TP14 – Introduction aux scripts shell : autoformation

Sommaire :

1. Saisie et exécution du script....1
2. Entrées-Sorties....3
3. Les variables BASH....4
4. La commande test....5
5. Structures conditionnelles....7
   5.1. Conditionnelle simple....7
   5.2. Conditionnelles imbriquées....7
   5.3. Choix multiples....9
6. Structures itératives....11
   6.1. Boucle for....11
   6.2. Boucle while....12
7. Commandes diverses....14
   7.1. La commande tr....14
   7.2. La commande set....14

1. Saisie et exécution du script

→ Saisissez le script **bonjourbis.sh** :

```
root@DS1: ~#vim bonjourbis.sh
```

La variable d'environnement **$USER** contient **le nom de login**.
L'option **-n** **empêche** le **passage** à la **ligne**. Le **;** sert de **séparateur** des **commandes** sur une même ligne.
La dernière ligne du script recherche $USER en début de ligne dans le fichier passwd puis extrait l'uid (3ème champ) et l'affiche.

```
#!/bin/bash
echo Bonjour $USER
echo -n "Nous sommes le " ; date
echo " Votre numéro d'utilisateur est" $(grep "^$USER" /etc/passwd | cut -d: -f3)
```

→ Accordez le droit d'exécution :

```
root@DS1: ~#chmod a+x bonjourbis.sh
root@DS1: ~#ls -l
total 104
-rw-r--r-- 1 root root  252 17 mai   13:20 affiche.sh
-rw-r--r-- 1 root root  146  8 déc.  16:17 avecdoublons
-rw-r--r-- 1 root root  146  8 déc.  16:17 avecdoublons.txt
-rwxr-xr-x 1 root root  148 20 mai   16:08 bonjourbis.sh
-rw-r--r-- 1 root root   65 17 mai   10:31 bonjour.sh
-rw-r--r-- 1 root root   66 17 mai   11:08 boucle_for.sh
-rw-r--r-- 1 root root    0  7 déc.  22:16 cal.txt
-rw-r--r-- 1 root root  148 17 mai   10:47 creer_rep.sh
-rwxr--r-- 1 root root  938 17 mai   13:10 decal.sh
-rw-r--r-- 1 root root    0  8 déc.  16:30 eleves.txt
-rw-r--r-- 1 root root   65  8 déc.  16:30 erreurs.log
-rw-r--r-- 1 root root   73  4 déc.  12:36 etudiants.txt
-rw-r--r-- 1 root root    0 17 mai   11:05 fla
-rw-r--r-- 1 root root    0 17 mai   11:06 flag
-rw-r--r-- 1 root root   91 17 mai   11:04 flag_existebis.sh
-rw-r--r-- 1 root root   67 17 mai   10:55 flag_existe.sh
-rw-r--r-- 1 root root  123 17 mai   10:51 heureux.sh
-rw-r--r-- 1 root root  249 17 mai   11:43 lecture.sh
-rw-r--r-- 1 root root  522 20 mai   15:37 menu2.sh
-rw-r--r-- 1 root root  268 17 mai   11:14 menu.sh
-rw-r--r-- 1 root root  211  7 déc.  22:56 notes.csv
-rwxr-xr-x 1 root root  171 17 mai   10:36 param.sh
-rwxr-xr-x 1 root root   78 17 mai   11:32 plusun.sh
-rw-r--r-- 1 root root  211  7 déc.  21:58 prenoms_tries
-rw-r--r-- 1 root root   73 21 déc.  20:56 prenoms_tries.txt
-rw-r--r-- 1 root root   73  8 déc.  16:19 sansdoublons.txt
drwxr-xr-x 2 root root 4096 17 mai   10:44 sauve
-rw-r--r-- 1 root root  130  8 déc.  16:38 sio1.txt
-rwxr-xr-x 1 root root   33 17 mai   10:12 un_script.sh
-rw-r--r-- 1 root root  238 17 mai   12:41 users.txt
root@DS1: ~#
```

→ Pour lancer l'exécution du script, tapez **./bonjourbis.sh,** ./ **indiquant** le **répertoire courant** (ou bien indiquez le chemin absolu à partir de la racine) ; ceci dans le cas où le répertoire contenant le script n'est pas listé dans le **PATH**.

```
root@DS1: ~#./bonjourbis.sh
Bonjour root
Nous sommes le lun. 20 mai 2024 16:09:25 CEST
 Votre numéro d'utilisateur est 0
root@DS1: ~#_
```

