Enoncé du TP 8 Système

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 12/12/2012

ⓘ Démarrer les PC sous Linux. Les exercices sont à faire via une connexion SSH (normale) sur allegro.

1 Utilisation des boucles

Dans cette partie, nous allons étudier le fonctionnement des boucles en les utilisant en ligne de commandes, avant de les exploiter dans des scripts dans la partie suivante.

Exercice 1

Se familiariser avec l'utilisation de la boucle **while** *avec un test d'expression arithmétique. L'exercice est complété par l'utilisation de l'instruction* **if** *dans le corps de la boucle.*

1. Créer une variable **`var`** contenant 10
2. Sur la ligne de commande, exécuter une boucle **while** qui applique l'algorithme suivant :

   ```
   Tant que var est strictement positif
   Faire
     Décrémenter var de 1
     Afficher "J'en ai enlevé un et var vaut" suivi de la valeur de var
   FinFaire
   ```

   Elle devrait donc afficher les 10 messages :

   ```
   J'en ai enlevé un et var vaut 9
   J'en ai enlevé un et var vaut 8
   J'en ai enlevé un et var vaut 7
   J'en ai enlevé un et var vaut 6
   J'en ai enlevé un et var vaut 5
   J'en ai enlevé un et var vaut 4
   J'en ai enlevé un et var vaut 3
   J'en ai enlevé un et var vaut 2
   J'en ai enlevé un et var vaut 1
   J'en ai enlevé un et var vaut 0
   ```

   Aide : Utiliser le test/évalutation des expressions arithmétiques — **`(`** **`(`**expression**`)`** **`)`** — pour tester la valeur de **`var`** et pour la décrémenter. Prendre soin de réinitialiser **`var`** à 10 après une tentative infructueuse de réponse à la question.
3. Réinitialiser **`var`** à 10 puis reprendre la question 2 en insérant une instruction **if** dans la boucle, pour que le message indique aussi si **`var`** est paire ou impaire. L'affichage devrait cette fois ressembler à :

   ```
   J'en ai enlevé un et var vaut 9 (impaire)
   J'en ai enlevé un et var vaut 8 (paire)
   ...
   J'en ai enlevé un et var vaut 1 (impaire)
   J'en ai enlevé un et var vaut 0 (paire)
   ```

   Aide : un entier est pair si son modulo 2 (c'est-à-dire le reste de sa division entière par 2) est nul.

4. Utiliser **unset** pour supprimer **`var`**

Exercice 2 (attente active)

Utiliser une boucle **while** *pour réaliser une attente active.*

> ⓘ L'exercice précédent a fait intervenir une boucle **while** dont le test de sortie dépend d'une expression arithmétique. À la place d'un **while**, l'utilisation d'une boucle **for** (de type C/C++) aurait été sans doute plus naturelle :
>
> **`for ((var=10; var-- > 0; )); do ... ; done`**
>
> Dans cet exercice, la boucle **while** a un test de sortie de toute autre nature, qui justifie pleinement son utilisation plutôt qu'un **for**.

1. Sur la ligne de commandes, utiliser une boucle **while** pour appliquer l'algorithme suivant (qui est qualifié d'**attente active**) :

```
Initialiser temps_total à 0
Afficher sans aller à la ligne "attente de la création de fictest"
Tant que le fichier ordinaire fictest n'existe pas
Faire
  Attendre 1 seconde          # utiliser la commande :  sleep 1
  Afficher un point sans aller à la ligne
  Incrémenter temps_total de 1
FinFaire
Afficher sur une ligne "secondes attendues : " suivi de temps_total
```

> ⓘ Pour que les messages soient affichés après la saisie de l'algorithme (et pas au fur et à mesure), il faut séparer les commandes de `;` et non d'un retour à la ligne. Mais c'est purement cosmétique et optionnel.

L'algorithme devrait ainsi commencer par afficher :

```
attente de la création de fictest
```

où la ligne se voit ajouter un point toutes les secondes. Si le fichier `fictest` est créé au bout de dix secondes (voir question suivante), on aura à la place :

```
attente de la création de fictest.........
secondes attendues : 10
```

2. Ouvrir un autre terminal et :
   (a) se placer dans le même répertoire (après s'être connecté sur allegro)
   (b) créer un répertoire `fictest` et attendre qu'un ou deux points soient affichés par la boucle : elle ne doit pas s'arrêter !
   (c) supprimer le répertoire `fictest` et créer un fichier ordinaire de même nom. La boucle doit alors s'arrêter.
   (d) supprimer `fictest` et fermer ce terminal

✍ Si la condition de sortie d'une boucle est le succès (ou l'échec) d'une commande, comme ici la commande **test**, alors les boucles **while** et **until** sont certainement les plus appropriées.

Exercice 3

Utilisation des boucles **for**

1. Créer un tableau **tab** contenant les huit éléments :

   ```
   zero un deux trois quatre cinq six sept
   ```

2. Utiliser la boucle **for** de type C/C++ pour ne faire afficher du tableau **tab** que les éléments dont l'indice est pair, en appliquant l'algorithme suivant :

   ```
   Pour i allant de 0 à la fin de tab, avec un pas de 2
   Faire
     Afficher l'élément i de tab
   FinFaire
   ```

   où, par soucis de généricité, la boucle ne doit pas mentionner explicitement la valeur 8 (taille de **tab**). L'affichage doit être :

   ```
   zero
   deux
   quatre
   six
   ```

3. Utiliser la boucle **for** de sh pour ne faire afficher que les éléments de **tab** dont l'indice est impair, en appliquant l'algorithme suivant :

   ```
   Initialiser afficher à faux (0)
   Pour chaque élément elt de tab
   Faire
     Si afficher est vrai
     Alors
       Afficher elt
     FinSi
     Inverser le sens de afficher
   FinFaire
   ```

   où la variable **afficher** contient un entier utilisé comme un booléen, passant de faux (0) à vrai (1) et inversement à chaque itération. Cette boucle doit afficher :

