Sujet : Mini Projet UE Unix

Réalisation d'une TODO LIST

M2 IMSD 2019-2020

NB : A renvoyer au plus tard le **15 mai 2020** par mail sur : chaaben.mohamed@outlook.com

- Copier/coller **M2_IMSD_2019_2020** dans l'Objet du mail envoyé
- Votre NOM en majuscule et prénom dans le corps du mail.
- Le script .sh en pièce jointe.

TABLE DES MATIERES

PROJET GERER UNE "TODO LIST"

DESCRIPTION

Pour éviter d'oublier des choses importantes à faire, je voudrais garder une liste dans mon ordinateur. J'aimerais pouvoir ajouter des tâches, les lister, et les supprimer.

Remarque: je pourrais également utiliser cette fonctionnalité pour garder :

- une liste de films à voir,
- une liste de course,
- etc.

OBJECTIF

Le script doit fournir une unique fonction `todo` qui offre 3 fonctionnalités :

- lister les tâches en attente
- supprimer une tâche
- ajouter une tâche

Voici un exemple d'utilisation :

```
$ source todo.sh
$ todo list
1 - finir TP d'info202
2 - téléphoner à tata
3 - inviter Edith à manger
4 - passer la serpillère dans l'entrée
$ todo done 1
La tâche 1 (finir TP d'info202) est faite !
$ todo list
1 - téléphoner à tata
2 - inviter Edith à manger
3 - passer la serpillère dans l'entrée
$ todo add 2 réviser la chimie
La tâche "réviser la chimie" a été ajoutée en position 2.
$ todo list
1 - téléphoner à tata
2 - réviser la chimie
3 - inviter Edith à manger
4 - passer la serpillère dans l'entrée
```

PRECISIONS

Votre fonction devra analyser son premier argument afin de décider quelle opération effectuer. Il faudra donc se reporter à la section **arguments individuels d'une fonction** et la description des **conditionnelles**.

STOCKAGE DES TACHES

La liste des tâche doit être sauvegardée dans un fichier (caché) `.todo_list` qui sera stocké dans votre dossier personnel. Ce fichier contiendra une ligne par tâche, *sans numéro*.

Pour simplifier la gestion de ce fichier, il est conseillé de le définir dans une variable au début de votre script:

```
TACHES=$HOME/.todo_list
```

Note : la variable `HOME` contient le chemin absolu vers votre dossier personnel...

MANIPULATION DU FICHIER

Pour manipuler les tâches, il faudra utiliser les choses suivantes.

- La commande `nl` permet d'afficher les lignes d'un fichier, en ajoutant un numéro de ligne. Il est possible d'ajouter un séparateur entre le numéro et la ligne si vous le souhaitez (voir `man nl` pour les détails).
- pour ajouter une tache en position `n`, il faudra :
    - rediriger les `n-1` premières taches dans un fichier temporaire avec la commande `head` et une redirection `>`,
    - ajouter la tache dans le fichier temporaire avec une commande `echo` et une redirection `>>`,
    - les tâches *à partir de la numéro* `n` dans le fichier temporaire avec la commande `tail` et une redirection `>>`,
    - remplacer le fichiers des tâches par le fichier temporaire.

  Les commandes head et tail seront utiles pour afficher les premières ou dernières lignes d'un fichier, et il faudra aussi lire la section sur les **opérations arithmétiques en bash**

- pour supprimer la tache en position `n`, il faudra
    - rediriger les `n-1` premières taches dans un fichier temporaire avec la commande `head` et une redirection `>`,
    - les tâches *à partir de la numéro* `n+1` dans le fichier temporaire avec la commande `tail` et une redirection `>>`,
    - remplacer le fichiers des tâches par le fichier temporaire.

POUR ALLER PLUS LOIN

- Gestion des erreurs (oubli du numéro de tache, etc.)
- Gestion de plusieurs fichiers pour des listes différentes.
- Filtre sur les taches en cours (avec grep) pour limiter l'affichage.
- etc.

Les commandes `head` et `tail` permette de récupérer des lignes au début ou à la fin d'un fichier :

- `head FICHIER` affiche par défaut les 10 premières lignes de `FICHIER`. Il est possible de changer cette valeur avec `head -n N FICHIER`. Pour afficher toutes les lignes, sauf les `N` dernières, on peut utiliser `head -n -N FICHIER`.
- `tail FICHIER` affiche par défaut les 10 dernières lignes de `FICHIER`. Il est possible de changer cette valeur avec `tail -n N FICHIER`. Pour afficher toutes les lignes à partir de la `N`ème, on peut utiliser `tail -n +N FICHIER`.

FONCTIONS SHELL

Un *script shell* est un petit programme écrit dans le langage du shell. C'est un fichier contenant des commandes et des constructions similaires à celles trouvées dans les langages de programmation plus "évolués".

La première ligne d'un tel fichier doit être `#!/bin/bash` afin que le système le reconnaisse comme script shell.

On peut définir une fonction dans un script bash de la manière suivante :

```
function test() {
    CMD1
    CMD2
    CMD3
    ...
}
```

Cela permet d'ajouter une commande : une fois que le fichier est lu par le shell, par exemple par :

```
$ source FICHIER
```

la commande `test` effectuera les commandes `CMD1`, `CMD2`, etc.

Comme avec Python, le fait d'avoir écrit une fonction n'est pas suffisant pour pouvoir l'exécuter. Il faut "charger" le fichier contenant la définition des fonctions avant de pouvoir les utiliser.

Dans le shell, on peut charger un fichier avec la commande

```
$ source FICHIER
```

D'autres fonctionnalités intéressantes sont :

instruction `echo`

On peut faire un affichage simple avec la commande `echo`

```
echo "FIN de la fonction "
```

variables

On peut définir des variables du shell avec le signe = :

```
var="..."
```

Attention, il ne faut pas mettre d'espace autour du signe =.

Pour utiliser la valeur d'une variable, il faut précéder son nom du signe `$`

```
echo "La valeur de var est $var"
```

Pour initialiser une variable avec le résultat d'une autre commande, il faut utiliser

```
var=$(CMD)
```

où `CMD` est la commande à exécuter.

conditionnelles

Pour faire des instructions conditionnelles, on peut utiliser un `if`.

La syntaxe est:

```
if [ TEST ]
then
    ...
elif [ TEST ]
then
    ...
elif [ TEST ]
then
    ...
else
    ...
fi
```

Attention, les espaces sont **obligatoires** après le symbole `[` et avant le symbole `]` !

`TEST` est une condition, qui porte en général sur des variables :

- `-z "$var"` pour tester si la variable est vide ou non,
- `-f "$var"` pour tester si la variable contient un nom de fichier qui existe,
- `"$var" == "hello"` pour tester si la variable contient la chaine `hello`,
- etc.

La liste des tests possibles est accessible avec

```
$ man test
```

BOUCLE SUR LES ARGUMENTS D'UNE FONCTION

Les boucles du langage bash ressemblent à

```
for i in LIST
do
    ...
done
```

où `LIST` est une liste de chaines, séparées par des espaces. Par exemple:

```
$ for i in chat chien souris
> do
>     echo "animal : $i"
> done
animal : chat
animal : chien
animal : souris
```

Les arguments d'une fonction sont automatiquement mis dans une variable spéciale appelée `$@`. Si le fichier `script.sh` contient

```
#!/bin/bash
function animaux() {
  for a in "$@"
  do
    echo "animal : $a"
  done
  echo "FIN de la fonction"
}
```

alors l'exécution donne :

```
$ source script.sh
$ animaux chat chien chauve souris canard
animal : chat
animal : chien
animal : chauve
animal : souris
animal : canard
FIN de la fonction
$ animaux
FIN de la fonction
```

ARGUMENTS INDIVIDUELS D'UNE FONCTION

Comme expliqué dans la section **boucle sur les arguments d'une fonction**, la liste des arguments d'une fonction est appelée `$@`.

Pour accéder aux premiers arguments individuellement, il faut utiliser les variables `$1`, `$2`, ... `$9`.

Pour accéder aux arguments suivants (après le numéro 9), il faut "décaler" les arguments. La commande `shift` supprime le premier argument et décale les suivants. Ainsi, après un `shift`, la variable `$1` contient l'argument numéro 2, etc.

Par exemple, si le fichier `args.sh` contient

```
#!/bin/bash
function montre_args() {
  echo "Tous les arguments : $@"
  echo "Argument 1: $1"
  shift
```

```
  echo "Autres arguments : $@"
}
```

on aura :

```
$ source ./args.sh
$ montre_args ananas pomme poire kiwi
Tous les arguments : ananas pomme poire kiwi
Argument 1: ananas
Autres arguments : pomme poire kiwi
```

OPERATIONS ARITHMETIQUES EN BASH

Les calcul arithmétique en bash doivent obligatoirement se trouver dans un `$((...))`.

```
$ echo "1 + 2"
1 + 2
$ echo "$((1 + 2))"
3
```

On peut utiliser des variables, et stocker le résultat dans une variable:

```
$ V=117
$ echo "la valeur de V est $V"
la valeur de V est 117
$ V=$(( $V / 2 ))
$ echo "la valeur de V est $V"
la valeur de V est 58
```

Attention le résultat d'une opération arithmétique n'est pas une commande :

```
$ $((1 + 2))
bash: 3 : commande introuvable
```