Les shell scripts

Introduction

Soit le fichier suivant

```
#!/bin/bash
echo "Entrez votre nom "
read nom
echo "vous vous appelez $nom."
```

Ce fichier qui contient des commandes du shell, écrites dans un ordre précis, est un shell script. On pourra l'invoquer de la façon suivante :
bash NomDuFichierScript

Autre solution, on rend le fichier exécutable, et lors de son invocation, il s'exécutera avec l'interpréteur indiqué en première ligne (ceci est vrai aussi pour d'autres langage interprété, comme PERL par exemple)
chmod +x LeFichierScript *(rend exécutable)*
./LeFichierScript *(pourquoi ./ ?)*

Remarquez les commandes
echo pour afficher, qui offre des options (man echo)
read qui permet d'affecter à une variable (non typée, non déclarée) la saisie de l'utilisateur.

les variables
Comme tout langage de programmation, le Shell permet de gérer des variables. Il y a :
des variables d'environnement (comme $PATH, $PS1.. Qui sont souvent définies par /etc/profile par exemple, et consultable avec le commande env)
des variables spéciales du shell : en général, créées par le programme shell lui même, elles sont en lecture seule.. par exemple $0, $# ou $$
les variables de l'utilisateur, c'est à dire vous, le programmeur.
Une variable n'a pas de type, et n'est pas déclarée. On s'en sert directement, en général par une affectation. Par exemple :
x=5
nom="Sylvain Cherrier"
ATTENTION : var est la variable, et $var est le contenu de la variable. Parfois, on utilisera ${var} pour indiquer son nom (notamment dans les substitutions ou de concatenation)
problème du non déclaré-non typé :
Possibilité d'erreur très important (mauvais nom sans alerte, confusion)
Pb du type : que vaut x=3 puis y=$x+2
La solution : y=$(($x+2))

Comment écrire un script ? Avec un éditeur de texte : **vi** ou **emacs** pour les furieux, **cat >LeScript** pour les tordus, et n'importe lequel pour commencer :**pico**, **nano**, ou un **éditeur graphique**.

Exercice Niveau 1 :

Créez un répertoire scripts, dans lequel vous rangerez vos scripts

Ex 1 : Créez un script Ex1 qui vous demande votre nom et votre prénom, et vous affiche 'vous êtes prénom nom'
ex 2 : Créez un script Ex2 qui vous demande votre nom et prénom (en une seule variable), puis votre mois de naissance et enfin votre année de naissance. Il affiche ensuite "Nom : xxxxx, né(e) en xx xxxx" suivi du calendrier de ce mois
Ex 3 : Écrire un script non interactif Ex3 qui crée un fichier /tmp/bilan, qui contient la date, le nom de la machine (uname -a), la liste des utilisateurs connectés, et la liste des processus. Vérifiez le bon fonctionnement de votre script.
Ex 4 : Écrire un script non interactif Ex4 qui affiche :
Nb users connectés : xx
Nb de processus : xxx
Ex 5 : Écrire un script qui vous demande un nom d'utilisateur, puis affiche les processus qui lui appartiennent
Ex 6 : Écrire un script qui crée dans /tmp un fichier qui contient le lastlog du jour.
Ex 7 : Améliorez le script précédent en faisant en sorte que le nom du fichier soit SAV-AA-MM-JJ (SAV suivi de la date du jour, calculée automatiquement)

Les paramètres

On peut passer des arguments à un script (comme à un programme). Soit le script Script1. Si on invoque
./Script1 test root "spike ne répond pas" fin
on a 4 arguments, qui seront directement récupérables dans le script dans des variables $0 à $n...
$0 vaut ./Script1
$1 vaut *test*
$2 vaut *root*
$3 vaut *spike ne répond pas*
$4 vaut *fin*
Nota : $# contient le nombre d'arguments
On ne peut pas modifier les valeurs $0 $1 $2 etc...
exemple d'utilisation : echo "le troisième argument est $3"
Enfin, sachez que l'on peut récupérer la valeur de retour d'une commande dans $?. La valeur de retour d'une commande indique si elle a bien fonctionné (0) ou non (<>0). Les valeurs sont documentées dans le man, en général. Par exemple, $@ contient la liste des parametres passés à votre shell-script

Les Alternatives

If commande1
then
.....
fi
permet de récupérer la valeur de retour de commande1 (pas son stdout, mais sa valeur de retour, ce que le main renvoie) afin de prendre un décision...
Ainsi, il est possible de tester le bon fonctionnement d'une majorité de commandes
ex

```
if   fdformat /dev/fd0
then
        echo "la disquette est formatée"
        #et un petit if imbriqué
        if  mkfs  /dev/fd0
        then
                echo "la disquette est maintenant utilisable"
        else
                echo "probleme à la création du FS"
        fi
else
        echo "la disquette est mauvaise"
fi
```

A noter que if se décide sur l'ensemble de la commande qui le suit, c'est à dire si il y a un tube, sur le dernier élément :
if cat fichier | grep reseau | grep arret
then ... (si il y a au moins une ligne contenant **arret** parmi celles contenant **reseau** dans le fichier fichier, alors...)