   ```
   un
   trois
   cinq
   sept
   ```

4. Supprimer le tableau **tab**.

Exercice 4

Utilisation de **for** *et des modificateurs de substitution de variables*

1. Dans votre répertoire `tp`, créer une copie (récursive et en mode verbeux) du répertoire `tpunix` nommée `testfor`
2. Se placer dans `testfor`
3. Utiliser la boucle **for** de sh pour renommer tous les fichiers d'extension `.txt` du répertoire en fichiers d'extension `.txt.old` et afficher le renommage réalisé, en appliquant l'algorithme suivant :

   ```
   Pour chaque fichier fic d'extension .txt
   Faire
     Renommer fic en fic.old
     Afficher "fichier 'fic' renommé en 'fic.old'"
   FinFaire
   ```

   où le message doit afficher l'ancien et le nouveau noms du fichier renommé. L'affichage doit ressembler à :

   ```
   fichier 'acrostiche.txt' renommé en 'acrostiche.txt.old'
   fichier 'amphigouri.txt' renommé en 'amphigouri.txt.old'
   fichier 'amphi.txt' renommé en 'amphi.txt.old'
   fichier 'cigale.txt' renommé en 'cigale.txt.old'
   fichier 'simpsons.txt' renommé en 'simpsons.txt.old'
   ...
   ```

   Vérifier que les fichiers ont bien été renommés
4. Utiliser à nouveau la boucle **for** de sh pour annuler l'opération précédente, c'est-à-dire supprimer l'extension `.old` de tous les fichiers dont le nom se termine par `.txt.old`, en appliquant l'algorithme suivant :

   ```
   Pour chaque fichier fic d'extension .txt.old
   Faire
     Renommer en mode bavard fic en supprimant son extension .old
   FinFaire
   ```

   où le nom du fichier sans l'extension `.old` est obtenu en utilisant les substitutions étendues des variables (sous-chaîne avec les motifs). Vérifier que les fichiers ont bien été renommés
5. Recommencer la question 3, en utilisant une boucle **for** de type C/C++ qui applique l'algorithme suivant :

   ```
   Créer un tableau fics contenant le nom des fichiers d'extension .txt
   Pour i allant de 0 à la taille de fics
   Faire
     Affecter à fic le ième élément de fics
     Renommer le fichier fic en lui ajoutant l'extension .old
     Afficher "fichier 'fic' renommé en 'fic.old'"
   FinFaire
   ```

6. Recommencer la question 4, en utilisant une boucle **for** de type C/C++ qui applique l'algorithme suivant :

   ```
   Créer un tableau fics contenant le nom des fichiers d'extension .txt.old
   Pour i allant de 0 à la taille de fics
   Faire
     Affecter à fic le ième élément de fics
     Renommer en mode bavard fic en supprimant son extension .old
   FinFaire
   ```

7. Revenir à `tpunix` et supprimer au passage le répertoire `testfor`

Exercice 5

Normaliser le nom de plusieurs fichiers en utilisant une boucle **for***, les substitutions de variables avec motifs et* **mv**

1. Copier le répertoire `~cpb/public/unix/fics-espaces` dans votre répertoire `tpunix`
2. Se placer dans `fics-espaces` et afficher son contenu. On y voit des fichiers dont le nom comporte des espaces et/ou des séquences **`%20`** (code HTML de l'espace dans les URL), ou aucun des deux.

> ⓘ Il vaut mieux éviter sous Unix de créer des fichiers dont le nom comporte un espace, car il peuvent mettre en défaut certains scripts qui ne les auraient pas prévus (donc mal codés !). Par exemple, si dans l'exercice précédent on a codé dans la boucle du premier renommage l'instruction :
>
> ```
> mv $fic $fic.old
> ```
>
> alors elle échouera si **`fic`** contient un espace. Le programmeur prudent doit prévoir ce cas et empêcher la décomposition en mots en codant l'instruction ainsi :
>
> ```
> mv "$fic" "$fic.old"
> ```
>
> Quant à la présence de la séquence **`%20`** dans un nom de fichier, elle n'est pas problématique mais elle est peu esthétique et nuit à la lisibilité du nom du fichier.

3. Utiliser une boucle **for** afin de normaliser le nom de ces fichiers en remplaçant tout espace et toute séquence **`%20`** par un *underscore* (_), en appliquant l'algorithme suivant :

```
Pour chaque fichier fic du répertoire
Faire
  Créer la variable newfic en remplaçant les espaces dans fic par des _
  Modifier newfic en remplaçant les séquences %20 par des _
  Si fic et newfic sont différents
  Alors
    Renommer en mode bavard fic en newfic
  Sinon
    Afficher "le fichier 'fic' est déjà normalisé"
  FinSi
FinFaire
```

et en tenant compte des considérations suivantes :

- supprimer puis recréer le répertoire si des fichiers ont été renommés mais qu'il y a une erreur dans la réponse ;
- la liste des fichiers substituée à un motif n'est pas sujette au traitement des caractères spéciaux (ni à la décomposition en mots) ;
- le remplacement des espaces et de `%20` peut se faire avec des substitutions de variables avec motif ;
- le caractère **`%`** a un rôle particulier dans cette substitution, qu'il faut lui enlever.

Au final, l'affichage devrait ressembler à :

```
`Albert%20Einstein.txt' -> `Albert_Einstein.txt'
`Asterix le Gaulois.txt' -> `Asterix_le_Gaulois.txt'
le fichier 'Dragon_Ball.txt' est déjà normalisé
`Mickey Mouse.txt' -> `Mickey_Mouse.txt'
le fichier 'Pikachu.txt' est déjà normalisé
```

```
`Une%20Chanteuse 10.txt' -> `Une_Chanteuse_10.txt'
`Une%20Chanteuse 1.txt' -> `Une_Chanteuse_1.txt'
`Une%20Chanteuse 2.txt' -> `Une_Chanteuse_2.txt'
`Une%20Chanteuse 3.txt' -> `Une_Chanteuse_3.txt'
`Une%20Chanteuse 4.txt' -> `Une_Chanteuse_4.txt'
`Une%20Chanteuse 5.txt' -> `Une_Chanteuse_5.txt'
`Une%20Chanteuse 6.txt' -> `Une_Chanteuse_6.txt'
`Une%20Chanteuse 7.txt' -> `Une_Chanteuse_7.txt'
`Une%20Chanteuse 8.txt' -> `Une_Chanteuse_8.txt'
`Une%20Chanteuse 9.txt' -> `Une_Chanteuse_9.txt'
`Winnie L\'ourson%20Face 1.txt' -> `Winnie_L\'ourson_Face_1.txt'
`Winnie L\'ourson%20Face 2.txt' -> `Winnie_L\'ourson_Face_2.txt'
```

