TP1 - Rappel sur Bash et scripts

B. Exercices

1 Shell: remise en forme

(*Interdit d'utiliser gedit ou n'importe quel éditeur de texte pour cette partie*)

1. Écrivez votre nom dans un fichier nom.txt via l'entrée standard (cat).

   Par exemple cat "Martin" > nom.txt

2. Écrivez votre prénom grâce à echo dans un fichier `prenom.txt`

   `echo "Jean" > prenom.txt`

3. À l'aide des redirections, concaténez les deux fichiers `nom.txt` et `prenom.txt` dans un fichier `id.txt`

   `cat nom.txt prenom.txt > id.txt`

4. Quelles sont les différences entre les trois commandes suivantes? Expliquez.

   `4.1 wc toto.c`

l'interpréteur de commandes lit l'argument et le passe en entrée à wc.

   `4.2 wc < toto.c`

le fichier toto.c est donné directement en entrée à la commande wc. Il y a un passage en moins par rapport à 4.1

   `4.3 cat toto.c | wc`

le fichier toto.c est lit comme argument de la commande cat, la sortie de cat est donnée comme entrée de wc par le moyen d'un tube.

La perception de l'utilisateur c'est que dans les 3 cas le résultat c'est le même.

5. Écrivez une commande renvoyant le nombre d'utilisateurs connectés à votre machine (who, wc).

who | wc -l

6. Écrivez une commande pour afficher la liste de tous les fichiers dans votre répertoire ayant:

   6.1 droits de lecture et d'exécution pour le propriétaire, group et tous les utilisateurs

ls -l | grep ".r.xr.xr.x"

   6.2 droits d'écriture pour le propriétaire seul

ls -l | grep ".r..-..-.."

7. Changez les droits de votre fichier id.txt pour qu'il soit modifiable par vous uniquement et pour qu'uniquement les membres de votre groupe puissent le lire.

chmod 640 id.txt

2 Programmation Bash

0. Téléchargez le fichier http://lipn.fr/~buscaldi/systemeTP1.tar.gz et décompressez -le dans un répertoire de votre choix

1. Exécuter le fichier `sc1.sh` donné qui comprend un exemple de « script de bash ». Indiquez les variables utilisées dans ce script. Quelles commandes du script ont défini ces variables, leur ont affecté des valeurs et les ont utilisé ?
Modifier ce script pour répéter la saisie afin d'éviter la chaîne vide à l'entrée. Tester le script modifié.

Avec while:

```
#!/bin/bash

echo -e  \\n  \\t On déclare la variable locale v1 \\n
v1=quelque_chose
echo -e  \\t  La valeur de v1 est :  $v1 \\n

echo -en  \\t  Entrez la nouvelle valeur de v1 :
read v1
while  [  -z $v1 ] ; do
        echo   La valeur de v1 est vide
        echo -e  Comme preuve je vous l\'affiche : $v1
        echo -en Entrez a nouveau une valeur pour v1 :
        read v1
done
echo   "  Nouvelle valeur de v1 : $v1 "
```

Il peuvent commencer par modifier le if pour demander une nouvelle saisie, mais après ils devraient voir que le while permet d'éviter le problème.

2. Comme les programmes en C, les scripts acceptent des arguments sur la ligne de commande. Ce mécanisme permet de passer des paramètres aux commandes se trouvant dans le script. Le shell considère ses arguments comme les variables 1, 2, .. et substitue leur valeur aux chaînes $1, $2, … Avec $0 on accède au nom du script et avec $# on obtient le nombre des arguments.
Exécutez le fichier « `sc2.sh` » avec des arguments différents. Expliquer ce que fait ce script. Modifiez sc2.sh avec la commande *shift* pour décaler de 2 la liste des arguments. Vérifiez le comportement du script après cette modification.

Le script affiche tous les arguments fournis par l'utilisateur ou un message d'erreur s'il n y a pas d'arguments..

```
#!/bin/bash

echo  " "          # pour sauter une ligne
```

```
if [ $# == 0 ]  ; then
   echo -e "   Ce script qui s'appelle $0 n'a pas d'argument ! \\n"
else
   shift 2
   echo -e  \\t Le premier argument de $0 :  $1 \\n
   if [ $#  >  1 ] ; then
         echo -e \\t Les arguments suivants du script $0 :  \( 1 arg par ligne \)
         for i in  $*  ; do
         echo  -e \\t \\t $i
         done
   fi
fi
```

après la modification, la liste des arguments est modifiée en enlevant le 2 premiers arguments. Si la liste est de taille < 2, le shift n'a pas d'effet.

