SCRIPTS SOUS LINUX

TABLE DES MATIERES

Source principale : Openclassrooms

1. PETITS RAPPELS

Les alias

Un alias permet d'enregistrer une commande

Ex : alias l='ls -lth' permet en tapant l d'avoir une présentation de la liste des fichiers et répertoires clarifiée

La recherche

`find /var/log/ -name "syslog*"` rechercher dans un répertoire

`find / -name "syslog*"` rechercher partout les noms contenant syslog

Appeler une commande

find -name ''scr*.sh'' -exec chmod 755 {} \ ; donne des droits d'exécution à tous les fichiers contenant scr

find -name ''scr*.sh'' -delete les supprimera tous

Les redirections

1. Pour rediriger le contenu d'une commande vers un fichier **>**

   Ls > liste

   Ls **>>** liste écrira à la fin du fichier si il existe

Rem : Il faut savoir que toutes les commandes produisent deux flux de données différents :

- **la sortie standard** : pour tous les messages (sauf les erreurs) ;
- **la sortie d'erreurs** : pour toutes les erreurs.

2. Pour rediriger les erreurs vers un fichier **2>**

   Cat fichierqquiexistepas 2>erreur.log

   Affichera cat : fichierquiexistepas : Aucun fichier ou dossier de ce type

3. Pour rediriger les erreurs dans le fichier de sortie standard 2>&1
4. Lire depuis un fichier <
   Cat < erreur.log
5. Lire depuis le clavier <<
   Sort -n << stop
   Stop est la commande de fin de saisie (sinon on ne sort pas du programme)
   Sort -n permet de trier des nombres

2. LES SHELLS

Il existe plusieurs environnements console : les shells

Les fonctionnalités offertes par l'invite de commandes peuvent varier en fonction du **shell**que l'on utilise.

Noms de quelques-uns des principaux shells qui existent.

sh : *Bourne Shell*. L'ancêtre de tous les shells.

bash : *Bourne Again Shell*. Une amélioration du *Bourne Shell*, disponible par défaut sous Linux et Mac OS X.

ksh : *Korn Shell*. Un shell puissant assez présent sur les Unix propriétaires, mais aussi disponible en version libre, compatible avec bash.

csh : *C Shell*. Un shell utilisant une syntaxe proche du langage C.

tcsh : *Tenex C Shell*. Amélioration du *C Shell*.

zsh : *Z Shell*. Shell assez récent reprenant les meilleures idées de bash, ksh et tcsh.

3. CREATION DU FICHIER

```
$ nano essai.sh
```

Le fichier

```
#!/bin/bash
```

`/bin/bash` peut être remplacé par `/bin/sh` si vous souhaitez coder pour sh, `/bin/ksh` pour ksh, etc.

4. COMMANDES

Le principe est très simple : il vous suffit d'écrire les commandes que vous souhaitez exécuter. Ce sont les mêmes que celles que vous tapiez dans l'invite de commandes !

- `ls` : pour lister les fichiers du répertoire.
- `cd` : pour changer de répertoire.
- `mkdir` : pour créer un répertoire.
- `grep` : pour rechercher un mot.
- `sort` : pour trier des mots.
- etc.

Exemple

```
#!/bin/bash
# Affichage le répertoire et  la liste des fichiers
Pwd
ls
```

5. DONNER LES DROITS D'EXECUTION AU SCRIPT

Si vous faites un `ls -l` pour voir votre fichier qui vient d'être créé, vous obtenez ceci :

```
$ ls -l
total 4
-rw-r--r-- 1 mateo21 toto 17 2009-03-13 14:33 essai.sh
```

Ce qui nous intéresse ici, ce sont les droits sur le fichier : `-rw-r--r--`.
Si vous vous souvenez un petit peu du chapitre sur les droits, vous devriez vous rendre compte que notre script peut être lu par tout le monde (`r`), écrit uniquement par nous (`w`), et n'est pas exécutable (pas de `x`).