Exercice 6

Familiarisation avec la boucle **select**

1. Sur la ligne de commandes, saisir (ou copier/coller) la boucle **select** suivante (l'indentation est facultative) :

```
select choix in "afficher RANDOM" "quitter"; do
  echo "vous avez saisi : $REPLY"
  case "$choix" in
    "afficher RANDOM" )
      echo "RANDOM = $RANDOM"
      ;;
    "quitter" )
      echo "bye bye"
      break
      ;;
    * ) echo 'mauvais choix !'
      ;;
  esac
done
```

2. Saisir différents choix (et notamment plusieurs fois le choix 1) puis quitter. S'assurer d'avoir compris le mode opératoire de cette boucle.

2 Écriture de scripts

Rappel : la première ligne de tous les scripts créés doit être :

```
#!/bin/bash
```

De plus, il faut avoir le droit d'exécution sur un script avant de pouvoir l'exécuter. Tester les scripts écrits.

Exercice 7

Script qui normalise le nom des fichiers contenus dans des répertoires passés en arguments.

En s'inspirant de l'exercice 5, écrire un script **`normrep.bash`** qui prend en arguments des répertoires, et qui normalise —remplace les espaces et les séquences `%20` par des underscores (_)— le nom des fichiers contenus dans ces répertoires. Si aucun argument n'est donné, le traitement se fera sur le répertoire de travail. Le script doit renvoyer 1 comme valeur de retour si les droits sont insuffisants pour traiter un répertoire, et 0 sinon.
Il est demandé de définir deux fonctions :

- **normfic** qui prend en unique argument le nom d'un fichier et qui le normalise (renomme). Cette fonction applique l'algorithme suivant :

```
Créer une variable locale fic contenant l'argument de la fonction
Créer une variable locale newfic contenant fic normalisé
Si fic et newfic sont différents
Alors
  Renommer en mode bavard fic en newfic
Sinon
  Afficher "le fichier 'fic' est déjà normalisé"
FinSi
```

- **trtrep** qui prend en unique argument le nom d'un répertoire et qui, si possible, normalise les fichiers qu'il contient. Cette fonction applique l'algorithme suivant :

```
Créer une variable locale rep contenant l'argument de la fonction
Si l'utilisateur courant n'a pas tous les droits sur rep
Alors
  Afficher l'erreur "droits insuffisants pour traiter 'rep'"
  Retourner 1
FinSi
Créer une variable locale repcourant contenant le répertoire de travail
Se déplacer dans le répertoire rep
Pour chaque fichier fic du répertoire
Faire
  Appliquer normfic sur fic
FinFaire
Revenir au répertoire repcourant
```

Enfin, le corps du script doit appliquer l'algorithme suivant :

```
Créer un tableau reps contenant le répertoire de travail
Si le script a des arguments
Alors
  Modifier reps pour contenir tous les arguments
FinSi

Initialiser une variable retour à 0
Pour chaque élément rep de reps
Faire
  Appliquer trtrep sur rep
  Si trtrep a échoué
  Alors
    Affecter 1 à retour
  FinSi
FinFaire
Terminer en renvoyant la valeur de retour
```

Exercice 8

Scripts affichant un menu proposant des actions à réaliser. Cet exercice illustre l'emploi de **select**, *ainsi que de* **test** *pour vérifier des conditions sur les fichiers/répertoires.*

1. Écrire un script **`menu1.bash`** qui utilise l'instruction **select** pour afficher un menu proposant les choix suivants :
   - afficher le (chemin) du répertoire de travail (répertoire courant)
   - se déplacer dans un répertoire
   - afficher le contenu détaillé du répertoire courant
   - afficher le contenu détaillé d'un répertoire
   - quitter

   et qui exécute les actions sélectionnées ou affiche un message d'erreur si le choix est incorrect. S'assurer que les messages d'erreur soient écrits sur la sortie d'erreur. Pour les choix 2 et 4, la référence du répertoire doit être demandée à l'utilisateur (saisie au clavier).
2. Copier **`menu1.bash`** en nommant la copie **`menu2.bash`**. Modifier **`menu2.bash`** et utiliser la commande **test** pour vérifier que l'action demandée est possible avant d'effectuer la commande. Par exemple, pour se déplacer dans un répertoire, il faut que la référence saisie par l'utilisateur soit celle d'un répertoire qui est accessible en exécution pour l'utilisateur. Si les conditions ne sont pas réunies, ne pas exécuter la commande mais écrire un message d'erreur explicatif.

> ✍ Il peut être utile de remarquer que si l'on change de répertoire dans le shell qui exécute le script, on n'en change pas sur le shell depuis lequel on a exécuté le script.

Exercice 9

Tri de nombres en arguments et liens physiques

1. Copier le script `~cpb/public/unix/`**`sortup.bash`** dans votre répertoire et l'éditer. Il prend en paramètre des entiers (ce qu'il ne vérifie pas) qu'il place dans un tableau pour les trier dans l'ordre croissant, en appliquant l'algorithme du **tri par sélection**. Le tableau trié est ensuite affiché.
2. Créer un lien physique de **`sortup.bash`** appelé **`sortdn.bash`**
3. Modifier **`sortup.bash`** (ou **`sortdn.bash`**, ce sont les mêmes !) de manière à ce que s'il est appelé par le nom **`sortdn.bash`**, le tri se fasse dans l'ordre décroissant.
   Aide : la référence par laquelle le script est exécuté est dans le paramètre positionnel **`$0`**. Cependant, si le script est appelé en utilisant un chemin tel que **`/chemin/vers/sortup.bash`** alors il faudra supprimer de **`$0`** la partie `/chemin/vers/`. La substitution de sous-chaîne de variable avec les motifs est très appropriée car **`${0##*/}`** donnera le nom seul du script (pour notre exemple, `sortup.bash`).

