La ligne de commande

- Conventions dans la documentation
- Memo en vrac
- SSH
  - Utilisation d'une YubiKey comme clé SSH avec FIDO2
  - Sécuriser SSH avec des clés
- Screen
- Raccourcis claviers

Conventions dans la documentation

Convention des Symboles `$` et `#` dans les Commandes Linux

Introduction

Dans les documentations, tutoriels, et exemples de commandes Linux, vous verrez souvent les symboles `$` et `#` au début des lignes de commande. Ces symboles ne font pas partie de la commande elle-même, mais servent à indiquer dans quel contexte la commande doit être exécutée, que ce soit en tant qu'utilisateur normal ou administrateur (root).

Symbole `$`

Le symbole `$` est utilisé pour représenter l'invite de commande d'un **utilisateur non-root** (c'est-à-dire un utilisateur sans privilèges administratifs). Cela signifie que la commande peut être exécutée directement par un utilisateur classique.

Exemple :

```
$ ls -l
```

Dans cet exemple :

- `$` indique que la commande `ls -l` est exécutée par un utilisateur normal.
- La sortie de la commande (le résultat) apparaîtra après l'exécution.

Symbole `#`

Le symbole `#`, quant à lui, est utilisé pour représenter l'invite de commande d'un **superutilisateur (root)** ou lorsque la commande nécessite des privilèges administratifs. Si vous

voyez une commande précédée de `#`, cela signifie que vous devez l'exécuter avec des privilèges élevés, généralement en utilisant `sudo` (si vous êtes un utilisateur non-root).

Exemple :

```
# apt update
```

Dans cet exemple :

- `#` signifie que la commande `apt update` nécessite des privilèges root pour s'exécuter correctement.
- Vous devrez soit vous connecter en tant que root, soit précéder la commande de `sudo` :

  ```
  $ sudo apt update
  ```

Résumé

- `$` : Invite de commande d'un utilisateur non-administrateur. Exécutez les commandes directement.
- `#` : Invite de commande pour un superutilisateur (root). Exécutez les commandes avec `sudo` si vous n'êtes pas root.

Exemple Comparatif :

```
$ lsmod | grep 8021q      # Commande exécutée par un utilisateur normal
# systemctl restart nginx  # Commande nécessitant des privilèges root
```

Remarque

Le symbole `$` ou `#` n'est **pas** à taper lors de l'exécution des commandes, il est uniquement là pour indiquer le contexte d'exécution.

Conclusion

Ces conventions sont très courantes dans les environnements Unix/Linux et aident à comprendre rapidement si une commande peut être exécutée par un utilisateur classique ou si elle nécessite des droits d'administrateur.

Memo en vrac

Pour quitter une session Telnet :
`Ctrl + $`

Pour afficher la version du système (Ubuntu):

```
lsb_release -a
```

Pour générer un secret aléatoire :

```
openssl rand -hex 32
```

Pour obtenir des informations sur le matériel :

```
sudo lshw
```

Pour obtenir des informations sur des composants précis :

```
sudo lshw -C cpu
sudo lshw -C memory
sudo lshw -C network
sudo lshw -C disk
```

SSH

Utilisation d'une YubiKey comme clé SSH avec FIDO2

Prérequis

Vérifier que OpenSSH 8.2 ou supérieur est installé sur le client et le serveur.

```
ssh -V
```

Vérifier la compatibilité de la YubiKey

Deux types de paires de clés sont compatibles avec FIDO2 : `ecdsa-sk` et `ed25519-sk`.

- L'extension -sk signifie "security key".
- Le type `ed25519-sk` nécessite une YubiKey avec le firmware 5.2.3 ou supérieur.

> Les clés `ed25519-sk` nécessitent une YubiKey avec un **firmware en version 5.2.3 ou supérieure**, compatible FIDO2.

