Antisèches Informatiques

« Mapper un serveur ftp dans l'explorateur Windows - Travailler sur un grand nombre de fichiers sous »

Configuration et utilisation de openssh

Par Rémi TAUPIN le vendredi 23 janvier 2009, - Lien permanent

commandes , debian , etch , firefox , lenny , linux , macosx , proxy , reseaux , securite , squeeze , ssh , systeme , windows

Openssh fournit une suite d'outils clients et serveurs qui permettent de faire des échanges sécurisés entre des machines à travers un réseau non sécurisé. Openssh est à la fois un protocole réseau et une suite d'exécutables tels que **sshd** et **sftpd** pour les logiciels serveurs, et **ssh**, **sftp** ou encore **scp** pour les logiciels clients.

Dans une utilisation de base, openssh permet de remplacer **telnet** et les outils **rlogin**, mais on peut également l'utiliser pour monter des tunnels vpn.

Cet article présente l'installation et l'utilisation de openssh sous **Linux**, **Windows** et **MacOSX** en utilisant une authentification simple par mot de passe puis une authentification grâce à une paire de clés privée/publique.

Il présente également la création de tunnels sécurisés et le montage de partitions distantes avec sshfs.

Sommaire

Openssh pour linux

Installation

Avec une distribution Debian, il est trés simple d'installer openssh :

```
homeserver:~# apt-get install ssh
```

La procédure d'installation dure quelques secondes au cours desquelles le serveur et le client ssh sont installés. Une fois terminée, tout est prêt pour pouvoir commencer à

travailler. La syntaxe pour se connecter à un serveur distant est :

```
client:~# ssh -l user serveur.exemple.com
```

ou

```
client:~# ssh user@serveur.exemple.com
```

Si l'utilisateur local et distant ont le même nom, il n'est pas utile de spécifier **user**. La ligne de commande devient alors :

```
client:~# ssh serveur.exemple.com
```

Lors de la première connexion à un serveur, le client ssh avertit que l'identité de ce serveur ne peut pas être vérifiée. C'est normal car il ne le connait pas encore. Il demande alors une confirmation :

```
The authenticity of host 'serveur.exemple.com (192.168.0.1)' can't be
established.
RSA key fingerprint is fa:0a:7e:04:bc:19:19:67:e1:46:51:eb:12:c4:5b:97.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'client.domaine.com' (RSA) to the list of known
hosts.
Password: ********
client:~# exit
```

Après confirmation, le client ssh ajoute l'empreinte de la clé (**key fingerprint**) dans le fichier **~/.ssh/known_hosts** de la machine cliente **client.domaine.com**. Par exemple si l'utilisateur est **root** et que le PC client fonctionne sous linux, le fingerprint du serveur sera ajouté au fichier "**/root/.ssh/known_hosts**".

On peut maintenant se connecter au serveur distant avec la commande `ssh root@serveur.exemple.com` (si on veut utiliser le compte root).

```
client:~# ssh root@serveur.exemple.com
Password: ********
Last login: Sun Dec 10 15:17:44 2006 from 192.168.0.10
client:~#
```

Authentification par bi-clés

Si on se connecte fréquemment sur ce serveur, il peut être pénible de rentrer le mot de passe à chaques fois. On va donc créer une paire de clés privée/publique et on va configurer le serveur pour qu'il nous reconnaisse automatiquement grâce à notre clé publique et ne nous demande plus de mot de passe.

Pour ce faire on utilise la commande `ssh-keygen` :

```
client:~# ssh-keygen -t rsa -b 1024 -C "Cle_rsa_1024_bits_de_root"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/root/.ssh/id_rsa): [Enter]
Enter passphrase (empty for no passphrase):[Enter]
Enter same passphrase again:[Enter]
Your identification has been saved in /home/root/.ssh/id_rsa.
Your public key has been saved in /home/root/.ssh/id_rsa.pub.
The key fingerprint is:
1a:2a:3e:4a:1a:aa:cb:29:f3:1f:a1:06:c7:8e:e9:58 Cle_rsa_1024_bits_de_root
```

Explication de la commande :

`ssh-keygen` : commande pour générer une paire de clés
`-t rsa` : on veut une clé de type rsa
`-b 1024` : la longeur de la clé est de 1024 bits
`-C "Cle_rsa...."` : c'est un commentaire optionnel pour identifier la clé. Il sera utile

pour l'identifier dans le fichier **~/.ssh/authorized_keys** (voir plus loin pour l'explication de ce fichier).

Note au sujet de la passphrase qui est demandée : cette passphrase et une sécurité supplémentaire au cas où la clé privée serait dérobée. Le problème est que même si elle augmente la sécurité du systéme, elle en complique l'utilisation dans le sens où elle oblige à mettre en place un autre mécanisme, **ssh-agent** sur le serveur distant. On la laissera donc vide.

Maintenant que les clés sont générées, il faut copier la clé publique sur le serveur distant. On peut le faire en utilisant l'utilitaire de copie sécurisée **scp** qui fait partie des outils fournis en standard par openssh :

```
client:~# scp -P 22 /root/.ssh/id_rsa.pub root@serveur.exemple.com:/root
/.ssh/idclient_rsa.pub
Password: *******
/root/.ssh\i 100% 230 0.2KB/s 00:00
```

Si le serveur ssh distant fonctionne sur le port tcp par défaut (le port 22) on peut omettre le paramètre `-P 22`.

/! Attention, dans la commande ci-dessus le fichier de destination est bien **idclient_rsa.pub** et non pas **id_rsa.pub**. En effet un fichier **id_rsa.pub** existe surement et on risque de l'écraser. Et celui qui est déjà présent est certainement la clé publique rsa de l'administrateur du serveur. Ce serait dommage de l'écraser...

Maintenant que la clé publique est copiée, on ajoute son contenu au fichier **/root/.ssh /authorized_keys** du serveur. Toujours à partir de la machine client, on utilise les commandes suivantes :

```
client:~# ssh root@serveur.exemple.com "cat /root/.ssh/idclient_rsa.pub >>
/root/.ssh/authorized_keys"
Password: ********
```

Comme dans la commande précédente, si le serveur ssh distant fonctionne sur le port tcp par défaut (le port 22) on peut omettre le paramètre `-p 22`. Notons que dans la commande `scp` on écrit `-P 22` (P majuscule) alors que pour la commande `ssh` on écrit `-p 22` (p minuscule).

/! Attention, dans la commande ci-dessus il faut bien mettre **>>** et non pas **>** sinon on écrase le fichier **authorized_keys** du serveur distant au lieu de le mettre à jour…

Si tout a fonctionné, on peut maintenant se connecter sans être obligé de rentrer le mot de passe :

```
client:~# ssh root@serveur.exemple.com
Last login: Sun Dec 10 20:18:09 2006 from 192.168.0.10
serveur:~#
serveur:~# exit
```

Ca marche, le serveur distant ne demande pas de mot passe !

Si on se connecte souvent sur ce serveur, il peut être pénible de rentrer à chaques fois la commande complète `ssh user@serveur.exemple.com`. Pour simplifier la ligne de commande, il suffit de créer ou de modifier le fichier **~/.ssh/config** de la façon suivante :

```
Host serveur
Port 22
User root
HostName serveur.exemple.com
```

A partir de maintenant, un simple `ssh serveur` suffira pour ce connecter sur la machine **serveur** en question. Le paramètre **Port 22** est facultatif si le serveur fonctionne sur le port par défaut.

Nous pouvons maintenant aller voir plus bas pour un résumé des commandes de base (http://www.antiseches.net/#cdebases) ou pour apprendre à gérer des tunnels ssh (http://www.antiseches.net/#tunnels) .

Openssh pour MacOSX

MacOSX est maintenant un beau système d'exploitation Unix Like puisqu'il est basé sur FreeBSD 4.4. C'est d'ailleurs un des systèmes Unix le plus utilisé dans le monde.

Sous MacOS 10.5, openssh est déjà installé en standard et il fonctionne exactement comme sous linux.

Pour l'utiliser en ligne de commande, il faut ouvrir un shell en allant dans **Finder / Applications / Utilitaires** et en cliquant sur l'icone **Terminal**.

Ensuite, comme sous linux on utilise la commande `ssh` pour se connecter à un serveur distant.

Le démon **sshd** de MacOSX est présent mais n'est pas lancé au démarrage. Pour le lancer manuellement, il faut utiliser la commande `service ssh start`. On peut alors se connecter à la machine avec un client ssh.

Openssh pour Windows

Openssh pour Windows est le portage win32 de openssh. Il s'utilise exactement comme sous linux. Il faut cependant noter que sa version n'a pas évolué depuis longtemps, contrairement à la version pour linux.

