Commandes de recherche avancée

création de cinq fichiers textes nommés “mon_texte.txt” contenant le texte suivant : « Que la force soit avec toi. ». Et repartir ces fichiers dans les répertoires suivants : “Bureau”, “Documents”, “Téléchargement”, “Vidéos” et “Images”

echo "Que la force soit avec toi." > Bureau/mon_texte.txt

echo "Que la force soit avec toi." > Documents/mon_texte.txt

echo "Que la force soit avec toi." > Téléchargement/mon_texte.txt

echo "Que la force soit avec toi." > Vidéos/mon_texte.txt

echo "Que la force soit avec toi." > Images/mon_texte.txt

localiser les cinq fichiers “mon_texte.txt” à l’aide de mot « « force »

grep -rl "force" ~/

- **grep**: **grep** est une commande qui est utilisée pour rechercher des expressions régulières dans des fichiers ou des flux de données. Elle est souvent utilisée pour filtrer des lignes de texte qui correspondent à un certain motif.
- **-r**: L'option **-r** (ou **--recursive**) indique à **grep** de rechercher récursivement dans les sous-répertoires. Cela signifie que **grep** va chercher non seulement dans le répertoire spécifié, mais également dans tous les sous-répertoires et leurs sous-répertoires, et ainsi de suite.
- **-l**: L'option **-l** (ou **--files-with-matches**) demande à **grep** de ne pas afficher les lignes contenant le motif, mais plutôt de lister seulement les noms des fichiers dans lesquels le motif est trouvé.
- **"force"**: C'est le motif que **grep** recherche. Dans ce cas, **grep** recherchera toutes les occurrences de la chaîne de caractères "force" dans les fichiers.
- **~/**: C'est le chemin d'accès au répertoire personnel de l'utilisateur actuel. Le tilde (**~**) est une abréviation pour le répertoire personnel de l'utilisateur, ce qui rend la commande rechercher dans tous les fichiers et sous-répertoires de ce répertoire.

Compression et décompression de fichiers

mkdir ~/Documents/Plateforme

#Créez un répertoire nommé "Plateforme" dans le dossier "Documents"

cp ~/mon_texte.txt ~/Documents/Plateforme

Dupliquation de fichier mon_texte.txt quatre fois dans le répertoire Plateforme

cd ~/Documents/Plateforme

cp mon_texte.txt mon_texte_2.txt

cp mon_texte.txt mon_texte_3.txt

cp mon_texte.txt mon_texte_4.txt

cp mon_texte.txt mon_texte_5.txt

tar -czvf Plateforme.tar.gz Plateforme # Utilisation de gzip pour la compression

tar -cjvf Plateforme.tar.bz2 Plateforme # Utilisation de bzip2 pour la compression

tar -cJvf Plateforme.tar.xz Plateforme # Utilisation de xz pour la compression

tar -xzvf Plateforme.tar.gz # Décompression avec gzip

tar -xjvf Plateforme.tar.bz2 # Décompression avec bzip2

tar -xJvf Plateforme.tar.xz # Décompression avec xz

fonction	Outil	Commande et Options	Exemple	Explication de la Commande
Compression Seulement	gzip	gzip fichier	gzip fichier.txt	gzip compressera le fichier spécifié en utilisant

				l'algorithme gzip.
	bzip2	bzip2 fichier	bzip2 fichier.txt	bzip2 compressera le fichier spécifié en utilisant l'algorithme bzip2.
	xz	xz fichier	xz fichier.txt	xz compressera le fichier spécifié en utilisant l'algorithme xz.
	lzma	lzma fichier	lzma fichier.txt	lzma compressera le fichier spécifié en utilisant l'algorithme lzma.
	zip (pour fichiers)	zip archive.zip fichier1 fichier2 ...	zip archive.zip fichier1 fichier2	zip créera une archive zip contenant les fichiers spécifiés.
Archivage Seulement	tar	tar -cvf archive.tar dossier	tar -cvf archive.tar dossier/	tar créera une archive contenant le dossier spécifié sans compression.
	cpio	`find .	cpio -o > archive.cpio`	`find .
Compression et Archivage	tar avec gzip	tar -cvzf archive.tar.gz dossier	tar -cvzf archive.tar.gz dossier/	tar créera une archive contenant le dossier spécifié, compressée avec gzip.
	tar avec bzip2	tar -cvjf archive.tar.bz2 dossier	tar -cvjf archive.tar.bz2 dossier/	tar créera une archive contenant le dossier spécifié, compressée avec bzip2.
	tar avec xz	tar -cvJf archive.tar.xz dossier	tar -cvJf archive.tar.xz dossier/	tar créera une archive contenant le dossier spécifié, compressée avec xz.

fichier désigne un fichier individuel à compresser.

dossier désigne un répertoire à archiver.

archive désigne le nom de l'archive résultante.