Exercice 10

Script gérant un compte bancaire

Écrire un script **`gestion.bash`** dont le rôle est de gérer un compte "bancaire". Ce script fonctionnera avec deux arguments : le premier indique le sens (**-c** pour crédit ou **-d** pour débit) de l'opération et le second la somme à débiter ou à créditer sur le compte. Le script demandera également de façon interactive (par saisie au clavier) un libellé pour l'opération. Il n'est pas demandé de vérifier que le second argument est bien numérique, ni que les fichiers peuvent bien être modifiés ou créés, même si en pratique il faudrait le faire.

Après avoir vérifié le nombre d'arguments et que le premier est bien **-c** ou **-d**, le script doit modifier les deux fichiers **`operations.txt`** et **`solde.txt`**, contenus ou créés dans le répertoire `tpunix` de l'utilisateur, comme suit :

- ajouter, à la fin du fichier `operations.txt`, une ligne de la forme :

  date tabulation opération tabulation montant tabulation libellé

  où *tabulation* est le caractère de tabulation, qui sépare les 4 champs :
  - ◇ *date* qui suit le format *jj/mm/aaaa* et qu'on obtient en exécutant la commande **date** avec l'argument (format) **`+%x`** ;
  - ◇ *opération* qui vaut **`crédit`** ou **`débit`** ;
  - ◇ *montant* qui est le montant (entier) de la transaction, communiqué en argument du script ;
  - ◇ *libellé* qui est le libellé saisi par l'utilisateur ;
- écraser le fichier `solde.txt` qui contient le solde précédent, et y stocker le nouveau solde. Si `solde.txt` n'existait pas, alors le solde précédent est nul.

Exercice 11

Gestion de fichiers musicaux au format MP3

Dans cet exercice, nous allons réaliser une gestion sommaire de fichiers musicaux au format MP3 contenant des ***tags*** (étiquettes) ID3V2. Ces tags permettent d'intégrer au fichier des informations telles que l'auteur, le titre, l'année, l'album, le numéro de piste et d'autres informations. Ils peuvent être utilisés pour classer les fichiers, les renommer, etc. Pour réaliser la gestion de fichiers MP3, on utilisera la commande **id3v2** qui est un utilitaire pouvant afficher/modifier/ajouter des tags ID3V2 d'un fichier MP3.

L'option **-l** (L minuscule) de **id3v2** extrait les tags ID3V2 du fichier MP3 indiqué en argument, et les écrit sur la sortie standard.

Exemple :

```
$ id3v2 -l unfichier.mp3
id3v2 tag info for unfichier.mp3:
TIT2 (Title/songname/content description): Gospel with no lord
TPE1 (Lead performer(s)/Soloist(s)): Camille
TALB (Album/Movie/Show title): Music Hole
TRCK (Track number/Position in set): 1
TYER (Year): 2008
TCON (Content type): Alternative (20)
COMM (Comments): ()[eng]: ---
TENC (Encoded by): LAME Ain't an MP3 Encoder
TLAN (Language(s)): English
```

➪ **id3v2** nous indique que le fichier `unfichier.mp3` contient notamment les tags ID3V2 suivants :

- **TIT2** indique le titre du morceau : «`Gospel with no lord`»;
- **TPE1** indique l'auteur (interprète) : «`Camille`»;
- **TALB** indique le titre de l'album : «`Music Hole`»;
- **TYER** indique l'année : «`2008`»;
- **TRCK** indique le numéro du morceau (piste) dans l'album : «`1`».

On peut voir que d'autres tags existent mais nous ne les utiliserons pas.

✍ Noter que l'ordre d'affichage des tags peut varier.

Pour modifier/ajouter un tag, il faut utiliser **id3v2** avec la syntaxe :

id3v2 --*tag mot référence*

où *tag* est le tag à ajouter/modifier (tel que **TIT2**, **TPE1**, **TALB**, **TYER** et **TRCK**) dans le fichier *référence*, et *mot* est la chaîne à lui attribuer (attention aux caractères spéciaux !).

Exemple :

```
$ id3v2 --TIT2 'Ceci est le nouveau titre' un_fichier.mp3
$ id3v2 -l un_fichier.mp3
id3v1 tag info for un_fichier.mp3:
Title  : Ceci est le nouveau titre       Artist: Camille
Album  : Music Hole                      Year: 2008, Genre: Unknown (255)
Comment: ---                             Track: 1
id3v2 tag info for un_fichier.mp3:
TPE1 (Lead performer(s)/Soloist(s)): Camille
TALB (Album/Movie/Show title): Music Hole
TRCK (Track number/Position in set): 1
TYER (Year): 2008
TCON (Content type): Alternative (20)
COMM (Comments): ()[eng]: ---
TENC (Encoded by): LAME Ain't an MP3 Encoder
TLAN (Language(s)): English
TIT2 (Title/songname/content description): Ceci est le nouveau titre
```

➪ on peut remarquer que le tag **TIT2** a changé de place suite à sa modification, et que **id3v2** a aussi ajouté des tags ID3V1 (le fichier contient maintenant les deux versions de tags ID3V2 et ID3V1, mais ça n'a aucune conséquence sur ce qui suit).