Pour afficher la version du firmware de la YubiKey :

```
lsusb -v 2>/dev/null | grep -A2 Yubico | grep "bcdDevice" | awk '{print $2}'
```

Installer les dépendances nécessaires

```
sudo apt install libfido2-dev
```

Générer une paire de clés SSH avec FIDO2

Créer une clé SSH sécurisée par YubiKey :

```
ssh-keygen -t ed25519-sk -C "$(hostname)-$(date +'%Y%m%d')-yubikey1"
```

Une partie de la clé privée est stockée dans la YubiKey. Elle repose sur un mécanisme de défi-réponse sécurisé. La clé publique, un handle et des données d’attestation sont générés.

Copier la clé publique sur un serveur

Sous Linux :

```
ssh-copy-id -i ~/.ssh/id_ed25519-sk.pub user@serveur
```

Sous Windows (PowerShell) :

```
type $env:USERPROFILE.ssh\id_ed25519-sk.pub | ssh user@serveur "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
```

Remplacer user@serveur par les informations du serveur cible.

Sources

https://forums.lawrencesystems.com/t/ssh-with-yubikey-fido-u2f-authentication/13024

https://developers.yubico.com/SSH/Securing_SSH_with_FIDO2.html

Sécuriser SSH avec des clés

Comprendre ce qui se passe lors d'une connexion

Avant de configurer les clés, il est important de comprendre ce qui se passe réellement lors d'une connexion SSH.

Connexion classique avec un mot de passe

Avec une connexion classique :

```
ssh admin@server01
```

SSH demande le mot de passe du **compte `admin` sur `server01`** :

```
admin@server01's password:
```

Il faut donc connaître le mot de passe du compte distant.

Le fonctionnement est essentiellement :

1. Le PC se connecte au serveur SSH.
2. Le serveur demande une méthode d'authentification.
3. Le client indique qu'il souhaite utiliser un mot de passe.
4. L'utilisateur saisit le mot de passe du compte `admin`.
5. Le serveur vérifie le mot de passe.
6. Si celui-ci est correct, la connexion est établie.

Connexion avec une clé SSH

Avec une clé SSH, le fonctionnement change.

On commence par générer deux éléments sur le PC :

```
clé privée
clé publique
```

La clé privée reste sur le PC.

La clé publique est installée sur le serveur.

Lors d'une connexion :

```
ssh admin@server01
```

le serveur constate que ce compte accepte une clé correspondant à celle présentée par le client.

SSH utilise alors un mécanisme cryptographique pour prouver que le client possède bien la clé privée.

La clé privée **n'est pas envoyée au serveur**.

---

3. Quel mot de passe est demandé avec une clé SSH ?

Il y a une différence essentielle entre deux mots de passe/passphrases.

Mot de passe du serveur

Avec une authentification par clé correctement configurée, le mot de passe du compte distant n'est normalement plus demandé.

Par exemple :

```
ssh admin@server01
```

peut simplement donner :

```
Last login: ...
admin@server01:~$
```

Aucun mot de passe du compte `admin` n'a été demandé.

Passphrase de la clé privée

Si la clé privée est protégée par une passphrase, SSH peut en revanche demander :

```
Enter passphrase for key '/home/user/.ssh/id_ed25519':
```

Cette passphrase ne correspond **pas** au mot de passe du serveur.

Elle sert à déverrouiller la clé privée présente sur ton ordinateur.

C'est donc :

> **Le mot de passe du serveur authentifie le compte sur le serveur.**

alors que :

> **La passphrase protège la clé privée présente sur le PC.**

---

4. Pourquoi mettre une passphrase sur la clé ?

On pourrait créer une clé sans passphrase.

Dans ce cas, quelqu'un qui récupère le fichier de clé privée pourrait potentiellement l'utiliser pour se connecter aux serveurs auxquels cette clé donne accès.

Avec une passphrase :

```
clé privée volée
      ↓
clé protégée
      ↓
passphrase nécessaire
      ↓
accès possible uniquement si la passphrase est également connue
```

La passphrase constitue donc une deuxième protection contre le vol de la clé privée.

Il est recommandé d'en utiliser une.

---

5. Pourquoi ne doit-on pas mettre la clé privée sur le serveur ?

Le serveur n'a besoin que de la clé publique.

La clé privée reste sur le PC :

```
PC
├── clé privée       ← secrète
└── clé publique     ← peut être copiée sur le serveur
```

Sur le serveur :

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

contient la ou les clés publiques autorisées.

Le serveur ne récupère jamais la clé privée.

6. Et si je dois utiliser plusieurs PC ?

C'est ici qu'une bonne organisation devient importante.

Supposons que tu utilises :

- un PC fixe ;
- un laptop ;
- un Mac ;
- éventuellement une machine dédiée à l'administration.

Il est préférable de créer **une clé différente sur chaque PC**.

Par exemple :

Machine	Clé
PC fixe	clé A
Laptop	clé B
Mac	clé C
PC administration	clé D

Le serveur peut alors autoriser les quatre clés.

Cela permet de révoquer individuellement un ordinateur.

Exemple

Si le laptop est perdu, on retire uniquement sa clé publique du serveur.

Les autres machines continuent de fonctionner.

Il n'est donc pas nécessaire de remplacer les clés de tous les autres ordinateurs.

7. Générer une clé SSH

Sur Linux, macOS ou Windows avec OpenSSH :

```
ssh-keygen -t ed25519 -C "pc-fixe"
```

Le programme demande où enregistrer la clé :

```
Enter file in which to save the key:
```

Pour une clé standard :

```
~/.ssh/id_ed25519
```

Puis :

```
Enter passphrase:
```

Choisir une passphrase.

Il est recommandé de ne pas laisser la clé privée sans protection.

---

8. Les fichiers créés

Après génération :

```
ls -la ~/.ssh/
```

On retrouve notamment :

```
id_ed25519
id_ed25519.pub
```

La différence est fondamentale :

```
id_ed25519
```

est la **clé privée**.

Elle doit rester secrète.

```
id_ed25519.pub
```

est la **clé publique**.

Elle peut être installée sur les serveurs.

Ne jamais copier :

```
id_ed25519
```

sur un serveur simplement pour permettre une connexion SSH.

---

9. Installer la clé publique sur le serveur

Tant que l'authentification par mot de passe est activée, on peut utiliser :

```
ssh-copy-id admin@server01
```

Le mot de passe du compte `admin` sur `server01` sera demandé.

`ssh-copy-id` ajoute ensuite la clé publique à :

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

du compte distant.

On peut alors tester :

```
ssh admin@server01
```

Si la clé privée est protégée par une passphrase, SSH peut demander :

```
Enter passphrase for key '/home/user/.ssh/id_ed25519':
```

Ce n'est donc plus le mot de passe de `admin` sur le serveur.

10. Première connexion après l'installation de la clé

Il peut également y avoir une question différente lors de la première connexion à un serveur.

Exemple :

```
The authenticity of host 'server01' can't be established.
ED25519 key fingerprint is SHA256:xxxxxxxx...
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Cette question concerne **l'identité du serveur**, pas ton compte utilisateur.

SSH veut savoir si tu acceptes la clé d'hôte présentée par `server01`.

Après avoir accepté cette clé, elle est enregistrée dans :

```
~/.ssh/known_hosts
```

Lors des connexions suivantes, SSH peut comparer la clé présentée par le serveur avec celle enregistrée.

---

11. Les trois éléments à ne pas confondre

Il existe donc trois choses différentes.

Élément	Où ?	Rôle
Mot de passe du compte	Serveur	Authentifier le compte lorsque l'authentification par mot de passe est utilisée
Passphrase de la clé privée	PC client	Protéger la clé privée

Élément	Où ?	Rôle
Clé d'hôte du serveur	PC client → `known_hosts`	Vérifier l'identité du serveur

C'est une distinction importante pour comprendre les messages affichés par SSH.

---

12. Utiliser `ssh-agent`

Une question peut alors se poser :

> Si ma clé est protégée par une passphrase, dois-je la saisir à chaque connexion ?

Pas nécessairement.

`ssh-agent` permet de conserver temporairement une clé déverrouillée en mémoire.

Ajouter la clé :

```
ssh-add ~/.ssh/id_ed25519
```

SSH demande alors la passphrase une fois.

Ensuite :

```
ssh server01
ssh server02
ssh nas
ssh proxmox
```

peuvent utiliser la clé sans redemander systématiquement la passphrase.

Vérifier les clés chargées :

```
ssh-add -l
```

Retirer une clé :

```
ssh-add -d ~/.ssh/id_ed25519
```

Retirer toutes les clés :

```
ssh-add -D
```

Le comportement exact de `ssh-agent` dépend du système d'exploitation et de l'intégration avec le gestionnaire de session ou le trousseau de clés.

---

13. Configurer plusieurs serveurs avec `~/.ssh/config`

Lorsque plusieurs serveurs sont administrés, il devient rapidement pratique de centraliser les paramètres SSH.

Créer :

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

Exemple :

```
Host server01
    HostName 192.168.1.10
    User admin
    IdentityFile ~/.ssh/id_ed25519

Host server02
    HostName 192.168.1.11
    User admin
    IdentityFile ~/.ssh/id_ed25519

Host nas
    HostName 192.168.1.20
    User admin
    IdentityFile ~/.ssh/id_ed25519

Host proxmox
    HostName 192.168.1.30
```

```
    User root
    IdentityFile ~/.ssh/id_ed25519
```

On peut alors simplement utiliser :

```
ssh server01
```

au lieu de :

```
ssh admin@192.168.1.10
```

---

14. Utiliser les noms DNS du homelab

Si le réseau local possède un DNS, utiliser des noms plutôt que des adresses IP est encore plus pratique.

Exemple :

```
Host server01
    HostName server01.home.arpa
    User admin
    IdentityFile ~/.ssh/id_ed25519

Host nas
    HostName nas.home.arpa
    User admin
    IdentityFile ~/.ssh/id_ed25519
```

On peut ensuite simplement faire :

```
ssh nas
```

---

15. Permissions des fichiers SSH

Sous Linux et macOS, vérifier les permissions :

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_ed25519
chmod 644 ~/.ssh/id_ed25519.pub
chmod 600 ~/.ssh/authorized_keys
chmod 600 ~/.ssh/config
```

La clé privée doit être lisible uniquement par son propriétaire.

---

16. Désactiver l'authentification par mot de passe

Une fois que l'authentification par clé fonctionne correctement, on peut désactiver l'authentification SSH par mot de passe.

Avant cela :

> **Ne fermer pas votre session SSH actuelle.**

Il faut d'abord ouvrir une **deuxième session** et vérifier que l'authentification par clé fonctionne.

Sur Linux, la configuration du serveur se trouve généralement dans :

```
/etc/ssh/sshd_config
```

ou dans :

```
/etc/ssh/sshd_config.d/
```

Une configuration de durcissement peut par exemple contenir :

```
PubkeyAuthentication yes
PasswordAuthentication no
KbdInteractiveAuthentication no
PermitRootLogin no
```

---

17. Vérifier la configuration SSH

Avant de redémarrer SSH :

```
sudo sshd -t
```

S'il n'y a aucune sortie, la syntaxe est généralement correcte.

Redémarrer ensuite le service :

```
sudo systemctl restart ssh
```

ou, selon la distribution :

```
sudo systemctl restart sshd
```

Tester immédiatement une nouvelle connexion depuis un autre terminal.

