



Informatique

iut  
de BORDEAUX

# Rapport TD noté

R1.04 - Introduction aux systèmes  
d'exploitation - 2023-2024

03/12/2023

# Sommaire

<b>Introduction.....</b>	<b>1</b>
<b>I. Fonctionnement du script.....</b>	<b>2</b>
1.1. Fonctionnement global.....	2
1.2. Syntaxe d'appel.....	2
<b>II. Travail réalisé.....</b>	<b>2</b>
2.1. Fonctionnement des fonctions.....	2
2.2. Difficultés rencontrées.....	3
<b>III. Améliorations réalisées.....</b>	<b>4</b>
3.1. Préserver les espacements.....	4
3.2. Déclarer et assigner séparément.....	5
3.3. Affichage de la donnée des tableaux ITEM et LENT.....	5
<b>IV. Traces d'exécution.....</b>	<b>6</b>
<b>Conclusion.....</b>	<b>7</b>

## Introduction

Notre objectif est de développer un script pour la gestion des prêts de documents en utilisant le format de données JSON pour stocker les informations sur les documents et les prêts ainsi que "jq" pour manipuler le JSON à partir du script.

Dans un premier temps, nous allons expliquer le fonctionnement du script en détaillant les fonctions générales qui seront créées pour répondre aux besoins, ainsi que la syntaxe d'appel qui déterminera la manière dont chaque fonction du script sera invoquée dans le terminal. Ensuite, nous présenterons le processus de mise en place du script, en mettant en lumière les éventuelles difficultés rencontrées pendant sa réalisation. Dans un troisième temps, nous exposerons les améliorations apportées grâce à l'utilisation de shell-check, contribuant ainsi à améliorer la qualité globale du script. Enfin, nous illustrerons les traces d'exécution du script une fois terminé, détaillant chaque cas d'utilisation possible.

# I. Fonctionnement du script

## 1.1. Fonctionnement global

Le fichier `pret.sh` est un script bash dédié à la gestion de prêts de documents d'une bibliothèque: il permet d'en ajouter et d'en emprunter mais également de supprimer de la liste d'emprunts, un emprunt lors d'un retour du document et de lister les informations présentes dans les différentes listes (Liste de documents disponible / liste des prêts). Cette bibliothèque est stockée dans un fichier json, que l'on modifie à l'aide de jq. Le script prend des paramètres en entrée, le premier étant une commande que l'on souhaite réaliser (ajouter : `add` | supprimer : `retrieve`), les autres paramètres étant ajoutés en fonction de la commande utilisée. Dans le cas où les paramètres seraient incorrects en nombre, le script retourne donc une erreur et précise la façon dont doit être utilisé le script avec le nombre de paramètres à utiliser.

## 1.2. Syntaxe d'appel

Nous allons maintenant vous expliquer comment vous servir correctement de ce script.

Tout d'abord, il faut avoir les droits d'exécution, qu'on obtient en écrivant la commande : « `chmod +x pret.sh` ». Ensuite, il faut créer (initialiser) le fichier JSON vide, qui représente la bibliothèque de documents : « `./pret.sh init` ». Une fois cela fait, la bibliothèque est prête à être utilisée.

- Pour ajouter un document, vous devrez écrire : « `./pret.sh add "codeDocument" "description de l'article"` ». La commande `add` ajoute le document ayant un code et une description, à écrire dans les espaces dédiés ("`codeDocument`" et "`description de l'article`").
- Pour réaliser un emprunt, vous devrez écrire : « `./pret.sh lend "codeDocument" "nom de l'emprunteur"` ». La commande `lend` ajoute toutes les informations du document dans le tableau `lent`, avec le nom de l'emprunteur, à écrire dans les espaces dédiés ("`codeDocument`" et "`nom de l'emprunteur`" et "`Date d'emprunt`").
- Pour récupérer un document, vous devrez écrire : « `./pret.sh retrieve "codeDocument"` ». La commande `retrieve` retire le document et toutes ses informations du tableau `lent`.
- Pour afficher la liste des documents, avec le PAGER de votre choix, vous devrez écrire : « `./pret.sh list items` » ou alors pour afficher la liste des documents empruntés, vous devrez écrire : « `./pret.sh list lends` » puis indiquer le PAGER que vous souhaitez (`less`, `more`).

# II. Travail réalisé

## 2.1. Fonctionnement des fonctions

Nous avons vu comment fonctionne le script `pret.sh`, nous allons maintenant voir les étapes de réalisation par lesquelles nous sommes passées avant d'arriver au résultat.

