Introduction aux systèmes d'exploitation (IS1)

TP 9 – Agenda
2. Fonctions avancées
CORRECTION

1 Structures de contrôle

1.1 Boucles `for`

Exercice 1 – *Échauffement.*

1. Pour afficher « Bonjour ! » indéfiniment, on peut utiliser une boucle du type

   `for (( expr1 ; expr2 ; expr3 )) do commandes ; done`

   en utilisant une condition *expr2* toujours réalisée. Par exemple

   ```
   for (( i=1 ; 0<1 ; i++ )) do echo Bonjour \! ; done
   ```

   Ce n'est pas possible avec l'autre type de boucle `for` puisqu'il faudrait avoir une liste infinie...
2. Pour afficher le nombre de fichiers et sous-répertoires présents dans le répertoire courant, on pourrait évidemment utiliser `ls | wc -l` (`ls` affiche tous les fichiers et sous-répertoires, et `wc -l` compte les lignes du résultat). Le but est ici de le faire avec une boucle. Pour cela, on utilise les expansions du shell : souvenez-vous que `*` fait la liste de tous les fichiers et sous-répertoires contenus dans le répertoire de travail (par exemple, `echo *` affiche les noms de tous les fichiers et sous-répertoires du répertoire courant). On peut donc écrire :

   ```
   cpt=0 ; for fic in * ; do cpt=$(($cpt+1)) ; done ; echo $cpt
   ```

   Le compteur `cpt` compte le nombre de fichiers et sous-répertoires dans la liste donnée par `*` qui est parcourue par la variable `fic`.

   Sans beaucoup plus d'efforts, on pourrait compter uniquement le nombre de sous-répertoires (sans compter les fichiers) : il suffit de tester (avec la commande `test -d`) si `fic` est un répertoire avant d'incrémenter le compteur.

   ```
   cpt=0
   for fic in *
   do
       if test -d $fic ; then cpt=$(($cpt+1)) ; fi
   done
   echo $cpt
   ```

   Remarquez que je peux passer à la ligne au lieu de mettre des points-virgules : c'est parfois plus lisible.
3. Pour afficher la valeur de 2^n, on écrit une boucle `for` dans laquelle on multiplie par 2 à chaque étape un accumulateur `res` (qu'on a initialisé à 1). À la fin, on affiche le résultat :


   ```
   echo -n "entrer un entier " ; read n
   res=1 ; for (( i=1 ; i<=n ; i++ )) do res=$(($res*2)) ; done ; echo $res
   ```

   Si on rentre un entier trop grand (précisément $n > 62$), on dépasse la taille de l'entier maximal (ils sont stockés sur 64 bits, dont un pour le signe). On revient donc dans les négatifs...

1.2 Boucles `while` et `until`

Exercice 2 – *Échauffement.*

1. Pour afficher le message « Bonjour ! » indéfiniment, je peux écrire une boucle `while` dont la condition d'arrêt est toujours vraie, ou une boucle `until` dont la condition d'arrêt est toujours fausse :

   ```
   while true ; do echo Bonjour \! ; done
   until false ; do echo Bonjour \! ; done
   ```

   Vous pouvez remplacer `true` par n'importe quelle commande dont la valeur de retour est toujours 0 (par exemple `(( 0<1 ))`) et `false` par n'importe quelle commande dont la valeur de retour est toujours 1 (par exemple `(( 0>1 ))`)
2. Pour afficher toutes les puissances de 2 inférieures à un nombre n, j'utilise à nouveau un accumulateur `res`. Je l'initialise à 1, et à chaque étape, je l'affiche et je le multiplie par 2, jusqu'à ce qu'il dépasse n :

   ```
   echo -n "entrer un entier " ; read n
   res=1 ; while (( $res<$n )) ; do echo $res ; res=$(($res*2)) ; done
   ```

2 Fonctions avancées de l'agenda

Exercice 3 – *Lire une date (bis)*

Je modifie mon exécutable `lireDate.sh` avec une boucle `while` dont la condition d'arrêt est la validité de la date :

LIREDATE.SH

```
#!/usr/local/bin/bash
#argument : un message d'invitation ($@)
#read : une date au format jj mm aaaa
#sortie : une date valide au format aaaammjj

j=00 ; m=00 ; a=0000
until (verifierDate.sh $j $m $a) ; do echo $@ >&2 ; read j m a ; done
echo $a$m$j
```

L'exécutable `lireHeure.sh` est modifié de manière analogue.

