

Ligne de commande

La méthode prédominante et celle utilisées à fin d'automatisation est évidemment l'outil SSH à la ligne de commande. Il est aussi possible à l'aide de bibliothèques telles que netmiko ou paramiko d'interagir avec un serveur SSH en Python pour des besoins spécifiques mais généralement, l'ensemble des manipulations qui pourraient être nécessaires peuvent être effectuées à partir d'un shell.

Installation

Linux

Dépendamment de la distribution de Linux que vous utiliserez, le gestionnaire de packages aura une structure différente mais le nom du package en question est généralement "ssh".

Voici quelques exemples.

```
# Debian/Ubuntu
sudo apt update && sudo apt install ssh

# RHEL/Fedora
sudo dnf install ssh

# ArchLinux
sudo pacman -S ssh
```

Un autre outil essentiel à l'automatisation de tâches par SSH est la commande "expect". Elle permet d'automatiser l'exécution de commandes complexes à travers une connexion ssh.

```
# Debian/Ubuntu
sudo apt update && sudo apt install expect

# RHEL/Fedora
sudo dnf install -y tcl tk expect

# ArchLinux
sudo pacman -S expect
```

Windows

Le client SSH sous Windows est une fonctionnalité facultative pouvant être installée en PowerShell.

```
# Exécuter en tant qu'administrateur
Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
```

Utilisation

Connexion simple par utilisateur et mot de passe

Si le serveur SSH le permet, la commande suivante est généralement suffisante pour établir une connexion au shell du serveur distant. Le domaine est optionnel, il sera généralement utilisé pour se connecter à un serveur Windows si celui-ci fait partie d'un domaine Active Directory.

```
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur distant]
```

Connexion par certificat

Si le serveur SSH le permet, les commandes suivantes permettent de créer une clé SSH publique permettant l'authentification sans mot de passe. Ceci est généralement essentiel lors d'automatisation de tâches. Il est à noter que ces commandes doivent être exécutées par l'utilisateur qui s'authentifiera au serveur. Celle-ci n'est pas pour l'ensemble du système.

```
# Génération du certificat, le certificat peut être placé ailleurs mais son emplacement par
défaut est généralement dans le dossier .ssh de l'utilisateur et porte généralement le nom
id_rsa. Ce chemin est ici mentionné sous la forme pour Linux.
ssh-keygen -f ~/.ssh/id_rsa -N [mot de passe, au besoin, sera demandé au moment de la
connexion] -t rsa -q

# Copie de la clé publique sur un serveur SSH auquel on pourra s'authentifier par certificat
ssh-copy-id -i [chemin du fichier] [domaine\utilisateur]@[adresse du serveur] -p [port du
serveur SSH]

# À ce moment, la connexion SSH ne devrait plus demander de mot de passe
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur SSH]
```

Exécution d'une commande individuelle sans réponse

Il est possible d'exécuter une seule commande de façon non interactive avec l'argument "-c"

```
# La commande suivante permettrait d'automatiser la mise à jour d'un système Debian/Ubuntu si
sudo est configuré pour laisser l'utilisateur exécuter des commandes avec sudo sans demander
de mot de passe
```

```
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur] -c "sudo apt update && apt upgrade"
```

```
# La commande suivante permettrait de redémarrer le serveur distant
```

```
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur] -c "sudo reboot"
```

