Les Scripts

1 Qu'est-ce qu'un "shell" ?

Un "shell" est un programme permettant d'isoler l'utilisateur des commandes internes du système d'exploitation. Nous donnerons une description des principales commandes qui peuvent être utilisées dans le “c-shell” et dans le “bash”.

2 Variables définies par l'usager

Les variables peuvent être définies pour être utilisées dans vos commandes. Il existe deux types de variables:

- **variables du "shell"**
- **variables d’environnement**

Les principales commandes discutées ici seront:

echo
set

Commande Set:

Cette instruction permet, dans des commandes scripts, d'initialiser et d'utiliser des chaînes de caractères et des valeurs qui pourront ultérieurement être utilisées dans vos scripts.
Exemple:

```
set  orig=fichier1.c
set dest=fichierautre.c
set rename=mv
```

On pourra ensuite utiliser, soit dans un fichier de commandes scripts ou directement au "prompt", les différentes variables configurées avec **set**. Par exemple:

```
.....> rename  orig  dest
```

Cette commande utilise la variable rename qui est en fait la commande **mv** pour renommer le fichier d'origine avec le nom donné dans la variable destination.

Évidemment, l'exemple précédent n'est pas d'une grande utilité en pratique puisque nous pouvons effectuer la même chose avec des **alias**. La commande **set** permet tout simplement une initialisation d'une variable qui pourrait par la suite être utilisée dans un boucle, une condition ou un calcul quelconque.

Cet opérateur permet également d’assigner à une variable le résultat d’une commande.

Exemple:
La commande “who” permet de faire afficher les usagers actuellement branchés au système.

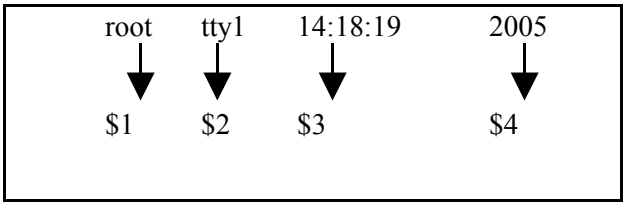
on tape: who
on obtient le résultat:
root tty1 14:13:18 1997
Ce qui veut dire qu’uniquement “root” est branché au système présentement.

Pour assigner le résultat de cette commande à une variable, on procède ainsi:

c-shell	bash
#!/bin/csh set w=`who` echo “$w[1] $w[2] $w[3] $w[4]”	#!/bin/bash set `who` echo “$1 $2 $3 $4”

Voyons ce qui s’est produit avec la ligne ***set `who`***

Chaque champ de la commande “**who**” s’est transporté directement dans les pré-définies du bash. Voici ce qui s’est passé plus schématiquement:



Opérations arithmétiques

Les opérations arithmétiques utilisent des opérateurs parmi les suivants:

Opérateurs	+, -, *, / et % (reste de la division) Exemples en bash: v=5 let v=v-2 let v=v*2 Exemples en cshell: set v=5 @ v=v+1 @ v=v/2
Opérations relationnelles	-lt (<) -gt (>) -le (<=) -ge (>=) -eq (==) -ne (!=)

Affichage du contenu d'une variable: Commande echo

La commande **echo** permet de faire afficher à l'écran:

toutes variables numériques définies au moyen de la commande **set**
toutes variables numériques définies au moyen de l'opérateur @
toutes chaînes de caractères.

Affichage d'une chaîne de caractères:

Une chaîne de caractères peut être affichée directement en utilisant la commande **echo** suivi de la chaîne à faire afficher:

echo "Bonjour tout le monde" ou echo Bonjour tout le monde
affiche la phrase correspondante à l'écran.

Affichage d'une variable numérque:

Toutes variables numériques peut être affichées en utilisant la commande echo et en faisant précéder la variable d'un signe de dollar "$".
Exemple:

```
set a=5            "en bourne shell"
@ j = 10           "en C-shell"
set chaine1 = "Bonjour tout le monde"

echo $a   #affiche le contenu de la variable a
echo $j  #affiche le contenu de la variable j
echo j          # affiche la lettre "j"
echo $chaine1  #affiche le contenu de la chaîne de caractères "chaine1"
```

Affichage d'une variable numérique à l'ntérieur d'une chaîne de caractères:

Les chaînes de caractères sont contenues à l'intérieur de double guilemet " ou d'apostrophe ' . Ainsi, si on reprend la variable chaine1 telle que définie précédemment à la page , nous obtenons les cas suivants:

```
echo  $chaine1  #affichage du contenu de la variable chaine1
echo "$chaine1"           # affichage du contenu de la variable chaine1
echo '$chaine1'  # affichage de la chaîne "$chaine1"
```

Ainsi les variables qui doivent être affichées à l'intérieur de chaînes de caractères suivent les trois cas précédent dépendemment de ce qui doit être affiché. Par exemple:

si a=5, chaine="Bonjour tout le monde"

On veut obtenir:	***Commande à taper:***
Le contenu de la variable est 5	echo Le contenu de la variable est $a ou echo "Le contenu de la variable est $a"

On veut obtenir:	*Commande à taper:*
Le contenu de la variable $a vaut 5	echo "Le contenu de la variable \$a vaut $a"
Le contenu de chaine vaut:Bonjour tout le monde	echo "Le contenu de chaine vaut $chaine"
Le contenu de $chaine vaut Bonjour tout le monde	echo "Le contenu de \$chaine vaut $chaine"

3 Les variables pré-définies

Il existe un moyen rapide de connaître le nombre de variables et le contenu des variables qui sont passées en paramètre à la ligne de commande. Comme vous avez pu le constater, les paramètres sont nommés $0 à $9 dans le système d'exploitation DOS. En Unix, les paramètres passées à la ligne de commande ont aussi leurs syntaxes et ont des noms différents dépendemment que vous travaillez dans un shell ou un autre.

3.1 Variables pré-définies du "BASH" (Bourne Again Shell)

$0 à $9	Les variables qui contiennent les paramètres passés à la ligne de commande.	$0 : Le nom de la commande $1 : Le premier paramètre $2 : Le deuxième paramètre etc...
$*	Donne la liste des paramètres de la commande.	echo $* Affiche la liste des paramètre $1, $2, $3, etc...
$?	Donne la valeur du code de retour de la dernière commande exécutée.	
$#	Donne le nombre de paramètre de la commande appelée (sauf $0).	ls -al fichier.c echo $# Affiche 2

Exemple:

Supposons le fichier "essai" qui contient les lignes suivantes:

```
echo $#
echo $*
```

Supposons que l'on tape ensuite la commande suivante:

```
essai  Bonjour  Allo  Salut  2  4
```

Le résultat sera:

5 echo $# donne le nombre de paramètre excluant la commande elle-même.

Bonjour Allo Salut 2 4

3.2 Variables pré-définies du « C-Shell »

$argv	Un tableau qui contient les paramètres passés à la ligne de commande. Cette variable est accessible sous la forme d'un tableau de variable.	$argv[1] : Le premier paramètre $argv[2] : Le deuxième paramètre etc...
$#argv	Donne le nombre de paramètre(s) passé(s) à la commande.	echo $#argv Affiche le nombre de paramètre présent dans la commande.

Exemple:

Soit le fichier qui contient les lignes suivantes:

```
#!/bin/csh
#Fichier script en c-shell qui affiche les parametres passes a la commande
echo "Il y a $#argv parametre(s) a la commande.  Les voici:  "
echo -n "le premier parametre: $argv[1]"
echo -n "Le deuxieme parametre: $argv[2]"
```

On sauvegarde le fichier sous le nom essai2 et on tape ensuite au prompt:

```
essai2  Allo  Bonjour
```

Le script affiche:

```
Il y a 2 parametre(s) a la commande.  Les voici:
Le premier parametre: Allo
Le deuxieme parametre: Bonjour
```

4 Étape dans la création d'un fichier script

Étape 1:

- Éditez le script avec votre éditeur de texte préféré. Tapez les lignes ci-dessous dans votre fichier.

```
#!/bin/csh
#Ce scripts affiche les arguments passés à la commande par le biais
# de la variable pré-définie  $argv.
#Script en bash
#
echo  "Il y a $#argv arguments sur la ligne de commande"

if ($#argv > 0)  then     # Le nombre d'argument est donne par: $#argv
   echo  "Le premier argument: $argv[1]"
endif
if ($#argv > 1) then
   echo "Le deuxieme argument: $argv[2]"
endif
```

Sauvegardez le fichier sous un nom qui deviendra ainsi le nom de la nouvelle commande. Pour cet exemple, sauvegardez sous le nom "***script1***"

Étape 2:

- Ajoutez l'attribut d'exécution à votre fichier script. (Cette opération ne se fait qu'une seule fois)

chmod u+x script1

Étape 3:

- Exécutez le script en tapant le nom du fichier et ses paramètres s'il y a lieu:

./script1 Salut Allo

C'est bel et bien la suite de caractères « ./ » qu'il faut mettre devant le nom.

L'exécution du script précédent avec la ligne de commande suivante:

script1 bonjour allo

donne

Il y a 2 arguments sur la ligne de commande
Le premier argument: bonjour
Le deuxième argument: allo

Exercices:

1. Réalisez un script bash nommé « lc » qui affichera le contenu du répertoire avec la couleur. (Souvenez-vous de la commande : ls –al --colors)

2. Faire un script bash nommé « script2 » qui permet de faire afficher la date du jour sur une ligne et sur la ligne suivante les usagers branchés au système. Souvenez-vous des commandes « date » et « who ».

3. Créez un script nommé "script2" en utilisant la syntaxe « bash » pour que celui-ci affiche les lignes suivantes:

   Il y a 3 arguments sur la ligne de commande
   Le premier argument: $1
   Le deuxieme argument: $2
   Le troisième argument: $3

4. Réaliser un script qui se nommera **"list"** et qui réalisera les possibilités suivantes :

   - Lorsque l'on tape : **list**
     Le script doit faire afficher le contenu du répertoire.
     (Commande ls –al)

   - Lorsque l'on tape : **list -a nom_fichier**
     Le script doit faire afficher le contenu du fichier nommé « nom_fichier » passé en paramètre. (Commande cat)

   - Lorsque l'on tape : **list -d nom_fichier**
     Le script doit détruire le fichier passé en paramètre. (Commande rm)

   Vous devez aussi planifier les possibilités suivantes :

   - L'usager entre un nombre plus grand de paramètre.
   - L'usager n'entre aucun paramètre.

   Dans les deux cas précédent, la syntaxe de la commande doit être affichée.

5. Écrire un script bash nommé « calc » qui accepte 2 chiffres en paramètre, additionne ces deux chiffres et affiche le résultat. Ainsi, si on tape :

Calc 3 5	calc 3
On reçoit : 8	On reçoit : Nombre invalide de paramètre
Calc 1 -7	calc 3 5 6
On reçoit : -6	On reçoit : Nombre invalide de paramètre

Structure de contrôle

Dans cette section, il sera question des structures de contrôle principales que l'ont retrouve dans les scripts. Ces structures de contrôles permettent d'ajouter des conditions aux scripts et de tester l'existence de certains paramètres pour ensuite effectuer certaines actions.

Vous allez étudier surtout 3 structures de contrôle. Il s'agit de:

- **if**
- **while**
- **for**

Vous retrouverez la syntaxe de ces commandes autant pour le "c-shell" que pour le "bash" shell.