TEST
Test (ou [ ]) : Puisque if teste la valeur de retour d'un programme, il était impossible de faire un if sur des valeurs !
D'où l'écriture d'un programme test, qui, selon les arguments qu'on lui passe, donne des valeurs de retour. Test sait tester des fichiers, et comparer des valeurs
test -f fichier (renvoie vrai si le fichier fichier existe)
test -d répertoire (renvoie vrai si répertoire existe et est un répertoire)
test -s fichier (renvoie vrai si le fichier fichier existe, et si sa taille est >0)
test -x fichier (oui si le fichier est executable)

En ce qui concerne les chaînes de caractères (pour tester les saisies par exemple)
test chaine1 = chaine2 **test chaine1 != chaine2**

et pour les valeurs numériques
test val1 -eq val2 (val1 egal val2)
test val1 -gt val2 (val1 plus grand que val2)

test val1 -ne val2 (val1 différent de 2)
test val1 -lt val2 (val1 plus petit que val2)
test val1 -ge val2 (val1 plus grand ou égal à val2)
test val1 -le val2 (val1 plus petit ou égal à val2)

Nota : on peut élégamment remplacer test par [ ] : **if [$i -eq 10]**

Exercices Niveau 2

notions : Doit on écrire un crochet pour tester le résultat d'un cp ?
Comment expliquer en un mot la différence important entre le if d'un langage de programmation et le if d'un shell ?

- *Ex 1 : Écrire un script qui liste son nom et ses arguments (grâce à **echo**), et affiche le total d'arguments*
- *Ex2 : Écrire un script qui rend exécutable le fichier dont le nom est passé en argument (pratique pour les scripts, justement)*
- *Ex3 : Améliorez le script précédent, afin qu'il vérifie si le fichier existe bien. Il doit aussi indiquer si celui-ci était déjà exécutable.*
- *Ex4 : En cas de mauvaise invocation (oubli de l'argument, trop d'arguments), rappelez l'usage de votre programme (usage : monprog un_script_à_rendre_exécutable)*
- *Ex5 : file est une commande qui vous indique quel est le type d'un fichier (man file). Améliorez encore le programme précédent afin qu'il vérifie si l'argument qu'on lui passe est bien un script shell. Testez d'abord la commande file. Regardez ce qu'elle écrit sur son stdout. Cherchez un moyen de savoir si cela à un rapport avec un script. Et modifiez votre programme en conséquence.*
- *Ajoutez à votre programme la possibilité d'interroger son numéro de version, grâce à un argument -v (par exemple : monprog -v affiche Monprog, version 0.17, date : 17/03/2004, auteur : SC)*
- *Ex 6 Écrire une commande qui vérifie si le processus qu'on lui donne en argument est présent en mémoire. Testez ce programme. Un problème apparaît lorsque l'on demande un processus dont le nom n'existe pas. Vérifiez pourquoi. En consultant le man de grep, (et notamment l'option -v), modifiez votre script afin qui ne se trompe plus.*
- *Ex 7 : écrire une commande qui vérifie si le user dont on lui passe le nom en argument est bien connecté, et affiche aussi son nom complet (dans /etc/passwd, 5ème champ)*
- *Ex8 : écrire un programme qui ping monge.univ-mlv.fr. Si celui-ci répond, affichez 'monge répond', sinon affichez 'monge ne répond pas'. Ces deux seuls messages doivent apparaître.*

Boucles

le while

while programme
do
    instructions...
done

La aussi, on boucle tant que le programme renvoie vrai
Par exemple, que fait ceci

```
while ! [ -f /tmp/stop ]
do
        echo « je continue »
        sleep 1 # attente d'une seconde
done
```

Que fait ce programme ?
Comment l'arrêter ?

le for

le for du shell est simpliste, il sert à se prendre les valeurs d'une liste... Cette liste est souvent générée par une commande... Or, on conçoit souvent le for comme étant paramétré par une borne de début et de fin, avec un pas... Ici, pour la même opération, il faudra écrire

for i in 1 2 3 4 5 6 7
do
    instructions...
done

Souvent on utilise la substitution du shell (*, ? et autres caractères) qui nous renvoie la liste des fichiers correspondant

*for i in **
do
    file $i
done

Ou encore, on récupère le résultat d'un sous shell...

