GNU - Linux // Commande Indispensables

L’interpréteur

Alias : Console / Terminal / Shell …

```
user@lab-debian-48:~/dossierTest$ date
Mon Jan 12 04:00:47 PM CET 2026
```

- **Login** "Qui suis-je ?" C'est l'utilisateur connecté **:** user
- **Hostname** "Sur quelle machine ?" C'est le nom de l'ordinateur **:** lab-debian-48
- **Path** "Dans quel dossier je travaille ?" **:** ~/dossierTest
  - **Le Tilde (~)** : Le symbole ~ est un raccourci qui signifie "Mon répertoire personnel" (soit /home/user) C'est la "maison" de l’utilisateur
- **Privilege level :**
  - $ (Dollar) : Utilisateur standard
  - # (Dièse) : Root (Super-utilisateur)

Les Wildcard (Les Jokers)

C'est la fainéantise intelligente. Pourquoi taper tous les noms de fichiers quand on peut dire "tous ceux qui finissent par .txt" ?

* Remplace *n'importe quel caractères* ou *chaine de caractères*.

```
user@lab-debian-48:~$ rm *.jpg
```

supprimera toutes les images JPG d’un coup

? Remplace *un seul* caractère.

```
user@lab-debian-48:~$ ls fichier?.txt
```

trouvera fichier1.txt et fichierA.txt, mais pas fichier10.txt.

[a-z] toutes les lettres entre a et z

```
user@lab-debian-48:~$ ls
fichier1.txt  fichier2.txt  fichiera.txt  fichierb.txt  fichier.txt
user@lab-debian-48:~$ ls fichier[a-z].txt
fichiera.txt  fichierb.txt
```

[^a-z] toutes les lettres sauf celles entre a et z

```
user@lab-debian-48:~$ ls
fichier1.txt  fichier2.txt  fichiera.txt  fichierb.txt  fichier.txt
user@lab-debian-48:~$ ls fichier[^a-z].txt
fichier1.txt  fichier2.txt
```

Les flux (Les 3 Tuyaux Standards)

Sous Linux, imagine que chaque commande est une petite machine dans une usine. Pour fonctionner, cette machine a besoin de branchements. Par défaut, Linux lui connecte automatiquement 3 "tuyaux".

Chaque tuyau porte un numéro d'identification (File Descriptor) qu'il est important de connaître pour les redirections avancées.

stdin (Standard Input) - Le Tuyau d'Entrée

Identifiant : 0

Fonction : C'est l'alimentation de la machine. C'est par là qu'arrivent les données à traiter.

Par défaut : Il est branché à ton **Clavier**. La commande attend que tu tapes quelque chose.

L'exemple : Quand tu lances la commande cat sans rien d'autre, elle attend. Tout ce que tu tapes (stdin) est avalé par la machine.

stdout (Standard Output) - Le Tuyau de Sortie

Identifiant : 1

Fonction : C'est le tapis roulant où sort le produit fini. C'est le résultat "réussi" de la commande.

Par défaut : Il est branché à ton **Écran** (Terminal).

L'exemple : Quand la commande ls liste tes fichiers, elle envoie le texte dans ce tuyau pour qu'il s'affiche.

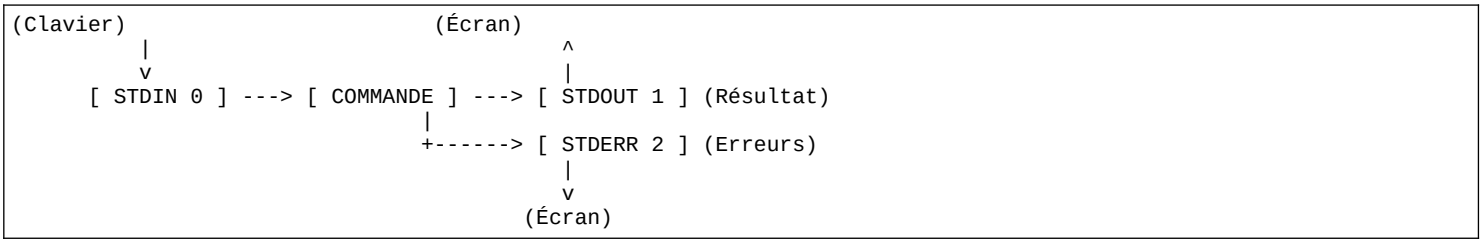
stderr (Standard Error) - Le Tuyau d'Erreur

