TP Linux Extended Shell

Administration des serveurs – Exploitation des fichiers log

Contexte : Les fichiers LOG sont une part importante des systèmes d'exploitation. Ils contiennent une trace de l'activité de la machine et permettent à l'administrateur de retrouver un évènement à l'origine d'un dysfonctionnement.

Sous Linux, la plupart de ces fichiers LOG sont dans le dossier /var/log.

Le LOG des messages systèmes (/var/log/messages) est accessible directement à l'administrateur, ou par la commande **dmesg**.

Le contenu de ces fichiers est souvent très technique et demande une certaine habitude. Parfois, seule une information nous intéresse et ce TP va nous montrer comment la retrouver par des commandes shell ou par un script.

Dans ce travail, vous serez administrateur d'un serveur d'accès à Internet pour une petite entreprise.

La passerelle Internet Linux est équipée du logiciel de proxy SQUID.

Vous exploitez le fichier journal de cette application (fichier ***access.log***) qui garde toutes les traces de la navigation internet des utilisateurs.

Ce fichier vous est fourni dans le dossier TP. Il a été récupéré sur un serveur proxy.

C'est un fichier texte, donc lisible par un éditeur de texte (nano, vi, notepad, gedit, kwrite, …)

Mais étant donné la taille importante du fichier, nous préfèrerons l'exploiter par des commandes shell évoluées.

PARTIE 1 : EXPLOITATION DES FICHIERS LOG PAR LA CONSOLE

Commandes liées aux fichiers texte :

➔ Copiez dans votre espace personnel le fichier *access.log* fourni avec ce TP.

Voici quelques commandes du Shell utiles pour gérer des fichiers textes simple :

- `head access.log` — Afficher les 10 premières lignes du fichier
- `head -40 access.log` — Afficher les 40 premières lignes du fichier
- `tail access.log` — Afficher les 10 dernières lignes du fichier
- `cat access.log` — Afficher tout le fichier
- `wc access.log` — Statistiques sur le fichier (lignes/mots/caract.)
- `wc –l access.log` — Statistiques : uniquement le nombre de lignes
- `grep 10.69.95.3 access.log` — Rechercher dans le fichier toutes des lignes qui contiennent `10.69.95.3`

Redirection de commandes shell : le Pipe

Le *pipe* (symbole |) est un outil puissant qui permet d'exécuter une commande shell à partir du résultat d'une autre commande shell.

Exemple : `ls -l | wc`

Explication : La commande *wc* va compter le nombre de lignes/mots/caractères affichés par la commande *ls -l*.

Tester les commandes suivantes :

1. `head access.log | sort -k 3`
   Tri les lignes par la colonne 3 (le séparateur est l'espace. L'ordre est alphabétique).

2. `head access.log | sort -k 3 -r`
   Idem mais l'option –r inverse l'ordre du tri

3. `head access.log | tr -s 't'`
   « Squeeze » la lettre 't' de chaque ligne : remplace plusieurs 't' qui se suivent par 1 seul 't'. Voir ce qui se passe sur le mot « http ».

4. `head access.log | tr [a-z] [A-Z]`
   Affiche les lignes en remplaçant les lettres minuscules par des majuscules

5. `head access.log | tr -s ' ' | cut -f2 -d ':'`
   Réduit les espaces inutiles, découpe la ligne délimitées par les espaces, et affiche la colonne 2

6. `head access.log | cut -f1 -d '.' | sort -u`
   Coupe la ligne en fonction des « . » et affiche la première colonne, en supprimant les doublons.

7. `head access.log | cut -c 7-13`
   Affiches les caractères 7 à 13 de chaque ligne

Exercices :

Une adresse IP correspond au numéro unique d'une machine sur un réseau d'ordinateurs (Internet par exemple). Nos machines utilisent la norme IPv4 qui donne des adresses sous la forme de 4 nombres séparés par des points. Exemple : 10.69.88.1

8. Trouver la commande qui permet d'afficher l'adresse IP (3eme position dans la ligne) des 30 premières lignes du fichier, sans doublon.
   Vous aurez besoin des commandes *head*, ***tr***, *cut*, et *sort*. A vous de trouver les bonnes options en vous aidant des exemples précédents.

Pourquoi **tr** ? Les différents mots de chaque ligne sont séparés par un ou plusieurs espaces. Pour faire un bon découpage (cut) il faut qu'il n'y ait qu'1 seul espace entre chaque mot : `tr -s ' '`

9. Trouver la commande qui permet d'afficher le nombre d'adresse IP différentes présentes dans la totalité du fichier.

10. Trouver la commande qui affiche la liste de tous les sites internet consultés, sans doublons. On ne veut que l'adresse du site, et pas le nom de la page.