2.1. Ecrire un fichier shell qui prend en paramètre 2 entiers et affiche la somme. S'il n'y a pas deux paramètres, il faut afficher un message d'erreur.

```
#!/bin/bash

if [ $# -ne 2 ]  ; then
   echo -e "   Ce script qui s'appelle $0 exactement 2 arguments ! \\n"
else
   v1=$1
   v2=$2
   somme=$((v1+v2))
   echo -e " La somme de $1 et $2 vaut $somme "
fi
```

ça peut être intéressant pour les étudiants de vérifier comment le résultat change si au lieu de nombres on donne en paramètre des chaînes de caractères

2.2. Ecrire un script qui prend en paramètre trois entiers et qui affiche le plus grand. On affichera un message d'erreur s'il n'y pas pas trois arguments passés sur la ligne de commande.

```
#!/bin/bash

if [ $# -ne 3 ]  ; then
   echo -e "   Ce script qui s'appelle $0 exactement 3 arguments ! \\n"
else
   v1=$1
   v2=$2
   v3=$3
   if [ $v1 -ge $v2 ]; then
         max=$v1;
         if [ $v3 -ge $v1 ]; then
```

```
                max=$v3;
        fi
  else
        max=$v2;
        if [ $v3 -ge $v2 ]; then
                max=$v3;
        fi
  fi
  echo -e " Le max entre $1, $2 et $3 vaut $max "
fi
```

3. En utilisant la structure de contrôle « if … then … else …. » écrire un script bash qui prend en paramètre un nom de fichier et affiche
- un message si le fichier n'existe pas sous le répertoire de travail,
- un avertissement si le fichier est un répertoire, et
- finalement si le fichier est exécutable ou non.

Tester le script écrit en l'exécutant avec plusieurs fichiers de types différentes..

une solution un peu pédestre:

```
#!/bin/bash

if [ $# -ne 1 ]  ; then
  echo -e "   Ce script qui s'appelle $0 prend exactement 1 argument! \\n"
else
  if [ -e $1 ]; then
        echo " Le fichier $1 existe "
        if [ -d $1 ]; then
                echo " et est un repertoire"
        fi
        if [ -x $1 ]; then
                echo " et il est exécutable "
        fi
  else
        echo " Le fichier $1 n'existe pas "
  fi
fi
```

3.1 Écrire un script shell qui affichera 5,4,3,2,1 en utilisant une boucle while.

```
#!/bin/bash

val=5
while [ $val -gt 0 ]; do
        echo -ne $val","
        val=$((val-1))
done
echo ""
```

alternativement, stocker les chiffres dans une chaîne que sera affichée à la fin

4. Taper et sauver le script suivant. Indiquer
- comment on saisie la valeur d'une variable dans un script,
- quel est la signification du cas *),
- le rôle de « ; ; » .
Expliquer le rôle de la variable « essai » en précisant son type.
Programmer les choix 3,4,5.

```
#!/bin/bash
# Exemple de script
clear
essai=1 ;
while [ 0 ] ; do

echo -e " \\n\\n\\n Menu - Essai : $essai "
echo " Afficher repertoire courant 1 "
echo " Lister les fichiers 2 "
echo " Informations sur un fichier 3 "
echo " Changement de repertoire 4 "
echo " n premieres lignes d'un fichier 5 "
echo " Sortie                0 "
echo -n " Choix : "
read choix ;
echo " "
case $choix in
     1) pwd ;;
     2) ls ;;
     3) ;;      # a remplir ...
     4) ;; # a remplir ...
     5) ;; # a remplir ...
     0) exit ;;
     *) echo Choix non propose ;;
esac
essai=$(( essai + 1 )) ;
done
```

ce n'est pas très compliqué, le ;; correspond au "break" du C, et on ne l'utilise que pour case. *) c'est le "default"

5. On veut écrire un script qui copie les fichiers ordinaires d'un répertoire source vers un répertoire destination. Un fichier du répertoire source n'est copié que s'il n'existe pas de fichier de même nom dans le répertoire de destination. Le script demande d'abord le nom du répertoire source et n'accepte qu'un nom valide (non vide et correspondant à un répertoire existant). Ensuite il demande à l'utilisateur d'entrer le nom du répertoire destination qui doit être également valide. Pour effectuer la copie, il faut vérifier le droit d'écriture sur le répertoire destination. Si ce droit n'existe pas , le script l'ajoute pour le retirer à la fin. Enfin il copie les fichiers.

```
#!/bin/bash

echo "Saisir le nom du repertoire source:"
read src

if [ -d $src ]; then
    echo "Saisir le nom du repertoire destination:"
    read dest
    if [ -d $dest ]; then
        if [ -x $dest ]; then
            chmod u+w $dest
        fi
        filelist=`ls $dest`
        for file in `ls $src` ; do
            if [ -f $file ]; then
                flag=0
                for f in $filelist ; do
                    if [ $file == $f ]; then
                        echo "un fichier $file existe déjà dans $dest"
                        flag=1
                    fi
                done
                if [ $((flag)) -eq 0 ]; then
                    cp $src/$file $dest
                fi
            fi
        done
        chmod u-w $dest
    else echo "Erreur: le nom ne correspond pas à un repertoire"
    fi
else echo "Erreur: le nom ne correspond pas à un repertoire"
fi
```

