Université Paul Sabatier **UE : ED3INFFM**
L2 Informatique **Systèmes 1**

Contrôle terminal – session 1

Mercredi 17 décembre 2014 – 2h00 – Sans document

L'épreuve comporte quatre exercices indépendants répartis en trois parties auxquelles vous devez répondre sur trois copies différentes, en n'oubliant pas de mettre votre nom et la partie (1, 2 ou 3) sur chaque copie. Même si vous ne répondez par à une ou plusieurs parties, vous devez tout de même rendre la ou les copies correspondantes. L'écriture des scripts en shell de Bourne sous-entend l'écriture de vérifications appropriées sur les paramètres, ainsi qu'une gestion cohérente des codes de retour. Il est conseillé de commencer par lire l'ensemble de l'énoncé et de tenir compte du barème. La propreté de la copie sera prise en compte, en particulier en ce qui concerne la lisibilité des caractères ′ et `\`.

PARTIE 1

Exercice 1 (6 points)

1. Réaliser chacune des taches suivantes à l'aide d'une seule ligne de commande :
   (a) Effacer tous les fichiers du répertoire `/tmp` dont le nom commence par une majuscule ou le caractère `$`, suivi d'une lettre (majuscule ou minuscule), suivi d'une chaîne quelconque éventuellement vide et se terminant par un espace suivi d'un chiffre.
   (b) Pour chacun des fichiers dont la désignation se trouve dans le fichier `liste.txt` (une désignation par ligne), afficher la désignation du fichier suivi du caractère ' :' suivi du nombre de lignes où apparaît le mot `commande` dans le fichier.
   (c) Écrire dans le fichier `tailles.txt` les noms des fichiers ordinaires non cachés du répertoire courant précédés de leur taille en octet. Si le fichier `tailles.txt` existe déjà, son contenu doit être écrasé.
2. Expliquer de manière précise ce que font les commandes suivantes :
   (a) `grep 'z' g.txt >> f.txt`
   (b) `nb=8; grep "[a-zA-Z]\{$nb\}" fichier.txt`
   (c) `a=da; b=bonjour ; echo '$a' "$b" '${a}te | cut -f4 -d ' '`

PARTIE 2

Exercice 2 (6 points)

On souhaite mettre en place une gestion simplifiée d'un historique des commandes en shell de Bourne en s'inspirant de ce qui existe dans d'autres shells comme `csh` ou `bash`. Pour cela, écrire un script dont la structure générale reproduit la boucle de l'interpréteur, c'est-à-dire qui :
- affiche le message d'invite ;
- lit une ligne de commande ;
`Tant que` la ligne de commande est différente de `exit`
`Faire`
- exécute la ligne de commande ;
- affiche le message d'invite ;
- lit la ligne de commande suivante ;

`FinTantque`

À cette structure générale, ajoutez ce qui est nécessaire pour que le script :

1. si la ligne de commande est réduite à `h`, affiche les commandes qui ont été éxécutées depuis le lancement du script ; chaque ligne de commande doit être précédée de son numéro, la première ligne commande ayant été exécutée portant le numéro 1 ; les lignes de commande exécutées sont stockées dans un fichier de nom `.historique` (une ligne de commande par ligne du fichier) ;
2. si la ligne de commande est réduite à `!!`, exécute la ligne de commande la plus récente de l'historique ;
3. si la ligne de commande est réduite à `!`num, exécute la ligne de commande numéro num de l'historique (si num est trop élevé, un message d'erreur doit être affiché mais l'exécution du script ne doit pas être interrompue) ;
4. dans tous les autres cas, exécute la ligne de commande (elle est constituée de commandes écrites en shell de Bourne ; attention, elle peut contenir des métacaractères du shell).