Les sous shell

Exprimés entre backquote ` `, ils permettent de lancer la commande backquotée, et de récupérer son résultat, qui devient alors l'argument de la commande en cours...
Exemples

var= `ps ax | wc -l`

ou encore

for i in ` who`
do
    //actions...
    ...
done

Exercices

Ex 1 : Récupérez le fichier access.log de mon site. Ce fichier est le log officiel d'apache, et contient toutes les informations demandées par les utilisateurs. Il est utilisé pour faire des statistiques sur les visites du site.
Écrivez un script qui sort le nombre de hits pour le répertoire ***images*** *pour plusieurs jours (comptez les hits du premier jour et affichez les, puis ceux du second jour et affichez les, etc..). (vous pouvez donner les dates en « dur ». Cependant, il est possible d'avoir la liste de tous les jours présents.. puis, pour chacun d'eux, compter le nombre d'accès au répertoire images)*
Ex 2 : amélioration du précédent, mais le nom du répertoire recherché est passé en paramètre par l'utilisateur.
Ex 3 : A quoi sert la commande ***seq*** *? Comment pourriez vous l'utiliser dans une boucle ? Ecrire un programme qui ping toutes les machines de 192.168.4.130 à 192.168.4.140.*

DIVERS

Il est possible d'afficher des informations par echo, et d'interroger l'utilisateur par read.
Cependant, on peut utiliser des outils plus puissants tels que dialog (mode texte curse, c'est à dire mieux présenté

Dialog est invoqué en précisant le titre de la fenêtre (grâce à title), et son type...
Par exemple, on peut demander une fenêtre yesno. On précisera en la fin la hauteur et la largeur de la fenêtre, en nombre de caractères.
exemple :
dialog --title ' reservation cantine' --yesno 'Mangez-vous à la cantine ?' 10 25

En analysant la valeur de retour ($?), vous saurez ce que l'utilisateur a cliqué (1 pour Oui, 0 pour non, et 255 pour escape)

Maintenant, récupérons une valeur avec la fenêtre de type inputbox : Ici, on redirigera la sortie (stderr) dans un fichier temporaire, afin de récupérer la valeur..

```
dialog --title "Ma fenêtre d'entrée" --clear \
        --inputbox "Bonjour, voici un exemple de fenêtre d'entrée\n
                    Essayez de saisir votre nom ci-dessous:" 16 51 2> $fichtemp

valret=$?

case $valret in
  0)
    echo "La chaîne d'entrée est `cat $fichtemp`";;
  1)
    echo "Appuyé sur Annuler.";;
  255)
    if test -s $fichtemp
    then
        cat $fichtemp
    else
        echo "Appuyé sur Echap."
```

```
    fi
    ;;
esac
```

Sed

Sed est un filtre un peu plus puissant, car il permet la programmation de son comportement. Il s'agit d'un mini éditeur de texte (en fait, il utilise les mêmes commandes que l'éditeur ed, et on peut retrouver des similitudes avec certaines commandes de vi).

commandes

Ainsi, la commande s permet de remplacer (substituer) des chaines par d'autres.
Sur un flux, on peut par exemple appliquer sed 's/bonjour/bonsoir/' qui remplacera par bonsoir la première occurrence du mot bonjour sur chaque ligne... (si on veut un remplacement multiple sur les lignes, il faut mettre g en fin de commande)
Pour faire exécuter à sed plusieurs commandes en une seule invocation, il faut utiliser l'option -e
sed -e 's/a/A/' -e 's/truc/bidule/g' -e 's/1000/2000/'

gestion des lignes avec sed

on peut limiter l'opération a certaines lignes, en l'indiquant avant la commande..
par exemple,pour remplacer truc par bidule uniquement entre les lignes 10 à 15
sed '10,15s/truc/bidule/'
On peut aussi donner des contenus pour borner les lignes
Par exemple :
cat /etc/passwd | sed '/oem/,/avahi/s/C/A/'
va remplacer par A le premier C de chaque ligne comprise entre la première ligne qui contient oem et celle qui contient avahi (l'ensemble du flux est affiché)

autres commandes

i pour insérer un texte avant la ligne
a pour ajouter après
q pour n'afficher que les x premières lignes
p pour afficher les lignes indiquées (avec sed -n !!!)
d pour supprimer les lignes indiquées