Identifiant : 2

Fonction : C'est le tuyau d'évacuation des déchets ou l'alarme. Si la commande plante ou a un message d'avertissement à te donner, ça passe par là.

Par défaut : Il est **aussi** branché à ton **Écran**.

La subtilité cruciale : Même si stdout et stderr s'affichent tous les deux à l'écran par défaut, ce sont deux canaux totalement séparés ! C'est ce qui permet de rediriger les erreurs vers un fichier journal (error.log) tout en continuant de voir le résultat normal à l'écran.



pipe

C'est un tuyau qui connecte deux machines. Il Renvoie la sortir standard stdout d’un programme sur l’entrée standard stdin d’un autre programme.

command-0 | command-1

```
user@lab-debian-48:~$ ls | grep "fichier"
fichier.txt
```

je liste tout, mais je ne garde que ce qui contient "fichier"

man

La commande man est un utilitaire externe (un binaire) qui interface les pages de manuel stockées sur le système (généralement dans /usr/share/man sous forme compressée gzip).

- **Cible :** Tout ce qui est installé sur le système. Les binaires (ls, grep, nginx), les fichiers de configuration (/etc/ssh/sshd_config), les appels système (Syscalls), et les fonctions de librairies C.
- **Fonctionnement :** Elle invoque un *pager* (par défaut less) pour formater et afficher la documentation.
- **Exemple :** man ls ouvre la documentation complète de l'exécutable ls.

Lancer man man affiche la documentation de l'outil de visualisation de documentation. C'est là que l’on apprends à maîtriser l'outil, notamment la structure des **Sections**, essentielle pour ne pas lire la mauvaise doc.

Rappel des sections critiques pour un Admin (détaillées dans man man) :

- **Section 1 :** Commandes utilisateur / Binaires (ls, passwd).
- **Section 5 :** Formats des fichiers de configuration (Vital pour un AIS : man 5 sshd_config, man 5 interfaces).
- **Section 8 :** Commandes d'administration système (mount, fdisk, systemctl).

Exemple de conflit : passwd est à la fois une commande (sect 1) et un fichier (sect 5).

- man passwd (ou man 1 passwd) -> Ouvre la doc de la commande pour changer le mot de passe.
- man 5 passwd -> Ouvre la doc sur la structure du fichier /etc/passwd.

Affiche le manuel d'utilisation d'une commande (Manual) : man commande

```
user@lab-debian-48:~$ man man
```

Affiche le manual de man

apropos

La commande apropos permet de rechercher des pages de manuel en scannant **les noms** et **les courtes descriptions** (le champ NAME au début d'une man page) à partir d'un mot-clé ou d'une expression régulière (Regex).

Il est important de noter que apropos est strictement équivalent à l'option -k de man

La mise à jour de l'indexation se fait via la commande `sudo mandb`

Contrairement à grep qui chercherait en temps réel dans les fichiers bruts (ce qui serait lent), apropos interroge une base de données d'indexation locale : **la base de données whatis**.

- Cette base est générée et maintenue par l'utilitaire mandb.
- Sur la plupart des distributions modernes (Debian/RHEL), une tâche planifiée (cron ou systemd timer) met à jour cette base régulièrement pour indexer les nouvelles documentations installées.

help

La commande help est elle-même une commande interne au shell (généralement **Bash**). Elle sert à documenter les commandes qui sont intégrées directement dans l'interpréteur de commande et qui ne sont pas des fichiers exécutables présents sur le disque.

