitudes avec la configuration TLS d'Apache ou de Nginx.

Nous allons profiter de la sortie de la version 1.7.1 de HAProxy pour monter notre environnement. Bien qu'elle soit annoncée en version stable, il est peut-être encore un peu tôt pour vraiment la déployer dans un environnement de production exigeant.

Voici les instructions pour compiler le serveur HAProxy avec OpenSSL :

```
$ cd /usr/src
$ wget http://www.haproxy.org/
download/1.7/src/haproxy-1.7.1.tar.gz
```

```
$ tar -xzf haproxy-1.7.1.tar.gz
$ cd haproxy-1.7.1
$ make TARGET=linux2628 CPU=native USE_
OPENSSL=1 SSL_INC=/opt/openssl-1.0.2j/
include SSL_LIB=/opt/openssl-1.0.2j/lib
$ make install PREFIX=/opt/haproxy-1.7.1-
openssl-1.0.2j
```

Commençons par créer le fichier de configuration `/etc/haproxy/haproxy.conf` :

```
global
    daemon

defaults
    timeout connect 2s
    timeout client 2s
    timeout server 2s

listen https
    mode http
    monitor-uri /
    bind :443 ssl crt /etc/haproxy/server.pem
    server srvweb1 127.0.0.1:80
```

Dans ce fichier de configuration très minimaliste, les *timeouts* sont purement indicatifs. Ils seront à ajuster en fonction de chaque cas. D'autre part, le paramètre **monitor-uri** permet de définir une URL spéciale, normalement utilisée par une solution de supervision pour vérifier l'état du service HAProxy. Nous allons détourner cette utilisation, en nous appuyant sur cette adresse pour valider la configuration de notre système, sans être dépendant du bon fonctionnement du service web final.

L'utilisation des certificats TLS avec HAProxy impose de créer un fichier regroupant à la fois la clé privée, le certificat final, ainsi que les certificats intermédiaires éventuels.

La branche 1.7 de HAProxy permet de gérer les deux certificats RSA et EC en fonction de leur prise en charge par le client. Pour cela, il est nécessaire de respecter des règles de nommage des fichiers. Chaque fichier doit avoir le même préfixe, par exemple **server.pem**, et avoir un suffixe en fonction de type de clé **rsa** ou **ecdsa**.

```
$ cat cert.rsa.pem pkey.rsa.pem > /etc/
haproxy/server.pem.rsa
$ cat cert.ecdsa.pem pkey.ecdsa.pem > /
etc/haproxy/server.pem.ecdsa
```

Il est maintenant temps de lancer HAProxy :

```
$ /opt/haproxy-1.7.1-openssl-1.0.2j/sbin/
haproxy -f /etc/haproxy/haproxy.conf
```

Pour tester notre configuration, essayons une requête depuis le serveur, en ignorant la validité du certificat (option **-k**) :

```
$ curl -k -v https://localhost/
[...]
* SSL connection using TLSv1.2 / ECDHE-RSA-
AES256-GCM-SHA384
* Server certificate:
* subject: C=FR; ST=TEST; O=Cert RSA Key;
CN=tls.loaec.fr
[...]
< HTTP/1.0 200 OK
< Cache-Control: no-cache
< Connection: close
< Content-Type: text/html
<
<html><body><h1>200 OK</h1>
Service ready.
</body></html>
[...]
```

Ce test montre le bon fonctionnement de la connexion TLS. La suite de chiffrement utilisée est **ECDHE-RSA-AES256-GCM-SHA384**, et le certificat transmis est celui qui contient la clé publique RSA.

## 2.4 Configuration du TLS

Le fichier de configuration HAProxy ne contient aucune configuration spécifique relative au TLS en dehors du certificat. Malgré tout, le service semble fonctionner si l'on s'en tient à notre test de connexion.

Pour valider une configuration TLS accessible sur Internet, une des références qui est souvent utilisée est l'outil **SSL Labs** de **Qualys** [7]. Si nous lançons un test sur notre configuration de base, comme visible en sur la figure 3, nous obtenons la note **C**, si on ne prend pas en compte le fait que le certificat n'est pas issu d'une autorité de certification de confiance.

