

TP 2 – Scripts Shell

Vous devez faire au moins les exercices 1 à 7.

Des exemples de sujets d'examen concernant cette partie sont donnés dans les exercices 8 et 9.

Les exercices 10 et 11 sont des compléments.

Exercice 1 - Premiers scripts

1. Créez le répertoire OSSE11/bin dans votre HOME, puis copiez le fichier /pub/FISE_OSSE11/shell/punition1.sh dans ce répertoire.

C'est le script vu en cours. La commande **punition1.sh n word** écrit n lignes contenant le mot **word** sur la sortie standard (stdout).

```
sh$ mkdir -pv ~/OSSE11/bin
sh$ cp /pub/FISE_OSSE11/shell/punition1.sh ~/OSSE11/bin
```

2. Testez que le script **punition1.sh** ainsi copié fonctionne.

```
sh$ sh ~/OSSE11/bin/punition1.sh 3 'It works!'
It works!
It works!
It works!
```

3. Écrivez le script ~/OSSE11/bin/punition2.sh. Ses arguments sont n , m et **word**. Il écrit n lignes contenant m fois le mot **word** séparés par un espace.

note : Utilisez deux boucles imbriquées et inspirez-vous de **punition1.sh**.

4. Écrivez le script ~/OSSE11/bin/punition3.sh, similaire à **punition2.sh**, mais qui n'utilise qu'une seule boucle et appelle le script **punition1.sh**.

aide:

```
1 str= ; i=0
2 while [ $i -lt $m ] ; do
3     str="$str $w"
4     i=$((i+1))
5 done
6 sh punishment1.sh "$n" "$str"
```

5. Écrivez le script ~/OSSE11/bin/punition.sh. Il est similaire à **punition2.sh** mais :

- avec 0 argument, il écrit 10 lignes contenant 3 fois la ligne "Je ne bavarde pas en cours !";
- avec 1 argument (**punition.sh word**), il écrit 10 lignes contenant 3 fois l'argument **word**;
- avec 2 arguments (**punition.sh m word**), il écrit 10 lignes contenant m fois l'argument **word**;
- avec 3 arguments (**punition.sh n m word**), il écrit n lignes contenant m fois l'argument **word**.

On l'implémentera sans boucle, en utilisant la commande **punition3.sh**.

aide:

```
1 if [ $# -eq 0 ] ; then
2     n=10; m=3; w='Je ne bavarde pas en cours !'
3 elif [ $# -eq 1 ] ; then
4     ...
5 fi
6 sh punishment3.sh "$n" "$m" "$w"
```

6. Testez que le script **punition.sh** fonctionne.

```
sh$ sh ~/OSSE11/bin/punition.sh
Je ne bavarde pas en cours !
Je ne bavarde pas en cours !
Je ne bavarde pas en cours !
sh$ sh ~/OSSE11/bin/punition.sh 3 5 '*'
* * * * *
* * * * *
* * * * *
```

Exercice 2 - Mise en place d'un wrapper

L'exécutable `/pub/FISE_OSSE11/shell/iacmp/iacmp` compare les chaînes de caractères passées en argument et les écrit sur le flux de sortie standard, triées par ordre alphabétique.

Pour fonctionner, il a besoin des variables d'environnement suivantes :

`IacmpDir` qui doit contenir la valeur `/pub/FISE_OSSE11/shell/iacmp`,
`LD_LIBRARY_PATH` qui doit contenir la valeur `/pub/FISE_OSSE11/shell/iacmp/lib`.

1. Lancez cet exécutable sans modifier votre environnement, avec les arguments `'chat roux'`, `'chat blanc'` et `'chat noir'`.

```
sh$ LD_LIBRARY_PATH=/pub/FISE_OSSE11/shell/iacmp/lib \  
    IacmpDir=/pub/FISE_OSSE11/shell/iacmp \  
    /pub/FISE_OSSE11/shell/iacmp/iacmp \  
    'chat roux' 'chat blanc' 'chat noir'
```

note : Les `\` sont suivis d'un saut de ligne (touche `<ENTER>`), ce qui permet de saisir la commande sur plusieurs lignes. Il est aussi possible de saisir toute la commande sur une seule ligne en ignorant les `\` mais c'est nettement moins agréable à l'écran.

