Le Voyage d'une Commande

De la Touche au Résultat : Démystifier Terminal, Shell, Processus et Jobs sous Linux.

Les Protagonistes : Le Terminal est le Véhicule, le Shell est le Pilote

Le Terminal - Le Véhicule

C'est l'interface (la fenêtre) qui vous transporte. Il affiche le texte et capture les entrées clavier, mais il ne décide pas de la destination.

Nature
- Un programme logiciel qui simule un terminal physique.

Exemples
- GNOME Terminal, iTerm2, PuTTY.

Le Shell - Le Pilote

C'est le programme qui comprend vos ordres (commandes) et dirige le véhicule. Il interprète les commandes et exécute les scripts.

Nature
- Un langage de commande et de script.

Exemples
- Bash, Zsh, Fish.

La Carte du Voyage : Les Couches du Système

Couche 4
Le Terminal (La Fenêtre)

Rôle : Capture vos frappes clavier et affiche les caractères en retour.

Couche 3
Le Shell (L'Interprète)

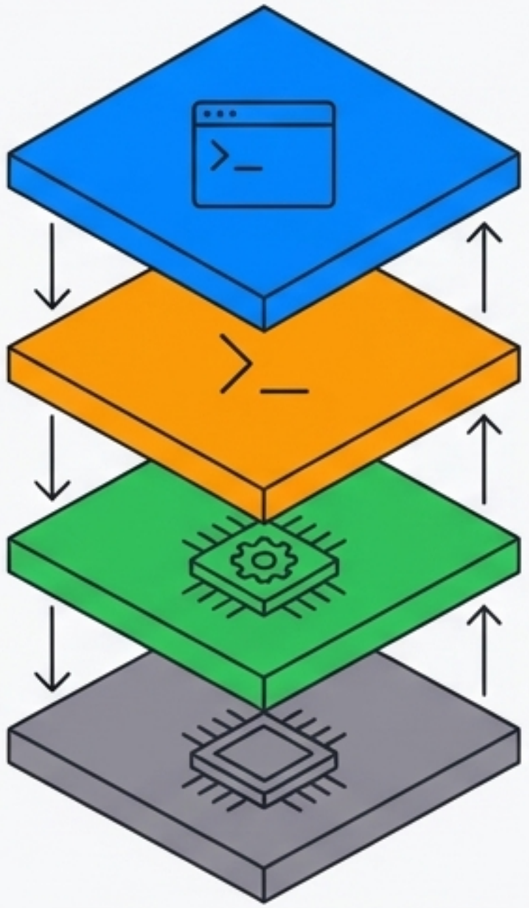
Rôle : Interprète vos commandes et dialogue avec le noyau du système.

Couche 2
Le Noyau (Le Cerveau)

Rôle : Gère les processus, la mémoire et les pilotes matériels.

Couche 1
Le Matériel (La Base)

Rôle : Les composants physiques (CPU, RAM, Disques).



Chaque commande que vous tapez traverse ces couches, de la fenêtre jusqu'au matériel, puis remonte avec le résultat.

Point de Départ : La Frappe au Clavier

Tout commence dans votre **émulateur de terminal**. Ce n'est pas un port physique, mais un programme (GNOME-Terminal, iTerm2, PuTTY) qui simule l'ancien matériel.

Rôle principal

- Capturer vos frappes clavier.
- Afficher les caractères de retour.

Ce programme ouvre une porte vers le système, appelée **PTS (Pseudo-Terminal Slave)**. C'est le point d'entrée logique pour votre commande.

Un Détour par l’Histoire : Pourquoi ’TTY’ ?

Début XXe siècle

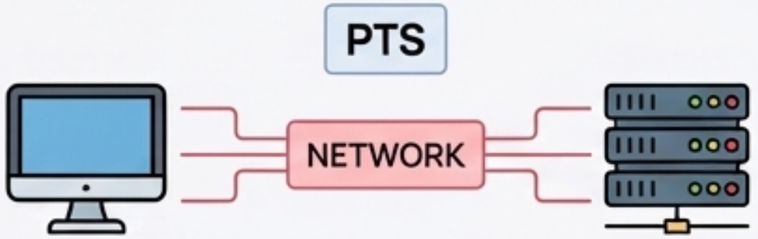
Années 70

Années 80

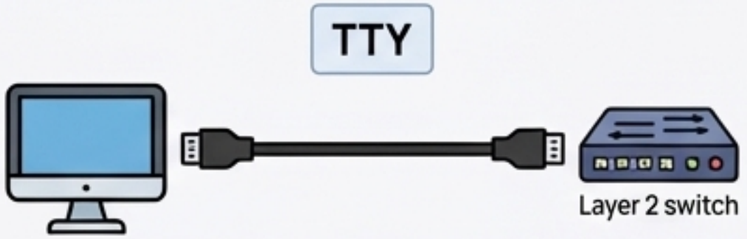
- Le terme `tty` vient de TeleTYpewriter (téléscripteur).
- À l'origine, interagir avec un ordinateur signifiait littéralement imprimer les commandes et les résultats sur du papier. Il n'y avait pas d'écran.