L'audit montre qu'il est possible d'initier une connexion SSLv3, connue notamment pour la vulnérabilité **Poodle** [8]. D'autre part, l'utilisation de l'algorithme **RC4** est désormais déconseillée [9], or notre configuration permet l'utilisation de cette méthode de chiffrement.

Avant de vouloir à tout prix obtenir la note maximale **A+**, il faut d'abord se poser les questions suivantes :

- À qui s'adresse le site ?

Afin de permettre le chiffrement TLS, il faut d'abord évaluer quels seront les utilisateurs de ce service, et comment il sera utilisé. En effet, chaque navigateur a ses propres caractéristiques qui vont nous amener à privilégier l'utilisation de telle ou telle suite de chiffrement.

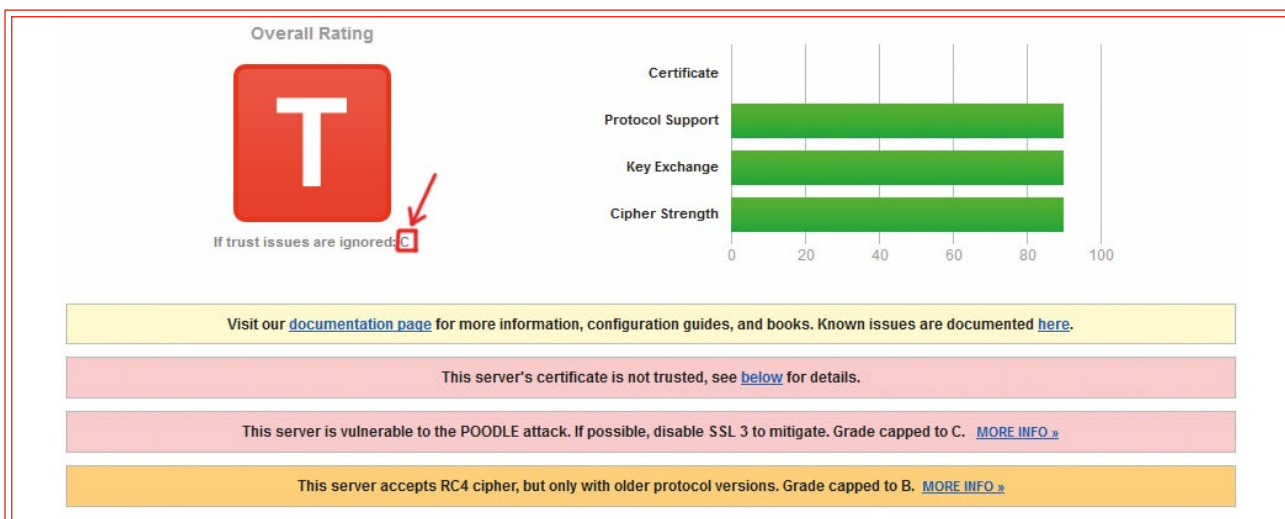


Fig. 3 : Résultat de l'audit SSL Labs réalisé sur notre configuration HAproxy initiale.

- Quelle est la nature des données qui vont transiter ?

Le choix d'utiliser le TLS peut être motivé par plusieurs raisons. Il peut s'agir de sécuriser les données échangées, de prouver l'identité d'un site, ou plus basiquement d'afficher le fameux cadenas dans le navigateur, sans autres préoccupations. En fonction de la sensibilité des données, le choix des algorithmes sera notamment orienté par le niveau de sécurisation qu'ils offrent.

- Quelle charge de trafic faut-il supporter ?

Comme évoqué plus tôt dans l'article, le chiffrement TLS demande des ressources de calcul. Selon les méthodes et les suites de chiffrement utilisées, les ressources nécessaires ne sont pas les mêmes, il est donc indispensable de connaître la fréquence des requêtes qu'il faudra prendre en charge, ainsi que le débit du trafic TLS.

Ces questions permettent de déterminer la configuration TLS à mettre en place. Il s'agit d'un compromis entre la compatibilité avec les navigateurs, le niveau de sécurité et la puissance de calcul à disposition.

### 2.4.1 Choix du protocole

Reprenons notre exemple, nous allons désactiver la prise en charge du SSLv3. Ce choix rendra le site inaccessible au navigateur **IE6** sous **Windows XP**.