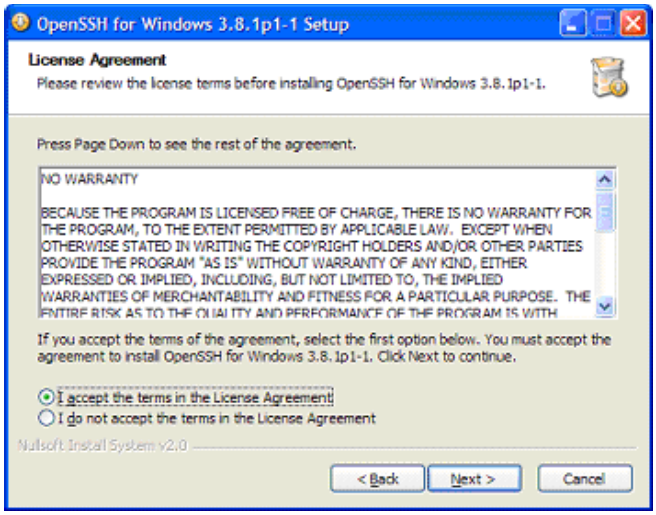
Téléchargement du logiciel

http://sshwindows.sourceforge.net/ (http://sshwindows.sourceforge.net/)
http://sshwindows.sourceforge.net/download/ (http://sshwindows.sourceforge.net/download/)

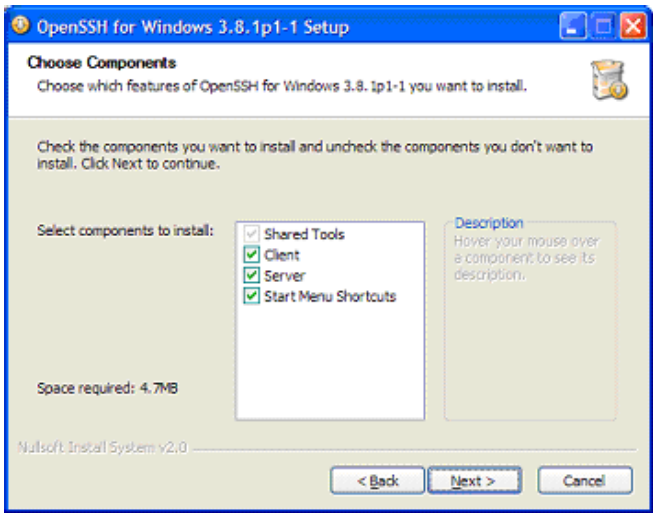
Installation de openssh pour win32

Faire un double-clic sur l'icone du programme d'installation et suivre les étapes suivantes :

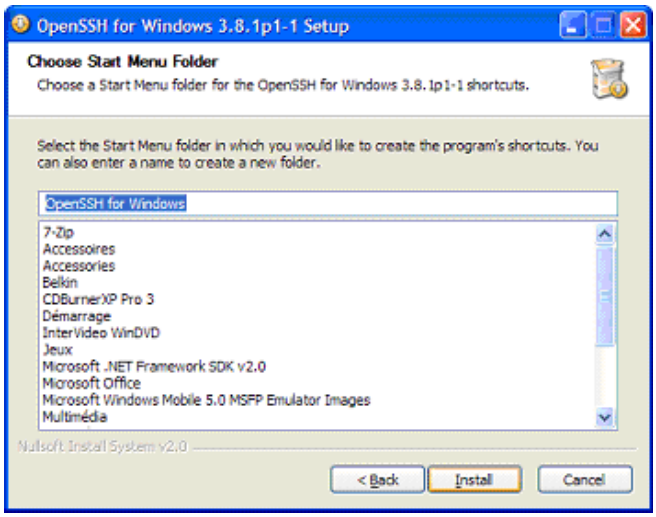
Lire et accepter les termes du contrat :

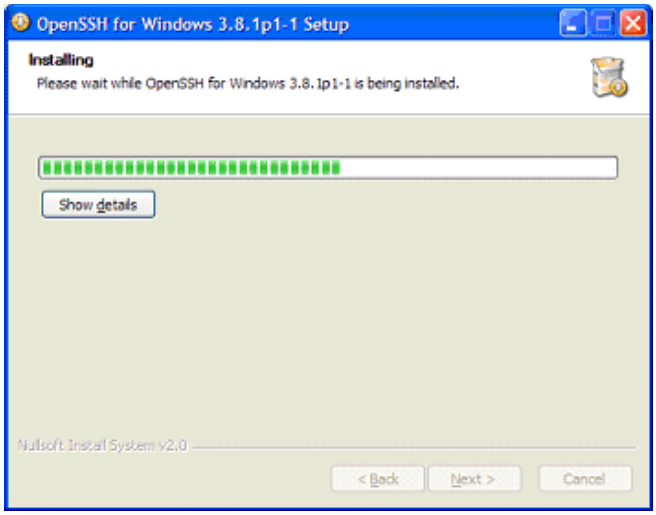


Choisir les composants à installer. Le serveur ssh ne sera sélectionné que si on désire se connecter en ligne de commande sur cette machine :

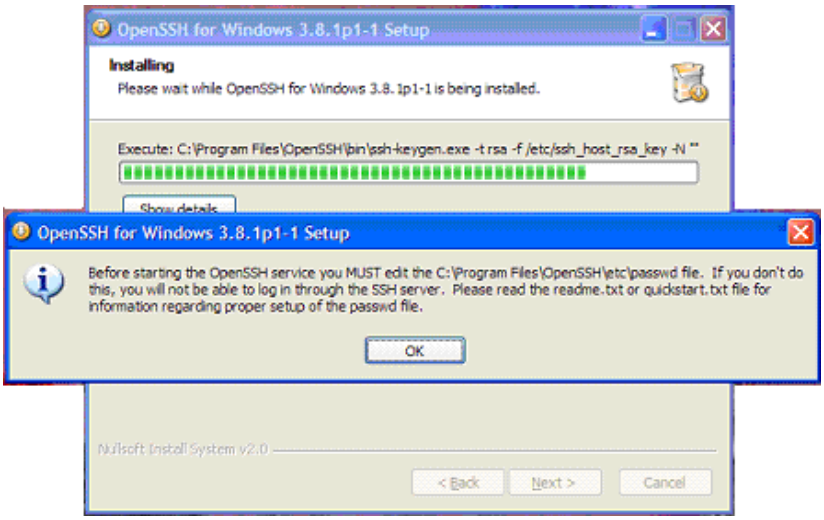


En général, on laisse toutes les options par défaut :



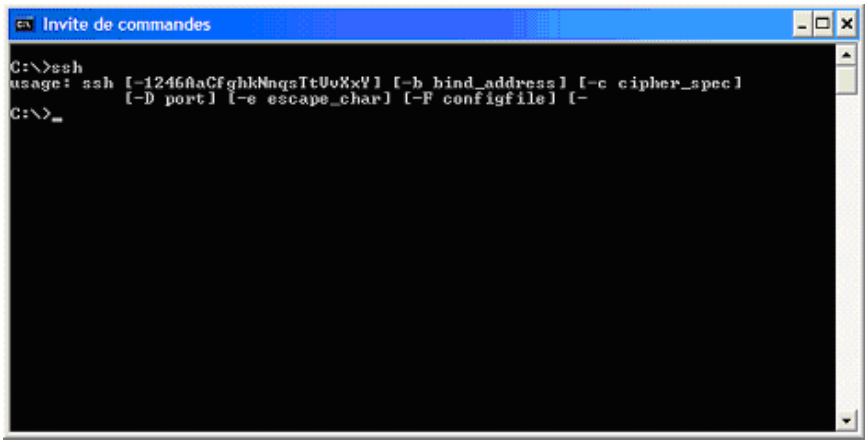


Si on a choisit d'installer la partie server d'open ssh, il faudra importer et configurer les comptes des utilisateurs qui seront autorisés à ce connecter sur cette machine. Le programme d'installation montre quel est le fichier qui explique la marche à suivre :



L'installation est maintenant terminée :

Sans plus de paramètrages, le client openssh fonctionne déjà. Pour l'utiliser, il suffit d'ouvrir une **Invite de Commande** :



Lors de la première connexion à un serveur, le client ssh avertit que l'identité de ce serveur ne peut pas être vérifié, ce qui est normal car il ne le connait pas encore. Il demande alors une confirmation.

Après confirmation, le client ssh ajoute l'empreinte de la clé (**key fingerprint**) dans le fichier **%profile%\.ssh\known_hosts** de la machine cliente. Par exemple si l'utilisateur est "**Administrateur**" et que le PC tourne sous Windows XP, le fingerprint sera dans le fichier "**C:\Documents and Settings\Administrateur\.ssh\known_hosts**" :

```
C:\>ssh user@serveur.exemple.com
The authenticity of host 'serveur.example.com (192.168.0.1)' can't be established.
RSA key fingerprint is a8:9a:7e:04:ab:19:19:67:e4:36:52:eb:12:f4:50:a7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'serveur.example.com, 192.168.0.1' (RSA) to the list of known hosts.
Password: ********
serveur:~# exit
```

Lors de la deuxième connexion au serveur ssh, l'identité du serveur sera vérifiée et si elle est confirmée, il n'y aura pas de question supplémentaire :

```
C:\>ssh user@serveur.exemple.com
Password: ********
Last login: Sun Dec 10 15:17:44 2006 from 192.168.0.10
serveur:~#
```

Authentification par bi-clés

Si on se connecte fréquemment sur ce serveur, il peut être pénible de rentrer le mot de passe à chaques fois. On va donc créer une paire de clés privée/publique et on va configurer le serveur pour qu'il nous reconnaisse grâce à notre clé au lieu du mot de passe.

Pour ce faire on utilise la commande `ssh-keygen` :

```
C:\>ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/Administrateur/.ssh/id_rsa):[Enter]
Enter passphrase (empty for no passphrase):[Enter]
Enter same passphrase again:[Enter]
Your identification has been saved in /home/Administrateur/.ssh/id_rsa.
Your public key has been saved in /home/Administrateur/.ssh/id_rsa.pub.
```

```
The key fingerprint is:
1a:2a:3e:4a:1a:aa:cb:29:f3:1f:a1:06:c7:8e:e9:58
Administrateur@machinecliente
```

Comme pour openssh pour linux, cette commande crée une paire de clés dsa. Avec la version win32 de openssh, on ne peut pas mettre d'autres paramètres que `-t rsa` sur la ligne de commande sinon on a un message d'erreur.

