MANUAL - Secure SHell

Note : SSH est la porte d'entrée de votre serveur. Si vous la laissez entrouverte, ne venez pas pleurer quand votre serveur minera du Bitcoin pour un hacker nord-coréen.

Création des clés (Keygen)

Avertissement : En situation de crise, la précision l'emporte sur la vitesse. Une commande tapée trop vite est souvent la dernière commande que vous tapez sur ce serveur.

Oubliez les mots de passe. Nous utilisons la cryptographie asymétrique.

```
ssh-keygen -t rsa -b 4096 -a 100 -f ~/.ssh/my-key -N "Ma Super Passphrase Indevinable"
```

-t **Type** Algorithme de chiffrement. RSA est le standard, mais **Ed25519** est supérieur (plus rapide, plus sûr). Si vous avez le choix, prenez Ed25519.

-b **Bits** Pour RSA, 2048 est "moyen". **4096** est le standard pro. Pour Ed25519, la taille est fixe (pas de -b).

-f **File** Ne tout mettez pas dans id_rsa. Organisez-vous ! Une clé par client/projet.

-a **Round** Elle définit le nombre de "tours" (rounds) de la fonction de dérivation de clé (KDF). En gros, elle rend le déchiffrement de ta clé privée beaucoup plus lent et coûteux en calcul. Si un attaquant vole ta clé privée chiffrée, il mettra des siècles à "brute-forcer" ta passphrase si le -a est élevé (ex: 100 ou plus).

-C **Comment** L'étiquette, elle est écrit à la fin de la clé publique (sur le serveur)

-N New Passphrase (**ATTENTION**, avec cette option, la passphrase est écrite en clair dans le terminal est il est donc possible de la retrouver avec la commande `history` !!!). À utiliser uniquement dans des scripts automatisés ou si vous savez effacer vos traces.

Elle génère une paire de clés :

id-rsa, qui contient la clé privée de la paire. Ce fichier ne doit jamais être partagé ou copié sur un système distant, sauf s'il s'agit d'un système contrôlé par le détenteur de la paire de clés.

id-rsa.pub, qui contient la clé publique de la paire. Ce fichier peut être partagé et copié sur d'autres systèmes, en particulier lorsque l'utilisateur prévoit de se connecter à ces systèmes à l'aide de SSH.

Bien sur vous pouvez aussi crée vos clé simplement avec cette commande (mais ou est le FUN!)

```
user@laptop:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Created directory '/home/user/.ssh'.
Enter passphrase for "/home/user/.ssh/id_rsa" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa
Your public key has been saved in /home/user/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:4sgeNZsngksY0fsULrpdoU4Bdj92MiXla38SNgc6xgs user@laptop
The key's randomart image is:
+---[RSA 3072]----+
|                 |
|     .           |
|    + + +        |
|   . = O o       |
|    E / S o      |
|   o X ^ X       |
|  o B O * .      |
| . = * + o       |
|  . o o          |
+----[SHA256]-----+
```

OK décomposons cela !

```
user@laptop:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Created directory '/home/user/.ssh'.
Enter passphrase for "/home/user/.ssh/id_rsa" (empty for no passphrase):
Enter same passphrase again:
```

Lors de la création, ssh-keygen vous demandera : *"Enter passphrase"*.

- **La Règle :** Ne laissez **jamais** ce champ vide pour une clé utilisateur. Une clé privée sans mot de passe est une clé "en clair". Si un attaquant accède à votre fichier (virus, vol de PC), il accède instantanément à tous vos serveurs sans aucun obstacle. La passphrase chiffre le fichier sur votre disque : même volée, la clé reste inutilisable sans ce code.
- **L'Exception :** Appuyer sur *Entrée* (vide) est toléré **uniquement** pour des clés dédiées à l'automatisation (scripts, backups) stockées sur des serveurs sécurisés, car un script ne peut pas taper de mot de passe. Dans ce cas, la sécurité repose entièrement sur la protection du fichier lui-même.

La Clé Privée (Identity File)

```
Your identification has been saved in /home/user/.ssh/id_rsa
```

Aussi appelée "fichier d'identité", c'est la partie secrète de votre bi-clé.

- **Emplacement standard :**
  - **Linux / macOS :** ~/.ssh/ (ex: /home/user/.ssh/)
  - **Windows :** %USERPROFILE%\.ssh\ (ex: C:\Users\Moi\.ssh\)
- **Nommage :** Par défaut, elle se nomme id_rsa ou id_ed25519. Cependant, pour une gestion multi-serveurs propre, elle portera le nom que vous avez défini avec l'option -f (ex: id_ed25519_pro).
- **Sécurité Critique :** Ce fichier ne doit **jamais** quitter votre machine sécurisée.

La Clé Publique (Extension .pub)

```
Your public key has been saved in /home/user/.ssh/id_rsa.pub
```

C'est la "serrure" de votre système. Contrairement à la clé privée, celle-ci est faite pour être vue et partagée.

- **Nommage et Emplacement :** Elle est générée automatiquement en binôme avec la clé privée. Elle porte strictement le même nom, suivi de l'extension .pub.
  - Exemple : Si votre clé privée est id_ed25519_pro, votre clé publique sera id_ed25519_pro.pub.
  - Elle se situe toujours dans le même répertoire (~/.ssh/).
- **Usage :** C'est **l'unique** fichier que vous devez envoyer sur les serveurs distants (dans le fichier authorized_keys).
- **Règle d'or :** Ne vous trompez jamais de fichier. .pub = Public (Serveur). Sans extension = Privé (Gardez-le !).

L'Empreinte (Fingerprint)

```
The key fingerprint is:
SHA256:4sgeNZsngksY0fsULrpdoU4Bdj92MiXla38SNgc6xgs user@laptop
```

Votre clé publique est un bloc de texte cryptographique trop long pour être lu par un humain. L'empreinte est son **identifiant unique** (son résumé).

- **Fonctionnement :** C'est une séquence courte générée par un hachage (le plus souvent **SHA-256**). Si un seul bit de votre clé change, l'empreinte change totalement.
- **Utilité Critique :** C'est le seul moyen pour un humain de vérifier l'identité d'une clé.
    - Quand vous donnez votre clé à un admin, il vérifie l'empreinte.
    - Quand vous vous connectez à un serveur inconnu, SSH vous montre son empreinte. Si elle ne correspond pas à ce qu'on vous a donné : **C'est une attaque (Man-in-the-Middle). Coupez tout.**
- **Format :** Elle ressemble à ceci : SHA256:4sgeNZsngksY0fs....

Le Randomart (Validation Visuelle)

```
The key's randomart image is:
+---[RSA 3072]----+
|                 |
|     .           |
|    + + +        |
|   . = O o       |
|    E / S o      |
|   o X ^ X       |
|  o B O * .      |
| . = * + o       |
|  . o o          |
+----[SHA256]-----+
```

C'est ce petit carré d'art ASCII qui apparaît lors de la génération. Ne l'ignorez pas.

- **Le constat :** Le cerveau humain est médiocre pour comparer des chaînes de caractères complexes (SHA256:4sgeNZ...), mais il excelle à reconnaître des formes géométriques.
- **Fonctionnement :** Les bits de votre signature numérique servent de "coordonnées" pour dessiner ce motif unique.
- **Utilité :** C'est un moyen rapide de validation. Si vous connaissez la "tête" de votre clé serveur (sa forme globale), vous repérerez une usurpation d'identité (Man-in-the-Middle) au premier coup d'œil, sans avoir à vérifier chaque caractère du hachage.

Copie de la clé publique

Le Déploiement Automatisé (ssh-copy-id)

Cette commande est le meilleur ami de l'admin. Elle ne se contente pas de "copier" un fichier, elle installe votre identité proprement.

