Job 07

```
man ls
```

La commande `man ls` ouvre le manuel de la commande `ls`, qui permet de lister les fichiers et répertoires dans un système Unix ou Linux. Elle fournit des informations détaillées sur les options disponibles.

```
ls -a
```

La commande `ls -a` affiche la liste de tous les fichiers et répertoires dans le répertoire actuel, y compris les fichiers cachés. Les fichiers cachés sont ceux dont le nom commence par un point. Habituellement, la commande `ls` n'affiche pas ces fichiers cachés par défaut, mais en utilisant l'option `-a`, elle les inclut dans la sortie.

- `-a` *: Option pour inclure tous les fichiers, y compris les fichiers cachés dont le nom commence par un point.*

```
ls -la
```

La commande `ls -la` affiche une liste détaillée de tous les fichiers et répertoires présents dans le répertoire actuel, y compris les fichiers cachés.

- `ls` *: Commande de base pour lister les fichiers et répertoires.*
- `-l` *: Option pour afficher les détails longs de chaque fichier ou répertoire, ce qui inclut des informations telles que les permissions, le propriétaire, le groupe, la taille, la date de modification et le nom du fichier.*
- `-a` *: Option pour inclure tous les fichiers, y compris les fichiers cachés dont le nom commence par un point.*

Job 08

```
cat nom_du_fichier
```

La commande `cat` est utilisée pour afficher le contenu d'un ou plusieurs fichiers texte directement dans le terminal.

```
head -n 10 /.bashrc
```

La commande `head -n 10 /.bashrc` affiche les dix premières lignes du fichier ".bashrc" situé dans le répertoire personnel de l'utilisateur.

- `head` *: La commande utilisée pour afficher les premières lignes d'un fichier.*
- `-n 10` *: L'option qui spécifie le nombre de lignes à afficher. Dans ce cas, "10" indique que nous voulons afficher les dix premières lignes.*
- `~/.bashrc` *: Le chemin vers le fichier* `.bashrc` *dans le répertoire personnel de l'utilisateur.*

```
tail -n 10 ~/.bashrc
```

La commande `tail -n 10/.bashrc` affiche les dix dernières lignes du fichier `.bashrc` *situé dans le répertoire personnel de l'utilisateur*

- `tail` *: La commande utilisée pour afficher les dernières lignes d'un fichier.*
- `-n 10` *: L'option qui spécifie le nombre de lignes à afficher. Dans ce cas, "10" indique que nous voulons afficher les dix dernières lignes.*
- `~/.bashrc` *: Le chemin vers le fichier* `.bashrc` *dans le répertoire personnel de l'utilisateur.*

```
head -n 20 ~/.bashrc
```

La commande `head -n 20 ~/.bashrc` affiche les vingt premières lignes du fichier `.bashrc` *situé dans le répertoire personnel de l'utilisateur.*

- `head`*: La commande utilisée pour afficher les premières lignes d'un fichier.*
- `-n 20`*: L'option qui spécifie le nombre de lignes à afficher. Dans ce cas, "20" indique que nous voulons afficher les vingt premières lignes.*

- `~/.bashrc`: Le chemin vers le fichier `.bashrc` dans le répertoire personnel de l'utilisateur.

```
tail -n 20 ~/.bashrc
```

La commande `tail -n 20 ~/.bashrc` affiche les vingt dernières lignes du fichier `.bashrc` situé dans le répertoire personnel de l'utilisateur.

- `tail`: La commande utilisée pour afficher les dernières lignes d'un fichier.
- `-n 20`: L'option qui spécifie le nombre de lignes à afficher. Dans ce cas, "20" indique que nous voulons afficher les vingt dernières lignes.
- `~/.bashrc`: Le chemin vers le fichier `.bashrc` dans le répertoire personnel de l'utilisateur.

Job 09

```
echo -e "User1\nUser2" > users.txt
```

La commande `echo -e "User1\nUser2" > users.txt` crée un fichier nommé `users.txt` et y écrit deux lignes de texte, "User1" et "User2", séparées par un retour à la ligne.