Si on lit la sortie générée par la commande précédente, on voit qu'elle a créé les 2 fichiers suivants :
- La clé publique RSA **id_rsa.pub** de l'utilisateur (ici administrateur) dans le répertoire **/home/Administrateur/.ssh/**
- La clé privée RSA **id_rsa** de l'utilisateur (toujours administrateur) dans le répertoire**/home/Administrateur/.ssh/**

En fait ce n'est pas tout à fait vrai : les clés ont été créées dans **C:\Documents and Settings\Administrateur\.ssh** au lieu de **/Home/Administrateur**, mais le message de sortie de la commande est faux. Il a été fait pour linux et n'a pas été modifié lors de son portage pour win32…

Maintenant que les clés sont générées, il faut copier la clé publique sur le serveur distant. On peut le faire en utilisant l'utilitaire de copie sécurisée **scp** qui fait partie des outils fournis en standard par openssh :

```
C:\>scp "\Documents and Settings\Administrateur\.ssh\id_rsa.pub"
root@serveur.exemple.com:/root/.ssh/idclient_rsa.pub
Password: *******
\Documents and Settings\Administrateur\.ssh\i 100%  230     0.2KB/s   00:00
```

Attention, dans la commande ci-dessus le fichier de destination est bien **idclient_rsa.pub** et non pas **id_rsa.pub**. En effet un fichier **id_rsa.pub** existe surement et on risque de l'écraser. Et celui qui est déjà présent est certainement la clé publique rsa de l'administrateur du serveur…

Maintenant que la clé est copiée, on ajoute son contenu au fichier **/root/.ssh /authorized_keys** :

```
C:\>ssh root@serveur.exemple.com "cat /root/.ssh/idclient_rsa.pub >>
/root/.ssh/authorized_keys"
Password: ********
```

Attention, dans la commande ci-dessus il faut bien mettre **>>** et non pas **>** sinon on écrase le fichier **authorized_keys** du serveur distant au lieu de le mettre à jour…

Si tout a fonctionné, on peut maintenant se connecter sans être obligé de rentrer le mot de passe :

```
C:\>ssh root@serveur.exemple.com
Last login: Sun Dec 10 20:18:09 2006 from 192.168.0.10
serveur:~#
serveur:~# exit
```

Ca marche !

On peut maintenant effacer la clé du client sur le serveur :

```
C:\>ssh root@serveur.exemple.com "rm /root/.ssh/idclient_rsa.pub"
C:\>
```

Bien sûr aucun mot de passe n'est demandé et il n'y a pas de demande de confirmation lors de l'effacement du fichier sur le serveur distant. Prudence donc…

Si on se connecte souvent sur ce serveur, il peut être pénible de rentrer à chaques fois la commande complète `ssh root@serveur.exemple.com`. Pour simplifier la ligne de commande, il suffit de créer ou de modifier le fichier **C:\Documents and Settings\Administrateur\.ssh\config** de la façon suivante :

```
Host serveur
Port 22
User root
HostName serveur.exemple.com
```

A partir de maintenant, un simple `ssh serveur` suffira pour ce connecter sur la machine en question. Le paramètre **Port 22** est facultatif si le serveur fonctionne sur le port par défaut.

Nous pouvons maintenant aller voir plus bas pour un résumé des commandes de base (http://www.antiseches.net/#cdebases) ou pour apprendre à gérer des tunnels ssh (http://www.antiseches.net/#tunnels) .

PuTTY pour Windows

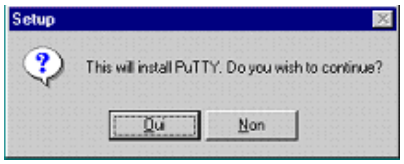
PuTTY est un client ssh et telnet graphique pour systèmes X-Window et pour windows. Il permet de se connecter à un serveur ssh en cliquant sur une icone. On peut également gérer des profiles, créer des tunnels, émuler des touches de fonctions, etc. Bref il est trés complet et trés simple d'utilisation. Un autre de ses avantages par rapport à openssh pour windows est de permettre les copier/coller trés facilement : la copie se fait tout simplement en sélectionnant le texte alors que le collage se fait avec un clic droit sur la souris.

Téléchargement du logiciel

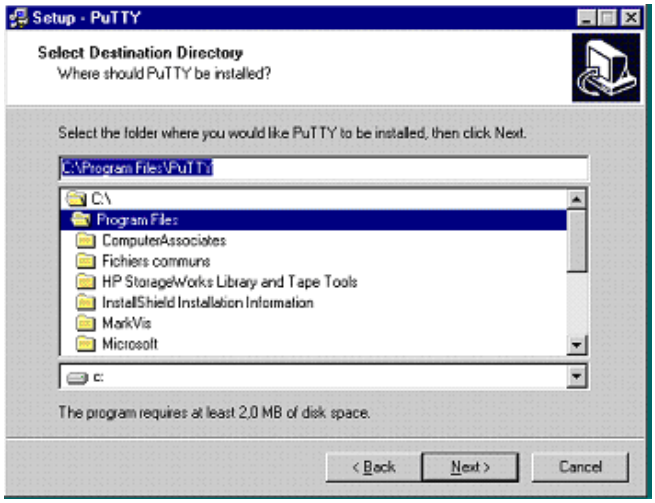
http://www.chiark.greenend.org.uk/~sgtatham/putty/
(http://www.chiark.greenend.org.uk/%7Esgtatham/putty/)
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
(http://www.chiark.greenend.org.uk/%7Esgtatham/putty/download.html)

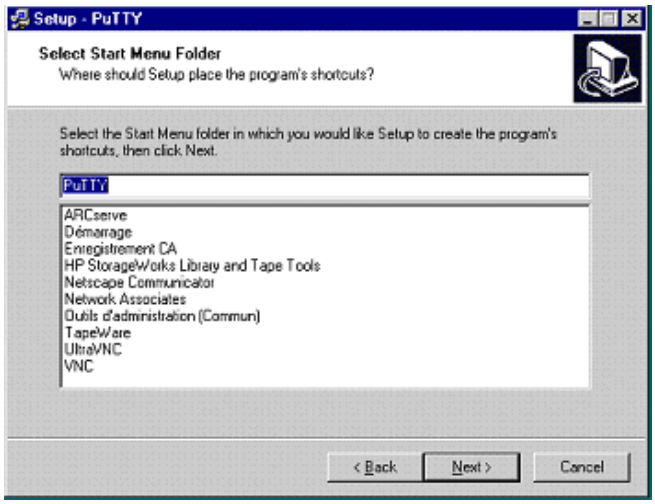
Installation

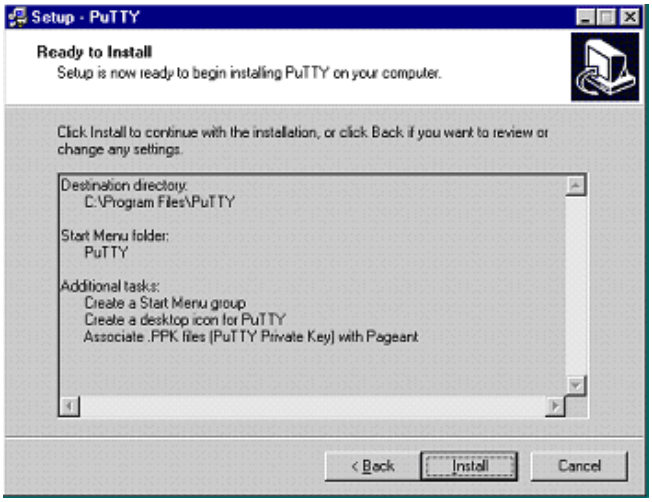
Démarrer l'installation de PuTTY en double-cliquant sur l'icone, et suivre les étapes suivantes :

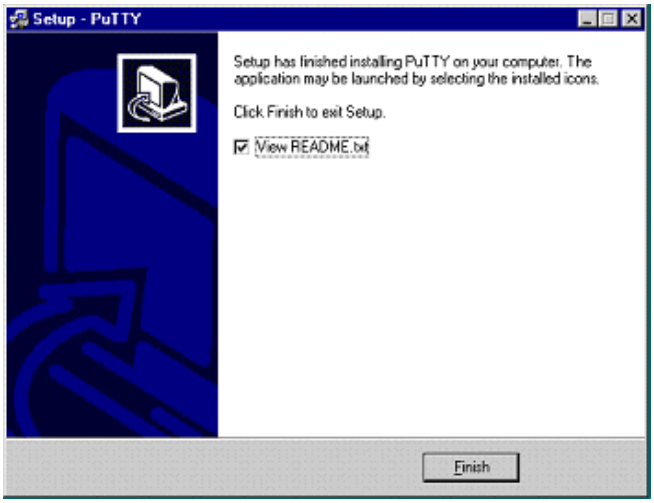



Setup - PuTTY
Welcome to the PuTTY Setup Wizard
This will install PuTTY version 0.57 on your computer.
It is strongly recommended that you close all other applications you have running before continuing. This will help prevent any conflicts during the installation process.
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel


Setup - PuTTY
Select Destination Directory
Where should PuTTY be installed?
Select the folder where you would like PuTTY to be installed, then click Next.
C:\Program Files\PuTTY
C:\
Program Files
ComputerAssociates
Fichiers communs
HP StorageWorks Library and Tape Tools
InstallShield Installation Information
MarkVis
Microsoft
c:
The program requires at least 2,0 MB of disk space.
< Back
Next >
Cancel


Setup - PuTTY
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Select the Start Menu folder in which you would like Setup to create the program's shortcuts, then click Next.
PuTTY
ARCserve
Démarrage
Enregistrement CA
HP StorageWorks Library and Tape Tools
Netscape Communicator
Network Associates
Outils d'administration (Commun)
TapeWare
UltraVNC
VNC
< Back
Next >
Cancel





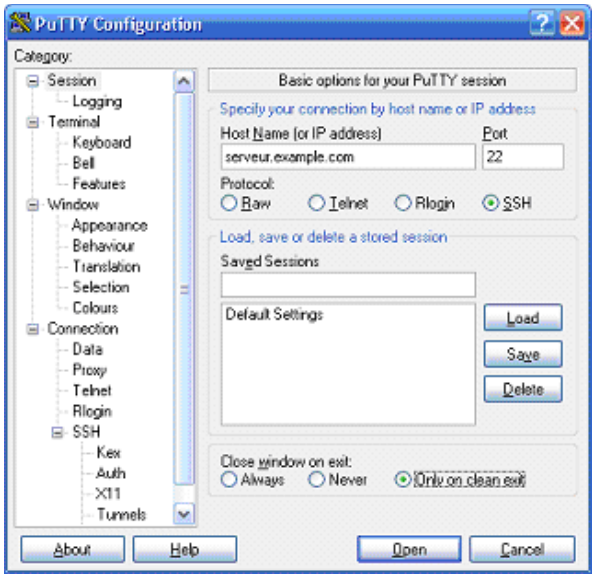
PuTTY est maintenant installé.

Utilisation de base

Maintenant que PuTTY est installé, on peut commencer à l'utiliser de facon tout à fait basique pour se connecter à une machine distante, soit en telnet soit en ssh. Pour cela, on lance le logiciel et on remplit les champs suivants :

Host Name or IP address : le non FQDN de la machine distante ou son adresse IP. Comme pour openssh, on peut mettre **user@serveur.exemple.com** pour se connecter avec un nom d'utilisateur particulier. Si on ne met pas **user@**, ne nom d'utilisateur sera demandé à chaque connexion.
Protocol : Telnet ou SSH. le n° de port par défaut change alors automatiquement : 22 pour SSH et 23 pour telnet.
Saved Session : un nom pour identifier la session si on désire la sauvegarder.
Close window on exit : permet de fermer ou non le fenêtre lorsque le connexion est fermée. Généralement on sélectionne **Only on clean exit** ce qui permet de garder les infos à l'écran en cas de coupure réseau.

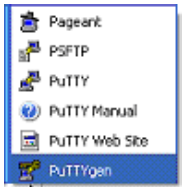
Si on veut sauvegarder la session, on appuie sur le bouton **Save**.



Création d'une paire de clés

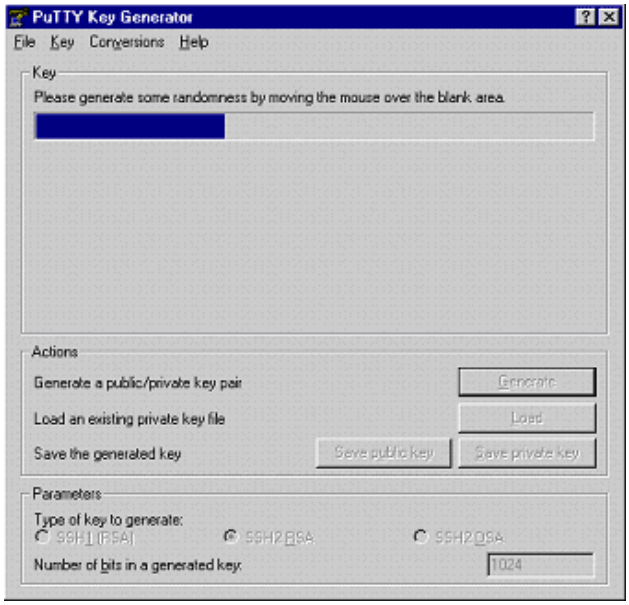
Pour éviter de se connecter en étant obligé de saisir le mot de passe à chaques fois, nous allons comme pour openssh créer une paire de clés et les utiliser pour s'authentifier.

Pour créer cette paire de clés, on utilise le programme PuTTYgen qui est fournit avec PuTTY :

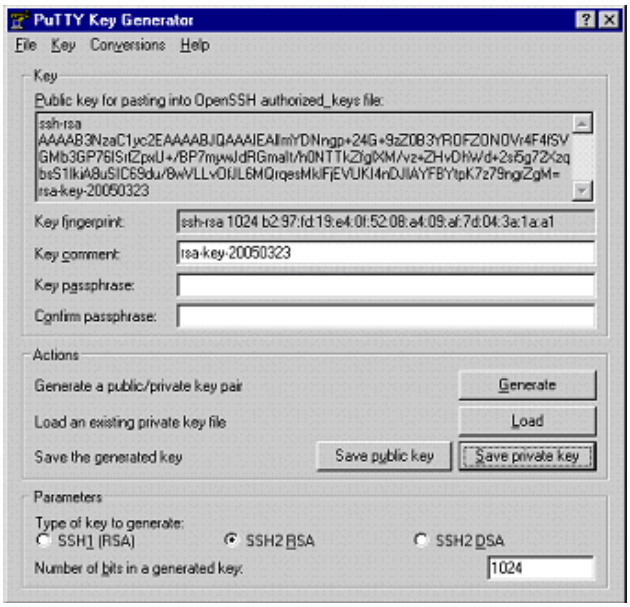


Une fois lancé, le programme ressemble à ca :

Aprés avoir sélectionné le type et la longueur de la clé dans **Parameters**, on appuie sur le bouton **Gererate**. Le programme demande alors de faire des mouvements aléatoires avec la souris pour que les clés soient générées :

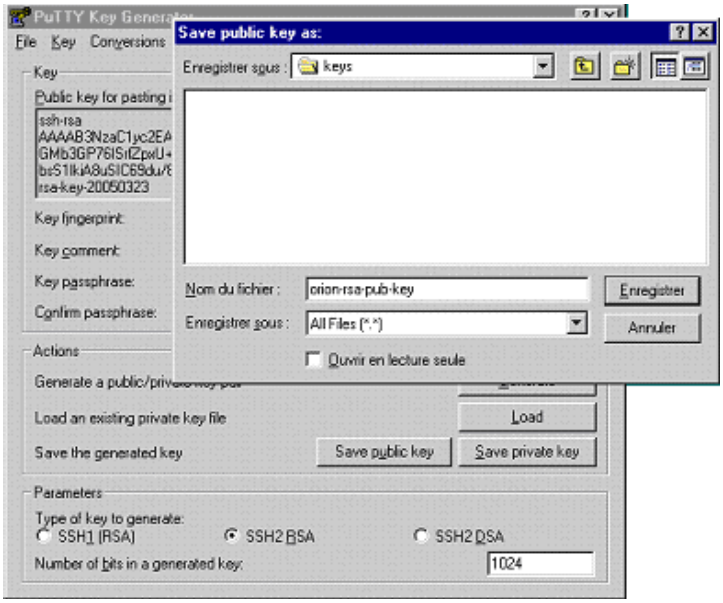


A la fin, on a la fenêtre suivante :

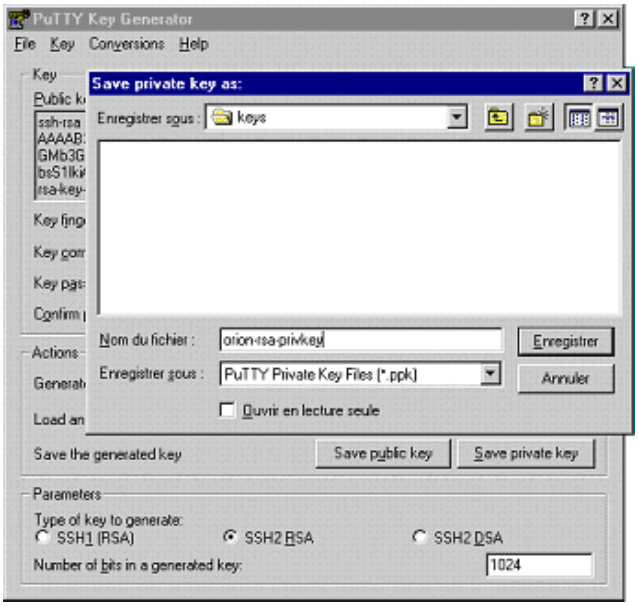


On entre alors un éventuel commentaire dans le champ **Key Comment** puis une passphrase dans les champs **Key passphrase** et **Confirm passphrase**. Bien qu'elle augmente la sécurité, l'utilisation d'une passphrase complique l'authentification par clés. Nous n'en mettons donc pas.