-c : crée une nouvelle archive.

-v : affiche les fichiers traités.

-f : spécifie le nom de l'archive.

-z : utilise gzip pour la compression avec tar.

-j : utilise bzip2 pour la compression avec tar.

-J : utilise xz pour la compression avec tar.

-r : récursivement ajoute des répertoires à l'archive avec zip.

*Voici un tableau présentant comment décompresser les différents formats d'archives sur Linux

Format de Compression	Outil	Commande et Options	Exemple	Explication de la Commande
gzip (.gz)	gzip	gzip -d fichier.gz	gzip -d fichier.gz	gzip -d décompresse le fichier spécifié en utilisant l'algorithme gzip.
bzip2 (.bz2)	bzip2	bzip2 -d fichier.bz2	bzip2 -d fichier.bz2	bzip2 -d décompresse le fichier spécifié en utilisant l'algorithme bzip2.
xz (.xz)	xz	xz -d fichier.xz	xz -d fichier.xz	xz -d décompresse le fichier spécifié en utilisant l'algorithme xz.
lzma (.lzma)	lzma	lzma -d fichier.lzma	lzma -d fichier.lzma	lzma -d décompresse le fichier spécifié en utilisant l'algorithme lzma.
zip (.zip)	unzip	unzip archive.zip	unzip archive.zip	unzip extrait le contenu de l'archive spécifiée.
tar.gz (.tar.gz)	tar avec gzip	tar -xvzf archive.tar.gz	tar -xvzf archive.tar.gz	tar -xvzf extrait le contenu de l'archive tar.gz avec affichage des

				fichiers traités.
tar.bz2 (.tar.bz2)	tar avec bzip2	tar -xvjf archive.tar.bz2	tar -xvjf archive.tar.bz2	tar -xvjf extrait le contenu de l'archive tar.bz2 avec affichage des fichiers traités.
tar.xz (.tar.xz)	tar avec xz	tar -xvJf archive.tar.xz	tar -xvJf archive.tar.xz	tar -xvJf extrait le contenu de l'archive tar.xz avec affichage des fichiers traités.

fichier désigne un fichier à décompresser.

archive désigne une archive à décompresser.

-d : décompresse le fichier spécifié.

-x : extrait le contenu de l'archive.

-v : affiche les fichiers traités lors de l'extraction.

-f : spécifie le nom du fichier ou de l'archive à décompresser.

*Pour décompresser des dossiers, on utilise généralement les mêmes commandes que pour les fichiers individuels, mais on spécifie le nom du dossier à décompresser.

Format de Compression	Outil	Commande et Options	Exemple
gzip (.gz)	gzip	gzip -d dossier.gz	gzip -d dossier.gz
bzip2 (.bz2)	bzip2	bzip2 -d dossier.bz2	bzip2 -d dossier.bz2
xz (.xz)	xz	xz -d dossier.xz	xz -d dossier.xz
lzma (.lzma)	lzma	lzma -d dossier.lzma	lzma -d dossier.lzma

zip (.zip)	unzip	unzip archive.zip -d dossier	unzip archive.zip -d dossier
tar.gz (.tar.gz)	tar avec gzip	tar -xvzf archive.tar.gz -C dossier	tar -xvzf archive.tar.gz -C dossier
tar.bz2 (.tar.bz2)	tar avec bzip2	tar -xvjf archive.tar.bz2 -C dossier	tar -xvjf archive.tar.bz2 -C dossier
tar.xz (.tar.xz)	tar avec xz	tar -xvJf archive.tar.xz -C dossier	tar -xvJf archive.tar.xz -C dossier

Manipulation de texte

```
import csv


# Données à ajouter au fichier CSV
data = [
    ["Jean", 25, "Paris"],
    ["Marie", 30, "Lyon"],
    ["Pierre", 22, "Marseille"],
    ["Sophie", 35, "Toulouse"]
]


# Nom du fichier CSV
csv_file = "personnes.csv"


# Écriture des données dans le fichier CSV
with open(csv_file, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Nom", "Âge", "Ville"])
    writer.writerows(data)


print("Fichier CSV créé avec succès:", csv_file)
```