Exercice 4 – *Afficher les événements chevauchant une plage horaire*

1. Je note (a, b) la plage horaire située entre l'instant a et l'instant b. Je considère deux plages horaires (a, b) et (c, d). Elles ne se rencontrent pas ssi l'une d'entre elles est complètement avant l'autre, c'est-à-dire si b est avant c ou si d est avant a (voir Fig. 1). Par conséquent, elles se chevauchent ssi b est après c et d est après a (voir Fig. 2).

FIG. 1 – Les plages horaires (a, b) et (c, d) ne se rencontrent pas.

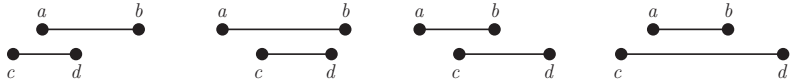

FIG. 2 – Les plages horaires (a, b) et (c, d) se chevauchent.

On écrit donc le programme suivant :

PLAGEHORAIRE.SH

```
#!/usr/local/bin/bash
#arguments : une plage horaire ($1) au format dd=hd=df=hf et un évènement ($2)
#valeur de retour : 0 si l'évènement chevauche la plage horaire, 1 sinon

test ${1:14} > ${2:0:13} && test ${2:${#2}-13} > ${1:0:13}
```

Trois remarques sur le script précédent :

(i) comme d'habitude, j'utilise les outils de manipulation de chaînes de caractères pour attraper ce qui m'intéresse : `${mot:n:m}` renvoie les caractères $n+1$ à $n+m$ de la chaîne contenue dans la variable `mot`.

(ii) écrire les dates à l'envers (*i.e.*, au format `aaaammjj=hhmm`) fait correspondre l'ordre alphabétique et l'ordre chronologique : si `D1` et `D2` sont deux dates au format `aaaammjj=hhmm`, alors `test $D1 < $D2` teste si `D1` est antérieur à `D2`.

(iii) contrairement à ce que prétend le manuel de `test`, il se trouve que sur certains shells, la commande `test string1 < string2` ne fonctionne pas : il confond < avec une redirection. Si c'est le cas, il faut remplacer `test string1 < string2` par `[[ string1 < string2 ]]` pour savoir si `string1` est lexicographiquement inférieur à `string2` (sans oublier les espaces après `[[`, avant `]]` et autour de <).

2. Dans l'exécutable suivant,

(i) je commence par demander à l'utilisateur une plage horaire `plage` : j'utilise ici les exécutables `lireDate.sh` et `lireHeure.sh`.

(ii) ensuite, je parcours tous les évènements de l'agenda, et je mets dans une liste `liste` tous ceux qui chevauchent la plage horaire `plage`. Je me sers pour cela de l'exécutable `plageHoraire.sh` et d'une boucle `for` (j'utilise la commande `cat ~/.agenda | wc -l` pour connaître le nombre de lignes de l'agenda, et la commande `sed -n "$i"p ~/.agenda` pour récupérer la $i$ème ligne de l'agenda).

(iii) finalement, j'utilise l'exécutable `afficherEvenements.sh` pour faire afficher la liste.

