INF600A — Langages de script et langages dynamiques
Examen intra (Hiver 2017)

Durée: 13h30 – 16h30
Documentation personnelle permise, y compris *laptop* ou tablette, mais en mode «lecture de documents personnels» — pas de sites Web!

Nom: _____

Code permanent: | | | | | | | | | | | | |

1	2	3	4	5	6	Total
/6	/6	/7	/7	/8	/6	/40

- Répondez directement sur le questionnaire, en utilisant seulement le recto des pages.
- Dans quelques cas, vous devez compléter un script partiellement défini.
  Les parties à compléter sont indiquées par un commentaire `# A COMPLETER` ou par des boites à remplir [ ] (pas de problème si vous dépassez de la boite)
- Quelques rappels :
  - `tr -d '...'`
    Supprime toutes les occurences des caractères indiqués.
  - `tr -s '...'`
    Supprime les répétitions (consécutives) des caractères indiqués pour ne conserver au plus qu'une seule occurrence.
  - item `cmd >> fich.txt`
    Ajoute (mode *append*) les lignes émises sur `stdout` au fichier `fich.txt`.
  - `cat fich.txt | while read ligne; do ... done`
    Permet, dans le corps de la boucle, de traiter chacune des lignes du fichier `fich.txt`, et ce par l'intermédiaire de la variable `ligne`.

1. Motifs et expressions régulières (6 pts)

Pour chacun des motifs ci-bas, donnez la **chaine la plus courte possible** qui *matche* le motif. Utilisez le caractère `X` lorsque vous pouvez/devez utiliser un caractère arbitraire ; lorsqu'un caractère doit provenir d'une classe (ensemble) de caractères, utilisez le premier caractère de la classe.

Dans le cas de l'expansion de noms de fichiers (*file globbing*), vous devez supposer que le motif est utilisé comme argument pour la commande `ls`. Notez que dans les cas de *file globbing* ci-bas, il n'y a aucune erreur signalée — les deux motifs présentés génèrent un nom de fichier valide!

Motif	**Expansion de noms de fichiers (*file globbing*)**
`bar{2,34}a+b?.sh*`	
`ab*[!ab]{2}[^0-9]`	

Motif	**Expression régulière simple**
`bar{2,34}a+b?.sh*`	
`ab*[!ab]{2}[^0-9]`	

Motif	**Expression régulière étendue**
`bar{2,34}a+b?.sh*`	
`ab*[!ab]{2}[^0-9]`	

2. Pipelines avec `awk`, `grep`, `sed`, `sort`, `tr` et `xargs` (6 pts)

Soit le fichier suivant :

```
$ cat inscriptions.txt
Joe,Bidon,(514) 888-3334,INF1120:A;INF1130:B;MET1105:C
Catherine,Doe,(514) 324-3454,INF2120:A;INF2170:C;MET1105:B
Nellie,Durocher,(450) 819-4493,
Guy,Tremblay,(450) 344-9539,INF1130:A;INF2170:A
```

Indiquez ce qui sera émis sur `stdout` par l'exécution de chacun des pipelines ci-bas.

```
cat inscriptions.txt |
   awk -F, '{ print $4 }' |
   grep -v "^$" |
   sed 's/;/\n/g' |
   sed 's/^.*://g' |
   sort -u
```

```
$ cat inscriptions.txt |
   awk -F, '{ print $3 }' |
   sed 's/).*$//' |
   tr -d '()' |
   sort -u
```

```
$ ls SousRep
Bidon.data      Doe.data        Durocher.data   Tremblay.data

$ cat inscriptions.txt |
   grep "\([0-9]\)\1.*-" |
   awk -F, '{ print $2 }' |
   xargs -I {} mv SousRep/{}.data SousRep/{}.xxx

$ ls SousRep
```

3. Script pour traiter un journal produit par `syslog` (7 pts)

Un journal `syslog` contient des informations sur les processus exécutés par divers *hosts* :

- la date (et l'heure) à laquelle a été émis le log
- le nom de l'équipement (*hostname*) ayant généré le log (e.g., `c98772`, `c34581`)
- une information sur le processus qui a déclenché cette émission (e.g., `systemd`, `dbus`)
- un identifiant du processus ayant généré le log (e.g., `830`, `2938`)
- un corps de message.

Les champs sont séparés par des blancs — voir Fig. 1 pour un exemple.

