Création d'un serveur de stockage de données utilisant le protocole SFTP et une plateforme Raspberry Pi

Réalisé par Guillaume EMERY – ING2 TD5
ECE Paris – École d'ingénieurs

ECE PARIS
ÉCOLE D'INGÉNIEURS

Table des matières

Introduction

La réalisation de ce projet est une idée qui me tient à cœur depuis un certain temps. En effet, utilisateur régulier de deux (voire trois) systèmes d'exploitation, je recherche un moyen de partager mes fichiers facilement entre les différents systèmes tout en m'affranchissant des systèmes propriétaires déjà existants (tels que Google Drive ou OneDrive). L'idée est donc de réaliser un système de partage privé reposant sur un serveur élaboré par mes soins. Ayant la chance d'avoir à mon domicile une connexion performante me permettant d'avoir accès à mon réseau rapidement (même en dehors de chez moi), il est alors envisageable pour moi de créer une solution privée et totalement sécurisée, tout en étant sûr que mes données ne transitent pas dans des endroits que je ne connaisse pas.

J'ai eu pour idée au début de ce projet de réaliser un système uniquement local mais j'ai fini par la suite par réaliser un système accessible de partout.

Les avantages de cette solution sont nombreux. Je ne comptais pas réaliser ce projet immédiatement mais plutôt pendant les vacances d'été. La réalisation de celui-ci dès aujourd'hui est pour moi une opportunité d'allier l'utile avec le pédagogique (la pédagogie est certes utile, mais mon projet aura pour finalité d'être utilisé tous les jours dès aujourd'hui).

Ayant effectué des recherches à ce sujet ces derniers mois, je me suis déjà arrêté sur un certain nombre d'aspects techniques :

- Utilisation d'une Raspberry Pi à la place d'un ordinateur (moins cher, système totalement open source et paramétrable selon mes envies).
- Utilisation de deux disques dur en RAID afin d'assurer la copie en permanence de mes données en cas de problème.
- Utilisation du protocole SFTP, alliant la connexion SSH (permettant de sécuriser les transferts) et le système de transfert FTP (permettant l'accès à mes données même en n'étant pas connecté sur mon réseau local). Cette technologie sera décrite dans la suite de rapport.
- Et, évidemment : utilisation d'une distribution Linux, dérivée de Debian et adaptée à l'architecture ARM de la Raspberry : Raspbian.

N'ayant pas à l'heure actuelle le temps (ni les moyens financiers) d'acheter deux disques durs et une interface permettant de les connecter à la Raspberry, je me contenterai, pour l'instant, d'utiliser un seul système de stockage. Mais le fonctionnement restera le même et me permettra pendant les vacances de faire évoluer mon système. J'ai également dans l'idée de créer une application (sur OS X, sur Windows et sur Linux), permettant de faire une sauvegarde et une synchronisation de mes données.

Je vais dans un premier temps présenter brièvement les domaines d'application de ce projet (bien qu'ayant déjà été plus ou moins abordés dans cette introduction). Puis, j'expliquerai comment j'ai procédé pour installer une distribution Linux sur la Raspberry. Ensuite, je montrerai comment j'ai fait pour utiliser la Raspberry chez moi (ne disposant ni d'un écran, ni d'un clavier, ni d'une souris mais uniquement d'un ordinateur portable). Enfin, je présenterai le système en lui-même (installation et configuration du système SFTP, étapes pour permettre l'accès hors de mon réseau local et montage du lecteur réseau sur les différents systèmes d'exploitation).

I. Domaines d'application

La réalisation de ce projet m'a semblée intéressante dans le sens où elle couvre un grand nombre des connaissances acquises durant l'initiation à Linux en cours. En outre, ayant été un utilisateur de Linux avant de passer sur un système Apple (qui, bien que de philosophies totalement différentes, se ressemblent en un certain nombre de points), j'ai eu grand plaisir à réutiliser toutes mes connaissances acquises précédemment. Ainsi, j'ai eu à utiliser, entre autres, les notions suivantes :

- Installation d'une distribution Linux sur une plateforme ARM (une première pour moi).
- Configuration d'un système de communication via SSH avec mise en place d'un accès permanent depuis l'extérieur.
- Installations et configurations de nombreux paquets nécessaires à la mise en place de ce projet.
- Optimisation d'une distribution Linux pour un système aux ressources matérielles faibles.
- Démonstration de l'ouverture des systèmes Linux en proposant un système multiplateforme (l'idée étant que je puisse accéder à ce système depuis OS X, Windows, Linux et – éventuellement – un Smartphone).

II. Installation d'une distribution Linux

La réalisation de ce projet m'a conduit à réaliser deux installations :

- Installation d'une distribution Ubuntu sur mon MacBook.
- Installation d'une distribution ARM sur la Raspberry.

A. Sur mon ordinateur

Je ne m'étendrai pas sur l'installation d'Ubuntu sur un Mac : ce n'est pas vraiment l'objet de ce rapport. J'ai utilisé jusqu'à ce jour une machine virtuelle mais celle-ci m'a posé problème pour accéder en SSH au Raspberry. Je ne sais pas exactement pourquoi mais j'imagine que, étant donné que VirtualBox simule un réseau pour la machine virtuelle (indépendant de la connexion réseau du système hôte), ma machine virtuelle n'est pas capable de trouver un périphérique de mon réseau

local « réel » (celle-ci ne se trouve pas sur celui-ci). Ce problème n'aurait été que provisoire car j'ai par la suite amélioré ma solution pour y accéder de l'extérieur.

Pour faire un bilan rapide, les différentes étapes ont été les suivantes :

- Installation d'un utilitaire, rEFIt, permettant à la fois le chargement des différents systèmes d'exploitation (l'EFI du Mac ne le permettant pas par défaut) et le démarrage depuis une *live USB*.
- Création d'une clé USB d'installation d'Ubuntu à l'aide d'une commande sur le terminal d'OS X (également présente sur les systèmes GNU/Linux) : **dd**. Je décrirai le fonctionnement de celle-ci dans la suite de ce rapport.
- Suppression de la partition de restauration créée par Apple sur tous les Macs (le nombre de partitions primaires étant trop important pour réaliser un *triple boot* avec la création d'une partition de *swap* pour linux) : j'ai pour cela utilisé **gparted** sur une live USB.
- Suppression de mon installation de Windows existante.
- Installation d'Ubuntu.
- Réinstallation de Windows.

B. Sur la Raspberry

1. Présentation

L'installation sur la Raspberry me semble plus intéressante, car elle est moins « courante » : j'ai déjà effectué de nombreuses installations par le passé et ce, sur différents types d'ordinateurs. Mais cela a été une première pour moi que de l'installer sur une plateforme ARM.

