

Introduction

- Bash □ Shell (interpréteur de commandes)
- Destiné à Linux mais aussi à Windows (intégré dans Windows 10)
- Objectif □ Création de scripts.

Langages de scripting

- Powershell
- Python
- Bash

Shebang :

- représenté par **#!**
- Se situe au début de chaque script (1ere ligne)
- Indique au système d'exploitation que ce fichier n'est pas un fichier binaire mais un script
- Sur la même ligne est précisé **l'interpréteur** permettant d'exécuter ce script
- **#** □ commandes
- **!** □ chemin qui mène à l'interpréteur **# !/bin/bash**
- Bash n'est pas le seul interpréteur
- Il en existe d'autres **# !/bin/sh ; # !/bin/csh ; # !/bin/zsh**
- Bash est implémenté par défaut sous Debian
- Par défaut le Shell est capable d'interpréter le script sous Linux car il est en bash. S'il s'agit d'un autre interpréteur cette 1ere ligne sera indispensable. Exemple : **# !/usr/bin/python**

Comment rendre un script exécutable ?

- Il faut mettre les droits sur le fichier **sh** (contenant le script) à tous les utilisateurs
- Exemple : **chmod a+x script.sh**
- Le **chmod a+x** permet de rajouter l'exécution sur tous les utilisateurs pour le fichier **script .sh**

Choix :

- Soit nous nous trouvons dans le répertoire où se situe le script (**chemin relatif**) ./script.sh
- Soit nous n'y sommes pas (**chemin absolu**) /home/.../script.sh

Les variables

- Espace de stockage qui possède un nom
- On associe au nom qui peut varier d'un script à un autre **NOM_DE_LA_VALEUR=« Valeur »**
- Les variables sont sensibles à la casse □ par convention □ MAJUSCULES
- **Attention : pas d'espaces entre la variable, le signe = et les « »**

TOUJOURS FAIRE chmod a+x POUR AVOIR LES PERMISSIONS.

Les commentaires

- Les commentaires sont précédés par **#** et ne sont pas interprétés

```
#!/bin/bash
echo "Vous êtes bien en BTS SIO"

#Ceci est un commentaire
#Ceci ne sera pas interprété
```

```

root@buster:~# chmod a+x BTSSIO.sh
root@buster:~# ./BTSSIO.sh
Vous êtes bien en BTS SIO
root@buster:~# _

```

- Pour utiliser les variables et afficher le contenu associé, il faut faire précéder le nom de la variable par un \$

```

#!/bin/bash
PRENOM="Matteo"
NOM="SIMON"
echo "Bonjour $PRENOM $NOM et bienvenu"

```

```

root@buster:~# ./BTSSIO.sh
Bonjour Matteo SIMON et bienvenu
root@buster:~# _

```

- Pour utiliser une variable dans un mot, il faut utiliser les {}.

```

#!/bin/bash
PRENOM="Matteo"
NOM="SIMON"
AGE="18"
echo "Bonjour $PRENOM ${NOM}, vous avez ${AGE}ans."

```

```

root@buster:~# ./script1.sh
Bonjour Matteo SIMON, vous avez 18ans.
root@buster:~# _

```

- Il est possible d'assigner la sortie standard d'une commande à une variable.

Il existe 2 solutions (L'une des deux peut ne pas marcher)

- Mettre le contenu entre parenthèses \$()

```

root@buster:~# MACHINE=$(hostname)
root@buster:~# echo $MACHINE
buster
root@buster:~# _

```

- Mettre le contenu entre '(ALT GR +7)'

```

root@buster:~# MACHINE=`hostname`
root@buster:~# echo $MACHINE
buster
root@buster:~# _

```

- Autre exemple :

```

root@buster:~# LISTE=`ls -l`
root@buster:~# echo $LISTE
total 8 -rwxr-xr-x 1 root root 81 mars 22 15:42 BTSSIO.sh -rwxr-xr-x 1 root root 98 mars 22 15:56 script1.sh
root@buster:~#

```

Attention à la syntaxe des variables. Quelques règles...

- Les variables ne peuvent commencer par un chiffre mais en contenir



Exemple : 1BONNEVARIABLE

Exemple : UNEBONNE8VARIABLE

- Les variables ne peuvent pas contenir de tiret –

Exemple : ~~UNE-BONNE-VARIABLE~~

Exemple : UNE_BONNE_VARIABLE

- De manière générale les variables ne peuvent contenir que des underscores, majuscules et chiffres.

Les conditions

Le if

- Si l'on veut effectuer un test afin de savoir si une condition est vraie, on utilise **if**.
- Si la condition est vraie, le script va exécuter les commandes tapées le **then**.
- Le **fi** marque la fin de la condition.

```
if [ condition-est-vraie ]
then
    command
    command2
fi
```

ATTENTION aux espaces avant et après les []

- Si l'on veut agir
- Cette commande de test avec **-e** permet de savoir si le fichier **hello.sh** existe bien

```
if [ -e /home/nometudiant/hello.sh ]
```

```
root@buster:~# cd home
root@buster:~/home# touch mattek.sh
root@buster:~/home# ls
mattek.sh nometudiant1
root@buster:~/home# if [ -e mattek.sh ]
> then
> echo "existe"
> else
> echo "existe pas"
> fi
existe
root@buster:~/home# _
```

Pour voir si le fichier existe bien

```
root@buster:~/home# FICHER="/home/mattek.sh"
root@buster:~/home# if [ -e $FICHER ]
> then
> echo "existe"
> else
> echo "existe pas"
> fi
existe pas
```

- Condition supplémentaire avec elif contraction de else if
- Possibilité de demander au script d'exécuter des commandes en fonction de la validité d'une ou de l'autre condition

```
root@buster:~# if [ condition-est-vraie ]
> then
> echo "reponse a"
> elif [ condition-est-vraie ]
> then
> echo "reponse b"
> else
> echo "reponse c"
> fi
reponse a
root@buster:~# _
```

```
#!/bin/bash
CHIFFRE1='16'
CHIFFRE2='17'
if [ $CHIFFRE1 -lt $CHIFFRE2 ]
then
    echo "$CHIFFRE1 est plus petit que $CHIFFRE2"
elif [ $CHIFFRE1 -gt $CHIFFRE2 ]
then
    echo "$CHIFFRE1 est plus grand que $CHIFFRE2"
else
    echo "$CHIFFRE1 est égal à $CHIFFRE2"
fi
```

Lt plus petit que

Gt plus grand que

```
root@buster:~/home/matteo# chmod a+x hello.sh
root@buster:~/home/matteo# ./hello.sh
16 est plus petit que 17
root@buster:~/home/matteo#
```

```
root@buster:~/home/matteo# ./hello.sh
18 est plus grand que 17
root@buster:~/home/matteo# _
```

```
root@buster:~/home/matteo# ./hello.sh
17 est égal à 17
root@buster:~/home/matteo#
```

On peut faire un **read -p** pour saisir une valeur.

La boucle for

```
for VARIABLE in OBJET1 OBJET2 OBJET3 OBJETn
do
    command
    comand2
done
```

- Si on veut effectuer un certain nombre d'actions sur une liste d'objets
- Après la variable `in` liste d'objets
- `do=then` dans les conditions
- On ferme toujours avec `done`

Exemple :

```
#!/bin/bash
for CHIFFRE in 10 11 12 13
do
echo "Chiffre: $CHIFFRE"
done
```

```
root@buster:~/home/matteo# chmod a+x script_boucle.sh
root@buster:~/home/matteo# ./script_boucle.sh
Chiffre: 10
Chiffre: 11
Chiffre: 12
Chiffre: 13
root@buster:~/home/matteo# _
```

Autre exemple :

```
#!/bin/bash
CHIFFRES="10 11 12 13"
for CHIFFRE in $CHIFFRES
do
echo "Chiffre: $CHIFFRE"
done
```

```
root@buster:~/home/matteo# ./script_boucle2.sh
Chiffre: 10
Chiffre: 11
Chiffre: 12
Chiffre: 13
root@buster:~/home/matteo# _
```

Encore un autre exemple :

```
#!/bin/bash
COULEURS="rouge vert noir bleu"
for COULEUR in $COULEURS
do
echo "La couleur actuelle est : $COULEUR"
done
```

```

root@buster:~/home/matteo# ./script_boucle3.sh
La couleur actuelle est : rouge
La couleur actuelle est : vert
La couleur actuelle est : noir
La couleur actuelle est : bleu
root@buster:~/home/matteo#

```

Les paramètres de position

- On lance un script avec les valeurs suivantes :
 - \$0 :script.sh
 - \$1 :argument1
 - \$2 :argument2

Qu'est ce qu'une variable de position :

- Les variables de position **stockent** le contenu des différents éléments de la ligne de commande utilisée pour lancer un script
- Il en existe 10 : \$0 à \$9