- ***Mécanisme :** Elle prend votre clé publique locale (.pub) et va l'ajouter (append) à la fin du fichier ~/.ssh/authorized_keys sur le serveur distant.*
- ***Pourquoi l'utiliser ?** Contrairement à une copie manuelle hasardeuse, elle gère automatiquement les permissions critiques (création du dossier .ssh en 700 et du fichier en 600).*
- ***L'Adieu au Mot de Passe :** Lors de l'exécution, le serveur vous demandera votre mot de passe utilisateur une **dernière fois**. Savourez cet instant, car si vous faites bien votre travail, vous ne le taperez plus jamais pour vous connecter.*
- ***La Vérification (Trust but Verify) :** Une fois la copie terminée, la commande vous invite à vérifier. Ne soyez pas naïf, connectez-vous immédiatement pour confirmer que la clé est bien acceptée avant de fermer votre session actuelle.*

```
user@laptop:~$ ssh-copy-id -i .ssh/id_rsa.pub main@server
The authenticity of host 'server (192.168.200.252)' can't be established.
RSA key fingerprint is 4c:87:08:f7:34:31:b6:2d:66:4d:19:bd:06:b7:6d:77.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'server,192.168.200.252' (RSA) to the list of known hosts.
main@server's password:
Now try logging into the machine, with "ssh 'main@server'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

user@laptop:~$
```

-i	**Identity file**	Force la commande à utiliser **une clé publique spécifique**.

Pourquoi c'est obligatoire ? Par défaut, ssh-copy-id scanne votre dossier .ssh à la recherche des fichiers standards (id_rsa.pub, etc.).

- Si vous avez suivi les bonnes pratiques et nommé votre clé id_ed25519_pro, l'outil l'ignorera royalement sans ce flag.
- Pire, si vous avez plusieurs clés, ne pas mettre -i risque d'installer toutes vos clés publiques sur le serveur. C'est sale.

L'Authentification Réussie

```
user@laptop:~$ ssh main@server
Last login: Mon Dec 23 01:42:26 2013
main@server:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA57Ak4JWW7zKbip1WPWxr4eQnaWWR0I/ ... /p9zIOqhUtjsvQXA9/bsFGhf6UqSscbQ==
user@laptop
```

Si la configuration est correcte, le serveur ne vous demande plus le mot de passe de l'utilisateur distant (ex: root@server's password).

- **Ce qui se passe :** La négociation cryptographique se fait en arrière-plan.
- **Nuance importante :** Si vous avez protégé votre clé privée avec une passphrase (ce que vous devriez faire), votre machine **locale** vous demandera de la déverrouiller.
    - *Différence clé :* Votre mot de passe ne traverse plus jamais le réseau. Le serveur vérifie simplement la signature cryptographique.

Connection en local

> **Note :** Si ssh-copy-id n'est pas disponible (ou si tu es sur un système minimaliste/ancien), tu dois savoir le faire à la main sans transformer ton serveur en passoire.

l’installation Manuelle

Parfois, le confort de ssh-copy-id n'est pas disponible (serveur non connecté, outil manquant, ou restrictions). Il faut alors opérer manuellement.

- **Le Principe :** Il s'agit de copier le *contenu* de votre clé publique locale (.pub) dans le fichier authorized_keys du serveur distant.
- **La Règle de Survie :** SSH est paranoïaque sur les permissions. Si vos fichiers sont trop ouverts (accessibles aux autres), le démon SSH refusera la connexion par sécurité ("Authentication refused: bad ownership or modes").

Procédure Pas-à-Pas

Création du dossier (.ssh) : Le dossier doit appartenir à l'utilisateur et être fermé à tous les autres.

```
mkdir -p ~/.ssh
chmod 700 ~/.ssh
```

Explication 700 (drwx------) : Seul le propriétaire peut lire, écrire et entrer dans ce dossier.

Ajout de la Clé : On colle la clé publique (copiée depuis votre PC) dans le fichier des clés autorisées.

```
nano ~/.ssh/authorized_keys
```

```
chmod 600 ~/.ssh/authorized_keys
```

Explication 600 (-rw-------) : Seul le propriétaire peut lire et écrire. Personne d'autre. Pas même en lecture.

Logs & Audit

> **Note :** Les logs ne sont pas là pour faire joli. C'est votre seule preuve en cas d'intrusion. Sur les systèmes modernes (systemd), on oublie le vieux /var/log/auth.log, on utilise journalctl.