Dans les cas 2, 3 et 4, ajoute à la fin du fichier `.historique` la ligne de commande qui va être exécutée (pas son résultat)

Au début du script, le fichier `.historique` doit être vide. Les commandes « spéciales », `h`, `!!` et `!`num, ne doivent pas être enregistrées dans l'historique. Le message d'invite doit être le numéro de la ligne de commande qui est attendue suivi du caractère > suivi d'un espace.

Voici un exemple d'utilisation du script de nom `hsh.sh` :

```
$ hsh.sh
1> date
Tue Dec 11 00:31:20 CET 2012
2> m=bonjour
3> echo $m
bonjour
4> h
     1  date
     2  m=bonjour
     3  echo $m
4> !!
bonjour
5> !1
Tue Dec 11 00:31:55 CET 2012
6> h
     1  date
     2  m=bonjour
     3  echo $m
     4  echo $m
     5  date
7> exit
$
```

PARTIE 3

Exercice 3 (4 points)

Cet exercice propose d'implanter en shell de Bourne des scripts permettant de chiffrer et de déchiffrer des fichiers textes selon méthode ROT13 (rotate by 13 places).

Le méthode ROT13 est une méthode simpliste de chiffrement de texte qui décale de 13 caractères chaque lettre (majuscule et minuscule) du fichier texte à chiffrer. Dans cette méthode, le codage et le décodage se font exactement de la même manière.

La correspondance entre les caractères en clair et chiffrés est la suivante :

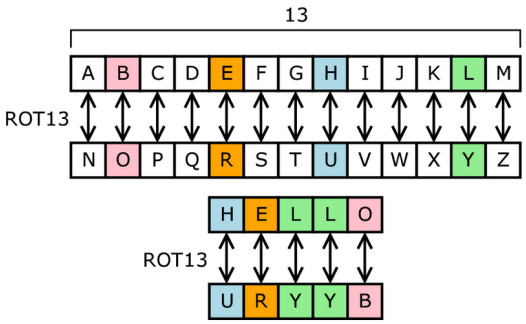
Caractères non chiffrés	A	B	C	…	L	M	N	…	Y	Z
Caractères chiffrés	N	O	P	…	Y	Z	A	…	L	M

Lettres majuscules

Caractères non chiffrés	a	b	c	…	l	m	n	…	y	z
Caractères chiffrés	n	o	p	…	y	z	a	…	l	m

Lettres minuscules

Par exemple :



Chiffré avec ROT13, le mot « HELLO » devient « URYYB » (et inversement).

1. Écrire le script `rot13.sh` dont la syntaxe d'appel est la suivante :
   `rot13.sh nom_fichier`
   qui décale de 13 caractères les lettres du fichier de désignation `nom_fichier` et affiche sur `stdout` le résultat (le contenu du fichier avec toutes ses lettres décalées).
   Indication : vous pouvez utiliser la commande `tr` avec des intervalles comme `A-E`.
   Par exemple :
   Pour le fichier texte `texte1.txt`, le script `rot13.sh` produit :

```
$ cat texte1.txt
Votre mission si vous     l'acceptez  est de réaliser cet exercice sans erreur ...

Mission de la plus haute importance !!!!
$ rot13.sh texte1.txt
Ibger zvffvba fv ibhf     y'npprcgrm  rfg qr eényvfre prg rkrepvpr fnaf reerhe ...

Zvffvba qr yn cyhf unhgr vzcbegnapr !!!!
```

2. Écrire le script `estRot13.sh` dont la syntaxe d'appel est la suivante :
   `estRot13.sh nom_fichier nom_dictionnaire`
   qui retourne 0 si `nom_fichier` est chiffré, 1 si `nom_fichier` n'est pas chiffré et une valeur >1 en cas d'erreur. Le fichier de désignation `nom_fichier` est considéré chiffré si après application du codage ROT13, au moins 50% des

mots qu'il contient appartiennent au dictionnaire `nom_dictionnaire`. Le dictionnaire `nom_dictionnaire` est un fichier texte contenant un mot par ligne écrit en majuscules.