La structure IF

	Syntaxe en C-shell	Syntaxe en bash

Structure IF simple	**if** (condition(s)) **then**
	Liste de commande(s)
	else
	Liste de commande(s)
	endif
		if [condition(s)]; **then**
		Liste de commande(s)
		else
		Liste de commande(s)
		fi

Structure IF imbriquée	**if** (condition(s)) **then**
	Liste de commande(s)
	else
	if (condition(s)) **then**
	Liste de commande(s)
	else
	Liste de commande(s)
	endif
	endif
		if [condition(s)]; then
		Liste de commande(s)
		else
		if [condition(s)]; then
		Liste de commande(s)
		else
		Liste de commande(s)
		fi
		fi

Exemple:

Le script suivant permet de simuler 3 commandes en unes. Voici les 3 possibilités que cette commande doit permettre:

1. list — Affiche le contenu du répertoire courant
2. list -a fichier — Affiche le contenu du fichier passé en 2e paramètre.
3. list -d fichier — Détruit le fichier passé en 2e paramètre.

c-shell	*bash*

if ($#argv == 0) then
  ls -al
else
  if (“$argv[1]” == “-a”) then
  cat $argv[2]
  else
  rm $argv[2]
  endif
endif
	if [$# = 0]; then
	  ls -al
	else
	  if [$1 = -a]; then
	  cat $2
	  else
	  rm $2
	  fi
	fi

Test de fichiers avec la structure IF

Il est souvent pratique de pouvoir tester les attributs d’un fichier pour savoir si ce dernier est exécutable, existe ou n’existe pas, etc...

Test en c-shell ou en bash	*Résultat*
-e fichier	Teste l’existance d’un fichier.
-d répertoire	Teste l’existance d’un répertoire.
-r fichier	Teste si le fichier peut être lu.
-w fichier	Teste si le fichier peut être modifié.
-x fichier	Teste si le fichier peut être exécuté.
-c fichier	Teste si le fichier est un fichier spécial de type “caractère”.
-b fichier	Teste si el fichier est un fichier spécial de type “bloc”.
-s fichier	Teste si le fichier existe et est de taille non nulle.

Exemple:

On veut faire afficher la phrase “C’est un repertoire” si le fichier passé en paramètre est un répertoire.

c-shell

```
if (-d $argv[1] ) then
  echo “$argv[1] est repertoire”
else
  echo “$argv[1] n’est pas un repertoire”
endif
```

bash

```
if [ -d $1 ]; then
  echo “$1 est un repertoire”
else
  echo “$1 n’est pas un repertoire”
fi
```

La structure SWITCH-CASE

Cette structure, tout comme en langage C, permet de sélectionner une branche d'action selon la valeur d'une variable. Voici la syntaxe:

c-shell	bash

switch ($variable)
  case valeur**:**
  liste de commande(s)
  breaksw
  case valeur2**:**
  liste de commande(s)
  breaksw
  .
  .
  .
  default:
  liste de commande(s)
endsw
	case $variable **in**
	  valeur**)**
	  liste de commande(s)
	  ;; #breaksw en c-shell
	  valeur2**)**
	  liste de commande(s)
	  ;;
	  .
	  .
	  .
	  ***)**
	  Liste de commande(s) #valeur défaut
	esac

Exemple :

L'exemple suivant montre l'utilisation de ce type de structure: il s'agit d'une commande qui attend au plus 1 paramètre et dans le cas où:

il n'y a pas de paramètre, affiche le répertoire courant (pwd)
un paramètre, affiche le contenu s'il s'agit d'un fichier et affiche le contenu du répertoire si c'est un répertoire.

c-shell	***bash***

#!/bin/csh
switch ($#argv)
  case 0:
  pwd
  breaksw
  case 1:
  if (-f $argv[1]) then
  cat $argv[1]
  else if (-d $argv[1]) then
  ls -al $argv[1]
  else
  echo "Erreur sur le parametre"
  endif
 	
  default:
  echo "$argv[1] n'est ni un rep ni un fichier"
endsw
	#!/bin/bash
	case $# in
	  0)
	  pwd
	  ;;
	  1)
	  if [-f $1]; then
	  cat $1
	  else
	  if [-d $1]; then
	  ls -al $1
	  else
	  echo "erreur sur parametre"
	  fi
	  fi
	  ;;
	  *) echo "$1 n'est ni un rep ni un fichier"
	  ;;
	esac

Structure de répétition

La structure while

Cette instruction permet de répéter un certain nombre de fois, dicté par une condtion, la liste de commandes qui se retrouve à l'intérieur de la boucle.

