Département Informatique

R1.04 – TP Admin System 8

Responsables : X.Roirand, F. Lesueur, N. Le Sommer, N. Delomez

Durée : 90mn machine

Le but de ce TP va être d'appréhender un peu plus le shell, et particulièrment les scripts utilisant le shell.

Pour pouvoir démarrer le TP, il vous faudra vous connecter sur une première machine distante, sur laquelle vous allez tous vous connecter, puis sur une seconde machine qui ne sera utilisée que par vous. Pour savoir quelle est cette seconde machine, il vous faudra un numéro unique, que l'enseignant vous donnera. Penez bien à changer le mot de passe de votre machine distante personnelle !

Prénom Nom :..... Date :.....
Groupe :.....

Instructions pour se connecter sur votre machine personnelle distante (vous devez avoir recu un numéro unique de la part de votre enseignant):

Connecter vous sur la sandbox dont l'adresse IP est 195.83.161.135. Le login est student et le mot de passe ?Student_56. Une fois sur cette sandbox, connectez-vous sur votre machine personnelle distante, pour cela il faut connaitre le numéro que le professeur vous a donné, et ajouter ce numéro au nom « machine » pour obtenir le nom de la machine sur laquelle vous devez vous connecter.

Exemple : si votre numéro est le 3 alors il faut faire un ssh sur la machine machine3
si votre numéro est le 54 alors il faut faire un ssh sur la machine machine54

L'utilisateur sur votre machine personnelle distante est "user" et le mot de passe est :

?user_56

Pourquoi utiliser bash ?

Bash est une version evoluée du shell sh (le "Bourne shell"). Le shell peut être utilisé comme un simple interpréteur de commande, mais il est aussi possible de l'utiliser comme langage de programmation interpreté (scripts).

La connaissance du shell est indispensable au travail de l'administrateur unix :

- le travail en "ligne de commande" est souvent beaucoup plus efficace qu'a travers une interface graphique
- dans de nombreux contextes (serveurs, systemes embarqués, liaisons distantes lentes) on ne dispose pas d'interface graphique
- le shell permet l'automatisation aisee des tâches répétitives (scripts)
- de tres nombreuses parties du systeme UNIX sont écrites en shell, il faut être capable de les lire pour comprendre et eventuellement modifier leur fonctionnement.

Autres versions de shell

Il existe plusieurs versions de shell : sh (ancêtre de bash), csh (C shell), ksh (Korn shell), zsh, etc. Nous avons choisi d'enseigner bash car il s'agit d'un logiciel libre, utilise sur toutes les

distributions recentes de Linux et de nombreuses autres variantes d'UNIX. Connaissant bash, l'apprentissage d'un autre shell sur le terrain ne devrait pas poser de difficultés

Shell ou Python ?

Nous avons vu qu'il etait possible d'écrire des programmes en shell. Pour de nombreuses tâches simples, c'est effectivement très commode. Neanmoins, le langage shell est forcément assez limité. Pour des programmes plus ambitieux, il est recommandé d'utiliser des langages plus evolués comme Python ou Perl, voir des langages compilés (C, C++) si l'on désire optimiser au maximum les performances (au prix de coûts de développement plus importants).

Les mauvais côtes des shell

Le shell possède quelques inconvénients:

- documentation difficile d'accès pour le débutant (la page de manuel "man bash" est très longue et technique) ;
- messages d'erreurs parfois difficiles à exploiter, ce qui rend la mise au point des scripts fastidieuse
- syntaxe cohérente, mais ardue (on privilégie la concision sur la clarté)
- relative lenteur (langage interpreté sans pré-compilation). Ces mauvais côtés sont compensés par la facilite de mise en oeuvre (pas besoin d'installer un autre langage sur votre système).

Scripts

Un script contient des séquences de commandes telles que l'on pourrait les taper dans un terminal. Les commandes successives peuvent être séparées par des retours a la ligne (qui sont interpretés comme des points virgules).

- Tout cfihier de script commence par une ligne permettant d'identifier le programme qui doit être utilisé pour l'exécuter. Dans le cas d'un script bash, la premiere ligne du fichier doit contenir :
  #!/bin/bash (verifiez le chemin de votre bash avec la commande "which bash")
- Une ligne commencant par le caractere: # est considérée comme un commentaire et n'est pas exécutee par le shell. Comme toujours, il est très important de bien commenter son script pour qu'il soit compréhensible pour le reste du monde.
- Par convention, l'extension d'un script est : .sh
- Pour qu'un utilisateur puisse exécuter un script, il doit posséder les droits en exécution, mais aussi en lecture sur ce script (c'est un cas particulier ou la lecture est necéssaire a l'exécution).
- Pour exécuter un script, on peut taper directement dans le terminal le chemin absolu du fichier, ou taper directement le chemin relatif du fichier en le faisant commencer par ./ ou taper la commande bash suivie d'un chemin du fichier.