Exemple concret :

```

#!/bin/bash
echo "Le premier argument a pour valeur $1"
echo "Le deuxieme argument a pour valeur $2"

```

```

root@buster:~/home/matteo# chmod a+x script_position.sh
root@buster:~/home/matteo# ./script_position.sh
Le premier argument a pour valeur
Le deuxieme argument a pour valeur
root@buster:~/home/matteo# _

```

Il n'y a pas de valeur car on n'a pas défini les arguments.

```

root@buster:~/home/matteo# ./script_position.sh argument1 argument2
Le premier argument a pour valeur argument1
Le deuxieme argument a pour valeur argument2
root@buster:~/home/matteo# _

```

- Il est possible d'accéder à toutes les variables de position à partir de \$1 grâce à @\$

```

#!/bin/bash
echo "Le premier argument à pour valeur $1"
echo "Le deuxieme argument à pour valeur $2"
echo "Les arguments ont pour valeur @$_"

```

```

root@buster:~/home/matteo# chmod a+x script_position2.sh
root@buster:~/home/matteo# ./script_position2.sh argument1 argument2
Le premier argument à pour valeur argument1
Le deuxieme argument à pour valeur argument2
Les arguments ont pour valeur argument1 argument2

```

- \$# : récupère le nombre de paramètres (à partir du \$1)
- \$* : récupère la liste des paramètres = @\$

```
#!/bin/bash
echo " Le premier argument à pour valeur $1"
echo "Le deuxieme argument à pour valeur $2"
echo "Les arguments ont pour valeur $@"
echo "Le nombre d'arguments est de $#"
```

```
echo "Les arguments ont pour valeur : $*"
_

root@buster:~/home/matteo# chmod a+x script_position3.sh
root@buster:~/home/matteo# ./script_position3.sh argument1 argument2
Le premier argument à pour valeur argument1
Le deuxieme argument à pour valeur argument2
Les arguments ont pour valeur argument1 argument2
Le nombre d'arguments est de 2
Les arguments ont pour valeur : argument1 argument2
root@buster:~/home/matteo# _
```

La commande read :

```
#!/bin/bash
echo "Quel est le prenom ?"
read PRENOM
echo "Votre prenom est : $PRENOM"
```

```
root@buster:~/home/matteo# chmod a+x script1.sh
root@buster:~/home/matteo# ./script1.sh
Quel est le prenom ?
Matteo
Votre prenom est : Matteo
root@buster:~/home/matteo# _
```

Autre exemple :

```
#!/bin/bash
read -p "Quel est votre prenom ?" PRENOM
echo "Votre prenom est : $PRENOM"
_
```

```
root@buster:~/home/matteo# ./script1.sh
Quel est votre prenom ?Matteo
Votre prenom est : Matteo
root@buster:~/home/matteo# _
```

La boucle while

- Tant qu'un test n'est pas vérifié, l'exécution continue
- On debut avec while
- Do = then
- La boucle se termine par done

Le code retour

- Valeur renvoyée à chaque fois qu'une commande est exécutée
- Valeur allant de 0 à 255 car codée sur 8 bits
- Dans de nombreux langages ☐ commande correctement exécutée ☐ 0
- Si différent de 0 ☐ erreur de l'exécution

Utiliser la commande man ls pour plus d'indications

```
Exit status:
  0      if OK,

  1      if minor problems (e.g., cannot access subdirectory),

  2      if serious trouble (e.g., cannot access command-line argument).
```

Utiliser la commande `ls -la`

Echo \$? Vous dira si la commande précédente s'est bien passée ou non

```
root@buster:~# ls -la
total 28
drwx----- 4 root root 4096 mars  29 16:11 .
drwxr-xr-x 18 root root 4096 janv.  7 2020 ..
-rw----- 1 root root  383 avril  1 08:57 .bash_history
-rw-r--r-- 1 root root  570 janv. 31 2010 .bashrc
drwxr-xr-x  3 root root 4096 mars  29 16:11 home
drwxr-xr-x  3 root root 4096 janv.  7 2020 .local
-rw-r--r-- 1 root root  148 août  17 2015 .profile
root@buster:~# echo $?
0
root@buster:~#
```

Possibilité de vérifier si une opération s'effectue

```
root@buster:~# [ -e /home/matteo/hello ]
root@buster:~# echo $?
1
root@buster:~#
```

Retour 0 si le fichier existe

Retour 1 si le fichier n'existe pas

Les opérateurs principaux :

- `-e` si le fichier existe
- `-d` s'il s'agit d'un dossier
- `-r` si le fichier est dispo en lecture pour l'utilisateur
- `-s` si le fichier existe et n'est pas vide
- `-w` si le fichier est dispo en écriture pour l'utilisateur
- `-x` si le fichier est dispo en exécution pour l'utilisateur
- Tapez la commande `help test` pour des renseignements complémentaires

Test sur les chaînes de caractères

0 elle est vide et 1 elle n'est pas vide

```
root@buster:~# [ -z $PRENOM ]
root@buster:~# echo $?
0
root@buster:~#
```



```

root@buster:~# PRENOM='JEAN'
root@buster:~# echo $PRENOM
JEAN
root@buster:~# [ -z $PRENOM ]
root@buster:~# echo $?
1
root@buster:~#

```

```

root@buster:~# PRENOM1='JEAN'
root@buster:~# PRENOM2='JENA'
root@buster:~# [ $PRENOM1 = $PRENOM2 ]
root@buster:~# echo $?
1
root@buster:~#

```

On peut comparer les chiffres entre eux :

- -eq □ égal
- -ne □ différent
- -lt □ plus petit que
- -le □ plus petit ou égal
- -gt □ plus grand que
- -ge □ plus grand ou égal

```

root@buster:~# CHIFFRE1=15
root@buster:~# CHIFFRE2=16
root@buster:~# [ $CHIFFRE1 -eq $CHIFFRE2 ]
root@buster:~# echo $?
1
root@buster:~# _

```

1 car ils ne sont pas égaux

```

root@buster:~# [ $CHIFFRE1 -ne $CHIFFRE2 ]
root@buster:~# echo $?
0
root@buster:~# _

```

0 car ils sont bien différents

&& = and

- Il permet d'exécuter une 2^{ème} commande **SI** la 1^{ère} a renvoyé un code erreur = 0 (signifie qu'il s'est bien exécutée)

|| □ ALT Gr +6

- Il permet d'exécuter une 2^{ème} commande **SI** la 1^{ère} a renvoyé un code erreur <> 0 (signifie qu'il s'est pas bien exécutée)

Utilisation des tableaux :

```

#!/bin/bash
tab=( vert rouge bleu )
echo ${tab[1]}

```

```

root@buster:~# chmod a+x script_tableau.sh
root@buster:~# ./script_tableau.sh
rouge
root@buster:~# _

```

Il prend la valeur 1 du tableau étant donné que cela commence par 0 il affiche rouge, si on met tab[0] il mettra vert et si on met tab[2] il mettra bleu

```

#!/bin/bash
tab=( vert rouge bleu )
echo ${!tab[*]}

```

```

root@buster:~# ./script_tableau.sh
0 1 2
root@buster:~#

```

```

#!/bin/bash
tab=( vert rouge bleu )
echo ${!tab[*]}

```

```

root@buster:~# ./script_tableau.sh
vert rouge bleu
root@buster:~# _

```

```

#!/bin/bash
tab=( vert rouge bleu )
echo $#tab[*]}

```

```

root@buster:~# ./script_tableau.sh
3
root@buster:~#

```

```

#!/bin/bash
tab=( vert rouge bleu )
for i in ${!tab[*]}
do
echo $i : ${tab[$i]}
done

```

```

root@buster:~/home/matteo# ./test.sh
0 : vert
1 : rouge
2 : bleu
root@buster:~/home/matteo#

```

IFS : Input Field Separator

- Définit les séparateurs de champ reconnus par l'interpréteur
- Détermine la façon dont bash reconnaît les champs ou les limites de mots lorsqu'il interprète des chaînes de caractères
- La valeur par défaut de \$ifs est le blanc
- On doit donc manuellement sauvegarder la valeur initiale de IFS

IFS_OLD=\$IFS

- L'utiliser comme bon nous semble

IFS=','

- Enfin rétablir la valeur initiale de IFS

IFS=\$IFS_OLD

- ...Pour qu'il n'y ait pas d'incidence sur les opérateurs effectuées par la suite dans le processus exécutant le shell.

```
#!/bin/bash
tab=( 192 168 42 24 )
IFS_OLD=$IFS
IFS='.'
ip=${tab[*]}
IFS=$IFS_OLD
echo $ip
```

```
root@buster:~/home/matteo# ./script_ip.sh
192.168.42.24
root@buster:~/home/matteo# _
```