2. Ouvrez un nouveau terminal et configurez votre environnement afin de pouvoir lancer la commande `/pub/FISE_OSSE11/shell/iacmp/iacmp` sans les affectations des variables d'environnement.

```
sh$ export LD_LIBRARY_PATH=/pub/FISE_OSSE11/shell/iacmp/lib  
sh$ export IacmpDir=/pub/FISE_OSSE11/shell/iacmp  
sh$ /pub/FISE_OSSE11/shell/iacmp/iacmp 'chat roux' 'chat blanc' 'chat noir'  
chat blanc  
chat noir  
chat roux
```

Fermez ce terminal.

3. Écrivez le script
`~/OSSE11/bin/iacmp` qui va jouer le rôle de *wrapper* pour `/pub/FISE_OSSE11/shell/iacmp/iacmp` (c'est-à-dire de script qu'on va lancer à la place de `/pub/FISE_OSSE11/shell/iacmp/iacmp` pour des raisons de facilité).

Ce *wrapper* va d'abord configurer l'environnement, puis lancer `/pub/FISE_OSSE11/shell/iacmp/iacmp` avec `exec` en lui transmettant ses arguments.

Ainsi la commande ci-dessous doit fonctionner quelque soit le répertoire dans lequel on se trouve.

```
sh$ sh ~/OSSE11/bin/iacmp 'chat roux' 'chat blanc' 'chat noir'  
chat blanc  
chat noir  
chat roux
```

4. Le problème de la commande précédente est qu'il faut utiliser explicitement la commande `sh` avec le chemin complet de notre *wrapper*.

Ouvrez un nouveau terminal, et faites en sorte de pouvoir appeler le *wrapper* directement via `iacmp`.

```
sh$ export PATH="$HOME/OSSE11/bin:$PATH" # attention, PAS D'ESPACE
```

```
sh$ iacmp 'chat roux' 'chat blanc' 'chat noir'  
bash: iacmp: Permission denied
```

```
sh$ chmod -v u+x ~/OSSE11/bin/iacmp      # ajout des droits d'exécution
sh$ iacmp 'chat roux' 'chat blanc' 'chat noir'
chat blanc
chat noir
chat roux
```

Fermez ce terminal.

note : On pourrait faire en sorte que le changement sur `$PATH` soit effectif dans tous les terminaux sans avoir à saisir la commande avec `export` à chaque fois. Pour cela, il suffirait de modifier le fichier `~/.bashrc` ou `~/.profile`.

NE faites PAS cela pour l'instant.

Exercice 3 - Boucles, shift, set

L'objectif de cet exercice est d'écrire une commande

sortarg $s_1 s_2 \dots s_n$

qui réécrit les chaînes de caractères s_i triées par ordre croissant sur une seule ligne. On propose pour cela trois méthodes.

1. Écrivez le script `~/OSSE11/bin/sortarg1` dont l'algorithme est :

- Écrire les arguments un par ligne dans le fichier `/tmp/1` ;
- Trier le fichier `/tmp/1` à l'aide de la commande **sort** et stocker le résultat dans `/tmp/2` ;
- Écrire le contenu du fichier `/tmp/2` sans les sauts de lignes sur le flux de sortie standard.

note : On utilisera pour cela les commandes **read** et **echo -n**.

Testez le avec la commande :

```
sh$ sh ~/OSSE11/bin/sortarg1 'chat roux' 'chat blanc' 'chat noir'
chat blanc chat noir chat roux
```

2. Écrivez le script `~/OSSE11/bin/sortarg2`, similaire à `sortarg1`, mais qui n'utilise pas de fichiers intermédiaires.

note : Utilisez des `|`.