AFFICHERPLAGEHORAIRE.SH

```
#!/usr/local/bin/bash
#lit une plage horaire avec lireDate.sh et lireHeure.sh
#sortie : les évènements qui chevauchent cette plage horaire

dd=$(lireDate.sh "Date de début")
hd=$(lireHeure.sh "Heure de début")
df=$(lireDate.sh "Date de fin")
hf=$(lireHeure.sh "Heure de fin")
if test $dd$hd > $df$hf ; then echo "erreur" >&2 ;  exit 1 ; fi
plage=$dd=$hd=$df=$hf
liste=""
n=$(cat ~/.agenda | wc -l)
for (( i=1 ; i<=n ; i++ ))
do
    evenement=$(sed -n "$i"p ~/.agenda)
    if plageHoraire.sh "$plage" "$evenement" ; then liste="$liste $evenement" ; fi
done
afficherEvenements.sh "$liste"
```

3. Dans mon fichier `.agenda`, il y a très peu d'évènements qui commencent un jour et finissent un autre jour (*i.e.*, d'évènements qui durent sur plusieurs jours). Par conséquent, si je me donne pour plage horaire un jour complet j,

(i) pour tous les évènements qui commencent et finissent le même jour j', je dois tester si $j = j'$,

(ii) et pour tous les évènements qui commencent un jour j' et finissent un jour j'', je dois tester si $j' \leq j \leq j''$.

Voici donc ce que ça donne en shell :

AFFICHERJOUR.SH

```
#!/usr/local/bin/bash
#lit une date lireDate.sh
#sortie : les évènements qui chevauchent cette plage horaire

jour=$(lireDate.sh "Date")
liste=$(grep "^$jour=[^=]*=[^=]*=[^=]*=$jour=[^=]*$" ~/.agenda)
grep -v "^\([0-9]\{8\}\)=[^=]*=[^=]*=[^=]*=\1=[^=]*$" ~/.agenda > pr
n=$(cat pr | wc -l)
for (( i=1 ; i<=n ; i++ ))
do
    evenement=$(sed -n "$i"p pr)
    dd=$(echo $evenement | cut -d= -f1) ; df=$(echo $evenement | cut -d= -f5)
    if test $dd <= $jour && test $jour <= $df ; then liste="$liste $evenement" ; fi
done
rm -f pr
afficherEvenements.sh "$liste"
```

Encore une fois, on utilise beaucoup les commandes `grep` et `sed` :

(i) `grep "^$jour=[^=]*=[^=]*=[^=]*=$jour=[^=]*$" ~/.agenda` attrape dans l'agenda tous les évènements qui commencent et finissent le jour `jour` ;

(ii) `grep -v "^\([0-9]\{8\}\)=[^=]*=[^=]*=[^=]*=\1=[^=]*$" ~/.agenda > pr` met dans un fichier provisoire `pr` tous les évènements de l'agenda qui durent sur plusieurs jours ;

(iii) enfin, `sed -n "$i"p pr` récupère la `i`ème ligne du fichier `pr`.

Quand il y a peu d'évènements qui durent sur plusieurs jours, ce nouvel exécutable est nettement plus rapide que le premier. Ceci vient du fait que les commandes `sed` et `grep`, même si elles ont le même effet que des boucles (on pourrait reprogrammer certaines de leurs fonctionnalités) sont bien plus rapides que des boucles en shell.

4. Je vais avoir besoin d'un exécutable `inverse.sh` qui transforme un fichier dont les lignes sont sous la forme `a=b=c=d=e=f` en un fichier dont les lignes sont sous la forme `e=f=c=d=a=b`. J'utilise pour cela les commandes `cut` et `paste`

INVERSE.SH

```
#!/usr/local/bin/bash
#argument : un nom de fichier dont les lignes sont au format a=b=c=d=e=f
#sortie : les lignes du fichier au format e=f=c=d=a=b

cut -d= -f1,2 $1 > pr1
cut -d= -f3,4 $1 > pr2
cut -d= -f5,6 $1 > pr3
paste -d= pr3 pr2 pr1
rm -f pr1 pr2 pr3
```

Les lignes du fichier `.agenda` étant triées par ordre chronologique, il n'est pas utile de parcourir tout le fichier `.agenda` : quand la date de début d'un évènement a dépassé la date de fin de la plage horaire, ce n'est plus la peine d'aller voir plus loin.