Ce code script Python écrit des données dans un fichier CSV en utilisant le module **csv**. Voici une explication ligne par ligne :

1. **import csv**: Cette ligne importe le module CSV de la bibliothèque standard Python, qui fournit des fonctionnalités pour lire et écrire des fichiers CSV.
2. **data = [...]**: Cette ligne crée une liste de listes contenant les données à écrire dans le fichier CSV. Chaque sous-liste représente une ligne de données, avec trois éléments : le nom, l'âge et la ville d'une personne.
3. **csv_file = "personnes.csv"**: Cette ligne définit le nom du fichier CSV dans lequel les données seront écrites. Dans cet exemple, le fichier s'appelle "personnes.csv".
4. **with open(csv_file, mode='w', newline='') as file:**: Cette ligne ouvre le fichier CSV en mode écriture ('w') et crée un contexte de gestion de fichier avec la syntaxe **with**. Le paramètre **newline=''** est utilisé pour spécifier qu'aucune ligne supplémentaire ne doit être ajoutée entre les lignes écrites dans le fichier CSV. Le fichier est ouvert dans ce contexte sous le nom **file**.
5. **writer = csv.writer(file)**: Cette ligne crée un objet **writer** à l'aide de la méthode **csv.writer()**, qui permet d'écrire dans le fichier CSV ouvert précédemment.
6. **writer.writerow(["Nom", "Âge", "Ville"])**: Cette ligne écrit une seule ligne d'en-tête dans le fichier CSV, avec les libellés des colonnes.
7. **writer.writerows(data)**: Cette ligne écrit toutes les lignes de données contenues dans la liste **data** dans le fichier CSV.
8. **print("Fichier CSV créé avec succès:", csv_file)**: Cette ligne affiche un message indiquant que le fichier CSV a été créé avec succès, suivi du nom du fichier. Cela confirme que l'écriture des données dans le fichier CSV s'est déroulée sans erreur.

#Les commandes

awk -F, '{print $3}' personnes.csv

- **awk** est un utilitaire de ligne de commande qui est souvent utilisé pour traiter des données textuelles, notamment pour filtrer et manipuler des fichiers tabulaires comme les fichiers CSV.
- **-F,** spécifie à **awk** que le délimiteur de champ dans le fichier est la virgule (**,**). Cela signifie que **awk** va diviser chaque ligne du fichier en champs en utilisant la virgule comme séparateur.
- **'{print $3}'** est un motif qui dit à **awk** d'imprimer le troisième champ de chaque ligne.

python3 create_csv.py

python3 est l'interpréteur Python 3 utilisé pour exécuter des scripts Python en ligne de commande.

Gestion des processus

Commande	Description
ps aux	Affiche tous les processus en cours d'exécution sur le système avec des informations détaillées.
`ps -e -o pid,state,cmd	grep " R "`	
`ps -e -o pid,state,cmd	grep " D "`	
top	Affiche en temps réel les informations sur l'utilisation des ressources système et les processus.
kill 12345	Envoie un signal SIGTERM au processus avec le PID (identifiant de processus) 12345 pour demander sa terminaison.
kill -9 12345	Envoie un signal SIGKILL au processus avec le PID 12345 pour forcer sa terminaison immédiate, sans possibilité de traitement.

- **ps -e -o pid,state,cmd** : Cette partie de la commande **ps** permet d'afficher la liste de tous les processus en cours d'exécution (**-e**), mais seulement avec les colonnes spécifiées, qui sont l'identifiant du processus (PID), l'état du processus et la commande exécutée (**-o pid,state,cmd**).
- **grep " R "** : Cette partie de la commande **grep** permet de filtrer les lignes de sortie de **ps** pour ne montrer que celles contenant le caractère " R ", indiquant ainsi les processus en cours d'exécution.
- **grep " D "** : De même, cette partie de la commande **grep** filtre les lignes de sortie de **ps** pour ne montrer que celles contenant le caractère " D ", qui indique les processus en attente bloquée.

En combinant **ps** avec **grep**, ces commandes permettent de filtrer les processus en fonction de leur état, ce qui peut être utile pour la surveillance et le dépannage des systèmes.

#les différences entre le processus de terminaison normale et celui effectué de manière forcée.