Exercices
afficher le contenu de /etc/passwd en remplacant le premier : par !
afficher le contenu de /etc/passwd en remplacant tous les : par !
Afficher les 10 premières lignes de /etc/passwd
afficher les lignes de 10 à 15 de ce même fichier
Afficher le contenu de ce fichier en insérant un ligne de tirets entre chaque ligne
Idem mais uniquement avant la 15ième ligne

Idem mais avant la ligne de l'utilisateur postfix

Awk

Awk est un outil très puissant pour gérer des flux, faire des remplacements, etc..C'est une sorte d'interpréteur de langage C. Il prend le flux entrant, et le considère comme des lignes ($1, $2, $3 sont les champs de ces lignes.. $0 l'ensemble des champs.. le tout selon le séparateur FS.. le numéro de ligne traitée est dans NR)
Awk dispose de trois section s : BEGIN, le corps, et END, toutes programmables. On peut faire des affectations de variables (avec =), afficher les variables (avec print).

tester les commandes suivantes :
cat /etc/passwd | awk '{print $1}'
Qu'obtenez vous ? Pourquoi pas seulement le champ 1 ?
cat /etc/passwd | awk '{FS=":" ; print $1}'
Qu'est ce que FS ?A quoi sert il?
Testez la commande suivante :
cat /etc/passwd | awk 'FS=":" {print NR,$1}'
Comment obtenir le nombre total de lignes (et uniquement cela)
comment obtenir :
liste des utilisateurs
1 root
2 bin
*****
42 sylvain

Il y a 42 utilisateurs.

les variables

On peut utiliser des variables. On peut par exemple les déclarer dans la section BEGIN, faire des opérations dessus dans le corps du programme, et afficher les valeurs dans la section END.
Par exemple, pour compter tous les utilisateurs du groupe 100, et ceux du groupe 0
cat /etc/passwd | awk 'BEGIN {FS=":"; cpt=0;gid=100} $4==gid {cpt++;} END {print "il y a "cpt" utilisateur dans le groupe "gid}'
Idem en comptant ceux du groupe 100 et ceux du groupe 0
cat /etc/passwd | awk 'BEGIN {FS=":"; cpt1=0;cpt2=0;gid1=100;gid2=0} $4==gid1 {cpt1++;} $4==gid2 {cpt2++} END {print "il y a "cpt1" utilisateur dans le groupe "gid1;print "il y a "cpt2" dans "gid2}'
AWK permet aussi d'utiliser les tableaux indicés ET associatifs.
Idem qu'au dessus en tableau indicé
cat /etc/passwd | awk 'BEGIN {FS=":"; gid1=100;gid2=0} {cpt[$4]++;} END {print "il y a

"cpt[gid1]" utilisateur dans le groupe "gid1;print "il y a "cpt[gid2]" dans "gid2}'
et en tableau associatif
cat /etc/passwd | awk 'BEGIN {FS=":"; gid1=100;gid2=0} $4==gid1 {cpt['users']++;} $4==gid2 {cpt['root']} END {print "il y a "cpt['users']" utilisateur dans le groupe users";print "il y a "cpt['root']" dans le groupe root"}'
Il est aussi possible d'utiliser toutes les structures de contrôle du C
Par exemple, pour lister toute les valeurs d'un tableau
awk 'BEGIN {for(i=0;i<10;i++) tab[i]=i*i;} END {for(i=0;i<10;i++) print "tab[i] vaut "tab[i];}'

exercices
Écrire un programme qui affiche le nom et l'uid des utilisateurs
Écrire un programme qui affiche la même chose dans l'ordre inverse.
Écrire un programme qui affiche le contenu de /etc/passwd en effacant le GID
Afficher tous les utilisateurs dont le UID est supérieur ou égal à 1000
Compter ces utilisateurs
Afficher les utilisateurs qui ont /bin/false en shell de connexion (attention aux caractères spéciaux... \ déspécialise... et $ exprime la fin de la ligne (regexp))
Compter les utilisateurs par shell de connexion.

Solution AWK

```
 cat /etc/passwd | awk  'BEGIN {FS=":";print "==============\nListe";}  { print NR," ",$1}END
{print "==========\n il y a ",NR," users";}'
```