---

18. Ce que l'utilisateur verra après le durcissement

Une fois :

```
PasswordAuthentication no
```

configuré, une connexion avec une clé fonctionnera ainsi :

```
ssh admin@server01
```

Si la clé n'est pas déjà déverrouillée dans `ssh-agent`, SSH peut demander :

```
Enter passphrase for key '/home/user/.ssh/id_ed25519':
```

Cette passphrase déverrouille la clé privée locale.

Le mot de passe du compte `admin` sur le serveur n'est plus demandé.

Si aucune clé valide n'est disponible, la connexion échoue au lieu de proposer le mot de passe.

C'est exactement le comportement recherché.

---

19. Tester que le mot de passe est désactivé

Depuis le poste client :

```
ssh \
    -o PreferredAuthentications=password \
    -o PubkeyAuthentication=no \
    admin@server01
```

La connexion doit être refusée si l'authentification par mot de passe est bien désactivée.

---

20. Ne pas exposer SSH directement sur Internet

Pour un homelab, il est préférable d'éviter d'exposer directement le port SSH à Internet lorsque ce n'est pas nécessaire.

Une architecture plus sûre consiste à accéder d'abord au réseau du homelab via un VPN, puis à utiliser SSH.

Par exemple :

```
Internet
    ↓
VPN
    ↓
Réseau d'administration
    ↓
Serveurs SSH
```

Les solutions VPN courantes incluent notamment WireGuard.

Le principe est de ne rendre SSH accessible qu'aux réseaux depuis lesquels l'administration doit réellement être possible.

---

21. Ne pas utiliser `root` directement lorsque ce n'est pas nécessaire

Sur les serveurs Linux, il est généralement préférable d'utiliser un compte d'administration classique :

```
ssh admin@server01
```

puis :

```
sudo commande
```

plutôt que :

```
ssh root@server01
```

On peut donc utiliser :

```
PermitRootLogin no
```

et conserver l'administration via `sudo`.

---

22. Gérer plusieurs clés sur une même machine

Dans certains cas, une seule clé par machine ne suffit plus.

Par exemple :

```
~/.ssh/
├── id_ed25519
├── id_ed25519.pub
├── id_ed25519_automation
└── id_ed25519_automation.pub
```

La clé personnelle peut servir à l'administration interactive.

Une autre clé peut être dédiée à une automatisation.

Exemple :

```
Host server01
    HostName server01.home.arpa
```

```
    User admin
    IdentityFile ~/.ssh/id_ed25519

Host server01-automation
    HostName server01.home.arpa
    User automation
    IdentityFile ~/.ssh/id_ed25519_automation
```

Il est préférable de ne pas réutiliser une clé personnelle pour des scripts ou des services.

---

23. Perte ou vol d'un PC

L'intérêt principal d'une clé par PC apparaît lorsqu'une machine est perdue.

Supposons que le laptop possède une clé qui est également installée sur tous les serveurs.

Il faut alors :

1. identifier la clé correspondant au laptop ;
2. supprimer cette clé de `authorized_keys` sur les serveurs concernés ;
3. révoquer les accès VPN associés si nécessaire ;
4. vérifier qu'elle n'est pas utilisée ailleurs ;
5. créer une nouvelle clé sur le nouveau poste.

Les autres clés continuent de fonctionner.

---

24. Identifier une clé avec son fingerprint

Pour identifier précisément une clé :

```
ssh-keygen -lf ~/.ssh/id_ed25519.pub
```

Exemple :

```
256 SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx pc-fixe (ED25519)
```

Il est utile de conserver les fingerprints dans la documentation du homelab.

Exemple :

Machine	Fingerprint	Date	Statut
PC fixe	`SHA256:AAA...`	2026-09	Active
Laptop	`SHA256:BBB...`	2026-09	Active
Ancien PC	`SHA256:CCC...`	2025-02	Révoquée

---

25. Rotation d'une clé

Une clé n'a pas nécessairement besoin d'être changée régulièrement uniquement parce qu'elle est ancienne.

Une nouvelle clé doit surtout être créée lorsqu'il existe une raison de considérer l'ancienne comme compromise ou lorsqu'une machine est remplacée.