J'ai dû pour cela trouver le système adéquat. Les documentations fournies sur le site de Raspberry m'ont pour cela bien aidé et m'ont orienté dans mes choix. J'en ai retenu deux dans un premier temps, pour finalement m'arrêter sur l'option Raspbian :

- NOOBS : (pour *New Out Of the Box Software*) est un package qui, comme son nom l'indique, est prêt à être utilisé directement après avoir créé la carte SD. Cette option m'a semblé adaptée lorsque j'ai commencé à m'intéresser à ce sujet, mais elle a finalement posé un certain nombre de problèmes, notamment :
    - Impossibilité (à ma connaissance) d'utiliser ce package sans l'utilisation d'un écran et d'un clavier (j'aurais toujours eu la possibilité d'aller à l'école pour brancher la Raspberry sur un vidéoprojecteur).
    - Prise de place plus importante sur la carte SD : ne possédant qu'une carte de 8Go, je veux quand même garder de la place sur celle-ci pour pouvoir stocker mes fichiers.

- Raspbian : mot valise composé de « Raspberry » et « Debian ». J'évolue donc ici en terrain partiellement connu, ayant déjà utilisé Debian sur un vieux portable ayant quelques difficultés à faire tourner correctement Ubuntu. De plus, cette solution présente pour moi toutes les caractéristiques dont j'ai besoin :
    - Support du SSH dès le premier allumage : je peux donc m'affranchir totalement d'un écran et d'un clavier.
    - Très légère : contrairement à NOOBS qui livre sur la carte SD plusieurs distributions, celle-ci est « seule ». Une fois le projet fini, la distribution occupe 2,1Go sur les 7,1Go disponibles et il est encore possible de retirer des paquets inutiles, dont l'interface graphique et les différents logiciels qui y sont associés et qui occupent beaucoup d'espace.
    - Configuration relativement facile : toutes les tâches à effectuer pour mener à bien ce projet sont faisables en mode texte.

2. Déroulement de l'installation

L'installation de Raspbian est finalement beaucoup plus simple que sur un ordinateur standard : il y a seulement à copier l'image fournie par le site officiel de la Raspberry sur une carte SD, mettre celle-ci dans la carte et la brancher sur le secteur.

Pour créer la carte SD, il a fallu tout d'abord récupérer l'image (sur http://www.raspberrypi.org). Une fois ceci fait, j'ai copié le contenu de l'image sur la carte SD à l'aide de la commande **dd**, comme je l'ai fait précédemment lorsque j'ai créé ma clé USB d'installation d'Ubuntu pour mon PC.

Remarque : cette commande est décrite ici comme étant utilisée sur Ubuntu, mais lors de la création de ma clé USB d'installation, j'ai effectué la procédure sur OS X. La démarche est identique hormis un utilitaire qui n'a pas le même comportement sur les deux systèmes d'exploitation.

Le fonctionnement de dd est assez simple. Après avoir formaté une carte SD (j'ai découvert plus tard que cette étape était inutile), j'ai ouvert un terminal et rentré la commande suivante :

```
sudo dd bs=1m if=/home/guillaume/Bureau/2014-01-07-wheezy-raspbian.img
of=/dev/sdb
```

`dd` est un utilitaire permettant de copier un élément source vers un autre élément cible. Cet utilitaire fonctionnant par blocs d'octets, il est nécessaire de lui fournir la taille des blocs de données à copier. C'est à cela que sert le paramètre `bs` (pour block size). Ici, j'ai rentré le paramètre `1m`, signifiant que la commande copiera des blocs d'un méga-octet.

Les deux paramètres suivants spécifient la source et la cible. On rentrera la source après le `if` et la cible après le `of`.
Ici, j'ai simplement rentré l'emplacement du fichier image de Raspbian et le numéro de la carte SD affecté par le système d'exploitation.

Pour trouver ce numéro, une commande UNIX existe :

```
sudo fdisk -l
```

Sur OS X, `fdisk` a un comportement différent. En effet, l'option `-l` n'existe pas sur l'OS d'Apple (« `illegal option` »). J'ai donc utilisé une commande différente : `diskutil list`. Celle-ci donne un résultat identique.

Une fois la procédure achevée (environ une vingtaine de minutes), mon installation de Raspbian était prête à être utilisée. Il me fallait alors accéder à la Raspberry en SSH afin de la paramétrer.

III. Mise en place d'une communication SSH entre le Raspberry et mon PC personnel

Comme je l'ai dit précédemment, je ne dispose chez moi ni d'un écran, ni d'un clavier, ni d'une souris. Il m'a semblé alors indispensable d'utiliser le protocole SSH afin de pouvoir effectuer les différentes manipulations pour paramétrer mon projet.

Le SSH est un protocole de communication sécurisé qui permet, outre le transfert de fichiers, le contrôle à distance d'un système. Il m'a donc été nécessaire dans un premier temps de récupérer l'adresse IP du Raspberry présent sur mon réseau local. J'ai utilisé pour cela l'utilitaire `nmap`.

`nmap` est un scanner de réseaux, disponible sur les dépôts d'Ubuntu. Après installation de celui-ci, j'ai rentré la commande suivante :

```
nmap -sP 192.168.1.1-255
```

L'option `-sP` effectue un « ping scan ». Cela détermine si les hôtes d'un réseau (en l'occurrence, le mien) sont en ligne et opérationnels. Cela affiche alors le temps de réponse de chaque hôte ainsi que son adresse MAC (et le fournisseur du matériel) et son IP.

Cette commande m'a permis, grâce aux indications sur les adresses MAC, de récupérer l'IP de ma Raspberry. En effet, après l'adresse, se trouve la mention « Raspberry Pi Fondation » : suffisamment explicite pour être sûr que l'IP soit la bonne.

Une fois l'IP récupérée, il m'a fallu installer les différents paquets nécessaires à l'utilisation du SSH sur Ubuntu. J'ai pour cela choisi d'installer le paquet `ssh`, qui regroupe les paquets nécessaires pour utiliser la machine en tant que client et en tant que serveur (même si dans mon cas seule la partie client est nécessaire).