Syntaxe:

c-shell	bash
while (condition) liste de commande(s) **end**	**while** [condition(s)]; do liste de commande(s) **done**

Exemple:

c-shell	bash
#!/bin/csh set v=5 while ($v > 1) echo "Toujours plus grand que 1 : v = $v" @ v-- #l'espace entre le @ et la variable est #important end	#!/bin/bash v=5 while [$v -gt 1]; do echo "Toujours plus grand que 1 : v = $v" let v=v-1 done

La structure for

Syntaxe:

c-shell	bash
foreach variable (liste de valeur(s)) liste de commande(s) **end**	**for** variable **in** chaine chaine ... **do** liste de commande(s) **done**

Exemple 1 :

L'exemple suivant permet de trouver le fichier donné en parametre et qui porte l'extension .o, .cpp ou .c et afficher si le fichier existe ou non.

c-shell	bash
#!/bin/csh	#!/bin/bash for i in .o .cpp do if [-e 1i]; then echo "Fichier existe" else echo "Fichier n'existe pas!" fi done

Exemple 2:

```
!/bin/bash
    for i in $( ls ); do
      echo item: $i
    done
```

Lecture d'une entrée au clavier

Les données entrées au clavier peuvent être lues par le biais de la commande "**read**" pour le **bash shell** et du caractère "$<" pour le **c-shell**.

Supposons que vous voulez demander à l'usager de confirmer le remplacement d'un fichier. Vous pourrez alors le faire de la façon suivante dans les deux shells suivants:

c-shell	bash
#!/bin/csh echo -n "Voulez-vous remplacer le fichier ? (o/n): " set remplace = $< # ????????????? Quelle instruction va ici ?	#!/bin/bash fichier=allo.txt echo -n "Voulez-vous remplacer le fichier ? (o/n): " **read** remplace if [$remplace = o]; then rm $fichier fi

Les alias

Il s'agit d'un mécanisme qui permet l'écriture de commandes par la définition d'abréviations et évitant de taper des commandes qui pourraient être longue. En ce sens, ce mécanisme ressemble à la notion de macro-instructions dans les langages de programmation. Autrement dit, l'alias que vous créer devient ainsi une sorte de synonyme pour la commande que vous rattachez à cet alias.

Habituellement, les alias permanents seront situés dans le fichier de configuration « .bashrc » dans le répertoire maison de l'usager.

Syntaxe:

c-shell	bash
alias nom_de_l'alias 'commande'	alias nom_de_l'alias = 'commande'

Exemple:
On veut créer des alias pour les commandes suivantes:
find -iname que l'on nommera "f"
rm -r que l'on nommera "deltree"

c-shell	bash
alias f ‘find -iname’ alias deltree ‘rm -r’	alias f='find -iname' alias deltree='rm -r'

Exemple #2:

Il arrive souvent que l’on veuille faire afficher le répertoire courant dans le “**prompt**” sur la ligne de commande. Voici les alias à faire:

c-shell	bash
alias cd ‘cd \!*;set prompt = “$cwd” ’	alias cd = 'cd \!*;set prompt=`pwd`'

Les points-virgules permettent de séparer deux commandes de suite sur une même ligne.
La variable “**prompt**” est un nom réservé pour l’attribution du prompt.
La suite de caractères “\!*” permet d’utiliser un paramètre lorsque cette commande sera tapée. Comme vous le savez, la commande cd a besoin d’un nom de répertoire pour se déplacer dans ce répertoire. Or, les caractères \!* seront remplacés par le nom du répertoire qui sera tapé sur la ligne de commande.

Exemple: cd travail_pratique grâce à l’alias la commande précédente devient:

cd \!*;set prompt=“$cwd” ce qui se traduit par la séquence des deux commandes:

cd travail_pratique
set prompt=“$cwd”

Le paramètre “travail_pratique” prend directement la place des caractères \!* dans la commande.