Surveillance en Temps Réel (Live)

Vous configurez un serveur et vous voulez voir ce qui se passe *maintenant*.

```
sudo journalctl -u ssh -f
```

-u	Unit	Filtre uniquement l'unité de service mentionnée
-f	Follow	(suit le flux en direct, Ctrl+C pour quitter)

Audit de Sécurité : "Qui échoue ?" (Attaques)

C'est ici qu'on repère les bots et les scripts de brute-force. Même si vous avez désactivé les mots de passe, les tentatives apparaîtront ici.

Filtre les logs pour identifier les échecs d'authentification. Cette commande est essentielle pour repérer les tentatives d'intrusion (brute-force) ou les erreurs de configuration client.

Ces entrées sont celles utilisées par des outils comme **Fail2Ban** pour bannir automatiquement les IPs malveillantes après X tentatives.

On cherche les échecs explicites de mot de passe, utile pour calibrer Fail2Ban

```
sudo journalctl -u ssh | grep "Failed password" | tail -n 20
```

On cherche les tentatives sur des utilisateurs inexistants

```
sudo journalctl -u ssh | grep "Invalid user" | tail -n 20
```

Audit d'Usage : "Qui est entré ?" (Succès)

Attention au piège des versions modernes d'OpenSSH (Debian 12+ / RHEL 9+). Le démon est très bavard et génère plusieurs lignes "Accepted" pour une seule connexion.

```
sudo journalctl -u ssh | grep "Accepted" | tail -n 10
```

Description : Affiche toutes les étapes de validation positive lors d'une connexion SSH.

Limitation (Debian 12+ / OpenSSH récents) : Sur les versions modernes, le démon SSH (sshd-session) est verbeux. Il génère plusieurs lignes contenant "Accepted" pour une seule connexion réelle (ex: une ligne pour dire que la clé est trouvée, une autre pour dire qu'elle est acceptée). *Risque :* Utiliser ce filtre pour des statistiques (avec wc -l) faussera les résultats en triplant le nombre de connexions perçues.

La commande pro : On filtre sur la méthode d'authentification validée.

Affiche uniquement les validations finales par clé publique, une ligne = Une vraie connexion réussie.

```
sudo journalctl -u ssh | grep "Accepted publickey" | tail -n 10
```

Description : Filtre uniquement la validation finale de l'authentification par clé asymétrique.

Avantage technique : Contrairement au filtre générique "Accepted", celui-ci garantit une relation **1 ligne = 1 session ouverte**. C'est le filtre recommandé pour le monitoring et le comptage précis des utilisateurs connectés, car il élimine les doublons de logs liés au même PID (Process ID).

Troubleshooting : "Pourquoi ça casse ?"

Quand la connexion échoue mais que ce n'est pas une erreur de mot de passe (ex: déconnexion brutale, erreur de protocole).

```
sudo journalctl -u ssh -n 50 | grep -E "disconnect|preauth|error"
```

-n 50 Affiche les 50 dernières lignes (pas tout l'historique)

Le Fichier Magique : ~/.ssh/config

Ce fichier agit comme un carnet d'adresses intelligent qui mappe des options complexes vers un nom simple.

Le constat :

```
ssh -i ~/.ssh/ma_cle_super_longue user_complique@192.168.1.50 -p 5022
```

Taper cette commande vingt fois par jour est un signe de troubles mentaux. Les sysadmins efficaces utilisent des alias.

Création et Edition

```
nano ~/.ssh/config
```

```
# Configuration pour le server
Host mon-vps
    HostName 192.168.1.50         # L'IP ou le domaine réel
    User main                     # L'utilisateur distant
    Port 50922                    # Le port SSH personnalisé (Sécurité)
    IdentityFile ~/.ssh/id_pro    # La clé spécifique à ce serveur

# Configuration pour le Gitlab interne
Host gitlab
    HostName git.entreprise.lan
    User git
    IdentityFile ~/.ssh/id_git
    UserKnownHostsFile ~/.ssh/known_hosts_gitlab        # On dit à SSH d'ignorer le fichier global known_hosts
                                                        pour cette machine et d'utiliser un fichier dédié.
```

Cas Particulier : La Gestion de la "Schizophrénie" (Dropbear vs OpenSSH) Quand vous utilisez le déverrouillage à distance (voir ANNEXE LUKS), votre serveur possède deux identités (Host Keys) différentes pour la même IP : une pour le Boot (Dropbear) et une pour l'OS (OpenSSH). Par défaut, SSH va crier au piratage ("WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!").

