KRIVOKUCA Ivan 11807642 (Groupe 5/SH1)

Compte rendu du projet : Agenda

Présentation du projet :

Le script Shell agda permet d'enregistrer des événements qui sont forcément accompagné d'une date et d'une heure, on doit renseigner si cet événement est seulement pour cette journée ou dans le cas contraire s'il a une date de fin et une heure de fin. On peut aussi ajouter des éléments facultatifs tel qu'une description, un lieu ou bien le fait que cet événement est privé. Ensuite cet événement sera enregistré dans un fichier qui a pour nom `agda_data.csv`. A l'aide des différentes commandes on peut choisir de rechercher un événement qui a un nom précis, on peut aussi choisir de supprimer un événement. Pour finir des options sont disponible pour trier les affichages, par exemple afficher les événements qui sont privés. Par défaut l'affichage se fait par ordre croissant en fonction de la date.

```
Événement : Rendre le projet Shell
Debut : Le 01/07/2020 à 23:45
Fin : Le 01/07/2020 à 23:45
Remarques: Joindre les fichiers
```

Affichage d'un événement dans le terminal

Les événements sont enregistrés dans un format .csv dont chaque ligne est constituée de ces éléments :

```
Evenement,Debut_date,Debut_heure,Fin_date,Fin_heure,Confirm_jour,Description,Lieu Prive
```

```
Préparer la fête pour Alice,15/07/2020,10:00,15/07/2020,23:30,n,Préparer la diapo + la playlist,Chez Alice,TRUE,
```

```
Acheter un cadeau d'anniversaire,05/12/2020,12:30,,,o,Acheter The Last Of Us 2,Centre comericale,,
```

Seuls les éléments `Evenement,Debut_date,Debut_heure` sont obligatoires, pour `confirm_jour` seule le résultat « o » aura une incidence pour l'affichage et c'est aussi le cas pour `Prive` « TRUE ».

Les dates doivent être de format jj/mm/aaaa et les heures de format hh:mm.

Description de chaque fonctionnalité :

Voici la liste de chaque argument qui sont utilisable dans le script :

```
-a, --add)
```

Cette fonctionnalité permet d'ajouter un événement dans le fichier en respectant le format du fichier (qui a été cité ci-dessus). Pour vérifier si l'événement est correct plusieurs tests sont effectués comme le fait que l'utilisateur doit rentrer une date qui

existe dans le calendrier pour qu'elle soit prise en compte, que l'heure doit être correcte et aussi qu'aucune virgule n'est rentrée par l'utilisateur pour créer un désordre lors de la lecture du fichier.

```
-s, --search)      nécessite un deuxième argument
```

Cette fonctionnalité demande un deuxième argument qui sera une chaine de caractère (ou bien un chiffre) qui va ensuite chercher ce résultat dans le fichier et puis l'afficher à l'aide de la fonction `affichage`.

```
-d, --delete)      nécessite un deuxième argument
```

Deux cas sont possibles :

Si l'utilisateur met un nombre en deuxième argument alors celui-ci va être interprété en tant qu'un numéro de ligne et celle-ci sera supprimé dans le fichier.

Si l'utilisateur met une chaîne de caractère alors le script cherchera cette chaîne dans le fichier, si cette chaîne correspond à un événement exisitant alors l'affichage de cette la ligne sera précisé, il pourra ainsi la supprimer.

```
-l, --live)
```

Cet argument permet d'afficher tous les événements dont la date de début est supérieure à la date de l'ordinateur.

```
-p)
```

Permet d'afficher tous les événements y compris ceux qui sont privés sans modification du fichier.

```
-h, --help
```

Affiche une aide résumant les fonctionnalités du script

Si l'utilisateur entre une commande non reconnu alors un message d'erreur sera affiché.

Description du code

J'ai dû implémenter plusieurs fonctions qui sont utiliser par les fonctions définis dans des paramètres.