3. On suppose pour cette question que les arguments s_i ne contiennent pas d'espace. Écrivez le script `~/OSSE11/bin/sortarg3`, qui trie les arguments selon l'algorithme :

tant que *il reste des arguments à traiter* **faire**

- 1 | $\text{min} \leftarrow$ plus petit des arguments ;
 - 2 | $\text{args} \leftarrow$ chaîne contenant tous les autres arguments ;
 - 3 | Écrire min argument sur la sortie standard ;
 - 4 | Mettre à jour la liste des arguments avec **set -- \$args** ;
- fin**

Pourquoi doit-on supposer que les s_i sont sans espace dans cette approche ?

4. Si vous avez le temps, proposez une variante du script précédent où l'utilisation de **set** et la boucle externe sont remplacés par un appel récursif du script.

Exercice 4 - Fonctions, if, . (source)

Le but de cet exercice est de créer le fichier `~/OSSE11/bin/s3tool.sh`, regroupant diverses fonctions que nous réutiliserons pour l'exercice suivant.

1. Recopiez dans le fichier `~/OSSE11/bin/s3tool.sh` la fonction `s36str` donnée ci-dessous et disponible dans le fichier `/pub/FISE_OSSE11/shell/s36str.sh`.

```
# usage: s36str a b c => s36str a a b b c c
# usage: s36str k1 s1 k2 s2 k3 s3
# affiche les chaînes s(i) en fonction de l'ordre
# des clés k(i). Les clés sont comparées de façon
# lexicographique.

s36str() {
    if [ $# -eq 3 ] ; then
        s36str "$1" "$1" "$2" "$2" "$3" "$3"
        return 0
    fi
    if [ ! "$1" \> "$3" -a ! "$3" \> "$5" ] ; then
        echo $2 ; echo $4 ; echo $6
    elif [ ! "$1" \> "$5" -a ! "$5" \> "$3" ] ; then
        echo $2 ; echo $6 ; echo $4
    elif [ ! "$3" \> "$1" -a ! "$1" \> "$5" ] ; then
        echo $4 ; echo $2 ; echo $6
    elif [ ! "$3" \> "$5" -a ! "$5" \> "$1" ] ; then
        echo $4 ; echo $6 ; echo $2
    elif [ ! "$5" \> "$1" -a ! "$1" \> "$3" ] ; then
        echo $6 ; echo $2 ; echo $4
    else
        echo $6 ; echo $4 ; echo $2
    fi
}
```

2. Dans un nouveau terminal, ajoutez `s36str` aux commandes utilisables dans le shell, puis testez la.

```
sh$ s36str cc 33 bb 22 aa 11
s36str: command not found
sh$ source ~/OSSE11/bin/s3tool.sh
sh$ s36str cc 33 bb 22 aa 11
11
22
33
sh$ s36str cc bb aa
aa
bb
cc
```

3. Dans `s3tool.sh`, ajoutez la fonction `s36int` similaire à `s36str`, mais comparant des clés à valeurs entières.

Testez `s36int`.

```
sh$ source ~/OSSE11/bin/s3tool.sh # s3tool.sh a changé depuis le dernier appel à source
sh$ s36int 01 03 2
01
2
03
```

4. Ajoutez maintenant dans `s3tool.sh` la fonction `isInt` ci-dessous.

```
# usage: isInt str  
# returns 0 if str represents an integer, 1 otherwise  
  
isInt() {  
    local n="$1"  
    local m=$(echo "$n" | sed -r -e '1s/[+-]?[0-9]+//')  
    test -z "$m"  
}
```

Testez la.

```
sh$ isInt -01 ; echo $?  
0  
sh$ isInt a01b ; echo $?  
1  
sh$ if isInt -1 ; then echo 'oui' ; else echo 'non' ; fi  
oui
```

5. Essayez de comprendre comment `isInt` fonctionne et trouvez son principal défaut.

Exercice 5 - Utilisation de s3tool.sh

Nous allons maintenant créer quelques scripts utilisant ce qui a été fait à l'exercice précédent dans le fichier `s3tool.sh`.