Nuance importante (--help) : Ne pas confondre la commande help avec l'argument --help (ex: ls --help). L'argument --help est une convention standard GNU pour qu'un binaire affiche son mode d'emploi succinct sur stdout. C'est le binaire lui-même qui gère cet affichage, pas le shell.

dpkg

Gestionnaire de paquets de bas niveau. **dpkg ne gère pas la résolution des dépendances**. Si le .deb a besoin d'une librairie non installée, l'installation échouera.

```
user@lab-debian-48:~$ sudo dpkg -i /home/user/Downloads/fichier.deb
```

-i **Install** Installe ou met à jour le paquet .deb spécifié. Le processus inclut le dépaquetage (unpacking) et la configuration.

-l **List** Liste les paquets installés. Utile pour vérifier la version exacte d'un paquet présent sur le système.

-r **Remove** Désinstalle le binaire mais garde les fichiers de configuration (dans /etc).

-P **Purge** Désinstalle tout, y compris la configuration.

À lancer impérativement après un échec de dpkg -i pour télécharger et installer les dépendances manquantes.

```
user@lab-debian-48:~$ sudo apt --fix-broken install
```

sudo

Permet à un utilisateur autorisé d'exécuter une commande avec les droits d'administrateur (root) (SuperUser DO) : sudo commande

```
user@lab-debian-48:~$ sudo apt update
```

si une commande répond Permission denied, c'est souvent qu'il faut mettre sudo devant.

echo

Affiche une ligne de texte à l'écran (stdout).

```
user@lab-debian-48:~$ echo Hello World
Hello World
```

who

C'est la commande de base (POSIX) pour lister les sessions actives.

Elles ne scannent pas les processus. Elles lisent un fichier binaire spécifique : **/var/run/utmp**.

Ce fichier est mis à jour en temps réel par les processus de login (login, sshd, etc.). *Note : Tu ne peux pas lire ce fichier avec cat ou nano car il est binaire. Tu dois utiliser les commandes who, w ou utmpdump.*

```
user@lab-debian-48:~$ who
seed     seat0        2025-12-17 07:30 (:0)
```

Petit script qui permet de savoir si root est connecter

```
if who | grep -wq "root"; then echo "ALERTE : L'utilisateur root est actuellement connecté !" else echo
"L'utilisateur root n'est pas en ligne." fi
```

w

Elle affiche qui est connecté **ET** ce qu'il est en train de faire (quelle commande il exécute), ainsi que la charge système (load average).

```
user@lab-debian-48:~$ w
 16:19:05 up  8:50,  1 user,  load average: 2.98, 2.90, 2.84
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
seed             -                07:30           0.00s   ?    lightdm --session-child 13 24
```

Les opérateurs de redirection

cmd > file

```
user@lab-debian-48:~$ echo Hello World > fichier.txt
```

Renvoie le résultat de la commande dans un fichier (ici fichier.txt)

Si le fichier n’existe pas, il est crée. Si le fichier existe, son contenu est effacé et remplacé

cmd < file

C'est l'inverse du fonctionnement habituel : au lieu que la commande lise ce que tu tapes au clavier, elle va "aspirer" le contenu d'un fichier.

```
user@lab-debian-48:~$ mysql -u utilisateur -p ma_base_de_donnees < sauvegarde.sql
```

Tu as un fichier sauvegarde.sql et tu veux l'envoyer dans le logiciel de base de données mysql.

cmd >> file

```
user@lab-debian-48:~$ echo Welcome to my world >> fichier.txt
```

Renvoie le résultat de la commande dans un fichier (ici fichier.txt)

Si le fichier n’existe pas, il est crée. Si le fichier existe, on écrit à la fin du fichier sans toucher à ce qu'il y a avant.

cmd 2>&1

Branche le tuyau 2 (Erreur) au même endroit que le tuyau 1 (résultat)", donc tout (résultat + erreurs) finit dans le fichier !

```
user@lab-debian-48:~$ ./backup.sh > journal_complet.txt 2>&1
```

"Linux, envoie le résultat normal (1) dans journal_complet.txt. Et ensuite, prends le tuyau d'erreur (2) et branche-le au même endroit que le tuyau (1)." Résultat : Succès et Erreurs sont mélangés dans le fichier texte.