SIGTERM et **SIGKILL** sont deux signaux utilisés dans les systèmes Unix/Linux pour contrôler les processus.

1. **SIGTERM (Signal Terminate)** :

- Ce signal est envoyé à un processus pour lui demander de se terminer. C'est une façon polie de demander à un processus de s'arrêter, car cela permet au processus de terminer ses activités en cours et de se fermer proprement.
- Lorsqu'un processus reçoit le signal SIGTERM, il a la possibilité de capturer ce signal et de gérer sa propre fermeture. En d'autres termes, il peut effectuer des actions de nettoyage ou d'arrêt appropriées avant de se terminer.
- Le signal SIGTERM est souvent utilisé comme première option pour arrêter un processus de manière normale et propre.

2. **SIGKILL (Signal Kill)** :
    - Ce signal est beaucoup plus radical que SIGTERM. Il est utilisé pour forcer immédiatement la terminaison d'un processus sans lui donner la possibilité de se fermer correctement.
    - Lorsqu'un processus reçoit le signal SIGKILL, il est arrêté immédiatement, sans avoir la possibilité de gérer proprement sa propre fermeture. Cela peut entraîner une perte de données ou d'autres problèmes, car le processus n'a pas l'occasion d'effectuer des opérations de nettoyage ou de libération de ressources.
    - Le signal SIGKILL est généralement utilisé en dernier recours, lorsque le processus ne répond pas à un signal SIGTERM ou lorsque la terminaison immédiate du processus est nécessaire, par exemple en cas d'urgence ou lorsque le processus est devenu incontrôlable.

Surveillance des ressources système

Ressource à surveiller	Commande de surveillance	Explication
Utilisation de la CPU	top -b -n 1 | awk 'NR>7 {print $1,$9}' | column -t > cpu_usage.csv	Cette commande utilise top pour afficher les processus actifs et leur utilisation de la CPU en pourcentage. awk est utilisé pour extraire les colonnes pertinentes et column est utilisé pour formater les résultats en tableau.
Utilisation de la mémoire	top -b -n 1 | awk 'NR>7 {print $1,$10}' | column -t > memory_usage.csv	Cette commande utilise top pour afficher les processus actifs et leur utilisation de la mémoire en pourcentage. awk est utilisé pour extraire les colonnes pertinentes et column est utilisé pour formater les résultats en tableau.
Surveillance du trafic réseau	sudo iftop -t -s 1 -n -N -o 2s | awk 'NR>2 {print $1,$3,$5}' | column -t > network_usage.csv	Cette commande utilise iftop pour surveiller le trafic réseau en temps réel. Les informations sur les connexions réseau sont extraites à l'aide de awk, puis formatées en tableau avec column et enregistrées dans un fichier CSV.
Utilisation du disque	df -h | sed 's/ \+/,/g' > disk_usage.csv	Cette commande utilise df pour afficher l'utilisation du disque, puis sed pour remplacer les espaces par des virgules afin de formater les résultats en CSV.
Utilisation de la bande passante	sudo nload	Cette commande utilise nload pour surveiller l'utilisation de la bande passante en temps réel.
Utilisation des E/S	sudo iotop	Cette commande utilise iotop pour surveiller les opérations d'entrée/sortie (E/S) en temps réel.
Utilisation de la mémoire virtuelle	vmstat 1	Cette commande utilise vmstat pour afficher des informations sur

		l'utilisation de la mémoire virtuelle, y compris la mémoire libre, utilisée et inutilisée, ainsi que la taille de la mémoire swap. 1 indique l'intervalle de rafraîchissement en secondes.
Utilisation de la mémoire tampon et de la mémoire cache	free -h | sed 's/ \+/,/g' > memory_buffer_cache_usage.csv	Cette commande utilise free pour afficher les informations sur l'utilisation de la mémoire, puis sed pour formater les résultats en CSV en remplaçant les espaces par des virgules.
Charge du système	uptime	Cette commande utilise uptime pour afficher la charge du système, y compris le temps écoulé depuis le démarrage, le nombre d'utilisateurs connectés et la charge moyenne sur les dernières 1, 5 et 15 minutes.

