Les scripts SHELL - Niveau 1

Le shell est l'interpréteur de commandes de Linux. Il ne fait pas partie du noyau, mais forme une couche supplémantaire, d'où son nom de "coquille".

Il a un rôle d'analyse des commandes saisies puis les fait exécuter par le système.

Il existe plusieurs shells, tous partageant des points communs :

- bourne again shell (bash),
- korn shell (ksh),
- c shell (csh),
- etc.

Le bash est présent par défaut sur les distributions Linux, il se caractèrise par ses fonctionnalités pratiques et conviviales.

Le shell est aussi un langage de programmation basique qui grâce à quelques commandes dédiées permet :

- L'utilisation de variables,
- l'exécution conditionnelle de commandes,
- la répétition de commandes.

Les scripts en shell ont l'avantage d'être réalisables rapidement et de manière fiable, sans compilation ni installation de commandes supplémentaires.

Pour écrire un script shell, il suffit de réunir dans un même fichier toutes les commandes nécessaires, une commande par ligne.

En rendant ce fichier exécutable, le shell le lira séquentiellement et exécutera une à une les commandes le comportant.

Lorsque le shell rencontre une erreur, il affiche un message permettant d'identifier le problème mais continue l'exécution du script.

Les erreurs propres aux commandes sont également affichées à l'écran.

Qu'est ce qu'un bon script ? C'est un script :

- fiable : son fonctionnement est irréprochable même en cas de mauvaise utilisation ;
- commenté : son code est annoté pour en faciliter la relecture et les futures évolutions;
- lisible : le code est indenté à bon escient, une seule commande par ligne, les commandes sont aérées…
- portable : le code fonctionne sur tout système Linux, gestion des dépendances, gestion des droits, etc.

1. Premier script

Pour commencer l'écriture d'un script shell , il est pratique d'utiliser un éditeur de texte gérant la coloration syntaxique.

vim est un outil adapté a cela.

Le nom du script devra respecter quelques règles :

- pas de majuscule,
- pas de nom de commandes existantes,
- extension en .sh pour indiquer qu'il s'agit d'un script shell.

hello-world.sh.

```
#!/bin/bash
#
# Auteur : Antoine Le Morvan
# Date : 18 septembre 2016
# Version 1.0.0 : Affiche le texte "Hello world !"
#

# Affiche un texte à l'écran :
echo "Hello world !"
```

Pour pouvoir exécuter ce script, il est nécessaire de lui donner le droit d'exécution :

```
stagiaire $ chmod u+x ./hello-world.sh
stagiaire $ ./hello-world.sh
Hello world !
```

La première ligne à écrire dans tout script permet d'indiquer le shell à utiliser pour l'exécuter. Si vous désirez utiliser le shell ksh ou le langage interprété python, vous remplaceriez la ligne :

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

par :

```
#!/bin/ksh
```

ou par :

```
#!/usr/bin/python
```

Tout au long de l'écriture, il faudra penser à la relecture du script en utilisant notamment des commentaires :

- un commentaire général en début pour indiquer le but du script, son auteur, sa version, etc.
- des commentaires au cours du texte pour aider à la compréhension des actions.

Les commentaires peuvent être placés sur une ligne à part ou bien à la fin d'une ligne contenant une commande.

Exemple :

```
# Ce programme affiche la date
date        # Cette ligne est la ligne qui affiche la date !
```

2. Variables

Comme dans tout langage de programmation, le script shell utilise des variables. Elles servent à stocker des informations en mémoire pour les réutiliser à volonté au cours du script.

Une variable est créée au moment où elle reçoit son contenu. Elle reste valide jusqu'à la fin de l'exécution du script. Puisque le script est exécuté

séquentiellement du début à la fin, il est impossible de faire appel à une variable avant qu'elle ne soit créée.

Le contenu peut être modifié au cours du script, la variable continue d'exister. Si le contenu est supprimé, la variable reste active mais ne contient rien.

Il n'y a pas de notion de type de variable en script shell. Le contenu d'une variable est toujours un caractère ou une chaîne de caractères.

