Introduction aux systèmes d'exploitation (IS1)

TP 9 – Agenda
2. Fonctions avancées

Dans ce TP, on utilise des boucles en `shell` pour programmer des fonctions plus avancées sur l'agenda. On utilise les exécutables développés dans les 4 premières parties du TP8.

1 Structures de contrôle

1.1 Boucle `for`

La boucle `for` permet d'exécuter successivement une suite de commandes pour chaque valeur que l'on donne à un paramètre. Voici deux syntaxes possibles :

• La première syntaxe a la forme suivante :

```
for var in liste ; do commandes ; done
```

Ici, le paramètre est une variable (et *var* est son nom), *liste* une liste d'*expressions régulières* séparées par des espaces et *commandes* une liste de commandes. Lors de l'exécution, *var* prend successivement chaque valeur apparaissant dans *liste*, et *commandes* est executée pour chacune de ces valeurs. Voici un exemple de boucle :

```
for prenom in Yvonne J{ean{ne,},acques} ; do echo Bonjour $prenom ; done
```

La sortie produite par cette boucle sur le terminal est :

```
Bonjour Yvonne
Bonjour Jeanne
Bonjour Jean
Bonjour Jacques
```

Notez que la liste de valeurs peut contenir également des expressions régulières du bash. Les valeurs correspondantes seront alors les fichiers représentés par les chemins donnés par ces expressions régulières.

Par exemple, `for fichier in *.java ; do echo $fichier ; done` est équivalent à `ls *.java`

• Il est également possible d'utiliser une syntaxe plus proche d'autres langages, tels Java, lorsque le paramètre est une variable prenant des valeurs entières :

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

Ici, *expr1*, *expr2* et *expr3* sont des expressions arithmétiques comme nous en avons déjà rencontrées. À l'initialisation de la boucle, *expr1* est évaluée. En fin de boucle, si *expr2* est évaluée à 0 alors *expr3* est évaluée et un nouveau tour de boucle commence. Sinon, la boucle se termine. Voici un exemple utilisant cette syntaxe :

```
for (( i=1 ; i <= 1024 ; i*=2 )) do echo $i ; done
```

La sortie produite par cette boucle est la suite des puissances de 2 jusqu'à 2^{10}.

Exercice 1 – *Échauffement.*

1. Ecrivez une boucle `for` qui affiche le message « Bonjour ! » indéfiniment.
   Est-ce possible avec les deux types de boucles `for` ?
2. Ecrivez une boucle `for` qui affiche le nombre de fichiers et sous-répertoires visibles dans le répertoire courant.
3. Ecrivez une boucle `for` qui attend un entier n donné par l'utilisateur et affiche la valeur de 2^n.
   Que se passe-t-il quand vous donnez un n "trop grand" ?

1.2 Boucles `while` et `until`.

La boucle `while` ou « tant que » suit la syntaxe suivante :

```
while condition ; do commandes ; done
```

Dans une boucle while, la liste de commandes *commandes* est exécutée de façon répétée tant que la commande *condition* donne une valeur de retour égale à 0.

La boucle `until` ou « jusqu'à ce que » fonctionne de manière identique, si ce n'est que la liste de commandes est exécutée jusqu'à ce que la valeur de retour de la condition soit égale à 0 :

```
until condition ; do commandes ; done
```

Exercice 2 – *Échauffement.*

1. Ecrivez une boucle `while` et une boucle `until` qui affichent le message « Bonjour ! » indéfiniment.
2. Ecrivez à l'aide d'une boucle `while` puis `until` une commande qui demande un nombre entier à l'utilisateur et affiche toutes les puissances de 2 inférieures à ce nombre.

2 Fonctions avancées de l'agenda

Dans cette deuxième partie, vous développez un peu plus votre application agenda.

Exercice 3 – *Lire une date (bis)*

Modifiez votre exécutable `lireDate.sh` de sorte qu'il demande une date à l'utilisateur jusqu'à ce que l'utilisateur lui entre une date valide. Faites de même avec `lireHeure.sh`.

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

Le but de cet exercice est d'écrire un exécutable qui prend en argument une plage horaire (*i.e.*, une date et heure de début et une date et heure de fin), et affiche tous les événements du fichier `.agenda` qui chevauchent cette plage horaire.

1. Écrire une ligne de commande qui teste si un événement chevauche une plage horaire. *(Il faut et il suffit que le début ou la fin de l'événement (ou les deux) soit situé dans la plage horaire).*
2. Écrire l'exécutable en utilisant une boucle.
3. Comment pourrait-on s'éviter d'écrire une boucle si la plage horaire est un jour complet ? Comparer les vitesses d'exécution (au besoin, créez un très gros fichier `.agenda` en utilisant une boucle...).
4. Pouvez-vous améliorer votre exécutable en utilisant le fait que les lignes du fichier `.agenda` sont triées par ordre chronologique ?