Figure 1 Extrait d'un journal (fichier `/var/log/messages`) produit par `syslog`.
Note : Les **«\» ne sont pas présents dans le journal!** Ils n'apparaissent ici que parce que certaines entrées du journal, trop longues, sont présentées ci-bas sur deux lignes.

```
Jan 31 08:20:01 c98272 systemd: Started Session 232 of user root.
Jan 31 08:20:01 c98272 systemd: Starting Session 232 of user root.
Jan 31 08:30:02 c34581 systemd: Started Session 233 of user root.
Jan 31 08:30:02 c34581 systemd: Starting Session 233 of user root.
...
Jan 31 10:12:13 c34581 dbus[830]: [system] Activating via\
       systemd: service name='org.freedesktop.hostname1'\
       unit='dbus-org.freedesktop.hostname1.service'
Jan 31 10:12:13 c34581 systemd: Starting Hostname Service...
...
Jan 31 10:12:13 c34581 udisksd[2938]:\
       Mounted /dev/sdc1 at /run/media/tremblay_g/COURSVERT on behalf of uid 1000
Jan 31 10:12:59 c34581 udisksd[2938]:\
       Cleaning up mount point /run/media/tremblay_g/COURSVERT (device 8:33 is not mounted)
...
```

On veut produire, pour chaque *hostname*, la liste des processus exécutés. De plus, pour les noms de processus, on ne veut conserver que le préfixe formé uniquement de lettres, donc sans le numéro (identifiant) du processus — par exemple, pour la dernière ligne de la figure, on veut indiquer «`udisksd`» et non «`udisksd[2938]:`».

La figure 2 présente un exemple de sortie, où la liste des lignes est ordonnée selon les *hostnames*, et les processus sur chaque ligne sont aussi ordonnés (ordre alphabétique).

Figure 2 Exemple de sortie attendue en utilisant le script `process-syslog.sh` sur le fichier `/var/log/messages` de la Figure 1.

```
$ ./process-syslog.sh /var/log/messages
c34581: avahi dbus dhclient dnsmasq fprintd kernel systemd udisksd
c98272: avahi dhclient dnsmasq gnome kernel NetworkManager nm systemd
d43299: gnome kernel NetworkManager systemd
```

Complétez le squelette de script `process-syslog.sh` présenté à la page suivante pour réaliser la fonctionalité désirée.

```
syslog="$1"

# Emet (sur stdout) la liste des noms de hostname dans syslog
# $1: Nom du fichier syslog
function hostnames {
    local syslog="$1"

    cat "$syslog" |
      # A COMPLETER




}

# Emet (sur stdout) la liste des processus executes par un hostname dans syslog
# $1: Nom du fichier syslog
# $2: Nom du hostname cherche
function processus {
    local syslog="$1"
    local hostname="$2"

    cat "$syslog" |
      # A COMPLETER









}

# On trouve les processus executes par chacun des hostname
for hostname in                                    ; do
                                                                                    
done
```

4. Script `generer-tests.sh` pour générer des tests d'acceptation de programmes filtres (7 pts)

Au labo #4, on a vu le script `tester_filtre.sh` qui permet de tester un *programme filtre*, et ce en spécifiant pour chaque cas de test un fichier `.input` — les données à utiliser pour le test — et un fichier `.output` — les sorties attendues.

Le désavantage de ce script est que chaque cas de test doit être décrit par **deux (2) fichiers**. Par exemple, pour un test `test1`, il faut définir les fichiers `test1.input` et `test1.output`.[1]

Pour faciliter l'écriture de tests, on veut donc développer un script, `generer-tests.sh`, qui va générer les fichiers `.input` et `.output` à partir d'un unique fichier `.test` : les lignes au début du fichier vont indiquer les données (transmises via `stdin` au programme testé) ; une ligne débutant par «`===`» va indiquer la fin des données ; les lignes qui suivent vont indiquer les résultats attendus (sur `stdout`). Un exemple est présenté ci-bas.

```
# AVANT l'appel a generer-tests.sh

$ tree Tests
Tests
|-- t1.test
‘-- t2.test

0 directories, 2 files

$ cat Tests/t1.test
abc def ghi
jkl mno
===
5

$ cat Tests/t2.test
===
0

# APPEL a generer-tests.sh.
$ ./generer-tests.sh Tests
```

```
# APRES l'appel a generer-tests.sh.

$ tree Tests
Tests
|-- t1.input
|-- t1.output
|-- t1.test
|-- t2.input
|-- t2.output
‘-- t2.test

0 directories, 6 files

$ cat Tests/t1.input
abc def ghi
jkl mno

$ cat Tests/t1.output
5

$ cat Tests/t2.input

$ cat Tests/t2.output
0
```

Complétez le script à la page suivante pour qu'il produise les fichiers `*.input`/`*.output` requis à partir **d'un répertoire contenant des fichiers** `*.test`.

