Initiation à la programmation shell : Partiel

15 Avril 2019 — Durée: 3 heures

Polycopié, notes de cours et de TP autorisés

À lire :

— Les exercices et les questions sont de difficulté variable et en général non croissante. Il est conseillé de parcourir l'énoncé avant de composer. Les exercices peuvent être traités dans l'ordre de votre choix.

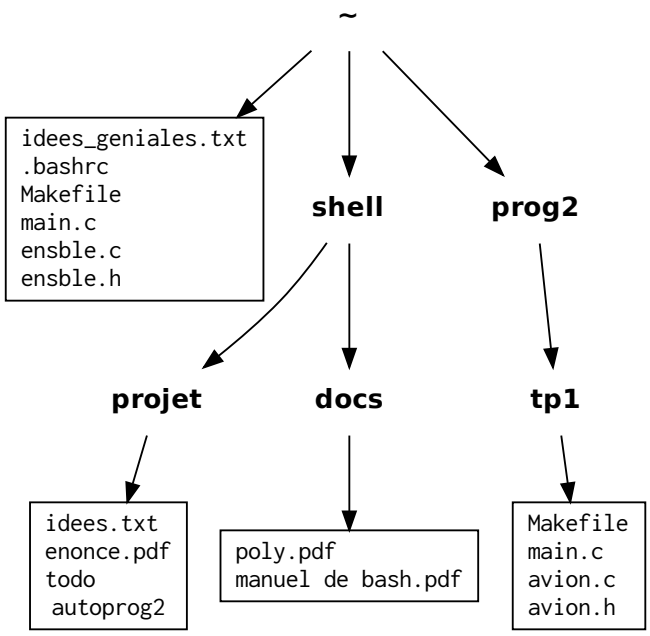
— Il est toujours possible de sauter des questions.

— Le soin, la clarté et la lisibilité seront pris en compte lors de la correction.

— Pour certaines questions, une réponse partielle ou incomplète *pourra* être prise en compte lors de la correction.

— Le barème est indicatif et pourra être modifié. *Pour l'instant*, le total des points est 104, et votre total sera divisé par 5 pour obtenir une note sur 20.

Exercice 1 : **37 points**



Dans l'arborescence représentée dans ci-dessus, les répertoires sont écrits en **gras** et les fichiers qui ne sont pas des répertoires sont écrits en `mono-châsse`, dans des rectangles. Le répertoire ~ est votre répertoire personnel.

Vous commencez cette grande aventure dans votre répertoire personnel.

Pour chacune des questions suivantes, vous écrirez sur la copie la ou les commande(s) que vous devez entrer pour réaliser les actions demandées.

1. Démarrage d'un TP en C **/10**

a) Créer le répertoire **tp2** dans **prog2** et s'y déplacer. Dans ce répertoire, créer en une seule commande les fichiers (vides pour l'instant) `main.c`, `scooter.c` et `scooter.h`.

b) On rappelle que votre répertoire courant est le sous-répertoire **tp2** du répertoire **prog2**. Quels sont les chemins relatifs et absolus du fichier `Makefile` situé dans **tp1** ?

c) Copier ce fichier dans le répertoire **tp2**.

d) Dans cette copie du fichier, remplacer toutes les occurences du mot « avion » par le mot « scooter ».

e) Écrire (avec une ou plusieurs commandes), dans le fichier `scooter.h` du répertoire **tp2** les lignes suivantes :

```
#ifndef SCOOTER_H
#include SCOOTER_H
#endif
```

2. Recherches dans le répertoire **tp1** **/7**

a) Se déplacer dans le répertoire **tp1** (depuis **tp2**).

b) Lister les fichiers de ce répertoire.

c) Afficher seulement les lignes des fichiers de ce répertoire qui contiennent la chaîne de caractère `avion_creer`.

d) Afficher seulement les lignes du fichier `avion.h` qui commencent par `#`

e) Afficher seulement les lignes de `avion.h` qui ne sont pas vides.

f) Compter les lignes du fichier `avion.h`.

g) Compter les lignes non vides du fichier `avion.h`.

3. Un peu de rangement ! **/5**

a) Créer le répertoire **tp_ensemble** comme sous-répertoire de **prog2**, puis y déplacer (en une seule commande) les fichiers `Makefile`, `main.c`, `ensble.c` et `ensble.h` du répertoire ~.

b) Renommer le répertoire **tp2** pour l'appeler **tp_scooter** (toujours dans **prog2**).

c) Copier (toujours dans **prog2**) le répertoire **tp1** en nommant la copie **tp_avion**.

d) Supprimer le répertoire **tp1** (et tout ce qu'il contient).