- `echo`: La commande utilisée pour afficher du texte.
- `-e`: L'option d'`echo` qui permet d'interpréter les séquences d'échappement dans le texte, telles que "\n" qui représente un retour à la ligne.
- `"User1\nUser2"`: Le texte que nous voulons afficher. "\n" indique à `echo` d'insérer un retour à la ligne entre "User1" et "User2".
- `>`: L'opérateur de redirection qui dirige la sortie de la commande `echo` vers un fichier.
- `users.txt`: Le nom du fichier dans lequel nous voulons écrire le texte.

```
groupadd Plateformeurs
```

La commande `groupadd Plateformeurs` crée un nouveau groupe nommé "Plateformeurs"

- `groupadd`: C'est la commande utilisée pour ajouter un nouveau groupe.
- `Plateformeurs`: C'est le nom du groupe que nous voulons créer. Dans ce cas, le groupe nouvellement créé sera appelé "Plateformeurs".

```
useradd User1
```

La commande `useradd User1` crée un nouvel utilisateur nommé "User1"

- `useradd`: C'est la commande utilisée pour ajouter un nouvel utilisateur.
- `User1`: C'est le nom de l'utilisateur que nous voulons créer. Dans ce cas, le nouvel utilisateur sera appelé "User1".

```
usermod -aG Plateformeurs User2
```

La commande `usermod -aG Plateformeurs User2` modifie les attributs d'un utilisateur existant nommé "User2" en ajoutant ce dernier au groupe "Plateformeurs".

- `usermod`: C'est la commande utilisée pour modifier les attributs d'un utilisateur existant.
- `-a`: Cette option indique à `usermod` d'ajouter l'utilisateur spécifié à un ou plusieurs groupes sans supprimer les affiliations existantes.
- `-G Plateformeurs`: Cette option spécifie le groupe auquel l'utilisateur doit être ajouté. Dans ce cas, l'utilisateur "User2" est ajouté au groupe "Plateformeurs".
- `User2`: C'est le nom de l'utilisateur auquel nous voulons ajouter le groupe "Plateformeurs".

```
cp users.txt droits.txt
```

La commande `cp users.txt droits.txt` copie le contenu du fichier `users.txt` vers un nouveau fichier nommé `droits.txt`.

- `cp`: C'est la commande utilisée pour copier des fichiers ou des répertoires.
- `users.txt`: C'est le nom du fichier source que nous voulons copier.
- `droits.txt`: C'est le nom du fichier de destination où nous voulons copier le contenu du fichier source.

```
cp users.txt groupes.txt
```

La commande `cp users.txt groupes.txt` copie le contenu du fichier `users.txt` vers un nouveau fichier nommé `groupes.txt`.

- *`cp`: C'est la commande utilisée pour copier des fichiers ou des répertoires.*
- *`users.txt`: C'est le nom du fichier source que nous voulons copier.*
- *`groupes.txt`: C'est le nom du fichier de destination où nous voulons copier le contenu du fichier source.*

```
chown User1 droits.txt
```

La commande `chown User1 droits.txt` modifie le propriétaire du fichier `droits.txt`, le désignant comme appartenant à l'utilisateur "User1".

- *`chown`: C'est la commande utilisée pour modifier le propriétaire et/ou le groupe d'un fichier ou d'un répertoire.*
- *`User1`: C'est le nom de l'utilisateur auquel nous voulons attribuer la propriété du fichier `droits.txt`.*
- *`droits.txt`: C'est le nom du fichier dont nous voulons modifier le propriétaire.*

```
chmod u=r,go= droits.txt
```

La commande `chmod u=r,go= droits.txt` modifie les permissions du fichier `droits.txt` pour accorder la permission de lecture uniquement au propriétaire du fichier, tandis que les permissions de lecture, écriture et exécution pour le groupe et les autres utilisateurs sont supprimées.

- *`chmod`: C'est la commande utilisée pour modifier les permissions d'un fichier ou d'un répertoire.*
- *`u=r,go=` : C'est l'argument utilisé avec `chmod` pour définir les permissions. Il se divise en deux parties :*
- *`u=r` : Cela spécifie que le propriétaire (user) du fichier (`u`) aura la permission de lecture (`r`).*
- *`go=` : Cela spécifie que le groupe (group) et les autres utilisateurs (others) (`go`) n'auront aucune permission (`=`).*
- *`droits.txt`: C'est le nom du fichier dont nous voulons modifier les permissions.*

```
chmod a=r groupes.txt
```

La commande `chmod a=r groupes.txt` définit les permissions du fichier `groupes.txt` de manière à accorder la permission de lecture à tous les utilisateurs : propriétaire, groupe et autres utilisateurs.