Le saviez-vous ? C'est pour cette raison que de nombreux langages de programmation utilisent encore le mot `print` pour afficher du texte à l'écran. Ce n'était pas une métaphore, mais un fait.

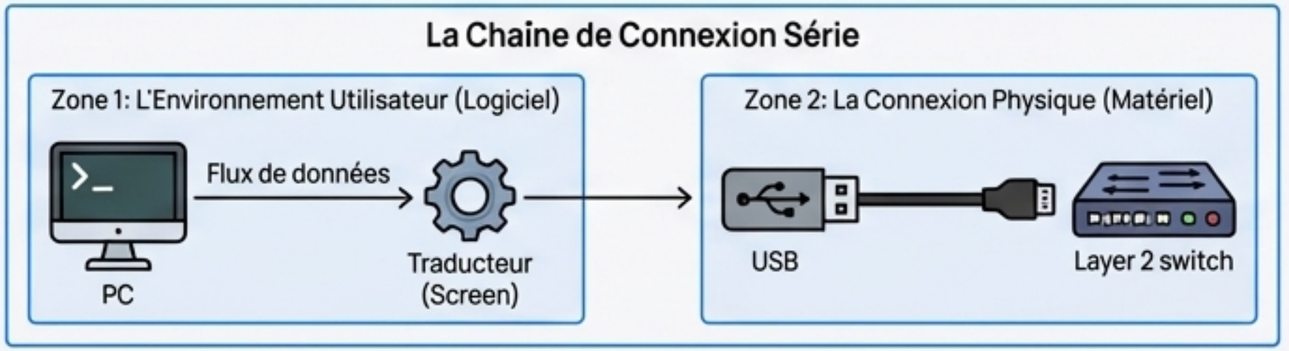
Le Pont vers le Système : TTY vs. PTS



Connexion à distance : Le tty est un PTS (Pseudo Terminal Slave)



Connexion liaison série : Le tty est un tty



TTY (TeleTYpewriter)

- **Quoi:** Une connexion **physique** et directe.
- **Exemples:** Un clavier et une souris directement branchés, ou un port série (`/dev/ttyS0`) pour la maintenance d'un équipement réseau.

PTS (Pseudo-Terminal Slave)

- **Quoi:** Une connexion **virtuelle** ou 'pseudo-terminal'.
- **Exemples:** Chaque fenêtre de terminal que vous ouvrez dans votre interface graphique, ou une session à distance via SSH.

La commande `tty` dans votre terminal vous montrera le fichier de périphérique auquel vous êtes connecté (ex: `/dev/pts/0`).

La Chaîne de Connexion Série : Du Clavier au Switch (La vérité sur le PTS vs TTY)

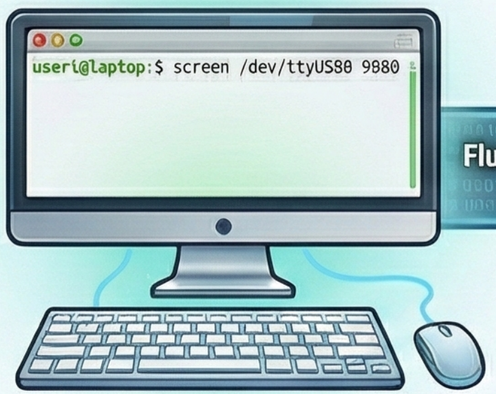
Pourquoi votre fenêtre de terminal moderne reste un émulateur (PTS) mème lorsque vous parlez à un port série physique (TTY).

ZONE 1 : L'ENVIRONNEMENT UTILISATEUR (LOGICIEL) [VOTRE ORDINATEUR]

1. L'INTERFACE : VOTRE ÉMULATEUR (PTS)

Exemples : GNOME Terminal, ITerm2, PuTTY.

Bloc A : Votre Fenêtre
(Le PTS - Pseudo-Terminal Slave)



1. L'INTERFACE : VOTRE ÉMULATEUR (PTS)

Exemples : GNOME Terminal, ITerm2, PuTTY.

Rôle : Capture vos frappes clavier, affiche les caractères de retour.

Nature : C'est un logiciel, pas un port physique. Géré par le noyau.

2. LE TRADUCTEUR : L'OUTIL screen

S'exécute dans votre Shell.