Ensuite, on sauvegarde les clés sur le disque dur :

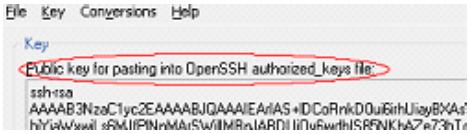


Sauvegarde de la clé publique



Sauvegarde de la clé privée

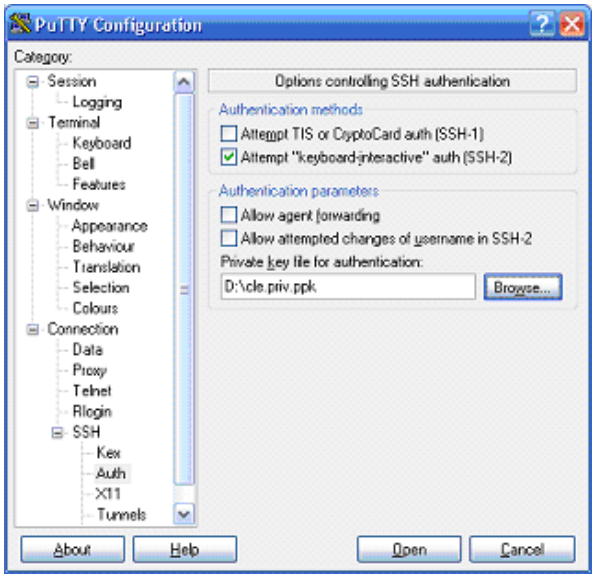
Une fois les clés sauvegardées, il faut copier le contenu de la clé publique sur le serveur distant. Attention, avec PuTTY on ne copie pas le fichier en entier, mais on fait un copier/coller de la chaine de caractères qui se trouve dans le champ **"Public key for pasting into OpenSSH authorized_keys file"** :



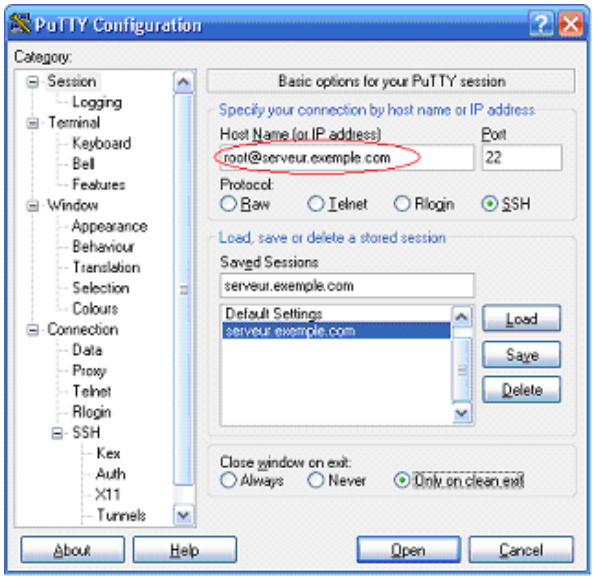
Pour copier le contenu dans le fichier en question, il y a plusieurs méthodes, mais la plus simple est de :
- Se connecter au serveur en ssh avec PuTTY.
- Editer le fichier **~/.ssh/authorized_keys** par exemple avec **vi**.
- Copier la chaine de caractères à la fin du fichier en faisant un clic droit sur la souris.

Configuration de PuTTY pour se connecter sans mot de passe

Maintenant que la clé est copiée sur le serveur distant, il faut configurer la session dans PuTTY en indiquant le chemin vers la clé privée que l'on vient de créer et qui est stockée sur le disque local.
On charge donc la session dans PuTTY en la sélectionnant dans **Saved Sessions** et en cliquant sur le bouton **Load.** Ensuite on va dans la fenêtre de gauche, on déroule l'arborescence pour aller dans **Connection / SSH / Auth** puis dans la case **Private key for file Authentication** et on indique le chemin de la clé privée :



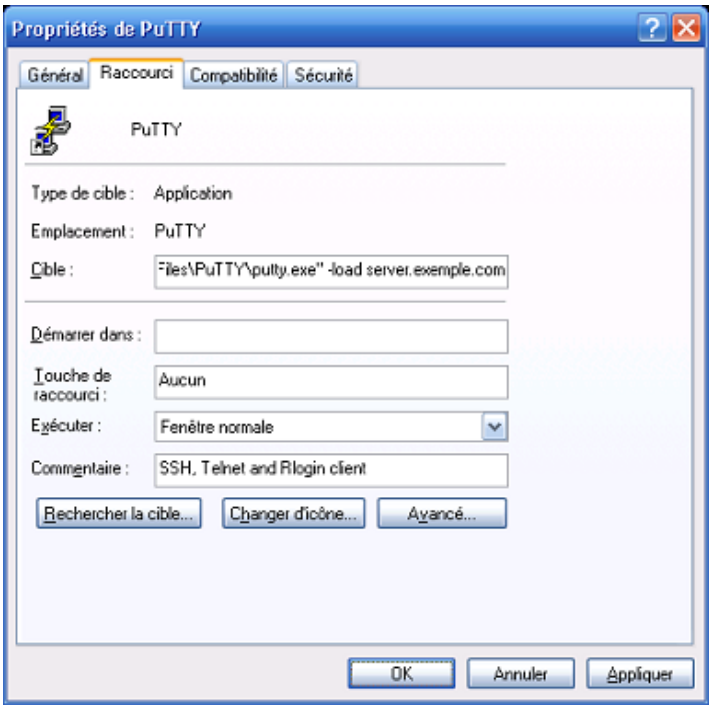
On revient ensuite dans la fenêtre principale et on sauvegarde la session avec le bouton **Save**. On peut enfin se connecter sans utiliser de mots de passe. Il ne faut pas oublier de mettre le nom du user lorsque on renseigne l'adresse IP ou le nom du serveur :



Le paramètrage est mainenant terminé, il suffit de faire un double-clic sur le profile pour pouvoir se connecter.

On maintenant utiliser PuTTY pour se connecter rapidement sur un serveur distant. Si on veut se connecter encore plus rapidement, il suffit de créer un raccourci vers l'icone de l'éxécutable et de modifier la ligne "**cible**" du raccourci en y incluant `-load nom_du_profile`. Par exemple si on a paramétré un profile qui s'appelle **serveur.exemple.com**, il suffit de créer un raccourci sur le bureau windows en mettant ceci sur la ligne "**cible**" :

```
"C:\Program Files\PuTTY\putty.exe" -load serveur.exemple.com
```



Ensuite il n'y a plus qu'à faire un double-clic sur ce raccourci pour se retrouver connecté à la machine **serveur.exemple.com**.

PuTTYtray : une version améliorée de PuTTY

Comme le code de PuTTY est librement accessible, on peut trouver plusieurs variantes de ce logiciel sur Internet, dont PuTTYtray (http://haanstra.eu/putty/) qui possède des améliorations intéressantes :

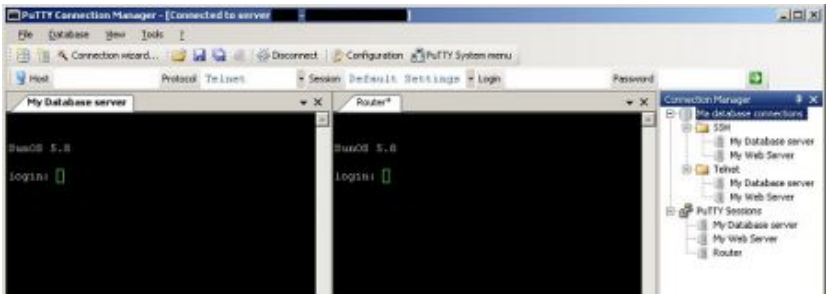
- Possibilité de se mettre dans le systray quand on réduit la fenêtre.
- Peut reconnecter une session quand un ordinateur sort de veille.
- Permet de rendre les hyperliens cliquables.
- ...

Pour bénéficier de toutes ces améliorations, il suffit de télécharger le binaire putty.exe et de remplacer l'original dans le répertoire d'installation. On peut différencier les 2 binaires putty.exe par leur icone qui est différente.

Dans le même style il y a également Kitty (http://www.9bis.net/kitty/) qui semble intéressant mais que je n'ai pas testé. Kitty permet d'intégrer des fonctions de transfert de fichier grâce à pcsp.exe.

Putty Connexion Manager

Putty Connexion Manager (http://puttycm.free.fr) est un logiciel gratuit qui permet de gérer les connexions PuTTY et d'ouvrir plusieurs connexions sous forme d'onglets dans le même fenêtre :



Cela peut s'avérer très pratique à l'utilisation.

Création de tunnels sécurisés

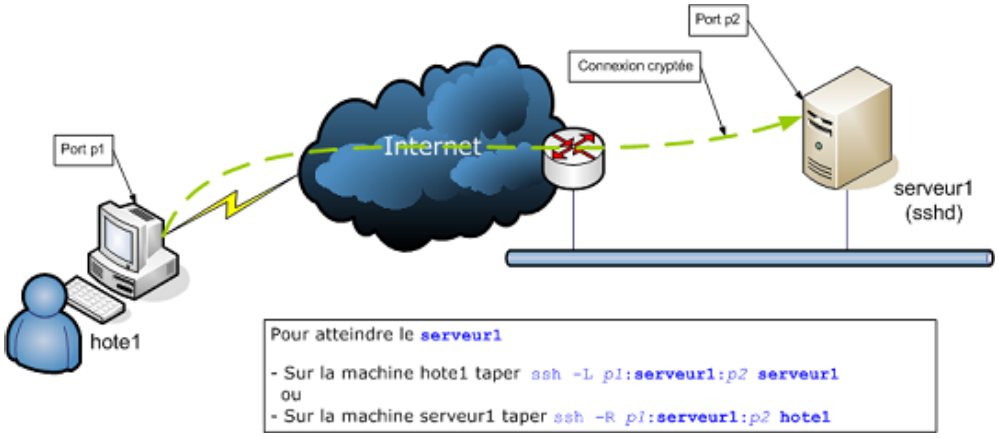
De nos jours internet est tellement peu sûr qu'il faut trouver des moyens pour sécuriser les communications, surtout si des informations doivent circuler en clair. Pour sécuriser ces communications, une technique simple est d'utiliser un serveur ssh et de créer des tunnels ssh entre le client et les differents services et/ou serveurs distants.