Grâce aux documentations fournies par la fondation Raspberry, j'ai trouvé facilement le nom d'utilisateur et le mot de passe par défaut de ma Raspberry. J'ai ainsi simplement rentré la commande suivante pour accéder à ma carte :

```
ssh pi@192.168.1.86
```

Le fonctionnement est simple :

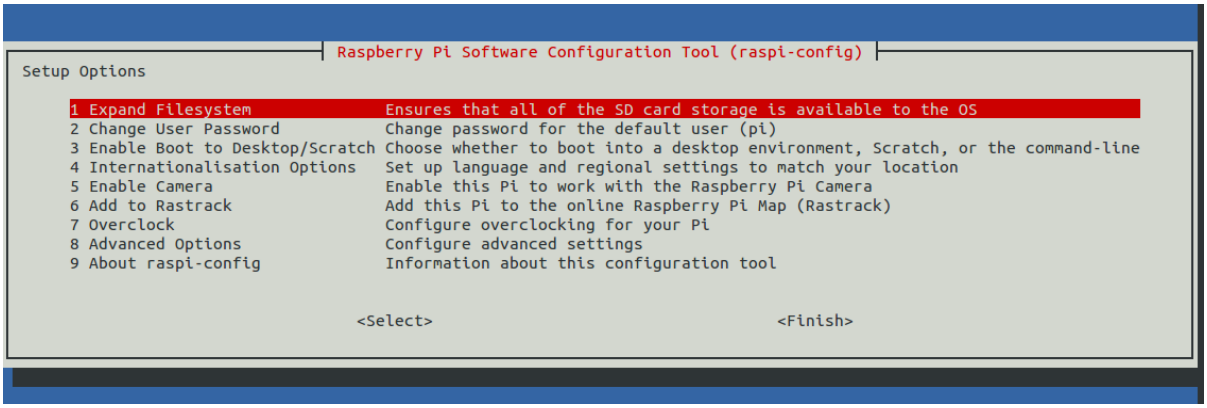
- `pi` désigne le nom d'utilisateur de la Raspberry (je l'ai laissé par défaut).
- `@192.168.1.86` désigne l'adresse IP de la carte.

Une fois cette commande rentrée, le mot de passe du compte `pi` sur la Raspberry est demandé. Par défaut, celui-ci est `raspberry`. Pour plus de sécurité, je l'ai immédiatement changé une fois la connexion établie.

La connexion est ensuite établie : le terminal de mon ordinateur correspond alors exactement à celui de la Raspberry. J'ai immédiatement accédé à l'interface de gestion de la Raspberry en entrant simplement :

```
sudo raspi-config
```

Cette commande m'a donné l'interface suivante :



La première étape de configuration de la Raspberry a été de changer le mot de passe. Ensuite, pour plus de confort, j'ai choisi d'étendre la taille de la partition système au maximum de la capacité de la carte mémoire. Une fois cette opération faite, un redémarrage de la carte a été nécessaire. Puis, après reconnexion, j'ai utilisé la commande suivante pour vérifier que l'opération a été effectuée correctement :

```
df -h
```

`df` (abréviation de *disk free*) affiche l'espace disponible sur les différentes partitions du disque. L'option `-h` spécifie juste au système d'afficher les résultats de façon à ce qu'ils soient lisibles (le manuel de cette commande m'a appris que j'aurais pu écrire `--human-readable`, cela aurait eu le même effet).

Après analyse des résultats donnés par cette commande, j'ai pu constater que l'opération a bien été faite. Arrivé à ce stade de la configuration de la Raspberry, j'ai donc une carte pleinement

fonctionnelle. Il ne reste donc plus qu'à installer le nécessaire pour utiliser `sshfs`. Avant toute chose, afin d'être sûr d'avoir tous les paquets nécessaires pour la suite et que mon système soit à jour, j'ai entré deux commandes :

```
sudo apt-get update && sudo apt-get upgrade
```

`update` va mettre à jour la liste des paquets depuis les dépôts de Raspbian. `upgrade` va quant à lui mettre à jour les paquets obsolètes.

IV. Installations et configurations des paquets nécessaires à l'ensemble du projet

L'utilisation du protocole SSHFS nécessite l'installation et la configuration de FUSE qui lui-même nécessite l'installation du protocole SAMBA.

Pour résumer, SAMBA est un logiciel d'interopérabilité qui permet à un ordinateur serveur de mettre à disposition des fichiers et d'autres ressources sur un réseau local. Après installation de celui-ci (le paquet est disponible sur les dépôts de Raspbian), j'ai pu commencer la configuration de FUSE.

FUSE (*Filesystem in USErspace*), quant à lui, est un système permettant à un utilisateur de monter un système de fichiers. Pour monter un système de fichiers, dans les cas « normaux », il faut être administrateur et il faut que le montage soit autorisé dans le fichier *fstab*. Ce fichier (pour *filesystem table*) liste les différentes partitions des différents disques utilisés au démarrage. Or, dans notre cas, le système de fichiers à monter n'est pas présent au démarrage et n'est donc de ce fait pas inscrit dans le fichier *fstab*. FUSE va donc permettre à l'utilisateur de monter directement un espace de stockage.

L'installation de FUSE nécessite deux paquets, également présents dans les dépôts de Raspbian : `fuse-utils` et `libfuse2`.

Pour pouvoir utiliser FUSE, plusieurs conditions doivent être vérifiées :

- il faut que le module soit chargé au démarrage de la machine.
- il faut que l'utilisateur soit autorisé à utiliser FUSE.

Pour charger le module au démarrage, il faut entrer la commande suivante dans le shell SSH du Raspberry :

```
sudo sh -c "echo fuse >> /etc/modules"
```

`sh` indique simplement que du texte va être écrit dans un interpréteur shell. L'option `-c` va indiquer que la commande sera lue depuis la chaîne écrite à la saisie. On utilise ensuite la commande `echo` pour simplement écrire dans le fichier `modules` que FUSE sera chargé.
Ensuite, pour charger le module (sans avoir à redémarrer le système), la commande `modprobe` peut être utilisée. Celle-ci permet de charger un module au noyau Linux « à chaud ». J'ai alors rentré simplement :

```
sudo modprobe fuse
```

Une fois ceci fait, il faut ajouter l'utilisateur du Raspberry au groupe fuse afin de l'autoriser à l'utiliser. Pour cela, il faut entrer une nouvelle commande :

```
sudo adduser pi fuse
```

Enfin, il faut modifier les autorisations du module de façon à ce que le groupe fuse puisse l'utiliser. Il faut pour cela utiliser la commande suivante :

```
sudo chgrp fuse /dev/fuse
```

`chgrp` permet de dire quel groupe d'utilisateurs possède un dossier. Dans notre cas, nous souhaitons que le groupe fuse possède le module fuse afin que l'utilisateur (ajouté au groupe fuse précédemment) puisse l'utiliser.