- La fonction `init` vérifie s'il existe déjà un fichier `pret.json`. S'il existe, il propose à l'utilisateur de le créer mais cela écraserait les données existantes. Sinon, il crée un nouveau fichier avec le JSON par défaut.

- La fonction `add` (suivie du code du document et de sa description, comme vue plus haut), vérifie si le document qu'on souhaite ajouter est déjà présent dans le fichier. S'il est présent, il affiche à l'utilisateur un message d'erreur, sinon, il l'ajoute.
- La fonction `lend` (suivie du code du document et du nom de la personne qui veut l'emprunter), vérifie que le document existe et n'est pas déjà prêté puis l'ajoute dans la liste des prêts. Si c'est le cas, le script copie les informations du document dans le tableau `lent`, sinon, il affiche un message d'erreur adapté.
- La fonction `retrieve` (suivie du code du document que l'on souhaite récupérer), vérifie si le document existe et s'il est prêté. Si c'est le cas, le document est retiré de la liste des prêts. Dans le cas contraire, il n'est pas retiré de la liste.
- La fonction `list` (suivie de « items » ou de « lends ») affiche une liste formatée des documents ou des documents prêtés en fonction des paramètres spécifiés (« items » ou de « lends »), à l'aide de `PAGER`, qui affiche les informations de façon lisible. Si aucun `PAGER` n'est défini par défaut (variable d'environnement), on propose à l'utilisateur de choisir entre `MORE` ou `LESS`.

## 2.2. Difficultés rencontrées

Nous allons vous présenter les difficultés rencontrées lors de la réalisation de ce TP noté.

La première difficulté que nous avons rencontrée est en lien avec la casse des chaînes de caractères utilisées dans le script, nous avons en premier lieu pensé à mettre en place une condition de vérification de la casse afin de convertir chaque chaîne de caractères minuscule. La commande `"tr"` répond à cet objectif cependant son utilisation nuire à la lisibilité du code donc nous avons cherché un autre moyen de convertir une chaîne sans utiliser cette commande. Nous avons trouvé ce que nous cherchions avec l'opérateur `"${chaîne,,}"` qui en utilisant des `","` après la chaîne de caractères permet de convertir chaque caractère en minuscule.

Secondement, nous avons eu du mal à appréhender le format de données qu'est JSON et encore plus lorsqu'il s'agit de l'utiliser en association avec `"jq"`. Nous avons donc pris le temps de tester afin de comprendre son fonctionnement ainsi que le fonctionnement général de la commande.

Troisièmement, lors de l'utilisation de `"jq"` pour l'ajout ou suppression d'éléments dans les tableaux `"ITEM"` et `"LENT"`, nous avons rencontré la difficulté suivante. Lors de l'ajout ou de la suppression d'un élément dans les différents tableaux de données avec `"jq"`, il n'est pas possible de modifier directement un fichier déjà existant et il nous aura donc fallu passer par un fichier temporaire permettant ainsi de sauvegarder le résultat de l'exécution de la commande et de par la suite le sauvegarder dans le fichier originel en écrasant les données présentes.

Quatrièmement, pour la mise en place de la fonction permettant de parcourir les tableaux `"ITEM"` et `"LENT"`, nous avons mis un moment à réussir à réaliser un parcours sur les

tableaux puis nous avons eu du mal à mettre en forme avec “jq” les objets JSON contenu dans les tableaux.

### III. Améliorations réalisées

Après passage du script dans le logiciel shell-check nous avons reçu une liste de recommandation à mettre en place pour améliorer la qualité de notre script.

Nous allons détailler les modifications réalisées pas à pas pour chacune des recommandations.

```
$ shellcheck myscript

Line 131:
  if test $TITLE = "item"
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean: (apply this, apply all SC2086)
  if test "$TITLE" = "item"

Line 133:
  declare -a item=$(jq --compact-output '.item[]' "$DATA_FILE")
    ^-- SC2155 (warning): Declare and assign separately to avoid masking return values.

Line 134:
  for i in ${item[@]}
    ^-- SC2068 (error): Double quote array expansions to avoid re-splitting elements.

Line 136:
  echo "Code : $(jq '.code' <<< $i) / Description : $(jq '.description' <<< $i)" >> $PRINT
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.
>>
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean: (apply this, apply all SC2086)
  echo "Code : $(jq '.code' <<< "$i") / Description : $(jq '.description' <<< "$i")" >> $PRINT

Line 138:
  elif test $TITLE = "lent"
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean: (apply this, apply all SC2086)
  elif test "$TITLE" = "lent"

Line 140:
  declare -a lent=$(jq --compact-output '.lent[]' "$DATA_FILE")
    ^-- SC2155 (warning): Declare and assign separately to avoid masking return values.

Line 141:
  for i in ${lent[@]}
    ^-- SC2068 (error): Double quote array expansions to avoid re-splitting elements.

Line 143:
  echo "Code : $(jq '.what' <<< $i) / To $(jq '.who' <<< $i) / $(jq '.when' <<< $i)" >> $PRINT
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.
>>
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean: (apply this, apply all SC2086)
  echo "Code : $(jq '.what' <<< "$i") / To $(jq '.who' <<< "$i") / $(jq '.when' <<< "$i")" >> $PRINT

Line 155:
  if [ $CHOOSSED -eq 1 ]
    ^-- SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean: (apply this, apply all SC2086)
  if [ "$CHOOSSED" -eq 1 ]

$
```

Figure 1: capture de shell-check

#### 3.1. Préserver les espacements

Shell-check nous donne comme information que pour les lignes 131,136,138,143,155 nous pouvons préserver les espacements qui pourraient contenir les variables utilisées en ajoutant autour d'elle des guillemets doubles par exemple : “\$VAR”.

## 3.2. Déclarer et assigner séparément

L'outil nous donne comme retour que nous avons dans notre script des variables qui sont déclarées et assignées directement. Shell-check nous met en garde via un "warning" en jaune nous disant qu'il serait plus judicieux de séparer les deux actions, cependant il n'y a pas que cela qui doit être pris en compte.

```
declare -a item=$(jq --compact-output '.item[]' "$DATA_FILE")
```

Figure 2: Avant amélioration

Nous avons pris la décision de modifier cette portion de code pour les raisons citées dans les difficultés rencontrées. Nous amenant ainsi à ce résultat :

```
item=$(jq --compact-output -r '.item[] | "Code : \"\(.code)\" / Description : \"\(.description)\""' "$DATA_FILE")
```

Figure 3: Après amélioration

Nous avons décidé de garder la déclaration et l'assignation sur une ligne car auparavant nous avons déclaré une variable en tant que tableaux or l'exécution de la commande "jq" nous retourne directement une sortie avec les données formatées de la façon souhaitées et il est possible de parcourir ces données via une boucle qui les ajoute dans un fichier texte créé au préalable. Le résultat de la modification nous permet de retirer les warnings de shell-check.

## 3.3. Affichage de la donnée des tableaux ITEM et LENT

Nous avons une dernière erreur à traiter afin que shell-check nous donne une approbation sur la qualité de notre script pour cela nous devons corriger un bout de code dans le script pour les différentes boucles for.

```
declare -a item=$(jq --compact-output '.item[]' "$DATA_FILE")
for i in ${item[@]}
do
    echo "Code : $(jq '.code' <<< $i) / Description : $(jq '.description' <<< $i)" >> $PRINT
done
```

Figure 4: Avant amélioration

Nous avons ajouté à l'intérieur des deux boucles for des doubles guillemets sur l'instruction suivant \${item[@]} permettant de s'assurer que chaque élément du tableau est traité comme une seule entité, même s'il contient des espaces ou d'autres caractères spéciaux.

Ainsi une fois combinée aux modifications apportées précédemment nous obtenons les résultats suivants :

```
item=$(jq --compact-output -r '.item[] | "Code : \"\(.code)\" / Description : \"\(.description)\""' "$DATA_FILE")
for i in "${item[@]}"
do
    echo "$i" >> "$PRINT"
done
```

Figure 5: Après amélioration

Une fois toutes les améliorations apportées au script nous relançons l'exécution du script pour s'assurer qu'il fonctionne toujours puis nous le soumettons à shell-check pour validation.



```
$ shellcheck myscript
No issues detected!

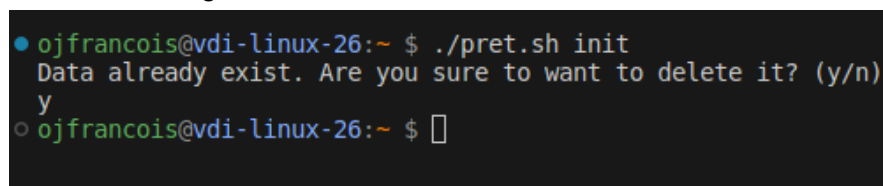
$
```

Figure 6: Approbation shell-check

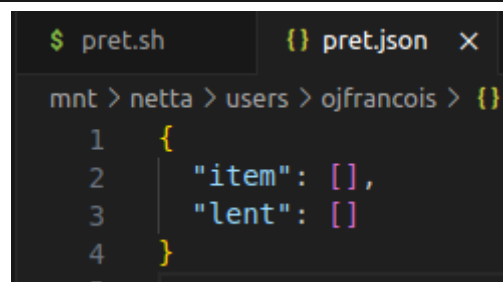
## IV. Traces d'exécution

Nous allons maintenant illustrer l'exécution du script via des captures d'écran.

