TRAVAUX PRATIQUES 5
Scripts avancés sous Linux

L'objectif de ce TP est de continuer à approfondir la réalisation de programmes scripts abordée lors du précédent TP sous les systèmes d'exploitation de type UNIX pour l'automatisation de traitements.
Une fois votre session ouverte sous un système d'exploitation de type UNIX, ouvrez un terminal pour entrer les commandes décrites dans l'énoncé. Dans la suite, le prompt du terminal sera symbolisé par « `$>` ».

EXERCICE 1

Dans cet exercice, nous allons nous intéresser aux boucles qui sont des structures de contrôle permettant de répéter l'exécution d'une suite de commandes. Parmi ces structures, deux types de boucle sont majoritairement utilisées : la boucle `while` et la boucle `for`.

Boucle `while`

La boucle `while` (signifiant « *tant que* » en anglais) exécute une suite de commandes en répétition qu'une condition réussie, c'est-à-dire a un code de retour vrai (code 0).
La syntaxe de ce type de boucles est la suivante :

```
while condition
do
     commandes
done
```

Les lignes entre les instructions `do` et `done` (appelé « *corps de la boucle* ») seront exécutées tant que `condition` (réalisée avec la commande `test` par exemple) retourne un code de retour vrai (code égale à 0). Après exécution du corps de la boucle condition est à nouveau testée, si elle renvoie faux alors le shell exécute la commande située après `done` dans le script.

Question 1 – Réalisez un script prenant en argument un entier supérieur à 0 et créant autant de fichier qu'indiqué dont les noms sont construit comme suit : f1, f2, …

Boucle infinie

Il est possible de réaliser une répétition infinie d'une suite de commandes à l'aide d'une boucle infinie. Pour cela, il faut utiliser les commandes spéciales `:` et `true` qui renvoie toujours vrai. Ainsi, utilisée avec une boucle `while` cela permet d'exécuter à l'infinie le corps de la boucle.
La syntaxe d'une boucle infinie avec la commande « `:` » est la suivante :

```
while :
do
     commandes
```

```
done
```

La syntaxe d'une boucle infinie avec la commande « `true` » est la suivante :

```
while true
do
     commandes
done
```

Question 2 – Réalisez un script affichant à l'infinie le message suivant sur le terminal : « Ceci ne doit jamais etre reproduit ! ».
Quelle procédure allez-vous suivre pour arrêter l'exécution de ce script ?

Boucle `for`

La boucle `for` (signifiant « *pour* » en anglais) affecte successivement à chaque répétition une variable `var` avec l'une des valeurs compris dans une liste de valeurs `val1 val2 … valn`. Le corps de boucle (lignes entre les instructions `do` et `done`) est exécuté à chaque fois en considérant le contenu de la variable `var`. La liste de valeurs peut être fournie directement dans le script, fournie en argument du script, générée par une commande ou par substitution de caractères spéciaux du shell.
La syntaxe de ce type de boucles avec une liste de valeurs fournies :

```
for var in val1 val2 val3
do
     commandes
done
```

La syntaxe de ce type de boucles avec une liste de valeurs fournie en argument du script :

```
for var in $*
do
     commandes
done
```

La syntaxe avec une liste de valeurs générée par substitution d'une commande :

```
for var in 'commande'
do
     commandes
done
```

La syntaxe avec une liste de valeurs générée par substitution de caractères de génération de noms de fichiers :

```
for var in *.txt
do
     commandes
done
```

Question 3 – Réalisez le script demandé à la Question 1 mais à l'aide d'une boucle `for` au lieu d'une boucle `while`.

Question 4 – Réalisez un script stockant dans un fichier de nom `log_proc.txt` la liste des processus associés à une liste d'utilisateurs indiquée en argument du script lors de son lancement.
Dans le cas où le nombre de processus récupérés avec la commande ps est nul, vous indiquerez par un message qu'il n'y aucun processus pour l'utilisateur considéré.

Question 5 – Réalisez un script prenant en argument lors de son lancement deux noms de fichiers. Le premier argument est un fichier texte dont le contenu de chaque ligne est lu et écrit dans le fichier dont le nom correspond au second argument en ajoutant en début de ligne le numéro de la ligne dans le fichier.

Pour la suite, nous allons utiliser une commande appelée `awk` (du nom de ses auteurs Aho, Weinberger et Kernighan) qui permet de réaliser des traitements sur des fichiers textes.
Voici ci-dessous la syntaxe associée à cette commande :

```
awk  [-F]  '{action-awk}'  [ fic1 … ficn ]
```

Cette commande prend en argument une liste d fichiers à traiter, si aucun fichier n'est fournit alors les données fournit sur l'entrée standard sont considérées.
Cette commande possède son propre langage pour définir les actions à réaliser pour chaque enregistrement issu des fichiers en entrées. Par défaut, un enregistrement correspond à une ligne d'un fichier.
Chaque enregistrement fournit en entrée est automatiquement découpé en champs et un certain nombre de variables internes à awk sont initialisées. Le tableau ci-dessous liste certaines de ces variables :

Nom de la variable	Description de la variable
`$0`	Valeur de l'enregistrement courant
`NF`	Number of Field : nombre de champs de l'enregistrement courant
`$1 $2 … $NF`	$1 contient la valeur du premier champ, $2 la valeur du deuxième champ …, $NF la valeur du dernier champ de l'enregistrement courant
`NR`	Number : indice de l'enregistrement courant (NR vaut 1 pour la première ligne)
`FNR`	File Number : indice de l'enregistrement courant relatif au fichier en cours de traitement
`FILENAME`	Nom du fichier en cours de traitement

Voici ci-dessous un exemple d'utilisation de la commande `awk` :

```
ps -ef | awk '{print "User : ", $1, "Command : ", $8}'
```

`awk` travaille sur le résultat fournit par la commande `ps`. Ici, awk exécute le traitement contenu entre les accolades « `{ }` », et les quotes spécifie au shell de ne pas interpréter ce qui est situé entre elles. La fonction `print` va afficher à l'écran les champs 1 et 8 de chaque ligne fournit par la commande `ps` (ici le propriétaire et le nom de chaque processus).

Il est possible de modifier le séparateur de champs considéré par `awk` à l'aide de l'option `-F` (par exemple `-F'|'` considérera le symbole `|` pour déterminer le découpage des champs de chaque ligne passé en entrée).
Comme indiqué plus précédemment un langage spécifique à cette commande est défini. Ainsi, il est possible d'appliquer une action si une certaine condition est sur l'enregistrement est vérifiée. Voici ci-dessous un tableau des principaux opérateurs de tests :

Opérateur	Description
`<`	Inférieur
`>`	Supérieur
`<=`	Inférieur ou égal
`>=`	Supérieur ou égal
`==`	Test d'égalité
`!=`	Test d'inégalité
`~`	Correspondance avec une expression régulière
`!~`	Non correspondance avec une expression régulière
`!`	Négation
`&&`	ET logique
`||`	OU logique
`(expression)`	Regroupement

L'exemple ci-dessous n'applique l'action d'affichage sur le résultat de la commande `ps` uniquement pour le deuxième enregistrement et si la commande n'est une commande bash :

```
ps -ef | awk 'NR == 2 && $8 != "bash" {print "User : ", $1, "Command : ", $8}'
```

Question 6 – Réalisez un script listant le contenu du répertoire courant et affichant le nom et la taille du fichier le plus volumineux contenu dans le répertoire courant.