Le "Trou Noir" (/dev/null)

C'est un concept vital sous Linux. /dev/null est un fichier spécial qui détruit tout ce qu'on lui envoie.

- **Usage :** Faire taire une commande bavarde ou ignorer les erreurs.
- **Exemple :** Tu cherches un fichier dans tout le système, mais tu ne veux pas voir les centaines de messages "Permission denied" pour les dossiers où tu n'as pas accès.

```
user@lab-debian-48:~$ find / -name "mon_fichier" 2> /dev/null
```

Traduction : "Cherche le fichier, et si tu as une erreur, jette-la à la poubelle."

tail

tail (queue) est un utilitaire qui affiche la fin d'un fichier sur la sortie standard (stdout). Par défaut, sans argument spécifique, elle affiche les **10 dernières lignes**.

Récupérer les dernières lignes (footer) : tail [option] file

```
user@lab-debian-48:~$ tail -1 fichier.txt
Hello World
```

-f **Follow** Le processus reste actif et attend que de nouvelles données soient écrites à la fin du fichier pour les afficher instantanément.

-F **Follow by name** Suit le **descripteur de fichier (inode)**, si le fichier subit une rotation, elle détectera la création du nouveau fichier et basculera dessus automatiquement.

-n **Nombre de lignes** Permet de spécifier le nombre de lignes à afficher, tail -n 50 ou tail -50 affichera les 50 dernieres lignes.

head

Récupérer les premières lignes (header) : head [option] file

```
user@lab-debian-48:~$ head -n 1 fichier.txt
Hello World
```

```
user@lab-debian-48:~$ head -c 3 fichier.txt
Hel
```

-n X X est un nombres, Affiche les X premières lignes

-c X X est un nombres, Affiche les X premières octets

pwd

Afficher le répertoire courant "Où suis-je ?" (Print Working Directory) :

```
user@lab-debian-48:~$ pwd
/home/user
```

grep

grep (Global Regular Expression Print) est un utilitaire de ligne de commande standard Unix utilisé pour rechercher des chaînes de caractères ou des motifs (patterns) dans du texte brut.

grep fonctionne comme un filtre. Il prend une entrée (soit un fichier, soit l'entrée standard stdin), analyse chaque ligne une par une, et imprime sur la sortie standard (stdout) **uniquement les lignes qui correspondent au motif demandé**.

```
user@lab-debian-48:~$ grep [OPTIONS] MOTIF [FICHIER...]
```

-i	**Insensible à la casse**	Ignorer maj/min lors d'une recherche (ex: chercher "error" ou "Error" dans des logs).
-v	**Inverser la recherche**	Afficher tout *sauf* le motif. Utile pour exclure des lignes (ex: grep -v "commentaires").
-r (ou -R)	**Récursif**	Chercher dans un dossier et ses sous-dossiers. Indispensable pour fouiller /etc/ ou /var/log/.
-n	**Numéro de ligne**	Affiche le numéro de la ligne dans le fichier. Permet d'aller éditer directement la ligne avec vim +n.
-E	**Extended Regex**	Active les expressions régulières étendues (ERE). Permet d'utiliser des métacaractères avancés comme `
-l	**Fichiers seulement**	Affiche uniquement le *nom des fichiers* contenant le motif, pas les lignes elles-mêmes.
-w	**Word**	Force la recherche du **mot exact**. Sans ça, si un utilisateur s'appelle "rootkit" ou "taproot", grep renverrait vrai, ce qui serait un faux positif.
-q	**Quiet**	on n'affiche rien, on veut juste le code de retour ($?) pour le if.

