Systèmes d'exploitation

GROUPE N°1

Joé FEUCHT
Isamettin AYDIN

Rapport de clôture de projet

–

Projet Scripting Bash

Sommaire

Introduction

La manipulation des fichiers et des processus et des services fait partie des opérations courantes d'un administrateur système. Les commandes de base sont pour ce faire régulièrement utilisées. Certains cas particuliers nécessitent néanmoins l'usage d'options supplémentaires utilisées rarement et qu'il faut du coup rechercher à grand coups de « man » ou de recherches « google mon ami ». L'accès aux informations système sont pour la majorité réservées à l'administrateur. Dans certains cas les utilisateurs pourraient tirer profit également d'un accès à ces informations mais la délégation de droits permettant l'accès à celles-ci est compliqué voire impossible à réaliser.

Le but de ce petit projet est donc de fournir un outil d'aide à l'analyse fichiers, processus et services à destination d'un administrateur système et ou d'un utilisateur auquel vous déléguerez les droits d'usage du script.

Objectifs :

Produire des scripts permettant une consultation simplifiée des fichiers et des processus et des services sur un système Linux. Le Shell utilisé est le Bash.

Comment exécuter le script ?

Dépendance(s)

Le script a une dépendance sur l'outil « tree » permettant d'afficher l'arborescence des répertoires et des fichiers présents sur le système. Si « tree » n'est pas installé sur le système, cela peut être fait en renseignant les commandes suivantes :

sudo apt-get update
sudo apt-get install tree

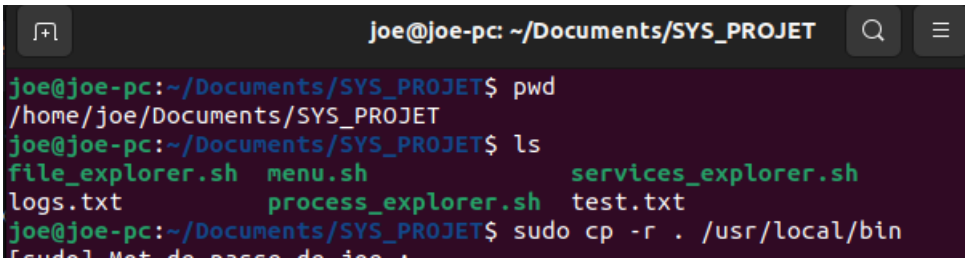
PATH & Exécution

Afin de pouvoir rendre le script opérationnel sur l'ensemble du système, nous avons plusieurs possibilités :

- Ajouter le chemin vers le répertoire contenant l'ensemble des scripts à la variable PATH. Afin de rendre cela persistant nous pouvons ajouter cela dans le fichier **.bashrc**, c'est un fichier qui est exécuté à chaque fois qu'un shell bash est exécuté.
- Copier les sources du script dans un répertoire déjà présent dans la variable PATH et accessible par l'ensemble des utilisateurs du système comme par exemple **/usr/local/bin**

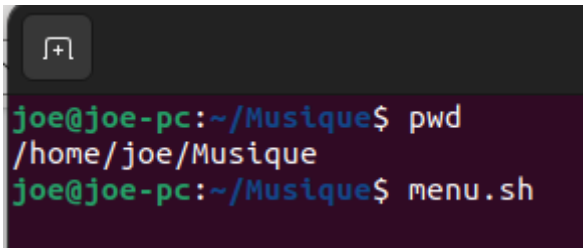
Dans notre cas nous avons choisis la seconde option car elle est plus simple dans sa configuration et ne nécessite pas d'ajouter des éléments dans la variable PATH ni de devoir toucher au fichier .bashrc.

- Tout d'abord nous allons copier le contenu du répertoire du projet dans /usr/local/bin

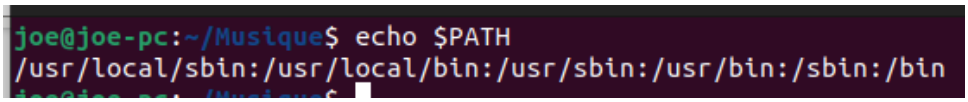


sudo cp –r . /usr/local/bin

- Par conséquent, cela devrait fonctionner...



- … car la variable PATH contient le chemin vers /usr/local/bin par défaut



Délégation des droits

Afin de permettre à un simple utilisateur de pouvoir utiliser ces scripts nous allons devoir mettre en place une configuration.

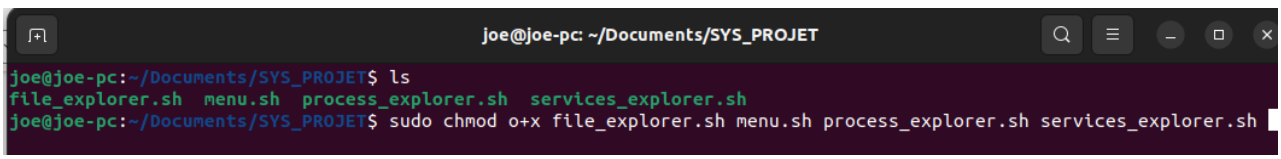
- Les scripts sont stockées à un emplacement accessible par les autres utilisateurs (voir PATH & Exécution)
- Les « autres utilisateurs » doivent avoir les droits nécessaires à l'exécution des scripts :

Avec un utilisateur ayant des droits root, et après s'être rendu dans le répertoire contenant les scripts, effectuez la commande suivante :

sudo chmod o+x file_explorer.sh menu.sh process_explorer.sh services_explorer.sh

ou

*sudo chmod o+x *.sh*



PS : On aurait pu également ajouter les utilisateurs ayant pour vocation à utiliser le script dans un groupe spécifique, définir le groupe d'appartenance des scripts comme étant celui-ci, puis utiliser la commande sudo chmod g+x afin d'attribuer les droits d'exécution des scripts pour les utilisateurs appartenant à ce groupe.

Description des scripts

Les fonctions ne seront pas entièrement détaillées pour des besoins de clarté du rapport…

Partie 1 – Explorateur de fichiers

Fichier

file_explorer.sh

Description

Script bash permettant d'afficher diverses informations sur les fichiers/répertoires présents sur le système.

Commandes

a) *Afficher le répertoire courant dans lequel je me trouve*

Fonction :

show_current_dir()

Fonctionnement :

Affiche le répertoire courant en utilisant la commande « pwd ».

b) *Indiquez la date et l'heure du système*

Fonction :

show_date_time()

Fonctionnement :

Affiche la date et l'heure du système en utilisant la commande « date ».

c) *Indiquer le nombre de fichiers présents dans le répertoire courant et leur taille (en Ko, Mo ou Go à vous de définir l'unité la plus pertinente)*

Fonction :

show_files_info()

Fonctionnement :

Affiche le nombre de fichiers et la taille totale des fichiers dans le répertoire courant en utilisant les commandes « ls » et « du ».

d) *Indiquer le nombre de sous-répertoires présents dans le répertoire courant*

Fonction :

show_subdirs_count()

Fonctionnement :

Affiche le nombre de sous-répertoires présents dans le répertoire courant en utilisant la commande « find » (la commande « wc » permet d'afficher le nombre de répertoire retourné par la commande « find »).

e) *Indiquer l'arborescence (en permettant de paramétrer une profondeur) du répertoire courant*

Fonction :

show_directory_tree()

Fonctionnement :

Affiche l'arborescence du répertoire courant jusqu'à une profondeur spécifiée par l'utilisateur en utilisant la commande « tree ».

f) *Indiquer le poids (quantité de données) de chaque sous-répertoire du répertoire courant (veiller à comptabiliser tous les fichiers présents dans l'arborescence de chaque sous répertoire)*

Fonction :

show_subdirs_size()

Fonctionnement :

Affiche le poids de chaque sous-répertoire dans le répertoire courant en utilisant la commande « du -sh */ ».

g) *Changer de répertoire courant pour poursuivre mes investigations*

Fonction :

change_directory()

Fonctionnement :

Permet de changer de répertoire courant vers un répertoire spécifié par l'utilisateur en utilisant la commande « cd ».

h) *Rechercher les fichiers plus récents qu'une date (définie en paramètre) pour le répertoire courant*

Fonction :

find_files_newer_than()

Fonctionnement :

Permet de trouver tous les fichiers dans le répertoire courant qui ont été modifiés depuis une date spécifiée par l'utilisateur, en utilisant la commande « find ».

i) *Rechercher les fichiers plus récents qu'une date (définie en paramètre) présents dans tous les sous-répertoires de l'arborescence du répertoire courant*

