

# Scripting sous Linux

## Principes

Les scripts sont un moyen d'automatiser des actions à répéter pour de nombreuses données, en évitant toute interaction.

Pour fonctionner, ils doivent :

- disposer d'une source de données nécessaires à leur action : on utilise les fichiers CSV (comma separated values ou valeurs séparées par des virgules) qui doivent comporter
  - sur une ligne, chaque information nécessaire au traitement d'un objet (un groupe, un compte, un dossier, la liste des droits à appliquer sur un dossier)
  - autant de lignes que d'objets à traiter
- parcourir cette source grâce à la boucle while ... done
- exécuter les commandes en ligne en remplaçant les valeurs par les données issues du CSV

## Illustration

```
#!/bin/bash
echo off
while [IFS=<séparateur>] read champ1 champ2
do
  echo "Ceci est le champ n°1: " $champ1
  echo "Et voici le champ 2 : " $champ2
done< [</chemin>]<fichierAire>
```

- **#! /bin/bash** : pour afficher la syntaxe en couleur
- **echo off** : pour ne pas afficher les commandes qui s'exécutent au fur et à mesure
- **while ... read champ1 champ2** : boucle de parcours qui précise la structure du contenu de chaque ligne
- **IFS=<séparateur>** : option qui permet de définir le séparateur des champs dans un fichier structuré (CSV). Le séparateur est généralement un des caractères : , ;. L'espace est le séparateur par défaut
- **Do** : début du code à exécuter dans la boucle
- **\$champ1** : variable contenant la première information de la ligne
- **echo -e** : echo réalise un affichage écran, -e permet d'ajouter un retour à la ligne
- **done** : fin de la boucle
- **<** : indique qu'on prend le contenu de ce qui suit
- **[</chemin>]<fichierAire>** : fichier dont on exploite le contenu (avec chemin éventuel)

## Variabes

Les variables dans les scripts se déclarent par leur nom, et s'utilisent avec le préfixe \$.

Exemple :

```
#il n'y a pas d'espace entre la variable et le caractère "="
mavariable="bonjour"
echo $mavariable
```

La concaténation de chaîne dans une variable s'écrit par la simple juxtaposition des portions de texte :

Exemple :

```
laDate=$(date +%Y%m%d)
nomFichier="fichier"$laDate".csv"
```

## Gestion des dates

On peut gérer l'affichage des dates avec la commande **date** : La syntaxe est différente selon les distributions (tester avec ou sans les ""):

```
date +"<format>"
#ou
date +<format>
```

Le format utilise les paramètres suivants :

- %d : affiche le jour
- %m : affiche le mois
- %y : affiche l'année sur deux caractères (18 pour 2018)
- %Y : affiche l'année sur quatre caractères (2018)
- %H : affiche l'heure
- %M : affiche les minutes
- %S : affiche les secondes

Exemples :

```
date +"%d/%m/%y"
```

Affichera 13/02/18.

```
date +"%Y-%m-%d"
```

Affichera 2018-02-13

## Exemple 1 : création de groupes en masse

Pour créer des groupes en masse :

- on utilise la commande **addgroup <nomgroupe>** qui prend donc en paramètre le nom d'un groupe.

- La source de données est donc un fichier ne contenant sur chaque ligne que le **nom du groupe**.

Voici la source de données dans le fichier donneesgroupes.csv:

```
groupe1
groupe2
groupe3
```

Le script doit parcourir ce fichier et pour chaque ligne utiliser la commande addgroup.

```
#!/bin/bash
#!/bin/sh
# Boucle qui lis le fichier "donneesgroupes.csv" et cree les groupes
correspondant
while IFS=, read legroupe
do
  groupadd $legroupe
  echo $legroupe "est ajoute"
done < donneesgroupes.csv
```

Sur la **boucle WHILE**

- l'information **IFS=** indique le caractère qui sépare les champs (ici **,** même s'il n'y en a pas besoin)
- après **read**, on indique le nom de la ou des variables qui prendront le contenu d'une ligne. Ici, il n'y a qu'une variable **legroupe** puisqu'il n'y a qu'une donnée sur la ligne.
- Après **done <** on indique le nom du fichier qui contient les données **donneesgroupes.csv**

## Exemple 2 : Script avec fichier de réponse

Certaines commandes Linux nécessitent une interaction. Par exemple, la création d'utilisateur demande de multiples informations :

```
root# adduser unutil
Adding user `unutil' ...
Adding new group `unutil' (1008) ...
Adding new user `unutil' (1008) with group `unutil' ...
Creating home directory `/home/unutil' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for unutil
Enter the new value, or press ENTER for the default
  Full Name []:
  Room Number []:
  Work Phone []:
  Home Phone []:
```

```
Other []:
Is the information correct? [Y/n] Y
```

On doit indiquer le mot de passe (2 fois), les 5 informations de l'utilisateur et valider par Y.

Un script demandera donc d'interagir pour fournir ces éléments.

Une astuce est de stocker les informations nécessaires dans le fichier CSV, puis, dans le script, de créer un **fichier de réponses temporaire** avec les données issues du CSV et de demander l'exécution de la commande interactive en se fournissant dans le fichier de réponse.

## Illustration

```
#!/bin/bash
#!/bin/sh
echo on
# Boucle qui lis le fichier "donneesutils.csv" et cree les utilisateurs correspondant dans les bons groupes
while IFS=, read lecompte lepass legroupe
do
#cree un fichier de reponse
#mot de passe. La premiere ligne cree le fichier responses.txt (>), les autres ajoutent a son contenu (>>)
echo $lepass > responses.txt
echo $lepass >> responses..txt
#infos du compte
echo >> responses.txt
#validation
echo "Y" >> responses.txt
#execution de la commande d'ajout avec les donnees presentes dans le fichier reponse.txt
adduser $lecompte --ingroup $legroupe <reponse
echo $lecompte "est ajoute"
#suppression des reponses
rm responses.txt
done < donneesutils.csv
```

Ici on a intégré le compte directement dans un groupe (**-ingroup**) pour éviter la création d'un groupe au nom de l'utilisateur

From:

<https://wiki.sio.bts/> - **WIKI SIO : DEPUIS 2017**



Permanent link:

<https://wiki.sio.bts/doku.php?id=scriptlinux&rev=1666012115>

Last update: **2022/10/17 13:08**