SHELL : PROJET SUR UN LOGICIEL D'AGENDA

13 mai 2020 — Marc Champesme, Rado Rakotonarivo et Pierre Rousselin

— Date limite : **1er juillet 2020 à 23h59**.
— Les enseignants du cours se réservent le droit :
  — **d'utiliser un logiciel pour détecter le plagiat**
  — **d'organiser des oraux à distance pour vérifier qu'un travail est personnel**

1 Sujet

Il s'agit d'écrire un script shell `agda` qui permet **au minimum :**

— d'enregistrer des événements : rendez-vous, dates d'anniversaires, dates limites, heures de cours, ..., associés à des *dates* (et éventuellement des *horaires*) ;
— d'afficher, de façon pratique certains de ces rendez-vous : par exemple, ceux de la semaine, de la journée, du mois, ou bien ceux qui contiennent la chaîne « médecin », ... ;
— de modifier ou supprimer un événement (à cause d'une faute de frappe de l'utilisateur ou d'un changement de date par exemple) ;
— le tout en utilisant un fichier de données propre à l'utilisateur *au format* `csv` (voir le sujet « informaticiens »), par exemple `~/.agda_data`.

Il est clair que le fichier de données doit contenir au minimum :

— un champ pour le nom de l'événement (par exemple « Rdv Ophtalmo ») ;
— un champ pour la date de l'événement, par exemple au format `jj/mm/aaaa` ;
— un champ optionnel pour l'horaire de l'événement, par exemple au format `hh:mm`.

D'autres champs sont possibles (et même souhaitables !), voir par exemple le format `csv` permettant d'importer un agenda dans `googleagenda` :
`https://support.google.com/calendar/answer/37118?co=GENIE.Platform%3DDesktop&hl=fr`

2 Exemple très partiel

Pour vous guider, voici un exemple très partiel de **ce qu'on peut faire. Vous n'êtes pas du tout obligés de le suivre, au contraire !**

J'ai suivi le format `csv` mentionné ci-dessus. Les premières lignes de mon fichier de données sont les suivantes (la première est l'en-tête) :

```
Subject,Start Date,Start Time,End Date,End Time,All Day Event,Description,Location,Private
Anniversaire Alice,25/06/2019,,,,True,,,True
Cours shell,13/05/2020,13:45,,17:00,False,,,False
Ophtalmo,18/05/2020,10:30,,11:30,,,,True
Coding challenge,20/05/2020,08:00,,11:00,,,,
Faire cuire le riz,20/05/2020,11:03,,,,,,
Déjeuner avec Daphnée,20/05/2020,12:00,,,,Salade de riz,,
rdv Charlie,20/05/2020,20:00,,,False,Acheter des gâteaux,Chez Bob,
Anniversaire Alice,25/06/2020,,,,True,,,True
Rendre projet shell,01/07/2020,23:59,,,,sur le dépot ENT,,True
```

Si je lance la commande `./agda` sans argument ou avec l'argument `-s`, tous les événements qui ne sont pas passés sont affichés.

```
$ ./agda
Événement : Cours shell
Début : le 13/05/2020 à 13:45
Fin : le 13/05/2020 à 17:00
----- * ------
Événement : Ophtalmo
Début : le 18/05/2020 à 10:30
Fin : le 18/05/2020 à 11:30
----- * ------
Événement : Coding challenge
Début : le 20/05/2020 à 08:00
Fin : le 20/05/2020 à 11:00
----- * ------
Événement : Faire cuire le riz
Début : le 20/05/2020 à 11:03
Fin : le 20/05/2020
----- * ------
Événement : Déjeuner avec Daphnée
Début : le 20/05/2020 à 12:00
Fin : le 20/05/2020
Remarques : Salade de riz
----- * ------
Événement : rdv Charlie
Début : le 20/05/2020 à 20:00
Fin : le 20/05/2020
Remarques : Acheter des gâteaux
Lieu : Chez Bob
----- * ------
Événement : Anniversaire Alice
Début : le 25/06/2020
Fin : le 25/06/2020
----- * ------
Événement : Rendre projet shell
Début : le 01/07/2020 à 23:59
Fin : le 01/07/2020
Remarques : sur le dépot ENT
----- * ------
```

Pour le moment, j'ai choisi d'ignorer les champs « *All Day Event* » et « *Private* ». De plus, pour ne pas polluer l'affichage, je n'ai pas affiché les champs vides. Enfin, j'ai décidé (ce n'est pas forcément pertinent !), que si la date de fin n'était pas renseignée, elle était égale à celle de début. En revanche, il est clair qu'il faut que *les événements soient triés par date et heure croissants.*

