Rapport Linux

I-Navigation dans l'arborescence

Chemins : un chemin indique , pour une application, l'emplacement d'un fichier dans l'arborescence

- **Chemin absolu** : Commence par / (la racine) et indique l'emplacement exact.
  Example : /home/user/docs.
- **Chemin relatif** : Dépend du répertoire courant.
  Example : docs/fichier.txt.

Commandes liées :

- **cd** : Change de répertoire.
  Exemple : cd /home/user.
- **ls** : Liste les fichiers d'un répertoire.
  Exemple : ls -l (Affiche des détails).

Variables liées:

- **$PATH** : Liste les répertoires où sont recherchées les commandes.
  Exemple : echo $PATH.

II-Piping

Le piping est une méthode puissante en ligne de commande pour transférer la sortie d'une commande comme entrée d'une autre. Cela permet de traiter et de transformer les données en plusieurs étapes, directement dans le terminal.

Utilisation du symbole | (pipe)

Le pipe redirige la **sortie standard (stdout)** d'une commande vers l'entrée **standard (stdin)** d'une autre commande.

Commande tee :

La commande **tee** est utilisée pour rediriger la sortie d'une commande à la fois vers un fichier et vers la sortie standard (le terminal).

Redirections avancées : Entrées et sorties

Les commandes en Bash utilisent trois flux principaux :

- **stdin (entrée standard, file descriptor 0)** : Reçoit des données (clavier, fichier, pipe).
- **stdout (sortie standard, file descriptor 1)** : Affiche les résultats (écran ou fichier).
- **stderr (erreur standard, file descriptor 2)** : Affiche les erreurs.

- **Pipe (|)** : Transfère la sortie d'une commande comme entrée d'une autre.
- **tee** : Duplique la sortie, l'envoyant à la fois dans un fichier et vers l'écran.
- **Redirections**:
    - > : Écrire dans un fichier (écrase).
    - >> : Ajouter à un fichier (sans écraser).
    - 2> : Capturer les erreurs dans un fichier.
    - 2>&1 : Combiner stdout et stderr.
    - < : Lire des données depuis un fichier.

III-File Globbing

Le **file globbing** est un mécanisme dans Linux qui permet d'utiliser des **wildcards** (caractères génériques) pour faire correspondre des noms de fichiers ou répertoires. En d'autres termes, cela permet de sélectionner des fichiers de manière flexible en utilisant des caractères spéciaux dans un terminal ou un script.

- ***** : Correspond à zéro ou plusieurs caractères.
  Exemple : *.txt → Correspond à tous les fichiers se terminant par .txt.

- **?** : Correspond à un seul caractère.
  Exemple : fichier?.txt → Correspond à fichier1.txt, fichierA.txt (pas fichier10.txt).
- **[]** : Correspond à un ensemble ou une plage.
  Exemple : fichier[1-3].txt → Correspond à fichier1.txt, fichier2.txt.
- **[!...]** : Négation.
  Exemple : fichier[!1-3].txt → Correspond à fichier4.txt.
- **{}** : Correspond à des options séparées par des virgules.
  Exemple : fichier{1,2,3}.txt → Correspond à fichier1.txt, fichier2.txt.

IV-Gestion des processus :

- **bg** : Exécute un processus en arrière-plan.
  Exemple : bg (Reprend le travail æ en arrière-plan).
- **fg** : Ramène un processus suspendu au premier plan.
  Exemple : fg (Ramène le travail au premier plan).
- **Ctrl+C** : Arrête un processus en cours d'exécution.
- **Ctrl+Z** : Suspend un processus en cours d'exécution. Utilisez fg pour le reprendre.

V-Permissions :

c**hmod**: Change les permissions d'un fichier.

Details des permissions:

- **r** : Lecture (permission de lire le fichier).
- **w** : Écriture (permission de modifier le fichier).
- **x** : Exécution (permission d'exécuter le fichier).

Les permissions sont attribuées à trois catégories d'utilisateurs :

1. **Utilisateur (u)** : Le propriétaire du fichier.
2. **Groupe (g)** : Les membres du groupe auquel appartient le fichier.
3. **Autres (o)** : Tous les autres utilisateurs.