Exercice 5 – *Ajouter un événement périodique*

Le but de cet exercice est d'écrire un exécutable qui ajoute dans le fichier `.agenda` un événement périodique. Par exemple, on voudrait ajouter toutes les semaines un événement pour le TP d'IS1 sans avoir à ajouter chaque TP séparément.

1. Copiez dans votre `~/bin/agenda/` les exécutables `date2jours.sh` et `jours2date.sh` situés dans le répertoire `/home/vpilau94/bin/agenda/`. Le 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.
2. En vous servant de ces deux commandes, écrivez un exécutable `ajouterDate.sh` qui prend comme arguments une date initiale `di` (au format `aaaammjj`), et un nombre de jours `j`, et calcule la date qu'il sera `j` jours après `di`.
3. Écrivez un exécutable `ajouterPeriodique.sh` qui vous demande un événement initial, une fréquence `j` et un nombre d'occurences `n` et ajoute au fichier `.agenda` exactement `n` événements en commençant par l'événement intial et en plaçant un événement tous les `j` jours.
4. Vérifiez votre programme en rentrant tous les TPs d'IS1 de l'année.
5. En utilisant les fonctions de l'agenda, affichez le nombre total de séances d'IS1 dans l'année et celles qui restent.

3 Interfaces

Exercice 6 – *Interface terminal*

1. Développez une petite interface `agenda.sh` qui affiche le menu

   1. Rechercher un événement
   2. Événements sur une plage horaire
   3. Ajouter un événement ponctuel
   4. Ajouter un événement périodique
   5. Supprimer un événement
   6. Quitter

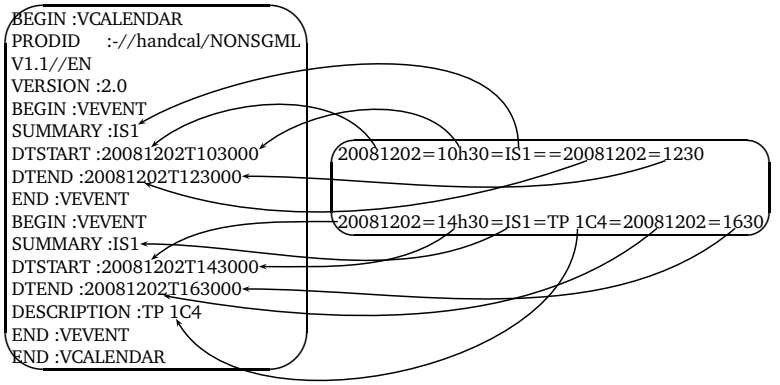
   puis vous demande ce que vous souhaitez faire et s'exécute.
2. Comment peut-on faire pour que l'on revienne toujours au menu de l'interface après l'exécution du programme correspondant à l'un des 5 premiers choix ?

Exercice 7 – *Format* `iCalendar`

Le format `iCalendar` est un standard (`RFC 2445`) pour les échanges de données de calendriers. On utilise l'extension `.ics` pour désigner un fichier réalisant ce format.

Le but de cet exercice est de transformer le fichier `.agenda` que vous avez créé dans votre répertoire racine en un fichier de type `iCalendar` que vous pourrez visionner grâce au logiciel `Sunbird` installé sur les machines du script. Nous n'allons donc pas explorer toutes les subtilités du format `iCalendar`.

La figure suivante donne le contenu d'un fichier `.agenda` et le contenu du fichier associé au format `iCalendar`.



Les 4 premières lignes et la dernière ligne du fichier `iCalendar` sont obligatoires et non modifiables (pour les besoins de l'exercice). Chaque événement est ensuite spécifé entre 2 balises `BEGIN:VEVENT` et `END:VEVENT`. On doit préciser le début de l'événement (attribut `DTSTART`), sa fin (attribut `DTEND`) et son nom (`SUMMARY`) ; on peut également spécifier sa description (`DESCRIPTION`).

Ecrire une commande `agenda2ics.sh` qui à partir du fichier `.agenda` du répertoire de login fabrique un fichier `.agenda.ics` dans le même répertoire contenant la liste des mêmes événements, mais cette fois au format `iCalendar`.

On pourra visualiser le résultat sous forme graphique en utilisant le logiciel `Sunbird`[1].

4 Pour aller plus loin

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

Faites en sorte que lorsque vous ouvrez un nouveau terminal, il vous affiche le prochain événement de la journée.

Exercice 9 – *Jours*

1. Écrivez un exécutable `jours.sh` qui demande à l'utilisateur une date et lui affiche le jour de la semaine (*i.e.*, lundi, mardi,...) à cette date.
2. Écrivez un exécutable `calendrier.sh` qui demande à l'utilisateur un mois `mm` et une année `aaaa` et lui renvoie un calendrier (comme si on tapait `cal mm aaaa`). *(Il est évidement hors de question d'utiliser* `cal` *dans votre exécutable.)*
3. Sauriez-vous programmer les commandes `date2jours` et `jours2date` ?

[1] De nombreux logiciels acceptent ce format, sur de nombreux systèmes d'exploitation.