Bloc B : L'Outil de Connexion
(Le Pont)

Flux de données stdin

SCREEN + MINICOM

GRANDE FLÈCHE DE SORTIE : Redirection vers le matériel

2. LE TRADUCTEUR : L'OUTIL screen

Rôle : Intermédiaire logiciel, S'éxécute dans votre Shell.

Action : Lit l'entrée du PTS et la redriige vers le fichier matériel.

Nature : Fait le pont entre le monde logiciel et la demande matérielle.

ZONE 2 : LA CONNEXION PHYSIQUE (MATÉRIEL)

Bloc C : La Vrais Cible
(Le TTY Physique)

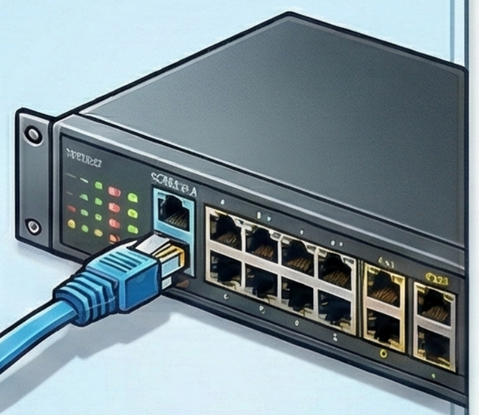
3. LE VRAI TTY : LE PORT SÉRIE PHYSIQUE

Identification : Le fichier de périphérique spécial (ex. /dev/ttyUSBO on /dev/tty50).

Rôle : Le point d'ancrage matériel réel. C'est ici que la donnde devient un signal électrique sur un câble.

ZONE 3 : LA DESTINATION

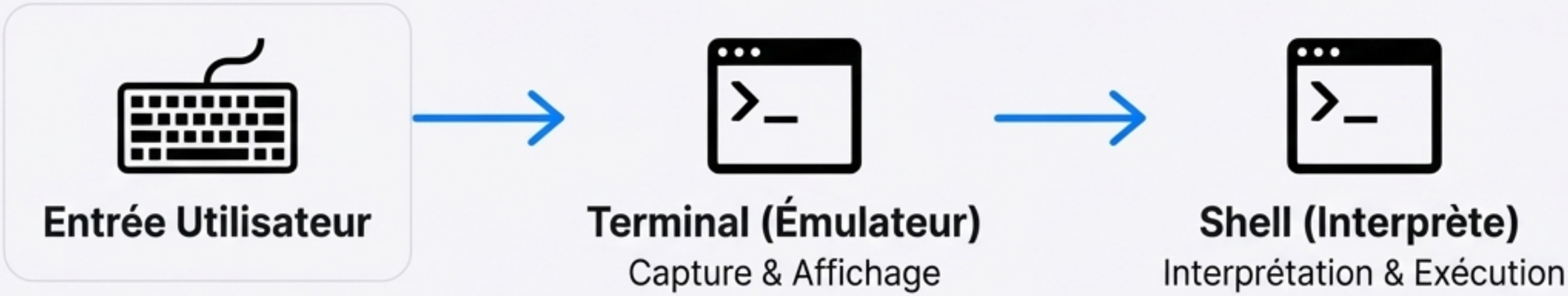
Bloc D : L'Équipement Distant



ÉQUIPEMENT CIBLE (Switch/Routour)

Action : Reçoit les commandes série, exécute et renvoie le résultat (qui refait le chemin inverse).

Le Pilote prend les Commandes : Le Rôle du Shell



Le **Terminal** a capturé le texte ; il le transmet maintenant au **Shell** (ex: Bash, Zsh).

Le **Shell** est le cerveau de l'opération. C'est une interface en ligne de commande qui :

1. **Interprète** ce que vous avez tapé.
2. **Évalue** la commande pour comprendre ce qu'il faut faire.
3. **Met en route** les programmes correspondants.
4. **Renvoie** le résultat qui sera affiché par le Terminal.

Chaque Shell possède son propre langage de script, permettant d'automatiser des tâches complexes.

L’Ordre d’Exécution : La Hiérarchie de Commandement

PROCESSUS ET JOBS : LA HIÉRARCHIE DE COMMANDEMENT LINUX

Zone 1 : La Chaîne de Commandement (La Hiérarchie)

1. Niveau 1 (Haut) : Le Noyau (Kernel) - "Le Grand Patron"
Rôle : Alloue le temps CPU et la mémoire. Il ne voit que des PIDs.

Action : Il supervise le processus Shell.

