CM-TP2

November 20, 2023

0.1 Exercice 1: utilisation de test et des codes de retour

- Mettez dans le fichier nommé `dedans.sh` :

```
#!/bin/bash
if grep $1 $2
then
  echo $? : $1 se trouve dans $2
else
  echo $? : $1 ne se trouve pas dans $2
fi
exit 0
```

- Combien de paramètres (au moins) prend le script `dedans.sh` ?

Il prend au moins deux paramètres (`$1` et `$2` dans le script).

- Créez un fichier `grain.txt` contenant le mot “grain”.

[1]:
```
cat grain.txt
```

```
Certains oiseaux mangent du grain.
D'autres oiseaux n'en mangent pas.
Il existe aussi des mammifères qui
mangent des graines !
```

- Exécutez :

```
./dedans.sh grain grain.txt
./dedans.sh toto grain.txt
./dedans.sh autre fichierquinexistepas.txt
```

[2]:
```
./dedans.sh grain grain.txt
```

```
Certains oiseaux mangent du grain.
mangent des graines !
0 : grain se trouve dans grain.txt
```

[4]:
```
./dedans.sh toto grain.txt
```

```
1 : toto ne se trouve pas dans grain.txt
```

[5]:
```
./dedans.sh autre fichierquinexistepas.txt
```

```
grep: fichierquinexistepas.txt: No such file or directory
2 : autre ne se trouve pas dans fichierquinexistepas.txt
```

- Que contient la variable `$?` (dans le script) dans chacun des cas ?

`$?` contient la valeur de retour de la dernière commande qui a été exécutée (ici, il s'agit de la commande `grep`).

- Dans un terminal, regarder quel est le code de sortie (`echo $?`) de `grep` exécuté en dehors du script, dans les 3 cas (`grep grain grain.txt`, …). Vérifier à l'aide du manuel.

[7]:
```
grep grain grain.txt; echo $?
```

```
Certains oiseaux mangent du grain.
mangent des graines !
0
```

Le fichier contient le motif recherché, et dans ce cas `grep` renvoie 0.

[8]:
```
grep toto grain.txt; echo $?
```

```
1
```

Le fichier existe mais ne contient pas le motif recherché : `grep` retour 1.

[10]:
```
grep grain fichierquinexsitepas.txt; echo $?
```

```
grep: fichierquinexsitepas.txt: No such file or directory
2
```

Le fichier n'existe pas, dans ce cas `grep` renvoie 2.

On veut supprimer la sortie ("standard") de grep pour que le seul message affiché en cas de succès soit celui du script (et non la ou les lignes imprimées par grep sur la sortie standard). Dans votre script, redirigez la sortie de grep vers `/dev/null`, puis utilisez `if test $? -eq 0` pour vérifier que le code de sortie est bien 0 (par exemple).

[12]:
```
grep grain grain.txt > /dev/null; echo $?
```

```
0
```

[13]:
```
cat dedans.sh
```

```
#!/bin/bash

if grep $1 $2 > /dev/null
then
  echo $? : $1 se trouve dans $2
else
  echo $? : $1 ne se trouve pas dans $2
```

```
fi
exit 0
```

[14]:
```
./dedans.sh grain grain.txt
```

```
0 : grain se trouve dans grain.txt
```

- Reproduire le comportement suivant de votre nouveau script :

```
./dedans.sh grain graincdczc.txt
grep: graincdczc.txt: Aucun fichier ou dossier de ce type
1 : grain ne se trouve pas dans graincdczc.txt
```

Expliquer l'impression nombre 2.

[15]:
```
./dedans.sh grain grainsdqsdq.txt
```

```
grep: grainsdqsdq.txt: No such file or directory
2 : grain ne se trouve pas dans grainsdqsdq.txt
```

Le fichier n'exsite pas, dans grep affiche le message "grep: grainsdqsdq.txt: No such file or directory" sur la sortie **d'erreur** standard. On envoie la sortie standard de `grep` dans `/dev/null`, mais pas la sortie **d'erreur** standard, donc le message s'affiche tout de même. En tout cas, la valeur 2 est renvoyée, comme on l'a déjà vu.