Exemple de contenu d'un fichier dictionnaire :

```
A
AH
AI
...
MELODRAME
MELOMANES
...
```

Exemple d'appel :

```
$ estRot13.sh texte1.txt dico.txt
```

Exercice 4 (4 points)

Écrire en shell de Bourne le script `profondeur.sh` dont la syntaxe d'appel est la suivante :
`profondeur.sh repertoire`
qui calcule et affiche sur `stdout` la profondeur, c-à-d la longueur de la plus longue branche de l'arborescence de racine `repertoire`. Uniquement les répertoires et sous-répertoires sont comptabilisés. La racine compte pour 1 dans le calcul de la profondeur.

Dans les exemples suivants, les répertoires sont écrits en majuscules et les fichiers en minuscules.

Pour l'arborescence :

```
TEST1
+- f1
+- f2
```

L'appel au script `profondeur.sh` suivant produit :

```
$ profondeur.sh TEST1
1
```

Et pour l'arborescence :

```
TEST2
+- f1
+- FILS1
+- f2
+- FILS2
|  +- S_FILS2
```

L'appel au script `profondeur.sh` suivant produit :

```
$ profondeur.sh TEST2
3
```

1. Écrire une fonction récursive `profond` qui retourne la profondeur de l'arborescence passée en paramètre.
2. Puis écrire le script `profondeur.sh` qui affiche sur `stdout` la profondeur à l'aide de la fonction `profond`.

Remarque : les informations données sur la page suivante n'impliquent pas que les commandes indiquées doivent obligatoirement apparaître dans les réponses aux exercices, ni que ce sont les seules qui doivent être utilisées.