Fonction :

find_files_newer_than_recursive()

Fonctionnement :

Permet de rechercher de manière récursive tous les fichiers modifiés après une certaine date saisie par l'utilisateur, en utilisant la commande « find ».

j) *Rechercher les fichiers plus anciens qu'une date (définie en paramètre) pour le répertoire courant*

Fonction :

find_files_older_than()

Fonctionnement :

Permet de rechercher les fichiers dans le répertoire courant qui ont une date de modification antérieure à celle spécifiée par l'utilisateur, en utilisant la commande « find ».

k) *Rechercher les fichiers plus anciens qu'une date (définie en paramètre) présents dans tous les sous-répertoires de l'arborescence du répertoire courant*

Fonction :

find_files_older_than_recursive()

Fonctionnement :

Permet de trouver tous les fichiers dans le répertoire courant et les sous-répertoires dont la date de modification est antérieure à une date spécifiée par l'utilisateur, en utilisant la commande « find ».

l) *Rechercher les fichiers de poids supérieur à une valeur indiquée présents dans le répertoire courant*

Fonction :

find_files_larger_than()

Fonctionnement :

Permet de chercher les fichiers dans le répertoire courant dont la taille est supérieure à une certaine valeur donnée en Ko, Mo ou Go, en utilisant la commande « find ».

m) *Rechercher les fichiers de poids supérieur à une valeur présents dans tous les sous-répertoires de l'arborescence du répertoire courant*

Fonction :

find_files_larger_than_recursive()

Fonctionnement :

Permet de chercher de manière récursive les fichiers dans le répertoire courant dont la taille est supérieure à une certaine valeur donnée en Ko, Mo ou Go, en utilisant la commande « find ».

n) *Rechercher les fichiers de poids inférieur à une valeur indiquée présents dans le répertoire courant*

Fonction :

find_files_smaller_than()

Fonctionnement :

Permet de chercher les fichiers dans le répertoire courant dont la taille est inférieure à une certaine valeur donnée en Ko, Mo ou Go, en utilisant la commande « find ».

o) *Rechercher les fichiers de poids inférieur à une valeur indiquée présents dans tous les sous-répertoires de l'arborescence du répertoire courant*

Fonction :

find_files_smaller_than_recursive()

Fonctionnement :

Permet de chercher de manière récursive les fichiers dans le répertoire courant dont la taille est inférieure à une certaine valeur donnée en Ko, Mo ou Go, en utilisant la commande « find ».

p) *Rechercher tous les fichiers d'une extension donnée (définie en paramètre) présents dans tous les sous-répertoires de l'arborescence du répertoire courant*

Fonction :

find_files_with_extension()

Fonctionnement :

Permet de chercher les fichiers avec une extension donnée dans le répertoire courant, en utilisant la commande « find ».

q) *Rechercher tous les fichiers d'une extension donnée (définie en paramètre) présents dans le répertoire courant*

Fonction :

find_files_with_extension_recursive()

Fonctionnement :

Permet de chercher les fichiers avec une extension donnée de manière récursive dans le répertoire courant et ses sous-répertoires, en utilisant la commande « find ».

r) *Rechercher tous les fichiers dont le nom contient une chaine de caractère (définie en paramètre) présents dans tous les sous-répertoires de l'arborescence du répertoire courant*

Fonction :

find_files_with_string()

Fonctionnement :

Permet de chercher les fichiers qui contiennent une chaîne de caractères donnée dans le nom de fichier, en utilisant la commande « find ».

s) *Produire un fichier de sortie contenant le résultat de la recherche effectuée qui n'écrase pas les résultats de la recherche précédente (tenir compte par exemple de la date et l'heure système)*

Fonction :

write_logs()

Fonctionnement :

Permet d'ajouter des informations (argument passé à la fonction) sur les sorties de certaines fonctions dans un fichier de logs.

t) *Proposer une fonctionnalité supplémentaire que vous jugez intéressante et qui manque à la liste précédente (argumentez votre choix)*

Fonction :

compare_files()

Fonctionnement :

Permet de comparer deux fichiers que l'utilisateur doit renseigner, en utilisant la commande « diff ».