- *`chmod` : C'est la commande utilisée pour modifier les permissions d'un fichier ou d'un répertoire.*
- *`a=r` : Cela spécifie que tous les utilisateurs (propriétaire, groupe et autres) auront la permission de lecture (`r`).*
- *`groupes.txt` : C'est le nom du fichier dont nous voulons modifier les permissions.*

```
chmod g=rw groupes.txt
```

La commande `chmod g=rw groupes.txt` modifie les permissions du fichier `groupes.txt` de telle sorte que le groupe propriétaire du fichier puisse lire et écrire dans celui-ci, tandis que les autres utilisateurs (propriétaire et autres) n'auront pas de changement dans leurs permissions. Voici une explication détaillée :

- *`chmod` : C'est la commande utilisée pour modifier les permissions d'un fichier ou d'un répertoire.*
- *`g=rw` : Cela spécifie que le groupe propriétaire (g) du fichier `groupes.txt` aura la permission de lire (`r`) et d'écrire (`w`) dans celui-ci.*
- *`groupes.txt` : C'est le nom du fichier dont nous voulons modifier les permissions.*

Job 10

```
echo "Je suis votre fichier texte" > une_commande.txt
```

La commande `echo "Je suis votre fichier texte" > une_commande.txt` crée un fichier nommé `une_commande.txt` et y écrit la phrase "Je suis votre fichier texte".

- *`echo` : C'est la commande utilisée pour afficher du texte dans le terminal.*
- *`"Je suis votre fichier texte"` : C'est le texte que nous voulons afficher.*
- *`>` : C'est l'opérateur de redirection qui redirige la sortie de la commande `echo` vers un fichier.*

- `une_commande.txt`: C'est le nom du fichier dans lequel nous voulons écrire le texte.

```
wc -l /etc/apt/sources.list > nb_lignes.txt
```

La commande `wc -l /etc/apt/sources.list > nb_lignes.txt` compte le nombre de lignes dans le fichier `/etc/apt/sources.list` et redirige ce nombre vers le fichier `nb_lignes.txt`.

- `wc -l /etc/apt/sources.list` : C'est la commande `wc` utilisée pour compter le nombre de lignes (`-l`) dans le fichier `/etc/apt/sources.list`.
- `>` : C'est l'opérateur de redirection qui redirige la sortie de la commande précédente (le nombre de lignes dans le fichier) vers un fichier.
- `nb_lignes.txt` : C'est le nom du fichier dans lequel nous voulons écrire le nombre de lignes.

```
cat /etc/apt/sources.list > save_sources
```

La commande `cat /etc/apt/sources.list > save_sources` copie le contenu du fichier `/etc/apt/sources.list` dans un nouveau fichier nommé `save_sources`.

- `cat /etc/apt/sources.list` : La commande `cat` est utilisée pour afficher le contenu d'un fichier. Dans ce cas, nous affichons le contenu du fichier `/etc/apt/sources.list`.
- `>` : C'est l'opérateur de redirection qui redirige la sortie de la commande précédente (le contenu du fichier) vers un fichier.
- `save_sources` : C'est le nom du nouveau fichier dans lequel nous voulons écrire le contenu du fichier `/etc/apt/sources.list`.

```
find -name ".*" -exec grep -H "alias" {} +
```

La commande `find -name ".*" -exec grep -H "alias" {} +` recherche récursivement tous les fichiers cachés (commençant par un point) dans le répertoire actuel et ses sous-répertoires, puis exécute la commande `grep` pour rechercher le motif "alias" dans ces fichiers.

- *`find`: C'est la commande utilisée pour rechercher des fichiers et des répertoires dans une hiérarchie de répertoires.*
- *`-name "."`: C'est l'option de* find` qui spécifie que nous voulons rechercher des fichiers dont le nom commence par un point, c'est-à-dire des fichiers cachés.*
- *`-exec grep -H "alias" {} +`: C'est l'option `exec` de `find`, qui permet d'exécuter une commande (ici `grep -H "alias"`) sur les fichiers trouvés. L'option `-H` de `grep` est utilisée pour afficher le nom du fichier où la correspondance est trouvée. `{}` est remplacé par la liste des fichiers trouvés par `find`, et `+` indique à `find` de passer autant de fichiers que possible à `grep` en une seule fois.*

Job 11

```
apt install tree
```

La commande `apt install tree` installe le programme "tree" sur votre système via le gestionnaire de paquets APT (Advanced Package Tool) utilisé par les distributions Linux basées sur Debian, telles qu'Ubuntu.