Tous les messages d'erreurs sont envoyés dans le canal d'erreur >&2

`est_date()` vérifie à l'aide de code de retour si le paramètre entré est bien une date valide ainsi le code de retour sera égale à 0 sinon ça retourne 1. Pour cela je commence par voir la longueur de la chaîne si celle-ci est inférieur à 10 *(taille de jj/mm/yyyy)* alors on retourne 0. Ensuite on vérifie si c'est bien une date qui existe dans le calendrier grâce a une substitution de commande, en effet on extrait le jour le mois et l'année et on vérifie à l'aide de la fonction `date -d` si cela existe bien *(la fonction date -d prend comme argument sous la forme YYYY-MM-DD).*

```
Date du début de l'événement (jj/mm/yyyy) : 31/02/2020
Date invalide (format jj/mm/yyyy),veuillez réessayer!
```

`est_date_passer()` vérifie à l'aide de code de retour si le premier paramètre qui est une date est bien antérieur au deuxième paramètre (aussi une date), si tel est le cas code de retour = 0 sinon ça retourne 1. J'utilise de nouveau `date -d` mais cette fois ci avec une différence en effet je l'utilise avec `'+%Y%m%d'`, cela permet de transformer la date en un seule nombre. Par exemple 12/10/2000 devient 20001012 et cela facilite énormément la comparaison. Du coup ayant mes deux dates sous un format avantageux j'ai juste a comparer les deux nombre et retourne le code de retour.

```
Date du fin de l'événement (jj/mm/yyyy) : 01/11/1988
Date de fin incompatible avec la date de début,veuillez réessayer !
(Rappel : Date de début 31/12/1999 )
```

`est_heure()` même principe que `est_date` sans l'utilisation de `date -d` on vérifie juste si l'heure n'est pas supérieur a 24:60

`pas_virgule()` on vérifie si il n'y a pas de virgule dans le paramètre et on retourne 0 si tel est le cas sinon on retourne 1.

`affichage()` On parcourt chaque ligne du fichier `agda_data.csv` et on affiche l'événement correspondant. Si celui-ci possède son champ « privé » qui est égale a « TRUE » alors on montre que cet événement est privé. Si l'événement possède le caractère « o » pour « confirm_jour » alors on affiche que c'est un événement pour la journée sinon on affiche la date de fin et l'heure de fin. Pour « description » et « lieu » c'est le même procédé on affiche si la chaîne dans le fichier n'est pas vide.

`en_cours()` tout d'abord on récupère la date de la machine à l'aide de `DATE=`date '+%Y-%m-%d'`et` on la transforme comme pour `est_date_passer` ensuite on la compare avec « date_début ». Si « date_debut » est après alors on affiche l'événement.

La fonction `date -d et` de son autre utilisation `$(date -d "$DATE" +"%Y%m%d")` ne sont pas standard et peuvent ne pas marcher sur tous les systèmes UNIX, j'ai cru comprendre que sous MacOS (+"%Y%m%d") ne marche pas.

Environnement de test

Le Shell utilisé est Bash/sh. J'ai effectué ce script sous Windows 10 avec la distribution Ubuntu (WSL) et Visual Studio Code comme éditeur de texte.

TODO

L'optimisation de code est surement la chose que je ferrai en première, en effet je trouve que l'utilisation des substituions de commande avec date (surtout pour

`est_date_passer`) peuvent être ardu, dommage qu'en France le format de date n'est pas YYYY-MM-JJ car m'aurai fait gagner du temps et simplifier le code….

Un petit souci c'est que lorsque j'ai deux événements de même date mais pas de même heure alors le tri ne prend pas en compte l'heure, il faudrait que je fasse un sort pour la colonne n°3.

Remerciement + documentation

Grâce a ces liens j'ai pu mener mon projet à bien.

Pour la couleur :
https://misc.flogisoft.com/bash/tip_colors_and_formatting#terminals_compatibility

Pour le triage :https://stackoverflow.com/questions/39151834/sorting-by-date-in-shell

Pour `date_passer`: https://unix.stackexchange.com/questions/84381/how-to-compare-two-dates-in-a-shell

Pour `est_date`: https://stackoverflow.com/questions/10759162/check-if-argument-is-a-valid-date-in-bash-shell Merci de m'avoir lu jusqu'au bout

Merci à Pierre Rousselin d'avoir répondu à mon interrogation à propos de la fonction date et surtout à la personne qui va devoir lire mon projet plus ce compte rendu ! 😊