2. Niveau 2 (Milieu) : Le Shell (ex: Bash) - "Le Chef d'Équipe"
Identité : C'est un Processus standard (ex: PID 1000).

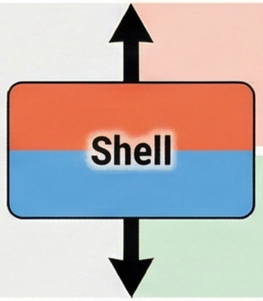
Rôle : Interprète les ordres de l'utilisateur. C'est le Processus Parent.

fork()

3. Niveau 3 (Bas) : La Commande (ex: ls) - "L'Ouvrier Spécialisé"
Identité : C'est le Processus Enfant (ex: PID 1001).
Création : Créé par le Shell via **fork()** pour exécuter une tâche spécifique.

Zone 2 : Le Shell, cette créature hybride

Le Shell est un Processus du point de vue du Noyau, mais un Gestionnaire du point de vue de l'Utilisateur.

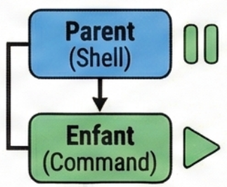


Vers le haut (Noyau) : Le Shell est un exécutant soumis aux règles du système (il a un PID, il consomme de la RAM).

Vers le bas (Utilisateur/Enfants) : Le Shell est un donneur d'ordre. Il gère le cycle de vie des commandes (Jobs).

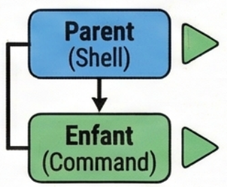
Zone 3 : La Dynamique Parent/Enfant (Règles de Vie)

1. Scénario Standard (Avant-plan)



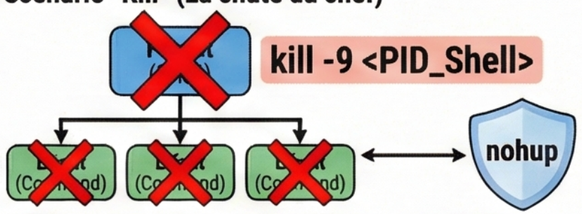
Visuel : Le Parent (Shell) est en pause ⏸. L'Enfant travaille ▶.
Texte : Le Shell attend que l'enfant finisse pour reprendre la main (sauf utilisation de &).

2. Scénario Arrière-plan (&)



Visuel : Le Parent et l'Enfant travaillent en parallèle.
Texte : Le Shell rend la main immédiatement (Job Background).

3. Scénario "Kill" (La chute du chef)



Visuel : Une croix rouge ❌ sur le Shell (Parent). Des croix rouges ❌ apparaissent en cascade sur les Enfants.
Texte : "Si le Shell est tué (kill -9 <PID_Shell>), ses enfants sont tués aussi (signal SIGHUP), car ils perdent leur parent."

Exception : Une bulle de protection nommée **nohup** protège un enfant de la mort du parent.

Zone 4 : Processus vs Job (Le Tableau Comparatif)

Caractéristique	PROCESSUS (Vue Système)	JOB (Vue Shell)
C'est quoi ?	Instance de programme (Ouvrier)	Tâche / Ligne de commande (Mission)
Identifiant	PID (Global, ex: 12450)	JOB ID (Local, ex: %1)
Gestionnaire	Le Noyau (Kernel)	Le Shell (Bash, Zsh)
Commande de gestion	**kill 12450**	**kill %1**
Portée	Tout le système	Fenêtre de terminal actuelle

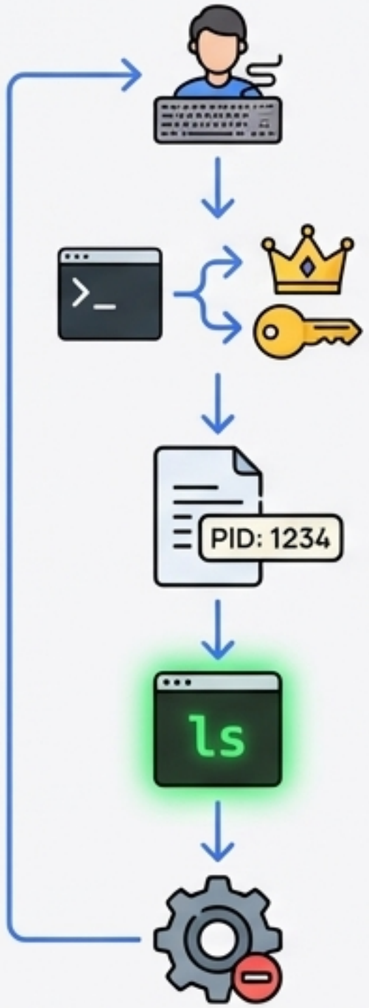
Zone 5 : L'Analogie du Chantier (Bas de page)