Pour pouvoir utiliser le contenu de `s3tool.sh`, il faut le *sourcer* grâce à

```
. ~/OSSE11/bin/s3tool.sh
```

ou

```
source ~/OSSE11/bin/s3tool.sh
```

sachant que seule la première syntaxe est acceptée par `sh`.

Les scripts que nous allons écrire seront placés dans le répertoire `~/OSSE11/bin`. La première étape consistera à vérifier le nombre et (éventuellement) la nature des arguments. Si un problème est détecté, le script devra afficher un message explicatif sur la sortie d'erreur et s'arrêter en renvoyant un code d'erreur (en appelant `exit` suivi d'un entier non nul).

1. Écrivez le script `s3s`, qui prend en argument trois chaînes de caractères, et qui les réécrit triées par ordre croissant.

Testez votre script :

```
sh$ sh ~/OSSE11/bin/s3s 'chat roux' 'chat blanc' ; echo $?
error: usage: s3s str1 str2 str3
1
sh$ sh ~/OSSE11/bin/s3s 'chat roux' 'chat blanc' 'chat noir' ; echo $?
chat blanc
chat noir
chat roux
0
```

2. Écrivez le script `s3e`, qui prend trois entiers en argument et les réécrit triés par ordre croissant.

Testez le, notamment avec les commandes :

```
sh$ sh ~/OSSE11/bin/s3e 'chat roux' 'chat blanc' ; echo $?
error: usage: s3e int1 int2 int3
1
sh$ sh ~/OSSE11/bin/s3e 01 2 'chat noir' ; echo $?
error: 'chat noir' is not an integer
2
sh$ sh ~/OSSE11/bin/s3e 01 03 2
01
2
03
```

3. Écrivez le script `s3f`, qui prend en argument trois chemins vers des fichiers réguliers, et qui les réécrit triés par ordre croissant de taille.

Testez le, notamment avec les commandes :

```

sh$ sh ~/OSSE11/bin/s3f /lib /dev /tmp ; echo $?
error: '/lib' is not a regular file
3
sh$ cd /pub/FISE_OSSE11/shell/s3/petit
sh$ sh ~/OSSE11/bin/s3f grand moyen corrige ; echo $?
error: 'corrige' does not exist
2
sh$ sh ~/OSSE11/bin/s3f grand moyen petit
petit
grand
moyen
sh$ cd -

```

4. Écrivez le script s3d, qui prend en argument trois chemins vers des répertoires, et qui les réécrit triés par ordre croissant du nombre de fichiers qu'ils contiennent. On comptabilisera tous les fichiers contenus dans l'arborescence de chaque répertoire.

Testez le, notamment avec les commandes :

```

sh$ sh ~/OSSE11/bin/s3d /lib /dev /dev/null ; echo $?
error: '/dev/null' does not exist or is not a directory
2
sh$ ( cd /pub/FISE_OSSE11/shell/s3 ; sh ~/OSSE11/bin/s3d grand moyen petit )
petit
moyen
grand

```

5. Écrivez le script s3flm, qui prend en argument trois chemins vers des fichiers, et qui les réécrit triés par ordre croissant du second mot de la première ligne du fichier.

On supposera qu'un tel mot existe toujours, et on utilisera la commande *built-in* `read` pour le récupérer.

Testez votre script s3flm avec les commandes :

```

sh$ sh ~/OSSE11/bin/s3flm /etc/group /etc/group /etc/shadow ; echo $?
error: '/etc/shadow' is not readable
3
sh$ ( cd /pub/FISE_OSSE11/shell/s3/petit ; \
      sh ~/OSSE11/bin/s3flm petit moyen grand )
grand
moyen
petit

```

Exercice 6 - sleep sort

Le but de cet exercice est d'écrire le script `sleepsort`, de sorte que

`sleepsort` n_1 n_2 \dots n_k

écrive sur la sortie standard les entiers positifs n_i dans l'ordre croissant.