Il y a deux méthodes pour créer un tunnel ssh en ligne de commande. La première, qui est aussi la plus utiliée consiste à créer le tunnel à partir du client, tandis que dans la deuxième permet de créer le tunnel à partir du serveur. Nous ne nous attarderons pas sur la deuxième car elle trés peu utilisée.

Passons maintenant à la pratique.

Tunnel ssh simple

Commençons par un exemple simple. Dans ce cas, on veut simplement sécuriser les échanges de données entre un client et un serveur http sur lequel tourne également un serveur ssh. Bien sûr on pourrait utiliser https à la place, mais ce n'est pas le but. Dans cet exemple on utilise seulement 2 machines, le client et le serveur :



Pour créer le tunnel ssh à partir de la machine cliente (Windows, Linux ou MacOSX) ouvrir un shell et entrer la commande suivante :

```
ssh -L  200:serveur1:80 serveur1
```

ou

```
ssh -L 200:serveur1:80 user@serveur1
```

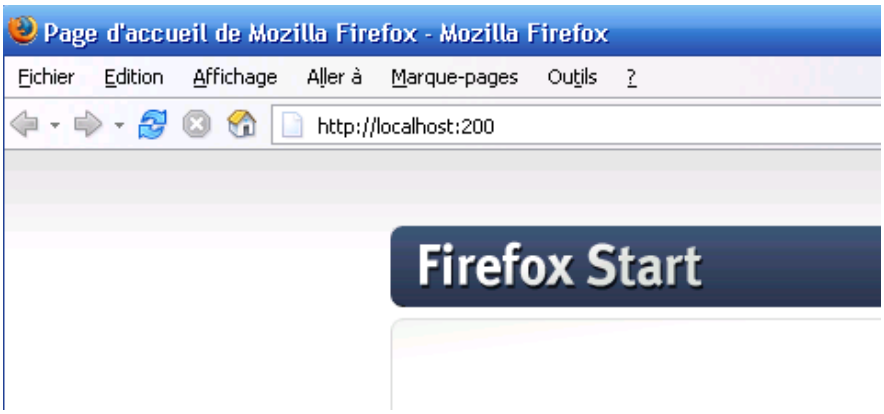
ou encore

```
ssh -l user -L 200:serveur1:80 serveur1
```

Explication de la syntaxe : `ssh -L portlocal:serveur1:portdistant serveur1`

`-L` signifie que c'est une redirection de port locale, c'est à dire initiée par le client
`portlocal` est le n° de port tcp qu'il faudra utiliser pour se connecter sur localhost. Ce n° de port est arbitraire, mais il faudra néanmoins être root sur la machine client s'il est inferieur à 1024.
`portdistant` est le port utilisé par le service du serveur distant, donc 80 pour un serveur http.
`serveur1` est le nom de la machine sur lequel tournent les serveurs http et ssh. A la place des noms des serveurs, on peut aussi utiliser les adresses IP.
`-l user` de la 3° commande est l'équivalent de **user@** de la 2° commande. Dans la première forme de syntaxe, on ne spécifie pas le nom de **user** car on part du principe que c'est le même que sur le client.

Une fois le tunnel créé, il suffira de taper `http://localhost:200` dans la barre d'adresse du navigateur pour être en fait connecté au serveur distant. On obtiendra alors les mêmes informations que si on avait taper `http://serveur1`, mais les échanges de données ne pourront pas être écoutées.



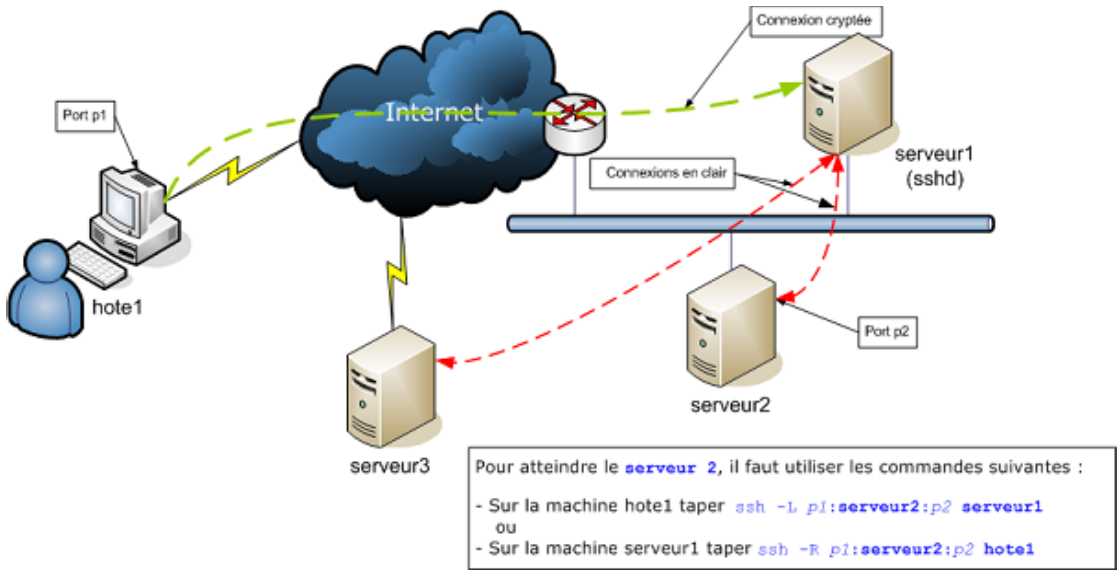
Tunnel ssh en utilisant une passerelle pour atteindre une autre machine

Dans ce cas de figure, on utilise un serveur ssh comme passerelle pour joindre une autre machine. Les échanges cryptés ne se font alors que entre le client (**hote1**) et la passerelle (**serveur1**), et c'est cette dernière qui relaye les informations en clair à la machine que l'on souhaite atteindre. Cette topologie peut être utilisée pour résoudre les problèmes suivants :

- cas n° 1: on souhaite atteindre une machine (**le serveur2**) dans un réseau privé, mais cette machine n'a pas de serveur ssh. Les informations qui transitent sur internet doivent être cryptées, mais pas les informations qui circulent sur le réseau local.
Exemple : on est connecté à un hotspot wifi public et on veut utiliser le serveur intranet de son entreprise.

- cas n° 2 : on est sur un réseau local et on désire se connecter à un serveur de news (nntp) qui est le serveur 3. Malheureusement le firewall du réseau local interdit ce genre de connexions. Par contre, on a la possibilité de se connecter sur un serveur ssh à l'extérieur et de s'en servir de passerelle pour atteindre ce serveur de news. Bien sûr le firewall du serveur doit laisser passer les connexions sortantes sur le port tcp 22 pour que ca marche.

Notons que dans ces 2 cas de figures, le serveur ssh est lui aussi derrière un routeur et a une adresse IP privée. Le routeur distant devra donc faire du Port Forwarding, c'est à dire qu'il faut le configurer pour que lorsqu'il recoive une demande de connexion sur le port TCP 22 (ssh) de son adresse IP publique, il forwarde la requete vers le port TCP 22 de l'adresse IP privée du serveur ssh.



Que l'on souhaite joindre le **serveur2** ou le **serveur3**, la commande sera la même. A partir de la machine cliente (Windows, Linux ou MacOSX) ouvrir un shell et entrer la commande suivante :

```
ssh -L  200:serveur2:119 serveur1
```

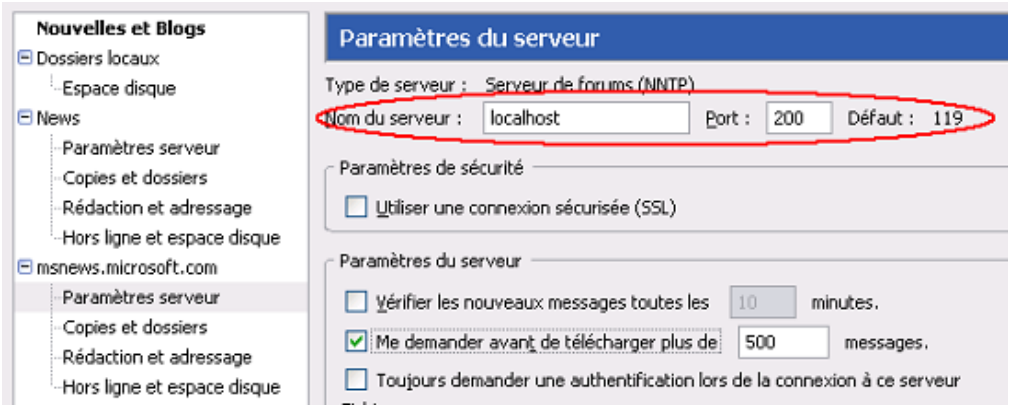
ou

```
ssh -L 200:serveur2:119 user@serveur1
```