01-backup.sh.

```
#!/bin/bash

#
# Auteur : Antoine Le Morvan
# Date : 18 septembre 2016
# Version 1.0.0 : Sauvegarde dans /root les fichiers passwd, shadow,
 group et gshadow
#

# Variables globales
FICHIER1=/etc/passwd
FICHIER2=/etc/shadow
FICHIER3=/etc/group
FICHIER4=/etc/gshadow

# Dossier destination
DESTINATION=/root

# Nettoie l'écran :
clear

# Lancer la sauvegarde
echo "La sauvegarde de $FICHIER1, $FICHIER2, $FICHIER3, $FICHIER4 vers
 $DESTINATION va commencer :"

cp $FICHIER1 $FICHIER2 $FICHIER3 $FICHIER4 $DESTINATION

echo "La sauvegarde est terminée !"
```

Ce script fait usage de variables. Le nom d'une variable doit commencer par une lettre mais peut ensuite contenir n'importe quelle suite de lettres ou de chiffres. Hormis le tiret bas "_", les caractères spéciaux ne sont pas utilisables.

Par convention, les variables créées par un utilisateur ont un nom en minuscules. Ce nom doit être choisi avec précaution pour n'être ni trop évasif ni trop compliqué. Une variable peut toutefois être nommée avec des majuscules, comme c'est le cas ici, s'il s'agit d'une variable globale qui ne doit pas être modifiée par le programme.

Le caractère "=" affecte du contenu à une variable :

```
variable=valeur
nom_rep="/home"
```

Il n'y a pas d'espace ni avant ni après le signe "=".

Pour afficher du texte en même temps que le contenu d'une variable, il est obligatoire d'utiliser les guillemets et non les apostrophes.

L'usage des apostrophes inhibe l'interprétation des caractères spéciaux.

```
stagiaire $ message="Bonjour"
stagiaire $ echo "Voici le contenu de la variable
 message : $message"
Voici le contenu de la variable message : Bonjour
stagiaire $ echo 'Voici le contenu de la variable
 message : $message'
Voici le contenu de la variable message : $message
```

Pour isoler le nom de la variable, il faut utiliser les apostrophes ou les accolades :

```
stagiaire $ touch "$fichier"1
stagiaire $ touch ${fichier}1
```

2.1. Supprimer et verrouiller les variables

La commande **unset** permet de supprimer une variable.

Exemple :

```
stagiaire $ nom="NOM"
stagiaire $ prenom="Prenom"
```

```
stagiaire $ echo "$nom $prenom"
NOM Prenom
stagiaire $ unset prenom
stagiaire $ echo "$nom $prenom"
NOM
```

La commande readonly verrouille une variable.

Exemple :

```
stagiaire $ nom="NOM"
stagiaire $ readonly nom
stagiaire $ nom="AUTRE NOM"
bash: nom: variable en lecture seule
stagiaire $ unset nom
bash: nom: variable en lecture seule
```

2.2. Variables d'environnements

Les variables d'environnements et les variables systèmes sont des variables utilisées par le système pour son fonctionnement. Par convention elles portent un nom en majuscules.

Elles peuvent être affichées ou modifiées dans un script comme n'importe quelle variable. Elles doivent cependant être modifiées avec précaution.

La commande **env** permet d'afficher toutes les variables d'environnement utilisées.

La commande **set** permet d'afficher toutes les variables système utilisées.

Parmi les dizaines de variables d'environnement, plusieurs ont un intérêt à être utilisées dans un script shell :

Table 1. Variables d'environnement

Variable	Observation
HOSTNAME	Nom d'hôte de la machine
USER, USERNAME et LOGNAME	Nom de l'utilisateur connecté sur la session
PATH	Chemin des commandes

Variable	Observation
PWD	Répertoire courant, mis à jour à chaque exécution de la commande cd.
HOME	Répertoire de connexion.

2.3. Exporter une variable

La commande **export** permet d'exporter une variable.

Une variable n'est valable que dans l'environnement du processus du script shell. Pour que les **processus fils** du script puissent connaître les variables et leur contenu, il faut les exporter.

La modification d'une variable exportée dans un processus fils ne peut pas remonter au processus père.