Scripting avancé

```
#!/bin/bash

# Chemin du dossier à sauvegarder
source_dir="/home/alaa/Documents/Plateforme"

# Chemin du dossier où sauvegarder les fichiers de sauvegarde
backup_dir="/home/alaa/Documents/"

# Nom du fichier de sauvegarde
backup_file="backup_$(date +'%Y-%m-%d_%H-%M-%S').tar.gz"

# Créer le dossier de sauvegarde s'il n'existe pas
mkdir -p "$backup_dir"

# Créer une archive compressée de la sauvegarde
tar -czf "$backup_dir/$backup_file" "$source_dir"

# Afficher un message de confirmation
echo "Sauvegarde effectuée avec succès dans $backup_dir/$backup_file"
```

#les commandes

Commande	Description
crontab -e	Cette commande permet de modifier (ou de créer si inexistant) la table de tâches cron de l'utilisateur.
* * * * * /home/alaa/documents/script.sh	Cette ligne représente une entrée dans la table cron. Elle spécifie la fréquence à laquelle une tâche doit être exécutée et le chemin complet du script à exécuter. Plus précisément : * signifie "tous les" pour chaque champ de temps. - Les cinq astérisques représentent respectivement : minute (0-59), heure (0-23), jour du mois (1-31), mois (1-12) et jour de la semaine (0-6, où 0 représente le dimanche). /home/alaa/documents/script.sh est le chemin complet du script à exécuter.

Automatisation des mises à jour logicielles

```
#!/bin/bash

# Fonction pour afficher le menu principal
afficher_menu() {
    clear
    echo "1. Rechercher les mises à jour disponibles"
    echo "2. Mettre à jour tous les logiciels"
    echo "3. Quitter"
}

# Fonction pour rechercher les mises à jour disponibles
rechercher_mises_a_jour() {
    echo "Recherche des mises à jour disponibles..."
    sudo apt update
}

# Fonction pour mettre à jour tous les logiciels
mettre_a_jour_logiciels() {
    echo "Mise à jour de tous les logiciels..."
    sudo apt upgrade -y
}
```

```
# Boucle principale du script
while true; do
    afficher_menu

    read -p "Veuillez sélectionner une option : " choix

    case $choix in
        1)
            rechercher_mises_a_jour
            read -p "Appuyez sur Entrée pour revenir au menu principal..."
            ;;
        2)
            mettre_a_jour_logiciels
            read -p "Appuyez sur Entrée pour revenir au menu principal..."
            ;;
        3)
            echo "Au revoir!"
            exit 0
            ;;
        *)
            echo "Option invalide. Veuillez sélectionner une option valide."
            read -p "Appuyez sur Entrée pour continuer..."
            ;;
    esac
done
```

Ce script est un script Bash qui crée un menu interactif pour effectuer des opérations de gestion de logiciels sur un système Linux utilisant le gestionnaire de paquets **apt**.

1. **#!/bin/bash**: Cette ligne indique que le script doit être interprété par l'interpréteur de commandes Bash.
2. **afficher_menu()**: Cette fonction affiche le menu principal avec trois options : rechercher les mises à jour disponibles, mettre à jour tous les logiciels et quitter.
3. **rechercher_mises_a_jour()**: Cette fonction effectue une mise à jour des informations sur les paquets disponibles à partir des dépôts de logiciels configurés sur le système à l'aide de la commande **sudo apt update**.
4. **mettre_a_jour_logiciels()**: Cette fonction met à jour tous les logiciels du système à l'aide de la commande **sudo apt upgrade -y**, où l'option **-y** répond automatiquement "oui" à toutes les questions.
5. La boucle **while true; do ... done** crée une boucle infinie qui affiche le menu principal, lit l'entrée de l'utilisateur, puis exécute l'action correspondante en fonction de l'option choisie.
6. **read -p "Veuillez sélectionner une option : " choix**: Cette ligne demande à l'utilisateur de sélectionner une option en entrant un nombre.
7. **case $choix in ... esac**: Cette structure **case** évalue la variable **choix** et exécute le bloc de code correspondant à l'option sélectionnée par l'utilisateur.
8. Les cas 1 et 2 appellent respectivement les fonctions **rechercher_mises_a_jour** et **mettre_a_jour_logiciels**, puis demandent à l'utilisateur d'appuyer sur Entrée pour revenir au menu principal.
9. Le cas 3 affiche un message de sortie et quitte le script avec un code de sortie 0.
10. Le cas par défaut (*) gère les options invalides en affichant un message d'erreur et en demandant à l'utilisateur d'appuyer sur Entrée pour continuer.