Argumentation :

La comparaison entre deux fichiers était une fonctionnalité qui manquait à ce projet. Devoir comparer manuellement les différences entres deux fichiers peut être un travail fastidieux, c'est pourquoi l'implémentation d'une fonction répondant à cette problématique s'est avérée logique.

Partie 2 – Explorateur de processus

Fichier

process_explorer.sh

Description

Script bash permettant d'obtenir diverses informations sur les processus présents sur le système

Commandes

a) *Identifier tous les processus présents sur le système et indiquer leur propriétaire*

Fonction :

show_all_processes()

Fonctionnement :

Affiche une liste de tous les processus présents sur le système en temps réel en utilisant une loop while et un minuteur de 30 secondes qui rafraîchit la liste toutes les 30 secondes. On utilise un formatage avec des couleurs afin de rendre cela plus « beau ». Diverses commandes importantes sont utilisées :

- « find /proc » : cherche les processus sur le système
- « ps -o user= » : permet d'obtenir le nom de l'utilisateur du processus
- « ps -o comm= » : permet d'obtenir le nom du processus correspondant au PID
- « read -s -n 1 -t 1 key
  if [[$key = "q"]] || [[$key = "Q"]] » : cela va nous permettre de quitter la boucle si l'utilisateur appuie sur Q ou q.

b) *Identifier uniquement les processus actifs et indiquer leur propriétaire*

Fonction :

show_active_processes()

Fonctionnement :

Similaire à la fonction *show_all_processes()* mais pour les processus actifs (on va utiliser « ps -eo » au lieu de « find /proc » afin d'avoir uniquement les processus actifs).

c) *Identifier les processus appartenant à un utilisateur donné en paramètre*

Fonction :

show_processes_by_user()

Fonctionnement :

Similaire à la fonction *show_all_processes()* mais pour un utilisateur spécifique, on va avoir une condition supplémentaire qui va vérifier si le propriétaire du processus == username (qui est renseigné par l'utilisateur du script).

d) *Identifier les processus consommant le plus de mémoire et indiquer leur propriétaire*

Fonction :

show_processes_by_memory()

Fonctionnement :

Similaire à la fonction *show_all_processes()* mais on va effectuer un tri avec la commande « sort -k 3 -r » afin de trier par utilisation de mémoire.

e) *Identifier les processus dont le nom contient une chaine de caractères (définie en paramètre) et indiquer leur propriétaire*

Fonction :

show_processes_by_name()

Fonctionnement :

Similaire à *show_all_processes()* mais pour les processus contenant une chaîne de caractères spécifique. On utilise une variable « filter » qui va uniquement chercher les processus contenant dans son nom ce filtre.

f) *Produire un fichier de sortie contenant le résultat de la recherche effectuée qui n'écrase pas les résultats de la recherche précédente (tenir compte par exemple de la date et l'heure système)*

Fonction :

write_logs()

Fonctionnement :

Permet d'ajouter des informations (argument passé à la fonction) sur les sorties de certaines fonctions dans un fichier de logs.

g) *Proposer une fonctionnalité supplémentaire que vous jugez intéressante et qui manque à la liste précédente (argumentez votre choix)*

Fonction :

show_processes_by_start_time()

Fonctionnement :

Similaire à la fonction *show_all_services()* mais les services sont triés par heure de lancement (« --sort=-start_time ») et une colonne supplémentaire avec cette information a été ajoutée (« ${START_TIME_COLOR}%-23s${NC} »).

Argumentation :

Il peut être utile de savoir depuis quand tourne un processus, c'est pourquoi nous avons décidés d'implémenter cette fonction.

Partie 3 – Explorateur de services

Fichier

services_explorer.sh

Description

Script bash permettant d'obtenir diverses informations sur les services présents sur le système

Commandes

a) *Identifier les services disponibles/installés sur le système*

Fonction :

show_all_services()

Fonctionnement :

Permet d'afficher une liste avec les différents services installés sur le système (+ leur statut), une loop permet de rafraîchir cette liste toutes les 30 secondes. On utilise un formatage avec des couleurs afin de rendre cela plus « beau ». Diverses commandes importantes sont utilisées :

- « systemctl list-unit-files --type=service --state=enabled,disabled » : cela va nous permettre de récupérer l'ensemble des services présents sur le système
- « if [["$status" = "active"]] » : cette condition va nous permettre d'avoir un affichage différent en fonction de si le service est actif ou non.
- « read -s -n 1 -t 1 key
  if [[$key = "q"]] || [[$key = "Q"]] » : cela va nous permettre de quitter la boucle si l'utilisateur appuie sur Q ou q.