Les versions 1.0, 1.1 et 1.2 du protocole TLS peuvent pour l'instant être déployées. La spécification de la prochaine version, TLS 1.3, n'est pour l'instant pas encore finalisée au sein de l'IETF [10].

Il reste encore beaucoup de navigateurs qui ne prennent en charge que la version 1.0. Il est donc important de permettre l'utilisation de cette version si l'on souhaite permettre l'accès au plus grand nombre.

### 2.4.2 Choix des suites de chiffrement

Bien configurer les suites de chiffrement est une étape primordiale dans la configuration d'un service TLS. En effet, même si votre seul but est de rassurer vos visiteurs en proposant un accès chiffré, certains gros acteurs comme Chrome, affichent un jugement sur le niveau des algorithmes utilisés, comme l'exemple de la figure 4.

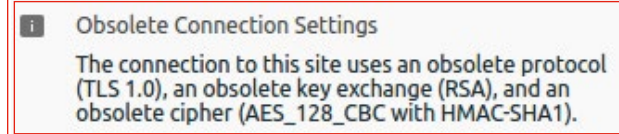


Fig. 4 : Exemple d'informations affichées dans Chrome pour une connexion TLS 1.0 avec la suite de chiffrement RSA-AES128-SHA.

Dans le cas d'**Apple**, il ne permet dans les dernières versions que des échanges compatibles avec le « Perfect Forward Secrecy ». Il est donc indispensable de permettre l'utilisation des suites de chiffrement spécifiques [11].

Il existe des bonnes pratiques diffusées par exemple par **Mozilla** [12], **SSL Labs** [13], etc. L'idée est de prioriser les algorithmes qui offrent le « Perfect Forward Secrecy », c'est-à-dire avec un échange de clé basé sur les courbes elliptiques (ECDHE) ainsi que Diffie-Hellman (DHE), pour terminer par les algorithmes qui utilisent uniquement le RSA, donc déchiffrables à partir de la clé privée du certificat.

Concernant le chiffrement symétrique, il faut privilégier le chiffrement AES en mode GCM, puis en mode CBC. L'utilisation du triple DES n'est pas recommandée. Cependant, afin de garder la compatibilité avec **IE8** sous Windows XP, il peut être nécessaire de continuer à le prendre en charge.

Il existe beaucoup de suites de chiffrement. Cependant, cette liste peut être fortement réduite, tout en gardant une compatibilité avec un grand nombre de clients. Il est important de garder la maîtrise des algorithmes, afin notamment de mieux appréhender les problématiques de performances.

Pour continuer avec notre exemple, voici une liste de suites de chiffrement qui assurent un bon niveau de compatibilité, tout en assurant un niveau de sécurité satisfaisant :

- `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` ;
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` ;
- `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384` ;
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384` ;
- `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` ;
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` ;
- `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA` ;
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA` ;
- `TLS_DHE_ECDSA_WITH_AES_128_GCM_SHA256` ;
- `TLS_DHE_RSA_WITH_AES_128_GCM_SHA256` ;
- `TLS_DHE_ECDSA_WITH_AES_128_CBC_SHA` ;
- `TLS_DHE_RSA_WITH_AES_128_CBC_SHA` ;
- `TLS_RSA_WITH_3DES_EDE_CBC_SHA`.

Dans le cas des algorithmes DHE, il est actuellement recommandé d'utiliser des paramètres Diffie-Hellman de 2K [14]. Voici la ligne de configuration à ajouter dans la section `global` :

```
tune.ssl.default-dh-param 2048
```

Afin d'appliquer cette configuration à notre service HAProxy, le `binding` doit être modifié de cette façon :

```
bind :443 ssl crt /etc/haproxy/cert/
cert.pem ciphers ECDHE-ECDSA-AES256-
GCM-SHA384:ECDSA-AES256-GCM-
SHA384:ECDSA-AES256-SHA384:ECDSA-
RSA-AES256-SHA384:ECDSA-AES128-GCM-
SHA256:ECDSA-RSA-AES128-GCM-SHA256:ECDSA-
ECDSA-AES128-SHA:ECDSA-RSA-AES128-
SHA:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES128-SHA:DES-CBC3-SHA no-sslv3
```

On notera la directive à la fin de la ligne qui désactive le SSLv3.

