Examen court de système d'exploitation

Département d'informatique – IUT Villetaneuse

Lundi 19 octobre 2009 – 1H30

Remarques : tous les documents de cours, td/tp sont autorisés. Le barème est indicatif. Veuillez expliquer ce que vous faites et ne pas vous contenter d'un programme.

1 Compréhension d'un texte - 3 pts

A) Que fait le code suivant ? Résumez le en un seul paragraphe de 6 lignes maximum.

```
#!/usr/bin/env bash

for myfile in /etc/r*
do
    if [ -d "$myfile" ]
    then
      echo "$myfile (dir)"
    else
      echo "$myfile"
    fi
done
x="$1"
case "${x##*.}" in
     gz)
           gzunpack ${SROOT}/${x}
           ;;
     bz2)
           bz2unpack ${SROOT}/${x}
           ;;
     *)
           echo "Archive format not recognized." ; exit ;
           ;;
esac
```

```
# Aide sur certaines notations utilisées dans le programme :
#
# ${paramètre#mot}
# ${paramètre##mot}
#    Le mot est développé pour fournir un motif, comme dans l'expansion
# des noms de fichiers. Si le motif correspond au début de la valeur du
# paramètre, alors le développement prend la valeur du paramètre après
# suppression du plus petit motif commun (cas « # »), ou du plus long
# motif (cas « ## »).
```

B) Que va t-il se passer si on passe en paramètre au script le nom all.tar.gz ?
Le code fourni est composé de deux parties. Dans la première partie qui concerne une boucle `for` on liste à l'écran les fichiers du sous-répertoire `/etc/` qui commence par un r et pour ceux qui sont des répertoires on affiche aussi l'annotation `(dir)` ; dans la deuxième partie du code qui concerne une structure cas-parmi on examine le nom du fichier passé en paramètre et selon que le suffixe du fichier est gz ou bz2 on applique un outil de décompression. Dans les autres cas on affiche que le format n'est pas reconnu. C'est la notation `${x##*.}` qui permet de considérer le plus grand prefixe commun entre le nom du fichier et le motif `*.` et faire en sorte que ce prefixe soit oublié pour qu'au final on ne considère que le suffixe. Ainsi quant on passe en paramètre le nom `all.tar.gz` il y aura un appel à `gzunpack`.

2 Langage de commande bash - 12 pts

A) La communauté des chercheurs en informatique utilise LaTeX pour écrire leurs articles. Dans ce langage la balise $$ est utilisée pour formater des mathématiques. Le code suivant :

```
$$
A_{1,1} + A_{2,1} = T_1
$$
```

produit :

$$A_{1,1} + A_{2,1} = T_1$$

On vous demande d'écrire un script Bash qui prend en paramètre un fichier et qui va contrôler que le nombre de double $$ est pair. On rappelle que % est l'opérateur *reste de la division* pour la commande `expr`.

B) Votre programme fonctionne t-il encore si la ligne est comme ceci :

```
$$A_{1,1} + A_{2,1} = T_1$$
```

Si ce n'est pas le cas, donner une solution (en expliquant ce que vous faites dans le détail) sans l'implémenter.

```
#!/bin/bash
#
if [ $# -ne 1 ]; then
    echo "usage: $0 <fichier au formatex LaTeX>"
    exit 0;
fi
if [ ! -e $1 ]; then
    echo "Le fichier $1 n'existe pas"
    exit 0;
fi
mycount=`egrep "[$]{2}" essai.txt | wc -l | awk '{ print $1 }'`
echo "$mycount"
mycount=`expr $mycount % 2`
if [ $mycount -eq 0 ]
then
   echo "Le nombre de \$\$ est balance"
else
   echo "Le nombre de \$\$ n'est pas balance"
fi

#
#
# Si nous avons plusieurs $$ sur une même ligne, le code ci dessus
# ne fonctionne plus. Nous proposons de systematiquement remplacer
# $$ par $$\n puis d'apliquer le code précédent. Mais il y a un problème
# de codage avec sed. Par exemple les codes suivants ne fournissent
# pas toujours le resultat escompté :
# ENVIRONEMENT UNIX:  conversion des retour de chariot (CR/LF) au format Unix.
# sed 's/.$//'  # assume que toutes les lignes se terminent avec CR/LF
# sed 's/^M$//' # sous bash/tcsh, enfoncer Ctrl-V puis Ctrl-M
# sed 's/\x0D$//'# fonctionne sous  ssed, gsed 3.02.80 ou plus récent
# ne permettent pas de supprimer les retour chariot dans un texte
#
#
```

