Système utilisateur
.:: Système UNIX ::

semaine 7 – révisions

Q.C.M.

Q.C.M.

1. Quel chemin est synonyme du répertoire personnel de l'utilisateur "toto" ?
   - A. -toto
   - B. ~toto
   - C. =toto

2. Le chemin ".." est synonyme du répertoire :
   - A. père du répertoire courant
   - B. racine de l'arborescence des fichiers
   - C. personnel de l'utilisateur

Q.C.M.

3. Un tube permet :
   - A. de connecter la sortie standard d'un processus vers l'entrée standard d'un autre processus
   - B. de connecter la sortie standard d'un processus sur son entrée standard
   - C. de lancer plusieurs processus indépendants en parallèle

4. Pour afficher un message sur la sortie d'erreurs standard il faut utiliser la redirection :
   - A. >&2
   - B. 2>
   - C. >stderr

Q.C.M.

5. Dans la commande "p1 || p2" où p1 et p2 sont deux processus, p2 sera exécuté si ...
   - A. p1 se termine normalement
   - B. p1 se termine anormalement
   - C. quelque soit le code de retour de p1

6. Une seule des commandes suivantes est correcte (fichier étant un fichier régulier):
   - A. ls < fichier
   - B. cat | fichier
   - C. cat < fichier
   - D. fichier | cat

Q.C.M.

7. Quelle E.R. permet de sélectionner les lignes ne contenant pas de chiffres ?
   - A. ^[^0-9]*$
   - B. ^.*[^0-9].*$
   - C. ^.*[!0-9].*$
   - D. ^[!0-9]*$

8. Quel test permet de comparer l'égalité numérique des variables A et B ?
   - A. if [$A = $B]; then ...
   - B. if [$A == $B]; then ...
   - C. if [$A -eq $B]; then ...

Q.C.M.

9. La liste des arguments d'un script shell est placée dans la variable
   - A. #
   - B. *
   - C. !
   - D. $

10. Pour retourner un chaine de caractères dans une fonction, on utilise :
   - A. return $chaine
   - B. echo $chaine
   - C. RETURN=$chaine

FIN DU QCM

Q.C.M.

1. Quel chemin est synonyme du répertoire personnel de l'utilisateur "toto" ?
   - A. -toto
   - B. ~toto
   - C. =toto

2. Le chemin ".." est synonyme du répertoire :
   - A. père du répertoire courant
   - B. racine de l'arborescence des fichiers
   - C. personnel de l'utilisateur

Q.C.M.

3. Un tube permet :

A. de connecter la sortie standard d'un processus vers l'entrée standard d'un autre processus

B. de connecter la sortie standard d'un processus sur son entrée standard

C. de lancer plusieurs processus indépendants en parallèle

4. Pour afficher un message sur la sortie d'erreurs standard il faut utiliser la redirection :

A. >&2

B. 2>

C. >stderr

Q.C.M.

5. Dans la commande "p1 || p2" où p1 et p2 sont deux processus, p2 sera exécuté si ...

   A. p1 se termine normalement
   B. p1 se termine anormalement
   C. quelque soit le code de retour de p1

6. Une seule des commandes suivantes est correcte (fichier étant un fichier régulier):

   A. ls < fichier
   B. cat | fichier
   C. cat < fichier
   D. fichier | cat

Q.C.M.

7. Quelle E.R. permet de sélectionner les lignes ne contenant pas de chiffres ?

   A. ^[^0-9]*$
   B. ^.*[^0-9].*$
   C. ^.*[!0-9].*$
   D. ^[!0-9]*$

8. Quel test permet de comparer l'égalité numérique des variables A et B ?

   A. if [$A = $B]; then ...
   B. if [$A == $B]; then ...
   C. if [$A -eq $B]; then ...

Q.C.M.