Une stratégie pour arriver à ce résultat est de procéder comme suit. Pour chaque n_i , on va lancer en arrière plan un groupe de deux commandes, à savoir un `sleep` suivi d'un `echo` :

`{ sleep n_i ; echo n_i ; } &`

Ensuite, on lancera k fois la commande `wait`.

1. Écrivez ce script et testez le.
2. Que se passe-t-il si on enlève les `{ }` et le `;` final au niveau du groupe de commandes ?
3. Quel est son temps d'exécution ?
4. À quoi servent les appels à `wait` ?

Exercice 7 - Script, find, stat, sort

L'objectif de cet exercice est de réaliser le script

pidfiles **dir**

qui affiche la liste des utilisateurs ayant des fichiers réguliers dans le répertoire **dir**, avec en plus le nombre de fichiers.

1. Écrivez le script `~/OSSE11/bin/pidfiles`.

Après les vérifications portant sur son argument, son algorithme est :

- a. Cherchez les utilisateurs ayant des fichiers réguliers dans le répertoire **dir**.
- b. Éliminez les doublons parmi les utilisateur.
- c. Pour chaque utilisateur, cherchez son nombre de fichiers.

Testez le avec la commande :

```
sh$ sh ~/OSSE11/bin/pidfiles /var/tmp
john.doe 2
...
```

2. Ajoutez au script l'option **-s** qui, en plus du nombre de fichiers, indique aussi la taille cumulée en octets de ces fichiers.

Considérons le script `util.sh` ci-dessous.

```
# util.sh

t() {
  n="$1"
  m=$(echo "$n" | sed -e '1s/[+-]\?[0-9]\+//')
  test -z "$m"
}

f() {
  x=$1
  if [ $x -le 1 ] ; then
    echo 1
  else
    n=$((x-1))
    echo $(( x * $(f $n) ))
  fi
}
```

1. Indiquez ce que fait la fonction `t`.
2. Indiquez ce que fait la fonction `f`.
3. En utilisant `util.sh`, écrivez un script shell qui prend en argument deux entiers n et m , et qui écrit sur le flux standard de sortie la valeur de $n!/m!$.

Pour cet exercice, nous allons travailler avec les hypothèses suivantes :

- Le fichier `/tmp/etudiants.txt` existe et contient la liste des étudiants convoqués à un TP noté. Plus précisément, ce fichier contient une ligne par étudiant, et chaque ligne contient uniquement le *nom d'utilisateur* d'un étudiant (généralement de la forme `prenom.nom`).
- Il existe un certain nombre de fichiers de la forme `/tmp/xxxx.tar.gz`, avec un *nom d'utilisateur* à la place de `xxxx`.
- Le répertoire `/tmp` ne contient rien d'autre.

Avant de procéder à la correction du TP noté, nous souhaitons préparer un nouveau répertoire `/tmp/correction`. Pour ce faire, nous voulons disposer d'un script `extract.sh` dont le travail est d'automatiser les traitements suivants :

- afficher un message d'erreur si `/tmp/correction` existe déjà, et créer ce répertoire sinon ;
- pour chaque *nom d'utilisateur* présent dans `/tmp/etudiants.txt`, créer un répertoire de la forme `/tmp/correction/xxxx` avec le nom de l'utilisateur à la place de `xxxx` ;
- pour chaque répertoire de la forme `/tmp/correction/xxxx` ainsi créé, déplacer le rendu `/tmp/xxxx.tar.gz` dans ce répertoire ou afficher un message d'erreur contenant le *nom d'utilisateur* de l'étudiant pour lequel il n'y a pas de rendu ;
- pour chaque rendu de la forme `/tmp/xxxx.tar.gz` toujours présent, afficher un message d'erreur contenant le nom du fichier qui ne devrait pas être présent.

1. Écrivez le code du script shell `extract.sh`.

