INF600A — Langages de script et langages dynamiques
Examen intra$_A$ (Hiver 2016)

Durée: 9h30 – 12h30
Documentation : Aide-mémoire + Feuille de scripts + Une (1) feuille personnelle.

Nom: _____

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

1	2	3	4	5	6	Total
/6	/6	/6	/6	/10	/6	/40

- L'examen comporte six (6) questions pour un total sur 40. L'examen comporte aussi une sous-question, optionnelle, pouvant donner lieu **à des points bonus** — il est donc possible d'obtenir une note > 40!
- Répondez directement sur le questionnaire, en utilisant seulement les rectos des pages — les versos sont pour vos brouillons et ne seront pas corrigés!
- **Ajout à l'aide-mémoire** : «`diff -q`» compare deux fichiers, mais en retournant simplement un code de statut (fichiers identiques=0 vs. fichiers différents=1), et ce sans émettre sur `stdout` les différences s'il y en a.

Andy Hunt (@PragmaticAndy)

2016-02-11 15:10

There's few things as lovely as a short shell script that just works, and would have been much harder to do by hand.

Note : Hunt est un des deux auteurs du livre «*The Pragmatic Programmer*» — un livre que tout informaticien/développeur de logiciels devrait lire!

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 `A` lorsque vous pouvez/devez utiliser un caractère arbitraire ; lorsqu'un caractère peut provenir d'une classe spécifique 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 tous les cas de *file globbing*, aucune erreur n'est signalée — les motifs présentés génèrent tous un nom de fichier valide!

Motif	**Expansion de nom de fichier**
`foo[123]+ab?.cd*`	
`.*foo[^a]\(ab\)?c{3}`	

Motif	**Expr. rég. simple**
`foo[123]+ab?.cd*`	
`.*foo[^a]\(ab\)?c{3}`	

Motif	**Expr. rég. étendue**
`foo[123]+ab?.cd*`	
`.*foo[^a]\(ab\)?c{3}`	

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

Soit le fichier suivant :

```
$ cat inscriptions.txt
Bidon,Joe,BIDJ11111111,INF1120+INF1130+MET1105
Doe,Jane,DOEJ22622222,INF2120+INF2170+MET1105
Durocher,Nellie,DURN08580808,
Tremblay,Guy,TREG05050505,INF1130+INF2170
```

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' |
  sort -u
```

```
$ pvs="[^,]*"
$ cat inscriptions.txt |
  grep "^$pvs,$pvs,$pvs,$" |
  awk -F, '{ print NR ": " $1 }'
```

```
$ ls SousRep
Bidon-Joe.data Doe-Jane.data Durocher-Nellie.data Tremblay-Guy.data

$ cat inscriptions.txt |
  grep -E ',[a-zA-Z]{4}[0-9]{2}[5-6]' |
  awk -F, '{ print $1 "-" $2 }' |
  xargs -i rm SousRep/{}.data

$ ls SousRep
```

3. Script `rv.sh` pour inverser les lignes d'un fichier (6 pts)

Lorsqu'on utilise «`sort -n`», le tri se fait en fonction des valeurs **numériques** apparaissant en début de ligne, et ce même si la ligne n'est pas composée uniquement de chiffres :

```
$ cat foo.txt          $ sort foo.txt         $ sort -n foo.txt
12 ab                  12 ab                  2. cd
2. cd                  2. cd                     7 efg hijk lm
   7 efg hijk lm       65 q r s               12 ab
65 q r s                  7 efg hijk lm       65 q r s
```

Utilisez cette propriété, ainsi que l'option `-r` de tri **en ordre inverse** (alphabétique ou numérique), pour écrire un script `rv.sh` qui émet sur `stdout` les lignes d'un fichier mais en ordre inverse — donc sans modifier le fichier original :

```
$ cat foo.txt          $ ./rv.sh foo.txt      $ cat foo.txt
12 ab                  65 q r s               12 ab
2. cd                     7 efg hijk lm       2. cd
   7 efg hijk lm       2. cd                     7 efg hijk lm
65 q r s               12 ab                  65 q r s
```

Indice : Utilisez le script `num.sh` (i.e., vous pouvez simplement l'appeler!), script développé dans le laboratoire #3, pour ajouter des numéros de ligne, trier le fichier numéroté, puis supprimer les numéros.