On peut donc écrire un exécutable qui :

(i) commence par demander à l'utilisateur une plage horaire `plage` (en utilisant les exécutables `lireDate.sh` et `lireHeure.sh`).

(ii) crée un fichier provisoire `pr` qui contient tous les évènements dont la date de début est antérieure à la date de fin de `plage`. Pour cela, je parcours le fichier `.agenda` et je m'arrête dès que j'ai dépassé la date de fin de `plage` (c'est valide puisque le fichier `.agenda` est trié par ordre chronologique). La commande `break` permet de sortir de la boucle.

(iii) sélectionne dans le fichier `pr` tous les évènements dont la date de fin est postérieure à la date de début de `plage`. Pour cela, je commence par transformer les lignes de `pr` pour les mettre au format `df=hf=nom=desc=dd=hd` (*i.e.*, j'inverse date de début et date de fin) avec le sous-exécutable `inverse.sh`. Ensuite, je trie le résultat avec la commande `sort`. Puis je le parcours à l'envers (de bas en haut) et je m'arrête dès que j'ai dépassé la date de début de `plage`. Il ne me reste alors plus qu'à réinverser toutes mes lignes sélectionnées avec `inverse.sh`.

(iv) finalement, j'ai gardé tous les évènements dont la date de début est antérieure à la date de fin de `plage` et la date de fin est postérieure à la date de début de `plage`. J'obtiens donc bien les évènements qui chevauchent `plage` (voir la question 1).

Voici ce que ça donne en shell :

AFFICHERPLAGEHORAIRERAPIDE.SH

```
#!/usr/local/bin/bash
#lit une plage horaire avec lireDate.sh et lireHeure.sh
#sortie : les évènements qui chevauchent cette plage horaire

dd=$(lireDate.sh "Date de début")
hd=$(lireHeure.sh "Heure de début")
df=$(lireDate.sh "Date de fin")
hf=$(lireHeure.sh "Heure de fin")
if test $dd$hd > $df$hf ; then echo "erreur" >&2 ;  exit 1 ; fi

n=$(cat ~/.agenda | wc -l)
for (( i=1 ; i<=n ; i++ ))
do
    evenement=$(sed -n "$i"p ~/.agenda)
    if test ${evenement:0:13} > "$df=$hf"; then break ; fi
done
head -n"$(($i-1))" ~/.agenda > pr1

inverse.sh pr1 | sort > pr2
m=$(cat pr2 | wc -l)
for (( i=$m ; i>=1 ; i-- ))
do
    evenement=$(sed -n "$i"p pr2)
    if test ${evenement:0:13} < "$dd=$hd"; then break ; fi
done
tail -n"$(($n-$i))" pr2 > pr3

afficherEvenements.sh $(inverse.sh pr3)
rm -f pr1 pr2 pr3
```

Exercice 5 – *Ajouter un événement périodique*

1. La commande `date2jours.sh` prend comme arguments une date au format `aaaammjj` et renvoie le nombre de jours écoulés depuis le 1/1/0. La commande `jours2date.sh` fait l'opération inverse. Par exemple `date2jours.sh 20081216` affiche `733757` et `jours2date.sh 733757` affiche `20081216`.

2. Avec ces deux commandes, l'exécutable `ajouterDate.sh` s'écrit facilement :

AJOUTERDATE.SH

```
#!/usr/local/bin/bash
#arguments : une date d ($1) au format aaaammjj et un nombre de jours j ($2)
#sortie : la date qu'il sera j jours après d, au format aaaammjj

echo $(jours2date.sh $(($(date2jours.sh $1)+$2)))
```