Parametres

Comme vous l'avez remarqué, la plupart des commandes Unix peuvent être suivies d'un ou plusieurs paramètres, qui peuvent être des options, des noms de fichiers ou de répertoires, etc. Il est aussi possible de passer des paramètres à vos scripts. Pour les manipuler, il existe plusieurs variables spéciales. En particulier:

- la variable $# contient le nombre de paramètres passés au script
- pour chaque entier i entre 1 et 9, la variable $i contient le i-ème paramètre
- la variable $@ contient la liste de tous les paramètres séparés par des espaces
- la variable $0 contient le nom du programme en cours d'exécution

Instructions

If...then...elif...else...fi :

```
if liste1 ; then
commandes1
elif liste ; then
...
...
else
...
...
fi
```

Le new line est équivalent au ";"

Si le code de retour de liste1 est 0,le bloc de commandes1 est exécuté, sinon on passe a la suite.

On peut aussi l'utiliser sous sa forme la plus simple:

```
if liste ; then
...
```

```
...
fi
```

while...do...done :

```
while liste ; do
...
...
done
```

Exemple :

```
while [ $# -ge 1 ] ; do
... # traitement de $1
shift
done
```

for...in...do...done :

```
for var in mot1 mot2 mot3 ;do ...
...
done
```

Exemple :

```
Prog
For var in 1 2 3 4 ;do
echo $var
done
```

```
%prog
1
2
3
4
%
```

```
ensemble=''element1 element2 element3''
...
...
for var in $ensemble ; do
...
...
done
```

Remarque :

```
for var in $* ;do
... # traitement de $var
done
est equivalent a
for var ;do
...
done
```

Les tests

Test de fichiers

Si le fichier existe et... :

-r :est lisible
-w :l'écriture est possible
-x :exécutable

Exemple :

```
if [ -r $2 ] ;then
...
...
else
echo "$0 :vous n'avez pas le droit de lire le fichier $2">&2 fi
```

-f :est un fichier ordinaire
-d :est un répertoire
-p :est une représentation interne d'un dispositif de communication
-c :est un pseudo-fichier du type accès caractère par caractère
-b :est un pseudo-fichier du type accès par bloc
-L :est un lien symbolique
-u : son Set UID=1
-g :son Set GID=1
-k :son Sticky Bit=1
-s :est non-vide et existe

Tests de chaines

test chaîne (ou [chaîne]) : vraie si chaîne est une chaîne vide

XAV

-z cha^ne :vraie si cha^ne est une cha^ne vide
-w cha^ne :vraie si cha^ne est une cha^ne non-vide

Tests binaires

chaine1 = chaine2 : vraie si chaine1 est egale a chaine2
chaine1 != chaine2 : vraie si chaine1 n'est pas egale a chaine2
n1 -eq n2 : vraie si n1 est egal a n2
n1 -ne n2 : vraie si n1 est di erent de n2
n1 -gt n2 : vraie si n1 est plus grand strictement a n2
n1 -ge n2 : vraie si n1 est plus grand ou egal a n2
n1 -lt n2 : vraie si n1 est plus petit strictement a n2
n1 -le n2 : vraie si n1 est plus petit ou egal a n2

Fonctions

L'interêt d'une fonction est que l'on peut en mettre plusieurs dans un script afin de gérer des repétitions de commandes (ou de groupes de commandes).

Déclaration :

```
nom(){
...
... ;
}
ou
nom(){.... ;}
```

A l'intérieur d'une fonction il est possible d'utiliser l'instruction "return n", cela permet de quitter la fonction avec le code de retour n.

Appel :

```
% nom argument1 argument2 ... argumentn
```

Note:

Vous pouvez utiliser l'URL http://shellcheck.net afin de vérifier la syntaxe de vos script shell pour les shells de type Bourne Shell (sh) ou avec:

bash -n <votre_script>

pour les shells de type bash

Exercice : paramètres

Q1) Ecrivez un script analyse.sh qui affiche :
Bonjour, vous avez rentré nombre de parametres parametres.
Le nom du script est nom du script
Le 3eme parametre est 3eme parametre
Voici la liste des parametres : liste des parametres

Bien sr, "nombre de paramètres", "nom du script" et "3ème paramètre" sont à remplacer par leurs vraies valeurs.

Copiez le script que vous avez écrit ici en-dessous:

Exercice : vérification du nombre de paramètres

Q2) Ecrivez un script concat.sh qui prend en paramètre 2 mots et fait ce qui suit.

- si l'utilisateur rentre autre chose que 2 paramètres, indique a l'utilisateur qu'il doit rentrer exactement 2 parametres, et quitte en renvoyant une erreur.
- sinon le script calcule dans une variable CONCAT la concaténation des 2 mots rentrés puis affiche le resultat.