    Exemple : L'adresse de la page est http ://comdyn.vauv.free.fr/abs_lgt_j.php
    Le nom du site est http ://comdyn.vauv.free.fr

PARTIE 2 : DES SCRIPTS ET DES VARIABLES

Une variable est un moyen de manipuler des données stockées en mémoire RAM de l'ordinateur.

Une variable est définie par son adresse, son contenu, son nom et son type.

En shell Linux, on accède à la donnée par son nom. C'est le programmeur qui choisi le nom.

Le shell Linux donne, aux variables déclarées automatiquement, le type alphanumérique.

On peut imposer le type d'une variable en la déclarant, par exemple, variable numérique.

Lorsqu'on utilise le contenu d'une variable, il faut ajouter le symbole $ devant son nom.

11. Testez les commandes suivantes :

```
a=3
echo $a
a=$a+1
echo $a
```

On crée la variable « a » et on l'initialise avec la valeur 3

Observations dans votre feuille de synthèse.

12. Déclarer les variables en numérique pour effectuer des calculs. Testez ceci :

```
declare -i  b
b=3
b=$b+1
echo $b
```

On crée la variable « b » de type *integer* (nombre entier)

Observations dans votre feuille de synthèse.

13. Nous avons déjà créé des shell-script (Revoir le TP 5-Premiers pas en Shell-Script). Un shell-script est un fichier qui contient des commandes shell : les même que celles qu'on utilise dans la console.
Pour créer ce fichier texte, il faut un simple éditeur (ex : nano, vim, ou notepad++)

Ensuite, il faut penser à donner un droit « exécution » à ce fichier : Exemple : `chmod +x essai1`
Et pour lancer son exécution, utiliser le nom du script précédée de « ./ ». Exemple : `./essai1`

Créer le shell script suivant que vous nommerez ***essai1*** :

```
#!/bin/bash
clear
echo bonjour !
echo Donnez votre nom :
read var1
echo Donnezvotre prenom :
read var2
echo Vous vous appelez $var1 $var2
echo Nous sommes $1
var3=$(whoami)
echo Votre nom utilisateur est $var3
```

Rendez-le exécutable et testez-le en tapant : **`./essai1 mercredi`**

14. Que contient la variable $1 ?
15. Que contient la variable $var3 ?

Conclusion de votre observation (A RETENIR):

Il y a différentes façons d'affecter une valeur à une variable :

a) Au clavier, pendant l'exécution d'un script : `read a`
b) Par une commande shell `a=3`
c) Par le résultat d'une commande shell : `a=$(whoami)`
d) Au lancement d'un script, par les ARGUMENTS : `./monscript il fait beau`

16. **Exercice** :

Créer un script nommé `./trouve`
Ce script affiche la liste des sites consultés par une adresse IP spécifique, dans le fichier access.log

Usage : `./trouve 10.69.88.31`
On recherche les sites consultés par le poste 10.69.88.31

PARTIE 3 : ENCHAINEMENT CONDITIONNEL && ET || :

Principe : Lorsqu'une commande s'exécute, soit elle réussi et affiche un résultat, soit elle échoue et elle affiche un message d'erreur. Si la commande n'affiche rien (exemple : mkdir), on considère qu'elle a réussi son travail.

Les commandes && et ||détectent si la commande côté gauche s'est exécutée correctement. Les exemples suivants montrent comment tirer parti de ce comportement :

Exemples d'utilisation :

```
ls toto || echo bonjour > toto
```

Si la commande *ls toto* réussit, c'est-à-dire si le fichier *toto* existe, la commande *echo* ne sera pas exécutée : c'est une opération logique de type OU EXCLUSIF.

```
ls toto && chmod 666 toto
```

Si la commande *ls toto* réussit, c'est-à-dire si le fichier *toto* existe, la commande *chmod* sera exécutée : c'est une opération logique de type ET.

Note : pour ne pas être dérangé par l'affichage des résultats, on peut rediriger l'affichage vers le « néant » :

`ls toto >/dev/null`	on supprime l'affichage normal
`ls toto 2>/dev/null`	on supprime les messages d'erreur
`ls toto 2&>/dev/null`	on supprime tous les messages et les affichages

S'il doit y avoir plusieurs commandes à exécuter côté droit, on utilise les accolades :

```
ls toto || {
           echo bonjour > toto;
           chmod 666 toto
        }
```

17. **TRAVAIL** : Reprenez l'exercice précédent (script *trouve*) et copiez le sous le nom *trouve2* (commande *cp).*
L'appel se fera de la façon suivante :

```
./trouve2   ./access.log    10.69.88.31
```

On indique ici le nom du fichier dans lequel il faut chercher. On utilise **donc 2 ARGUMENTS** :

$1 qui correspond au premier (./access.log)

$2 qui correspond au deuxième (10.69.88.31)

Si le fichier mentionné n'existe pas (la commande *ls* échoue), alors on affiche un message d'alerte (commande *echo*) et on quitte le script (commande *exit 1*).