1. Copier le répertoire `~cpb/public/unix/music` dans votre répertoire `tpunix`. Il contient des (extraits de) fichiers musicaux, et des fichiers texte. Utiliser **id3v2** pour afficher les informations sur certains de ces fichiers.
2. Écrire un script bash **tagedit.bash** qui prend en argument (paramètre) le nom (référence) d'un fichier musical MP3 et qui :
   (a) définit les fonctions **get_auteur**, **get_titre**, **get_album**, **get_annee** et **get_piste** qui prennent en argument le nom d'un fichier et qui écrivent sur la sortie l'information correspondante si elle est présente dans le fichier (n'écrivent rien si elle ne l'est pas) ;
   (b) définit les fonctions **set_auteur**, **set_titre**, **set_album**, **set_annee** et **set_piste** qui prennent en arguments le nom d'un fichier ainsi qu'une chaîne, et qui fixent le tag adéquat du fichier, à la chaîne passée en deuxième argument ;

(c) dans le corps du script :

i. vérifie qu'il y a bien un seul argument sinon affiche un message d'erreur et se termine en renvoyant 2 comme une valeur de retour ;

ii. vérifie que l'argument correspond à un fichier ordinaire accessible en lecture, sinon se termine en affichant un message d'erreur et en renvoyant 1 comme valeur de retour ;

iii. boucle sur un menu proposant les choix suivants :

- afficher l'auteur ;
- afficher le titre ;
- afficher l'album ;
- afficher l'année ;
- afficher le numéro de piste ;
- modifier/ajouter l'auteur ;
- modifier/ajouter le titre ;
- modifier/ajouter l'album ;
- modifier/ajouter l'année ;
- modifier/ajouter le numéro de piste ;
- quitter.

Si l'utilisateur choisit un affichage, faire appel à la fonction correspondante. Pour une modification, vérifier que le fichier est modifiable sinon afficher un message d'erreur. Puis, afficher l'information courante et demander à l'utilisateur de saisir la nouvelle valeur du tag. Pour le numéro de piste et l'année, vérifier qu'il s'agit d'un nombre (entier positif ou nul), sinon afficher un message d'erreur. En cas de mauvais choix du menu, afficher un message d'erreur.

Exemples :

```
$ id3v2 -l un_fichier.mp3
id3v2 tag info for un_fichier.mp3:
TIT2 (Title/songname/content description): Gospel with no lord
TPE1 (Lead performer(s)/Soloist(s)): Camille
TALB (Album/Movie/Show title): Music Hole
TRCK (Track number/Position in set): 1
TYER (Year): 2008
TCON (Content type): Alternative (20)
COMM (Comments): ()[eng]: ---
TENC (Encoded by): LAME Ain't an MP3 Encoder
TLAN (Language(s)): English
un_fichier.mp3: No ID3v1 tag
$ tagedit.bash un_fichier.mp3
1) Afficher l'auteur           7) Modifier/Ajouter le titre
2) Afficher le titre           8) Modifier/Ajouter l'album
3) Afficher l'album         9) Modifier/Ajouter l'année
4) Afficher l'année        10) Modifier/Ajouter la piste
5) Afficher le numéro de piste    11) Quitter
6) Modifier/Ajouter l'auteur
Que voulez-vous faire sur 'un_fichier.mp3' ? 1
Camille
Que voulez-vous faire sur 'un_fichier.mp3' ? 2
Gospel with no lord
```

```
Que voulez-vous faire sur 'un_fichier.mp3' ? 7
Titre actuel : Gospel with no lord
Nouveau titre ? Mon nouveau titre
Que voulez-vous faire sur 'un_fichier.mp3' ? 10
Piste actuelle : 1
Nouvelle piste ? premiere
erreur : 'premiere' n'est pas un nombre
Que voulez-vous faire sur 'un_fichier.mp3' ? 10
Piste actuelle : 1
Nouvelle piste ? 21
Que voulez-vous faire sur 'un_fichier.mp3' ? 11
$ id3v2 -l un_fichier.mp3
id3v1 tag info for un_fichier.mp3:
Title  : Mon nouveau titre              Artist: Camille
Album  : Music Hole                     Year: 2008, Genre: Unknown (255)
Comment: ---                            Track: 21
id3v2 tag info for un_fichier.mp3:
TPE1 (Lead performer(s)/Soloist(s)): Camille
TALB (Album/Movie/Show title): Music Hole
TYER (Year): 2008
TCON (Content type): Alternative (20)
COMM (Comments): ()[eng]: ---
TENC (Encoded by): LAME Ain't an MP3 Encoder
TLAN (Language(s)): English
TIT2 (Title/songname/content description): Mon nouveau titre
COMM (Comments): (ID3v1 Comment)[XXX]: ---
TRCK (Track number/Position in set): 21
```

3. Écrire le script bash **mp3import.bash** et y copier les fonctions **get_auteur**, **get_titre**, **get_album**, **get_annee**, et **get_piste**. Ce script prend en argument un répertoire, et importe les fichiers musicaux qu'il contient dans la collection musicale de l'utilisateur courant, organisée dans le répertoire **~/musique** comme suit :
   - `~/musique` contient autant de répertoires que d'auteurs d'un fichier musical de la collection : chaque répertoire porte le nom d'un auteur.
     Exemple : `~/musique/Camille`
   - chaque `~/musique/`*auteur* contient autant de répertoires que d'albums de cet auteur mentionnés dans la collection. Le nom de ces répertoires suit le format *année*`-`*album* (les morceaux d'un même album sont supposés être taggés avec la même année).
     Exemple : `~/musique/Camille/2008-Music␣Hole`
   - chaque `~/musique/auteur/`*annee*`-`*album* contient les morceaux (fichiers) de l'album. Ils ont les mêmes tags *auteur*, *album* et *annee*. Le nom des morceaux a comme format *piste*`-`*titre*`.mp3` où la *piste* est formée de 2 caractères (comme `01`, `02`, etc.).
     Exemple : `01-Gospel␣with␣no␣lord.mp3`

Exemple de collection :

```
$ ls ~/musique
Camille  Gossip
```

➪ il y a 2 auteurs dans la collection : *Camille* et *Gossip*

```
$ ls ~/musique/Camille
2008-Music Hole
```

➪ pour l'auteur *Camille*, la collection contient l'album *Music Hole* de *2008*


```
$ ls ~/musique/Camille/'2008-Music Hole'
01-Gospel with no lord.mp3     05-The monk.mp3       09-Winter's child.mp3
02-Canards sauvages.mp3        06-Cats and dogs.mp3  10-Waves.mp3
03-Home is where it hurts.mp3  07-Money note.mp3     11-Sanges Sweet.mp3
04-Kfir.mp3                    08-Katie's tea.mp3
```

➪ cet album contient 11 morceaux. L'ordre lexicographique du nom des fichiers correspond à leur ordre dans l'album.