La Solution Propre : On isole la mémoire de cette machine dans un fichier à part, avec l’option UserKnownHostsFile.

> **Note :** "C'est comme avoir deux passeports pour la même personne. Si vous les mélangez à la douane, vous finissez en salle d'interrogatoire. Avec UserKnownHostsFile, vous avez un guichet dédié. Circulez, y'a rien à voir."

Utilisation

Désormais, pour vous connecter, vous tapez uniquement :

```
ssh mon-vps
```

SSH va lire le fichier, trouver le bloc mon-vps et appliquer tous les paramètres (IP, Port, Clé) automatiquement.

Permissions (Détail Pro)

Comme pour les clés, ce fichier contient des infos sensibles.

```
chmod 600 ~/.ssh/config
```

La Bannière d'Avertissement (Legal Warning)

> **Note :** La bannière SSH, c'est le paillasson "Bienvenue" remplacé par un panneau "Tir à vue". Ça n'arrête pas les balles, mais ça couvre tes arrières juridiques.

Le Principe : Avant même de demander un login, le serveur affiche un message.

- **Utilité Juridique :** Pour qu'une intrusion soit pénalement répréhensible dans certaines juridictions, il faut que l'accès soit explicitement notifié comme "Restreint/Interdit".
- **Utilité Psychologique :** Ça fait pro. Et ça peut effrayer un script kiddie égaré (bon, on peut rêver).

Rédaction du Message

On crée le fichier dans un emplacement standard et propre

```
sudo nano /etc/ssh/banner.advert
```

Rédiger un avertissement !

```
+-----------------------------------------------------------------+
|                       WARNING: RESTRICTED ACCESS                |
|                                                                 |
|  This system is for the use of authorized users only.           |
|  Individuals using this computer system without authority, or   |
|  in excess of their authority, are subject to having all of     |
|  their activities on this system monitored and recorded.        |
|                                                                 |
|  Anyone using this system expressly consents to such monitoring |
|  and is advised that if such monitoring reveals possible        |
|  evidence of criminal activity, system personnel may provide    |
|  the evidence of such monitoring to law enforcement officials.  |
+-----------------------------------------------------------------+
```

(C'est le texte standard utilisé par l'armée US et les grandes corpos. C'est froid, c'est juridique, c'est parfait.)

Activation dans la Config

Il faut lier le fichier au démon SSH.

```
sudo nano /etc/ssh/sshd_config
```

Pas de commentaire (#) devant !

```
Banner /etc/ssh/banner
```

Validation et Relance

On ne redémarre jamais sans tester la syntaxe.

```
sudo sshd -t
```

Si aucune erreur ne s'affiche, on relance

```
sudo systemctl restart ssh
```

Configuration Serveur : Les Bases

Règle d'Or (Le Filet de Sécurité) : Avant de toucher à un fichier système critique, on le sauvegarde. Toujours. Si vous cassez la conf, vous pourrez revenir en arrière en une seconde au lieu de pleurer sur StackOverflow.

Le Backup (Obligatoire)

On copie le fichier avec une extension explicite (.bak ou .old).

```
sudo cp -p /etc/ssh/sshd_config /etc/ssh/sshd_config.bak
```

-p préserve les permissions (très important pour SSH)

Changement du Port (Réduire le Bruit)

Par défaut, SSH écoute sur le 22. C'est là que tous les bots de la planète tapent en permanence. Changer ce port ne vous rend pas invulnérable (un scan complet vous trouvera), mais cela **allège considérablement** vos logs (/var/log/auth.log).

```
sudo nano /etc/ssh/sshd_config
```

Cherchez la ligne Port 22 (souvent commentée #) et modifiez-la :

On choisit un port élevé (> 1024) et non standard

```
Port 50922
```

Validation et Application (Le Crash Test)

Ne redémarrez jamais le service à l'aveugle. Si vous avez fait une faute de frappe, le service ne redémarrera pas et vous serez éjecté.