ou encore

```
ssh -l user -L 200:serveur2:119 serveur1
```

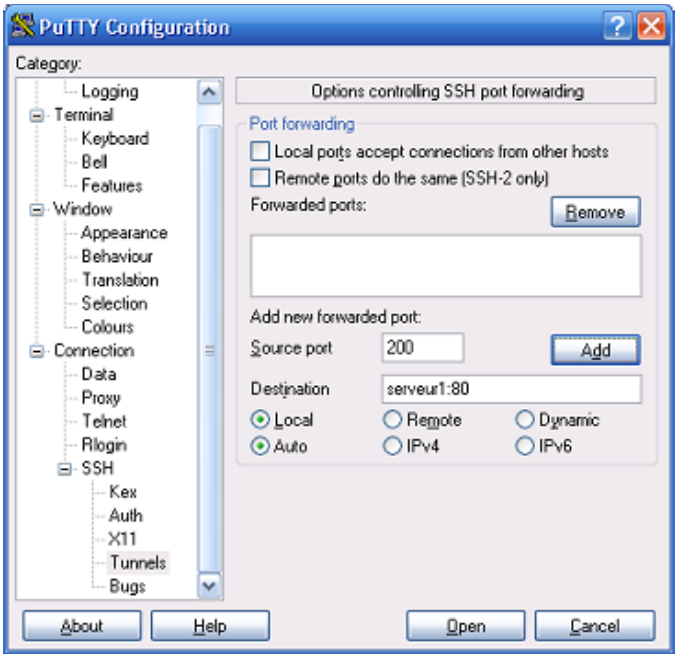
On voit que la syntaxe est quasiment la même que dans le cas précédant. On remplace juste le serveur de destination par **serveur2** au lieu de **serveur1**. Une fois le tunnel créé, il suffira de parametrer le client nntp pour qu'il se connecte sur le port 200 de **localhost**. Voiçi un exemple de configuration avec thunderbird :



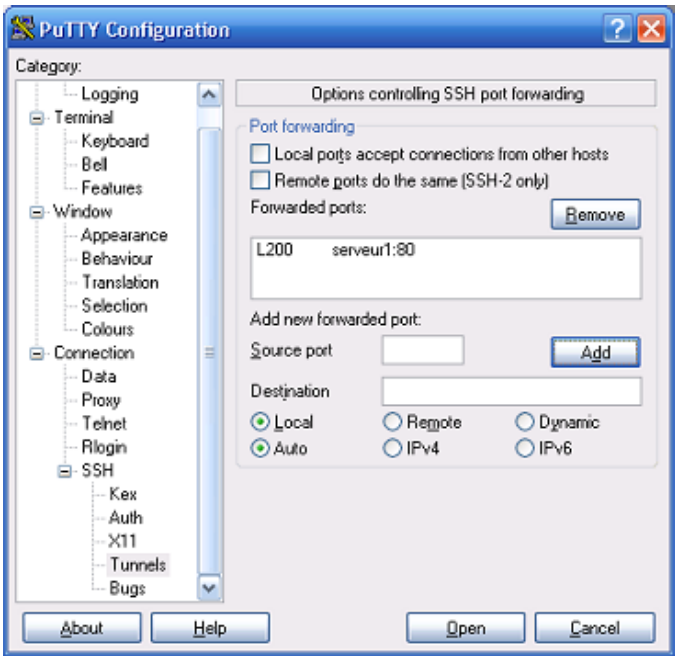
Tunnels SSH avec PuTTY

PuTTY permet également de créer des tunnels ssh. En fait il fait la même chose que les commandes précédentes, mais il suffit de cliquer sur un bouton pour monter le tunnel.

Pour monter un tunnel simple (cf notre premier cas), il faut configurer une session comme on vient de le voir précédemment. Ensuite, dans le menu de gauche, dérouller l'arborescence des menus PuTTY pour aller **Connections / SSH / Tunnels**. On renseigne alors les champs **Source Port** et **Destination** de la façon suivante :



Ensuite on appuie sur le bouton **Add** et on a l'écran suivant :



On revient dans **Session** sur la fenêtre de gauche et on appuie sur le bouton **Save** de la fenêtre de droite. On peut maintenant se connecter comme si on avant entré la commande :

```
ssh -l root -L 200:serveur1:80 serveur1.
```

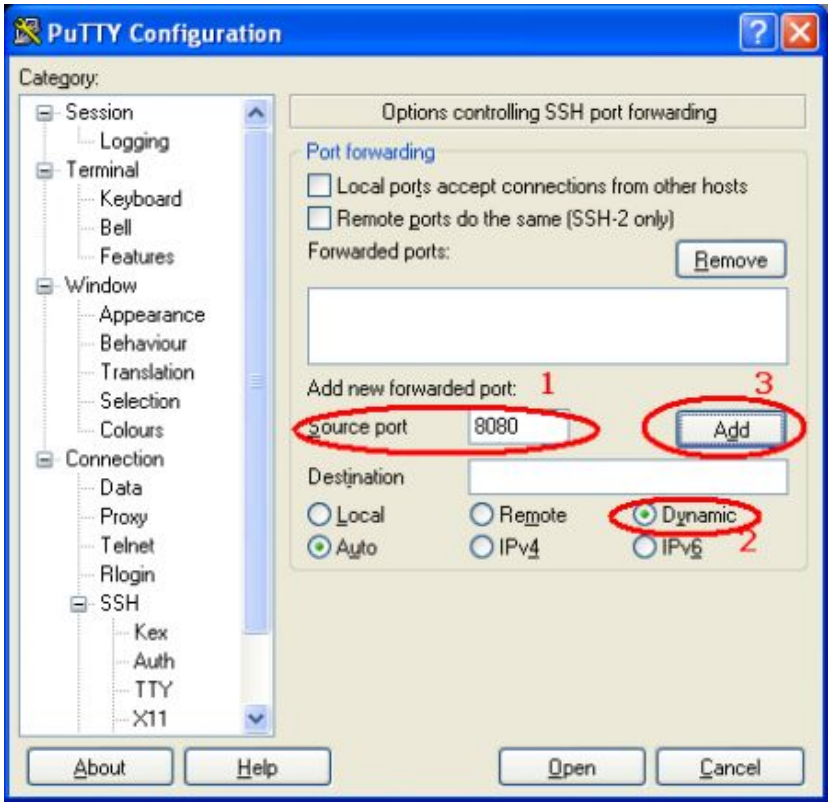
Si on veut utiliser PuTTY pour créer un tunnel pour se connecter sur une machine tiers (cf notre cas n° 2), on procéde exactement de la même façon mais à la place de `serveur1:80` on écrit `serveur2:80` ou `serveur3:80` dans le champ **Destination** du formulaire de PuTTY. Par contre, on ne change rien dans le champ **Host name or IP address** du menu **Session**. On a alors l'équivalent de la commande :

```
ssh -l root -L 200:serveur2:80 serveur1
```

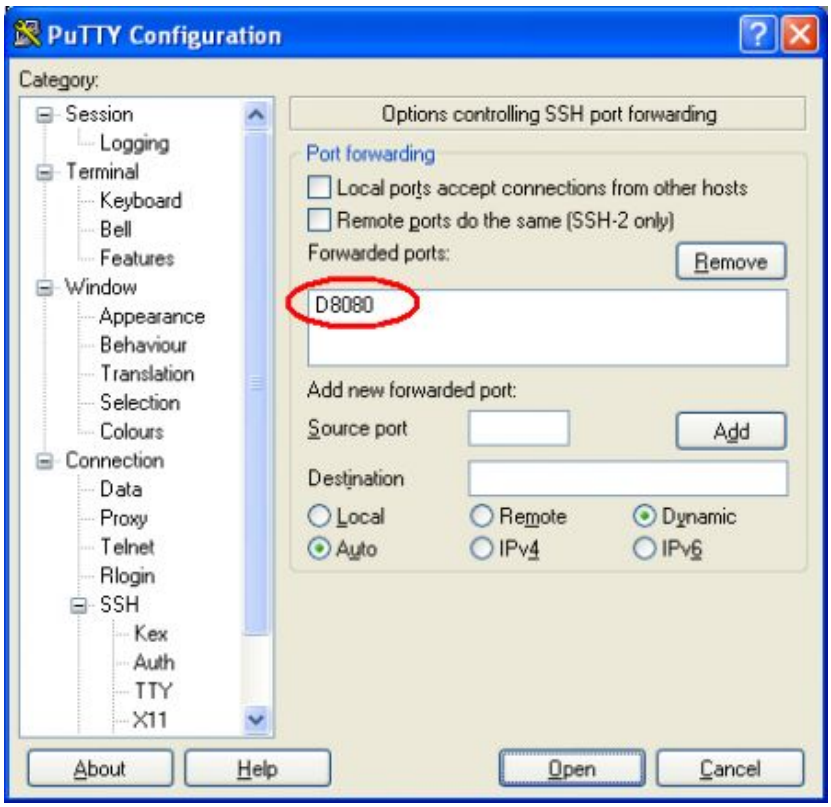
Proxy Socks5 avec Putty

Putty peut être utilisé comme client **Proxy Sock 5**. La configuration de cette fonctionnalité est identique à celle d'un tunnel avec les paramètres suivants (par exemple) :

Source Port : Entrer un port local qui peut être utilisé sur le PC (par exemple **8080**)
Destination : Laisser le champ vide et sélectionner l'option **Dynamic** à la place de **Local**



Cliquer ensuite sur le bouton **Add** et vérifier qu'on obtienne bien l'écran suivant :