### 2.4.3 Gestion des reprises de session

Comme présenté en première partie, pour améliorer les performances, il peut être important de permettre la reprise de session SSL. Sur HAProxy, il est possible de définir une liste de clés utilisées pour le chiffrement des tickets de session. Ensuite cette liste peut être modifiée dynamiquement au moyen de la CLI à travers la commande `set ssl tls-keys`. Actuellement la documentation indique `tls-key` au singulier, or l'implémentation est faite avec `tls-keys`, il est probable que la commande soit corrigée dans les prochaines versions.

Afin de garantir le « Perfect Forward Secrecy », il est important d'assurer une rotation régulière des clés utilisées pour les tickets de session. Il faut d'autre part que ces clés ne soient pas sauvegardées sur le disque. La première difficulté est donc de mettre en place un fichier sur un système de fichier volatile.

L'autre point concerne la rotation. En effet, même si HAProxy permet de gérer la rotation des clés, ces modifications sont également volatiles. En cas de redémarrage du serveur, c'est de nouveau le fichier qui sera rechargé avec les valeurs qu'il contient, rendant ainsi tous les tickets obsolètes.

Dans le cas où le trafic est réparti sur plusieurs serveurs HAProxy, il est nécessaire que les clés soient les mêmes entre les serveurs. Une des solutions est de mettre en place un système de gestion centralisé de clés, qui s'occupe de configurer les clés au lancement du service, ainsi que de gérer la rotation régulière.

Afin de continuer sur notre exemple, compte tenu de la difficulté à obtenir une configuration fiable, performante et sécurisée, nous choisissons de ne pas prendre en charge les tickets de session. Il faut ajouter à la fin de la ligne de configuration du `binding` la directive `no-tls-tickets`.

En revanche, il est intéressant de gérer les reprises de session « statefull », ce qui est également le cas par défaut sur HAProxy. Il est possible d'agir sur la taille du cache, ainsi que sur la durée de rétention des sessions :

```
tune.ssl.cachesize 100000 # nombre de blocs
tune.ssl.lifetime 600 # durée en secondes
```

Actuellement, il n'est pas possible d'avoir un cache de session partagé entre plusieurs serveurs HAProxy. Pour gérer une répartition du trafic SSL sur plusieurs instances HAProxy, il faudra faire en sorte qu'il y ait une affinité sur le serveur qui initie la session TLS.

En fonction du type de service web (site, API, etc.), ou de la configuration du site (domaines multiples, `keepalive`, etc.), le taux de reprise de session peut varier.

## 2.5 Optimisation HAProxy

Par défaut HAProxy fonctionne avec un processus sur un seul *thread*. La première limitation arrive donc dès que la CPU qui traite le *thread* arrive à saturation. Afin d'exploiter au mieux les systèmes équipés de plusieurs processeurs, il est possible de spécifier dans la configuration de lancer HAProxy sur plusieurs processus. Il est même possible d'affecter à un processus une CPU en particulier.

```
tune.ssl.default-dh-param 2048
```

Vous noterez que les index des processus commencent à **1**, tandis que les index des CPU commencent à **0**.

En relançant le service, on peut constater qu'il y a quatre instances du processus HAProxy en cours d'exécution :

```
$ pidof haproxy
19244 19243 19242 19241
```

Si la majorité du temps processeur est utilisé pour le chiffrement TLS, il est probable que les performances soient améliorées en désactivant l'*hyperthreading*.

Le cache utilisé pour stocker les sessions SSL est partagé entre les différents processus, il est donc tout à fait possible d'avoir un même site distribué sur l'ensemble des CPU du serveur. En revanche, toutes les statistiques sont spécifiques à chaque processus. Il est nécessaire de configurer une *socket* d'administration sur chaque processus, et d'agréger les données obtenues :

```
stats socket *:3001 process 1
stats socket *:3002 process 2
stats socket *:3003 process 3
stats socket *:3004 process 4
```

## 3. VÉRIFICATIONS

Maintenant que nous avons une configuration fonctionnelle sécurisée et qui correspond parfaitement à nos besoins, nous allons faire quelques vérifications.