Autre manière :

```
#!/bin/bash
tab=( 192 168 42 24 )
ip=$( IFS='.';echo "${tab[*]}" )
echo $ip
```

```
root@buster:~/home/matteo# ./script_ip1.sh
192.168.42.24
root@buster:~/home/matteo# _
```

Envoyer des éléments d'un tableau vers des variables :

- Utilisation du read et d'une redirection <<< afin d'affecter le contenu du tableau aux variables a, b, c et d.

```
#!/bin/bash
tab=( 192 168 42 24 )
read a b c d <<< "${tab[*]}"
echo "a=$a b=$b c=$c d=$d"
```

```
root@buster:~/home/matteo# ./script_ip.sh
a=192 b=168 c=42 d=24
root@buster:~/home/matteo# _
```

Envoyer les éléments d'un tableau vers des variables :

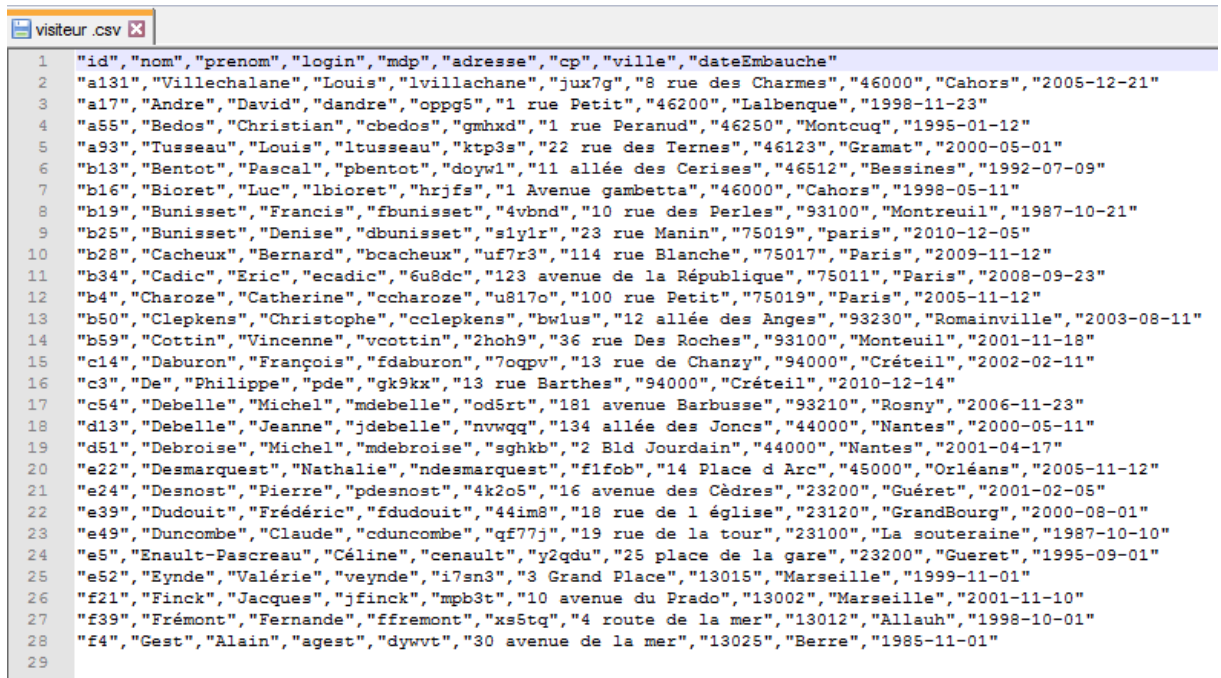
- A tester : si le nombre de variables est inférieur au nombre d'éléments
- ...La dernière variable reçoit la liste des éléments non assignés séparés par une espace

```
#!/bin/bash
tab=( 192 168 42 24 )
read a b <<< "${tab[*]}"
echo "b=$b"
```

```
root@buster:~/home/matteo# ./script_ip.sh
b=168 42 24
root@buster:~/home/matteo# _
```

Retravailler un fichier .CSV :

- Plusieurs points à vérifier



```
1 "id","nom","prenom","login","mdp","adresse","cp","ville","dateEmbauche"
2 "a131","Villechalane","Louis","lvillachane","jux7g","8 rue des Charms","46000","Cahors","2005-12-21"
3 "a17","Andre","David","dandre","oppg5","1 rue Petit","46200","Lalbenque","1998-11-23"
4 "a55","Bedos","Christian","cbedos","gmhxd","1 rue Peranud","46250","Montcuq","1995-01-12"
5 "a93","Tusseau","Louis","ltusseau","ktp3s","22 rue des Ternes","46123","Gramat","2000-05-01"
6 "b13","Bentot","Pascal","pbentot","doyw1","11 allée des Cerises","46512","Bessines","1992-07-09"
7 "b16","Bioret","Luc","lbioret","hrjfs","1 Avenue gambetta","46000","Cahors","1998-05-11"
8 "b19","Bunisset","Francis","fbunisset","4vbnd","10 rue des Perles","93100","Montreuil","1987-10-21"
9 "b25","Bunisset","Denise","dbunisset","slylr","23 rue Manin","75019","Paris","2010-12-05"
10 "b28","Cacheux","Bernard","bcacheux","uf7r3","114 rue Blanche","75017","Paris","2009-11-12"
11 "b34","Cadic","Eric","ecadic","6u8dc","123 avenue de la République","75011","Paris","2008-09-23"
12 "b4","Charoze","Catherine","ccharoze","u817o","100 rue Petit","75019","Paris","2005-11-12"
13 "b50","Clepkens","Christophe","cclepkens","bwls","12 allée des Angès","93230","Romainville","2003-08-11"
14 "b59","Cottin","Vincenne","vcottin","2hoh9","36 rue Des Roches","93100","Monteuil","2001-11-18"
15 "c14","Daburon","François","fdaburon","7oqp","13 rue de Chanzy","94000","Créteil","2002-02-11"
16 "c3","De","Philippe","pde","gk9kx","13 rue Barthes","94000","Créteil","2010-12-14"
17 "c54","Debelle","Michel","mdebelle","od5rt","181 avenue Barbusse","93210","Rosny","2006-11-23"
18 "d13","Debelle","Jeanne","jdebelle","nvwq","134 allée des Joncs","44000","Nantes","2000-05-11"
19 "d51","Debroise","Michel","mdebroise","sghkb","2 Bld Jourdain","44000","Nantes","2001-04-17"
20 "e22","Desmarquest","Nathalie","ndesmarquest","flfob","14 Place d Arc","45000","Orléans","2005-11-12"
21 "e24","Desnost","Pierre","pdesnost","4k2o5","16 avenue des Cèdres","23200","Guéret","2001-02-05"
22 "e39","Dudouit","Frédéric","fdudouit","44im8","18 rue de l église","23120","GrandBourg","2000-08-01"
23 "e49","Duncombe","Claude","cduncombe","qf77j","19 rue de la tour","23100","La souteraine","1987-10-10"
24 "e5","Enault-Pascreau","Céline","cenault","y2qdu","25 place de la gare","23200","Gueret","1995-09-01"
25 "e52","Eynde","Valérie","veynde","i7sn3","3 Grand Place","13015","Marseille","1999-11-01"
26 "f21","Finck","Jacques","jfinck","mpb3t","10 avenue du Prado","13002","Marseille","2001-11-10"
27 "f39","Frémont","Fernande","ffremont","xs5tq","4 route de la mer","13012","Allauh","1998-10-01"
28 "f4","Gest","Alain","agest","dywvt","30 avenue de la mer","13025","Berre","1985-11-01"
29
```

Vérifier l'encodage qui doit être en UTF-8

- On va dans Encodage et il faut que ce soit UTF-8

Vérifier si c'est UNIX

- On va dans Edition et on convertit les sauts de lignes en UNIX (EOL Conversion)

Enlever les guillemets

- On va dans Chercher et on remplace les « » par un espace

Pour importer il faut ouvrir WinSCP sur une machine Windows puis transférer les fichiers CSV, ensuite on retourne sur la machine Debian

Travail : Ecrire un script permettant d'afficher la liste des codes de tous les visiteurs en utilisant le fichier visiteur.csv

```
#!/bin/bash
while IFS=',' read code reste
do
echo $code
done < /home/sio/visiteur.csv
```

```
root@buster:/home/sio# ./script.sh
id
a131
a17
a55
a93
b13
b16
b19
b25
b28
b34
b4
b50
b59
c14
c3
c54
d13
d51
e22
e24
e39
e49
e5
e52
f21
f39
f4
root@buster:/home/sio#
```