La configuration de FUSE est terminée, je peux maintenant rentrer « dans le vif du sujet » : SSHFS.

L'installation de SSHFS est simple : il suffit juste d'installer le paquet. La configuration de FUSE ayant été, a priori, effectuée correctement, il n'y a rien d'autre à faire. Le logiciel étant présent dans les dépôts de Raspbian, un simple `sudo apt-get install sshfs` suffit.

V. Configuration

Lors de l'élaboration de mon projet, la configuration s'est faite en deux temps. Dans un premier temps, j'ai voulu vérifier le fonctionnement de tous mes paramètres : j'ai donc choisi de passer uniquement par mon réseau local. Dans un second temps, j'ai voulu rendre accessible mon dossier depuis l'extérieur. Je décrirai donc les différentes configurations à effectuer afin de rendre ceci possible.

A. Configuration sur mon réseau local

Pour utiliser SSHFS, il est nécessaire de créer plusieurs dossiers :

- Un dossier sur le serveur (la Raspberry)
- Un dossier sur chaque machine voulant accéder au système de fichiers.

Pour créer un dossier sur la Raspberry, il faut entrer :

```
mkdir mydrive
```

`mydrive` est le nom choisi pour mon projet, j'appellerai donc ce dossier par ce nom. Ensuite, un simple `ls` permet de vérifier que le dossier s'est bien créé.

Ensuite, il faut créer un dossier sur les machines clientes. Ce dossier peut être n'importe où et s'appeler n'importe comment, cela ne change pas grand chose quand le montage se fait.

Une fois les deux dossiers créés, il faut ouvrir un terminal local (une deuxième fenêtre qui ne contrôle pas le Raspberry mais bien le PC) et entrer la commande suivante :

```
sshfs pi@192.168.1.86:/home/pi/mydrive /home/guillaume/Bureau/mydrive
```

`sshfs` va recevoir plusieurs paramètres :

- le nom d'utilisateur (`pi`) et la machine sur laquelle se connecter (désignée par son adresse IP)
- le dossier distant sur la Raspberry (`/home/pi/mydrive`)
- le dossier local sur le PC (`/home/guillaume/Bureau/mydrive`)

Le bash va ensuite demander le mot de passe utilisateur de la Raspberry : une fois celui-ci rentré, le dossier distant sera monté localement et tout son contenu sera accessible en lecture/écriture.

Arrivé à ce stade du projet, j'ai donc une solution pleinement fonctionnelle sur mon réseau local. Il s'agit maintenant de rendre cette solution utilisable à distance.

B. Configuration pour une utilisation à distance

Remarque : cette partie n'a pas pour vocation de parler des notions apprises en cours sur Linux, mais, étant une étape capitale à la réalisation de mon projet, et ayant été l'étape m'ayant pris le plus de temps car ne trouvant pas de documentation ou tutoriel m'indiquant exactement quoi faire, il me semble opportun de résumer les différents obstacles qui me sont apparus.

L'utilisation à distance est la vocation finale de ce projet : avoir un système de stockage privé accessible de partout. Cependant, c'est cette étape qui a été la plus difficile pour moi, n'ayant quasiment aucune connaissance en architecture réseau et ne sachant pas quoi modifier exactement dans les paramétrages de mon routeur.

Il m'a donc fallu réaliser plusieurs opérations :

- Utiliser un système permettant de rediriger un nom d'hôte vers une adresse IP dynamique : **no-ip**.
- Paramétrer correctement no-ip dans l'interface de gestion de mon routeur
- Effectuer une redirection de connexions pour le port utilisé par le protocole SSH de façon à ce que les connexions entrantes sur mon réseau arrivent sur la Raspberry.

Dans un premier temps, il a fallu que je crée un compte no-ip et que je me créé un nom d'hôte. Je passerai ici les étapes nécessaires à la création du compte : je risquerais de m'éloigner un peu trop de la thématique de ce projet.

Une fois le nom d'hôte créé, il m'a fallu rediriger les connexions entrantes à cet hôte vers l'adresse IP de ma box. Pour cela, j'ai dû effectuer différents paramétrages sur mon routeur :

- Attribuer une IP statique à ma Raspberry sur mon réseau local (à l'aide de son adresse MAC). J'ai pensé lors de l'élaboration de ce projet qu'attribuer une IP fixe est suffisant pour se connecter depuis une connexion extérieure à mon réseau local. J'ai ensuite compris que cette IP est fixe seulement sur mon propre réseau et qu'elle n'existe pas réellement depuis l'extérieur.
- Utiliser un service DynDNS permettant d'attribuer un nom d'hôte à l'adresse IP dynamique de mon routeur (seule adresse visible de l'extérieur depuis Internet). Ce service, par chance, est fourni par SFR directement dans leur interface de gestion du routeur et est compatible avec no-ip. Ainsi, à chaque fois que je me connecterai sur le nom d'hôte choisi sur le service no-ip, la redirection se fera sur l'adresse IP dynamique de mon routeur. Le système proposé par SFR se charge quant à lui de mettre directement à jour l'adresse IP de la box chez no-ip.
- Effectuer une translation de ports (NAT) de façon à ce que chaque connexion entrante se faisant vers mon routeur via le port 22 (utilisé pour les connexions SSH) aille directement sur la Raspberry.

Après ces différents paramétrages (et après un certain nombre d'essais infructueux), la redirection se fait maintenant sans problème en me connectant depuis l'extérieur (il semblerait par contre que l'école n'accepte pas les connexions sortantes SSH étant donné que c'est le seul endroit depuis lequel je n'ai pas réussi à me connecter).

Cette étape étant finalisée, je peux enfin faire fonctionner mon projet. Il ne me reste plus qu'à le faire fonctionner sur les différents systèmes d'exploitation que j'utilise.

VI. Phases de test sur les différents systèmes d'exploitation

Avant toute chose, il me semble nécessaire, arrivé à cette étape, de sécuriser mon système.
L'identification par SSH fonctionne avec un identifiant et un mot de passe. Or, l'ouverture du port 22 sur le routeur expose ma Raspberry aux dangers de l'extérieur, il me semble donc primordial de renforcer la sécurité (une sécurité par mot de passe pouvant être facilement brisée).

Pour cela, j'utiliserai des clés d'authentification publiques/privées. Je créerai des clés sur chaque machine cliente que je transfèrerai ensuite sur le serveur. Outre le fait de gagner en sécurité, cela me permettra d'ouvrir une session SSH depuis une machine cliente sans avoir à saisir de mot de passe, ce qui pourrait s'avérer utile pour automatiser le montage du système de fichiers.