Tout d'abord, vérifions la liste des suites de chiffrement. Même si SSL Labs permet de le faire simplement, il peut arriver que le système ne soit pas accessible depuis Internet. L'outil **nmap** permet de faire quelques vérifications :

```
$ nmap --script ./ssl-enum-ciphers.nse -p 443 127.0.0.1
[...]
PORT      STATE SERVICE
443/tcp   open  https
| ssl-enum-ciphers:
|   SSLv3: No supported ciphers found
|   TLSv1.0:
|     ciphers:
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA - strong
|     compressors:
|       NULL
|   TLSv1.1:
|     ciphers:
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA - strong
|     compressors:
|       NULL
|   TLSv1.2:
|     ciphers:
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 - strong
|       TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 - strong
|       TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 - strong
|       TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 - strong
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA - strong
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 - strong
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 - strong
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 - strong
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA - strong
|     compressors:
|       NULL
|_ least strength: strong
```

Nous pouvons ainsi valider que notre proxy bloque l'utilisation du SSLv3, et que l'ensemble des suites de chiffrement est bien celui que nous avons défini, et offre une sécurité forte (*strong*). Attention cependant, la commande **nmap** ne retourne pas les suites de chiffrement dans l'ordre spécifiquement configuré sur le serveur.

La prise en charge des reprises de session peut se vérifier de plusieurs manières. Une des méthodes simples est d'utiliser le couteau suisse **openssl** :

```
$ openssl s_client -reconnect -connect 127.0.0.1:443
[...]
New, TLSv1/SSLv3, Cipher is ECDHE-ECDSA-AES256-GCM-SHA384
```



```
[...]
SSL-Session:
  Protocol   : TLSv1.2
  Cipher     : ECDHE-ECDSA-AES256-GCM-
SHA384
  Session-ID: FDF266D4E[...]1E34946D9D22
[...]
drop connection and then reconnect
[...]
Reused, TLSv1/SSLv3, Cipher is ECDHE-ECDSA-
AES256-GCM-SHA384
[...]
SSL-Session:
  Protocol   : TLSv1.2
  Cipher     : ECDHE-ECDSA-AES256-GCM-
SHA384
  Session-ID: FDF266D4E[...]1E34946D9D22
[...]
```

Nous pouvons constater que l'identifiant de session de la première connexion (**New**) est ensuite réutilisé pour la seconde connexion. La même vérification pourrait être réalisée directement avec un sniffer réseau comme Wireshark.

Quand cela est possible, un test de la configuration par SSL Labs permettra d'obtenir un rapport plus détaillé, ainsi qu'une note. Il arrive que des sociétés s'en servent comme base de référence quand il s'agit d'évaluer rapidement la sécurité de la configuration TLS d'un partenaire par exemple.

Il est également intéressant d'avoir des logs spécifiques au TLS. Voici un exemple de format de log qui peut être utilisé :

```
log-format "%ci:%cp [%tr] %ft %b/%s
%TR/%Tw/%Tc/%Tr/%Ta %ST %B %CC %CS %tsc
%ac/%fc/%bc/%sc/%rc %sq/%bq %hr %hs %{+Q}r %sslv
%sslc %[ssl_fc_session_id,base64]"
```

Cet exemple reprend le format par défaut des logs HTTP, en ajoutant 3 paramètres :

- **%sslv** : la version du protocole TLS (par exemple **TLS1.2**) ;
- **%sslc** : la suite de chiffrement utilisée (par exemple **ECDHE-RSA-AES128-GCM-SHA256**) ;
- **%[ssl\_fc\_session\_id,base64]** : l'identifiant de session codé en base 64. Cette information peut notamment servir à tracer toutes les connexions d'une même session, dans le cas où l'adresse IP de connexion n'est pas toujours la même.

Des indicateurs sont également disponibles à travers la *socket* d'administration, on peut notamment y trouver le nombre d'échanges de clés réalisés ainsi que l'utilisation du cache de session :

## Enseignants, Lycées, Écoles, Universités...

Besoin de  
ressources  
pédagogiques ?

...Permettre à mes élèves  
de consulter la base  
documentaire ?



C'est possible ! Rendez-vous sur :

<http://proboutique.ed-diamond.com>  
pour consulter les offres !