2. Plutôt que d'utiliser systématiquement le répertoire `/tmp`, on souhaite pouvoir passer le nom d'un répertoire en argument du script `extract.sh`.

Listez les modifications à apporter à votre code pour obtenir cette fonctionnalité.

Exercice 10 - Code source auto-exécutable, #!

Copiez le fichier `/pub/FISE_OSSE11/shell/hello.c` chez vous et ajoutez `#!cce` en première ligne et première colonne. Rendez `hello.c` exécutable (`chmod u+x`).

L'objectif de cet exercice est que la commande

`./hello.c 2`

compile le fichier C sans la première ligne et lance l'exécutable généré si il n'y a pas eu d'erreurs de compilation.

1. Écrivez le script `~/OSSE11/bin/cce`. Il a un seul argument, qui est le chemin d'un fichier C. On créera l'exécutable `/tmp/N.a.out` où N est le PID du processus `cce`, et on le supprimera après son exécution.

aide :

- La commande `gcc -x c -` compile le code issu du flux standard d'entrée en supposant qu'il s'agit d'un code en C.
- La commande `sed -e 1d file` écrit sur le flux standard de sortie le contenu du fichier `file` sans sa première ligne.
- L'expansion de `$$` donne le PID du processus courant.

2. Vérifiez que le script fonctionne.

```
sh$ sh ~/OSSE11/bin/cce hello.c
sh$ cce hello.c
sh$ ./hello.c
```

3. Faites en sorte que le script `cce` transmette ses paramètres à l'exécutable, et que son code de retour soit :

- 0 si l'exécution du programme a renvoyé le statut 0,
- 1 si l'exécution du programme a renvoyé un statut différent de 0,
- 2 si la compilation a échoué.

```
sh$ ./hello.c 2 good ; echo "status = $?"
... #warning ...
good
good
status = 0
```

```
sh$ ./hello.c 2 good x ; echo "status = $?"
... #warning ...
/tmp/N.a.out: too many args.
status = 1
```

4. Ajoutez au script les fonctionnalités suivantes :

- a. Si la variable d'environnement `SILENCE` n'est pas vide, les messages du compilateur ne sont pas affichés.
- b. Si la variable d'environnement `DEBUG` n'est pas vide, le code C est compilé avec l'option `-g` et l'exécution est lancée dans le débogueur `gdb`.

Exercice 11 - Archive auto-décompressable

L'objectif de cet exercice est de réaliser le script `shzip` qui crée des archives auto-décompressables.

On crée l'archive `archive.shz` qui contient l'arborescence du répertoire source `dir-src` via :

```
sh$ shzip archive.shz dir-src
```

On peut ensuite la restaurer ailleurs grâce à :

```
sh$ mkdir dir-dest
sh$ ./archive.shz dir-dest
```

Après ces commandes, le répertoire destination `dir-dest` contient les mêmes fichiers que le répertoire source `dir-src`.

1. Le format `base64` permet d'encoder n'importe quelle suite d'octets en utilisant uniquement 64 des caractères affichables : les minuscules, majuscules, chiffres, plus deux autres caractères (généralement `+` et `/` pour les pièces jointes dans les emails, ou encore `-` et `_` dans les URL).

La commande `base64` est l'outil usuel pour gérer ce format.

- a. Créez (ou vous voulez) le fichier `file1` qui contient une ligne avec trois caractères `'a'`.

```
sh$ echo 'aaa' >file1
```

- b. Encodez le fichier `file1` en `base64`.

```
sh$ base64 file1
```

- c. Stockez le résultat dans le `/tmp/file1.64`.

```
sh$ base64 file1 >/tmp/file1.64
```

- d. Décodez le fichier `/tmp/file1.64` et vérifiez que le résultat est cohérent.

```
sh$ base64 -d /tmp/file1.64
```

- e. Comme la plupart des commandes Unix, `base64` travaille par défaut avec les flux standards d'entrée et de sortie.

```
sh$ cat /tmp/file1.64 | base64 -d
```

2. Écrivez le début du script `shzip`, c'est-à-dire la partie qui vérifie que le nombre et la nature des arguments sont corrects.
3. Ajoutez au script la fonction `encodefile`, qui prend en argument le chemin vers un fichier, et qui écrit sur le flux de sortie standard une suite de commandes permettant de recréer le fichier dans `/tmp` à partir de son encodage en `base64`.