3. L'exécutable `ajouterPeriodique.sh` s'écrit alors simplement avec une boucle :

AJOUTERPERIODIQUE.SH

```
#!/usr/local/bin/bash
#demande dates et heures de début et de fin d'un évènement
#demande nom et description de l'évènement
#vérifie la validité des dates et heures
#demande une période per et un nombre d'occurences nmbr
#ajoute n lignes  dans le fichier .agenda correspondant à l'évènement périodique
#retrie le fichier .agenda

dd=$(lireDate.sh "date de debut (jj/mm/aaaa)")
hd=$(lireHeure.sh "heure de debut (hh:mm)")
df=$(lireDate.sh "date de fin (jj/mm/aaaa)")
hf=$(lireHeure.sh "heure de fin (hh:mm)")
echo "Nom" ; read nom
echo "Description (optionnelle)" ; read desc
echo "Periode entre deux evenements" ; read per
echo "Nombre d'occurences" ; read nmbr

if ( test $dd$hd < $df$hf && expr "$per" : "[0-9]*$" > /dev/null)
then
    for (( i=0 ; i<nmbr ; i++ ))
    do
        echo $(ajouterDate.sh $dd $(($i*$per)))=$hd=$nom=$desc=$(ajouterDate.sh $df
$(($i*$per)))=$hf >> ~/.agenda
    done
    sort ~/.agenda -o ~/.agenda
else echo "erreur" >&2 ;  exit 1
fi
```

4/5. Questions 4 et 5 laissées au lecteur.

3 Interfaces

Exercice 6 – *Interface terminal*

Pour revenir toujours au menu de l'interface après l'exécution du programme correspondant à l'un des 5 premiers choix, j'utilise une boucle `while` sans fin :

AGENDA.SH

```
#!/usr/local/bin/bash
#affiche le menu et effectue la tache demandée par l'utilisateur

while(true)
do
    efface.sh
    echo "1. Rechercher un évènement"
    echo "2. Évènements sur une plage horaire"
    echo "3. Ajouter un évènement ponctuel"
```



```
    echo "4. Ajouter un évènement périodique"
    echo "5. Supprimer un évènement"
    echo "6. Quitter"
    echo "Entrer le numero de la commande" ; read c
    case $c in
        1) efface.sh ; chercher.sh ; read ;;
        2) efface.sh ; afficherPlageHoraire.sh ; read ;;
        3) efface.sh ; ajouter.sh ; read ;;
        4) efface.sh ; ajouterPeriodique.sh ; read ;;
        5) efface.sh ; supprimer.sh ; read ;;
        6) clear; exit 0 ;;
        *) echo \007 ;;
    esac
done
```

Dans le `script` précédent,

1. la commande `read` sert à faire attendre le terminal après l'exécution d'un des cinq premiers choix du menu, pour que l'utilisateur ait le temps de voir le résultat ;
2. la commande `echo \007` emet un bib sonore ;
3. la commande `efface.sh` sert juste à effacer le terminal et à afficher date et heure :

EFFACE.SH

```
#!/usr/local/bin/bash
#efface le terminal et affiche date et heure

clear
echo "AGENDA IS1"
date "+date : %d %h %y%nheure : %H:%M"
echo
```

Exercice 7 – *Format* `iCalendar`

Pour transformer le fichier `.agenda` au format `icalendar`, il y a plusieurs méthodes (on a déjà fait ce genre de choses dans l'exercice 14 du TP8). Ici, je commence par écrire un exécutable `ligne2ics.sh` qui transforme une ligne de l'agenda en un évènement au format `icalendar`. J'utilise pour cela la commande `cut` :

LIGNE2ICS.SH

```
#!/usr/local/bin/bash
#argument : une ligne de l'agenda
#sortie : évènement au format icalendar

echo "BEGIN:VEVENT"
echo "SUMMARY:$(echo $1 | cut -d= -f3)"
echo "DTSTART:$(echo $1 | cut -d= -f1)T$(echo $1 | cut -d= -f2)00"
echo "DTEND:$(echo $1 | cut -d= -f5)T$(echo $1 | cut -d= -f6)00"
desc=$(echo $1 | cut -d= -f4)
if test $desc ; then echo "DESCRIPTION:$desc" ; fi
echo "END:EVENT"
```