N'hésitez pas à nous contacter pour un devis personnalisé par e-mail :  
[abopro@ed-diamond.com](mailto:abopro@ed-diamond.com) ou par téléphone : +33 (0)3 67 10 00 20



```
$ nc 127.0.0.1 3001 <<< "show info" |
grep -i "ssl"
MaxSslConns: 0
CurrSslConns: 0
CumSslConns: 0
SslRate: 0
SslRateLimit: 0
MaxSslRate: 0
SslFrontendKeyRate: 0
SslFrontendMaxKeyRate: 0
SslFrontendSessionReuse_pct: 0
SslBackendKeyRate: 0
SslBackendMaxKeyRate: 0
SslCacheLookups: 0
SslCacheMisses: 0
```

Parmi ces indicateurs, on retrouve le taux de réutilisation des sessions : **SslFrontendSessionReuse\_pct**, qui permet d'évaluer le bon fonctionnement de la configuration ainsi que de comprendre le comportement des clients.

De même, la valeur **SslCacheMisses** permet d'observer le nombre de connexions pour lesquelles la session n'a pas été trouvée dans le cache. Une trop forte valeur permet d'indiquer que la durée ou la taille du cache des sessions est peut-être trop faible.

Comme indiqué précédemment, dans le cas où HAProxy est utilisé avec plusieurs processus, ces valeurs ne concernent que l'instance sur laquelle les statistiques sont associées. Il peut donc être nécessaire d'interroger séparément toutes les instances.

## CONCLUSION

La configuration du chiffrement TLS est désormais une étape presque incontournable dans la mise en place d'un site web. Il s'agit de quelque chose qui paraît simple au premier abord, mais comme nous l'avons évoqué, il existe un très grand nombre de paramètres qu'il convient de maîtriser afin d'adapter parfaitement la configuration à ses besoins.

Cette présentation permet d'éclaircir une partie de l'écosystème autour de la configuration TLS. Pour aller plus loin dans la mise en œuvre d'une architecture performante, il sera nécessaire de se pencher sur la configuration système (gestion des IRQs, configuration de la couche réseau du noyau, etc.).

Des solutions matérielles propriétaires sont aujourd'hui massivement déployées pour leur simplicité (relative), leur fiabilité ainsi que leur performance. Il est important de noter

qu'une solution *open source* telle que celle décrite dans l'article peut tout à fait trouver sa place, y compris dans les cas qui nécessitent la prise en charge de gros volume de requêtes. ■







## RÉFÉRENCES

- [1] TLS Cipher Suite Registry : <http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4>
- [2] [https://fr.wikipedia.org/wiki/%C3%89change\\_de\\_cl%C3%A9s\\_Diffie-Hellman](https://fr.wikipedia.org/wiki/%C3%89change_de_cl%C3%A9s_Diffie-Hellman)
- [3] TLS Session Resumption without Server-Side State : <https://www.ietf.org/rfc/rfc5077.txt>
- [4] <https://software.intel.com/en-us/articles/accelerating-ssl-load-balancers-with-intel-xeon-v3-processors>
- [5] <https://software.intel.com/en-us/articles/accelerating-ssl-load-balancers-with-intel-xeon-e5-v4-processors>
- [6] Elliptic Curve Cryptography Cipher Suites for TLS : <https://www.ietf.org/rfc/rfc4492.txt>
- [7] SSL Server Test : <https://www.ssllabs.com/sslltest/>
- [8] <https://fr.wikipedia.org/wiki/POODLE>
- [9] <https://threatpost.com/attack-exploits-weakness-rc4-cipher-decrypt-user-sessions-031413/77628/>
- [10] Draft TLS 1.3 : <https://tools.ietf.org/html/draft-ietf-tls-tls13-18>
- [11] <https://developer.apple.com/library/prerelease/ios/technotes/App-Transport-Security-Technote/>
- [12] [https://wiki.mozilla.org/Security/Server\\_Side\\_TLS](https://wiki.mozilla.org/Security/Server_Side_TLS)
- [13] <https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices>
- [14] Weak Diffie-Hellman and the Logjam Attack : <https://weakdh.org/>