Le Noyau : Le propriétaire du terrain (fournit l'électricité et l'espace)

Le Shell (Chef de chantier) : Reçoit les plans (commandes) et engage des ouvriers. Il possède son propre badge (PID 1000)

La Commande (Ouvrier) : Construit le mur. Il a son badge (PID 1001). Si le chef de chantier est viré, les ouvriers arrêtent de travailler et partent (sauf s'ils sont indépendants/nohup)

De la Commande au Processus : L'Instant de Création



1. Lancement
Vous tapez `ls`. Le Shell (Chef d'équipe) interprète.

2. Création (fork)
Le Shell demande au Noyau de créer un processus enfant.

3. Identification
Le Noyau alloue un **PID** unique et de la mémoire.

4. Exécution
Le processus exécute le code de `ls`.

5. Terminaison
Le processus se termine et rend la main au Shell.

Un **processus** est une instance d'un programme en cours d'exécution.

Point clé : Un processus est l'unité de travail fondamentale, gérée exclusivement par le Noyau.

Processus vs Job : Le Tableau Comparatif

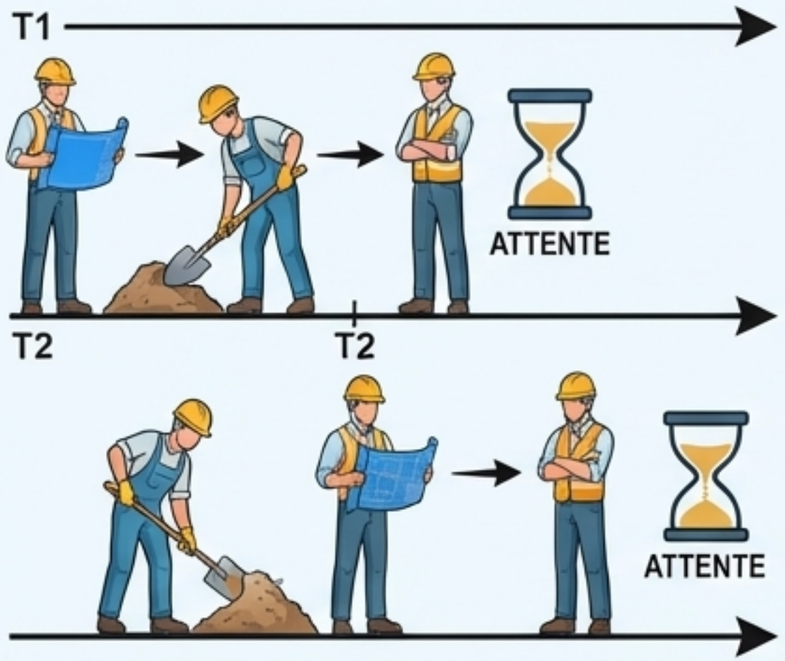
Caractéristique	PROCESSUS (Vue Noyau)	JOB (Vue Shell)
C'est quoi ?	Instance de programme (Ouvrier)	Tâche / Ligne de commande (Mission)
Identifiant	PID (Global, ex: 12450)	JOB ID (Local, ex: %1)
Gestionnaire	Le Noyau (Kernel)	Le Shell (Bash, Zsh)
Commande de gestion	`kill 12450`	`kill %1`
Portée	Tout le système	Fenêtre de terminal actuelle

Un **Job** est une abstraction du Shell pour gérer un ou plusieurs Processus.
Le Noyau, lui, ne connaît que les Processus.

Gestion des Jobs dans un Script Shell : Le Chef de Chantier et ses Ouvriers

CONCEPT CENTRAL : Le Chef (Script/Shell) et l'Équipe (Commandes/Processus Enfants)

1. EXÉCUTION SÉQUENTIELLE (Par défaut) : Un Job à la fois



Le Chef donne un ordre, attend la fin de la tâche, puis passe à la suivante. 1 Job actif (avant-plan).

2. EXÉCUTION PARALLÈLE (Avec &) : Plusieurs Jobs simultanés

Le Chef lance plusieurs ordres en arrière-plan (&) et ne les attend pas. Plusieurs Jobs tournent en même temps.

3. AVEC LES PIPES | : Un Job, Plusieurs Processus connectés

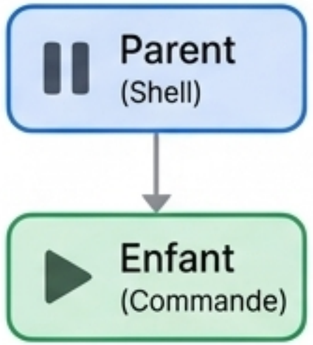
VUE DU SHELL : 1 SEUL JOB (Mission coordonnée)

VUE DU NOYAU : Plusieurs processus distincts connectés par un tuyau, travaillant en tandem.