Vérification de la syntaxe

```
sudo sshd -t
```

Redémarrage du service

```
sudo systemctl restart ssh
```

Rappel de survie : Gardez votre session actuelle ouverte et testez la connexion dans un **nouveau terminal** avant de tout fermer !

Hardening (Le Durcissement)

Philosophie : "Tout ce qui n'est pas explicitement autorisé est interdit." Nous allons utiliser un fichier de surcharge. Cela permet de garder le fichier /etc/ssh/sshd_config d'origine propre (et évite les conflits lors des mises à jour du système).

Création du fichier de surcharge

Le fichier principal contient une ligne Include /etc/ssh/sshd_config.d/*.conf. Nous allons créer notre propre fichier qui sera lu en priorité.

SSH lit les fichiers dans l'ordre alphabétique (et numérique). Le fichier principal /etc/ssh/sshd_config contient généralement une seule ligne active tout en haut : `Include /etc/ssh/sshd_config.d/*.conf`.

On crée un fichier avec un nom explicite, le préfixe "00-" assure qu'il est lu en premier

```
sudo nano /etc/ssh/sshd_config.d/00-hardering.conf
```

Les Directives

Voici la configuration blindée. Chaque ligne est commentée pour que vous compreniez ce que vous faites (pour changer).

```
# --- 1. ACCÈS & AUTHENTIFICATION ---

# Interdiction ABSOLUE de se connecter en root
# Si un attaquant veut entrer, il doit deviner un nom d'utilisateur ET avoir la clé.
PermitRootLogin no

# Désactivation totale des mots de passe
# On ne fait confiance qu'aux clés cryptographiques.
PasswordAuthentication no
PermitEmptyPasswords no

# Liste Blanche (Whitelist) des utilisateurs
# Seuls ces comptes ont le droit de SEULEMENT tenter une connexion.
# Remplacez 'admin_sys' par votre user réel !
AllowUsers admin_sys

# Nombre maximum d'essais avant rejet (Lutte contre le Brute Force)
# Par défaut c'est 6. On descend à 3.
MaxAuthTries 3

# Nombre maximum de sessions simultanées par connexion réseau
# Limite l'impact d'un script automatisé qui tenterait de lancer 50 terminaux.
MaxSessions 2

# --- 2. RÉSEAU & TUNNELS ---

# Désactiver le déport d'affichage graphique (X11)
# Sauf si vous aimez faire transiter des fenêtres Linux sur le réseau (Lent et insécurisé).
X11Forwarding no

# Désactiver le transfert de ports TCP (Port Forwarding)
# Empêche un attaquant d'utiliser votre serveur comme rebond (Pivot) pour attaquer le reste du réseau.
# À mettre à "yes" UNIQUEMENT si vous utilisez ce serveur comme bastion/VPN SSH.
AllowTcpForwarding no

# Désactiver le transfert d'agent SSH
# Empêche un admin malveillant sur le serveur de récupérer vos identifiants locaux.
AllowAgentForwarding no

# --- 3. LOGS & SURVIE ---

# Augmenter la verbosité des logs
# Permet de voir les détails des échanges de clés dans /var/log/auth.log ou journalctl.
LogLevel VERBOSE

# Désactiver le KeepAlive TCP
# On préfère que ce soit le client qui maintienne la connexion, pas le serveur.
TCPKeepAlive no

# Déconnexion automatique des fantômes (Idle Timeout)
# Si le client n'envoie aucune donnée pendant 300 secondes (5 min)...
ClientAliveInterval 300
# ... on vérifie 2 fois. Si pas de réponse : Dehors.
ClientAliveCountMax 2
```

Validation et Application

Ne sautez jamais cette étape, sauf si vous aimez perdre l'accès à vos serveurs distants.

```
sudo sshd -t
```

Redémarrer le service

```
sudo systemctl restart ssh
```

Le Tunneling SSH (Port Forwarding)

> **Note :** Jusqu'ici, nous avons appris à fermer des portes. Maintenant, nous allons apprendre à percer les murs. Le Tunneling est la capacité de faire passer n'importe quel protocole réseau à travers le tuyau sécurisé du SSH. C'est comme construire une ligne de métro privée sous une ville en guerre.