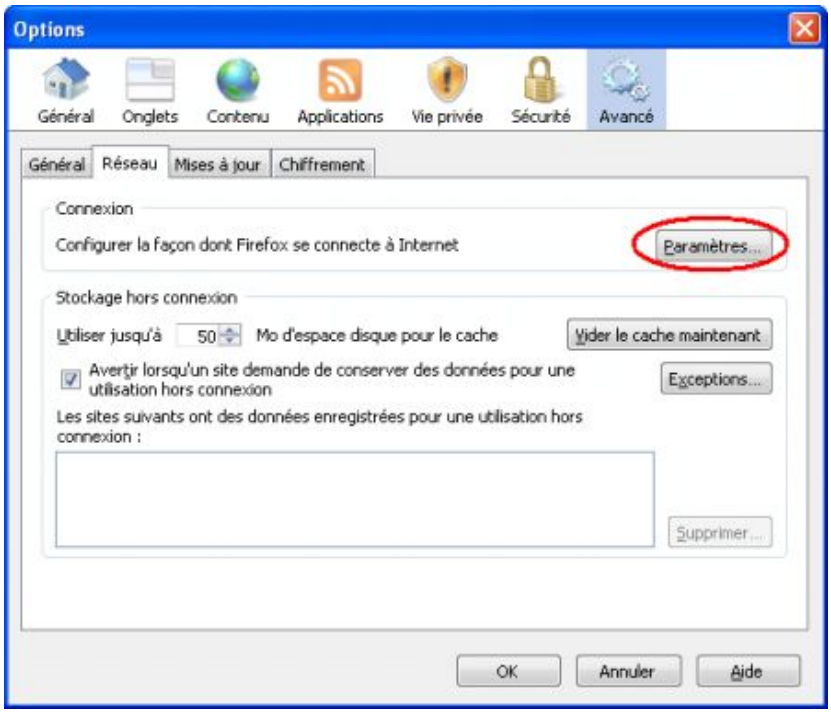
Sauvegarder la session Putty et lancer la connexion.

Exemple d'utilisation : surfer de façon sécurisée avec Firefox

Nous allons paramètrer firefox pour que le traffic qu'il engendre soit chiffré et donc qu'il ne puisse pas être intercepté par un sniffer réseau ou un firewall.
Cette configuration se fait en 2 étapes : le traffic http et les requêtes DNS.

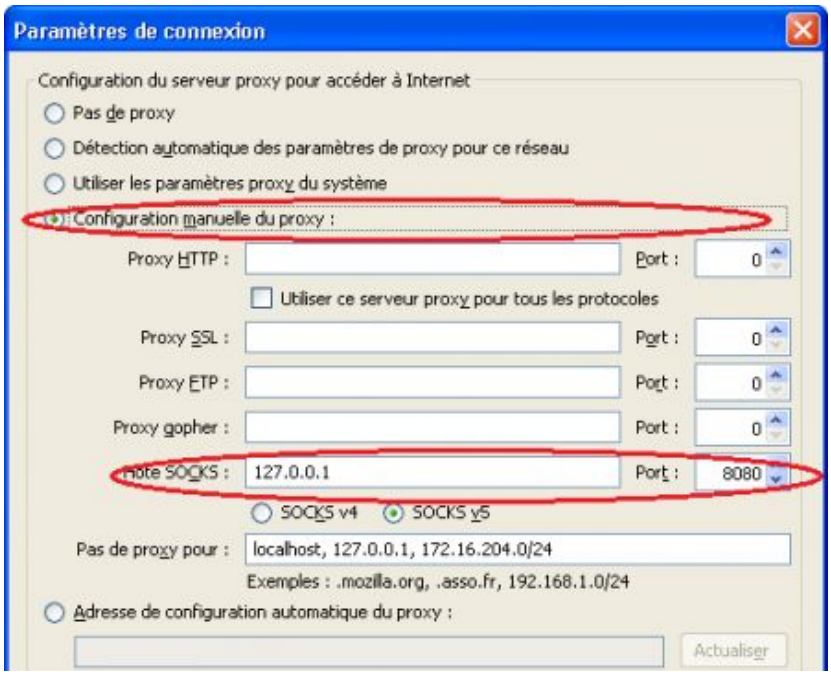
Configuration du traffic http

Aller dans le menu **Outils** / **Options**, puis onglet **Réseau** et enfin bouton **Paramètres** :



Renseigner les champs suivants :

- **Configuration manuelle du Proxy**
- **Hôte SOCKS** : **127.0.0.1** et **8080** pour être en phase avec la configuration de notre exemple ci-dessus
- **SOCKS V5**

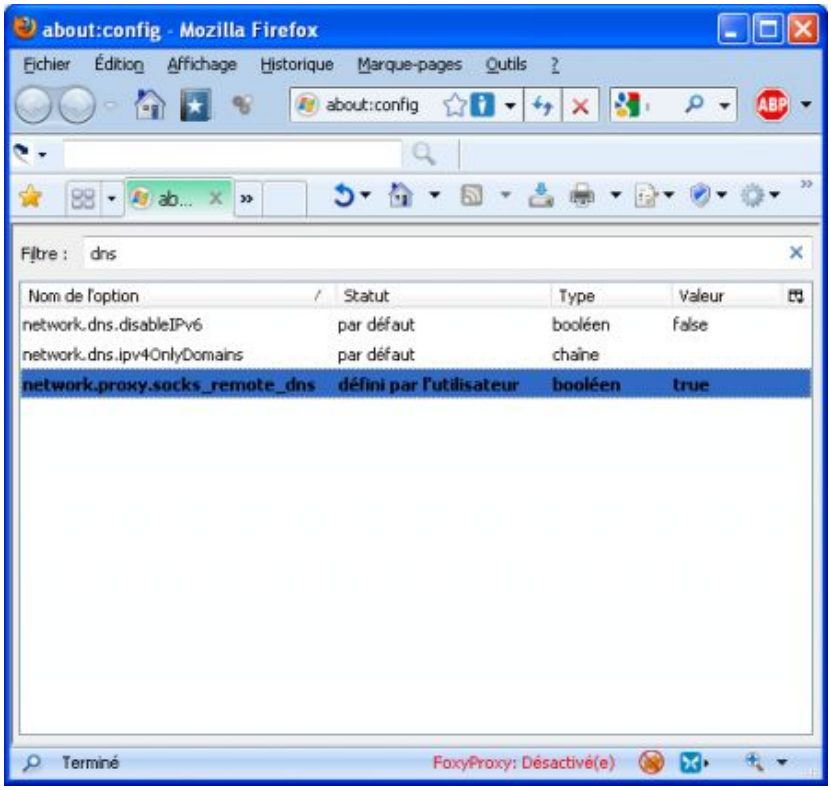


Puis valider en cliquant sur **OK** et fermer la fenêtre des options.

Configuration des requetes DNS

Si on ne passe pas par cette deuxième étape, le traffic http sera crypté mais les requêtes DNS continueront de se faire en clair par l'intermédiaire du serveur DNS qui se trouve sur le réseau local.

Aller dans la barre d'adresse de firefox et taper `about:config`
Taper `dns` dans la zone de filtre
Localiser la chaine **network.proxy.socks_remote_dns** et la modifier pour qu'elle soit à **True**. Si elle n'existe pas, il suffit de la créer.



Redémarrer Firefox. Celui-ci est maintant prêt à être utilisé pour surfer en utilisant le serveur ssh distant comme proxy. Toutes les communications entre firefox et le serveur ssh seront cryptées.

Gestion des tunnels ssh sous Windows

Parfois, on veut juste créer un tunnel ssh mais on ne veut pas ouvrir de terminal.

Il existe plusieurs logiciels qui permettent de faire ca. Le logiciel reste ouvert en tache de fond et se met dans le Systray à coté de l'horloge. Parmis ces logiciels, on peut citer :

- XShell (http://www.netsarang.com/products/xsh%5Ffeatures.html) (payant)
- PuTTYtray (http://haanstra.eu/putty/) (gratuit)
- Tunnelier (http://www.bitvise.com/tunnelier) (gratuit pour une utilisation non commerciale)
- MyEnTunnel (http://nemesis2.qx.net/pages/MyEnTunnel) (gratuit)
- PuTTY Tunnel Manager (http://code.google.com/p/putty-tunnel-manager/) (gratuit)

Mon choix se porte sur le dernier, PuTTY Tunnel Manager car il s'agit d'un simple exécutable. Il suffit de le télécharger dans le même répertoire que PuTTY et de l'éxécuter.

Quand on le lance, il propose les profils déjà configurés avec PuTTY, puis il se met dans la barre du Systray et tourne en tache de fond.

Résumé des commandes de base

Connexion simple à un serveur :
`ssh user@serveurdistant` ou `ssh -l user serveurdistant`

Connexion en utilisant un autre port que celui par défaut
`ssh -p 2222 user@serveurdistant` ou `ssh -p 2222 -l user serveurdistant`

Connexion à un serveur et exécution de commandes sur celui-ci
`ssh -l user serveurdistant "uptime"`

Transfert de fichier en utilisant la copie sécurisée scp ou sftp
```
scp /tmp/fichier user@serveurdistant:/tmp
scp user@serveurdistant:/tmp/fichier /tmp
sftp -o Port=22 user@serveurdistant:/tmp/fichier /tmp/
```

Création de tunnels ssh
`ssh -l user -L portlocal:serveurcible:portcible serveurssh`

Rémi TAUPIN, 2011 - Propulsé par Gandi Hébergement avec DotClear 2 - Informations légales