Gérer les Tâches : Avant-plan, Arrière-plan et Jobs

1. Scénario Standard (Avant-plan)

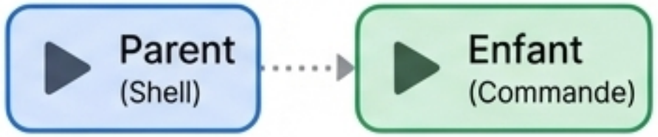
```
commande
```



Le Shell (Parent) est en pause et attend que la commande (Enfant) finisse. Vous ne pouvez rien taper d'autre.

2. Scénario Arrière-plan (&)

```
commande &
```

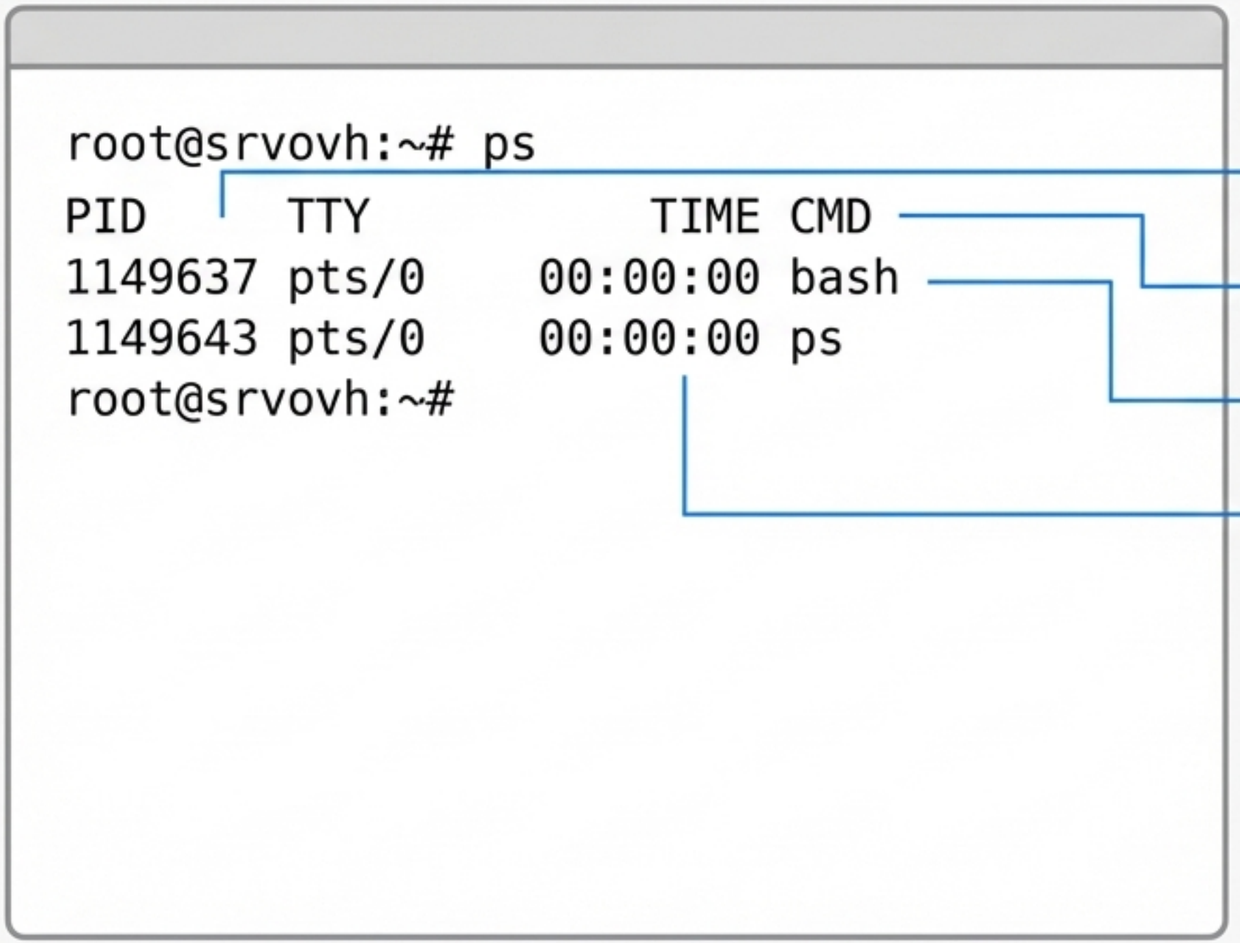


Le Shell lance la commande et vous rend la main immédiatement. La commande devient un Job d'arrière-plan.