Dans un premier temps, il est nécessaire de créer sur chaque machine cliente une clé SSH. Pour ce faire, j'utilise la commande suivante :

```
ssh-keygen –t rsa
```

Comme son nom l'indique, cette commande va générer une clé SSH avec une méthode de chiffrement RSA. Une fois cette clé créée et enregistrée dans le dossier .ssh du dossier utilisateur (clés appelées id_rsa pour la clé privée et id_rsa.pub pour la clé publique), je la transfère sur mon serveur. Pour ce faire, j'utilise justement le protocole SSH :

```
ssh-copy-id -i ~/.ssh/id_rsa.pub pi@guysmv.noip.me
```

Cette commande va copier la clé depuis ma machine cliente vers le serveur. Une fois mon mot de passe entré, le transfert se fait. Pour vérifier que tout fonctionne, je ferme le terminal SSH et j'en rouvre un autre : aucun mot de passe ne m'est demandé.

Une fois la procédure effectuée sur toutes mes machines clientes (comprendre sur tous mes OS), je peux désactiver l'authentification par mot de passe pour le protocole SSH sur la Raspberry. Celle-ci est alors complètement protégée de l'extérieur. Dans le cas d'un éventuel ajout d'une machine à mon serveur, je n'aurai qu'à me reconnecter avec une machine déjà autorisée et à réactiver l'authentification par mot de passe (ou utiliser un PC déjà identifié avec une clé pour transférer la

clé du PC à ajouter sur la Raspberry : c'est d'ailleurs la procédure que j'ai faite pour transférer la clé de mon installation Windows, celui-ci n'étant pas capable de faire cette tâche « tout seul »).

Je peux maintenant passer au paramétrage sur chacune de mes installations.

A. Paramétrage sur Ubuntu

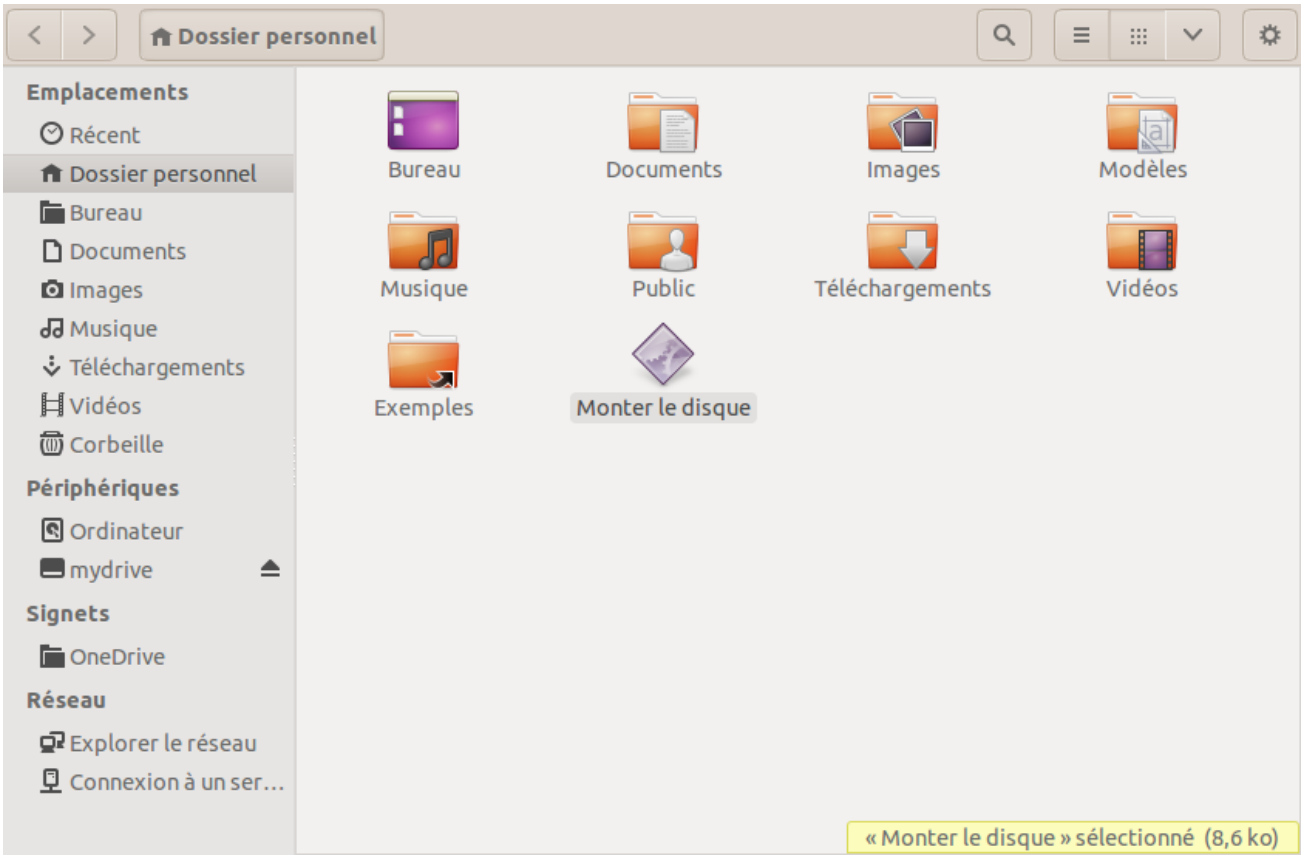
Le paramétrage sur Ubuntu est le plus facile à effectuer : tous les paquets nécessaires à l'exécution sont soit déjà installés par défaut soit directement disponibles dans les dépôts de la distribution. Ayant effectué les manipulations précédentes sur Ubuntu, tous les paquets nécessaires sont déjà installés (openssh-client, sshfs, les différents paquets nécessaires au fonctionnement de FUSE, etc.). En outre, la clé SSH est déjà enregistrée sur le serveur.

L'ouverture du système de fichiers est donc très simple. Je n'ai qu'à rentrer la commande suivante :

```
sshfs pi@guysmv.noip.me:/home/pi/mydrive /home/guillaume/Bureau/mydrive
```

Je constate donc que tous les paramétrages se sont faits correctement : le système de fichiers distant apparaît dans la barre latérale de Nautilus sans que je n'aie à rentrer de mot de passe.

Pour automatiser la tâche, la solution la moins contraignante que j'ai trouvée a été de créer un petit programme en C lançant cette commande. Je n'aurai ensuite qu'à exécuter ce programme pour monter mon espace de stockage. La solution est simple d'usage : quand j'ouvre ma session, après avoir vérifié que mon ordinateur est bien connecté à Internet : je lance le programme et le lecteur se monte automatiquement (*cf. page suivante*).