5.1 Modifier ce script pour qu'il copie chaque fichier du répertoire source vers la destination même s'il y existe un fichier de même nom mais plus ancien. Si la destination contient une version plus récente, la copie n'aura pas lieu. Tester le script modifié.

```
dans le 2ème for:
if [ $file == $f ]; then
        echo "un fichier $file existe déjà dans $dest"
         if [ $file -ot $f1 ]; then
                echo "et il est plus recent"
                 flag=1
         fi
 fi
```

6. Écrire un script qui concatène puis trie (commande **sort**) deux fichiers `file1` et `file2` dans un nouveau fichier `file3` et qui affiche le nombre total de lignes. Les noms des trois fichiers doivent être passés en paramètre.

```
cat $1 $2 | sort > $3
nlines=`wc -l $3 | cut -f1 -d' '`
echo $nlines
```

7. Dans le but de voir le mécanisme d'exécution des commandes et des scripts, on exécute la commande ps sous le shell et dans deux scripts similaires. Les scripts commencent en affichant leur nom. Comme seconde commande ils affichent la valeur de la variable SHLVL. Ensuite le premier script exécute la commande ps tandis que le seconde exécute le premier script. Tous les deux se terminent avec un dernier affichage qui indique le nom et la fin du script en cours.

A. Ecrire ces deux scripts.
B. Exécuter la commande ps et afficher la valeur de la variable SHLVL dans un shell

C. Exécuter sous le même shell le premier, puis le second script.

Qu'est ce que l'on peut déduire des affichages obtenus ?

premier script (sp71.sh)

```
#!/bin/bash
echo "je suis le script $0 et mon SHLVL est $SHLVL"
ps
echo "$0 terminé"
```

deuxième script (sp72.sh)

```
#!/bin/bash
echo "je suis le script $0 et mon SHLVL est $SHLVL"
bash sp71.sh
echo "$0 terminé"
```

SHLVL dans un shell utilisateur donne 1. L'exécution des scripts montre que la variable SHLVL augmente à chaque fois qu'on exécute un script et/ou on ouvre un nouveau shell.

8. On veut écrire un script qui constitue une liste des fichiers ordinaires du répertoire courant, qui répondent à certains critères. Au début du script on définit la liste comme une variable vide. Le script constitue la liste progressivement, en ajoutant à cette variable le nom du fichier que l'on est en train de tester si ce dernier est à sélectionner. Quand on a terminé le test sur toutes les références du répertoire courant, on affiche la liste à l'écran. Si un argument est donné sur la ligne de commande, il est considéré comme le nom du fichier dans lequel le script doit sauvegarder la liste. Contrairement à l'affichage qui a lieu dans tous les cas, la sauvegarde est optionnelle.
Ecrire le script décrit ci-dessus pour sélectionner les fichiers sur lesquels le propriétaire a le droit d'écriture. Tester le script dans un répertoire qui contient au moins 3 fichiers à sélectionner, un sous-répertoire et plusieurs fichiers à exclure. N'oubliez pas de tester l'option.

```
#!/bin/bash

filelist=""

if [ $# -gt 0 ]; then outfile=$1; fi;

for file in `ls`; do
      if [ -f $file ]; then
            if [ -w $file ] ; then
                  filelist=$filelist$file" "
            fi
      fi
done

echo $filelist

if [ $# -gt 0 ]; then echo $filelist > $outfile; fi;
```

il y a des solutions meilleures...

8.1. Ecrire un script qui prend en paramètre un entier et affiche l'entier en ordre inverse (123 -> 321).

solution maline qui nécessite un peu de intro système sur les systèmes de numération positionnels et avoir compris comment l'operateur $(( )) fonctionne

```
#!/bin/bash

nb=$(($1))

chaine=""
while [ $nb -gt 0 ]; do
      chiffre=$(($nb%10))
      nb=$((nb/10))
      chaine=$chaine$chiffre
done

echo $chaine
```

8.2. Ecrire un script qui prend en paramètre un entier et affiche la somme des digits qui le compose (123 -> 1+2+3 = 6).

s'ils ont compris avec l'exercice antérieur, ça devrait être une modification triviale:

```
#!/bin/bash

nb=$(($1))

sum=0
while [ $nb -gt 0 ]; do
```

```
    chiffre=$(($nb%10))
    nb=$((nb/10))
    sum=$(($sum+$chiffre))
done

echo $sum
```

9. Expliquer ce qui est fait par la fonction checkpid( ) suivante:

```
checkpid() {
        local i

        for i in $* ; do
                [ -d "/proc/$i" ] || return 1
        done
        return 0
}
```

Une fonction définie peut être utilisée comme une commande.
Utiliser la fonction checkpid( ) pour observer au moins une fois chaque valeur qu'elle peut retourner.

Les fonctions ne sont pas trop utilisées, ne pas forcer trop sur ces questions, c'est surtout pour les introduire à l'idée de processus et où on trouve certaines informations sur la machine

10. (facultatif) Téléchargez le fichier http://lipn.fr/~buscaldi/space.sh . Étudiez le code de ce script. Listez les fonctions définies dans ce script. Modifiez les touches nécessaires pour déplacer le vaisseau et tirer (au lieu de a,l,f utiliser des autres touches de votre choix).