Or, pour exécuter un script, il faut que le fichier ait le droit « exécutable ». Le plus simple pour donner ce droit est d'écrire :

```
$ chmod +x essai.sh
```

Vous pouvez vérifier que le droit a bien été donné :

```
$ ls -l
total 4
-rwxr-xr-x 1 mateo21 toto 17 2009-03-13 14:33 essai.sh
```

Actuellement, le script doit être lancé via `./essai.sh` et vous devez être dans le bon répertoire, ou alors vous devez taper le chemin en entier,comme `/home/toto/scripts/essai.sh`.

6. Le Path

est une variable système qui indique où sont les programmes exécutables sur votre ordinateur. Si vous tapez `echo $PATH` vous aurez la liste de ces répertoires « spéciaux ».

Les programmes pour pouvoir être exécutés depuis n'importe quel répertoire sans « ./ » devant sont placés dans un des répertoires du PATH.

Il vous suffit de déplacer ou copier votre script dans un de ces répertoires,

`/bin`, `/usr/bin` ou `/usr/local/bin` (ou encore un autre répertoire du PATH).

7. Les variables

- Comme dans la plupart des langages de programmation, on peut créer des variables en shell qui stockent temporairement des valeurs en mémoire. Une variable nommée `variable` est accessible en écrivant `$variable`.

Lire et afficher

- La commande `echo` affiche un texte ou le contenu d'une variable dans la console.
- `read` attend une saisie au clavier de la part de l'utilisateur et stocke le résultat dans une variable.

Opérations

- On peut effectuer des opérations mathématiques sur des nombres à l'aide de la commande `let`.
- Certaines variables sont accessibles partout, dans tous les scripts : ce sont les variables d'environnement. On peut les lister avec la commande `env`.
- Les paramètres envoyés à notre script (comme `./script -p`) sont transmis dans des variables numérotées : `$1`, `$2`, `$3`… Le nombre de paramètres envoyés est indiqué dans la variable `$#`.

8. BOUCLES

On effectue des tests dans ses programmes grâce aux `if, then, [[ elif, then, fi] else,] fi`.

1. On peut comparer deux chaînes de caractères entre elles, mais aussi des nombres. On peut également effectuer des tests sur des fichiers : est-ce que celui-ci existe ? Est-il exécutable ? Etc.
2. Au besoin, il est possible de combiner plusieurs tests à la fois avec les symboles`&&`(ET) et`||`(OU).
3. Le symbole`!`(point d'exclamation) exprime la négation dans une condition.
4. Lorsque l'on effectue beaucoup de tests sur une même variable, il est parfois plus pratique d'utiliser un bloc`case in… esac`plutôt qu'un bloc`if… fi`.

Pour exécuter une série de commandes plusieurs fois, on utilise des boucles.

5. `While` permet de boucler tant qu'une condition est remplie. Le fonctionnement des conditions dans les boucles est le même que celui des blocs`if`découverts dans le chapitre précédent.
6. `For` permet de boucler sur une série de valeurs définies. À l'intérieur de la boucle, une variable prend successivement les valeurs indiquées.

Un exemple tout simple

```
#!/bin/bash
        read  -p 'Entrez 2 noms de fichiers : '  fic1 fic2
if [ $fic1 -nt $fic2 ]
        then
                echo "fichier1 plus recent"
        else
                echo "fichier2 plus recent"
fi
```

Ce petit script compare lequel des deux fichiers est le plus récent.

9. PLANIFIER L'EXECUTION DE SCRIPTS

Cron est un planificateur de tâches périodiques pour les sytèmes de type Linux

https://itx-technologies.com/fr/blog/2259-executer-un-script-recurrent-avec-cron-exemples-sous-linux

10. DES EXEMPLES DE SCRIPTS DE SAUVEGARDE

Avec copie et compression de fichiers

https://www.memoinfo.fr/tutoriels-linux/script-sauvegarde-linux/

http://www.responsive-mind.fr/script-sauvegarde-shell/