Ensuite, j'utilise ce petit exécutable dans une boucle de l'exécutable `agenda2ics.sh` :

AGENDA2ICS.SH

```
#!/usr/local/bin/bash
#transformation du fichier ~/.agenda au format ics.
#Le résultat est un fichier ~/agendaIS1.ics

echo "BEGIN:VCALENDAR" > ~/agenda.ics
echo "PRODID    :-//handcal/NONSGML" >> ~/agenda.ics
echo "V1.1//EN" >> ~/agenda.ics
echo "VERSION:2.0" >> ~/agenda.ics
n=$(cat ~/.agenda | wc -l)
for (( i=1 ; i<=n ; i++ ))
do
    ligne2ics.sh $(sed -n "$i"p ~/.agenda) >> ~/agenda.ics
done
echo "END:VCALANDAR" >> ~/agenda.ics
```

Enfin, on peut visualiser le résultat avec le logiciel `Sunbird` (en tapant `sunbird` dans un terminal, puis en ouvrant le fichier `agenda.ics` depuis `Sunbird`). Le résultat est visuellement intéressant (voir par exemple Fig. 3).

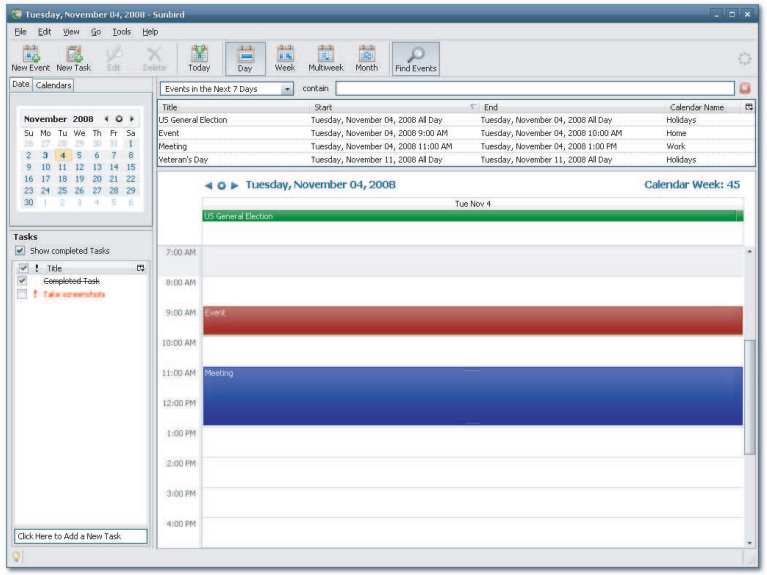

FIG. 3 – Un exemple d'affichage avec `Sunbird`.

4 Pour aller plus loin

Exercice 8 – *Agenda à l'ouverture d'un nouveau terminal (bis)*

Comme d'habitude, pour effectuer une commande à l'ouverture de chaque terminal, il faut mettre cette commande dans le fichier `.bashrc`. Je rajoute donc les lignes suivantes :

```
D=$(date +%Y%m%d)
H=$(date +%H%M)
n=$(grep -c "^$D" ~/.agenda)
for (( i=1 ; i<=n ; i++ ))
do
    h=$(grep "^$D" ~/.agenda | sed -n "$i"p | cut -d= -f2)
    if test $H < $h ; then breack ; fi
done

if (( $i==$n+1 )) ; then echo "plus d\'évènement dans la journée" ;
else afficherEvenement.sh $(grep "^$D" ~/.agenda | sed -n "$i"p) ; fi
```

Quelques explications :

(i) La commande `date`, utilisée avec `+%Y%m%d` (voir le manuel) permet d'avoir la date au format `aaaammjj`. De même, la commande `date +%H%M` permet d'avoir l'heure au format `hhmm`.