3. Contrôle des Jobs

- `Ctrl+Z` : Suspend (met en pause) un job en cours d'exécution.
- `jobs` : Affiche la liste des jobs en cours dans le terminal.
- `fg %[num]` : Ramène un job en avant-plan.
- `bg %[num]` : Reprend un job suspendu en arrière-plan.

Observer les Ouvriers : La Commande `ps`



Analyse de la sortie

PID: L'identifiant unique du processus.

TTY: Le terminal utilisé (ici, `pts/0`).

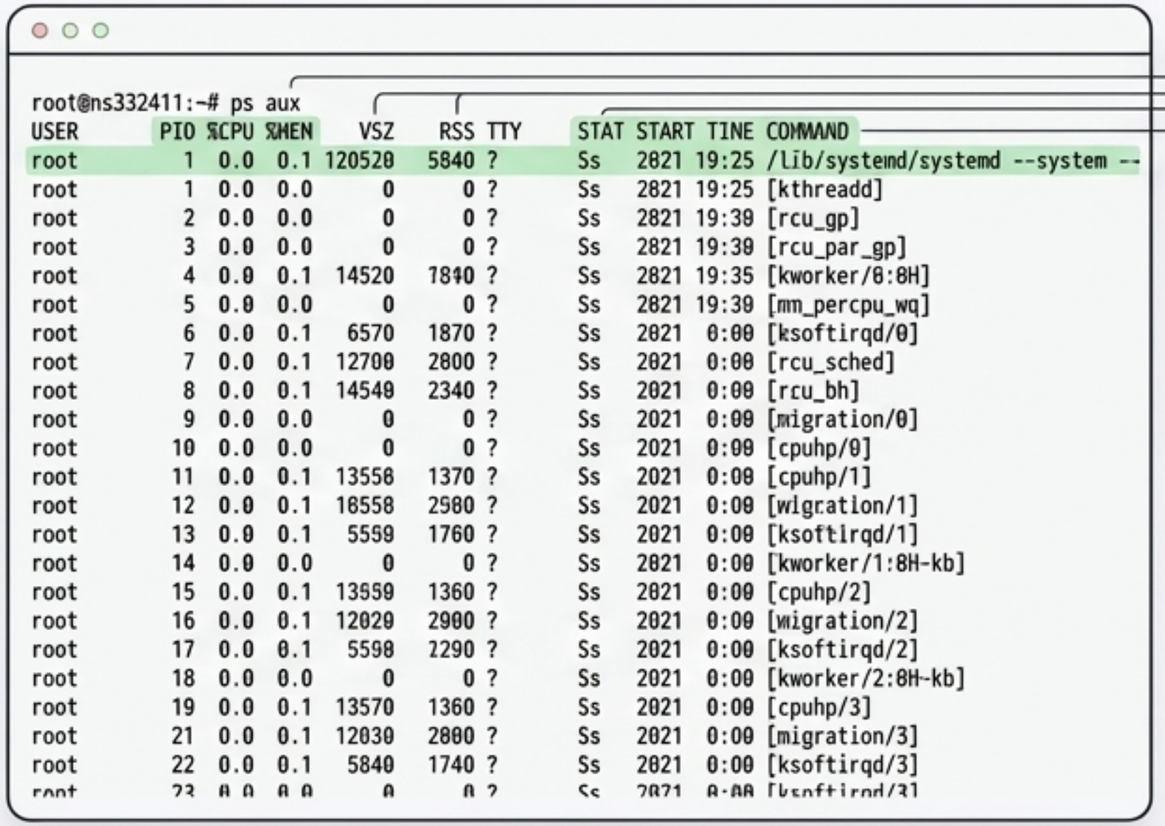
TIME: Le temps CPU total consommé.

CMD: Le nom de la commande.

👁️ **Observation**: Notez que le shell (`bash`) et la commande `ps` elle-même apparaissent comme des processus distincts.