# SERVEURS DÉDIÉS EN PROMOTION

QUANTITÉ LIMITÉE\* À PRIX EXCEPTIONNEL

## OFFRES SANS ENGAGEMENT DE DURÉE

SERVEURS	CPU	PROCESSEUR	RAM	DISQUE DUR	SETUP <sup>(3)</sup>	PRIX HT/MOIS
 <b>Green G460</b>	Intel® Celeron® G460 HT	1 CPU (1C/2T) @1,8 GHz	8 Go DDR3	2 To SATA <sup>(2)</sup>	<b>OFFERT</b>	39,99 € <b>12,99 €</b>
 <b>Crazy Fish</b>	Intel® Xeon® 3000	1 CPU (2C/2T) @1,86 GHz min.	4 Go DDR2	2 To SATA (5 400 tr/min)	<b>OFFERT</b>	29,99 € <b>14,99 €</b>
 <b>IKX-G620</b>	Intel® Pentium® G620	1 CPU (2C/2T) @2,6 GHz	8 Go DDR3	1 To SATA	19 €	39,99 € <b>9,99 €</b>
 <b>IKX-Core i3</b>	Intel® Core™ i3 2100T HT	1 CPU (2C/4T) @2,5 GHz	8 Go DDR3	1 To SATA <sup>(2)</sup>	19 €	69,99 € <b>12,99 €</b>
 <b>IKX-Core i5</b>	Intel® Core™ i5 2400S	1 CPU (4C/4T) @2,5 GHz	16 Go DDR3	1 To SATA <sup>(2)</sup>	19 €	69,99 € <b>17,99 €</b>
<b>IKX-Core i7</b>	Intel® Core™ i7 2600	1 CPU (4C/8T) @3,4 GHz	16 Go DDR3	1 To SATA <sup>(2)</sup>	19 €	<b>22,99 €</b>
<b>IKX-3430</b>	Intel® Xeon® Quad Core 3430	1 CPU (4C/4T) @2,4 GHz	8 Go DDR3 <sup>(1)</sup>	2 x 1 To SATA <sup>(2)</sup>	39 €	189,99 € <b>25,99 €</b>
<b>Green i5</b>	Intel® Core™ i5 Quad Core 3450	1 CPU (4C/4T) @3,1 GHz	32 Go DDR3	2 To SATA <sup>(2)</sup>	29 €	<b>25,99 €</b>
 <b>Green i7</b>	Intel® Core™ i7 Quad Core 3770 HT	1 CPU (4C/8T) @3,4 GHz	32 Go DDR3	2 To SATA <sup>(2)</sup>	29 €	<b>34,99 €</b>
<b>IKX-1220L</b>	Intel® Xeon® E3-1220L HT	1 CPU (2C/4T) @2,2 GHz	32 Go DDR3	2 x 1 To SATA <sup>(2)</sup>	39 €	129,99 € <b>34,99 €</b>
<b>IKX-R410</b>	Intel® Bi-Xeon® Quad Core 5000	2 CPU (4C/8T) @2 GHz	32 Go DDR3	4 x 1 To SATA Raid 1 Hard <sup>(2)</sup>	49 €	429,99 € <b>109,99 €</b>
<b>IKX-R710</b>	Intel® Bi-Xeon® Quad Core E5520 HT	2 CPU (4C/8T) @2,26 GHz	32 Go DDR3 <sup>(1)</sup>	6 x 1 To SATA Raid 1 Hard <sup>(2)</sup>	49 €	499,99 € <b>189,99 €</b>
<b>IKX-R910</b>	Intel® Quad-Xeon® Hexa Core E7540 HT	4 CPU (6C/12T) @4 GHz	64 Go DDR3 <sup>(1)</sup>	6 x 300 Go SAS Raid 1 Hard 2,5 <sup>(2)</sup>	49 €	569,99 € <b>249,99 €</b>



SYSTÈMES LINUX :



\*Serveurs dédiés disponibles en quantité limitée et sous réserve de disponibilité sur le site : [express.ikoula.com/serveur-dedie#promo](https://express.ikoula.com/serveur-dedie#promo)

<sup>(1)</sup> Possibilité d'augmenter le niveau de RAM.

<sup>(2)</sup> Possibilité d'augmenter la taille du / des disque(s) ou d'avoir une alternative SSD / SAS et RAID HARD.

<sup>(3)</sup> Frais de setup OFFERTS dans le cas d'un engagement annuel non cumulable avec les promotions en cours.

