CH_5 : Commandes Linux essentielles

1. Redirections

Les redirections permettent de contrôler les flux d'entrée/sortie des commandes.

Redirection de la sortie standard (stdout) vers un fichier :

```
commande > fichier.txt
```

Exemple : `ls > fichier_listing.txt`

Redirection de la sortie d'erreur (stderr) vers un fichier :

```
commande 2> erreur.txt
```

Exemple : `grep motif_inexistant fichier.txt 2> erreurs.txt`

Redirection des deux flux (stdout et stderr) vers un même fichier :

```
commande >& fichier.txt
```

Exemple : `commande >& sortie_et_erreurs.txt`

Redirection de l'entrée standard (stdin) depuis un fichier :

```
commande < fichier.txt
```

Exemple : `cat < fichier.txt`

2. Pipe (|)

Le pipe permet de connecter la sortie d'une commande à l'entrée d'une autre commande.

```
commande1 | commande2
```

Exemple : `ls | grep '.txt$'`

3. Tri (sort)

La commande `sort` permet de trier les lignes d'un flux de données.

Options :

- `-k` : trier selon une colonne spécifique
- `-n` : tri numérique
- `-u` : supprimer les doublons

Exemple :

```
cat fichier.txt | sort -k3 -n
```

4. Découpage de colonnes (cut)

La commande `cut` permet d'extraire des colonnes spécifiques d'un flux de données.

Options :

- `-d` : spécifier le délimiteur de colonnes
- `-f` : spécifier les numéros de colonnes à extraire

Exemple :

```
cat fichier.txt | cut -d' ' -f1,3
```

5. Comptage (wc)

La commande `wc` permet de compter le nombre de caractères, mots et lignes d'un flux de données.

Options :

- `-l` : nombre de lignes
- `-c` : nombre de caractères
- `-w` : nombre de mots

Exemple :

```
cat fichier.txt | wc -l
```

6. Affichage de la tête et de la queue (head, tail)

- `head` : affiche les premières lignes d'un flux de données
- `tail` : affiche les dernières lignes d'un flux de données

Options :

- `-n` : spécifier le nombre de lignes à afficher
- `-c` : spécifier le nombre d'octets à afficher

Exemples :

```
head -n 5 fichier.txt
tail -c 10 fichier.txt
```

7. Recherche de fichiers (find, locate)

- `find` : recherche de fichiers selon des critères avancés
    - Exemple : `find ~ -name core`
- `locate` : recherche rapide de fichiers à partir d'une base de données

8. Recherche de motifs (grep)

La commande `grep` permet d'afficher les lignes contenant un motif donné, en utilisant les expressions régulières.

Exemple :

```
grep "Ahmed" fichier.txt
```

TP : 4

Exercice 1 : Que fait ce programme ? Le programme shell suivant vérifie si un utilisateur spécifié en argument est connecté au système :

```
#!/bin/sh
w=`who | grep $1`
if [ -z "$w" ]; then echo "$1 n'est pas connecté"; fi
```

Explications :

- La commande `who` affiche la liste des utilisateurs actuellement connectés au système.
- Le pipe `|` redirige la sortie de `who` vers la commande `grep` qui cherche la présence du nom d'utilisateur passé en argument `$1`.
- Si la variable `w` est vide (`-z "$w"`), cela signifie que l'utilisateur n'est pas connecté, donc on affiche un message en conséquence.

Exercice 2 : Compter les utilisateurs Bash Le script `bash_user` compte le nombre d'utilisateurs ayant Bash comme shell par défaut :

```
#!/bin/bash

nb_bash_users=$(grep -c /bin/bash /etc/passwd)

if [ $nb_bash_users -eq 0 ]; then
  echo "Il n'y a aucun utilisateur dont le login shell est bash."
else
  echo "Il y a $nb_bash_users utilisateurs de ce systeme dont le login shell est
bash"
fi
```

Explications :

- La commande `grep -c /bin/bash /etc/passwd` compte le nombre de lignes dans le fichier `/etc/passwd` qui contiennent `/bin/bash`, correspondant aux utilisateurs ayant Bash comme shell par défaut.
- Le résultat est stocké dans la variable `nb_bash_users`.
- On utilise une structure `if-then-else` pour afficher le message approprié selon que le nombre d'utilisateurs Bash soit 0 ou non.

Exercice 3 : Changement d'extension Le script `change_ext` renomme les fichiers d'une extension donnée vers une nouvelle extension :

```
#!/bin/bash

if [ "$#" -ne 2 ]; then
  echo "Usage: $0 <extension_actuelle> <nouvelle_extension>"
  exit 1
fi

for file in *$1; do
  new_name=$(basename "$file" $1)$2
  mv "$file" "$new_name"
done
```

Explications :

- On vérifie d'abord que le script a bien reçu deux arguments, sinon on affiche un message d'usage et on quitte.
- On utilise une boucle `for` pour parcourir tous les fichiers ayant l'extension actuelle `$1`.
- Pour chaque fichier, on utilise la commande `basename` pour extraire le préfixe du fichier, puis on concatène la nouvelle extension `$2`.
- Enfin, on renomme le fichier avec la commande `mv`.

Exercice 4 : Affichage de date Le script `horloge.sh` affiche la date et l'heure dans un format plus lisible :

```
#!/bin/bash

# Récupération des informations de date
current_date=$(date)
day=$(date +%A)
month=$(date +%B)
date_num=$(date +%d)
year=$(date +%Y)
time=$(date +"%H heures, %M minutes et %S secondes")

# Affichage formaté
echo "On est le $day $date_num $month $year"
echo "L'horloge indique $time"
```

Explications :

- On stocke les différentes composantes de la date (jour, mois, date numérique, année, heure) dans des variables en utilisant des commandes `date +` avec des formats spécifiques.

- Puis on affiche ces informations dans un format plus lisible en français.

Exercice 5 : Renommage interactif de fichiers Le script `rename_files.sh` permet de renommer interactivement les fichiers d'un répertoire :

```
#!/bin/bash

if [ "$#" -eq 0 ]; then
  dir="."
elif [ -d "$1" ]; then
  dir="$1"
else
  echo "Le répertoire '$1' n'existe pas ou n'est pas accessible."
  exit 1
fi

for file in "$dir"/*; do
  if [ -f "$file" ]; then
    filename=$(basename "$file")
    echo "Fichier actuel : $filename"
    read -p "Nouveau nom (ou Entrée pour passer) : " new_name
    if [ -n "$new_name" ]; then
      if [ -e "$dir/$new_name" ]; then
        read -p "Le fichier '$new_name' existe déjà. Voulez-vous l'écraser ? (o/n) "
confirm
        if [ "$confirm" == "o" ]; then
          mv "$file" "$dir/$new_name"
        else
          echo "Fichier '$filename' ignoré."
        fi
      else
        mv "$file" "$dir/$new_name"
      fi
    else
      echo "Fichier '$filename' ignoré."
    fi
  fi
done
```

Explications :

- On vérifie d'abord les arguments du script. S'il n'y a pas d'argument, on considère le répertoire courant. Sinon, on vérifie que le répertoire passé en argument existe et est accessible.
- On parcourt ensuite tous les fichiers du répertoire avec une boucle `for`.
- Pour chaque fichier, on affiche son nom actuel et on demande à l'utilisateur un nouveau nom.
- Si l'utilisateur saisit un nouveau nom, on vérifie s'il existe déjà. Dans ce cas, on demande confirmation avant de l'écraser.
- Enfin, on renomme le fichier avec la commande `mv`.