Mettre deux parenthèses ; une pour la variable et une pour l'opération.

```
#!/bin/bash
a=1
while IFS=',' read code nom prenom login mdp reste
do
    echo $a $login
    a=$((a+1))
done < /home/sio/visiteur.csv
```

```
root@buster:/home/sio# ./script.sh
1 login
2 lvillachane
3 dandre
4 cbedos
5 ltusseau
6 pbentot
7 lbioRET
8 fbunisset
9 dbunisset
10 bcacheux
11 ecadic
12 ccharoze
13 cclepkens
14 vcottin
15 fdaburon
16 pde
17 mdebelle
18 jdebelle
19 mdebroise
20 ndesmarquest
21 pdesnost
22 fdudouit
23 cduncombe
24 cenault
25 veynde
26 jfinck
27 ffremont
28 agest
root@buster:/home/sio# _
```

Exercice supplémentaire

Exercice 1 : Afficher le nom de l'utilisateur

```
#!/bin/bash
read -p "Quel est votre prénom ? : " PRENOM
read -p "Quel est votre nom ? : " NOM
echo "Bonjour, $NOM $PRENOM"
```

Pour exécuter le script, il suffit de faire un chmod a+x sur le script puis de lancer avec ./

```
root@Matteo:~# ./exercice1.sh
Quel est votre prénom ? : Mattéo
Quel est votre nom ? : SIMON
Bonjour, SIMON Mattéo
root@Matteo:~# _
```

Exercice 2 : Faire la somme de deux nombres entrée par l'utilisateur

```
#!/bin/bash
read -p "Entrer le premier nombre : " NOMBRE1
read -p "Entrer le deuxième nombre : " NOMBRE2
RESULTAT=$((NOMBRE1+NOMBRE2))
echo "La somme des deux nombres est $RESULTAT"
```

Warning : Il faut mettre les parenthèses sinon le résultat affiché sera "2+3".

```
root@Matteo:~# ./exercice2.sh
Entrer le premier nombre : 2
Entrer le deuxième nombre : 3
La somme des deux nombres est 5
root@Matteo:~# _
```

Exercice 3 : Tester l'existence d'un fichier

```
#!/bin/bash
if [ -e /root/sio.txt ]
then
    echo "Le fichier existe"
else
    echo "Le fichier n'existe pas"
fi
```

```
root@Matteo:~# ls
exercice1.sh exercice2.sh exercice3.sh figlet
root@Matteo:~# ./exercice3.sh
Le fichier n'existe pas
root@Matteo:~#
```

```
root@Matteo:~# touch sio.txt
root@Matteo:~# ls
exercice1.sh exercice2.sh exercice3.sh figlet sio.txt
root@Matteo:~# ./exercice3.sh
Le fichier existe
root@Matteo:~#
```

Maintenant, on va laisser l'utilisateur choisir quel fichier il souhaite vérifier l'existence.

```
#!/bin/bash
read -p "Saisissez le nom du fichier : " FICHIER
if [ -e $FICHIER ]
then
    echo "Le fichier $FICHIER existe"
else
    echo "Le fichier $FICHIER n'existe pas"
fi
```

```
root@Matteo:~# ./exercice3.sh
Saisissez le nom du fichier : sio.txt
Le fichier sio.txt existe
root@Matteo:~# _
```

Exercice 4 : Écrire un script qui prend en entrée un nom de fichier et affiche son contenu s'il existe

```
#!/bin/bash
echo "Saississez le nom du fichier : $1"
if [ -e $1 ]
then
    cat $1
else
    echo "Le fichier $1 n'existe pas"
fi
```

```
Bla Bla Bla Bla Bla
```

```
root@Matteo:~# ./exercice4.sh sio.txt
Saississez le nom du fichier : sio.txt
Bla Bla Bla Bla Bla
root@Matteo:~#
```

Exercice 5 : Afficher les fichiers d'un répertoire donné

```
#!/bin/bash
read -p "Saississez le repertoire : " REPERTOIRE
if [ -d $REPERTOIRE ]
then
    ls -p $REPERTOIRE
else
    echo "Le repertoire $REPERTOIRE n'existe pas"
fi
```

```
root@Matteo:~# ./exercice5.sh
Saississez le repertoire : SIO/Maths
b2.txt config.txt script.sh
root@Matteo:~#
```