Attention : Pour que cela fonctionne, la directive AllowTcpForwarding du serveur doit être sur yes (ou local).

Le Transfert Local (-L) : "Le Grappin"

C'est la méthode la plus courante. Elle permet d'accéder à un service distant (ex: Base de données, Interface Web admin ...) comme s'il était sur votre propre machine.

```
ssh -L [PortLocal]:[CibleDistante]:[PortCible] user@server
```

PortLocal Le port qui s'ouvrira sur *votre* machine (Laptop).

CibleDistante L'adresse de la cible *vue depuis le serveur*. Souvent localhost.127.0.0.1

PortCible Le port réel du service sur le serveur distant.

Exemple Concret (Le cas Syncthing) : Vous voulez accéder à l'interface Syncthing (port 8384) du serveur, qui est bloquée par le pare-feu, en passant par le port 9090 de votre laptop.

```
ssh -L 9090:127.0.0.1:8384 user@ip_serveur
```

Résultat : Ouvrez votre navigateur sur http://localhost:9090. Vous y êtes.

Le Transfert Distant (-R) : "Le Cheval de Troie"

C'est l'inverse absolu. Vous voulez donner accès à un service de *votre* machine locale (ou de votre réseau local) à quelqu'un situé sur le serveur distant (ou à l'internet public si la config le permet).

```
ssh -R [PortDistant]:[CibleLocale]:[PortLocal] user@serveur
```

PortDistant Le port qui s'ouvrira sur le *serveur*.

CibleLocale L'adresse du service sur *votre* réseau.

Scénario (Support Technique) : Vous développez une app Web sur votre laptop (port 3000) et vous voulez la montrer à un collègue connecté sur le VPS, sans déployer.

```
ssh -R 8080:localhost:3000 user@vps
```

Résultat : Sur le VPS, un curl http://localhost:8080 tapera directement sur votre laptop.

Le Tunnel Dynamique (-D) : "Le Fantôme (Proxy SOCKS)"

Transforme votre serveur SSH en proxy complet (SOCKS5). Tout le trafic de votre navigateur passera par le tunnel. C'est le "VPN du pauvre" (ou de l'intelligent).

```
ssh -D [PortLocal] user@serveur
```

Exemple : Vous êtes sur un Wi-Fi public douteux (hôtel, aéroport).

```
ssh -D 1080 user@vps
```

Configuration Client : Configurez votre navigateur ou OS pour utiliser un Proxy SOCKS5 sur 127.0.0.1:1080.
Résultat : Votre IP publique devient celle du VPS. Le trafic est chiffré localement.

Les Options "Ninja" (Confort & Discrétion)

Ces options sont souvent combinés avec les tunnels pour éviter de lancer un shell inutile.

-N	**No Remote Command**	Ne lance pas de shell. SSH se connecte, monte le tunnel et... attend.
-f	**Fork to Background**	Passe la commande en arrière-plan une fois l'authentification (clé/mot de passe) réussie.
-C	**Compression**	Compresse les données dans le tunnel. Utile pour le texte/web lent, inutile pour la vidéo/zip.

La Commande "Presse-Bouton" (Alias recommandé) :

 # Lance le tunnel en fond, sans bruit, compressé.

```
ssh -f -N -C -L 9090:127.0.0.1:8384 user@server
```

> **Avertissement :** Un tunnel lancé avec -f est facile à oublier. Il reste actif tant que vous ne le tuez pas. Si vous lancez ps aux | grep ssh trois jours plus tard et que vous voyez 15 tunnels zombifiés, ne venez pas vous plaindre que votre RAM fuit.
>
> **Nettoyage :** killall ssh (pour les barbares) ou kill [PID] (pour les chirurgiens).