[1] Pour cette question, contrairement au labo #4, on ignore les arguments fournis au programme testé, qui pouvaient être spécifiés par un fichier `.arguments`, e.g., `test1.arguments`.

```
#! /bin/sh -

readonly SEPARATEUR="==="

function erreur { echo "*** Erreur: $1" 1>&2; exit 1; }

repertoire_tests="$1"; shift
[[ |                              | ]] ||
    erreur "Pas un repertoire: '$repertoire_tests'"


les_tests= |                                                    |


for test in $les_tests; do

    nom_du_test= |                                              |

    rm -f $nom_du_test.{input,output}
    touch $nom_du_test.{input,output}

    fichier="$nom_du_test.input"

    cat |                                  | | while read ligne; do
        if [[ $ligne =~ ^$SEPARATEUR ]]; then
            [[ $fichier == $nom_du_test.input ]] ||
                erreur "Plusieurs separateurs dans $fichier!?"
            fichier= |                     |
        else
            | # A COMPLETER |



        fi
    done

done
```

5. Script `ga.sh` : Une commande pour trouver les cours qui ont un certain sigle en préalable (8 pts)

Dans le script `ga.sh` du Devoir #1, on veut ajouter une commande pour identifier **chacun des cours qui a un cours donné dans sa liste de préalables directs**.

Plus précisément, cette opération permet, étant donné un sigle de cours `sigle`, de trouver tous les cours qui ont `sigle` comme préalable **direct**. Pour les divers cours trouvés, on affiche alors les sigles, en ordre — évidemment sans répétition!

La figure 3 présente des exemples d'exécution de cette commande (p. 10).

Complétez le code de la fonction `ont_en_prealable`, dont le squelette est fourni à la p. 11.

Hypothèses ou contraintes à respecter :

- Comme dans le devoir #1, le séparateur utilisé dans le dépôt (la banque de cours) est donné par la variable `SEP` (ou `SEPARATEUR`) — associée au caractère «`,`», .
- La fonction `assert_depot_existe` termine l'exécution si le dépôt indiqué n'existe pas. Donc, dans la fonction `ont_en_prealable` (p. 11), on est assuré que le dépôt à utiliser existe **après** l'exécution des instructions déjà indiquées.
- La fonction `ont_en_prealable` est appelée du script principal (`ga.sh`) comme toutes les autres fonctions, donc elle doit retourner le nombre d'arguments «consommés».
- Pour valider la forme d'un sigle de cours, vous devez utiliser la variable suivante, qui définit le motif approprié :

```
readonly MOTIF_SIGLE="[A-Z]{3}[0-9]{3}[A-Z0-9]"
```

- Pour signaler une erreur, vous devez utiliser la fonction suivante :

```
#-------
# Affiche un message d'erreur (sur stderr).
#
# Arguments: msg
#-------
function erreur { ...; exit 1; }
```

Figure 3 Exemples d'exécution de la commande `ont_en_prealable`.

```
$ ./ga.sh lister
INF1120 "Programmation I" ()
INF1130 "Mathematiques pour informaticien" ()
INF2120 "Programmation II" (INF1120)
INF3105 "Structures de donnees et algorithmes" (INF1130:INF2120)
INF3135 "Construction et maintenance de logiciels" (INF1120)

$ ./ga.sh ont_en_prealable INF2120
INF3105

$ ./ga.sh ont_en_prealable INF1120
INF2120
INF3135

$ ./ga.sh ont_en_prealable INF3135

$ ./ga.sh ont_en_prealable INF313
*** Erreur: Sigle de forme incorrecte: INF313

$ ./ga.sh ont_en_prealable INF3100
*** Erreur: Aucun cours avec le sigle 'INF3100'.
```

```
# Commande ont_en_prealable
#
# Arguments: depot sigle
#
# Erreurs:
# - depot inexistant
# - sigle de forme invalide ou inexistant
#
function ont_en_prealable {
    depot=$1; shift
    assert_depot_existe $depot
    # A COMPLETER

}
```

6. Appels à un script `mystere.sh` (6 pts)

Soit la fonction présentée à la page suivante — page que vous pouvez détacher pour en faciliter la lecture et la consultation.

Soit le fichier `foo.txt`, le seul fichier avec une extension présent dans le répertoire courant :

```
$ ls *.*
foo.txt
```

```
$ cat foo.txt
cat
def
echo
```

Indiquez ce qui sera émis sur `stdout` par chacun des appels ci-bas.

```
$ foo -2 echo foo bar abc def xyz
```

```
$ foo -3 $(echo echo abc) $(cat foo.txt) 1 2 3 4
```

```
$ foo $(grep h foo.txt) 10 20 30
```

```
$ foo -2 $(grep a foo.txt) foo.txt abc.def
```

```
function foo {
  n=1
  if [[ $1 =~ ^- ]]; then
      [[ $1 =~ ^-[0-9]+$ ]] || exit 1
      n=${1#-}; shift
  fi

  c=$1; shift
  [[ $c ]] || exit 1

  while [[ $# -ge $n ]]; do
      as=""
      for (( i = 0; i < n; i++ )); do
          as="$as $1"
          shift
      done
      $c $as
  done
}
```