Alias utiles

Deux autres alias sont fréquemment utilisés. Il s'agit de "**popd**" et de "**pushd**". Ces deux alias sont déjà créer pour vous automatiquement par le système.

pushd

syntaxe: pushd nom_répertoire
Description: Permet de changer de répertoire et de sauvegarder sur la "pile", le nom du répertoire précédent.
Exemple: Si on tape, à partir du répertoire "/usagers/user1/tp2" la commande
pushd ../tp1
La commande pushd effectue exactement ce que la commande cd exécute, donc on change de répertoire et on se déplace dans /usagers/user1/tp1 et du même coup, on sauvegarde le répertoire d'origine "/usagers/user1/tp2" sur la "pile".

popd
Syntaxe: popd
Description: Permet de revenir au répertoire qui a été le dernier a être placé sur la "pile".

Exemple: En tapant popd tout de suite après l'exemple du "pushd" précédent, on revient au répertoire "/usagers/user1/tp1".

Exercices récapitulatifs

Niveau Débutant

1. Nommez 2 variables pré-définies du **c-shell**

2. Nommez 2 variables pré-définies du **bash** shell.

3. Construire une script "**c-shell**" qui affiche la liste des usagers branchés au système et la date du jour :
   "Il y a xx usagers branchés presentement" où xx représente le nombre d'usager branché.
   On obtient la liste des usagers branchés avec la commande « who ».

4. Construisez un alias qui permet de connaître le nombre d'usager branché au système en tapant la commande « users ».

5. Construisez un alias qui permet d'obtenir le contenu d'un répertoire en tapant la commande « dir ».

Niveau Intermédiaire

6. Faire un **script** qui permet de compter le nombre de fichier que vous pouvez exécuter dans un répertoire donné

7. Réaliser une nouvelle commande que vous nommerez "**list**" et qui en fera un peu plus que la commande « ls ». Voici ce que votre script doit réaliser:

   - faire afficher la liste du contenu du répertoire à l'écran et à la fin
   - faire afficher le nombre de répertoire rencontré,
   - faire afficher le nombre de fichier rencontré.

Niveau intermédiaire, avancé

8. Faire une **script** qui permet de simuler une calculatrice. La commande se nommera **calc** et aura 3 paramètres qui sont les suivants:

calc <opérande 1> <opérande 2> <opérateur>

où

opérande 1 et opérande 2 représente les chiffres sur lesquels on doit effectuer l'opération

opérateur est l'une des opérations suivante

+	addition
-	soustraction
#	multiplication
/	division

Ainsi: calc 2 5 # affichera 10

9. Vous devez faire un script en utilisant le « bash » pour permettre la création de plusieurs comptes d'usager sur le système. Pour ce faire, vous lirez un fichier texte qui contient le nom du compte ainsi que le mot de passe. Le format de ce fichier est le suivant:

nom_compte:mot_de_passe:groupe
nom_compte:mot_de_passe:groupe
.
.
.

Il s'agit donc d'isoler le nom du compte de même que le mot de passe et le nom du groupe pour ensuite utiliser les commandes d'administration relatives à la création de comptes.

Par exemple, si le fichier se nomme « liste.txt » alors ou peut procéder ainsi pour isoler le nom du compte :

compte=`cut -f1 -d : liste.txt`

L'option « -f » permet de choisir le numéro de la colonne pour laquelle on veut obtenir la valeur.

Dans le fichier ayant le format suivant:

Nom:Prenom:Telephone

La colonne 1 est celle du nom.
La colonne 2 est celle du prénom.
La colonne 3 est celle du numéro de téléphone.

Ainsi, si je veux obtenir le numéro de téléphone, je procède ainsi:

cut -f « numéro de la colonne » -d « caractère délimiteur » « nom_fichier ».

Avec l'exemple précédent, la commande devient:

cut -f3 -d : liste.txt