Exemple : chmod u=rwx,g=rx,o=r fichier.txt (Définit les permissions rwxr-xr--).

chown : Change la propriété d'un fichier.
Exemple : chown utilisateur:groupe fichier.txt.

chgrp : Change le groupe d'un fichier.
Exemple : chgrp groupe fichier.txt

VI-Fichiers spéciaux :

- **/etc/shadow** : Contient les mots de passe cryptés des utilisateurs.
- **/dev/null** : Supprime toute donnée écrite.
  Exemple : commande > /dev/null.

VII-Chaînage de commandes :

Le chaînage permet d'exécuter plusieurs commandes dans une seule ligne.

- **;** : Exécute les commandes **indépendamment** de leur succès ou échec.
  Exemple : mkdir dossier; cd dossier
  - Crée un dossier puis change de répertoire.
- **&&** : Exécute la commande suivante **uniquement si la précédente réussit**.
  Exemple : mkdir dossier && cd dossier
  - Si le dossier est créé, alors on se déplace dedans.
- || : Exécute la commande suivante **uniquement si la précédente échoue**.
  Exemple : ls dossier || echo "Dossier introuvable"
  - Si le dossier n'existe pas, affiche un message d'erreur.

VIII-Scripting :

Un **script Bash** est un fichier contenant une suite de commandes exécutables. Cela permet d'automatiser des tâches répétitives.

- **Le shebang** : (#!) est une ligne spéciale placée au début d'un script pour indiquer au système quel interpréteur utiliser pour exécuter le fichier. Par exemple :

#!/bin/bash

Cela signifie que le script sera exécuté avec l'interpréteur **Bash**.

1. **Récupérer des données de l'utilisateur avec read :**
   - La commande **read** demande une entrée utilisateur et la stocke dans une variable.
     Exemple :
   - read -p "Entrez votre nom : " nom
     - Affiche : *Entrez votre nom :*
     - L'utilisateur entre un texte, qui est enregistré dans la variable nom.
2. **Conditions avec if :**
   - Les conditions permettent d'exécuter une commande seulement si une situation spécifique est vraie.
     Exemple :
   - Si un fichier existe, afficher un message :

```
if [ -f fichier.txt ]; then
   echo "Le fichier existe."
else
   echo "Le fichier n'existe pas."
fi
```

Explication :

- **[-f fichier.txt]** vérifie si fichier.txt existe.
- Si oui, affiche "Le fichier existe". Sinon, affiche "Le fichier n'existe pas".

3. **Boucles :**
    - Les boucles exécutent une commande plusieurs fois.
      Exemple :
    - Traiter tous les fichiers .txt d'un dossier :

      ```
      for fichier in *.txt; do
          echo "Traitement de $fichier"
      done
      ```

      Explication :

        - Parcourt tous les fichiers se terminant par .txt.
        - Pour chaque fichier, affiche un message indiquant son traitement.
4. **<u>Gestion des options avec getopts :</u>**
    - **getopts** est utilisé pour créer des scripts qui acceptent des options comme -a ou -b.
      Exemple :
    - Si ton script doit demander un nom et un âge :
        - Option -n pour le nom, et -a pour l'âge.
        - **getopts** permet de traiter ces options proprement.

    <u>Explication simplifiée :</u>

    - Si tu exécutes :

      ```
      ./script.sh -n Alice -a 25
      ```

        - La commande comprend que Alice est le nom et 25 est l'âge.
        - Le script peut utiliser ces informations sans poser de questions supplémentaires.

    <u>Rsync</u> : est un outil très puissant pour la gestion de fichiers entre répertoires locaux ou distants.
    Il offre des options de **synchronisation** efficaces grâce à son algorithme qui ne copie que les différences.

Il supporte la **compression** et peut fonctionner sur **SSH** pour des transferts sécurisés entre machines distantes.

Il permet également d'**exclure certains fichiers**, de faire des **simulations** avant d'effectuer des changements, et de **supprimer des fichiers** obsolètes de la destination.

rsync [options] source destination

Options courantes :

- **-a** : Archive, pour copier récursivement et préserver les attributs des fichiers (permissions, timestamps, etc.).
- **-v** : Verbose, affiche des détails pendant le transfert.
- **-z** : Compression des données pendant le transfert.
- **-r** : Copie récursive des répertoires.
- **-u** : Copie seulement les fichiers plus récents que ceux de la destination.
- **--delete** : Supprime les fichiers de destination qui n'existent plus dans la source.

Fait par : Seydina Oumar Keita