- *`apt`: C'est la commande utilisée pour gérer les paquets logiciels sur les systèmes basés sur Debian, notamment pour installer, supprimer ou mettre à jour des logiciels.*
- *`install`: C'est l'option d'`apt` utilisée pour installer de nouveaux paquets.*
- *`tree`: C'est le nom du paquet que vous souhaitez installer. Le paquet "tree" est un utilitaire en ligne de commande qui affiche une représentation arborescente de la structure des répertoires sur votre système.*

```
tree / > tree.save &
```

La commande `tree / > tree.save &` crée un fichier nommé `tree.save` contenant la structure arborescente du répertoire racine (/) de votre système, généré par la commande `tree`, et exécute cette opération en arrière-plan.

- *`tree /`: C'est la commande `tree` qui affiche la structure arborescente du répertoire spécifié, dans ce cas le répertoire racine (/) de votre système.*
- *`> tree.save`: C'est l'opérateur de redirection qui redirige la sortie de la commande `tree` vers un fichier nommé `tree.save`. Cela crée un nouveau fichier `tree.save` ou écrase le fichier existant s'il en existe un.*

- `&`: C'est l'opérateur de tâche en arrière-plan qui permet à la commande de s'exécuter en arrière-plan, ce qui signifie que vous pouvez continuer à utiliser votre terminal sans attendre la fin de la génération du fichier.

```
ls | wc -l
```

La commande `ls | wc -l` compte le nombre de fichiers et de répertoires dans le répertoire actuel en utilisant une combinaison de commandes `ls` et `wc`.

- `ls` : La commande `ls` est utilisée pour lister les fichiers et les répertoires dans le répertoire actuel.
- `|` : C'est l'opérateur de tube (pipe) qui permet de rediriger la sortie (output) de la commande précédente vers l'entrée (input) de la commande suivante.
- `wc -l` : La commande `wc` est utilisée pour compter les lignes dans une entrée donnée. L'option `-l` spécifie à `wc` de compter le nombre de lignes.

```
sudo apt update && sudo apt upgrade
```

La commande `sudo apt update && sudo apt upgrade` est utilisée pour mettre à jour les informations sur les paquets disponibles (mise à jour de la liste des paquets) à l'aide de la commande `apt update`, puis pour effectuer la mise à jour réelle des paquets installés sur le système à l'aide de la commande `apt upgrade`.

- `sudo`: C'est la commande qui permet d'exécuter une autre commande avec les privilèges de superutilisateur (root).
- `apt update`: Cette partie de la commande met à jour la liste des paquets disponibles dans les dépôts de logiciels configurés sur le système. Cela permet de récupérer les informations les plus récentes sur les versions disponibles des paquets.
- `&&`: C'est l'opérateur logique "et" qui permet d'exécuter la commande suivante seulement si la commande précédente s'est terminée avec succès (c'est-à-dire sans erreur).
- `sudo apt upgrade`: Cette partie de la commande effectue la mise à jour réelle des paquets installés sur le système. Elle télécharge et installe les nouvelles versions des paquets disponibles si elles sont disponibles dans les dépôts.

Job 12

```
echo "echo I’m a script" > myfirstscript.sh
```

La commande `echo "echo I’m a script" > myfirstscript.sh` crée un fichier de script nommé `myfirstscript.sh` contenant une seule ligne de code qui imprime le message "I’m a script" dans le terminal.

- `echo "echo I’m a script"` *: Cela affiche la chaîne de caractères* `"echo I’m a script"` *dans le terminal. Cette chaîne de caractères contient une commande* `echo` *qui imprime le message "I’m a script".*
- `>` *: C'est l'opérateur de redirection qui redirige la sortie (la chaîne de caractères affichée par* `echo`*) vers un fichier.*
- `myfirstscript.sh` *: C'est le nom du fichier dans lequel nous voulons écrire la chaîne de caractères.*

Job 13

```
echo "sudo apt update && sudo apt upgrade -y" > myupdate.sh
```

La commande `echo "sudo apt update && sudo apt upgrade -y" > myupdate.sh` crée un fichier de script appelé `myupdate.sh` contenant la commande `sudo apt update && sudo apt upgrade -y`.