basename	`basename` *désignation_fichier_ou_répertoire* affiche sur `stdout` la chaîne de caractères constituée de la désignation du fichier ou du répertoire privée des caractères situés avant le dernier caractère `/`. Par exemple, la commande `basename /users/linfg/l2inf201/REP/fich.txt` produit l'affichage de `fich.txt`.
cat	`cat` [*fichier*] affiche sur `stdout` le contenu du fichier dont le nom est passé en paramètre (ou de `stdin` par défaut). Avec l'option `-n`, les lignes affichées sont précédées de leur numéro, la première ligne portant le numéro 1.
cp	`cp` *désignation_fichier* *désignation_répertoire* copie le fichier *désignation_fichier* dans le répertoire *désignation_répertoire*. La date et l'heure de dernière modification du fichier créé correspondent à la date et à l'heure de l'exécution de la commande.
cut	`cut -c` *liste* [*fichier* `...`] ne retient sur chaque ligne que les caractères situés aux positions spécifiées par *liste*. `cut -f` *liste* `-d` *délimiteur* ne retient sur chaque ligne de `stdin` que les champs spécifiés par *liste* séparés par *délimiteur*. Le numéro du premier champ sur la ligne est 1.
date	La commande `date` affiche sur `stdout` la date et l'heure courantes sous la forme de six champs, deux champs successifs étant séparés par un espace, le quatrième champ étant l'heure.
echo	`echo $var` affiche la valeur de la variable `var` suivie d'un caractère `'\n'`. `echo "$var\c"` affiche la valeur de la variable `var`.
eval	La commande `eval` permet d'effectuer une double évaluation (avec interprétation des métacaractères du shell lors des deux évaluations).
exit	La commande `exit` [*code*] permet de quitter le script en retournant le code *code*. Sans paramètre, le code retourné est le code de retour de la dernière commande précédemment exécutée.
expr	La commande `expr` permet de réaliser des opérations entières grâce aux opérateurs `+`, `-`, `\*`, `/`, `%`.
grep	`grep` *expression* [*fichier* `...`] affiche sur `stdout` les lignes des fichiers passés en paramètres (les lignes de `stdin` par défaut) qui correspondent au modèle décrit par l'expression régulière *expression*. Dans cette dernière, `^` désigne le début de la ligne, `$` la fin de la ligne, `.` un caractère quelconque, `\{`*n*`\}` une répétition de *n* fois le caractère qui précède, `*` une répétition (de longueur quelconque éventuellement vide) du caractère qui précède et `\` « protège » le caractère qui suit. Le code de retour de `grep` est nul si au moins une occurrence est trouvée. Avec l'option `-n`, chaque ligne qui correspond à l'expression régulière est précédée de son numéro (la première ligne a le numéro 1), suivi du caractère `:` Avec l'option `-c`, la commande `grep` affiche uniquement sur `stdout` le nombre de lignes qui correspondent à l'expression régulière.
head	Affichage sur `stdout` du début du fichier passé en paramètre (de `stdin` par défaut). `head -n 5 fich` produit l'affichage des cinq premières lignes de `fich` (si elles existent).
ls	La commande `ls REP` affiche sur `stdout` la liste des noms de fichiers et de répertoires contenus dans le répertoire `REP`. La commande `ls -al REP` affiche sur `stdout` des renseignements détaillés concernant le contenu complet du répertoire `REP`. Sans paramètre, c'est le répertoire courant qui est considéré. Sans l'option `-a`, les fichiers cachés sont ignorés. Si une liste de noms de fichiers est passée en paramètres à la commande `ls -l`, alors les renseignements détaillés concernant ces fichiers sont affichés sur `stdout` (un fichier par ligne). Les renseignements détaillés qui sont affichés sont constitués de neuf champs (la taille en octets est le cinquième champ et le nom du fichier ou du répertoire est le neuvième champ), deux champs successifs étant séparés par une séquence non vide d'espaces.
mkdir	`mkdir` *répertoire* [`...`] permet de créer un ou plusieurs répertoires passés en paramètres.
read	La commande `read a` permet d'affecter à la variable `a` une ligne provenant de `stdin`.
rm	`rm` *fichier* [`...`] permet de supprimer un ou plusieurs fichiers passés en paramètres. `rm -rf` *désignation_répertoire* [`...`] supprime les répertoires passés en paramètres ainsi que leur contenu (fichiers et sous-répertoires).
tail	Affichage sur `stdout` de la fin du fichier passé en paramètre (de `stdin` par défaut). `tail -5 fich` produit l'affichage des cinq dernières lignes de `fich` (si elles existent).
test	À la suite de la commande `test`, peuvent apparaître les opérateurs logiques `-a`, `-o`, `!`, les comparateurs d'entiers `-eq`, `-ne`, `-gt`, `-ge`, `-lt`, `-le`, les comparateurs de chaînes de caractères `=`, `!=`, les opérateurs sur un fichier ou un répertoire `-f`, `-d`, `-r`, `-w`, `-x`.
touch	`touch -r` *désignation_fichier_réf* *désignation_fichier* remplace les horodatages (date et heure d'accès et date et heure de modification) de *désignation_fichier* par les horodatages de *désignation_fichier_réf*.
tr	`tr -d` *chaîne* supprime toutes les occurrences des caractères composant *chaîne*. `tr -c -d` *chaîne* supprime toutes les occurrences des caractères n'appartenant pas à *chaîne*. `tr -s` *ch* élimine les répétitions des caractères composant *ch* en n'en laissant qu'un à la fois. `tr '`*abc*`' '`*ABC*`'` cette commande attend l'entrée de données au clavier. Si on tape `calebasse` chaque occurrence d'un `a` est remplacé par un `A`, chaque occurrence d'un `b` est remplacé par un `B` et chaque occurrence d'un `c` est remplacé par un `C`, ce qui donne dans le cas présent : `CAleBAsse`. `tr '`*A-C*`' '`*1-3*`'` cette commande attend l'entrée de données au clavier. Si on tape `AABBCC` chaque occurrence d'un `A` est remplacé par un `1`, chaque occurrence d'un `B` est remplacé par un `2` et chaque occurrence d'un `C` est remplacé par un `3`, ce qui donne dans le cas présent : `112233`.