Avec l'argument `-a`, je rentre un nouvel événement :

```
./agda -a
Événement : Réunion très importante
date de début (jj/mm/yyyy) : 19/05/2020
heure de début (hh:mm) : 14:00
date de fin (jj/mm/yyyy) :
heure de fin (hh:mm) : 16:00
Remarques : très important !
Lieu : chez moi
```

J'ai laissé un champ vide (la date de fin).
Je peux afficher cette nouvelle entrée, par exemple avec une recherche.

```
./agda --search 'Réunion'
Événement : Réunion très importante
Début : le 19/05/2020 à 14:00
Fin : le 19/05/2020
Remarques : très important !
Lieu : chez moi
----- * ------
Événement : Vacances à la Réunion
Début : le 01/08/2025
Fin : le 31/08/2025
Remarques : On peut toujours rêver...
----- * ------
```

3 Consignes de rendu et conseils

3.1 Fichiers à rendre sur l'ENT

Vous rendrez sur un dépôt de l'ENT :

— un fichier qui contiendra votre script ;

— un fichier de données au format `csv` pour votre agenda permettant de tester l'ensemble de vos fonctionnalités ;

— un compte-rendu de quelques pages (voir description plus loin).

3.2 Premières lignes du script

Les premières lignes de votre script *devront* être (bien sûr en remplaçant `NOM`, `Prénom` et `numéro_étudiant` par les valeurs appropriées) :

```
#!/bin/sh
# NOM Prénom numéro_étudiant
# Je déclare qu'il s'agit de mon propre travail.
```

3.3 Consignes et conseils concernant l'écriture du script

— Le plus important est le programme lui-même **dans ce qu'il apporte à l'utilisateur**. **Pensez toujours à l'utilisateur (qui peut être vous dans un an !)** : quelle fonctionnalité lui servirait ? Veut-il un menu ? Faut-il vérifier qu'il entre une date correcte, une

heure correcte ? Vous devrez en particulier écrire une option `-h` et/ou `--help` pour donner un mode d'emploi bref de votre programme (le compte-rendu contiendra une description plus longue), vérifier que ses arguments sont conformes à la syntaxe du programme, écrire des messages d'erreur clairs, ...

— Il est primordial que votre script soit *propre et lisible*. Le plus important est l'indentation (décaler le code dans les blocs) qui doit être parfaite et cohérente. Donnez-vous une règle simple et suivez-là. Vous pouvez vous inspirer des très nombreux exemples du cours.

Les lignes du script ne devraient pas être trop longues (se limiter à environ 80 caractères), vous pouvez les « découper » en utilisant \ suivi d'une fin de ligne.

Quelques commentaires, **écrits de façon correcte et concise (en français ou en anglais)**, lorsqu'ils sont pertinents sont bienvenus (mais pitié, pas *trop de commentaires non plus* !).

— Vous devrez un minimum découper votre code en fonctions shell avec quelques lignes de commentaires les spécifiant (voir exemples du cours ou des exercices).

— Dans la mesure du possible, élevez le niveau du code à celui du cours et **montrez que vous êtes un grand maître du shell** : utilisez les constructions, les développements, les redirections, les tubes, les commandes standards, ...

— Le cours a présenté le shell et les utilitaires selon le standard `POSIX`. Il faudrait que vous le respectiez également dans la mesure du possible, surtout pour ce qui est du langage shell. Essayez donc d'éviter tout ce qui est propre à `bash` ou à un autre shell et priviliégiez ce qui a été vu en cours. En cas d'écarts trop importants, vous risquez d'être pénalisé. Si un utilitaire non standard vous permet d'ajouter une fonctionnalité vraiment intéressante, vérifiez qu'il est très largement disponible et utilisez-le, en précisant qu'il est non standard et en le listant comme dépendance dans votre compte-rendu. En cas de doute, contactez l'un de vos enseignants.

— Vous pouvez vous entraider, partager des idées de fonctionnalités, échanger sur les utilitaires utilisés, ... , mais vous ne devez **jamais vous montrer tout ou partie du code du projet** avant la date limite (après, par contre, vous êtes encouragés à crâner en montrant votre joli script !).

— Enfin, et c'est le plus important, **amusez-vous, plongez-vous totalement dans ce projet et rendez un travail dont vous êtes fier.**

3.4 Compte-rendu

Votre compte-rendu de quelques pages doit être rendu **au format pdf**. Il doit être écrit dans un **français correct et concis** suivant le plan suivant :

1. Présentation générale : fonctionnalités principales, vos réussites et vos échecs...

   Les fonctionnalités minimales (cf. partie 1 du sujet du projet) ont elles été toutes implémentées ? Y a t'il des spécificités notables par rapport au sujet ?