```
$ cat rv.sh
```

4. Script `tries.sh` pour trouver les fichiers déjà triés (6 pts)

On veut définir un script `tries.sh` qui reçoit un *flag* spécifiant **une extension** et un argument indiquant **un nom de répertoire**. Ce script émet sur `stdout` les noms de fichiers ayant l'extension indiquée, accessibles dans le répertoire indiqué ou dans les sous-répertoires, **dont le contenu est déjà trié**.

Voici un exemple d'utilisation, avec l'appel «`./tries.sh -e .txt .`», donc qui analyse les fichiers avec extension «`.txt`» dans le répertoire courant :

```
$ cat UnDir/t0.txt
123
456
789

$ cat t1.txt
abc
def
ghi

$ cat pt1.txt
def
abc
ghi
```

```
$ ls *.txt UnDir/*.txt
pt1.txt  t1.txt UnDir/t0.txt

$ ls -1 -R | grep .txt
pt1.txt
t1.txt
t0.txt

$ ./tries.sh -e .txt .
./UnDir/t0.txt
./t1.txt
```

Un squelette pour ce script est présenté à la page suivante.
Complétez les deux parties indiquées : (*i*) la fonction `est_trie` et (*ii*) l'expression après `in` dans l'instruction «`for fich in ...`». N'oubliez pas de supprimer tout fichier auxiliaire que votre script pourrait avoir créé.

Indice : Un fichier est déjà trié si, lorsqu'on le trie, on obtient le même contenu que le fichier initial, c'est-à-dire, il n'y a pas de différence entre le fichier original et la version triée de ce fichier!

Bonus (3 pts) : Lorsqu'on tri un fichier avec `sort`, les lignes vides ou qui ne contiennent que des blancs apparaissent au début du fichier trié :

```
$ cat f.txt

abc

def
```

```
$ sort f.txt


abc
def
```

Concevez la fonction `est_trie` pour qu'un fichier soit considéré comme trié même lorsque le fichier contient des lignes vides ou des lignes avec uniquement des espaces, en autant que les autres lignes soient correctement ordonnées. Notez que dans ce cas, les options «`-iwB`» de `diff` **ne suffisent pas**, donc inutile d'utiliser ces options!

```
usage() { echo "usage:"; echo " $0 -e ext repertoire"; exit 1; }
verifier_arguments () { ... } # Voir plus bas.

function est_trie {
  # A COMPLETER: Corps de la fonction est_trie!









}

################################
verifier_arguments "$@"
#  A ce point, on a:
#    a. $repertoire existe et est effectivement un repertoire
#    b. $ext contient l'extension specifiee en argument

# A COMPLETER: l'expression apres in, qui doit utiliser $repertoire et $ext

for fich in
do
    est_trie $fich && echo "$fich"
done
```

5. Script `biblio.sh` : Une opération pour rappeler un livre (10 pts)

Dans le script `biblio.sh` du devoir 1, on veut ajouter une opération pour effectuer **le rappel d'un livre**. Plus précisément, cette opération permet, étant donné un titre, de déterminer l'emprunteur du livre ayant (exactement) ce titre et d'envoyer un courriel à cette personne, sauf si l'adresse courriel qui lui est associée est «`@`».

La figure 1 présente des exemples d'exécution de cette commande (p. 8).

Complétez le code de la fonction `rappeler`, dont le squelette est donné p. 10.

Vous pouvez/devez supposer ce que suit :

- Le séparateur utilisé dans le dépôt (la BD) est le caractère «`%`». Voir, à la page 9, le code pour la fonction `emprunter` (et une fonction auxiliaire utilisée par cette fonction).
- La fonction `rappeler` est appelée du script principal comme toutes les autres fonctions — donc elle doit retourner le nombre d'arguments «consommés».
- La fonction `assert_depot_existe` termine l'exécution si le dépôt indiqué n'existe pas. Donc, dans la fonction `rappeler` (p. 10), le dépôt à utiliser existe bien après l'exécution des deux instructions déjà indiquées.
- Vous devez supposer qu'une fonction `envoyer_courriel_de_rappel` est déjà définie et vous devez l'utiliser comme suit pour effectuer l'envoi effectif d'un message courriel :

  ```
  envoyer_courriel_de_rappel $courriel $nom $titre
  ```

  C'est cette fonction, que vous n'avez pas à définir, qui composera le message pour le couriel et qui effectuera l'envoi. Par contre, c'est bien votre fonction `rappeler` **qui doit afficher le message de confirmation d'envoi.**

  Pour simplifier, vous pouvez supposer que `envoyer_courriel_de_rappel` ne génère pas d'erreur.