Quelques exemple :

Surveiller uniquement les erreurs 404 dans les logs Nginx

```
user@lab-debian-48:~$ tail -f /var/log/nginx/access.log | grep " 404 "
```

Lister les processus, filtrer pour ssh, et exclure la ligne du grep lui-même

```
user@lab-debian-48:~$ ps aux | grep sshd | grep -v grep
```

Cherche "192.168.1.50" dans tout le dossier /etc, affiche le numéro de ligne

```
user@lab-debian-48:~$ grep -rn "192.168.1.50" /etc/
```

Petit script pour savoir si root est présent dans le fichier passwd

```
if grep -q "root" /etc/passwd; then echo "L'utilisateur root est présent."; fi
```

cd

Se déplacer dans un répertoire (Change Directory) :

```
user@lab-debian-48:~$ cd dossierTest/
user@lab-debian-48:~/dossierTest$
```

```
user@lab-debian-48:~$ cd /home/user/dossierTest/
user@lab-debian-48:~/dossierTest$
```

On peut utiliser le chemin relatif ou absolu

- **Le Chemin Absolu (L'adresse GPS complète)**
  - **Règle :** Commence *toujours* par un slash /
  - **Exemple :** cd /home/user/dossierTest/
  - **Quand l'utiliser ?** Quand tu veux aller à un endroit précis peu importe où tu te trouves actuellement. C'est comme dire au GPS : *"Emmène-moi au 12 rue de la Paix, Paris"*.
- **Le Chemin Relatif (Le guidage local)**
  - **Règle :** Ne commence *jamais* par un slash. On part du dossier actuel.
  - **Exemple :** cd dossierTest/
  - **Quand l'utiliser ?** Quand le dossier est juste devant toi. C'est comme dire : *"Entre dans la pièce juste en face"*.

```
user@lab-debian-48:~/dossierTest$ cd
user@lab-debian-48:~$
```

```
user@lab-debian-48:~/dossierTest$ cd ~
user@lab-debian-48:~$
```

C'est le bouton "Maison". Il te ramène instantanément à ton répertoire personnel (/home/ton_user), peu importe où tu étais perdu dans le système.

```
user@lab-debian-48:~/dossierTest/dossierBidule$ cd ..
user@lab-debian-48:~/dossierTest$
```

Permet de revenir au dossier parent.

Astuce : on peux les enchaîner ! cd ../.. te fait remonter de deux étages.

```
user@lab-debian-48:~/dossierTest/dossierBidule$ cd -
/home/user/dossierTest
user@lab-debian-48:~/dossierTest$ cd -
/home/user/dossierTest/dossierBidule
user@lab-debian-48:~/dossierTest/dossierBidule$
```

Permet de revenir au dossier ou l’ont était juste avant, très utile quand ont fait des allers-retours !

ls

Lister les fichiers : ls [options] file-expression

```
user@lab-debian-48:~$ ls -lahin
total 36K
1569794 drwx------ 4 user user 4.0K Jan 12 16:12 .
1569793 drwxr-xr-x 3 root root 4.0K Jan 12 10:21 ..
1569804 -rw------- 1 user user 1.4K Jan 12 12:12 .bash_history
1569797 -rw-r--r-- 1 user user  220 Jan 12 10:21 .bash_logout
1569796 -rw-r--r-- 1 user user 3.5K Jan 12 10:21 .bashrc
1569798 drwxrwxr-x 2 user user 4.0K Jan 12 17:12 dossierTest
1569800 -rw-rw-r-- 1 user user   12 Jan 12 16:12 fichier.txt
1569801 drwxrwxr-x 3 user user 4.0K Jan 12 10:35 .local
1569795 -rw-r--r-- 1 user user  807 Jan 12 10:21 .profile
```