Durant l'importation, **mp3import** doit créer les répertoires nécessaires et déplacer les fichiers musicaux en les renommant. Pour cela, il doit commencer par créer la collection musicale de l'utilisateur (`~/musique`) si elle n'existe pas. Puis, pour chaque fichier du répertoire en argument :

- vérifie que c'est un fichier ordinaire d'extension `.mp3` sans distinction de casse, sinon le saute en mentionnant sur la sortie qu'il est ignoré.
- crée les variables **`auteur`**, **`titre`**, **`annee`**, **`album`** et **`piste`** en utilisant les fonctions adéquates et modifie éventuellement **`piste`** pour qu'elle contienne exactement 2 caractères (en ajoutant si besoin un zéro (**`0`**) en début). On suppose que les tags nécessaires sont présents dans le fichier ;
- vérifie si l'auteur existe déjà dans la collection, sinon crée le répertoire correspondant et affiche un message informant de la création d'un nouvel auteur ;
- vérifie si l'album (répertoire de nom *annee-album*) existe déjà pour cet auteur, sinon crée le répertoire correspondant et affiche un message informant de la création d'un nouvel album de l'auteur ;
- déplace le fichier musical dans le répertoire de l'album, en le renommant selon le format :

piste`-`*titre*`.mp3`

Exemple :

```
$ ls depot
fichier_divers    fic_music_15.MP3  fic_music_20.mp3  fic_music_5.mp3
fic_music_10.mp3  fic_music_16.mp3  fic_music_21.mp3  fic_music_6.mp3
fic_music_11.mP3  fic_music_17.mp3  fic_music_22.mp3  fic_music_7.mp3
fic_music_12.mp3  fic_music_18.mp3  fic_music_2.mp3   fic_music_8.mp3
fic_music_13.Mp3  fic_music_19.mp3  fic_music_3.mp3   fic_music_9.mp3
fic_music_14.mp3  fic_music_1.mp3   fic_music_4.mp3   ...
```

➪ le répertoire `depot` semble contenir des fichiers musicaux ayant diverses extensions

```
$ ./mp3import.bash depot
fichier 'depot/fichier_divers' ignoré
Nouvel auteur : Izia
Nouvel album de 'Izia' : Izia
Nouvel auteur : Applause
Nouvel album de 'Applause' : Where It All Began
$ ls depot
fichier_divers
```

➪ à part `fichier_divers` qui a été ignoré, tous les fichiers de `depot` ont été classés dans la collection musicale

```
$ ls ~/musique
Applause  Camille  Gossip  Izia
```

➪ deux nouveaux auteurs ont été ajoutés à la collection : *Applause* et *Izia*

```
$ ls ~/musique/Applause
2011-Where It All Began
```

➪ pour *Applause*, l'album *Where It All Began* de *2011* a été importé

```
$ ls ~/musique/Applause/'2011-Where It All Began'
01-All about you.mp3       07-Feelings.mp3
02-Road to nowhere.mp3     08-The woods.mp3
03-Black sand.mp3          09-A way out of blue.mp3
04-Hope youre better.mp3   10-Where it all began.mp3
...
```

➪ les fichiers de l'album sont des fichiers de `depot` qui ont été renommés.

Exercice 12

Gestion d'une bibliothèque

Nous allons écrire des scripts permettant de gérer les emprunts de livres par des adhérents d'une bibliothèque. Pour cela, 4 fichiers de données sont utilisés (`membres`, `livres`, `exemplaires` et `emprunts`), contenant des champs séparés par un `;` (point-virgule). Chaque ligne de ces fichiers représente un tuple de la base de données :

- `membres` contient la liste des membres (adhérents) de la bibliothèque. Il est constitué de lignes de la forme :

  num_membre ; *nom_membre* ; *prénom_membre* ; *adresse_membre*

  où les champs contiennent respectivement le numéro, le nom, le prénom et l'adresse de l'adhérent (membre). **Le fichier est trié par ordre croissant de numéro de membre.** Ce numéro est attribué automatiquement lors de l'inscription du membre ;

- `livres` contient la liste des livres de la bibliothèque. Il est constitué de lignes de la forme :

  num_livre ; *titre_livre* ; *auteur*

  où les champs contiennent respectivement le numéro, le titre et l'auteur du livre. **Le fichier est trié par ordre croissant de numéro de livre.** Ce numéro est attribué automatiquement lors de l'inscription du livre ;

- `exemplaires` contient la liste des exemplaires des livres de la bibliothèque et indique pour chacun s'il est disponible ou non. Il est constitué de lignes de la forme :

  num_livre ; *num_exemplaire* ; *disponibilité*

  où *disponibilité* vaut **`oui`** ou **`non`** selon que l'exemplaire *num_exemplaire* du livre *num_livre* est disponible (non emprunté) ou non ;

- `emprunts` contient la liste des emprunts en cours. Il est constitué de lignes de la forme :

  num_membre ; *num_livre* ; *num_exemplaire* ; *date*

  indiquant que l'adhérent *num_membre* a emprunté (et non encore rendu) à la date *date* l'exemplaire *num_exemplaire* du livre *num_livre*. Le format de *date* est :

  num_jour ␣ *num_mois* ␣ *année* ␣ *heure* : *minutes* : *secondes*

  où *num_jour* est le numéro du jour dans le mois (de 1 à 31) et *num_mois* est le numéro du mois dans l'année (de 1 à 12). La date courante au bon format est obtenue par la commande :

  `date '+%-d %-m %Y %X'`

Exemples de contenus :

```
$ cat membres
1;dupont;albert;marseille
2;martin;marcelle;nimes
3;mart;leon;aix
$ cat livres
1;quatre_mousquetaires;dumas
2;robinson_crusoe;defoe
3;de_la_terre_a_la_lune;vernes
4;bible;anonyme
$ cat exemplaires
1;1;non              Ce qui signifie qu'il existe les exemplaires 1, 2 et 3
1;2;non              du livre n°1 (quatre_mousquetaires)
1;3;oui              et que seul le 3 est disponible
2;1;oui              ...
3;1;oui
4;2;oui
4;3;oui
4;4;oui
4;5;oui
4;1;non
$ cat emprunts
2;1;1;1 2 2007 14:20:10
2;4;1;1 2 2007 14:22:22
1;1;2;2 4 2007 17:37:41
```

L'ordre des lignes n'est pas important pour les fichiers `exemplaires` et `emprunts`.