   Fonctionnalités supplémentaires.

2. Description du format de fichier utilisé :

   — Signification de chaque champ : est-il obligatoire ? Quel est le format/codage (notamment dates et heures) ?

- Comment les champs sont interprétés par le script : par exemple, si l'heure de début n'est pas renseignée, où ce rendez-vous est-il placé lors d'un tri ? Y a t'il une valeur par défaut pour un champ non fourni ? Si oui, laquelle ?
- Donner un exemple de fichier de données sur environ 5 lignes (un fichier complet et permettant de tester l'ensemble des fonctionnalités doit être déposé sur l'ENT).

3. Description détaillée de chaque fonctionnalité

   Chaque fonctionnalité doit être présentée en supposant que l'utilisateur n'a pas accès au code source du script (voir une page de manuel). Autant que possible le fonctionnement de chaque commande doit être illustrée par un ou plusieurs exemples d'utilisation. Préciser pour chaque commande si elle modifie ou non le fichier.

   a) Options de la ligne de commande :

      Description précise de ce que fait (ou pas) chaque option, quelles commandes sont optionnelles, certaines options sont elles exclusives les unes des autres ou complémentaires, réaction du programme en cas d'erreur (message d'erreur sur l'erreur standard, code de retour, ...)

   b) Le cas échéant, fonctionnalités accessibles après le lancement du script (par exemple depuis un menu, ...)

      Quels vérifications sont faites sur les entrées des utilisateurs ? Quelle est la réaction du programme en cas d'entrée erronée (demande d'une nouvelle saisie ou sortie avec message d'erreur) ?

4. Description du code

   a) Description de chaque fonction et de son rôle pour exécuter les différentes fonctionnalités présentées à la partie précédente (ex : la fonction add est utilisée pour traiter l'option -a), si la fonction utilise ses paramètres positionnels, lit sur l'entrée standard et/ou le fichier, écrit sur la sortie standard et/ou le fichier, écrit sur l'erreur standard, ...

   b) Des spécificités non POSIX ont-elles été utilisées ? Pourquoi ?

   c) Si besoin, précisez les *dépendances* non standard (c'est-à-dire les utilitaires non standard sur lesquels s'appuie votre script).

   d) Description précise du fonctionnement d'un ou deux passages du programme qui vous paraissent intéressants et/ou compliqués.

5. Environnement de test et de développement

   - Shell avec lequel les tests ont été faits (`bash`, `dash`, ... ?).
   - Système d'exploitation (exemple MacOS, GNU/Linux avec distribution Debian « Buster », ...)
   - Éditeur de texte et éventuellement autres outils utilisés pendant le développement.

6. Bugs et TODO

   Le cas échéant, dresser une liste de *bugs* que vous n'avez pas réussi à corriger. Si possible décrire comment *reproduire* chaque *bug* et la façon dont vous pensez pouvoir le corriger.

   Vous pouvez aussi dresser une liste *TODO*, (c'est-à-dire « de choses à faire ») qui décrit les améliorations et/ou fonctionnalités supplémentaires à traiter dans le futur.

7. Sources de documentation et remerciements

   Si vous avez utilisé d'autres ressources que le cours et les pages de manuel, les lister ici.

   Si une personne (par exemple un autre étudiant) vous a aidé d'une façon ou d'une autre, le remercier en indiquant la nature de l'aide. Par exemple :

   Remerciements : Ma cousine Berthe m'a aidé en relisant le présent compte-rendu et en corrigeant mes fautes d'orthographe. Mon camarade Roger m'a conseillé d'utiliser la commande `sed` avec l'action `d` pour pouvoir modifier facilement le fichier. Ma camarade Joséphine m'a expliqué les expressions rationnelles. Alice et Bob ont servi de cobayes pour le programme.

8. Facultatif : Autres commentaires

   Quartier libre.

4 Quelques idées

Voici une liste de choses que vous pouvez (ou pas !) faire dans votre script. Bien sûr vous êtes vivement encouragés à en trouver d'autres !

— Validation des données : vérifier qu'une date est valide, une heure est valide, un texte ne contient pas de virgule.

— Répétition des événements : proposer de répéter (toutes les semaines, mois, années) un événement, par exemple un anniversaire.

— Importer un autre fichier de données (le fusionner avec le fichier principal).

— Valider un fichier de données (vérifier qu'il a le bon nombre de champs et que ceux-ci sont valides).

— ...