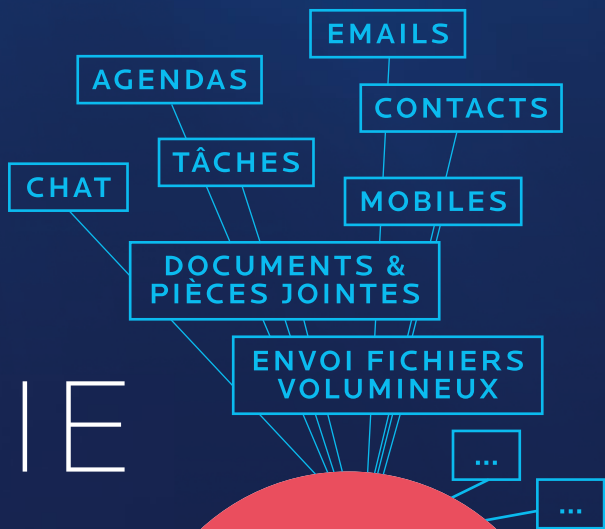
TOUTES LES PROMOTIONS SUR : [EXPRESS.IKOULA.COM](https://express.ikoula.com)



# BlueMind

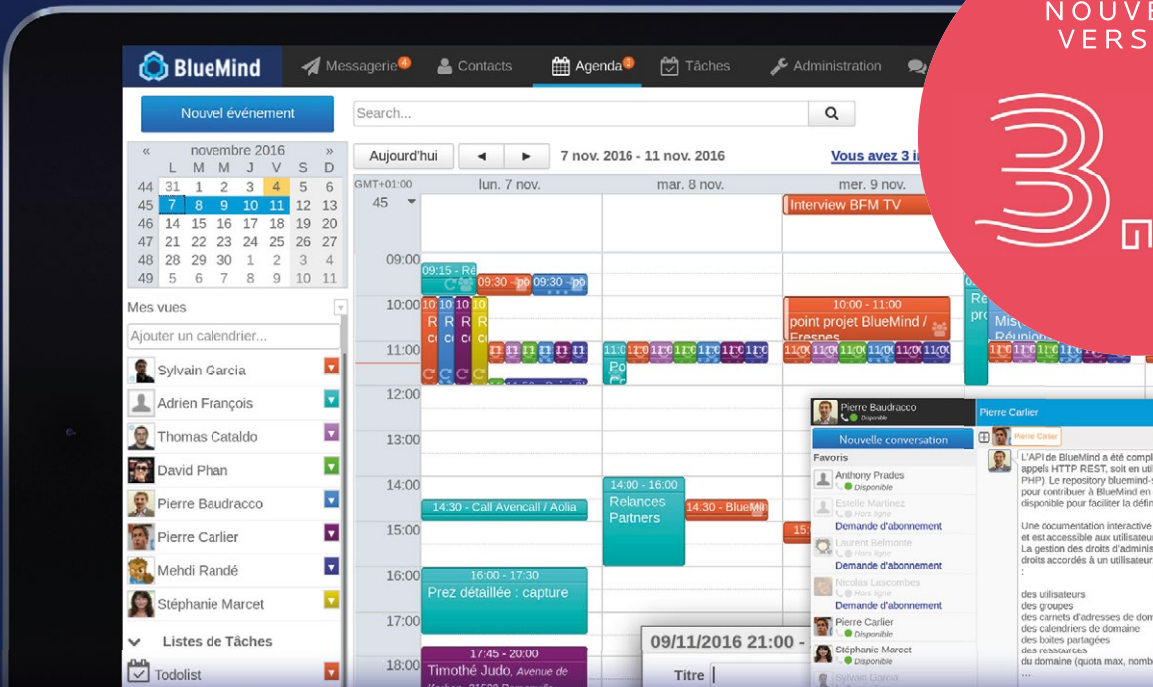
SOLUTION OPENSOURCE  
PROFESSIONNELLE DE MESSAGERIE  
COLLABORATIVE

# LIBÉREZ VOTRE MESSAGERIE



NOUVELLE  
VERSION

# 3.5



FRANCAIS / NOMBREUSES RÉFÉRENCES / ERGONOMIQUE / ÉVOLUTIF / ÉCONOMIQUE

Découvrez l'écosystème BlueMind et toutes les fonctionnalités sur

[www.bluemind.net](http://www.bluemind.net)