4. Permissions **/6**

a) Rendre les fichiers `todo` et `autoprog2` du répertoire **projet** exécutables par vous.

b) Sur le fichier `idees_geniales.txt`, enlever toutes les permissions du groupe et des autres utilisateurs.

c) Supprimer, pour tout le monde, la permission d'écrire sur le fichier `autoprog2`.

d) Que faire pour s'assurer que personne (à part root) ne puisse supprimer le fichier `todo` ? Avec quelle commande ?

e) On a tapé dans le terminal la commande suivante, avec le résultat suivant (le `$` représente l'invite de commande) :

```
$ umask
022
```

Si le *fichier normal* `idees_pourries.txt` et le répertoire **web** sont créés dans votre répertoire personnel, quelles sont les permissions qui leur sont associées ?

5. Documents **/6**

a) Le programme `evince` permet d'ouvrir les fichiers `pdf`. Lancer ce programme *en arrière-plan* pour ouvrir le fichier `poly.pdf`.

b) Afficher le PID de ce processus.

c) Mettre fin à ce processus (ou, au pire, donner une méthode pour le faire en utilisant seulement le terminal).

d) Ouvrir, cette fois au premier plan, toujours avec `evince`, le fichier `manuel de bash.pdf`.

e) Mettre ce processus en arrière-plan (ou, au pire, donner une méthode pour le faire en utilisant seulement le terminal).

6. Pot-pourri **/3**

a) Quelle est la particularité du fichier `.bashrc` ? Comment le lister avec la commande `ls` ?

b) Comment afficher le nom de tous les fichiers `pdf` dans cette arborescence ?

c) Comment afficher le nom de tous les répertoires dans cette arborescence ?

Exercice 2 : 17 points

1. Voici une partie de la sortie d'une commande `ls -ld *` :

```
-rw-r--r-- 2 rob genies  271 17 janv. 12:35 a_faire.txt
-rwx--x--x 1 rob genies  872  9 fevr. 23:09 ranger.sh
drwxr-xr-x 2 rob genies 4096 27 mars  11:03 solutions/
lrwxrwxrwx 1 rob genies   10  8 dece. 07:07 urgent -> ../attention
```

a) Que signifient respectivement `-`, `d` et `l` au tout début des lignes ci-dessus ?

b) À quoi correspondent les mots `rob` et `genies` pour ces fichiers ?

c) Que signifient les permissions associées à `ranger.sh` ? Les écrire en représentation octale.

d) À quoi correspond le nombre qui suit `genies` ? La date et l'heure qui suivent ce nombre ?

e) Que signifie pour le fichier `a_faire.txt` le premier nombre `2` ?

f) Qu'indique la chaîne `urgent -> ../attention` ?

2. Expliquer en une phrase ce que fait chacune des commandes suivantes :

a) `ls *.???`

b) `kill -SIGCONT 38470`

c) `ls -l /etc | wc -l`

d) `cat ~/tata/c.txt >> ~/toto/a.txt`

e) `cp /var/log/[b-m]?*[!3-9] /tmp`

f) `ps -ef > psliste.txt`

g) `ps -ef | grep ^root`

3. Donner une commande permettant de lister les fichiers du repertoire courant qui ont pour extension `.c` ou `.h` et dont le nom *contient* la chaîne de caractère `liste`

Exercice 3 : **17 points**

On rappelle que la commande `wc -c` permet d'afficher *la taille en octets d'un fichier passé en arguments*. En voici un exemple de sortie sur le fichier `index.html` (encore une fois `$` représente l'invite de commande) :

```
$ wc -c index.html
2628 index.html
```

On considère dans cet exercice le script (moyennement utile) `tailles` suivant :

```
1  #! /bin/bash
2
3  repertoire=$1
4  limite=100 #taille limite en Ko (kilooctets)
5  limite_octets=$(( 1024 * $limite ))
6  for fichier in "$repertoire"/* ; do
7      taille=$(wc -c "$fichier" | sed 's/ .*//')
8      if [ $taille -gt $limite_octets ] ; then
9          echo "Le fichier $fichier fait plus de $limite Ko !"
10     fi
11 done
```

1. Expliquer à quoi sert la ligne 1 de ce script.

2. Expliquer ce que font les lignes 3 et 4 de ce script.

3. Que va faire la ligne 5 ? À quoi sert la syntaxe `$((  ))` ?

4. À quoi sert la ligne 6 ?

5. La ligne 7 sert à mettre la taille (en octets) du fichier nommé `$fichier` dans la variable `taille`. Pour ce faire :

a) À quoi sert la syntaxe `$(  )` ?

b) À quoi sert le symbole `|` ?

c) Que fait la commande `sed 's/ .*//'` ?