Sans option, la commande **export** affiche le nom et les valeurs des variables exportées dans l'environnement.

2.4. La substitution de commande

Il est possible de stocker le résultat d'une commande dans une variable.

Cette opération n'est valable que pour les commandes qui renvoient un message à la fin de leur exécution.

La syntaxe pour sous-exécuter une commande est la suivante :

Syntaxes pour la substitution de commandes.

```
variable=`commande`
variable=$(commande)
```

Exemples :

```
stagiaire $ jour=`date +%j`
stagiaire $ homedir=$(pwd)
```

2.5. Améliorations du script de sauvegarde

Quelques pistes d'améliorations.

```
#!/bin/bash

#
# Auteur : Antoine Le Morvan
# Date : 18 septembre 2016
# Version 1.0.0 : Sauvegarde dans /root les fichiers passwd, shadow,
 group et gshadow
# Version 1.0.1 : Création d'un répertoire avec le quantième du jour.
#            Améliorations diverses

# Variables globales

## Fichiers a sauvegarder
FICHIER1=/etc/passwd
FICHIER2=/etc/shadow
FICHIER3=/etc/group
FICHIER4=/etc/gshadow

## Dossier destination
DESTINATION=/root

## Variables en readonly
readonly FICHIER1
readonly FICHIER2
readonly FICHIER3
readonly FICHIER4
readonly DESTINATION

# Un nom de dossier avec le quantieme du jour
rep="backup-$(date +%j)"

# Nettoie l'écran :
clear

# Lancer la sauvegarde
echo "*********************************************************************"
echo "     Script de sauvegarde - Sauvegarde sur la machine $HOSTNAME "
echo "*********************************************************************"
echo "La sauvegarde sera faîte dans le dossier ${rep}."
echo "Création du répertoire..."
mkdir -p $DESTINATION/$rep
```

```
echo "                                                    [ OK ]"
echo "La sauvegarde de ${FICHIER1}, ${FICHIER2}, ${FICHIER3},
 ${FICHIER4} vers ${DESTINATION}/$rep va commencer :"

cp $FICHIER1 $FICHIER2 $FICHIER3 $FICHIER4 $DESTINATION/$rep

echo "La sauvegarde est terminée !"

# La sauvegarde est notée dans le journal d'évènements du système :
echo "La sauvegarde est renseignée dans syslog :"
logger "Sauvegarde des fichiers systèmes par ${USER} sur la machine
 ${HOSTNAME} dans le dossier ${DESTINATION}/$rep."
echo "                                                    [ OK ]"
```

Exécution de notre script de sauvegarde.

```
root # ./02-backup-enhanced.sh
************************************************************
     Script de sauvegarde - Sauvegarde sur la machine formateur1
************************************************************
La sauvegarde sera faîte dans le dossier backup-262.
Création du répertoire...
                                                    [ OK ]
La sauvegarde de /etc/passwd, /etc/shadow, /etc/group, /etc/gshadow
 vers /root/backup-262 va commencer :
La sauvegarde est terminée !
La sauvegarde est renseignée dans syslog :
                                                    [ OK ]
```

Le lancement de la commande peut être visualisée dans le journal syslog :

Evènement dans syslog.

```
root # tail -f /var/log/messages
sept. 18 19:35:35 formateur1 antoine[9712]: Sauvegarde des fichiers
 systèmes par antoine sur la machine formateur1 dans le dossier /root/
b...
```

3. Saisie et manipulations

Selon l'objet du script, il peut être nécessaire d'envoyer des informations en cours d'exécution.

Ces informations, non connues lors de l'écriture du script, peuvent être extraites à partir de fichiers ou saisies par l'utilisateur.

Il est aussi possible d'envoyer ces informations sous forme d'arguments lors de la saisie de la commande du script. C'est le mode de fonctionnement de nombreuses commandes Linux.

3.1. La commande read

La commande **read** permet de saisir une chaîne de caractère pour la stocker dans une variable.