Procédure :

1. Générer une nouvelle clé.
2. Installer la nouvelle clé publique.
3. Tester la nouvelle connexion.
4. Retirer l'ancienne clé.
5. Supprimer l'ancienne clé privée du poste si elle n'est plus nécessaire.

Il est préférable de **ne retirer l'ancienne clé qu'après avoir validé la nouvelle**.

---

26. Comprendre `known_hosts`

`known_hosts` et `authorized_keys` ont des rôles différents.

`authorized_keys`

Se trouve côté serveur :

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

Il contient les clés publiques des utilisateurs autorisés à se connecter.

`known_hosts`

Se trouve côté client :

```
~/.ssh/known_hosts
```

Il contient les clés d'hôte des serveurs déjà rencontrés.

Le premier protège l'accès **au serveur**.

Le second permet au client de vérifier **l'identité du serveur**.

---

27. Ne pas copier une clé privée dans Git, Docker ou Ansible

Une clé privée personnelle ne doit pas être stockée dans :

- un dépôt Git ;
- une image Docker ;
- un fichier de configuration partagé ;
- un script ;
- un serveur ;
- un partage réseau non sécurisé.

Pour l'automatisation, utiliser une clé dédiée et un mécanisme adapté de gestion des secrets.

---

28. Clés SSH FIDO2

OpenSSH prend également en charge des clés utilisant des authentificateurs matériels FIDO2.

Des types comme :

```
ed25519-sk
ecdsa-sk
```

permettent d'utiliser un périphérique matériel compatible pour renforcer la protection de l'identité SSH.

Cette solution est particulièrement intéressante pour les accès très sensibles.

Pour commencer dans un homelab, le modèle suivant constitue déjà une très bonne base :

```
clé Ed25519
+
passphrase
+
une clé par PC
+
ssh-agent
+
authentification par clé uniquement
```

---

29. Modèle recommandé pour mon homelab

Pour chaque PC d'administration :

```
1 PC
→ 1 clé privée
→ 1 clé publique
→ 1 passphrase
```

Pour chaque serveur :

```
authorized_keys
→ clé du PC fixe
→ clé du laptop
→ clé des autres postes autorisés
```

Pour chaque connexion :

```
ssh server01
```

SSH utilise :

1. la configuration de `~/.ssh/config` ;
2. la clé privée correspondante ;
3. `ssh-agent` si la clé y est chargée ;
4. la clé publique présente dans `authorized_keys`.

Le mot de passe du compte serveur n'est pas utilisé lorsque l'authentification par clé réussit.

---

30. Checklist

Sur chaque PC

- [ ] Générer une clé Ed25519
- [ ] Protéger la clé avec une passphrase
- [ ] Ne jamais partager la clé privée
- [ ] Configurer `ssh-agent`
- [ ] Configurer `~/.ssh/config`
- [ ] Vérifier les permissions de `~/.ssh`
- [ ] Documenter le fingerprint de la clé

Sur chaque serveur

- [ ] Créer le compte d'administration
- [ ] Installer les clés publiques nécessaires
- [ ] Vérifier `authorized_keys`
- [ ] Tester la connexion par clé
- [ ] Ouvrir une deuxième session SSH
- [ ] Vérifier `sshd -t`
- [ ] Désactiver `PasswordAuthentication`
- [ ] Désactiver le login direct de `root`
- [ ] Tester une nouvelle connexion
- [ ] Vérifier le firewall
- [ ] Éviter l'exposition directe de SSH sur Internet

En cas de perte d'un PC

- [ ] Identifier la clé concernée
- [ ] Supprimer sa clé publique de `authorized_keys`
- [ ] Vérifier les autres serveurs
- [ ] Révoquer les accès VPN associés
- [ ]

- [ ] Générer une nouvelle clé
- [ ] Installer la nouvelle clé
- [ ] Tester les accès