Pour tous les exercices suivants vous verifierez systématiquement le nombre de paramètres.

Copiez le script que vous avez écrit ici en-dessous:

Exercice : argument type et droits

Q3) Créez un script test-fichier, qui précisera le type du fichier passe en parametre, ses permissions d'accès pour l'utilisateur, ou s'il n'existe pas.

Exemple de résultat:

Le fichier /etc est un répertoire "/etc" est accessible par root en lecture écriture exécution

Le fichier /etc/smb.conf est un fichier ordinaire qui n'est pas vide "/etc/smb.conf" est accessible par jean en lecture.

Copiez le script que vous avez écrit ici en-dessous:

Exercice : Afficher le contenu d'un repertoire

Q4) Ecrire un script bash listedir.sh permettant d'afficher le contenu d'un répertoire en séparant les fichiers et les (sous)répertoires.

Exemple d'utilisation :

$ ./listdir.sh /boot

affichera :

####### fichier dans /boot/
/boot/config-3.16.0-4-amd64
/boot/initrd.img-3.16.0-4-amd64
/boot/System.map-3.16.0-4-amd64
/boot/vmlinuz-3.16.0-4-amd64
####### repertoires dans /boot/
/boot/grub

Copiez le script que vous avez écrit ici en-dessous:

Exercice : Lister les utilisateurs

Q5) Ecrire un script bash affichant la liste des noms de login des utilisateurs définis dans /etc/passwd ayant un UID superieur a 100.

Indication : for user in $(cat /etc/passwd); do echo $user; done permet preque de parcourir les lignes du dit fichier. Cependant, quel est le problème ? Résoudre ce probleme en utilisant cut (avec les bons arguments) au lieu de cat. Faites la même chose avec la commande awk.

Copiez le script que vous avez écrit ici en-dessous:

Exercice : Mon utilisateur existe t'il

Q6) Ecrire un script qui veri e si un utilisateur existe deja.
- en fonction d'un login passe en parametres
- en fonction d'un UID passe en parametres
Si l'utilisateur existe renvoyer son UID a l'a chage.
Sinon ne rien renvoyer.

Copiez le script que vous avez écrit ici en-dessous:

Exercice : Création utilisateur

Q7) Ecrire un script pour creer un compte utilisateur voir : man useradd

Utilisez votre script de vérification d'existance d'utilisateur avant de creer.
Il faudra vérifier que l'utilisateur en cours d'exécution est bien root voir echo $USER
Il faudra créer son home dans /home apres avoir verifier qu'il n'y a pas deja un répertoire portant le même nom.
Il faudra répondre à une suite de question : voir man read
- login
- Nom
- Prenom
- UID
- GID
- Commentaires

Copiez le script que vous avez écrit ici en-dessous:

Exercice : lecture au clavier

La commande bash read permet de lire une chaîne au clavier et de l'affecter a une variable. Essayer les commandes suivante :

```
echo -n "Entrer votre nom: "
read nom
echo "Votre nom est $nom"
```

La commande file affiche des informations sur le contenu d'un fichier (elle applique des règles basees sur l'examen rapide du contenu du chier).

Les fichiers de texte peuvent être affiche page par page avec la commande more (ou less, qui est légerement plus sophistiquée, car less is more...).

- Q8) Question Tester les trois commandes : read, file, more.

1. comment quitter more ?
2. comment avancer d'une ligne ?
3. comment avancer d'une page ?
4. comment remonter d'une page ?
5. comment chercher une chaîne de caractères ?
6. comment passer a l'occurence suivante ?

Q9) Ecrire un script qui propose à l'utilisateur de visualiser page par page chaque fichier texte du répertoire specifié en argument. Le script affichera pour chaque fichier texte (et seulement ceux la, utiliser la commande file) la question "voulez vous visualiser le fichier machintruc ?". En cas de reponse positive, il lancera more, avant de passer a l'examen du fichier suivant.

Copiez le script que vous avez écrit ici en-dessous:

Exercice : appreciation

Q10) Créez un script qui demande a l'utilisateur de saisir une note et qui affiche un message en fonction de cette note :
- "tres bien" si la note est entre 16 et 20 ;
- "bien" lorsqu'elle est entre 14 et 16 ;
- "assez bien" si la note est entre 12 et 14 ;
- "moyen" si la note est entre 10 et 12 ;
- "insuffisant" si la note est inferieur a 10.

Pour quitter le programme l'utilisateur devra appuyer sur q.

Copiez le script que vous avez écrit ici en-dessous:

Document tiré de:
Licence de Sciences et Technologies Mention Informatique
Systèmes/Services Unix M. Le Cocq - 17.01.2017