→ Saisissez le script **bonjourter.sh**

```
root@DS1: ~#vim bonjourter.sh_
```

```
#!/bin/bash
if [ $# = 2 ]
then
echo "Bonjour $2 $1 et une bonne journée ! "
else
echo "Syntaxe : $0 nom prenom"
fi
```

→ Appelez le script sans et avec arguments :

```
root@DS1: ~#sh bonjourter.sh
Syntaxe : bonjourter.sh nom prenom
root@DS1: ~#sh bonjourter.sh Lopez Lucy
Bonjour Lucy Lopez et une bonne journee !
root@DS1: ~#
```

2. Entrées-Sorties

Ce sont les voies de **communication** entre le programme **bash** et la **console** :

▼ Exemple :

```
root@DS1: ~#echo "Bonjour \nà tous !"
Bonjour \nà tous !
root@DS1: ~#echo -e "Bonjour \nà tous !"
Bonjour
à tous !
root@DS1: ~#echo -e "Bonjour \nà toutes \net à tous ! \c"
Bonjour
à toutes
et à tous ! root@DS1: ~#_
```

• **read** permet **l'affectation directe** par **lecture** de la **valeur saisie sur l'entrée** standard au clavier.

→ Saisissez le script saisie_clavier.sh.

```
root@DS1: ~#vim saisie_clavier.sh
#!/bin/bash
echo "Donnez votre prénom et votre nom :"
read prenom nom
echo "Bonjour $prenom $nom"
```

→ Appelez le script saisie_clavier.sh.

```
root@DS1: ~#sh saisie_clavier.sh
Donnez votre prénom et votre nom :
Lucy Lopez
Bonjour Lucy Lopez
root@DS1: ~#
```

3. Les variables BASH

De façon générale, elles sont de **type texte**. On distingue les **variables définies** par le **programmeur** et les **variables systèmes**.

• Substitution de variable :
Si une **chaine** contient la **référence** à une **variable**, le **shell** doit d'abord **remplacer** cette **référence** par **sa valeur avant d'interpréter la phrase globalement**. Cela est effectué par l'utilisation de " ", dans ce cas obligatoire à la place de ' '.

▼ Exemple :

```
root@DS1: ~#n=123
root@DS1: ~#echo "La variable \$n vaut $n"
La variable $n vaut 123
root@DS1: ~#salut="bonjour à tous !"
root@DS1: ~#echo "Alors moi je dis : $salut"
Alors moi je dis : bonjour à tous !
root@DS1: ~#echo 'Alors moi je dis : $salut'
Alors moi je dis : $salut
root@DS1: ~#echo "Alors moi je dis : \"$salut\""
Alors moi je dis : "bonjour à tous !"
root@DS1: ~#readonly salut
root@DS1: ~#salut="Bonjour à tous sauf à toto"
-bash: salut : variable en lecture seule
root@DS1: ~#echo "Alors moi je dis : $salut"
Alors moi je dis : bonjour à tous !
root@DS1: ~#_
```

• Opérateur {} dans les variables :
Dans certains cas en programmation, on peut être amené à utiliser des noms de variables dans d'autres variables. Comme il n'y a pas de substitution automatique, **la présence de {} force l'interprétation des variables incluses.**

▼ Exemple :

```
root@DS1: ~#user="/home/stagiaire"
root@DS1: ~#echo $user
/home/stagiaire
root@DS1: ~#u1=$user1
root@DS1: ~#echo $u1

root@DS1: ~#u1=${user}1
root@DS1: ~#echo $u1
/home/stagiaire1
root@DS1: ~#_
```

4. La commande test

Comme son nom l'indique, elle sert à **vérifier des conditions.** Ces conditions portent sur des **fichiers**, des **chaines** ou une **expression numérique**.

Cette commande est utilisée dans les **structures conditionnelles if...then...else.**

La commande test retourne 0 si la condition est considérée comme vraie et sinon une valeur différente de 0 pour signifier qu'elle est fausse.