Gestion des dépendances logicielles

```
#!/bin/bash
# Fonction pour installer Apache ou Nginx
install_web_server() {
    read -p "Choisissez le serveur web à installer (Apache/Apache2 ou Nginx): " server_choice
    case "$server_choice" in
        Apache|Apache2)
            sudo apt-get update
            sudo apt-get install apache2
            ;;
        Nginx)
            sudo apt-get update
            sudo apt-get install nginx
            ;;
        *)
            echo "Choix invalide. Veuillez choisir Apache/Apache2 ou Nginx."
            install_web_server
            ;;
    esac
}

# Fonction pour installer phpMyAdmin
install_phpmyadmin() {
    sudo apt-get update
    sudo apt-get install phpmyadmin
}
```

```
# Fonction pour installer MySQL ou MariaDB
install_database_server() {
    read -p "Choisissez le système de gestion de base de données (MySQL ou MariaDB): " db_choice
    case "$db_choice" in
        MySQL)
            sudo apt-get update
            sudo apt-get install mysql-server
            ;;
        MariaDB)
            sudo apt-get update
            sudo apt-get install mariadb-server
            ;;
        *)
            echo "Choix invalide. Veuillez choisir MySQL ou MariaDB."
            install_database_server
            ;;
    esac
}

# Fonction pour installer Node.js et npm
install_nodejs_npm() {
    sudo apt-get update
    sudo apt-get install nodejs
    sudo apt-get install npm
}
# Fonction pour installer Git
install_git() {
    sudo apt-get update
```

```
    sudo apt-get install git
}

# Installation des composants
install_web_server
install_phpmyadmin
install_database_server
install_nodejs_npm
install_git
echo "Installation terminée."
```

une explication du script :

1. **#!/bin/bash**: C'est un shebang, une instruction spécifiant l'interpréteur à utiliser pour exécuter le script, dans ce cas-ci, il indique que le script doit être exécuté en utilisant Bash.
2. **install_web_server() {**: Définition d'une fonction nommée **install_web_server**. Les fonctions sont des blocs de code qui peuvent être appelés à partir d'autres parties du script.
3. **read -p "Choisissez le serveur web à installer (Apache/Apache2 ou Nginx): " server_choice**: Affiche un message à l'utilisateur et attend une entrée. L'entrée de l'utilisateur est stockée dans la variable **server_choice**.
4. **case "$server_choice" in**: Début d'une structure de contrôle de cas. Elle permet d'exécuter différents blocs de code en fonction de la valeur de **server_choice**.
5. **Apache|Apache2)**: Première option du cas. Si **server_choice** est égal à "Apache" ou "Apache2", alors le code suivant jusqu'à **;;** sera exécuté.
6. **sudo apt-get update**: Met à jour les référentiels des paquets pour s'assurer que les dernières versions des logiciels disponibles sont accessibles.
7. **sudo apt-get install apache2**: Installe le serveur web Apache.

8. **Nginx)**: Deuxième option du cas. Si **server_choice** est égal à "Nginx", alors le code suivant jusqu'à **;;** sera exécuté.
9. **sudo apt-get install nginx**: Installe le serveur web Nginx.
10. ***)**: Option par défaut du cas. Si **server_choice** ne correspond à aucune des options précédentes, alors le code suivant jusqu'à **;;** sera exécuté.
11. **echo "Choix invalide. Veuillez choisir Apache/Apache2 ou Nginx."**: Affiche un message d'erreur indiquant que le choix est invalide.
12. **install_web_server**: Appel à la fonction **install_web_server** pour commencer l'installation du serveur web.

Ce schéma est répété pour les autres composants (**install_phpmyadmin**, **install_database_server**, **install_nodejs_npm**, **install_git**), chacun ayant sa propre logique pour installer le logiciel correspondant.

À la fin, **echo "Installation terminée."** affiche un message indiquant que l'installation est terminée.

C'est ainsi que le script fonctionne pour installer et configurer les différents composants requis pour un projet web.