On voit bien ici le petit exécutable et le lecteur monté dans la barre latérale de Nautilus (« mydrive » dans les périphériques).

B. Paramétrage sur OS X

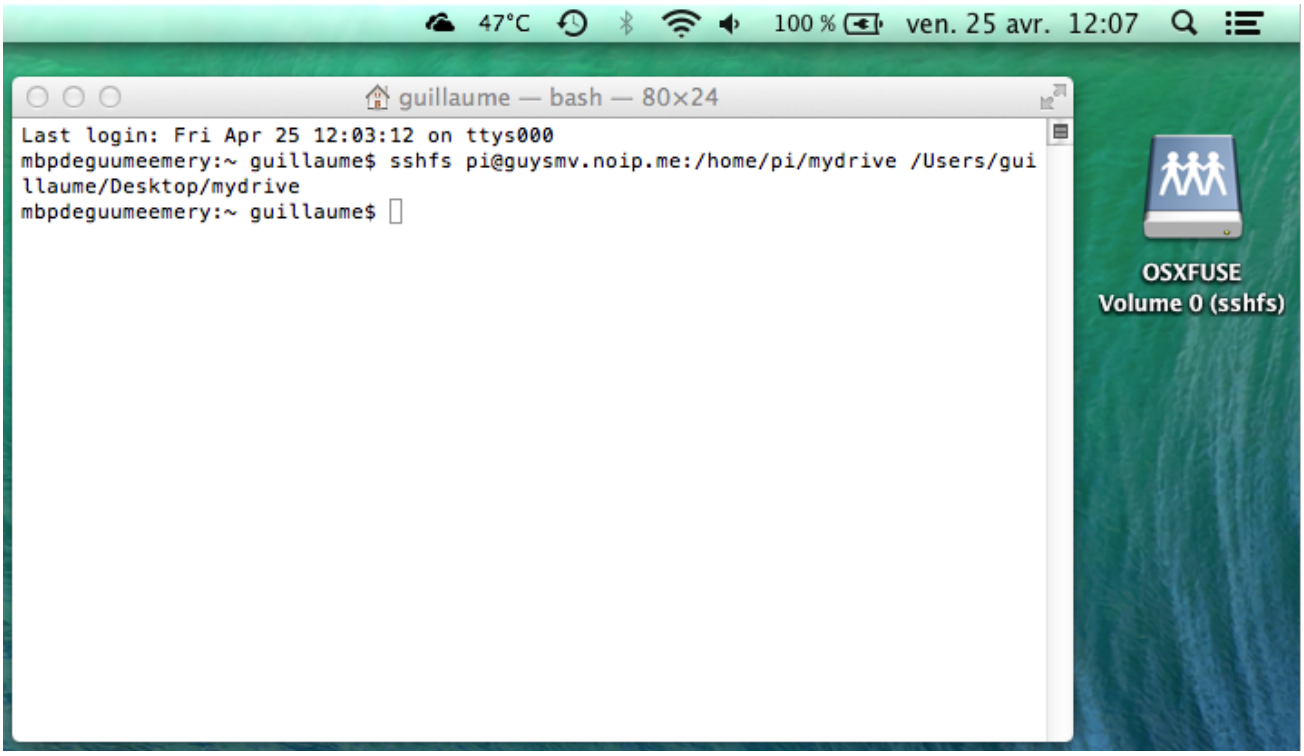
Le paramétrage sur OS X a (presque) été aussi simple que sur Linux. En effet, j'avais déjà un utilitaire de gestion de paquets sur ce système, fonctionnant à peu près comme `aptitude` pour les commandes à rentrer dans le terminal mais allant récupérer des sources sur GitHub pour ensuite les compiler. Ce système s'appelle **homebrew**.

Il m'a simplement fallu entrer la commande suivante :

```
sudo brew install osxfuse openssh sshfs ssh-copy-id
```

On peut remarquer à quel point le fonctionnement est similaire entre les deux systèmes (bien évidemment, ce n'est pas la politique d'Apple que de fournir un logiciel de gestion de paquets open-source qui passerait entre les mailles de son Mac App Store, mais une fois celui-ci installé, on peut se rendre compte que OS X n'est pas aussi fermé que ses créateurs l'auraient voulu).

Une fois ces paquets installés, j'ai pu renouveler exactement la même procédure que sur Ubuntu pour ajouter ma clé SSH à la Raspberry. Puis, j'ai lancé la même commande `sshfs` que sur Ubuntu : le dossier s'est automatiquement monté sur le bureau sans aucun problème :



Pour automatiser l'action, j'ai repris exactement le code en C que j'avais écrit sur Ubuntu. J'ai de même réutilisé `gcc -o` pour le compiler sur le système d'Apple, et, lorsque je double clique sur l'exécutable, le lecteur se monte automatiquement.

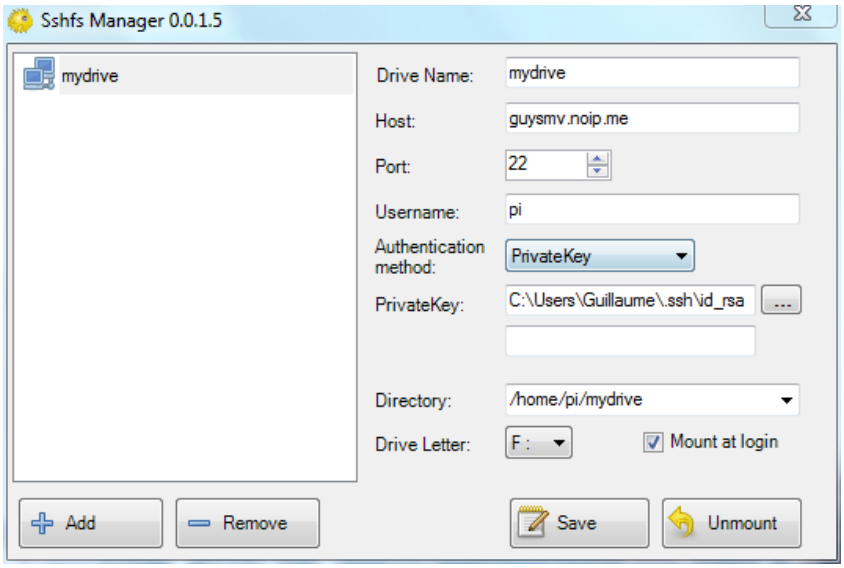
C. Paramétrage sur Windows

La configuration sur Windows se fait malheureusement « à l'ancienne » : on doit récupérer différents installeurs sur Internet et les installer à la main. Les deux logiciels à récupérer sont :