-l	long	Affiche les détails (permissions, propriétaire, taille ...)
-a	all	Affiche les fichiers cachés (ceux qui commencent par un point .)
-i	inode	Affiche le numéro d'identifiant unique du fichier à gauche du nom
-R	recursive	Liste le dossier courant, **puis** descend dans chaque sous-dossier pour le lister aussi, et ainsi de suite.
-h	human	Affiche la taille en Ko, Mo, Go plutôt qu'en octets
-t	time	Trie les fichiers par date de modification (le plus récent en premier).
-r	reverse	Inverse le sens du tri.
--help		

cp

Copier des fichiers et des répertoires (copy) : cp [options] source destination

```
user@lab-debian-48:~$ cp -rpu Documents/ /Backups/
```

copie le dossier Documents et tout ce qu’il contient dans le dossier /Backups

-r	recursive	Copie récursivement. C'est l'option obligatoire si on veux copier un répertoire et tout ce qu'il contient (sous-dossiers, fichiers).
-p	preserve	Copie le fichier en préservant ses attributs originaux : le propriétaire, le groupe, les permissions (droits) et surtout la date de modification.
-f	force	Force la copie. Si un fichier de destination existe déjà et ne peut pas être ouvert (par exemple s'il est protégé en écriture), cp va tenter de le supprimer pour le remplacer quand même.
-u	update	Copie uniquement si le fichier source est plus récent que le fichier de destination, ou si le fichier de destination n'existe pas.

mv

déplacer des fichiers (move) : mv [options] source destination

Attention, le comportement change selon ce que l'on mets en "Destination".

Déplacer : Si la destination est un **dossier** existant.

```
user@lab-debian-48:~$ mv fichier.txt /home/user/Documents/
```

Le fichier garde son nom mais change de place

Renommer : Si la destination est un **nom de fichier** (ou si on reste dans le même dossier).

```
user@lab-debian-48:~$ mv fichier.txt nouveau_nom.txt
```

Le fichier reste au même endroit mais change d'étiquette

-b	backup	Si un fichier du même nom existe déjà à la destination, mv ne l'écrase pas tout de suite. Il crée d'abord une copie de sauvegarde de l'ancien fichier (souvent en ajoutant un tilde ~ à la fin, ex: fichier.txt~)
-i	interactive	Demande une confirmation avant d'écraser un fichier existant
-f	force	Force le déplacement sans jamais demander confirmation, même si ça doit écraser des fichiers ou si les permissions semblent bloquantes

cat

Affiche le contenu d'un fichier (conCATener) : cat file

Si la plupart des gens l'utilisent pour lire un texte, sa véritable mission est de coller plusieurs fichiers ensemble pour n'en faire qu'un seul.

```
user@lab-debian-48:~$ cat fichier1.txt
```

-n	number	Affiche le contenu du fichier en ajoutant le numéro de ligne dans la marge gauche

le colle-tout (la vrai fonction de cat)

```
user@lab-debian-48:~$ cat fichier1.txt fichier2.txt > fichier-complet.txt
```

Prend le contenu de fichier1, colle celui de fichier2 juste après, et enregistre le tout dans fichier-complet.txt. C'est ça, la concaténation !

Le créateur rapide

```
user@lab-debian-48:~$ cat > fichier.txt
^C
```

si on tapes ça sans nom de fichier en entrée, cat attend que tu tapes au clavier (stdin)

On tapes notre texte (ou rien du tout), puis tu fais **CTRL + C** (pour dire "Fin de fichier")

touch

créer des fichiers (touch) : touch file

Littéralement, la commande signifie "Toucher". Son but premier n'est pas de créer, mais de **mettre à jour les dates** d'un fichier sans l'ouvrir.

L'usage quotidien : Créer du vide

C'est le lien parfait avec cat.

- **Avec cat > file** : Tu crées un fichier et tu écris du texte dedans immédiatement.
- **Avec touch file** : Tu crées un fichier **vide** instantanément.
    - *Usage :* Très utile pour créer un fichier "repère" ou pour préparer un fichier avant qu'un script n'écrive dedans.

```
user@lab-debian-48:~$ touch fichier.txt
```

La vraie fonction : "J'ai touché ce fichier"

Imaginez que l'on a un fichier rapport.txt datant de l'année dernière. On veut faire croire au système que l'on vient juste de travailler dessus (pour qu'il soit sauvegardé par le script de backup qui ne prend que les fichiers récents, par exemple).