- Traiter intégralement le cas d'erreur (redirection de la sortie d'erreur, et impression d'un message adéquat si le fichier est absent).

[16]:
```
cat dedans.sh
```

```
#!/bin/bash

if test -f $2; then
  if grep $1 $2 > /dev/null
  then
    echo "$? : $1 se trouve dans $2"
    exit 0
  else
    echo "$? : $1 ne se trouve pas dans $2"
    exit 1
  fi
else
  echo "Le fichier $2 n'existe pas ou n'est pas régulier…"
  exit 2
fi

exit 0
```

- Que passe-t-il si on exécute `./dedans.sh mot` (avec un seul paramètre) ? Pourquoi ?

Si `$2` est vide, `grep` lit sur son entrée standard. Même si le motif `$1` est envoyé sur l'entrée standard, rien ne s'affiche, car la sortie de `grep` est envoyée dans `/dev/null`.

- Ajoutez un test qui vérifie que le nombre de paramètres est égal à 2, et qui termine avec le code 1 si se n'est pas le cas (`exit 1`).

[17]:
```
cat dedans.sh
```

```
#!/bin/bash

if test $# -ne 2; then
  echo "Il faut deux paramètres au script !"
  exit 2
fi

if test -f $2; then
  if grep $1 $2 > /dev/null
  then
    echo "$1 se trouve dans $2"
    exit 0
  else
    echo "$1 ne se trouve pas dans $2"
    exit 1
  fi
else
  echo "Le fichier $2 n'existe pas ou n'est pas régulier…"
  exit 2
fi

exit 0
```

Attention : il est recommandé d'utiliser `if` en conjonction avec la commande test plutôt qu'avec les crochets `[ ... ]`. Vous trouverez beaucoup d'exemples avec les crochets sur le web, ou dans des scripts existants, mais ils sont sources d'erreurs (car vous oubliez toujours des espaces), donc nous vous conseillons de ne pas les utiliser pour l'instant. Par exemple, privilégiez

```
if test $a -eq 5; then echo "egalité"; fi
```

plutôt que

```
if [ $a -eq 5 ]; then echo "egalité"; fi
```

Avec les crochets, on a toujours tendance à oublier les espaces, avant et après le `[`, et aussi devant le `]`, ce qui est source de bogues. Dans tous les cas, n'oubliez pas non-plus les espaces autour des opérateurs de comparaison.

0.2 Exercice 3 : Un petit script « en partant de rien »

Écrivez un script `existuser.sh` qui prend un login ou nom comme paramètre et qui regarde (avec `grep`) dans `/etc/passwd` si ce nom existe. Si tel est le cas, il affiche "oui nom existe" ou "nom

n'existe pas".

[ ]:

0.3 Exercice 4 : un script pour collecter des infos sur le processeur

Le fichier `/proc/cpuinfo` est mis à jour par le noyau Linux, et contient des informations sur votre processeur. Affichez le contenu de ce fichier avec `cat`.

[ ]:

Vous pouvez observer que :

- vous avez des informations détaillées pour chacun des cœurs du processeur.
- les lignes `processor` donnent le numéro de chaque cœur en partant de 0 (si k est le numéro du dernier cœur à s'afficher, alors k+1 est le nombre de cœurs).
- les lignes `model name` donnent à chaque fois le modèle du processeur.
- les lignes `cpu MHz` donnent la fréquence de chaque cœur : on va dire que la fréquence du dernier cœur est celle du processeur.

Ecrivez un script `mycpu.sh` pour afficher le nombre de cœurs du processeur, le modèle du processeur, et sa fréquence. Par exemple, cela pourra donner :

```
nlouvet@LIP-E6220:~/TP$ ./mycpu.sh
Votre processeur comporte 4 coeurs
Son modèle est : "Intel(R) Core(TM) i3-2310M CPU @ 2.10GHz"
Actuellement, la fréquence est de 1400 MHz
```

Pour aboutir à une solution, commencez pas faire des tests en ligne de commande. Vous pouvez utiliser `grep`, `cut`, `sed`, `echo`, `expr` et les variables du shell.

[ ]: