TD noté

Étapes de réalisation :

1. **Planification :**
   - Répartir les tâches entre les membres du binôme.
   - Planifier des réunions régulières pour discuter de la progression du travail.
2. **Création du script Shell:**
   - Commencer par créer un fichier script bash (pret.sh).
   - Inclure les commandes initiales mentionnées dans le TD au début du script.
3. **Développement des Fonctions :**
   - Développer des fonctions pour chaque commande (init, add, lend, retrieve, list).
   - Chaque fonction doit être bien commentée et documentée.
4. **Test et Debug:**
   - Tester chaque fonction individuellement pour s'assurer qu'elle fonctionne comme prévu.
   - Corriger les erreurs et bugs rencontrés lors des tests.
5. **Validation avec ShellCheck:**
   - Passer le script dans ShellCheck et corriger les erreurs jusqu'à ce qu'il n'y en ait plus.
6. **Rapport PDF:**
   - Rédiger un rapport incluant toutes les sections demandées dans le TD.
7. **Soumission:**
   - Créer un fichier .tar.gz contenant le script et le rapport PDF.
   - Soumettre le fichier sur la plateforme d'enseignement avant la date limite.

Algorithme:

```
#!/bin/bash
set -o errexit # Exit if command failed.
set -o pipefail # Exit if pipe failed.
set -o nounset # Exit if variable not set.
IFS=$'\n\t'
# Fonction pour initialiser la base de données
function init() {  # Code}
# Fonction pour ajouter un nouvel article
function add() {  # Code}
# Fonction pour prêter un article
function lend{  # Code}
# Fonction pour récupérer un article prêté
```

function retrieve() { # Code}

Fonction pour lister les articles/prêts

function list() { # Code}

Code principal pour appeler les fonctions basées sur l'input utilisateur

Code

Fonctions à Développer :

- init:
  - Vérifie si le fichier pret.json existe déjà.
  - Si oui, demande à l'utilisateur s'il veut le supprimer.
  - Initialise une nouvelle base de données vide.
- add (CODE, DESCRIPTION):
  - Ajoute un nouvel article à la base de données.
  - Vérifie si l'article existe déjà, si oui, affiche une erreur.
- lend (CODE, WHO):
  - Marque un article comme prêté dans la base de données.
  - Vérifie si l'article existe et n'est pas déjà prêté.
- retrieve (CODE):
  - Supprime un article de la liste des prêts.
  - Vérifie si l'article est actuellement prêté, sinon affiche une erreur.
- list (items|lends):
  - Affiche une liste formatée des articles ou des articles prêtés basé sur le paramètre.
  - Utilise un PAGER pour afficher les informations.

Pour exécuter le script, voici les étapes à suivre :

1. **Donner les droits d'exécution au script :** Ouvrez un terminal et naviguez vers le dossier où se trouve le script. Ensuite, donnez-lui les droits d'exécution avec la commande suivante :

   chmod +x pret.sh

2. **Initialiser le fichier JSON de données :** Exécutez le script avec la commande init pour initialiser la base de données.

   ./pret.sh init

3. **Ajouter un article :** Pour ajouter un article, utilisez la commande add suivie du code de l'article et de sa description.

   ./pret.sh add "CODE1" "Description de l'article 1"

4. **Prêter un article :** Pour prêter un article, utilisez la commande lend suivie du code de l'article et du nom de la personne à qui l'article est prêté.

./pret.sh lend "CODE1" "Nom de la personne"

5. **Récupérer un article :** Pour récupérer un article, utilisez la commande retrieve suivie du code de l'article.

./pret.sh retrieve "CODE1"

6. **Lister les articles ou les prêts :** Pour lister tous les articles ou seulement les articles prêtés, utilisez la commande list.

./pret.sh list items

Pour lister seulement les articles prêtés :

./pret.sh list lends

N'oubliez pas que le script doit être exécuté dans un environnement où jq est installé, car il est utilisé pour manipuler des données JSON. Si ce n'est pas déjà fait, vous pouvez installer jq avec la commande suivante (sur un système basé sur Debian/Ubuntu) :

sudo apt-get install jq

Assurez-vous également que le chemin du fichier JSON (variable DATA_FILE dans le script) est correct et que le script a les permissions nécessaires pour lire et écrire dans ce fichier et dans le dossier où il se trouve.

Script de configuration d'erreur et d'initialisation

```
#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset
IFS=$'\n\t'

DATA_FILE="pret.json"

function init() {
    if [[ -f $DATA_FILE ]]; then
        echo "Data already exist. Are you sure to want to delete it? (y/n)"
        read -r answer
        if [[ $answer != "y" ]]; then
            exit 1
        fi
    fi
    echo '{ "items": [], "lends": [] }' > "$DATA_FILE"
}
```

1. #!/bin/bash

- C'est la ligne shebang. Elle indique au système d'exploitation que ce fichier doit être exécuté en tant que script bash.

2. set -o errexit, set -o pipefail, set -o nounset

- Ces lignes configurent le comportement du script en cas d'erreur.

- set -o errexit : Cela signifie que le script sortira immédiatement si une commande échoue.
- set -o pipefail : Cela signifie que le script sortira avec une erreur si un pipeline échoue. Un pipeline est une séquence de commandes séparées par |.
- set -o nounset : Cela signifie que le script sortira avec une erreur si une variable non définie est utilisée.

3. IFS=$'\n\t'

➢ IFS (Internal Field Separator) est utilisé par le shell pour déterminer comment diviser une ligne de texte en mots. Cela est configuré pour reconnaître les nouvelles lignes et les tabulations comme séparateurs de champ, ce qui est utile pour la manipulation des données.

4. DATA_FILE="pret.json"

➢ Ceci définit une variable DATA_FILE qui contient le nom du fichier JSON utilisé pour stocker les données des articles et des prêts.

5. function init() { ... }

➢ Cette partie définit une fonction nommée init qui initialise (ou réinitialise) le fichier de données.

- if [[-f $DATA_FILE]]; then : Ceci vérifie si le fichier de données existe déjà.
- echo "Data already exist. Are you sure to want to delete it? (y/n)" : Si le fichier existe, un message est affiché demandant à l'utilisateur s'il souhaite supprimer les données existantes.
- read -r answer : Ceci lit la réponse de l'utilisateur et la stocke dans la variable answer.
- if [[$answer != "y"]]; then exit 1; fi : Si la réponse n'est pas "y" (oui), alors le script est arrêté avec une erreur.
- echo '{ "items": [], "lends": [] }' > "$DATA_FILE" : Si la réponse est "y", le fichier de données est initialisé avec un objet JSON contenant deux tableaux vides, items et lends.

En résumé, cette partie du script configure certains comportements en cas d'erreur, initialise quelques variables et définit une fonction pour initialiser le fichier de données JSON.

Fonction add

```
20 function add() {
21     CODE="$1"
22     DESCRIPTION="$2"
23 
24     if jq -e --arg code "$CODE" '.items[] | select(.code == $code)' "$DATA_FILE" > /dev/null; then
25         echo "[ERROR] $CODE already exists and cannot be added" >&2
26         exit 1
27     fi
28 
29     jq --arg code "$CODE" --arg description "$DESCRIPTION" '.items += [{"code": $code, "description": $description}]' "$DATA_FILE" >
   "$DATA_FILE".tmp && mv "$DATA_FILE".tmp "$DATA_FILE"
30 }
```

Cette fonction add est définie dans le script bash pour ajouter un nouvel article dans la base de données (fichier JSON). Voici une explication détaillée de chaque ligne de cette fonction :

1. CODE="$1" et DESCRIPTION="$2"

- Ces lignes assignent les valeurs des deux premiers arguments passés à la fonction à des variables. $1 est le premier argument et $2 est le deuxième.
    - CODE="$1" : Assigner le premier argument (code de l'article) à la variable CODE.
    - DESCRIPTION="$2" : Assigner le deuxième argument (description de l'article) à la variable DESCRIPTION.

2. if jq -e --arg code "$CODE" '.items[] | select(.code == $code)' "$DATA_FILE" > /dev/null; then

Cette ligne utilise la commande jq pour rechercher un article avec le même code dans le fichier de données JSON.

jq -e --arg code "$CODE" : jq est utilisé pour manipuler des données JSON. L'option -e fait que jq retournera une valeur de sortie false si le filtre ne retourne pas de résultats. --arg code "$CODE" définit une variable jq appelée code avec la valeur de la variable shell CODE.

'.items[] | select(.code == $code)' : Ceci est un filtre jq. Il sélectionne tous les objets dans le tableau .items où la valeur de code est égale à la valeur de la variable jq code.

"$DATA_FILE" > /dev/null : Ceci spécifie le fichier de données à utiliser et redirige la sortie vers /dev/null, ce qui signifie que la sortie ne sera pas affichée à l'écran.

3. echo "[ERROR] $CODE already exists and cannot be added" >&2

Si la commande jq précédente trouve un élément avec le même code, un message d'erreur est affiché indiquant que l'article existe déjà. >&2 signifie que la sortie sera affichée sur la sortie d'erreur standard (stderr).

4. exit 1

Si un article avec le même code est trouvé, la fonction quitte et retourne une valeur de 1, indiquant une erreur.

5. jq --arg code "$CODE" --arg description "$DESCRIPTION" '.items += [{"code": $code, "description": $description}]' "$DATA_FILE" > "$DATA_FILE".tmp && mv "$DATA_FILE".tmp "$DATA_FILE"

Si aucun article existant avec le même code n'est trouvé, un nouvel article est ajouté au fichier de données JSON.

- --arg code "$CODE" --arg description "$DESCRIPTION" : Ceci définit deux variables jq avec les valeurs des variables shell CODE et DESCRIPTION.
- '.items += [{"code": $code, "description": $description}]' : Ceci est un filtre jq qui ajoute un nouvel objet avec les valeurs de code et description au tableau .items.
- " $DATA_FILE" > "$DATA_FILE".tmp : Ceci spécifie le fichier de données à utiliser et écrit la sortie modifiée dans un fichier temporaire.
- && mv "$DATA_FILE".tmp "$DATA_FILE" : Si la commande jq est réussie, le fichier temporaire est renommé pour remplacer le fichier de données original.

En résumé, cette fonction add ajoute un nouvel article au fichier de données JSON, à moins qu'un article avec le même code n'existe déjà.

```
73 if [[ $# -lt 1 ]]; then
74     echo "[ERROR] one command is expected." >&2
75     echo "Usage: ./pret.sh COMMAND [PARAMETER]..." >&2
76     exit 1
77 fi
78
79 COMMAND="$1"
80 shift
81
82 case "$COMMAND" in
83     init) init ;;
84     add) add "$@" ;;
85     lend) lend "$@" ;;
86     retrieve) retrieve "$@" ;;
87     list) list "$@" ;;
88     *) echo "[ERROR] Unknown command: $COMMAND" >&2; exit 1 ;;
89 esac
```