B) Donner une expression régulière qui permette de tester si une ligne est sous la forme suivante (elle commence et termine par $$) :

```
$$A_{1,1} + A_{2,1} = T_1$$
```

```
> egrep "^\\$\\$.*\\$\\$\$" essai.txt
$$ee$$

Avec essai.txt:
$$ee$$
fdg $$
fsdf
sdf $$ sdfsdf
sdf
$
sdf
$$sdfds $sdfsf$$df
dfgdf $$ dfgdfg
```

C) Écire une expression régulière la plus courte possible qui teste si une ligne d'un fichier est de la forme :

```
<entier> + <entier>
ou
<entier> / <entier>
```

```
> grep -E "^[[:digit:]]*[ ]*[\+/][ ]*[[:digit:]]*$" essai.txt
123 + 23
123+23
1/4

> cat essai.txt
123 + 23
123+23
1/4
11+22-33/44
```

D) Écrire un script Bash qui prend en entrée un fichier contenant des entiers (éventuellement plusieurs par ligne) et qui fait la somme de tous les entiers. On supposera de plus que le fichier d'entrée contient des entiers espacés par exactement un seul blanc comme ceci :

```
3 2 1
5
4 6
```

```
#!/bin/bash
#
if [ $# -ne 1 ]
then
    echo "usage: $0 <fichier>"
    exit 0;
fi
if [ ! -e $1 ]; then
    echo "Le fichier $1 n'existe pas"
    exit 0;
fi

cat < $1 | while true
  do
     read ligne
     if [ "$ligne" = "" ]; then
        echo "Resultat : $somme"
        break
     fi
     # on substitue les blancs par des +
     # et on additionne les valeurs
     nb=`echo "$ligne" |sed 's/ /\+/g' | bc -l`
     echo "$nb"
     somme=$((somme+nb))
done

> /bin/bash SommeEntier.sh essai.txt
6
5
10
Resultat : 21

Avec essai.txt
3 2 1
5
4 6
```

3 Popen - 5 pts

Écrire un programme C qui prend en paramètre un fichier texte, qui appelle un script Bash ou une commande Unix qui retourne le nombre de caractères

de chaque ligne du fichier texte, puis qui au niveau du programme C fait la somme des longueurs des lignes et l'affiche. Vous utiliserez les outils que vous voulez (awk, opération sur les chaines de caractères...). Voici un exemple de ce qu'il faut produire :

```
> gcc -O4 -Wall SommePopen.c
> a.out essai.txt
5
1
3
0
0
Resultat : 9

----------------
Pour le fichier essai.txt suivant :
3 2 1
5
4 6


----------------
```

```
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
int main(int argc, char *argv[], char *arge[])
{
  char buf[256];
  char *tmp;
  FILE *pp;
  int somme=0;

  // on prepare la commande a lancer
  tmp=calloc(256,sizeof(char));
  sprintf(tmp, "%s%s", "awk \'{ print length($0) }\' ", argv[1]);
  // on lance le script qui affiche la longeur de chaque ligne
  if ((pp = popen (tmp, "r")) == NULL)
    {
      perror ("popen error");
      exit (1);
    }
  // on fait la somme des valeurs
```

```
  while(fgets (buf, sizeof (buf), pp) != NULL) {
    buf[strlen (buf) - 1] = '\0';
    printf("%s\n",buf);
    somme+=atoi(buf);
  }
  pclose (pp);
  printf("Resultat : %d\n",somme);
  return 1;
}
```