9. La liste des arguments d'un script shell est placée dans la variable
   - A. #
   - B. *
   - C. !
   - D. $

10. Pour retourner un chaine de caractères dans une fonction, on utilise :
   - A. return $chaine
   - B. echo $chaine
   - C. RETURN=$chaine

Révisions – les droits utilisateurs

=> Écrire un script qui liste tous les fichiers que les étudiants de votre promo vous autorisent à modifier :

--> /etc/password vous permet de trouver la liste des étudiants et leur home à l'aide de leur groupe

```
paul@linux:~> cat /etc/passwd
masegal:x:1327:205:Mathieu SEGAL:/home/1a/masegal:/bin/bash
[...]
```

1. Écrire une fonction getHome qui lit les lignes de /etc/password sur son entrée standard et retourne le champ *home* si le groupe correspond à l'argument passé.

2. Ecrire la fonction getListe qui retourne la liste des homes à partir du fichier /etc/password, en utilisant getHome.

Révisions – les droits utilisateurs

1. Écrire une fonction getHome qui lit les lignes de /etc/password sur son entrée standard et retourne le champ *home* si le groupe correspond à l'argument passé.

```
getHome() {
  OLDIFS=$IFS; IFS=":"
  while read login pass uid gid name home shell; do
    if [ $gid -eq $1 ]; then
        echo $home
    fi
  done
  IFS=$OLDIFS
  return 0
}
```

Révisions – les droits utilisateurs

2. Écrire la fonction getListe qui retourne la liste des homes à partir du fichier /etc/password, en utilisant getHome.

```
getListe() {
  echo `cat /etc/passwd | getHome $1 | column`
  return 0
}
```

Révisions – les droits utilisateurs

3. Rechercher à partir de cette liste dans tous les fichiers pour lequel vous avez le droit de lecture et d'écriture...

   => Les fichiers sont recherchés à l'aide de la commande find

   => Les droits sont vérifiés à l'aide de tests shell

Révisions – les droits utilisateurs

3. Rechercher à partir de cette liste dans tous les fichiers pour lequel vous avez le droit de lecture et d'écriture...

=> Les fichiers sont recherchés à l'aide de la commande find

=> Les droits sont vérifiés à l'aide de tests shell

```
main() {
  for myHome in `getListe $1` ; do
    for fichier in `find $myHome -type f -name "*" 2>/dev/null` ; do
      if [ -r $fichier ] && [ -w $fichier ] ; then
        echo $fichier
      fi
    done
  done
}
main $1
```

Révisions – les droits utilisateurs

=> Écrire un script permettant de donner tous les fichiers d'un utilisateur à un autre.

changeUser *oldUser* *newUser*

=> L'option -user de find permet de sélectionner les fichiers appartenant à l'utilisateur indiqué juste après.

Révisions – les droits utilisateurs

=> Écrire un script permettant de donner tous les fichiers d'un utilisateur à un autre.

changeUser *oldUser* *newUser*

```
#!/bin/bash
find / -user $1 -exec chown $2 {} ";"
```

Révisions – enchainement de commandes

Lancement d'une commande :

paul@linux> ls

Lancement d'une commande en arrière plan :

paul@linux>ls &

Le shell n'attend pas la fin de la commande pour rendre la main.

exemple avec ***find / -name "bash" &***

Lancement de commandes successives :

paul@linux> cd /etc/ **;** ls

Les commandes sont séparées par des « ; »

Révisions – enchainement de commandes

Enchaînement conditionnés :

paul@linux> cd /toto && ls

La commande **ls** n'est exécutée que si la commande **cd /toto** termine sans erreur

paul@linux> cd /toto || echo "répertoire inexistant"

La commande **echo** n'est exécutée que si la commande **cd /toto** se termine avec erreur

Lancement en parallèle :

Utilisation du mécanisme des tubes.

paul@linux> ls -l /etc | grep passwd

Révisions – enchainement de commandes

Regroupement de commandes :

paul@linux> (cd /toto || echo "repertoire inexistant" >&2) > resultat.txt

Les commandes placées entre parenthèses sont exécutées dans un sous processus du shell il est ainsi possible de rediriger toutes les E/S de ce nouveau shell.

Le code de retour d'un enchaînement ou d'un regroupement est celui du dernier processus exécuté

Pour forcer un code de retour il est possible d'utiliser la commande *exit n* où *n* prend la valeur à retourner :

paul@linux> (cd /toto 2> /dev/null || (echo "repertoire inexistant" >&2 ; exit 1)) && ls

La commande *ls* n'est exécutée que si le répertoire existe, les sorties sont redirigées ...

