INF600A: Laboratoire #3
Scripts Unix

Jeudi 27 septembre 2018
13h30–15h30
PK-S1565

Le but de ce laboratoire est de vous familiariser avec l'écriture de **scripts** Unix.

Pour ces exercices, des fichiers vous sont fournis sous forme d'un dépôt `git`, que vous devez obtenir en exécutant la commande suivante :

```
git clone http://www.labunix.uqam.ca/~tremblay_gu/git/LaboScripts.git
```

Note : La première chose à faire une fois ce dépôt cloné est d'exécuter la commande «`make perms`» pour assurer **que les divers scripts** `{tf,num,map}.sh` **soient exécutables** :

```
$ cd LaboScripts
$ make perms
```

Quelle machine utiliser pour ce laboratoire?

Les scripts (solutions) des exercices ont été testés sur `java.labunix.uqam.ca` ainsi que sur mes machines personnelles — Linux CentOS et Mac OS X.

Ce qui vous est fourni

Comme pour le premier labo, un certain nombre de fichiers vous sont fournis :

- `{tf,num,map}.sh` : Scripts `bash` que vous devez compléter, pour qu'ils réalisent les fonctionnalités décrites plus bas.
- `makefile` : Pour exécuter les scripts — avec un exemple simple ou avec les tests.

  Initialement, `WIP=wip_ex` et `COMMANDE=tf`, donc la cible par défaut est l'exécution de l'exemple pour le script `tf.sh`.

  Pour exécuter plutôt les tests pour ce script, il suffit de supprimer le «`#`» devant «`WIP:`» :

  ```
  #WIP: wip_test
  ```

  Idem pour le 2^e ou le 3^e exercice, i.e., modifiez la valeur de `WIP` et/ou `COMMANDE`.
- `Fichiers/*.{txt}` : Des fichiers de données utilisés par les scripts que vous devez définir.
- `MapFonctions/*.{f}` : Deux fichiers contenant des définitions de fonctions utilisées pour le script `map.sh` du 3^e exercice.
- `Attendus/*.txt` : Les résultats attendus pour les scripts.

1. Complétez le script `tf.sh` qui trouve, dans un ou plusieurs fichiers `*.sh`, la ou les définitions d'une `function` dont le nom est spécifié en argument. Si plusieurs fichiers contiennent une telle définition, les définitions sont présentées dans l'ordre des noms de fichier. Voir le fichier `tf.sh` pour une description plus détaillée des fonctionnalités. Voir aussi le fichier `makefile`, qui contient quelques tests. Et voici un exemple d'exécution — notez que la définition pour «`usager`» n'est pas produite, uniquement celle pour «`usage`» :

```
$ cat Fichiers/n1.sh
#!/usr/bin -
function usage {
    echo "usage"
}

function usager {
    echo "usager"
}
```

```
$ tf.sh usage Fichiers/n1.sh
function usage {
    echo "usage"
}
```

Le `makefile` fourni définit les cibles `ex_tf` et `test_tf` pour éxécuter un exemple ou les tests.

a. Il faut utiliser `sed` avec un motif qui spécifie un **intervalle de lignes** — le premier item matche la déclaration de `function`, le deuxième matche l'accolade fermante.

   De plus, on ne veut qu'imprimer ces lignes et ne rien faire avec les autres lignes. L'appel à `sed` aura donc l'allure suivante, l'option `-n` faisant qu'aucune impression ne s'effectue **sauf lorsque explicitement indiqué avec l'action** `p` :

   ```
   debut_function="..."
   fin_function="..."
   ...
   sed -n "/$debut_function/,/$fin_function/p" ...
   ```

b. Le motif «\b» peut être utilisé pour matcher une **frontière de mot** :[1]

   ```
   $ ligne='abcdef dEf dXfgh'

   $ echo $ligne | grep -o 'd.f'
   def
   dEf
   dXf

   $ echo $ligne | grep -o '\bd.f'
   dEf
   dXf

   $ echo $ligne | grep -o 'd.f\b'
   def
   dEf

   $ echo $ligne | grep -o '\bd.f\b'
   dEf
   ```

[1] Mais cela ne fonctionne pas sous Mac OS X!

2. Complétez le script `num.sh` qui ajoute des numéros de ligne devant chaque ligne d'un fichier. Voir le fichier `num.sh` pour une description plus détaillée des fonctionnalités, ainsi que le fichier `makefile`, qui contient deux tests «normaux» ainsi que quelques tests avec des arguments erronés. Et voici aussi deux exemples d'exécution (avec un fichier qui n'est pas fourni dans les jeux de test) :

```
$ cat Fichiers/n0.sh
#!/usr/bin -
function usage {
    echo "usage"
}

$ num.sh Fichiers/n0.sh
1. #!/usr/bin -
2. function usage {
3.     echo "usage"
4. }

$ num.sh -4 Fichiers/n0.sh
   1. #!/usr/bin -
   2. function usage {
   3.     echo "usage"
   4. }
```

Le `makefile` qui vous est fourni définit les cibles `ex_num` et `test_num` pour éxécuter votre script avec divers fichiers et/ou arguments — exemples vs. vrais tests.

Quelques indices et suggestions :

- Le script `num.sh` contient une fonction `num` qui fait le travail effectif, en supposant des arguments corrects et validés. Quant à la validation, elle est faite dans le corps du «programme principal».
- Lorsqu'on utilise `awk`, la variable `NR` indique le numéro de l'enregistrement (*Number of records*), i.e., le numéro de ligne.
- Lorsqu'on utilise un test de la forme `[[ chaine =~ motif ]]`, `motif` est une expression régulière **étendue**.
- Si le «script» passé en argument à la commande `awk` est mis entre guillemets (`"..."`), dans le but d'effectuer l'expansion de variable, alors un numéro de champ d'enregistrement dans le script pour `awk` doit être précédé d'un *backslah* — c'est-à-dire «`\$1`».
- Le format utilisé comme premier argument de `printf` peut être une chaine quelconque, y compris une variable dans laquelle des substitutions de variables ont été effectuées.

3. Complétez le script `map.sh`, qui définit une opération `map` dans le style fonctionnel.

Plus spécifiquement, le script `map.sh` reçoit comme premier argument un nom de fonction, fonction dont la définition est soit globale (commande, primitive), soit obtenue à partir d'un fichier `MapFonctions/*.f`. Cette fonction est alors appliquée à chacun des éléments d'un flux, c'est-à-dire, éléments provenant *i*) soit de `stdin` si aucun autre argument n'est spécifié, *ii*) soit de la concaténation des éléments des fichiers indiqués en argument.

Voici des exemples d'exécution :

```
$ cat Fichiers/nombres.txt
10
20
30

$ ./map.sh echo Fichiers/nombres.txt
10
20
30

$ cat MapFonctions/plus1.f
plus1 () {
  ligne="$1"

  (( res = $ligne + 1 ))
  echo $res
}

$ cat Fichiers/nombres.txt | ./map.sh plus1
11
21
31
```

Le `makefile` fourni définit les cibles `ex_map` et `test_map` pour éxécuter un exemple ou les tests.

Indice : Une façon de traiter chaque ligne provenant d'un fichier `foo.txt` est comme suit :

```
cat foo.txt | while read ligne; do
  traiter $ligne ...
done
```