(ii) Un `grep` sur la date permet d'avoir tous les évènements de la journée (l'option `-c` permet de les compter).

(iii) Ensuite, on parcourt ces évènements jusqu'à ce qu'on en trouve un dont l'heure dépasse l'heure donnée par `date +%H%M`. La commande `break` permet alors de sortir de la boucle.

(iv) Pour terminer, il y a deux possibilités : si on n'a pas trouvé d'évènement dans la journée dont l'heure dépasse l'heure donnée par `date +%H%M`, on est sorti de la boucle et `i` vaut `n+1`. Sinon, le prochain évènement de la journée est le `i`ème.

Exercice 9 – *Jours*

1. Pour savoir quel jour de la semaine était une certaine date, il suffit de transformer cette date en un nombre de jours `n` avec la commande `date2jours.sh`, et de faire un `case` sur le reste de `n` modulo 7 :

LMMJVSD.SH

```
#!/usr/local/bin/bash
#lit une date lireDate.sh
#sortie : le jour de la semaine à cette date

d=$(lireDate.sh "entrer une date")
case $(($(date2jours.sh $d)%7)) in
    2) echo lundi ;;
    3) echo mardi ;;
    4) echo mercredi ;;
    5) echo jeudi ;;
    6) echo vendredi ;;
    0) echo samedi ;;
    1) echo dimanche ;;
esac
```

2. Pour afficher un calendrier comme la commande `cal`, on écrit l'exécutable suivant :

CALENDRIER.SH

```
#!/usr/local/bin/bash
#read : lit un mois et une année
#sortie : un calendrier

j=00 ; m=00 ; a=0000
until (verifierDate.sh $j $m $a) ; do echo "mois et année" ; read m a ; done

echo -e "\n$(mois.sh $m) $a\n"
echo " L  M  M  J  V  S  D"

j=0
for (( i=1 ; i<=($(date2jours.sh $a${m}01)-2)%7 ; i++ ))
do
    echo -n "   "
    j=$(($j+1))
done

for (( i=1 ; i<=$(nombreJours.sh $m $a) ; i++ ))
do
    echo -n "$(printf "%02d" $i) "
    j=$(($j+1))
    if (( $j%7==0 )) ; then echo -ne "\n" ; fi
done
echo -ne "\n"
```

Dans cet exécutable, on se sert de deux petits programmes simples : `mois.sh` prend en argument le numéro d'un mois et affiche ce mois en toutes lettres, et `nombreJours.sh` prend en arguments un mois `m` et une année `a` et affiche le nombre de jours du mois `m`/`a`.

MOIS.SH

```
#!/usr/local/bin/bash
#argument : le numéro d'un mois
#sortie : le mois en toutes lettres

case $1 in
    01) echo Janvier ;;
    02) echo Fevrier ;;
    03) echo Mars ;;
    04) echo Avril ;;
    05) echo Mai ;;
    06) echo Juin ;;
    07) echo Juillet ;;
    08) echo Aout ;;
    09) echo Septembre ;;
    10) echo Octobre ;;
    11) echo Novembre ;;
    12) echo Decembre ;;
esac
```


NOMBREJOURS.SH

```
#!/usr/local/bin/bash
#argument : un mois ($1) et une année ($2)
#sortie : le nombre de jours dans le mois

case $1 in
    01|03|05|07|08|10|12) echo 31 ;;
    04|06|09|11) echo 30 ;;
    02) if bissextile.sh $2 ; then echo 29 ; else echo 28 ; fi ;;
esac
```

Voici ce que je récupère quand je tape `calendrier.sh` puis que je rentre `01 2009` :

```
Janvier 2009

 L  M  M  J  V  S  D
        01 02 03 04
05 06 07 08 09 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

3. La question 3 est laissée au lecteur.

NB. La correction pour tous les exécutables de l'agenda est disponible à l'adresse : `http://www.di.ens.fr/~pilaud/enseignement/TP/IS1/agenda.tar`