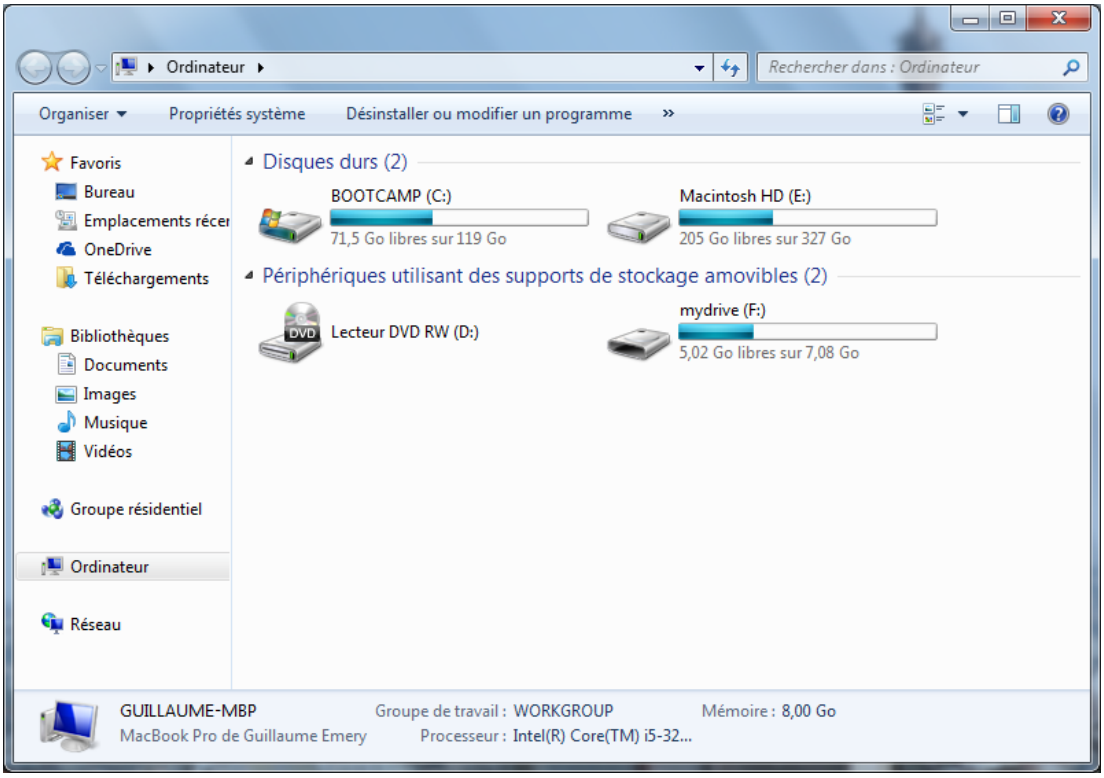
- OpenSSH for Windows : portage sur Windows du logiciel openSSH
- win-sshfs : de même, portage de sshfs sur Windows avec une interface graphique permettant de rentrer les différents paramètres (le terminal de Windows ne supportant pas cette option par défaut).

Une fois ceci fait, il a fallu créer une clé SSH de connexion. La commande pour la créer est la même que sur Ubuntu et sur OS X, mais pour la déplacer sur le serveur, la commande `ssh-copy-id` n'existe pas. J'ai donc copié la clé rsa_id.pub créé sur Windows et je l'ai transférée sur la Raspberry avec OS X exactement de la même façon que précédemment.

Une fois toutes ces étapes réalisées, j'ai pu redémarrer sur Windows pour vérifier le bon fonctionnement de mon programme. L'interface graphique de sshfs-manager se présente de la façon suivante :



On constate qu'ici, le montage se fait automatiquement au démarrage de Windows. Une fois la clé SSH retrouvée, le lecteur se monte sans problème :



VII. Optimisations

Maintenant que mon projet fonctionne sur mes trois systèmes d'exploitation, il est temps de l'optimiser.
Pour cela, j'ai eu plusieurs idées :

- Alléger la distribution Raspbian pour gagner de l'espace de stockage.
- Achever la protection de mon système de l'extérieur en désactivant la connexion par mot de passe en SSH sur ma Raspberry.

A. Alléger Raspbian

Raspbian, bien qu'étant légère, peut l'être encore plus. Mon idée est de retirer tous les paquets ne servant pas, notamment :

- L'interface graphique : LXDE
- Le serveur X (utilisant uniquement des commandes bash).
- Tous les logiciels (et services inutiles) installés par défaut (navigateur, fonctions multimédia…)

Cette optimisation a pour but principalement de gagner de la place sur mon espace de stockage mais aussi d'optimiser les performances globales du système (bien que j'imagine que cela ne joue pas énormément).

J'ai dû dans un premier temps afficher l'ensemble des paquets installés sur mon système. Pour cela, j'ai rentré la commande suivante :

```
dpkg --get-selections
```

`dpgk` est la base du gestionnaire de paquets de Debian (donc aussi de Raspbian). Cette commande permet de lister l'ensemble des paquets installés sur le système. Une fois cette liste apparue, j'ai pu faire le tri de ce dont ma Raspberry a besoin et ce dont elle n'a pas besoin. J'ai choisi tous les logiciels « graphiques » que je connais, ainsi que tous ceux qui concernent le serveur X et LXDE. Comme j'avais certains doutes, je suis allé vérifier sur les sites de Debian ou d'Ubuntu l'utilité de chaque paquet.

J'ai ensuite supprimé en plusieurs étapes, en faisant des autoremove entre chaque suppression.

```
sudo apt-get remove desktop-base galculator gnome-icon-theme gnome-
themes-standard-data hicolor-icon-theme lightdm lightdm-gtk-greeter
lxappearance lxde lxde-common lxde-core lxde-icon-theme lxinput lxmenu-
data lxpanel lxpolkit lxrandr lxsession lxsession-edit lxshortcut
lxtask lxterminal menu-xdg midori omxplayer
```

Les paquets ayant été automatiquement supprimés par la suite sont les suivants : `libfm-data liblightdm-gobject-1-0 libmenu-cache1 libqt4-network libqtwebkit4 libunique-1.0-0 libvte-common libvte9 libwnck-common libwnck22 libxklavier16 libxres1`

Ceci a principalement permis de retirer l'interface LXDE, qui, bien que ne prenant que peu de place, est inutile. J'ai également retiré quelques autres programmes inutiles par la même occasion.