Sécuriser ses scripts

1. Injection de code malveillant : Si nos scripts prennent des entrées utilisateur sans validation adéquate, nous pourrions être vulnérables aux attaques par injection de code. Cela pourrait permettre à un attaquant d'exécuter des commandes arbitraires sur notre système.
2. Exécution de privilèges élevés : Certains de nos scripts peuvent nécessiter des privilèges élevés pour effectuer certaines tâches. Si un attaquant parvient à exploiter une faille dans nos scripts, il pourrait obtenir un accès non autorisé au système avec ces privilèges élevés.
3. Exposition de données sensibles : Nos scripts peuvent manipuler des données sensibles telles que des informations d'identification ou des données personnelles. Une mauvaise gestion de ces données pourrait entraîner leur exposition non autorisée.
4. Faiblesse dans les dépendances logicielles : Nos scripts peuvent dépendre de bibliothèques ou de modules tiers. Si ces dépendances sont obsolètes ou contiennent des vulnérabilités connues, cela pourrait compromettre la sécurité de notre système. Pour sécuriser nos scripts précédemment développés, voici quelques mesures à prendre :
5. Validation des entrées utilisateur : Nous devons toujours valider et filtrer les entrées de l'utilisateur pour prévenir les attaques par injection de code. Par exemple, nous pouvons utiliser des fonctions comme **read -p** pour la saisie utilisateur.
6. Échappement des données : Lorsque des données sont incorporées dans des commandes système, nous devons les échapper correctement pour éviter les injections de commandes. Par exemple, nous pouvons utiliser **sudo apt-get install apache2** plutôt que d'exécuter directement **apt-get install apache2**.
7. Privilèges minimaux : Nous devons réduire les privilèges de nos scripts au strict nécessaire. Évitons d'exécuter nos scripts avec des privilèges de superutilisateur (**sudo**) sauf si cela est absolument nécessaire.
8. Protection des données sensibles : Nous ne devons pas stocker de données sensibles, telles que des mots de passe ou des informations d'identification, dans nos scripts eux-mêmes. Utilisons des mécanismes sécurisés pour gérer ces informations, comme les variables d'environnement ou les fichiers de configuration chiffrés.
9. Mise à jour des dépendances : Assurons-nous que toutes les dépendances logicielles utilisées par nos scripts sont à jour et ne contiennent pas de vulnérabilités connues. Mettons régulièrement à jour les bibliothèques et les modules tiers.

Utilisation d'API Web dans un script

```
#!/bin/bash

# URL de l'API Web OpenWeatherMap

API_URL=http://api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=98e435d2a01ded2a727d23c9bbcf5eec

# Chemin du fichier de journal
LOG_FILE="api_log.txt"

# Fonction pour journaliser les messages
log_message() {
   echo "$(date +'%Y-%m-%d %H:%M:%S') $1" >> "$LOG_FILE"
}


# Fonction pour effectuer une requête GET à l'API
get_data_from_api() {
   local response=$(curl -s -o /dev/null -w "%{http_code}" "$API_URL")
   if [ "$response" -eq 200 ]; then
      local data=$(curl -s "$API_URL")
      echo "$data"
      log_message "Requête GET vers l'API réussie."
      log_message "Réponse de l'API : $data"
   else

      log_message "Erreur lors de la requête GET vers l'API. Code de réponse HTTP : $response"
      exit 1
   fi
}


# Exécuter la fonction pour obtenir les données de l'API
get_data_from_api
```

Ce script est un script shell (bash) qui récupère les données météorologiques de la ville de Londres depuis l'API Web OpenWeatherMap.

Voici une explication ligne par ligne :

1. **#!/bin/bash**: Cette ligne indique que le script doit être interprété par l'interpréteur de commandes Bash.
2. **API_URL=http://api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=98e435d2a01ded2a727d23c9bbcf5eec**: Cette ligne définit l'URL de l'API Web OpenWeatherMap avec la

ville de Londres et une clé d'API. Cette URL est utilisée pour récupérer les données météorologiques.

3. **LOG_FILE="api_log.txt"**: Cela définit le nom du fichier de journal où les messages de journal seront enregistrés.
4. **log_message()**: Cette fonction prend un message en argument et le journalise en ajoutant la date et l'heure actuelles au début de chaque message.
5. **get_data_from_api()**: Cette fonction effectue une requête GET à l'API définie dans **API_URL**. Elle utilise **curl** pour effectuer la requête. Si la réponse du serveur est 200 (OK), elle enregistre la réponse dans une variable et journalise le succès de la requête avec le contenu de la réponse. Sinon, elle journalise l'échec de la requête avec le code de réponse HTTP et quitte le script avec le code d'erreur 1.
6. **get_data_from_api**: Cette ligne exécute la fonction **get_data_from_api**, déclenchant ainsi la récupération des données météorologiques de l'API.