b) *Identifier les services actifs sur le système*

Fonction :

show_active_services()

Fonctionnement :

Similaire à la fonction *show_all_services()* mais avec uniquement « --state=enabled » dans la commande « systemctl »

c) *Identifier le statut d'un service dont le nom contient une chaine de caractères (définie en paramètre)*

Fonction :

show_services_by_name()

Fonctionnement :

Similaire à la fonction *show_all_services()* mais pour les services contenant une chaîne de caractères spécifique. On utilise une variable « filter_string » qui va uniquement chercher les services contenant dans son nom ce filtre.

d) *Proposer une fonctionnalité supplémentaire que vous jugez intéressante et qui manque à la liste précédente (argumentez votre choix)*

Fonction :

start_service() **et** *stop_service()* ***(2 fonctions pour le prix d'une… c'est cadeau)***

Fonctionnement :

Fonctions permettant de démarrer et de stopper un service, en utilisant les commandes « systemctl start » et « systemctl stop »

Argumentation :

Il est utile de pouvoir voir les différents services présents sur un système, mais il est encore mieux de pouvoir les manipuler, c'est pourquoi il nous paraissait pertinent d'ajouter au projet deux fonctions permettant de démarrer et de stopper des services.

Partie 4 – Interface globale

Fichier

menus.sh

Description

Point d'entrée de l'outil, propose une interface globale permettant ensuite d'accéder aux autres scripts (services_explorer.sh, process_explorer.sh et file_explorer.sh)

Difficultés rencontrées & solutions apportées

Création du fichier contenant les logs

Lorsque l'on copie les scripts dans le répertoire **/usr/local/bin** et qu'on tente d'utiliser l'une des fonctionnalités du script générant des logs, on peut se retrouver avec le message « Permission non accordée » car l'utilisateur n'a pas les droits pour créer un fichier de logs dans ce répertoire. Par défaut le script tente de créer le fichier dans celui-ci.

Afin de résoudre ce problème, le script va créer un fichier de logs dans le répertoire : « /home/*username*/script_logs ». Il n'y aura pas de problème de droits, car l'utilisateur courant a les droits pour créer un répertoire/fichier dans son répertoire personnel.

Limite de droits

Les scripts Bash utilisent diverses commandes, toutefois, celles-ci ne permettent pas de contourner les permissions définies sur les fichiers ou les répertoires. Ainsi, si un utilisateur n'a pas les droits adéquats sur ces derniers il ne pourra pas effectuer toutes les commandes. Les droits d'accès doivent donc être configurés en amont…

Compréhension des besoins

Il nous semblait parfois difficile de comprendre les besoins exprimés dans l'énoncé du projet. Ceux-ci pouvaient parfois prêter à confusion.

Difficulté générale

La difficulté du projet était globalement moyenne, en revanche le projet était assez long à réaliser car il y avait un nombre élevé de fonctionnalités à implémenter.

Fonctionnalités additionnelles

Se référer à :

- *Explorateur de fichiers :*

Proposer une fonctionnalité supplémentaire que vous jugez intéressante et qui manque à la liste précédente (argumentez votre choix)

- *Explorateur de processus :*

Proposer une fonctionnalité supplémentaire que vous jugez intéressante et qui manque à la liste précédente (argumentez votre choix)

- *Explorateur de services :*

Proposer une fonctionnalité supplémentaire que vous jugez intéressante et qui manque à la liste précédente (argumentez votre choix)

Conclusion

Ce projet était très amusant à réaliser !

D'abord, il peut s'avérer réellement utile pour les néophytes mais également pour les administrateurs systèmes. En effet, cela peut permettre d'accélérer la productivité et nous faire gagner du temps grâce aux fonctions toutes prêtes présentes !

De plus, ce projet nous a permis de mettre un premier pas dans la programmation en Bash que nous ne connaissions pas plus que cela tout en nous permettant de découvrir ou redécouvrir des commandes courantes du Shell.

Enfin, le projet a mis en évidence l'importance de réaliser une documentation complète afin de faciliter la compréhension du code ainsi que de faire une synthèse des tenants et aboutissants de ce dernier.