Figure 7: Initialisation du fichier JSON via

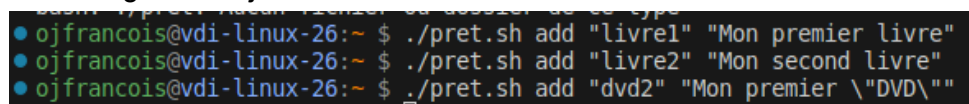


```
ojfrancois@vdi-linux-26:~ $ ./pret.sh init
Data already exist. Are you sure to want to delete it? (y/n)
y
ojfrancois@vdi-linux-26:~ $
```



```
$ pret.sh {} pret.json x
mnt > netta > users > ojfrancois > {}
1 {
2   "item": [],
3   "lent": []
4 }
```

Figure 8: Ajout de nouveaux éléments dans le tableau ITEM



```
ojfrancois@vdi-linux-26:~ $ ./pret.sh add "livre1" "Mon premier livre"
ojfrancois@vdi-linux-26:~ $ ./pret.sh add "livre2" "Mon second livre"
ojfrancois@vdi-linux-26:~ $ ./pret.sh add "dvd2" "Mon premier \DVD\""
```



```
$ pret.sh {} pret.json x
mnt > netta > users > ojfrancois > {} pret.json > ...
1 {
2   "item": [
3     {
4       "code": "livre1",
5       "description": "Mon premier livre"
6     },
7     {
8       "code": "livre2",
9       "description": "Mon second livre"
10    },
11    {
12      "code": "dvd2",
13      "description": "Mon premier \DVD\"
14    }
15  ],
16  "lent": []
17 }
```

Figure 9 : Ajout de nouveaux éléments dans le tableau LENT



```
● ojfrancois@vdi-linux-26:~ $ ./pret.sh lend "livre2" "toto"
● ojfrancois@vdi-linux-26:~ $ ./pret.sh lend "dvd2" "machin"

16      "lent": [
17      {
18          "when": "29/11/2023",
19          "who": "toto",
20          "what": "livre2"
21      },
22      {
23          "when": "29/11/2023",
24          "who": "machin",
25          "what": "dvd2"
26      }
27  ]
```

Figure 10: Suppression d'un élément dans le tableau LENT :

```
ojfrancois@vdi-linux-26:~ $ ./pret.sh retrieve "livre2"

16      "lent": [
17      {
18          "when": "29/11/2023",
19          "who": "machin",
20          "what": "dvd2"
21      }
22  ]
```

Figure 11: Affichage des différents tableaux :

```
● ojfrancois@vdi-linux-26:~ $ ./pret.sh list "item"
You don't add PAGER by default, to use a PAGER select (Type 1 for less, Any other Number for more)
2
Code : "livre1" / Description : "Mon premier livre"
Code : "livre2" / Description : "Mon second livre"
Code : "dvd2" / Description : "Mon premier DVD"
● ojfrancois@vdi-linux-26:~ $ ./pret.sh list "lent"
You don't add PAGER by default, to use a PAGER select (Type 1 for less, Any other Number for more)
1
● ojfrancois@vdi-linux-26:~ $ ./pret.sh list "lent"
You don't add PAGER by default, to use a PAGER select (Type 1 for less, Any other Number for more)
2
Code : "dvd2" / To machin / 29/11/2023
○ ojfrancois@vdi-linux-26:~ $
```

## Conclusion

En conclusion, nous avons développé un script shell qui intègre les fonctionnalités de gestion des emprunts de documents. Ces fonctionnalités comprennent l'initialisation d'un fichier JSON, l'ajout à un fichier JSON existant, la suppression dans un fichier JSON existant, ainsi que le parcours d'un fichier JSON existant pour extraire les données nécessaires. Ce projet nous a permis d'apprendre et d'appréhender le format JSON ainsi que le programme "jq" mais également la mise en place d'un script bash conséquent. Nous avons également pu voir les améliorations réalisables sur un script une fois réalisé via l'outil shell-check.