Sujet 02 : Avançons dans l'utilisation de `bash`

Exercice 1 : Saisissons l'utilisateur

Q 1. Écrire un script, nommé `oui-non`, qui demande à l'utilisateur de saisir `oui` ou `non` jusqu'à ce que celui-ci ait effectué une saisie correcte.

oui-non

```
#!/bin/bash

while test "$a" != "oui" -a "$a" != "non"
do
    read -p "Entrer 'oui' ou 'non' : " a
done
```

Q 2. Écrire un script, nommé `voir-variable`, qui, indéfiniment, demande, le nom d'une variable à l'utilisateur et affiche sa valeur. Le script s'arrête uniquement si l'utilisateur entre le mot `FINI`.

voir-variable

```
#!/bin/bash

read -p "Donner le nom d'une variable : " var
while test "$var" != "FINI"
do
    eval "echo \$$var"
    read -p "Donner le nom d'une variable : " var
done
```

Q 3. Écrire un script, nommé `compter-arguments`, qui affiche un à un ses arguments sous la forme :

```
$ compter-arguments toto bonjour titi
L'argument 1 est toto
L'argument 2 est bonjour
L'argument 3 est titi
$
```

compter-arguments

```
#!/bin/bash

n=1

for a in "$@"
do
    echo "L'argument $n est $a"
    n=$(expr $n + 1)
done
```

Exercice 2 : Dessinons un arbre

Q 1. Écrire un script, nommé `arbre`, qui affiche l'arborescence dont la racine est passée en paramètre.

Q 2. Modifier la commande pour ajouter une indentation suivant la profondeur du fichier par rapport à la racine.

arbre

```
#!/bin/bash

# On utilise un appel récursif de la commande avec le second paramètre qui
# permet de faire faire circuler le préfixe d'indentation. S'il est absent
# (lors de l'appel initial) on n'indente pas.

```

```
if test $# -lt 1 ; then
    echo "erreur: mauvais nombre d'argument" 1>&2
    exit 1
elif test ! -d "$1" ; then
    echo "erreur: $1 pas un répertoire" 1>&2
    exit 2
fi

echo "$2$(basename $1)"         # $(...) = `...`

decalage="  $2"                 # le reste de l'affichage doit être indenté

for f in $1/*                   # $(ls $1)
do
    if test -d "$f" ; then
        $0 "$f" "$decalage"     # $0 "$1/$f" "$decalage"
    else
        echo "$decalage$(basename $f)" # echo "$decalage$f"
    fi
done
```

Le script existe aussi en version plus lisible (avec des fonctions) sous le nome `arbre-2`.

Q 3. En utilisant le contenu du répertoire **`/proc`** écrivez un script, nommé **`psarbre`**, qui affiche l'arborescence des processus (leur PID et leur ligne de commande) en cours d'exécution sur le système.

psarbre

```
#!/bin/bash

# On utilise un appel récursif de la commande avec le premier paramètre qui
# est le PID dont il faut afficher la descendance, le second qui permet de
# faire circuler l'indentation et le troisième qui est le numéro de processus
# de la commande initiale (pour éviter les appels infinis dû à la création des
# processus pendant la récursion).
#
# On utilise aussi des fonctions pour faciliter la lecture du code.

# afficher la liste des PID des fils d'un processus
enfants ()
{
    if ! test "$1" = "$exclude" ; then
        grep "PPid:" /proc/[0-9]*/status | grep -v "/proc/$1/" | grep -w "$1" | cut -d / -f 3 | sort -n
    fi
}

# réussi si et seulement si le processus spécifié a des enfants
est_pere ()
{
    cut -d ' ' -f 4 /proc/[0-9]*/stat | grep -qw "$1"
}

# affiche le numero de processus et la ligne de commande d'un processus
affiche ()
{
    echo "$decalage"$1 $(cat /proc/$1/cmdline | tr '\0' ' ')
}

################################################################################

# lors de l'appel initial on fixe les paramètres explicitement
if test $# -eq 0 ; then
    set 1 "" $$
elif test $# -eq 1; then
    set $1 "" $$
fi

pid=$1
decalage="$2"
exclude=$3


```

```
45 affiche $pid
46
47 decalage="  $decalage"          # le reste de l'affichage doit être indenté
48
49 for p in $(enfants $pid)
50 do
51     if est_pere $p ; then
52         $0 $p "$decalage" $exclude
53     else
54         affiche $p
55     fi
56 done
```