- `echo "sudo apt update && sudo apt upgrade -y"` *: Cela affiche la chaîne de caractères* `"sudo apt update && sudo apt upgrade -y"` *dans le terminal.*
- `>` *: C'est l'opérateur de redirection qui redirige la sortie (la chaîne de caractères affichée par* `echo`*) vers un fichier.*
- `myupdate.sh` *: C'est le nom du fichier dans lequel nous voulons écrire la chaîne de caractères.*

Job 14

```
nombre1=$1
nombre2=$2

somme=$((nombre1 + nombre2))

echo "$nombre1 + $nombre2 = $somme"
```

Ce script prend deux nombres en entrée lors de son exécution et calcule leur somme.

1. `nombre1=$1` : *Cette ligne attribue la valeur du premier argument passé au script (accessible via* `$1`*) à la variable* `nombre1`.
2. `nombre2=$2` : *Cette ligne attribue la valeur du deuxième argument passé au script (accessible via* `$2`*) à la variable* `nombre2`.
3. `somme=$((nombre1 + nombre2))` : *Cette ligne calcule la somme des deux nombres en utilisant l'opération* `$((...))`. *Les valeurs de* `nombre1` *et* `nombre2` *sont additionnées et le résultat est stocké dans la variable* `somme`.
4. `echo "$nombre1 + $nombre2 = $somme"` : *Cette ligne affiche la somme des deux nombres ainsi que les nombres eux-mêmes. Les valeurs de* `nombre1`, `nombre2` *et* `somme` *sont affichées à l'aide de la commande* `echo`.

Job 15

```
fichier_nouveau=$1
fichier_source=$2

cat "$fichier_source" > "$fichier_nouveau"
```

Ce script prend deux arguments en entrée lors de son exécution : le nom du fichier de destination et le nom du fichier source. Il copie le contenu du fichier source vers le fichier de destination.

1. `fichier_nouveau=$1` : *Cette ligne attribue le premier argument passé au script (accessible via* `$1`*) à la variable* `fichier_nouveau`, *qui représente le nom du fichier de destination.*
2. `fichier_source=$2` : *Cette ligne attribue le deuxième argument passé au script (accessible via* `$2`*) à la variable* `fichier_source`, *qui représente le nom du fichier source.*
3. `cat "$fichier_source" > "$fichier_nouveau"` : *Cette ligne utilise la commande* `cat` *pour lire le contenu du fichier source spécifié par la variable* `fichier_source`, *puis redirige la sortie (le contenu du fichier source) vers le fichier de destination spécifié par la variable* `fichier_nouveau`. *Cela a pour effet de créer un nouveau fichier (ou d'écraser un fichier existant) avec le contenu du fichier source.*

Job 16

```
if [ "$1" = "Hello" ]; then
        echo "Bonjour, je suis un script !"
elif [ "$1" = "Bye" ]; then
        echo "Au revoir et bonne journée !"
fi
```

Ce script vérifie si le premier argument passé est égal à "Hello" ou "Bye". S'il s'agit de "Hello", il affiche "Bonjour, je suis un script !", sinon s'il s'agit de "Bye", il affiche "Au revoir et bonne journée !".

Job 17

```
nombre1=$1
operateur=$2
nombre2=$3

case $operateur in
        +)
        resultat=$(($nombre1 + $nombre2))
        ;;
        /)
        resultat=$(($nombre1 / $nombre2))
        ;;
        x)
        resultat=$(($nombre1 * $nombre2))
        ;;
        -)
        resultat=$(($nombre1 - $nombre2))
        ;;
esac

echo "Résultat : $resultat"
```

Ce script prend trois arguments en entrée : deux nombres et un opérateur mathématique (+, -, , /). Il utilise une structure de contrôle `case` pour effectuer l'opération mathématique appropriée en fonction de l'opérateur spécifié et affiche le résultat.

Job 18

```
for i in {1..10}
do
        echo "Je suis un script qui arrive à faire une boucle $i"
done
```

Ce script utilise une boucle `for` pour répéter une action un nombre déterminé de fois.

- `for i in {1..10}` *: Cette ligne initialise une boucle* `for` *qui va parcourir une séquence de nombres de 1 à 10, stockant chaque nombre dans la variable* `i`*.*
- `do` *: Cette ligne marque le début du bloc de code à exécuter à chaque itération de la boucle.*
- `echo "Je suis un script qui arrive à faire une boucle $i"` *: Cette ligne affiche un message dans le terminal à chaque itération de la boucle. Le nombre actuel de l'itération est représenté par* `$i`*.*
- `done` *: Cette ligne marque la fin du bloc de code à exécuter à chaque itération de la boucle.*