J'ai enfin fini par retirer complètement le serveur X :

```
sudo apt-get remove openbox penguinspuzzle sonic-pi squeak-plugins-scratch squeak-vm supercollider supercollider-common supercollider-server x11-common x11-utils x11-xkb-utils xarchiver xdg-utils xfonts-encodings xfonts-utils xkb-data xpdf xserver-common xserver-xorg xserver-xorg-core xserver-xorg-input-all xserver-xorg-input-evdev xserver-xorg-input-synaptics xserver-xorg-video-fbdev xserver-xorg-video-fbturbo
```

Les paquets suivants ont ensuite été retirés avec un `autoremove` :

```
aspell aspell-en console-setup-linux cups-bsd cups-client dconf-gsettings-backend dconf-service dictionaries-common esound-common fonts-droid ghostscript gir1.2-glib-2.0 glib-networking glib-networking-common glib-networking-services gsettings-desktop-schemas gsfonts gvfs-common gvfs-libs iso-codes jackd jackd2 libarchive12 libaspell15 libasyncns0 libaudiofile1 libaudit0 libavahi-glib1 libbluetooth3 libbluray1 libcaca0 libcairo-gobject2 libcdio-cdda1 libcdio-paranoia1 libcdio13 libcolord1 libcupsimage2 libcwiid1 libdconf0 libdirectfb-1.2-9 libdrm2 libenchant1c2a libesd0 libexif12 libfftw3-3 libflac8 libfontenc1 libgail-3-0 libgail18 libgd2-xpm libgdu0 libgeoclue0 libgif4 libgirepository-1.0-1 libgl1-mesa-glx libglapi-mesa libgphoto2-2 libgphoto2-port0 libgs9 libgs9-common libgstreamer-plugins-base0.10-0 libgstreamer0.10-0 libgtk-3-0 libgtk-3-bin libgtk-3-common libhunspell-1.3-0 libicu48 libid3tag0 libijs-0.35 libimlib2 libimobiledevice2 libjack-jackd2-0 libjavascriptcoregtk-1.0-0 libjavascriptcoregtk-3.0-0 libjbig2dec0 libjson0 liblcms1 liblcms2-2 libltdl7 libmad0 libmikmod2 libmng1 libmtdev1 libnettle4 libnotify4 libobt0 libogg0 libopenjpeg2 liborc-0.4-0 libpaper-utils libpaper1 libpciaccess0 libplist1 libpoppler19 libportmidi0 libproxy0 libqt4-dbus libqt4-xml libqtcore4 libqtdbus4 libruby1.9.1 libsclang1 libscsynth1 libsmbclient libsndfile1 libsoup-gnome2.4-1 libsoup2.4-1 libts-0.0-0 libusbmuxd1 libutempter0 libvorbis0a libvorbisenc2 libvorbisfile3 libwayland0 libwebkitgtk-1.0-common libwebkitgtk-3.0-common libwebp2 libxcb-glx0 libxcb-shape0 libxcb-xfixes0 libxfont1 libxkbfile1 libxp6 libxpm4 libxslt1.1 libxv1 libxxf86dga1 libxxf86vm1 libyaml-0-2 menu poppler-data poppler-utils pypy-upstream-doc python-dbus python-dbus-dev python-gi python-numpy python-support qdbus ruby1.9.1 tcl8.4 tcl8.5 tk8.4 tsconf usbmuxd xbitmaps zenity-common
```

Une fois la manipulation effectuée et après un redémarrage, mon système, heureusement, fonctionne encore. La commande `df -h` m'informe que le système pèse moins d'un giga-octet. J'ai donc décidé de m'arrêter là pour les optimisations.

B. Protection du système

Maintenant que mon système est optimisé et que l'accès est sécurisé par des clés d'authentification SSH, il m'est possible de désactiver l'authentification par mot de passe lors d'une connexion SSH entrante.
Cette fonction s'ajoute au fichier de configuration SSH sur la Raspberry. Dans un shell SSH, j'ai entré la commande suivante :

```
sudo nano /etc/ssh/ssh_config
```

Il ne me reste plus qu'à ajouter une option à ce fichier de configuration, permettant d'interdire cette authentification :

```
PasswordAuthentication no
```

Ensuite, je redémarre la Raspberry : pour vérifier que cette option a bien été prise en compte, j'ai d'abord utilisé mon ordinateur autorisé avec une clé SSH pour me connecter : la connexion se fait toujours. Ensuite, j'ai utilisé une live USB pour démarrer sur une machine Linux non autorisée : la connexion est alors impossible.

VIII. Conclusion

Ce projet a pour moi été un véritable plaisir : il me tient à cœur depuis quelques temps et j'ai réussi à en poser les bases. Il m'a permis de réutiliser les notions vues en cours et d'y apporter les connaissances que je possédais déjà.

Je suis satisfait de mon travail : l'ensemble est utilisable et j'ai déjà stocké un certain nombre de fichiers dessus. Bien que de capacité de stockage faible pour l'instant, il remplace aisément une clé USB.

Je vois encore un certain nombre de points à améliorer. J'envisage à terme de créer un système de synchronisation et de sauvegarde : c'est mon objectif pour les vacances de cet été, et le travail déjà effectué me met en confiance pour la suite. J'ai aussi pour projet d'ajouter une fonction de connexion par simple FTP pour permettre la lecture de vidéos à distance depuis VLC (celui-ci ne supportant pas le protocole SSH dans sa version Smartphone).

Pour finir, ce projet m'a rouvert au monde du logiciel libre, que je n'utilise plus depuis deux ans, étant passé sur un système Mac. Souhaitant m'orienter dans une majeure SI, ce projet m'a donné un aperçu de ce qui m'attend et je m'en réjouis d'avance.

IX. Sources

http://doc.ubuntu-fr.org/dd : pour créer mes différents supports d'installation

http://elinux.org/RPi_Easy_SD_Card_Setup : installation d'une distribution Linux sur une carte SD compatible avec la Raspberry

http://www.raspberrypi.org/downloads/ : téléchargement et informations sur Raspbian (notamment le login)

http://doc.ubuntu-fr.org/fuse#installation_de_fuse : installation et configuration de FUSE

http://doc.ubuntu-fr.org/sshfs : utilisation de SSHFS

http://fr.wikipedia.org/wiki/DynDNS : explication simple sur le DynDNS

http://fr.wikipedia.org/wiki/Network_address_translation : fonctionnement de la translation de ports

http://packages.ubuntu.com / https://packages.debian.org/fr/sid/allpackages : descriptions des paquets essentiels à ne pas supprimer