Syntaxe de la commande read.

```
read [-n X] [-p] [-s] [variable]
```

Exemple de la commande read.

```
stagiaire $ read nom prenom
stagiaire $ read -p "Veuillez saisir votre nom : " nom
```

Table 2. Options de la commande read

Option	Observation
-p	Affiche un message de prompt
-n	Limite le nombre de caractères à saisir
-s	Masque la saisie

Lors de l'utilisation de l'option **-n**, le shell valide automatiquement la saisie au bout du nombre de caractères précisés. L'utilisateur n'a pas à appuyer sur la touche [ENTREE].

```
stagiaire $ read -n5 nom
```

La commande **read** permet d'interrompre l'exécution du script le temps que l'utilisateur saisisse des informations. La saisie de l'utilisateur est découpée en mots affectés à une ou plusieurs variables prédéfinies. Les mots sont des chaînes de caractères séparées par le séparateur de champs.

La fin de la saisie est déterminée par la frappe sur la touche **[ENTREE]** ou le caractère spécial de fin de ligne.

Une fois la saisie validée, chaque mot sera stocké dans la variable prédéfinie.

Le découpage des mots est défini par le caractère séparateur de champs. Ce séparateur est stocké dans la variable système **IFS** (*Internal Field Separator*).

```
[root] # set | grep IFS
IFS=$' \t\n'
```

Par défaut, l'IFS contient l'espace, la tabulation **\t** et le saut de ligne **\n**.

Utilisée sans préciser de variable, cette commande met simplement le script en pause. Le script continue son exécution lors de la validation de la saisie.

Cette utilisation permet de faire une pause lors du débogage d'un script ou pour inciter l'utilisateur à appuyer sur **[ENTREE]** pour continuer.

```
[root]# echo -n "Appuyer sur [ENTREE] pour continuer..."
[root]# read
```

3.2. La commande cut

La commande **cut** permet d'isoler une colonne dans un fichier.

Syntaxe de la commande cut.

```
cut [-cx] [-dy] [-fz] fichier
```

Syntaxe de la commande cut.

```
[root]# cut -d: -f1 /etc/passwd
```

Table 3. Options de la commande cut

Option	Observation
-c	Spécifie les numéros d'ordre des caractères à sélectionner
-d	Spécifie le séparateur de champs
-f	Spécifie le numéro d'ordre des colonnes à sélectionner

Le principal intérêt de cette commande sera son association avec la commande grep et le pipe **|**.

La commande **grep** travaille verticalement (*isolation d'une ligne parmi toutes celles du fichier*).

La combinaison des deux commandes permet d'isoler un champ précis du fichier.

Syntaxe de la commande cut.

```
[root]# grep "^root:" /etc/passwd | cut -d: -f3
0
```

Les fichiers de configurations comportant une structure unique utilisant le même séparateur de champs sont des cibles idéales pour cette combinaison de commandes.

3.3. La commande tr

La commande **tr** permet de convertir une chaîne de caractères

Syntaxe de la commande tr.

```
tr [-csd] chaîne1 chaîne2
```

Table 4. Options de la commande cut

Option	Observation
-c	Tous les caractères qui ne sont pas spécifiés dans la première chaîne sont convertis selon les caractères de la seconde.
-d	Efface le caractère spécifié
-s	Réduire à une seule unité le caractère spécifié

```
[stagiaire]$ tr -s " " < /etc/hosts
```

Exercice : extraire le niveau d'exécution du fichier /etc/inittab

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

```
#
# Auteur : Antoine Le Morvan
# Date : 18 septembre 2016
# Version 1.0.0 : Extrait le niveau d'exécution du fichier /etc/inittab

# Variables globales

INITTAB=/etc/inittab

niveau=`grep "^id" $INITTAB | cut -d: -f2`

# Affichage du résultat :
echo "Le niveau d'init au démarrage est : $niveau"
```

3.4. Extraire le nom et le chemin d'un fichier

La commande **basename** permet d'extraire le nom du fichier à partir d'un chemin. La commande **dirname** permet d'extraire le chemin parent d'un fichier.

Exemple :

```
[root]# echo $FICHIER=/usr/bin/passwd
[root]# basename $FICHIER
passwd
[root]# dirname $FICHIER
/usr/bin
```

3.5. Arguments d'un script

La demande de saisie d'informations grâce à la commande **read** interrompt l'exécution du script tant que l'utilisateur ne fait pas de saisie.