Pour le fichier `file1` précédent, le code généré devrait être

```
base64 -d > /tmp/file1 << EOF
YWFhCg==
EOF
chmod u=rw,go=r /tmp/file1
```

En bas du script, ajoutez les commandes

```
encodefile file1 > /tmp/archive.shz
encodefile file2 >> /tmp/archive.shz
```

puis testez votre script :


```

sh$ echo aaa > file1
sh$ echo bbbb > file2
sh$ echo bbbb >> file2
sh$ shzip /tmp/archive.shz .
sh$ cat /tmp/archive.shz # vérification à l'oeil
sh$ ( cd /tmp ; rm -fv file* ; sh /tmp/archive.shz . )
sh$ cat /tmp/file1
aaa
sh$ cat /tmp/file2
bbbb
bbbb

```

4. Dans ce qui précède, on a utilisé `/tmp` en dur dans le contenu de l'archive. Modifier `encodefile` afin que le répertoire passé en argument à `archive.shz` soit utilisé à la place.
5. Ajoutez au script la fonction `encodedir`, qui prend en argument le chemin vers un répertoire `dir`, et qui écrit dans l'archive (premier argument de `shzip`) le code pour recréer tous les fichiers réguliers contenu dans `dir`.

Les répertoires et les autres fichiers seront ignorés avec un message d'avertissement (sur la sortie d'erreur).

En bas du script :

- enlevez les commandes `encodefile` de la question précédente,
- appelez `encodedir` à la place.

Testez votre script :

```

sh$ mkdir -v 123
sh$ mv -v file1 file2 123/
sh$ cat 123/file1 123/file2 > '123/file 3'
sh$ shzip /tmp/archive.shz 123
sh$ cat /tmp/archive.shz # vérification à l'oeil
sh$ rm -frv /tmp/testdir
sh$ mkdir -v /tmp/testdir
sh$ ( cd /tmp/testdir ; sh /tmp/archive.shz . )
sh$ cat /tmp/testdir/file1
aaa
sh$ cat /tmp/testdir/file2
bbbb
bbbb
sh$ cat '/tmp/testdir/file 3'
aaa
bbbb
bbbb

```

6. Modifiez la fonction `encodedir` pour qu'elle traite aussi les répertoires (attention aux cas particuliers `.` et `..`).

Testez votre script :

```

sh$ mkdir -pv 123/123
sh$ cat 123/file1 123/file1 > 123/123/file1bis
sh$ cat 123/file2 123/file2 > 123/123/file2bis
sh$ shzip /tmp/archive.shz 123
sh$ cat /tmp/archive.shz # vérification à l'oeil
sh$ rm -frv '/tmp/new testdir'
sh$ mkdir -v '/tmp/new testdir'
sh$ sh /tmp/archive.shz '/tmp/new testdir'
sh$ cat '/tmp/new testdir/file 3'
aaa
bbbb
bbbb
sh$ cat '/tmp/new testdir/123/file2bis'
bbbb
bbbb
bbbb
bbbb

```

7. Modifiez le script `shzip` pour ajouter l'entête `#!/bin/sh` à l'archive générée et la rendre exécutable.

Testez votre script :

```

sh$ shzip /tmp/archive.shz 123
sh$ cat /tmp/archive.shz # vérification à l'oeil
sh$ rm -frv '/tmp/new testdir'
sh$ mkdir -v '/tmp/new testdir'
sh$ /tmp/archive.shz '/tmp/new testdir'

```

8. Ajoutez au script `shzip` l'option facultative `-z` qui compresse les fichiers avec `gzip` avant de les encoder.