Tester un fichier

- Exemples :

```
root@DS1: ~#[ -e ./fichier ]
root@DS1: ~#echo $?
1
root@DS1: ~#touch fichier
root@DS1: ~#[ -e ./fichier ]
root@DS1: ~#echo $?
0
root@DS1: ~#date > fichier
root@DS1: ~#[ -s ./fichier ]
root@DS1: ~#echo $?
0
root@DS1: ~#_
```

```
root@DS1: ~#[ -r "/etc/passwd" ]
root@DS1: ~#echo $?
0
root@DS1: ~#[ -r "/etc/shadow" ]
root@DS1: ~#echo $?
0
root@DS1: ~#su - sio
sio@DS1:~$  [ -r "/etc/shadow" ]
sio@DS1:~$ echo $?
1
sio@DS1:~$ [ -r "/etc/shadow" ] || echo "lecture du fichier interdite"
lecture du fichier interdite
sio@DS1:~$ [ -r "/etc/passwd" ]
sio@DS1:~$ echo $?
0
sio@DS1:~$ _
```

-e = il existe
-f = c'est un fichier normal
-d = c'est un répertoire
-r | -w | -x = il est lisible | modifiable | exécutable
-s = il n'est pas vide

Tester une chaîne

- Exemples :

```
root@DS1: ~#[ -z "est-ce que la chaîne est vide ?" ] ; echo $?
1
root@DS1: ~#ch="Bonjour" ; [ "$ch" = "Bonjour" ] ; echo $?
0
root@DS1: ~#ch="Bonjour" ; [ "$ch" = "bonjour" ] ; echo $?
1
root@DS1: ~#_
```

```
root@DS1: ~#[ $USER != "root" ] && echo "l'utilisateur n'est pas le \"root\" !" || echo "l'utilisate
ur est le \"root\" !"
l'utilisateur est le "root" !
root@DS1: ~#su - sio
sio@DS1:~$ [ $USER != "root" ] && echo "l'utilisateur n'est pas le \"root\" !" || echo "l'utilisateu
r est le \"root\" !"
l'utilisateur n'est pas le "root" !
sio@DS1:~$ _
```

-z | -n = la chaîne est vide / n'est pas vide
= | != = les chaînes comparées sont identiques | différentes

Tester un nombre

- Exemples :

```
root@DS1: ~#a=15 ; [ $a -lt 15 ] ; echo $?
1
root@DS1: ~#a=15 ; [ $a -le 15 ] ; echo $?
0
root@DS1: ~#_
```

-lt = strict. inf | strict. sup
-le = inf ou égal | sup ou égal

Opérations dans une commande test

- Exemples :

```
root@DS1: ~#f="/root" ; [ -d $f -a -x $f ] ; echo $?
0
root@DS1: ~#ls -ld /root
drwx------ 5 root root 4096 20 mai   16:34 /root
root@DS1: ~#
```

```
root@DS1: ~#note=9 ; [ $note -lt 8 -o $note -ge 10 ] && echo "tu n'es pas convoqué à l'oral"
root@DS1: ~#note=7 ; [ $note -lt 8 -o $note -ge 10 ] && echo "tu n'es pas convoqué à l'oral"
tu n'es pas convoqué à l'oral
root@DS1: ~#
```

[expr1 -a expr2] = 0 si les 2 expressions sont vraies (and)
[expr1 -o expr2] = 0 si l'une des 2 expressions est vraie (or)
[! expr1] = négation

5. Structures conditionnelles

5.1. Conditionnelle simple

- Exemples :

```
root@DS1: ~#vim compte.sh
if grep "^sio" /etc/passwd
then
echo "sio a déjà un compte"
fi
root@DS1: ~#sh compte.sh
sio:x:1000:1000:sio,,,:/home/sio:/bin/bash
sio a déjà un compte
root@DS1: ~#_
```

```
if grep "^sio" /etc/passwd > /dev/null_
then
echo "sio a déjà un compte"
fi
root@DS1: ~#sh compte.sh
sio a déjà un compte
root@DS1: ~#_
```

```
root@DS1: ~#vim note.sh_
echo "Saisissez votre note :"
read note
if [ $note -gt 16 ]
then echo "C'est très bien !"
fi
root@DS1: ~#sh note.sh
Saisissez votre note :
17
C'est très bien !
root@DS1: ~#
```

5.2. Conditionnelles imbriquées

- Exemples :

```
root@DS1: ~#vim examen.sh
```

```
echo "Saisissez votre moyenne : "
read note
if [ $note -lt 8 ]
then
echo "Vous êtes recalé"
elif [ $note -lt 10 ]
then
echo "Vous êtes convoqué à l'oral de rattrapage"
else
echo "Vous êtes admis"
fi
```

```
root@DS1: ~#vim devoir.sh_
```

```
fichier=/home/sio/devoir1.txt
if [ -f $fichier -a -r $fichier ]
then
echo "je vais vérifier ton devoir"
elif [ ! -e $fichier ]
then
echo "ton devoir n'existe pas !"
else
echo " je ne peux pas le lire !"
fi_
```