Partage de ressources réseau/établissement de tunnels

Si le paramètre "AllowTcpForwarding" est défini à "yes" dans la configuration du serveur, il est possible d'établir des tunnels pour mettre à disposition du client ou du serveur des ressources réseau se situant d'un côté ou de l'autre. [Un autre livre](#) décrit ce concept plus en profondeur.

```
# La commande suivante mettrait à disposition du client un serveur MySQL exécuté sur le serveur SSH comme s'il était exécuté sur le client. Ceci est le "remote forwarding" où on prend un port du serveur SSH et on le met à la disposition du client.
```

```
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur] -R 3306:localhost:3306
```

```
# La commande suivante mettrait à disposition du client un serveur web auquel le serveur SSH aurait accès comme si le serveur était exécuté sur le client. Ceci est aussi une forme de "remote forwarding"
```

```
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur] -R 443:[adresse du serveur web, ex. www.google.com]:443
```

```
# La commande suivante mettrait à disposition du serveur un serveur RDP (bureau à distance) exécuté localement sur le client. Ceci est une forme de "local forwarding" et est l'inverse du remote
```

```
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur] -L 3389:localhost:3389
```

```
# La commande suivante mettrait à disposition du serveur un serveur SSH auquel le client a accès comme par exemple le routeur du client. Ceci est une forme de "local forwarding" et est l'inverse du remote. Il est à noter ici que le premier port mentionné est le port qui sera utilisé sur le serveur distant pour se connecter et doit donc différer de son propre port SSH. Ce genre de connexion permettrait une porte d'entrée dans un réseau du client au serveur distant. Ceci est nommé un "backdoor".
```

```
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur] -L 8022:[adresse du routeur]:22
```

```
# La commande suivante établirait une redirection de ports dynamiques qui pourrait être utilisé à titre de proxy SOCKS. Ceci porte aussi le nom d'un "poor man's VPN". On pourrait donc ici configurer un proxy pointant vers localhost au port 8443 et le trafic traverserait la
```

```
connexion SSH et sortirait du serveur SSH.
```

```
ssh [domaine\utilisateur]@[adresse du serveur] -p [port du serveur] -D localhost:8443
```

Transfert de fichiers

SSH permet généralement le transfert de fichiers à l'aide du protocole "SCP". Les fichiers peuvent être récupérés ainsi que déposés sur le serveur. Il est à noter que le compte du serveur auquel on se connecte doit posséder les autorisations nécessaires pour effectuer l'opération demandée.

```
# La structure de SCP est la suivante. Il est à noter qu'avec cette structure, il est possible de transférer un fichier d'un serveur SSH à un autre. Si aucun utilisateur et adresse sont mentionnés d'un côté ou de l'autre et que seulement le chemin complet du fichier (source ou destination) est mentionné, il s'agit alors d'un fichier situé sur le stockage du client.
```

```
scp [domaine\utilisateur]@[adresse de la source]:[chemin complet du fichier source]
[domaine\utilisateur]@[adresse de la destination]:[chemin complet du fichier source]
```

```
# La commande suivante permettrait de récupérer une copie du journal système d'un serveur distant
```

```
scp root@[adresse du serveur]:/var/log/syslog [chemin de la destination]
```

```
# La commande suivante permettrait de déposer un fichier de configuration bash dans le dossier de l'utilisateur distant
```

```
scp [chemin du fichier de configuration] [utilisateur]@[serveur]:/home/[utilisateur]/.bashrc
```

Exécution interactive de commandes avec "expect"

La commande "expect" comme son nom l'indique fonctionne en réagissant au résultat des commandes exécutées. Il utilise des expressions régulières pour valider qu'il est le bon moment pour exécuter certaines commandes. Il peut être utilisé tant à des fins d'exécutions locales qu'à l'exécution de commandes distantes à travers une connexion SSH.

Le script suivant permettrait de s'authentifier en SSH à l'aide d'un mot de passe automatiquement et d'élever l'invite de commande à l'utilisateur "root".

```
#!/usr/bin/expect
```

```
# Utilisation ssh_switchuser.expect <host> <ssh user> <ssh password> <su password>
```

```
set timeout 60
```

```
# Établissement de la connexion SSH
```

```
spawn ssh [lindex $argv 1]@[lindex $argv 0]
```

```
# S'il s'agit de la première connexion, accepter l'identité du serveur SSH
expect "yes/no" {
    \send "yes\r"
    # Si le serveur demande un mot de passe, mentionner le mot de passe
    \expect ".*?assword" { send "[lindex $argv 2]\r" }
    \} ".*?assword" { send "[lindex $argv 2]\r" }

# Exécution de la commande d'élévation à root
expect "# " { send "su - \r" }
# Si un mot de passe est demandé, mentionner le mot de passe
expect ": " { send "[lindex $argv 4]\r" }
expect "# " { send "ls -ltr\r" }

# Retourner le contrôle à l'utilisateur dans l'état actuel de l'invite de commande, ici dans
une connexion SSH établie
interact
```

Ce script aurait pu continuer pour effectuer d'autres actions automatiquement et fermer la connexion SSH suite à ces actions ainsi que d'écrire le résultat des commandes dans un fichier.

Revision #10

Created 2025-01-18 00:35:11 UTC by Alexandre Arsenault-Jetté

Updated 2025-07-10 01:16:52 UTC by Alexandre Arsenault-Jetté