1. Écrire un script **`inscrire.bash`** qui avec l'option **-a** ajoute un nouvel adhérent à la fin du fichier `membres`. Les 3 arguments suivants sont le nom, le prénom et l'adresse du membre. Le numéro de membre doit être attribué automatiquement par le script (incrémenter le numéro du membre figurant en dernière ligne). On suppose pour simplifier qu'il n'existe pas d'homonyme, et que les informations ne contiennent pas d'espace.

   Avec l'option **-l**, le script ajoute un livre à la fin du fichier `livres` et ajoute ses exemplaires à la fin du fichier `exemplaires`. Les 3 aguments qui suivent sont le nombre d'exemplaires, le titre et l'auteur du livre. Le numéro de livre doit être attribué automatiquement (incrémenter le numéro du livre figurant en dernière ligne). Pour simplifier, on suppose que le livre n'existe pas déjà (il ne s'agit donc pas d'ajout d'exemplaire), et que les informations ne contiennent pas d'espace.
2. Écrire un script **`demande.bash`** qui prend 2 arguments : le nom de l'adhérent et le titre du livre. Le script vérifiera la validité de ceux-ci (présence dans leur fichier respectif) puis contrôlera si un exemplaire du livre est disponible. Si tout est correct, le script insére un nouvel emprunt à la fin du fichier `emprunts` et modifie le fichier `exemplaires` pour marquer l'exemplaire emprunté comme indisponible.
   S'il y a erreur, le script affichera le message adéquat et le code de retour sera mis à jour suivant l'erreur rencontrée.
3. Écrire un script **`rend.bash`** qui accepte 2 arguments : le nom de l'adhérent et le titre du livre. Le script doit vérifier la validité de ceux-ci (présence dans leur fichier respectif) puis contrôler si l'emprunt est bien enregistré dans le fichier `emprunts`. Si oui le script le supprime du fichier et modifie le fichier `exemplaires` pour remettre l'exemplaire disponible (il n'est pas nécessaire de vérifier qu'il ne l'était pas).
4. Écrire un script **`comptemps.bash`** qui compare la date courante à une date donnée en arguments dans le même format que celui obtenu par **`date '+%-d %-m %Y %X'`** et renvoie :
   - 0 si la date en argument est postérieure ou égale à la date courante ;
   - 1 si la date en argument est antérieure à la date courante de plus d'un an ;

- 2 si la date en argument est antérieure à la date courante de plus d'un mois ;
- 3 si la date en argument est antérieure à la date courante de plus d'un jour.

Le script reçoit 4 arguments représentant la date : le premier est le numéro de jour dans le mois, le second le numéro du mois dans l'année, etc.
La comparaison des dates est une opération fastidieuse si l'on veut prendre en compte les particularités (nombre de jours différents selon les mois et les années, etc.). On peut simplifier en transformant les dates en nombre de jours depuis la date 00/00/0000 en considérant qu'un mois comprend 30 jours et une année 365 jours. Ensuite, il suffit de comparer ces dates "normalisées"...

5. Écrire un script **`rappel.bash`** qui vérifie les emprunts datant de plus d'un mois et affiche les lignes correspondantes. Pour cela, traiter le fichier `emprunts` ligne à ligne pour en extraire la date d'emprunt, la comparer à la date courante en utilisant **`comptemps.bash`**, et afficher la ligne si un rappel doit être fait.

Exercice 13

Le pendu

Écrire un script **`pendu.bash`** qui permet de jouer au pendu. Pour cela, on suppose qu'on dispose d'un fichier `mots` contenant une liste de mots, à raison d'un par ligne. Au démarrage, le script utilise la variable **`RANDOM`** pour prendre un mot au hasard dans `mots`, qui sera le mot à trouver. Attention, **`RANDOM`** contient une valeur aléatoire comprise entre 0 et $2^{15} - 1$ donc ne correspondant forcément à un numéro de ligne existant du fichier `mots`. Le script doit alors écrire le premier caractère de ce mot ainsi que le dernier, et entre les deux autant de points qu'il y a de caractères à trouver. Une lettre est alors demandée à l'utilisateur. Si cette lettre figure parmi les caractères du mot non encore découverts, alors le joueur vient de découvrir une lettre qui sera par la suite toujours affichée à la place du (ou des) point(s) correspondant(s) avec les autres caractères découverts. Sinon, le joueur a fait une erreur. Le joueur a droit à 7 erreurs. S'il découvre la totalité du mot, alors il a gagné et le script se termine avec un code de retour à 0. S'il fait plus de 7 erreurs, il a perdu et le code de retour est 1. **Aide :** on aura besoin d'accéder à des caractères du mot à trouver. Pour cela, il y a plusieurs possibilités parmi lesquelles :

- (1re version du corrigé) travailler directement sur les (sous-)chaînes. Soit une variable **`mot`** (contenant une chaîne de caractère), les opérations suivantes peuvent être utilisées :
  - ◇ **`${mot:i:1}`** est le caractère de **`mot`** d'indice **`i`** (le premier est à l'indice 0)
  - ◇ **`${mot: -1}`** est le dernier caractère de **`mot`**
  - ◇ **`${mot:0:i}`** est le début de **`mot`** jusqu'au caractère d'indice **`i`** non compris. Cette sous-chaîne est éventuellement vide
  - ◇ **`${mot:i+1}`** est la fin de **`mot`** à partir du caractère d'indice **`i+1`**. Cette sous-chaîne est éventuellement vide
- (2e version du corrigé) placer le mot à trouver dans un tableau, où chaque élément est un caractère du mot, ce qu'on peut réaliser en utilisant la commande **sed**. *A priori* plus lourde, elle permet toutefois de modifier les éléments du tableau sans utiliser les sous-chaînes.