```
$ ./biblio.sh init --detruire

$ ./biblio.sh lister

$ ./biblio.sh emprunter "Bidon" "bidon.joe@uqam.ca" "Titre 1" "Auteurs 1"

$ ./biblio.sh emprunter "Tremblay" "@" "Titre 2" "Auteurs 2"

$ ./biblio.sh lister
Bidon :: [ Auteurs 1  ] "Titre 1"
Tremblay :: [ Auteurs 2  ] "Titre 2"

$ ./biblio.sh rappeler "Titre 1"
Un courriel a ete transmis a bidon.joe@uqam.ca

$ ./biblio.sh rappeler
*** Erreur: Nombre incorrect d'arguments.

$ ./biblio.sh rappeler "itre 1"
*** Erreur: Aucun livre emprunte avec le titre 'itre 1'.

$ ./biblio.sh rappeler "Titre 2"
*** Erreur: Aucune adresse de courriel n'est specifiee:
            titre = 'Titre 2'; emprunteur = Tremblay.

$ ./biblio.sh rappeler "Titre 1" "FOO"
Un courriel a ete transmis a bidon.joe@uqam.ca
*** Erreur: Argument(s) en trop: 'FOO'
```

Figure 1: Exemples d'exécution de la commande `rappeler`.

```
function obtenir_emprunteur {
    depot=$1
    titre="$2"
    cat $depot | awk -F% "/$titre/ {print \$1}"
}


function emprunter {
    [[ $# == 5 ]] || erreur_nb_arguments "$@"

    depot=$1
    assert_depot_existe $depot

    nom="$2"
    courriel="$3"
    titre="$4"
    auteurs="$5"

    emprunteur=$( obtenir_emprunteur $depot "$titre" )
    [[ -z $emprunteur ]] ||
        erreur "Un livre avec le meme titre est deja emprunte."

    # Le livre n'est pas deja emprunte: On l'ajoute
    echo "$nom%$courriel%$titre%$auteurs%" >> $depot

    return 4
}
```

```
# A COMPLETER!
function rappeler {
  depot=$1
  assert_depot_existe $depot









}
```

6. Script `mystere.sh` (6 pts)

Soit le script présenté à la page 13, page que vous pouvez détacher pour analyser le script et déterminer les résultats produits par les trois (3) appels ci-bas.

```
$ cat test1.input
10,20,30
11,12,13
abc,def,ghi
```

```
$ cat test1.input | ./mystere.sh -F, -f1
```

```
$ cat test5.input
10,20,30
120,240,300
0909,def
```

```
$ ./mystere.sh -F0 -f2 test5.input test5.input
```

```
$ cat test4.input
10#20#30
11#12#13
abc#ef#ghi
```

```
$ ./mystere.sh -F# -f3,1 test4.input
```

```
$ cat mystere.sh
#!

function usage { echo "usage:"; echo "  $0..."; exit 1; }

function erreur { echo "Erreur!"; exit 1; }

NOMBRE='[0-9]+'

function generer_script {
    args=$1

    script="{ print"
    while [[ $args =~ ^$NOMBRE(,$NOMBRE)+$ ]]; do
        f=${args%%,*}
        script="$script \$$f, "
        args=${args#*,}
    done
    script="$script \$$args }"
}

########################################
# PROGRAMME PRINCIPAL
########################################

# RAPPEL: =~ utilise un motif qui est une expr. reg. *etendue*

[[ $# != 0 ]] || usage

[[ $1 =~ -F. ]] || erreur
arg1=${1##-F}
shift

[[ $1 =~ ^-f$NOMBRE(,$NOMBRE)*$ ]] || erreur
args2=${1##-f}
shift

generer_script $args2

cat "$@" | awk -F$arg1 "$script"

exit 0
```