La Vue d'Ensemble : `ps aux` pour tout voir

Pour une vue complète de tous les processus en cours sur le système (y compris ceux des autres utilisateurs et les services système), on utilise `ps aux`.



```
root@ns332411:~# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START TIME COMMAND
root         1  0.0  0.1 120520  5840 ?        Ss   2021 19:25 /lib/systemd/systemd --system --
root         1  0.0  0.0      0     0 ?        Ss   2021 19:25 [kthreadd]
root         2  0.0  0.0      0     0 ?        Ss   2021 19:39 [rcu_gp]
root         3  0.0  0.0      0     0 ?        Ss   2021 19:39 [rcu_par_gp]
root         4  0.0  0.1  14520  1840 ?        Ss   2021 19:35 [kworker/0:0H]
root         5  0.0  0.0      0     0 ?        Ss   2021 19:39 [mm_percpu_wq]
root         6  0.0  0.1   6570  1870 ?        Ss   2021  0:00 [ksoftirqd/0]
root         7  0.0  0.1  12700  2800 ?        Ss   2021  0:00 [rcu_sched]
root         8  0.0  0.1  14540  2340 ?        Ss   2021  0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        Ss   2021  0:00 [migration/0]
root        10  0.0  0.0      0     0 ?        Ss   2021  0:00 [cpuhp/0]
root        11  0.0  0.1  13550  1370 ?        Ss   2021  0:00 [cpuhp/1]
root        12  0.0  0.1  18550  2580 ?        Ss   2021  0:00 [migration/1]
root        13  0.0  0.1   5559  1760 ?        Ss   2021  0:00 [ksoftirqd/1]
root        14  0.0  0.0      0     0 ?        Ss   2021  0:00 [kworker/1:0H-kb]
root        15  0.0  0.1  13550  1360 ?        Ss   2021  0:00 [cpuhp/2]
root        16  0.0  0.1  12020  2900 ?        Ss   2021  0:00 [migration/2]
root        17  0.0  0.1   5590  2290 ?        Ss   2021  0:00 [ksoftirqd/2]
root        18  0.0  0.0      0     0 ?        Ss   2021  0:00 [kworker/2:0H-kb]
root        19  0.0  0.1  13570  1360 ?        Ss   2021  0:00 [cpuhp/3]
root        21  0.0  0.1  12030  2800 ?        Ss   2021  0:00 [migration/3]
root        22  0.0  0.1   5840  1740 ?        Ss   2021  0:00 [ksoftirqd/3]
```

Analyse des colonnes principales

- **USER**: L'utilisateur qui a lancé le processus.
- **PID**: L'identifiant du processus.
- **%CPU**: Utilisation du CPU.
- **%MEM**: Utilisation de la mémoire.
- **STAT**: L'état du processus (S: sleeping, R: running, Z: zombie...).
- **COMMAND**: La commande complète avec ses arguments.

💡 **Exemple avec PID 1:**: Le processus avec le PID 1 est toujours le premier processus démarré par le noyau, généralement `systemd` ou `init`. Il est le parent de tous les autres processus.

Le Contrôle Final : Signaux et Priorités

Envoyer des Signaux avec `kill`

La commande `kill` n'est pas seulement pour tuer. Elle envoie des **signaux** aux processus.

👋 `kill -SIGTERM [pid]`: Demande poliment au processus de se terminer.

⚡ `kill -SIGKILL [pid]` (ou `kill -9 [pid]`): Force la terminaison immédiate.

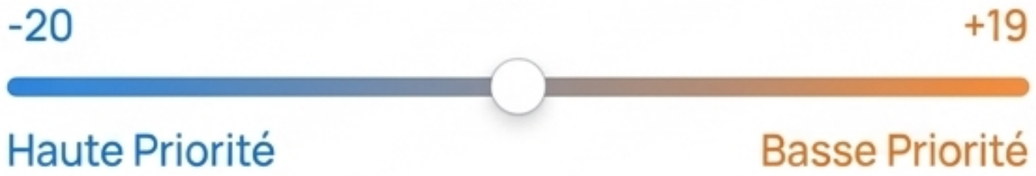
⏸️ `kill -SIGSTOP [pid]`: Met un processus en pause.

▶️ `kill -SIGCONT [pid]`: Reprend un processus en pause.

Utilisez `kill -l` pour lister tous les signaux disponibles.

Gérer la Priorité avec `nice` et `renice`

La 'niceness' d'un processus définit sa priorité d'accès au CPU.



`nice -n [valeur] commande`: Lance une commande avec une priorité spécifique.

`renice [valeur] -p [pid]`: Modifie la priorité d'un processus existant.

Le Voyage d'une Commande : La Synthèse

1 Utilisateur

Tape la commande `ls -l &`.

2 Terminal (Véhicule)

Capture le texte dans un PTS.

3 Shell (Pilote)

Interprète la commande et le `&`.

4 Noyau (Grand Patron)

Le Shell demande via `fork()` la création d'un **Processus** avec un PID unique.

5 Shell (Chef d'équipe)

Gère le processus comme un **Job** d'arrière-plan (ex: `%1`).

6 Processus (Ouvrier)

Exécute `ls -l`.

7 Résultat

La sortie est renvoyée au Shell, qui la transmet au Terminal pour affichage.

8 Main Libre

Le prompt est immédiatement disponible car le job est en arrière-plan.

De la simple touche à l'arrière-plan, chaque commande est une histoire complexe où Terminal, Shell, Noyau, Processus et Job jouent un rôle précis et essentiel.