Révisions – interception de signaux

```
$ cat alarme
#!/bin/bash

trap 'echo "Le temps est écoulé !"; exit 1' SIGALRM

(/usr/bin/sleep 3; kill -SIGALRM $$ 2> /dev/null )&
echo -n "Vous disposez de 3 s. pour entrez un mot: "
read MOT
trap - SIGALRM; kill -SIGKILL $! 2> /dev/null
echo "le mot lu: $MOT"
```

Révisions – expressions régulières

=> Rechercher dans un fichier toutes la variables de type int déclarées comme suit : « int var1; » ou « int var1 = » (une seule déclaration par ligne)

=> Écrire un script qui remplace toutes les variables d'un fichier par une variable anonyme de type a0, a1, a2 ...

exemple : int maVariableBienEcrire = 2;
int maSecondeVariable;

deviendra int a0 = 2;
int a1;

Révisions – expressions régulières

=> Rechercher dans un fichier toutes la variables de type int déclarées comme suit : « int var1; » ou « int var1 = » (une seule déclaration par ligne)

```
sed -e "s/^.*int[ ]+\([^ ;=]+\)[ ;=].*$/\1/"
```

^.*int => les lignes contenant la chaîne « int »
[ ]+ => suivie d'espace(s)
[^ ;=]+ => suivi d'une chaîne sans espace ni ; ni = correspondant au nom de la variable
[;=] => suivi d'un caractère espace ou ; ou =
.*$ => se terminant de façon quelconque
=> Sont remplacées par le seul nom de la variable

Révisions – expressions régulières

=> Écrire un script qui remplace toutes les variables d'un fichier par une variable anonyme de type a0, a1, a2 ...

```
#!/bin/bash
# $1 : nom du fichier
compteur=0
for var in `sed -e "s/^.*int[ ]+\([ ;=]+\)[ ;=].*$/\1/" $1 | column`
do
    sed -e "s/$var/a$compteur/g" $1 > $1
    compteur=$(( $compteur + 1 ))
done
```

Révisions – Scripts

Écrire une commande « cat » qui affiche le contenu d'un fichier à l'aide de la commande read qui lira ligne à ligne ce fichier envoyé sur l'entrée standard. Chaque ligne sera précédée de son numéro.

Révisions – Scripts

Écrire une commande « cat » qui affiche le contenu d'un fichier passé comme argument à l'aide de la commande read qui lira ligne à ligne ce fichier envoyé sur l'entrée standard. Chaque ligne sera précédée de son numéro.

```
#!/bin/bash
lit() {
    local ligne=0
    while read ; do
            echo "$ligne : $REPLY"
            ligne=$(( $ligne + 1 ))
    done
    return 0
}
lit < $1
```

Révisions – Scripts

La commande ps -aux retourne pour chaque processus, en colonne 3 la charge CPU qu'il consomme sous la forme d'un nombre décimal (nn.n)

Écrire la commande sysinfo qui affiche la charge du système en additionnant la charge de chaque processus. Cette commande pourra être appelée avec l'option -user *user* limitant alors le calcul aux processus de l'utilisateur *user*.

1) Écrire la fonction analysePs qui analyse les lignes de ps lues sur son entrée standard et calcule la somme des charges CPU

2) Écrire la fonction main qui analyse la ligne de commande et appelle analysePS en lui passant les lignes de ps.

Révisions – Scripts

1) Écrire la fonction analysePs qui analyse les lignes de ps lues sur son entrée standard et calcule la somme des charges CPU

```
function analysePs() {
   local cpu=0
   read
   while read ; do
      local icpu=`echo $REPLY | tr -s " " | cut -d " " -f 3`
      cpu=`echo "scale=2; $cpu + $icpu " | bc`
   done
   echo $cpu
   return 0
}
```

Révisions – Scripts

2) Écrire la fonction main qui analyse la ligne de commande et appelle analysePS en lui passant les lignes de ps.

```
main() {
    filtre='.*'
    if [ $# -gt 0 ] && [ $1 == "-user" ] ; then
        filtre=$2
    fi
    gcpu=`ps aux | grep "$filtre" | analysePs`
    echo "utilisation du cpu à  : $gcpu %"
}

main $*
```