Cette méthode, bien que très conviviale, présente des limites s'il s'agit d'un script à l'exécution programmée la nuit par exemple. Afin de palier ce problème il est possible d'injecter les informations souhaitées via des arguments.

De nombreuses commandes Linux fonctionnent sur ce principe.

Cette façon de faire à l'avantage qu'une fois le script exécuté, il n'aura plus besoin d'intervention humaine pour se terminer.

Son inconvenient majeur est qu'il faudra prévenir l'utilisateur du script de sa syntaxe pour éviter des erreurs.

Les arguments sont renseignés lors de la saisie de la commande du script. Ils sont séparés par un espace.

```
[root]# ./script argument1 argument2
```

Une fois exécuté, le script enregistre les arguments saisis dans des variables prédéfinies : les variables positionnelles.

Ces variables sont utilisables dans le script comme n'importe quelle autre variable, à l'exception faite qu'elles ne peuvent pas être affectées.

Les variables positionnelles non utilisées existent mais sont vides.

Les variables positionnelles sont toujours définies de la même façon :

Table 5. Les variables positionnelles

Variable	Observation
$0	contient le nom du script tel qu'il a été saisi.
1 à **$9**	contiennent les valeurs du 1er et du 9ème argument.
${x}	contient la valeur de l'argument **x**, supérieur à 9.
$#	contient le nombre d'arguments passés.
$*	contient en une variable tous les arguments passés

Exemples :

```
[root]# ./script.sh un deux trois
[root]# echo $3 $2 $1
trois deux un
[root]# echo $0 $# $*
./script.sh 3 un deux trois
```

Attention : il existe une autre variable positionnelle, **$@**, qui contient tous les arguments passés. La confusion avec **$*** est aisée.

> La différence se fait au niveau du format de stockage des arguments : **$*** : Contient les arguments au format "$1 $2 $3 …" **$@** : Contient les arguments au format "$1" "$2" "$3" …

La commande shift

La commande **shift** permet de décaler les variables postionnelles.

Exemples :

```
[root]# ./script.sh un deux trois
[root]# echo $1
un
[root]# shift 2
[root]# echo $1
trois
```

> Attention : Lors de l'utilisation de la commande **shift**, les variables **$#** et **$*** sont modifiées en conséquence.

La commande set

La commande **set** découpe une chaîne en variables positionnelles.

Syntaxe de la commande set.

```
set [valeur] [$variable]
```

Exemple :

```
[root]# set un deux trois
[root]# echo $1 $2 $3 $#
un deux trois 3
[root]# variable="un deux trois"
[root]# set $variable
[root]# echo $1 $2 $3 $#
un deux trois 3
```

Ci-dessous, la version de notre script de sauvegarde mettant en oeuvre les variables positionelles :

```
#!/bin/bash

#
# Auteur : Antoine Le Morvan
# Date : 18 septembre 2016
# Version 1.0.0 : Sauvegarde dans /root les fichiers passwd, shadow,
 group et gshadow
# Version 1.0.1 : Création d'un répertoire avec le quantième du jour.
#    Améliorations diverses
# Version 1.0.2 : Modification pour utiliser les variables
 positionnelles
#                 Limitation à 5 fichiers

# Variables globales

## Dossier destination
DESTINATION=/root

# Un nom de dossier avec le quantieme du jour
rep="backup-$(date +%j)"

# Nettoie l'écran :
clear

# Lancer la sauvegarde
echo "*********************************************************************"
echo "     Script de sauvegarde - Sauvegarde sur la machine $HOSTNAME "
echo "*********************************************************************"
echo "La sauvegarde sera faîte dans le dossier ${rep}."
echo "Création du répertoire..."
mkdir -p $DESTINATION/$rep
echo "                                                    [ OK ]"
echo "La sauvegarde de ${1} ${2} ${3} ${4} ${5} vers ${DESTINATION}/$rep
 va commencer :"

cp $1 $2 $3 $4 $5 $DESTINATION/$rep

echo "La sauvegarde est terminée !"
```