Exemple :

```
$ pendu.bash
p.........n : lettre ? a
p...a.a...n : lettre ? d
Erreur : plus que 6 erreurs permises
p...a.a...n : lettre ? t
p...a.at..n : lettre ? i
p...a.ati.n : lettre ? o
```

```
p...a.ation : lettre ? g
Erreur : plus que 5 erreurs permises
p...a.ation : lettre ? e
p.e.a.ation : lettre ? r
pre.aration : lettre ? p
Gagné : preparation
```

Exercice 14

Scripts gérant un carnet d'adresses

1. Copier dans votre répertoire `tpunix`, le script `~cpb/public/unix/`**`phone1.bash`**. Ce script permet soit :
   - de rajouter une ligne dans un fichier appelé `telephones.txt` (contenu dans le répertoire `tpunix` de l'utilisateur ou créé à cette occasion). La ligne doit être constituée d'un nom, d'un prénom, et d'informations (comme un numéro de téléphone ou autres), séparés par une tabulation.
     En toute rigueur, il faudrait vérifier que `tpunix` existe, que si `telephones.txt` existe, il est modifiable ou, s'il n'existe pas, que `tpunix` est modifiable. Mais on considèrera que les droits sont bien positionnés pour `tpunix` et pour `telephones.txt` (s'il existe).
   - d'afficher le contenu de `telephones.txt` en paginant. L'existence de `telephones.txt` est testée pour afficher un message en conséquence (plutôt que rien). En revanche, son accessibilité en lecture n'est (et ne sera) pas testée.

   L'utilisateur peut déclencher ces opérations par menu si aucun argument n'est spécifié, ou directement à partir de la ligne de commandes en spécifiant comme 1er argument :
   - **-a** pour l'ajout. Dans ce cas, il faut au moins 4 arguments : les nom et prénom sont les 2e et 3e arguments, et les informations sont tout ce qui suit le 3e argument ;
   - **-l** pour l'affichage paginé du fichier. Dans ce cas, **-l** doit être le seul argument.

   > ✍ Dans ce qui suit, il est plus simple de considérer que le nom ne contient pas d'espace.

   Analyser et tester ce script. Il propose un squelette pour la gestion du répertoire, mais vous êtes libre de le modifier comme bon vous semble.

   Son code est en gros composé des parties suivantes :
   - des définitions :
     - ◇ les variables **`fait`**, **`echec`** et **`erreur`** contiennent les valeurs respectives du code de retour selon que l'opération a été réalisée avec succès, est infructueuse, ou en cas de mauvaise utilisation ;
     - ◇ le tableau **`tab_menu`** contenant la liste des actions possibles proposées par le menu
     - ◇ la tableau **`tab_fonc_menu`** contenant la liste des fonctions à appeler selon l'action choisie
   - les différentes fonctions utiles :
     - ◇ des fonctions d'usage général
     - ◇ des fonctions pour la gestion du répertoire et de lecture d'informations selon l'action choisie par le menu

- l'affichage du menu si aucun argument n'est spécifié : d'autres choix seront proposés par la suite, simplement en modifiant **`tab_menu`** et **`tab_fonc_menu`** et en écrivant la ou les fonctions correspondantes
- le traitement des arguments s'il y en a : les arguments (et notamment le premier) pourront avoir d'autres formes par la suite ;

2. Copier **`phone1.bash`** dans **`phone2.bash`**. Modifier **`phone2.bash`** de manière à rajouter la possibilité d'afficher en paginant les lignes de `telephones.txt` qui commencent par une certaine chaîne. Le script renvoie **`$fait`** uniquement si la recherche trouve des lignes correspondantes. Cette recherche peut être demandée :
   - à partir du menu, et la chaîne devra être saisie ;
   - à partir de la ligne de commandes, où la chaîne est le premier argument. Pour plus de rigueur, on peut exiger que la chaîne ne commence pas par un tiret, pour éviter les ambiguïtés avec une option non reconnue.

   Si `telephones.txt` n'existe pas, le script doit renvoyer **`$echec`** comme code de retour. Si aucun élément n'est trouvé, indiquer un message en conséquence et le code de retour doit aussi être **`$echec`**.

   ✍ Pour effectuer la recherche, utiliser **grep** qui renvoie 1 si aucune ligne ne correspond à l'expression régulière.

3. Copier **`phone2.bash`** dans **`phone3.bash`**. Modifier **`phone3.bash`** de manière à rajouter la possibilité de supprimer les lignes de `telephones.txt` dont le premier mot (le nom) est une certaine chaîne. Le script renvoie **`$fait`** uniquement si des lignes sont supprimées. Cette suppression peut être demandée :
   - à partir du menu, et la chaîne devra être saisie ;
   - à partir de la ligne de commandes, où le premier argument est **-d** et la chaîne est le second argument.

   Si `telephones.txt` n'existe pas, le script doit renvoyer **`$echec`** comme code de retour. Afficher les éléments qui seront supprimés. Si aucun élément n'est supprimé, indiquer un message en conséquence et le code de retour doit aussi être **`$echec`**.

   ✍ Pour supprimer des lignes d'un fichier, il suffit de le recréer avec toutes ses lignes sauf celles dont on ne veut plus. Attention, on ne peut pas rediriger la sortie de **grep** sur le même fichier qui lui sert d'entrée. Il faut passer par un fichier intermédiaire...

4. Copier **`phone3.bash`** dans **`phone4.bash`**. Modifier **`phone4.bash`** de manière à rajouter la possibilité de supprimer les lignes de `telephones.txt` dont le premier mot est une certaine chaîne, et de les remplacer par une unique ligne. Le script renvoie **`$fait`** uniquement si des lignes sont remplacées. Cette modification peut être demandée :
   - à partir du menu, et la chaîne, le nouveau nom, le nouveau prénom et les nouvelles informations devront être saisies ;
   - à partir de la ligne de commandes, où le premier argument est **-u**, le second argument est la chaîne, le troisième argument est le nouveau nom, le quatrième argument est le nouveau prénom et les nouvelles informations sont tout ce qui suit le quatrième argument.

   Si `telephones.txt` n'existe pas, le script doit renvoyer **`$echec`** comme code de retour. Afficher les éléments qui seront remplacés. Si aucun élément n'est remplacé, indiquer un message en conséquence et le code de retour doit aussi être **`$echec`**.