Exercice 6 : Écrire un script Bash qui affiche les fichiers dans un répertoire donné et compte leur nombre

```
#!/bin/bash
read -p "Saississez le repertoire : " REPERTOIRE
if [ -d $REPERTOIRE ]
then
    LISTE=$(ls -p $REPERTOIRE)
    NOMBRE=$(ls $REPERTOIRE | wc -l)
    echo "Il y a $NOMBRE fichiers : " $LISTE
else
    echo "Le repertoire $REPERTOIRE n'existe pas"
fi
```

```
root@buster:~# ./exercice6.sh
Saississez le repertoire : SIO/Maths
Il y a 3 fichiers : b2.txt config.txt script.sh
root@buster:~# _
```



```
root@buster:~# cd SIO/Maths
root@buster:~/SIO/Maths# ls
b2.txt config.txt script.sh
root@buster:~/SIO/Maths#
```

Autre solution :

```
#!/bin/bash
read -p "Saississez le repertoire : " REPERTOIRE
if [ -d $REPERTOIRE ]
then
    ls -p $REPERTOIRE
    NOMBRE=$(ls $REPERTOIRE | wc -l)
    echo "Il y a $NOMBRE fichiers"
else
    echo "Le répertoire $REPERTOIRE n'existe pas"
fi
```

```
root@Matteo:~# ./exercice6.sh
Saississez le repertoire : SIO/Maths
b2.txt config.txt script.sh
Il y a 3 fichiers
root@Matteo:~# _
```

Exercice 7 : Demander le nom de l'utilisateur et son mot de passe, puis ajouter cet utilisateur à la liste des utilisateurs du système

```
#!/bin/bash
read -p "Saississez votre nom d'utilisateur : " LOGIN
read -sp "Saississez votre mot de passe : " PASSWORD
if id $LOGIN >/dev/null 2>&1;
then
    echo "L'utilisateur $LOGIN existe déjà"
else
    useradd -m $LOGIN -p $PASSWORD
    echo "L'utilisateur $LOGIN a bien été ajouté"
fi
VERIF=$(cat /etc/passwd | grep $LOGIN)
echo "Votre compte est présent : " $VERIF
```

```
root@Matteo:~# ./exercice7.sh
Saississez votre nom d'utilisateur : theo
Saississez votre mot de passe : L'utilisateur theo existe déjà
Votre compte est présent : theo:x:1001:1001::/home/theo:/bin/sh
root@Matteo:~# _
```

Autre solution :

```
#!/bin/bash
read -p "Saississez votre nom d'utilisateur : " LOGIN
read -sp "Saississez votre mot de passe : " PASSWORD
if id $LOGIN >/dev/null 2>&1;
then
    echo "L'utilisateur existe déjà"
else
    useradd -m $LOGIN
    echo -e "$PASSWORD\n$PASSWORD" | passwd $LOGIN
    echo "L'utilisateur $LOGIN a bien été ajouté"
fi
VERIF=$(cat /etc/passwd | grep ^$LOGIN)
echo "Votre compte est présent :" $VERIF
```

Exercice 8 : Saisir un nom d'utilisateur puis le supprimer de la liste des utilisateurs du système et créer au préalable un utilisateur btssioessai

Exercice 9 : Prendre en entrée deux chaînes de caractères et qui affiche la longueur de la plus grande des deux chaînes

Exercice 10 : Prendre en entrée un nombre variable d'arguments et qui affiche leur somme, on lancera la commande suivante par exemple : ./script.sh 10 20 30

```
#!/bin/bash
function creation() {
    read -p "Saississez un nom d'utilisateur : " login
    read -p "Saississez un mot de passe : " password
    useradd -m $login
    echo -e "$password\n$password" | passwd $login
    echo "L'utilisateur a été crée"
}

function suppression() {
    read -p "Saississez un nom d'utilisateur : " login
    deluser $login
    echo "L'utilisateur a été supprimé"
}

function modification() {
    read -p "Saississez un nom d'utilisateur : " login
    read -p "Saississez un mot de passe : " password
    echo "$login:$password" | chpasswd
    echo "Le mot de passe a été modifié"
}

function quitter() {
    exit 0
    echo "Bye"
}
```

```
while true
do
    echo "1-      Création d'un utilisateur"
    echo "2-      Suppression d'un utilisateur"
    echo "3-      Modification du mot de passe d'un utilisateur"
    echo "4-      Quitter"
    read -p "Choisissez une option : " choice
    case $choice in
        1) creation;;
        2) suppression;;
        3) modification;;
        4) quitter;;
        *) echo -e "\e[31mChoisir une des options ci-dessous\e[0m"
    esac
done
```