Référence rapide

Générer une clé

```
ssh-keygen -t ed25519 -C "pc-fixe"
```

Afficher la clé publique

```
cat ~/.ssh/id_ed25519.pub
```

Installer une clé

```
ssh-copy-id admin@server01
```

Se connecter

```
ssh admin@server01
```

Ajouter la clé à `ssh-agent`

```
ssh-add ~/.ssh/id_ed25519
```

Voir les clés de l'agent

```
ssh-add -l
```

Afficher le fingerprint

```
ssh-keygen -lf ~/.ssh/id_ed25519.pub
```

Vérifier la configuration SSH

```
sudo sshd -t
```

Fichiers importants

```
~/.ssh/id_ed25519          # clé privée
~/.ssh/id_ed25519.pub      # clé publique
~/.ssh/config              # configuration SSH du client
~/.ssh/known_hosts         # serveurs connus
~/.ssh/authorized_keys     # clés autorisées sur le serveur
/etc/ssh/sshd_config       # configuration du serveur SSH
```

Screen

La commande `screen` permet de gérer des sessions terminal détachables et persistantes. Utile pour lancer des processus longs sur un serveur distant et se déconnecter sans interrompre leur exécution.

Commandes de base

Lancer une session

```
screen
```

Ou avec un nom pour la session :

```
screen -S nom_session
```

Détacher une session

Raccourci clavier : `Ctrl + a`, puis `d`

Lister les sessions existantes

```
screen -ls
```

Reprendre une session

```
screen -r nom_session
```

Ou, si plusieurs sessions sont ouvertes :

```
screen -r ID_session
```

Fermer une session

Dans le terminal de la session :

```
exit
```

Ou terminer une session détachée :

```
screen -X -S nom_session quit
```

Commandes internes utiles

> Toutes commencent par `Ctrl + a`.

Raccourci	Action
`c`	Créer une nouvelle fenêtre
`n`	Aller à la fenêtre suivante
`p`	Aller à la fenêtre précédente
`"`	Liste des fenêtres
`k`	Fermer la fenêtre active
`A`	Renommer la fenêtre active

Astuces

Démarrer un processus dans une session détachée directement

```
screen -dmS nom_session commande
```

Reconnecter une session même si elle est attachée ailleurs

```
screen -x nom_session
```

Personnaliser le fichier de configuration

`~/.screenrc`

Raccourcis claviers

- `Ctrl + A` : aller au début de la ligne
- `Ctrl + E` : aller en fin de ligne
- `Ctrl + L` : effacer le contenu de l'écran (équivalent de `clear`)
- `Ctrl + U` : effacer la ligne jusqu'au début
- `Ctrl + K` : effacer la ligne jusqu'à la fin

Anciennes commandes

Retrouver d'anciennes commandes

Utiliser les flèches haut et bas

Rechercher d'anciennes commandes

Utiliser le raccourci **`Ctrl + R`** pour effectuer une recherche dans l'historique des commandes.

- Appuyer sur `Ctrl + R`, puis saisir une partie de la commande recherchée.
- Le terminal affiche la dernière commande de l'historique correspondant au texte saisi.
- Appuyer de nouveau sur `Ctrl + R` pour rechercher la correspondance précédente.
- Appuyer sur `Entrée` pour exécuter la commande affichée.
- Appuyer sur `Échap` pour quitter la recherche en conservant la commande affichée dans la ligne de commande.
- Appuyer sur `Ctrl + G` pour annuler la recherche et revenir à la ligne de commande initiale.

Exemple :

```
(reverse-i-search)`apt': sudo apt update
```

Dans cet exemple, la recherche porte sur le texte `apt` et affiche une ancienne commande contenant ce texte.

Astuce : il n'est pas nécessaire de saisir le début exact de la commande. Il suffit de taper une partie du texte présent dans celle-ci. La recherche s'effectue dans l'historique des commandes du shell, généralement Bash.