Test :

```
root@DS1: ~#sh devoir.sh
ton devoir n'existe pas !
root@DS1: ~#su -sio
su: échec d'exécution de io: Aucun fichier ou dossier de ce type
root@DS1: ~#su - sio
sio@DS1:~$ touch devoir1.txt
sio@DS1:~$ exit
```

```
root@DS1: ~#sh devoir.sh
je vais vérifier ton devoir
root@DS1: ~#_
```

→ Supposons que le **script** exige la **présence de deux paramètres**. Il faut tester la valeur de **$#**. Est-elle nulle ?

```
root@DS1: ~#vim argument.sh_
```

```
if [ $# = 0 ]
then
echo "Erreur, la commande exige deux arguments"
elif [ $# = 1 ]
then
echo "Donnez le second argument : "
read arg2
fi
```

```
root@DS1: ~#sh argument.sh
Erreur, la commande exige deux arguments
root@DS1: ~#sh argument.sh arg1
Donnez le second argument :

root@DS1: ~#
```

5.3. Choix multiples

▼ Exemples :

→ Supposons que le **script** doive **réagir différemment selon** la **valeur** de **$USER** :

```
root@DS1: ~#vim cas.sh
case $USER in
root)
        echo "Mes respects Monsieur le $USER" ;;
guest | sio?)
        echo "Salut $USER" ;;
*)
        echo "Bonjour $USER" ;;
esac
```

```
root@DS1: ~#chmod 755 cas.sh
root@DS1: ~#cas.sh
Mes respects Monsieur le root
root@DS1: ~#
root@DS1: ~#cp /root/cas.sh /bin/
root@DS1: ~#su - sio
sio@DS1:~$ cas.sh
Bonjour sio
sio@DS1:~$
```

→ Le **script attend** une **réponse oui**/**non** de l'utilisateur

```
root@DS1: ~#vim poursuite.sh
```

```
echo "Voulez-vous vraiment exécuter le script ?"
read reponse
case $reponse in
[nN]*)
        echo "Script interrompu"
        exit 0
        ;;
[yYoO]*)
        echo "Attention pour le décompte final"
        for i in 10 9 8 7 6 5 4 3 2 1
        do
        echo "===>$i"
        sleep 1
        done
        echo "Allumage Vulcain"
        sleep 2
        echo "Allumage des 2 EAP"
        sleep 2
        echo "Décollage"
        sleep 2
        echo "Tous les parametres a bord sont normaux"
        sleep 2
        echo "Victor est en orbide"
        ;;
esac
```

```
root@DS1: ~#sh poursuite.sh
Voulez-vous vraiment exécuter le script ?
oui
Attention pour le décompte final
===>10
===>9
===>8
===>7
===>6
===>5
===>4
===>3
===>2
===>1
Allumage Vulcain
Allumage des 2 EAP
Décollage
Tous les parametres a bord sont normaux
Victor est en orbide
root@DS1: ~#
```

6. Structures itératives

6.1. Boucle for

▼ Exemples :

→ La **liste** peut être **explicite** :

```
root@DS1: ~#vim liste.sh
```

```
for nom in Benoit Nocolas Pierre
do
echo "$nom, bonjour"
done
```

```
root@DS1: ~#sh liste.sh
Benoit, bonjour
Nocolas, bonjour
Pierre, bonjour
root@DS1: ~#_
```

→ La **liste** peut être **calculée** à partir d'une e**xpression modèle**

```
root@DS1: ~#vim copie.sh
```

```
if [ ! -e /tmp/sio ]
then
mkdir /tmp/sio
fi
for fich in /home/sio/*
do
        cp $fich /tmp/sio
done
```

```
root@DS1: ~#sh copie.sh
root@DS1: ~#ls -l /tmp/sio
total 0
-rw-r--r-- 1 root root 0 20 mai   18:09 devoir1.txt
-rw-r--r-- 1 root root 0 20 mai   18:09 fichier1
-rw-r--r-- 1 root root 0 20 mai   18:09 fichier2
root@DS1: ~#
```

→ Si **aucune liste** n'est **précisée**, les **valeurs** sont **prises** dans la **variable système $@,** c'est-à-dire en parcourant la liste des paramètres positionnels courants.

```
root@DS1: ~#vim sanslisteprecise.sh_
```

```
cd /tmp/sio ; set *
for nom in $@
do echo $nom
done_
```

```
root@DS1: ~#sh sanslisteprecise.sh
boucle_for.sh
cal.txt
cas.sh
compte.sh
copie.sh
creer_rep.sh
decal.sh
devoir.sh
eleves.txt
erreurs.log
etudiants.txt
examen.sh
fichier
fla
flag
flag_existe.sh
flag_existebis.sh
heureux.sh
lecture.sh
liste.sh
menu.sh
menu2.sh
note.sh
notes.csv
param.sh
plusun.sh
poursuite.sh
prenoms_tries
prenoms_tries.txt
saisie_clavier.sh
sansdoublons.txt
sanslisteprecise.sh
sauve
sio1.txt
un_script.sh
users.txt
root@DS1: ~#
```

6.2. Boucle while

• La répétition se poursuit **TANT QUE** la **dernière** **commande** de la liste est **vraie** (tant qu'elle renvoie un code de retour nul).

▼ Exemples :

→ Dire bonjour toutes les secondes (arrêt par CTRL-C) :

```
root@DS1: ~#vim true.sh
```

```
while true
do
echo "Bonjour Mr $USER"
sleep 1
done
~
root@DS1: ~#sh true.sh
Bonjour Mr root
Bonjour Mr root
Bonjour Mr root
Bonjour Mr root
Bonjour Mr root
```

→ **Lecture** des **lignes** d'un **fichier** pour **traitement** (notez que la redirection de l'entrée de la commande **while...do...done** est placée à la fin) :

```
root@DS1: ~#vim fic.sh
fich=/etc/passwd
while read ligne
do
echo $ligne
more
done < $fich
~
```

• Sortie de boucle

La commande **break**, **placée** dans le **corps** d'une **boucle**, provoque **une sortie définitive** de cette boucle

▼ Exemple :

→ **Lecture** des **lignes** d'un **fichier**

```
root@DS1: ~#vim sortie.sh
fich="/etc/passwd"
grep "^sio" $fich | while true
do
        read ligne
        if [ "$ligne" = "" ] ; then break ; fi
        echo $ligne
done
~
root@DS1: ~#sh sortie.sh
sio:x:1000:1000:sio,,,:/home/sio:/bin/bash
root@DS1: ~#
```

7. Commandes diverses

7.1. La commande tr

• Cette **commande** de **filtre** permet d'effectuer des **remplacements** de **caractères** dans une **chaîne.**

▼ Exemples :

→ On souhaite **transformer** une **chaîne** en **minuscules** :

```
root@DS1: ~#chaine="Bonjour, comment allez-VOUS aujourd'hui ?"
root@DS1: ~#echo $chaine | tr 'A-Z' 'a-z'
bonjour, comment allez-vous aujourd'hui ?
root@DS1: ~#
```

→ Pour permettre **l'utilisation** de la **commande** **set** (cf. ci-dessous), il est nécessaire que le **séparateur** de **champ** sur une ligne soit l'espace, et non pas par exemple **« : ».** On souhaite créer un fichier **passwd**.**txt** qui **introduit** un **espace** à la place de « **: »** dans une copie de /etc/passwd :

```
root@DS1: ~#cat /etc/passwd | tr ":" " " > passwd.txt
root@DS1: ~#head passwd.txt
root x 0 0 root /root /bin/bash
daemon x 1 1 daemon /usr/sbin /usr/sbin/nologin
bin x 2 2 bin /bin /usr/sbin/nologin
sys x 3 3 sys /dev /usr/sbin/nologin
sync x 4 65534 sync /bin /bin/sync
games x 5 60 games /usr/games /usr/sbin/nologin
man x 6 12 man /var/cache/man /usr/sbin/nologin
lp x 7 7 lp /var/spool/lpd /usr/sbin/nologin
mail x 8 8 mail /var/mail /usr/sbin/nologin
news x 9 9 news /var/spool/news /usr/sbin/nologin
root@DS1: ~#
```

7.2. La commande set

▼ Exemple :

→ On reprend un exemple de lecture de fichier analogue à celui de la page 6. La commande **tr** est utilisée à la **place** du **recours** à la **variable** **prédéfinie** **IFS**.

```
root@DS1: ~#vim read_set_tr.sh
```

```
fichier="/etc/passwd"
cat $fichier | head | tr ":" " " | while true
do
        read ligne
        if [ "$ligne" = "" ] ; then break ; fi
        set $ligne
        echo $1
done
~
```