TD1 M3206 (BASH1)

Dubreuil/Pouchoulon

1 Que font ces oneliners ou ces scripts BASH ? écrivez ce que vous pensez voir en sortie de la commande ou du script bash

1. quel résultat donne cette ligne ?

   ```
   DATE=$(date); echo $DATE
   ```

2. quel résultat donne cette ligne ?

   ```
   cat /etc/passwd |wc -l
   ```

3. ```
   find /var/log -type f -name *.log -exec ls -alh {} \;
   ```

4. expliquez ce que fait cet alias. Quels sont les autres alias sur votre compte ?

   ```
   alias cds='cd /Users/pouchou/ownCloud/cours\ iut\ 2/cours\ iut/latex/inc/scripts'
   ```

5. Que fait cette ligne ?

   ```
   echo 111-{aa,bb,cc}+{xx,yy,zz}-222
   ```

6. Pourquoi le résultat diffère-t-il entre les deux boucles ? que pourriez vous en retenir comme conseil systématique ?

   ```
   MESSAGE="hello world"
   for i in  "$MESSAGE"; do echo $i; done
   hello world
   for i in  $MESSAGE; do echo $i; done
   hello
   word
   ```

7. Utilisez l'expansion de variables afin de simplifier cette commande :

   ```
   cp dirname-et-basename.sh dirname-et-basename.sh.bak
   ```

8. Déterminez quel est le PID de votre bash via la commande ps -ef. comparer avec la variable $$? Que fait cette commande ?

   ```
   kill -9 $$
   ```

9. ```
   ping -c1 localhost && { echo succes;} || { echo pasglop; }
   ```

2 Explorons $* et $@ : quels résultats donnent les scripts suivants ?

1. Quel résultats pour la commande suivante : ./loopargs1.sh un deux trois quatre rappel du cours : Les variables $* et $@ contiennent la liste des arguments d'un script shell. Lorsqu'elles ne sont pas entourées par des guillemets, elles sont équivalentes. $* ou $@ ensemble des paramètres positionnels, équivalent à $1 $2 ... $n
   "$*" ensemble des paramètres positionnels, équivalent à "$1 $2 ... $n"
   "$@" ensemble des paramètres positionnels, équivalent à "$1" "$2" ... "$n"

```bash
#!/bin/bash

for i in $*
do
    echo $i
done

echo  -e '\n'
for i in $@
do
    echo $i
done

echo  -e '\n'
for i in "$*"
do
    echo $i
done

echo  -e '\n'
for i in "$@"
do
    echo $i
done
```

3 Solutions pour lire un fichier.

1. Découverte de la variable IFS.
   a) Expliquez les résultats de ce script :

```bash
mkdir "repertoire avec espace"{1,2} &>/dev/null && touch "fichier "{1,2} \
& > /dev/null || echo -e "rep et fichiers presents"
for n in $(ls) ; do echo $n ; done
echo -e "-------------------"
echo -e "Avec IFS modifié"
IFS=$'\n'; for n in $(ls) ; do echo $n ; done
```

```
./test-ifs.sh
fichier
1
fichier
2
repertoire
avec
espace1
repertoire
avec
espace2
test-ifs.sh
-------------------
Avec IFS modifie
fichier 1
fichier 2
repertoire avec espace1
repertoire avec espace2
test-ifs.sh
```

   A quoi sert la variable IFS ?

b) Que fait ce script ?

```
#!/bin/bash
_file="${1:-/dev/null}"    # sécurité en cas d'erreur
while IFS= read -r line
do
  echo "$line"
done < "$_file"
```

2. Que fait ce script ?

```
#!/bin/bash
while IFS= read -r line
do
  echo "$line"
done < "/etc/passwd"
```

On rappelle que les champs de /etc/passwd sont séparés par ' :' A quoi sert IFS ?

3. Que fait ce script ?

```
#/bin/bash
IFS=:
while read login mdp uid gid gecos home shell;
    do echo "${gecos:=undef} > login $login (home : $home, shell : $shell)" ;
    done < /etc/passwd
```

A quoi sert IFS ? A quoit sert le undef ?

4. A quoi sert la commande set -x ?