- **Commande :** touch rapport.txt
- **Résultat :** Le contenu du fichier ne change pas (il n'est pas effacé !), mais sa **date de modification** devient "maintenant".
- *Analogie :* C'est comme composter un billet de train. Le billet est le même, mais il a maintenant une date récente.

Il existe une option qui permet de tricher avec le temps, souvent utilisée par les développeurs pour tester des programmes de tri par date.

-t	time	Permet de dater un fichier dans le passé ou le futur. `touch -t 202501011200 fichier.txt` (Le fichier sera daté du 1er Janvier 2025 à midi)

mkdir

créer des dossiers (Make Directory) : mkdir [options] directory

```
user@lab-debian-48:~$ mkdir -pv dossierTest/dossierBidule
mkdir: created directory 'dossierTest'
mkdir: created directory 'dossierTest/dossierBidule'
```

-p	parents	Créer la structure des répertoires parents si nécessaire
-v	verbose	Affiche un message pour chaque dossier créé

tar

Utilitaire d'archivage et de compression.

Créer une archive compressée (Pack)

```
user@lab-debian-48:~$ tar -czvf archive.tar.gz /dossier/source/
```

Extraire une archive (Unpack)

```
user@lab-debian-48:~$ tar -xzvf archive.tar.gz -C /dossier/destination/
```

-c **Create** Crée une nouvelle archive.

-x **Extract** Extrait le contenu d'une archive.

-v **Verbose** Mode verbeux. Affiche la liste des fichiers traités (essentiel pour vérifier que ça ne plante pas sur un gros backup).

-f **File** Indique le nom du fichier archive. **Attention :** Cette option doit toujours être la **dernière** de la liste des options (juste avant le nom du fichier). Ex: -cvf est bon, -cfv est faux.

-z **Gzip** Génère du .tar.gz. Rapide, compression standard. Le standard universel sous Linux.

-j **Bzip2** Génère du .tar.bz2. Plus lent, mais compresse mieux que gzip.

-J **Xz** Génère du .tar.xz. Très lent à compresser, mais taux de compression excellent. Devenu le standard pour les paquets (Debian, Arch).

-C **Change directory** Permet d'extraire l'archive **dans un dossier spécifique** sans avoir à se déplacer avec cd au préalable.

-t **List** Liste le contenu de l'archive **sans l'extraire**. Vital pour vérifier le contenu d'un backup avant de l'écraser sur la prod.

rm

Supprimer des fichiers (remove) : rm [options] file

Contrairement à Windows ou macOS, **il n'y a pas de Corbeille** en ligne de commande. Quand tu valides une commande rm, le fichier est instantanément supprimé de l'index du système de fichiers.

```
user@lab-debian-48:~$ rm -ri dossierTest/
rm: descend into directory 'dossierTest/'? y
rm: remove directory 'dossierTest/dossierBidule'? y
rm: remove directory 'dossierTest/'? y
```

-r	recursive	Permet de supprimer un répertoire **et tout ce qu'il contient** (sous-dossiers, fichiers cachés, etc.)
-i	interactive	Demande une confirmation pour *chaque* fichier avant de le supprimer
-f	force	Supprime sans poser de questions et sans afficher d'erreur si le fichier n'existe pas

ATTENTION : la commande rm accepte aussi les wildcards !

```
user@lab-debian-48:~$ rm -rf /*
```

 # Le combo nucléaire : rm -rf. -r pour récursif (les dossiers et leur contenu) et -f pour force (sans confirmation). À utiliser avec une extrême prudence !

rmdir

Supprimer des répertoire vide (remove directory) : rmdir [options] directory

Contrairement à rm -r (le bulldozer), rmdir refusera de supprimer le dossier s'il contient encore le moindre fichier (même caché).

```
user@lab-debian-48:~$ rmdir dossierTest/
```