6. Expliquer en détail la ligne 8.

7. Dans la ligne 9, que se passe-t-il si les guillemets anglais `" "` sont remplacés par des apostrophes `' '` ?

8. Que fait ce script ?

9. Modifier le script de façon à ce qu'à la fin soit produit également

— un affichage de la somme des tailles de *tous les fichiers* (non cachés du répertoire donné en argument) ;

— un affichage de la somme des tailles des fichiers (non cachés du répertoire donné en argument) *dont la taille dépasse la limite.*

Exercice 4 : Copier en toute sécurité — 14 points

1. Créez la commande `copier` qui permet de copier le contenu d'un fichier source dans un fichier destination. La commande reçoit en arguments deux noms de fichiers : la source et la destination.

Le script se termine avec un code de sortie d'erreur, et affiche un message d'erreur si l'une des conditions suivantes est réalisée :

— le nombre d'arguments est incorrect ;

— le fichier source n'existe pas ou n'est pas copiable (pas d'accès en lecture) ;

— le fichier source n'est pas un fichier ordinaire ;

— le fichier destination existe.

2. Modifiez votre script de manière à ce que, si le fichier destination existe, au lieu de se terminer, votre script demande confirmation à l'utilisateur, *à la manière de* l'exemple suivant :

```
$ copier bonjour.c salut.c
Le fichier salut.c existe, l'écraser (o/n) ? o
$ copier liste.c salut.c
Le fichier salut.c existe, l'écraser (o/n) ? n
```

3. Enfin, modifiez votre script de manière à ce que, si l'utilisateur entre une réponse qui n'est pas `o` ou `n`, la question lui soit reposée, jusqu'à ce que sa réponse soit valide.

Exercice 5 : Ranger — 19 points

Le but de cet exercice est de construire pas à pas un script `ranger` qui range les fichiers passés en arguments dans les sous-répertoires **images**, **documents** ou **videos** du répertoire de travail selon le principe suivant :

— les fichiers dont l'extension est `.pdf` sont rangés (déplacés) dans **documents** ;

— les fichiers dont l'extension est `.png` ou `.gif` sont rangés dans **images** ;

— les fichiers dont l'extension est `.avi` ou `.mov` sont rangés dans **videos**.

1. Écrire le script `ranger` dans le cas où il n'a qu'un seul argument.

2. Le modifier pour qu'il accepte un nombre quelconque d'arguments.

3. Vérification des répertoires cible : modifier le script de manière à ce que si le répertoire **images** n'existe pas dans le répertoire personnel, il soit créé. *Le cas des autres répertoires étant similaire, il n'est pas nécessaire de le traiter sur la copie.*

4. Vérification des répertoires cible (bis) : modifier le script de manière à ce que si un fichier **images** existe déjà dans le répertoire personnel mais :

— n'est pas ou répertoire, ou bien

— est un répertoire qui n'a pas les permissions suffisantes pour y déplacer des fichiers (à vous de dire lesquelles)

alors le script affiche un message approprié sur la sortie appropriée, puis sort avec un code d'erreur approprié.

5. Rangement des fichiers `JPEG` : les fichiers `JPEG` sont des images compressées dont l'extension est souvent `.jpg`, `.JPG`, `.jpeg` ou bien `.JPEG`. Quel désordre ! Modifier votre script pour qu'il range les images `JPEG` dans le répertoire **images** *en faisant en sorte que l'extension soit toujours* `.jpeg` (par exemple le fichier `lion.JPG` est renommé `lion.jpeg`).

Vous *pourrez* utiliser la commande `basename` (voir ci-dessous pour une aide).

```
$ basename --help
Utilisation : basename NOM [SUFFIXE]
         ou : basename OPTION... NOM...
Afficher NOM en retirant les parties constituant des répertoires au début du nom.
Si indiqué, enlever aussi le SUFFIXE à la fin.

Les arguments obligatoires pour les options longues le sont aussi pour les
options courtes.
  -a, --multiple       accepter plusieurs arguments et les traiter comme un NOM
  -s, --suffix=SUFFIXE supprimer le SUFFIXE à la fin ; implique -a
  -z, --zero           terminer chaque ligne de sortie par NUL au lieu d'un
                         saut de ligne
      --help     afficher l'aide et quitter
      --version  afficher des informations de version et quitter

Exemples :
  basename /usr/bin/sort                 → « sort »
  basename include/stdio.h .h            → « stdio »
  basename -s .h include/stdio.h         → « stdio »
  basename -a nimp/chaine1 nimp/chaine2  → « chaine1 » suivi par « chaine2 »

Aide en ligne de GNU coreutils : <https://www.gnu.org/software/coreutils/>
```